SC22
Dallas, TX | hpc accelerates.

Drishti
Guiding End-Users in the I/O Optimization Journey

Jean Luca Bez, Hammad Ather, Suren Byna

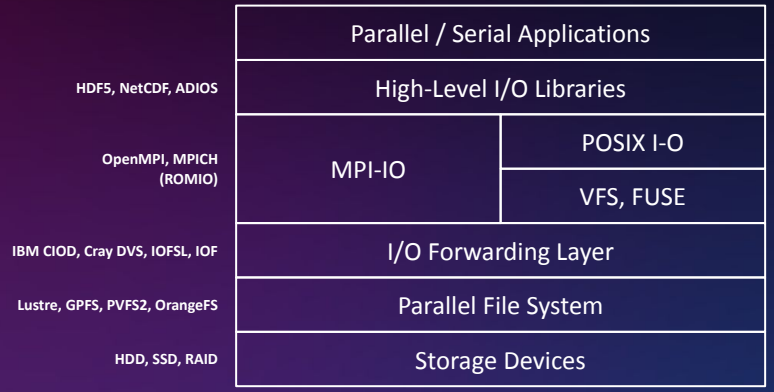
Lawrence Berkeley National Laboratory
jlbez@lbl.gov

ECP EXASCALE COMPUTING PROJECT

BERKELEY LAB
Bringing Science Solutions to the World

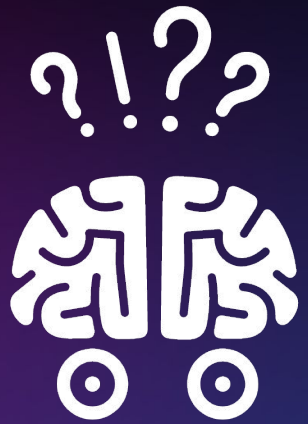
Complex I/O stack!

- Using the HPC I/O stack **efficiently** is a **tricky problem**
- Interplay of factors can affect I/O **performance**
- Various **optimizations techniques** available
- Plethora of **tunable parameters**
  - Each layer brings a new set of parameters



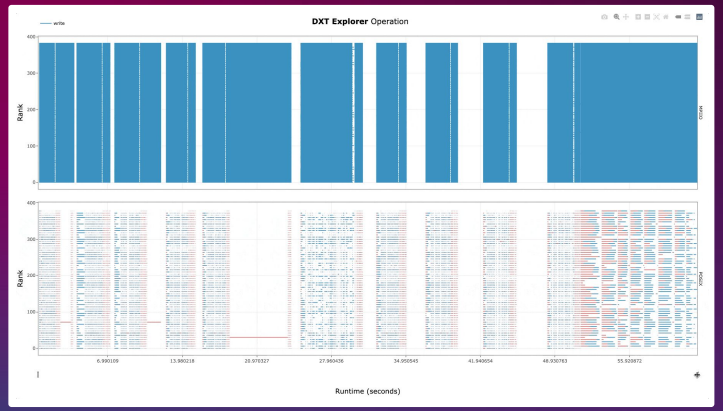
Metrics to the rescue?

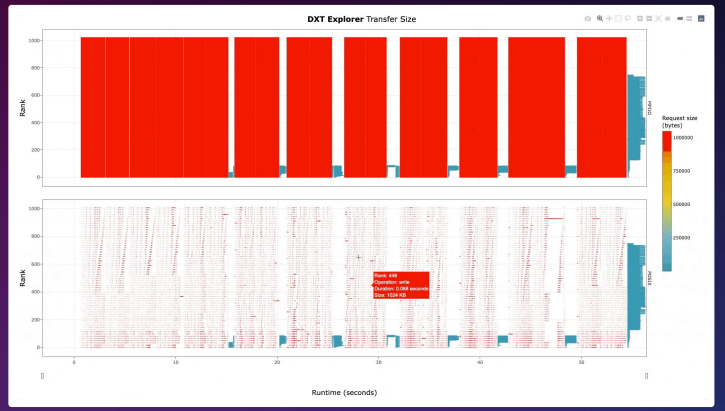
- **Darshan** is a popular tool to collect I/O **profiling**
  - It **aggregates** information to provide insights
  - Extended tracing mode (**DXT**) for a **fine grain** view
- **Recorder** and **TAU** are other I/O profiling tools
- **How to optimize the I/O of my application?**



Visualization to the rescue?

- **DXT Explorer** can aid visualizing I/O behavior
  - It **requires** tracing to be enabled in Darshan





- **How to optimize the I/O of my application?**

What is the problem?

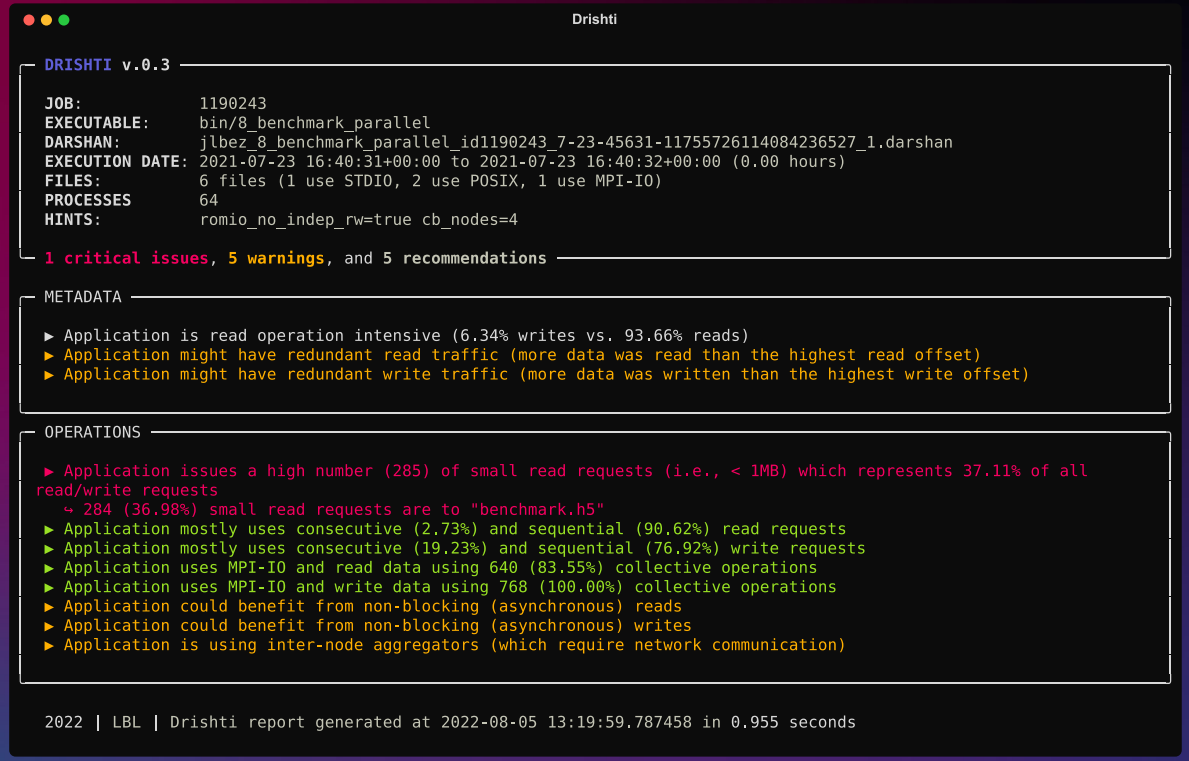
- There is still a **gap** between **profiling** and **tuning**
- **Drishti**: from I/O profiles to meaningful information
  - **Detect** root causes of I/O bottlenecks
  - **Map** I/O bottlenecks into actionable items
  - **Guide** end-user to tune I/O performance
- 4 levels of triggers (**HIGH**, **WARN**, **OK**, **INFO**)
- > 30 triggers are checked for each .darshan log

Overall information about the Darshan log and execution

Number of critical issues, warning, and recommendations

Drishti checks metrics for **over 30 triggers**

Highlight the **file** that triggered the issue

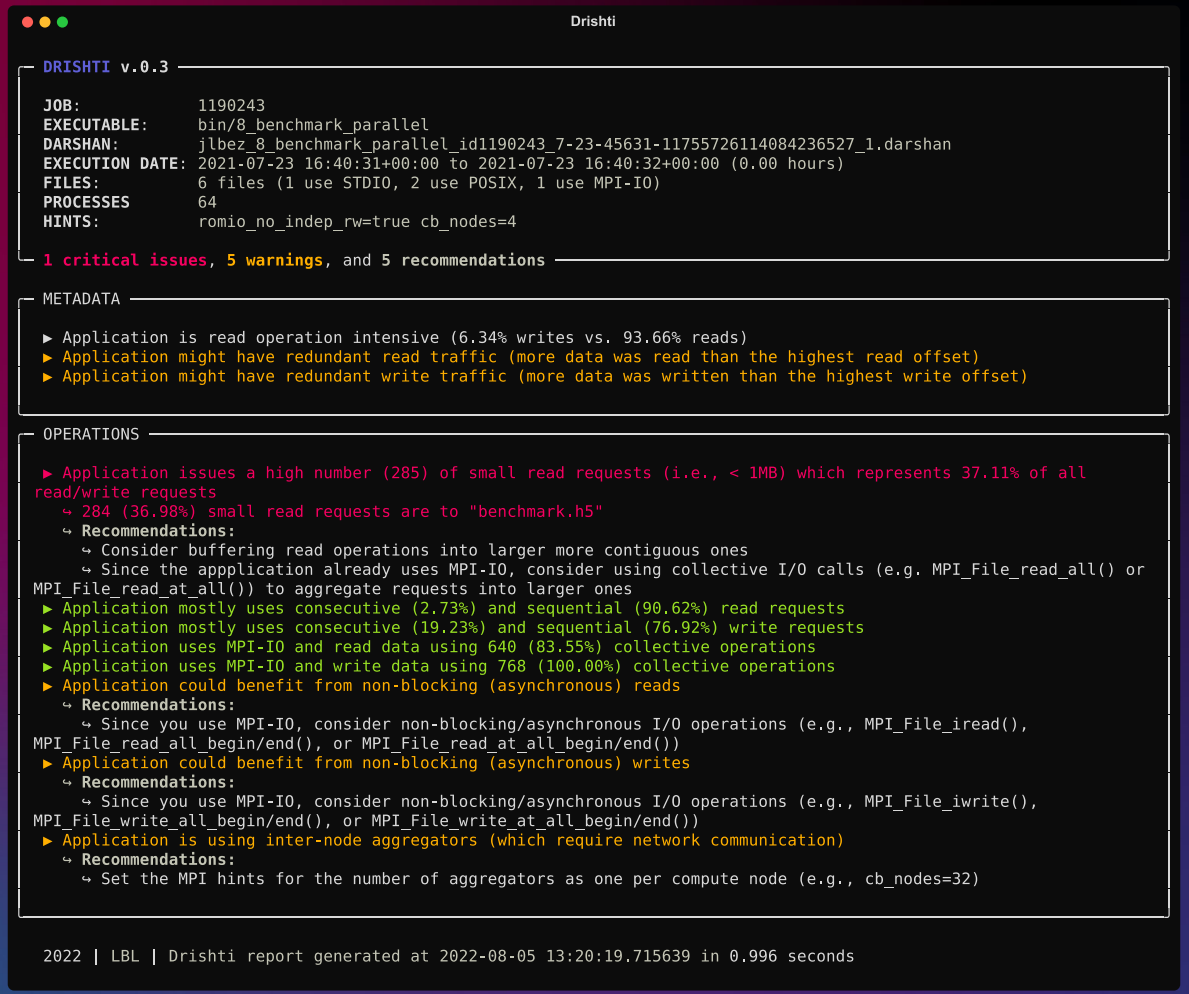


Current version only checks **profiling** metrics

Severity based on certainty and impact: **high**, **medium**, **low**, info

Multiple output formats: textual, SVG, HTML

Provides **actionable** feedback for users



Drishti can check for HDF5 usage to **fine tune** the recommendations

Sample **code solutions** are provided

OPERATIONS

▶ Application issues a high number (285) of small read requests (i.e., < 1MB) which represents 37.11% of all read/write requests
↳ 284 (36.98%) small read requests are to "benchmark.h5"
↳ **Recommendations:**
↳ Consider buffering read operations into larger more contiguous ones
↳ Since the appplication already uses MPI-IO, consider using collective I/O calls (e.g. MPI_File_read_all() or MPI_File_read_at_all()) to aggregate requests into larger ones

Solution Example Snippet

```
1 MPI_File_open(MPI_COMM_WORLD, "output-example.txt", MPI_MODE_CREATE|MPI_MODE_RDONLY, MPI_INFO_NULL,
2 ...
3 MPI_File_read_all(fh, &buffer, size, MPI_INT, &s);
```

▶ Application mostly uses consecutive (2.73%) and sequential (90.62%) read requests
▶ Application mostly uses consecutive (19.23%) and sequential (76.92%) write requests
▶ Application uses MPI-IO and read data using 640 (83.55%) collective operations
▶ Application uses MPI-IO and write data using 768 (100.00%) collective operations
▶ Application could benefit from non-blocking (asynchronous) reads
↳ **Recommendations:**
↳ Since you use MPI-IO, consider non-blocking/asynchronous I/O operations (e.g., MPI_File_iread(), MPI_File_read_all_begin/end(), or MPI_File_read_at_all_begin/end())

Solution Example Snippet

```
1  MPI_File fh;
2  MPI_Status s;
3  MPI_Request r;
4  ...
5  MPI_File_open(MPI_COMM_WORLD, "output-example.txt", MPI_MODE_CREATE|MPI_MODE_RDONLY, MPI_INFO_NULL
6  ...
7  MPI_File_iread(fh, &buffer, BUFFER_SIZE, n, MPI_CHAR, &r);
8  ...
9  // compute something
10 ...
11 MPI_Test(&r, &completed, &s);
12 ...
13 if (!completed) {
14     // compute something
15
16     MPI_Wait(&r, &s);
17 }
```

▶ Application could benefit from non-blocking (asynchronous) writes
↳ **Recommendations:**
↳ Since you use MPI-IO, consider non-blocking/asynchronous I/O operations (e.g., MPI_File_iwrite(), MPI_File_write_all_begin/end(), or MPI_File_write_at_all_begin/end())

Solution Example Snippet

```
...
MPI_Test(&r, &completed, &s);
...
if (!completed) {
    // compute something

    MPI_Wait(&r, &s);
}
```

▶ Application is using inter-node aggregators (which require network communication)
↳ **Recommendations:**
↳ Set the MPI hints for the number of aggregators as one per compute node (e.g., cb_nodes=32)

Solution Example Snippet

```
# ------------------------------ #
# MPICH                          #
# ------------------------------ #
export MPICH_MPIIO_HINTS="*:cb_nodes=16:cb_buffer_size=16777216:romio_cb_write=enable:romio_ds_wri

# * means it will apply the hints to any file opened with MPI-IO
# cb_nodes        ---> number of aggregator nodes, defaults to stripe count
# cb_buffer_size  ---> controls the buffer size used for collective buffering
# romio_cb_write  ---> controls collective buffering for writes
# romio_cb_read   ---> controls collective buffering for reads
# romio_ds_write  ---> controls data sieving for writes
# romio_ds_read   ---> controls data sieving for reads

# to visualize the used hints for a given job
export MPICH_MPIIO_HINTS_DISPLAY=1

# ------------------------------ #
# OpenMPI / SpectrumMPI (Summit)  #
# ------------------------------ #
export OMPI_MCA_io=romio321
export ROMIO_HINTS=./my-romio-hints

# the my-romio-hints file content is as follows:
cat $ROMIO_HINTS

romio_cb_write enable
romio_cb_read  enable
romio_ds_write disable
romio_ds_read  disable
cb_buffer_size 16777216
cb_nodes 8
```

Sample **configurations** are provided

How to get Drishti?

```
# Install Drishti on your local machine
$ pip install drishti
# Run Drishti with the provided .darshan DXT traces
$ drishti --verbose samples/REPLACE_WITH_FILE_NAME.darshan


# On NERSC systems you can also use the container version with Shifter
$ shifter --image=docker:hpcio/drishti -- drishti samples/REPLACE_WITH_FILE_NAME.darshan
```

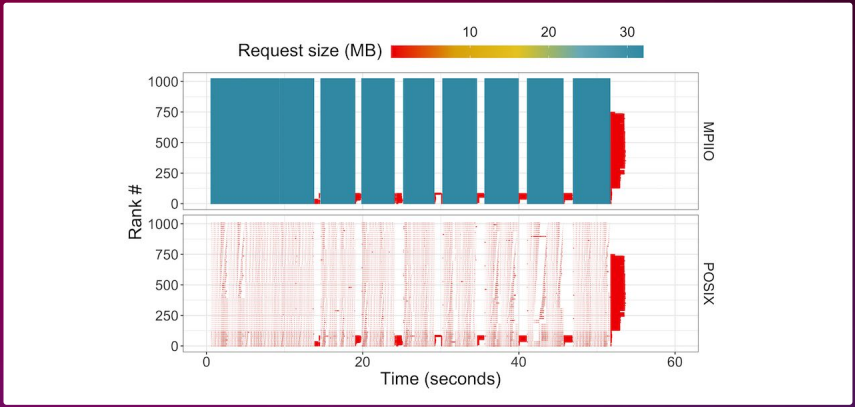
How to run Drishti?

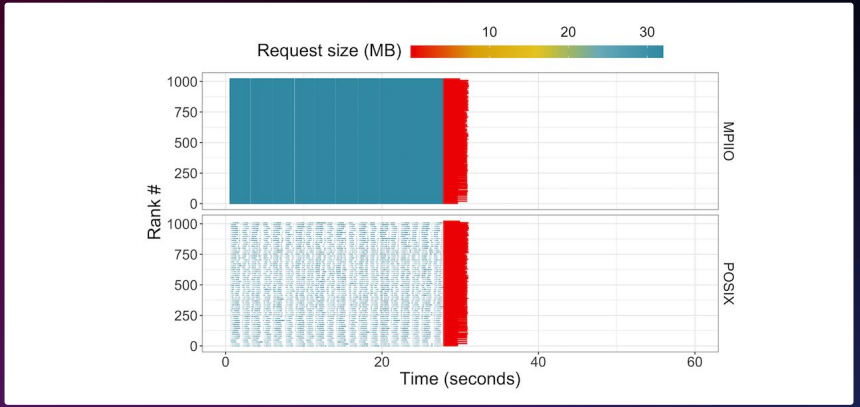
```
usage: drishti [-h] [--issues] [--html] [--svg] [--verbose] [--code] darshan

Drishti:
positional arguments:
  darshan     Input .darshan file

optional arguments:
  -h, --help  show this help message and exit
  --issues    Only displays the detected issues and hides the recommendations
  --html      Export the report as an HTML page
  --svg       Export the report as an SVG image
  --verbose   Display extended details for the recommendations
  --code      Display insights identification code
```

OpenPMD Use Case





METADATA

- ▶ Application is write operation intensive (60.83% writes vs. 39.17% reads)
- ▶ Application is write size intensive (64.15% write vs. 35.85% read)
- ▶ Application issues a high number (100.00%) of misaligned file requests

OPERATIONS

- ▶ Application issues a high number (275840) of small read requests (i.e., < 1MB) which represents 100.00% of all read/write requests
  - ↪ 275840 (100.00%) small read requests are to "8a_parallel_3Db_0000001.h5"
- ▶ Application issues a high number (427386) of small write requests (i.e., < 1MB) which represents 99.75% of all read/write requests
  - ↪ 275840 (64.38%) small write requests are to "8a_parallel_3Db_0000001.h5"
- ▶ Application mostly uses consecutive (97.67%) and sequential (2.16%) read requests
- ▶ Application mostly uses consecutive (97.85%) and sequential (1.17%) write requests
- ▶ Application uses MPI-IO and write data using 7680 (92.50%) collective operations
- ▶ Application could benefit from non-blocking (asynchronous) reads
- ▶ Application could benefit from non-blocking (asynchronous) writes

METADATA

- ▶ Application is write operation intensive (90.85% writes vs. 9.15% reads)
- ▶ Application is write size intensive (91.14% write vs. 8.86% read)
- ▶ Application might have redundant read traffic (more data read than the highest offset)

OPERATIONS

- ▶ Application is issuing a high number (565) of random read operations (35.25%)
- ▶ Application mostly uses consecutive (88.56%) and sequential (7.02%) write requests
- ▶ Application uses MPI-IO and write data using 8448 (100.00%) collective operations
- ▶ Application could benefit from non-blocking (asynchronous) reads
- ▶ Application could benefit from non-blocking (asynchronous) writes

Drishti's Overview of Cori's Darshan Logs @ NERSC

- **112,612** Darshan logs collected in **Cori**
- March 1st to March 5th, 2022
- **Runtime** depends on the size of .darshan file
  - min. of **0.02** seconds, a mean of **10.49** seconds
  - max. of **134.98** seconds to generate a report

Level	Interface	Detected Behavior	Jobs	Total (%)	Relative* (%)
HIGH	STDIO	**High STDIO usage (>10% of total transfer size uses STDIO)**	**43,120**	**38.29**	**52.1**
OK	POSIX	High number of sequential read operations (≥ 80%)	38,104	33.84	58.14
OK	POSIX	**High number of sequential write operations (≥ 80%)**	**64,486**	**57.26**	**98.39**
INFO	POSIX	Write operation count intensive (>10% more writes than reads)	26,114	23.19	39.84
INFO	POSIX	Read operation count intensive (>10% more reads than writes)	23,168	20.57	35.35
INFO	POSIX	Write size intensive (>10% more bytes written then read)	23,568	20.93	35.96
INFO	POSIX	Read size intensive (>10% more bytes read then written)	40,950	36.36	62.48
WARN	POSIX	Redundant reads	14,518	12.89	22.15
WARN	POSIX	Redundant writes	59	0.05	0.09
HIGH	POSIX	**High number of small (<1MB) read requests (>10% of total read requests)**	**64,858**	**57.59**	**98.96**
HIGH	POSIX	**High number of small (<1MB) write requests (>10% of total write requests)**	**64,552**	**57.32**	**98.49**
HIGH	POSIX	High number of misaligned memory requests (>10%)	36,337	32.27	55.44
HIGH	POSIX	**High number of misaligned file requests (>10%)**	**65,075**	**57.79**	**99.29**
HIGH	POSIX	**High number of random read requests (>20%)**	**26,574**	**23.6**	**40.54**
HIGH	POSIX	High number of random write requests (>20%)	559	0.5	0.85
HIGH	POSIX	**High number of small (<1MB) reads to shared-files (>10% of total reads)**	**60,121**	**53.39**	**91.73**
HIGH	POSIX	**High number of small (<1MB) writes to shared-files (>10% of total writes)**	**55,414**	**49.21**	**84.55**
HIGH	POSIX	High metadata time (at least one rank spends >30 seconds)	9,410	8.36	14.35
HIGH	POSIX	Data transfer imbalance between ranks causing stragglers (>15% difference)	40,601	36.05	61.95
HIGH	POSIX	Time imbalance between ranks causing stragglers (>15% difference)	40,533	35.99	61.84

Level	Interface	Detected Behavior	Jobs	Total (%)	Relative* (%)
WARN	MPI-IO	**No MPI-IO calls detected from Darshan logs**	**109,569**	**97.3**	**-**
HIGH	MPI-IO	Detected MPI-IO but no collective read operation	169	0.15	5.55
HIGH	MPI-IO	Detected MPI-IO but no collective write operation	428	0.38	14.06
WARN	MPI-IO	Detected MPI-IO but no non-blocking read operations	3,043	2.7	100.00
WARN	MPI-IO	Detected MPI-IO but no non-blocking write operations	3,043	2.7	100.00
OK	MPI-IO	Detected MPI-IO and collective read operations	402	0.36	13.21
OK	MPI-IO	Detected MPI-IO and collective write operations	2,592	2.3	85.17
HIGH	MPI-IO	**Detected MPI-IO and inter-node aggregators**	**2,496**	**2.22**	**82.02**
WARN	MPI-IO	Detected MPI-IO and intra-node aggregators	304	0.27	9.99
OK	MPI-IO	Detected MPI-IO and one aggregator per node	29	0.03	0.95

Conclusion

- **Drishti**: a solution to **guide end-users in optimizing** their applications
  - Towards **closing the gap** between I/O metrics and tuning solutions
  - **Detect** typical performance I/O pitfalls
  - **Provide** a set of recommendations
- Evaluated Drishti with user-cases and large volume of Darshan logs
- **Future work:**
  - Include additional sources of logs (e.g., DXT, Recorder)
  - Integrate with DXT-Explorer

Drishti

Guiding End-Users in the I/O Optimization Journey

Jean Luca Bez, Hammad Ather, Suren Byna
jlbez@lbl.gov

docker pull hpcio/drishti

github.com/hpc-io/drishti

BERKELEY LAB
Bringing Science Solutions to the World