Argonne NATIONAL LABORATORY

ILLINOIS

UNIVERISTY OF ILLINOIS AT URBANA-CHAMPAIGN

Pilgrim

Near-Lossless MPI Tracing and Proxy Application Autogeneration

Chen Wang [1], Yanfei Guo [2], Pavan Balaji [3], Marc Snir [1]
[1] University of Illinois Urbana-Champaign
[2] Argonne National Laboratory
[3] Meta, Inc

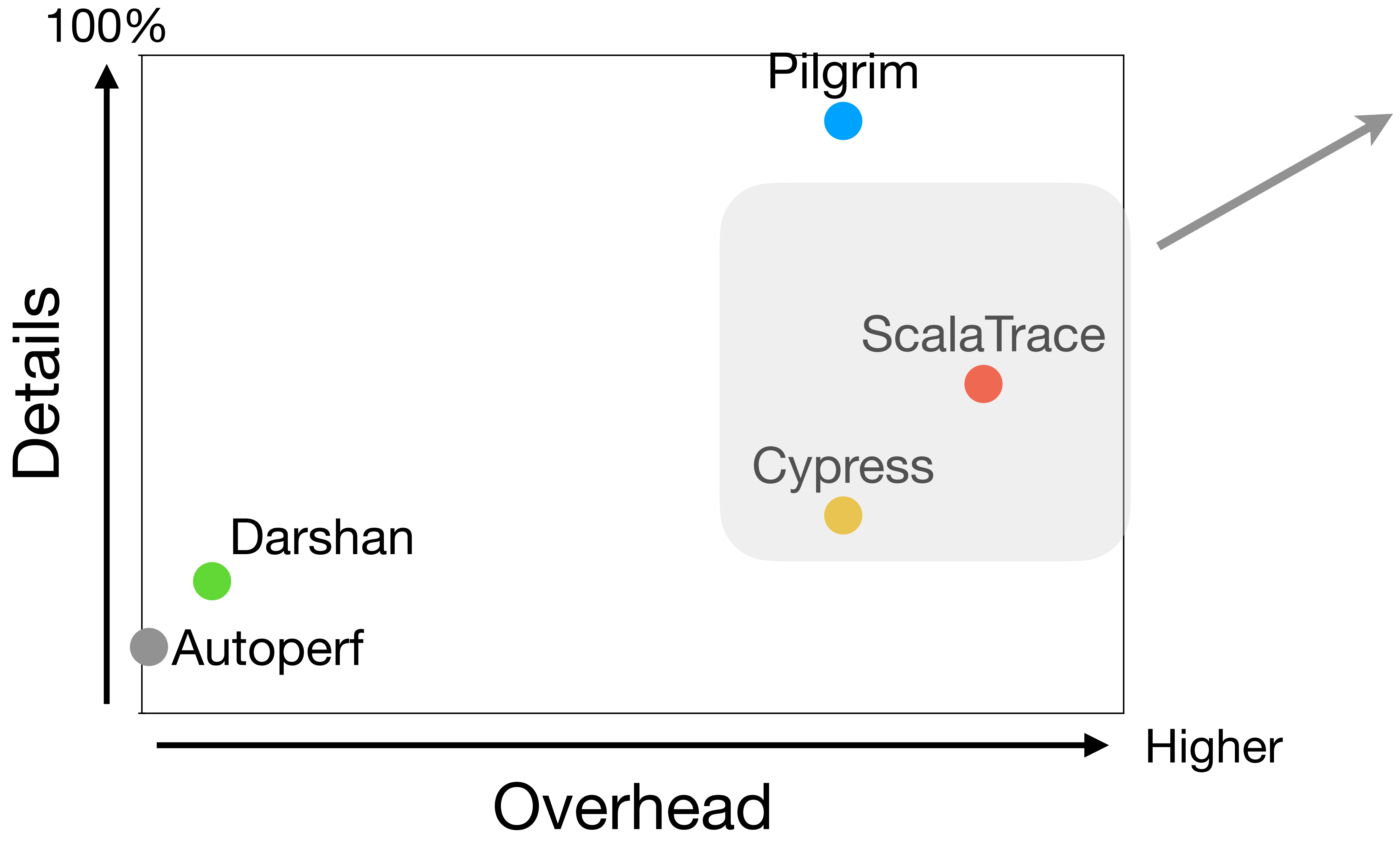
2022/05/11

Why do we collect MPI information

- For MPI and application users:
  - How frequent calls are?
  - Am I providing the right hints to MPI for my usage?
  - Am I using MPI correctly?
- For MPI developers:
  - What features are used and in what way?
  - Message sizes, communicator sizes, buffer reuse?
  - Are send/recv sizes the same or different?
  - Are collective operation datatype on all processes the same or different?

Pilgrim vs. Existing tools

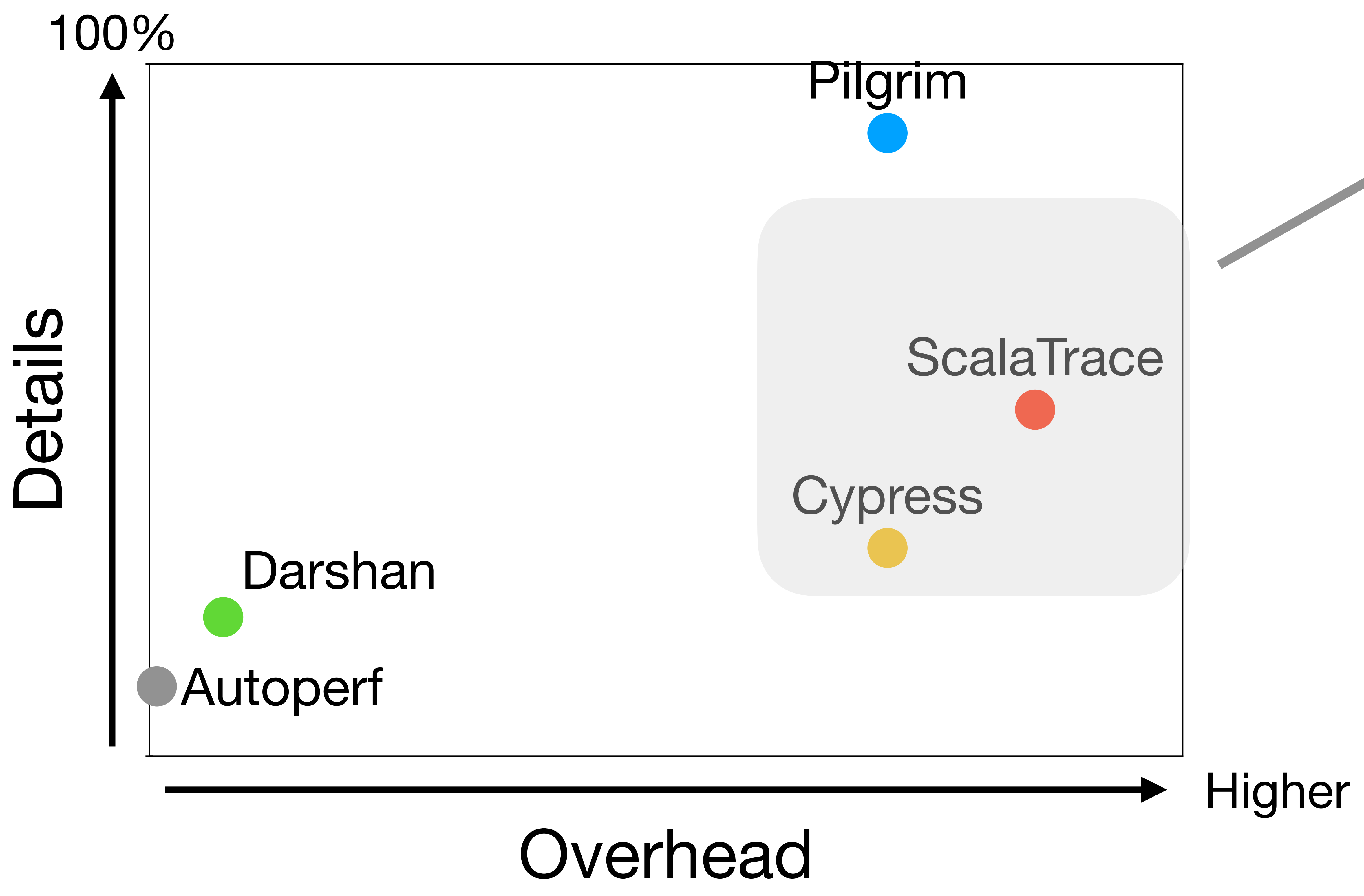
Pilgrim stores every parameter of every MPI call



Existing tools are either incomplete or have unacceptable overhead (time or space)

Pilgrim vs. Existing tools

Pilgrim stores every parameter of every MPI call



Existing tools are either incomplete or have unacceptable overhead (time or space)

Functions Supported	Cypress	ScalaTrace	Pilgrim
Total: 446	56	125	446

Popular Parameters	Cypress	ScalaTrace	Pilgrim
MPI_Status	✓	✓	✓
MPI_Request	×	✓	✓
MPI_Comm	intra	intra and inter	intra and inter
MPI_Datatype	only the size	✓	✓
src/dst/tag	✓	✓	✓
memory pointer	×	×	✓

Challenges

1. **Scalability**:
    - the longer an application runs or the more nodes it runs on, the more function calls it will make
2. **Usefulness**: meaningful for post-processing
    - `MPI_Comm, MPI_Request,` Memory pointers?
3. **Correctness and completeness**:
    - Over 400 MPI functions; Many corner cases, e.g., non-blocking communicator creation.

How?

1. **Scalability**:
   - the longer an application runs or the more nodes it runs on, the more function calls it will make

Recurring Pattern Recognization

2. **Usefulness**: meaningful for post-processing
   - `MPI_Comm`, `MPI_Request`, Memory pointers?

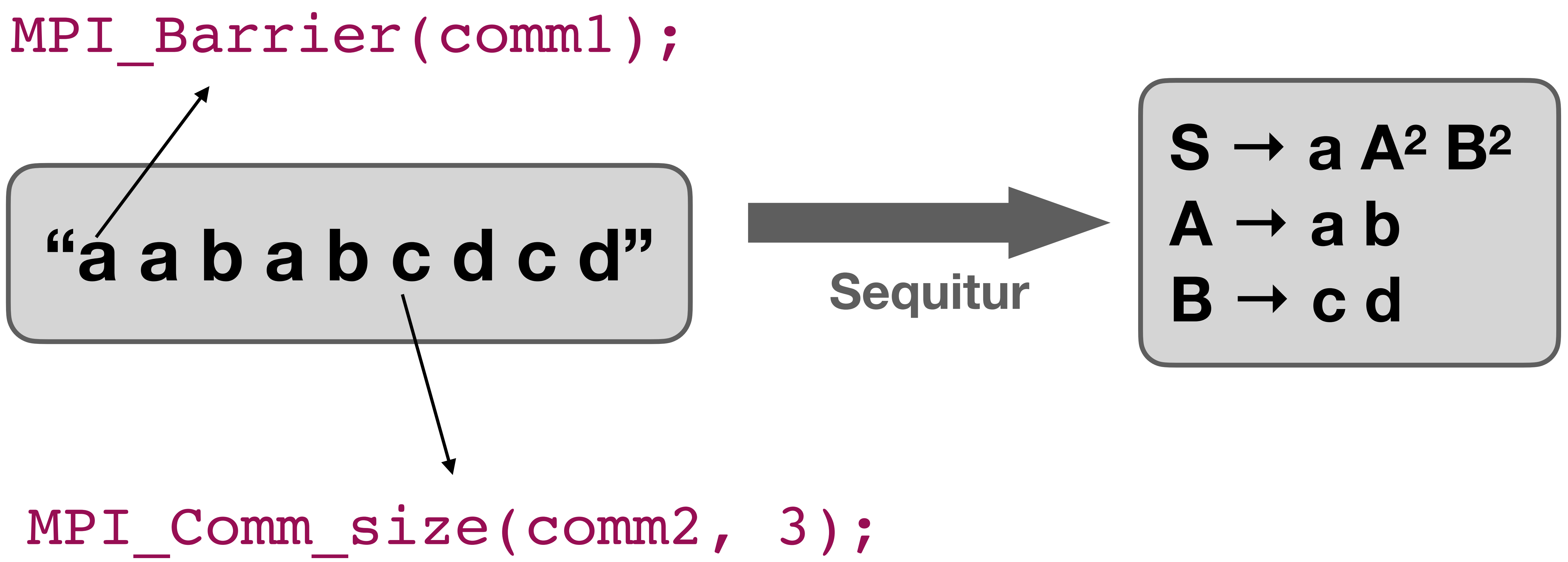
Memory Operation Interception
Symbolic Representation

3. **Correctness and completeness**:
   - Over 400 MPI functions; Many corner cases, e.g., non-blocking communication creation.

Wrappers are automatically
generated from MPI Standard

How?

- Primarily relies on “**recurring pattern recognition**”
  - The key is to detect as many patterns as possible.
  - Store the patterns in a context-free-grammar (CFG)



```
MPI_Barrier(comm1);
```

“a a b a b c d c d”

Sequitur

$S \rightarrow a\ A^2\ B^2$

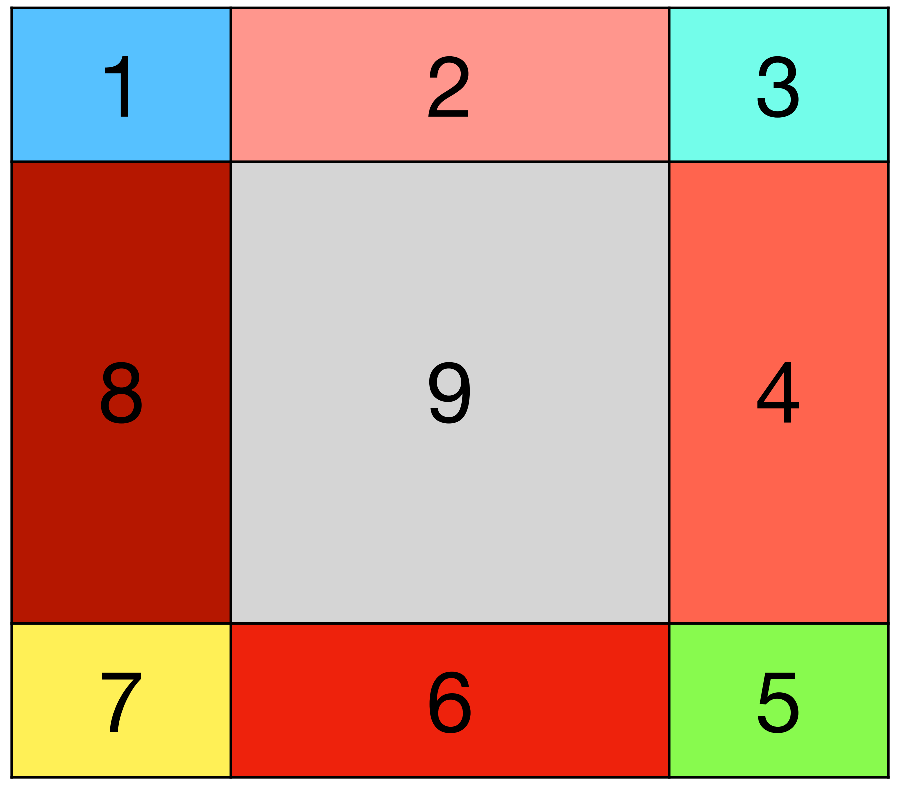
$A \rightarrow a\ b$

$B \rightarrow c\ d$

```
MPI_Comm_size(comm2, 3);
```

How?

- Primarily relies on “**recurring pattern recognition**”
  - The key is to detect as many patterns as possible.
  - Store the patterns in a context-free-grammar (CFG)

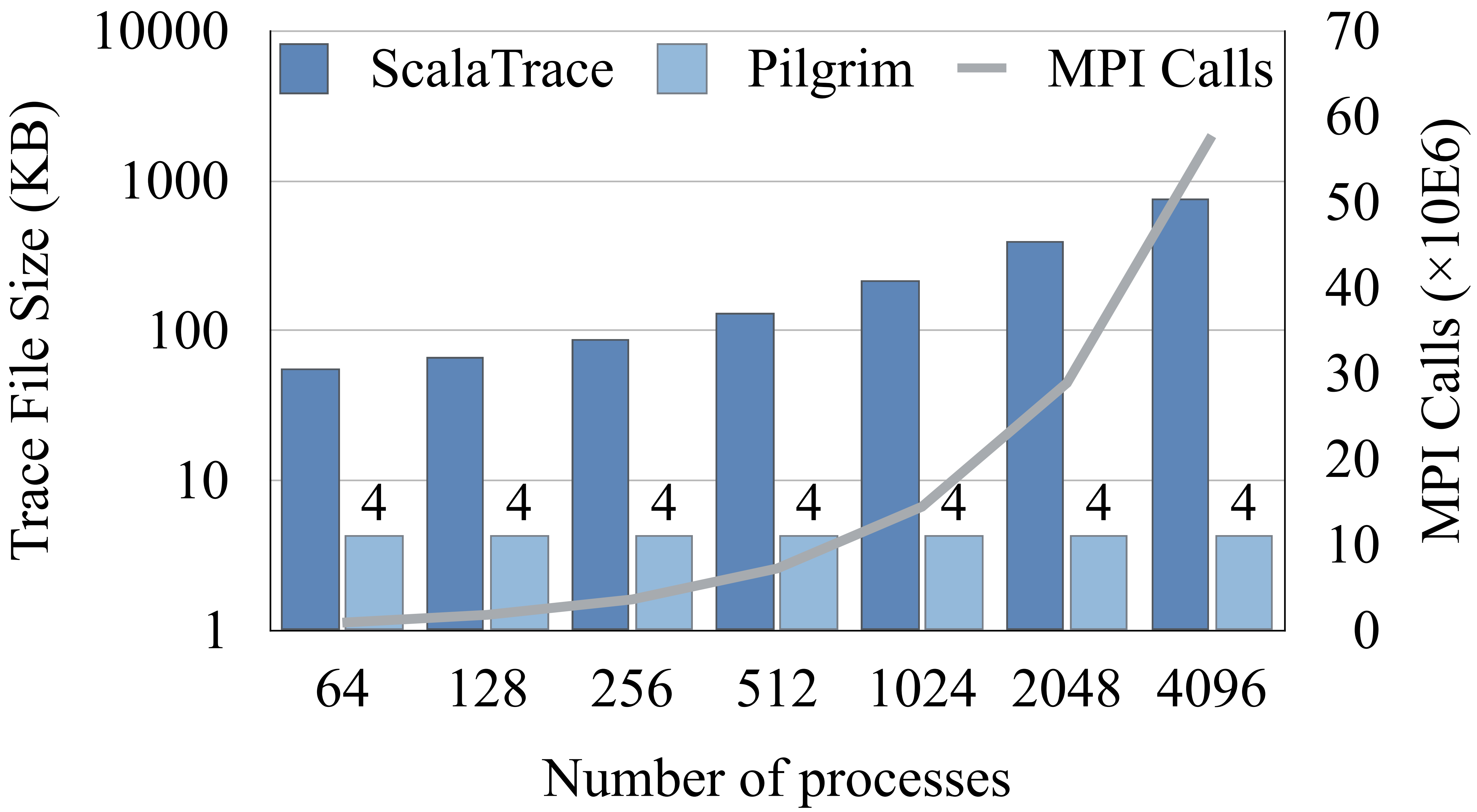


e.g., 2D 5-points periodical stencil will exhibit up to 9 unique communication patterns

Result

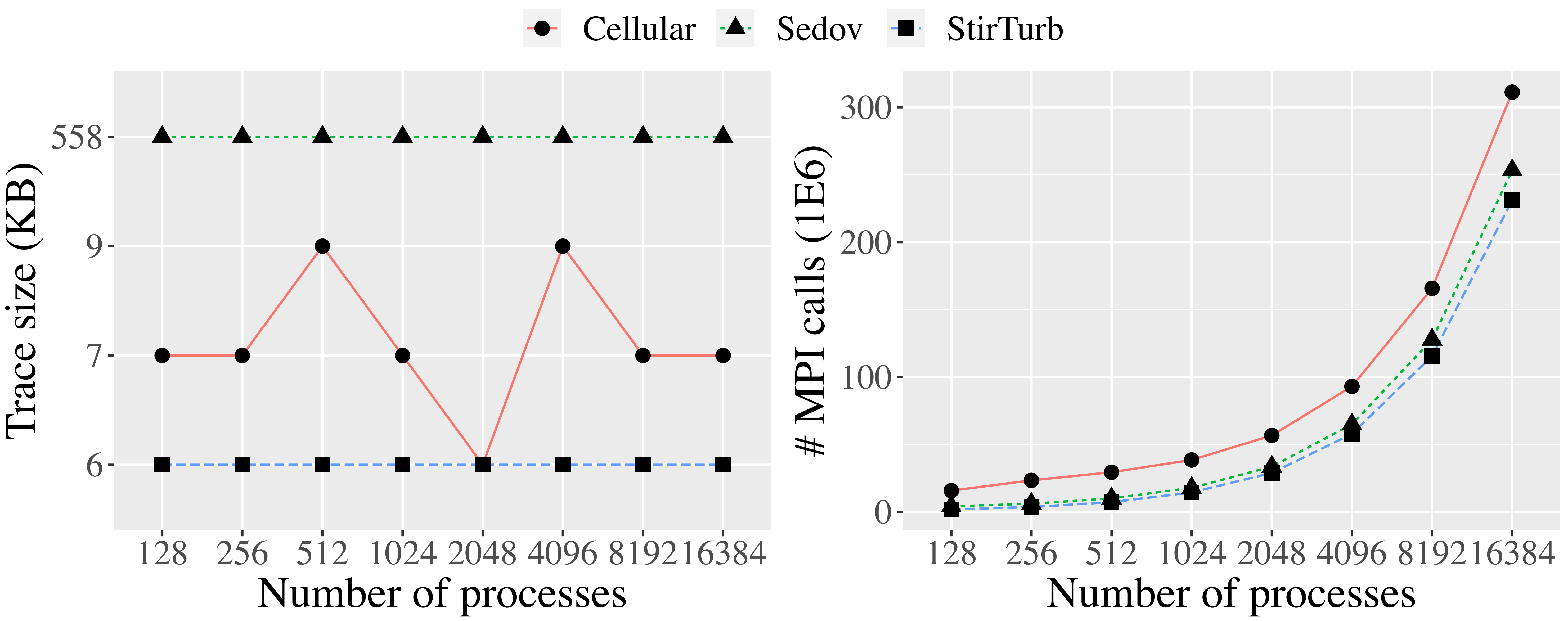
- **Pilgrim stores more information with less space and time overhead.**
- Only unique communication patterns matter
  - Trace size will stay constant if all patterns are recognized
- Overhead depends on communication-to-computation ratio.

FLASH StirTurb 3D Simulation



Result

- Pilgrim stores more information with less space and time overhead.
- Only unique communication patterns matter
    - **Trace size will stay constant if all patterns are recognized**
- Overhead depends on communication-to-computation ratio.



Proxy app generation

Issues of the current method

Designing a proxy app manually is a nontrivial task and exposes three major issues:

1. It requires the involvement of the experts of the original application when it tries to best mimic the original behavior. It takes significant efforts since the original application code is most likely huge.

2. It requires access to the source code of the original application, which may be infeasible for classified applications.

3. Removing part of the intrinsic logic (e.g., computation) from the original application makes debugging and correctness checking difficult.

Proxy app generation

Autogeneration algorithm

With the detailed information preserved by Pilgrim, however, we can design a proxy app generator that addresses these issues.

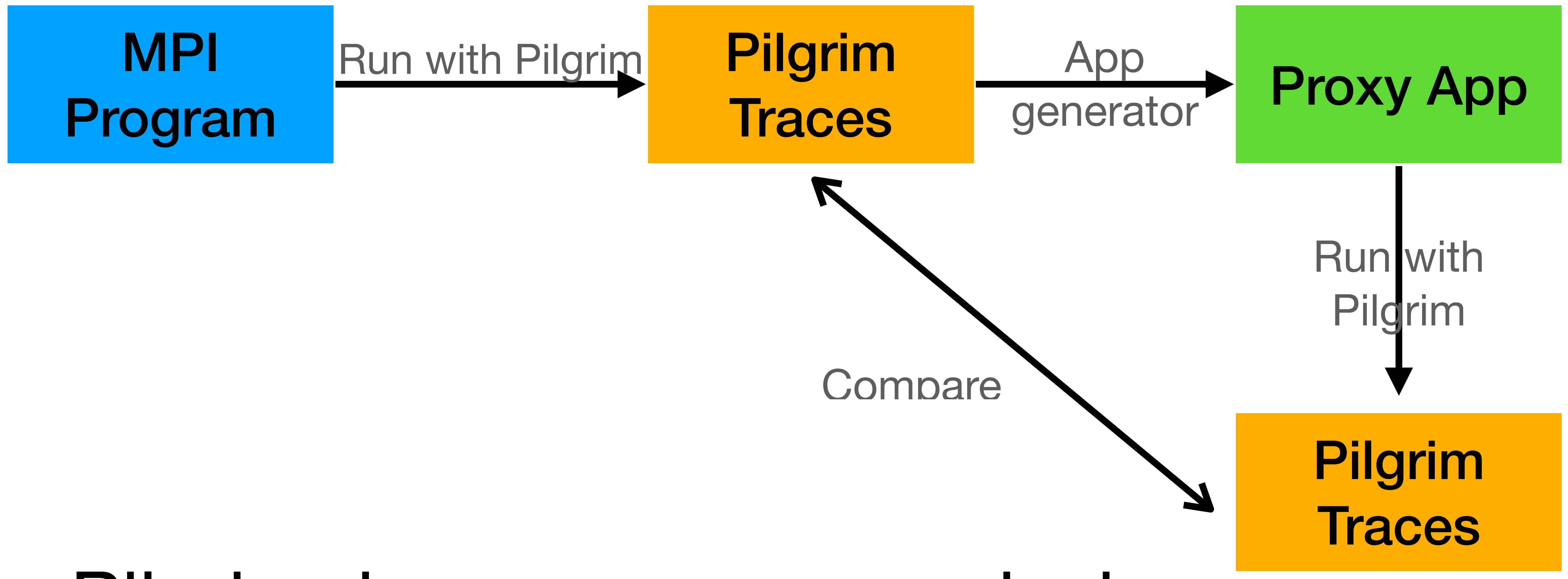
1. The proxy app is generated automatically from Pilgrim traces with little or no human intervention.

2. It relies only on the traces of the targeted application, not the source code; and the tracing process has already stripped away the computation information.

3. Correctness checking is easy because we can run the generated proxy app with Pilgrim again and compare its traces with the original application's traces.

Proxy app generation

Autogeneration algorithm



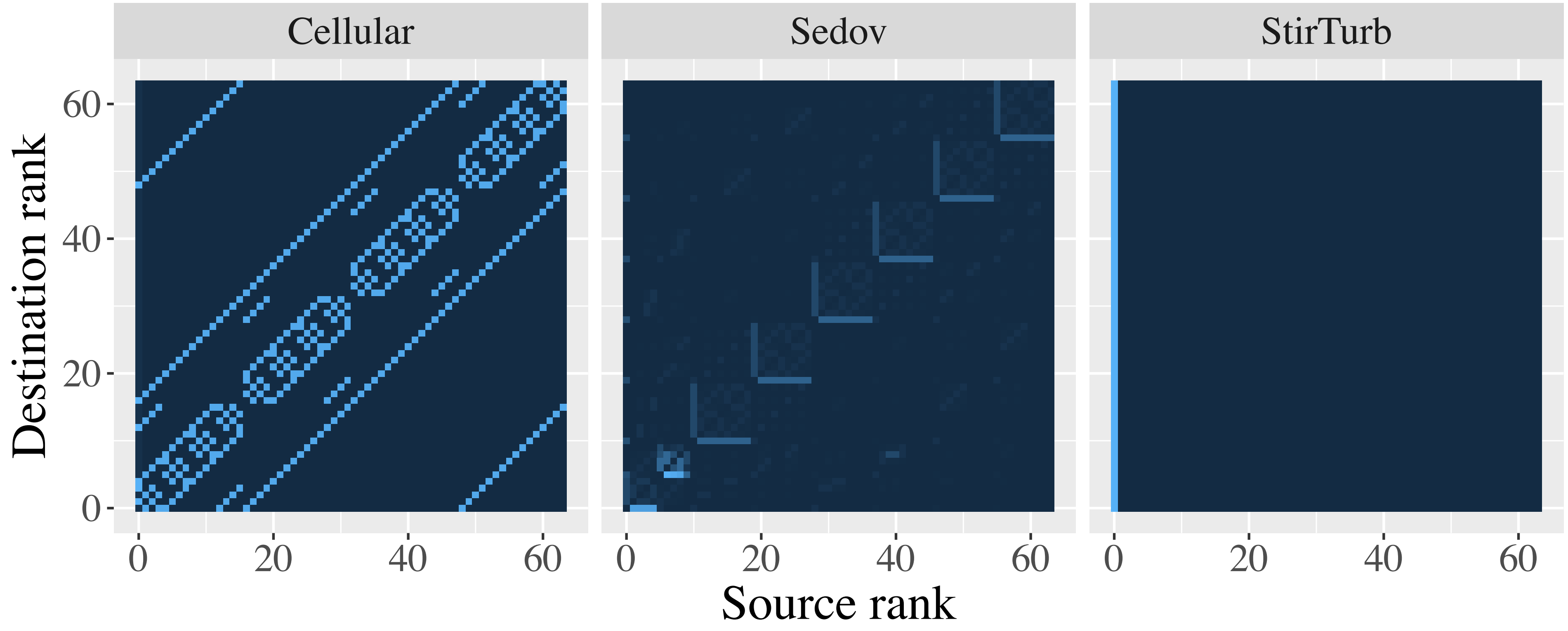
With the detailed information preserved by Pilgrim, however, we can design a proxy app generator that addresses these issues.

1. The proxy app is generated automatically from Pilgrim traces with little or no human intervention.

2. It relies only on the traces of the targeted application, not the source code; and the tracing process has already stripped away the computation information.

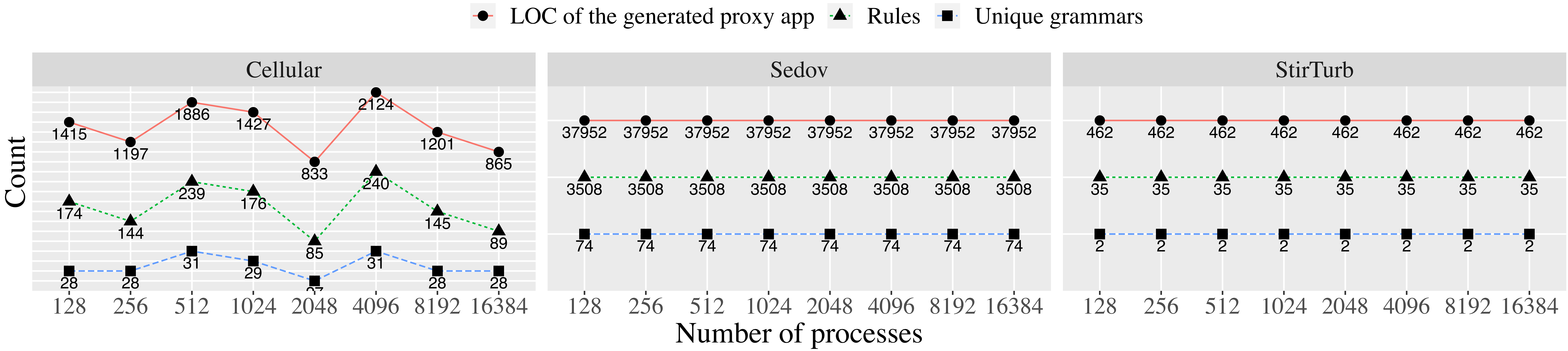
3. Correctness checking is easy because we can run the generated proxy app with Pilgrim again and compare its traces with the original application's traces.

Proxy app generation



- Restore the captured recurring patterns, which represent the original code structure.

- The proxy app will be small and clean if Pilgrim compresses well
  - **The generated code size is proportional to the size of the compressed trace**



Other features and limitations

- Pilgrim tracks all memory management operations (CUDA and CPU).
    - It also knows which GPU device was the memory allocated
- Multi-threading support is almost done, e.g., MPI_THREAD_MULTIPLE
- For the proxy-app generator, user-defined MPI functions (e.g., MPI_Copy_function) are not supported
    - Pilgrim traces function calls only at runtime and does not perform compile-time analysis

Resources

- Pilgrim is publicly available at: https://github.com/pmodels/pilgrim
- Papers:
    - Chen Wang, Pavan Balaji, and Marc Snir. “Pilgrim: Scalable and (near) Lossless MPI Tracing”, SC, 2021.
    - Chen Wang, Yanfei Guo, Pavan Balaji, and Marc Snir. “Near-Lossless MPI Tracing and Proxy Application Autogeneration”, under review, TPDS.
- Contact:
    - Chen Wang (chenw5@illinois.edu)
    - Yanfei Guo (yguo@anl.gov)