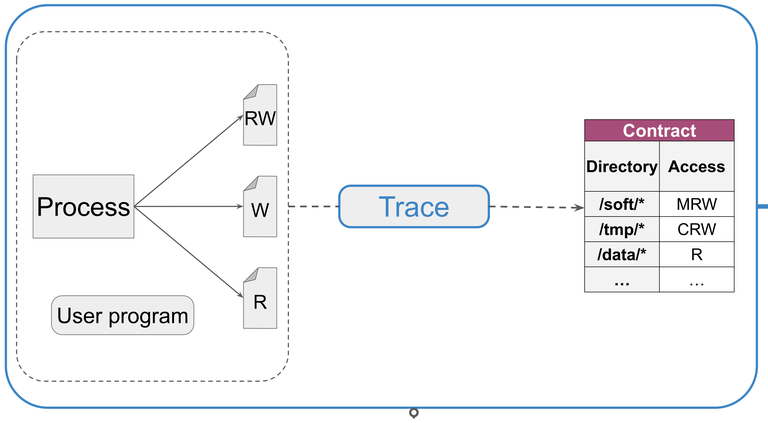
PLEDGE:
Accelerating Data Intensive Scientific Applications with Consistency Contracts

Andres Iglesias and Ben Tovar {aiglesi3,btovar}@nd.edu
Acknowledgements: Prof. Scott Hampton, Prof. Paul Brenner

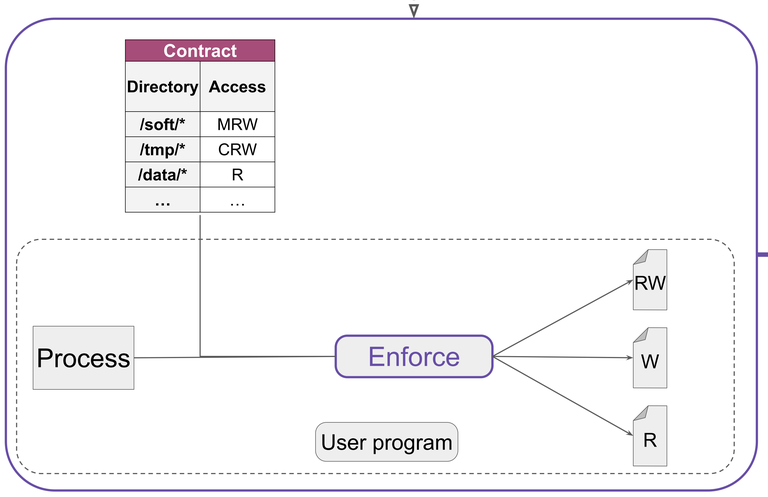
NSF Grant
CSR-2317556

Access types	
Letter	**Access**
M	Metadata
C	Create
D	Delete
R	Read
W	Write
L	List

Advanced scientific computing depends on **applications** that run on large **high performance clusters**. The **filesystem** fulfills the **I/O** needs of these applications, such as, moving files, synchronization between tasks, delivering complex software trees and providing buffers between tasks. End users often don't know what complex **applications** are going to require from the **filesystem** until runtime. **PLEDGE** addresses this problem by **tracing**, **summarizing** and **enforcing** the applications **I/O** behavior.



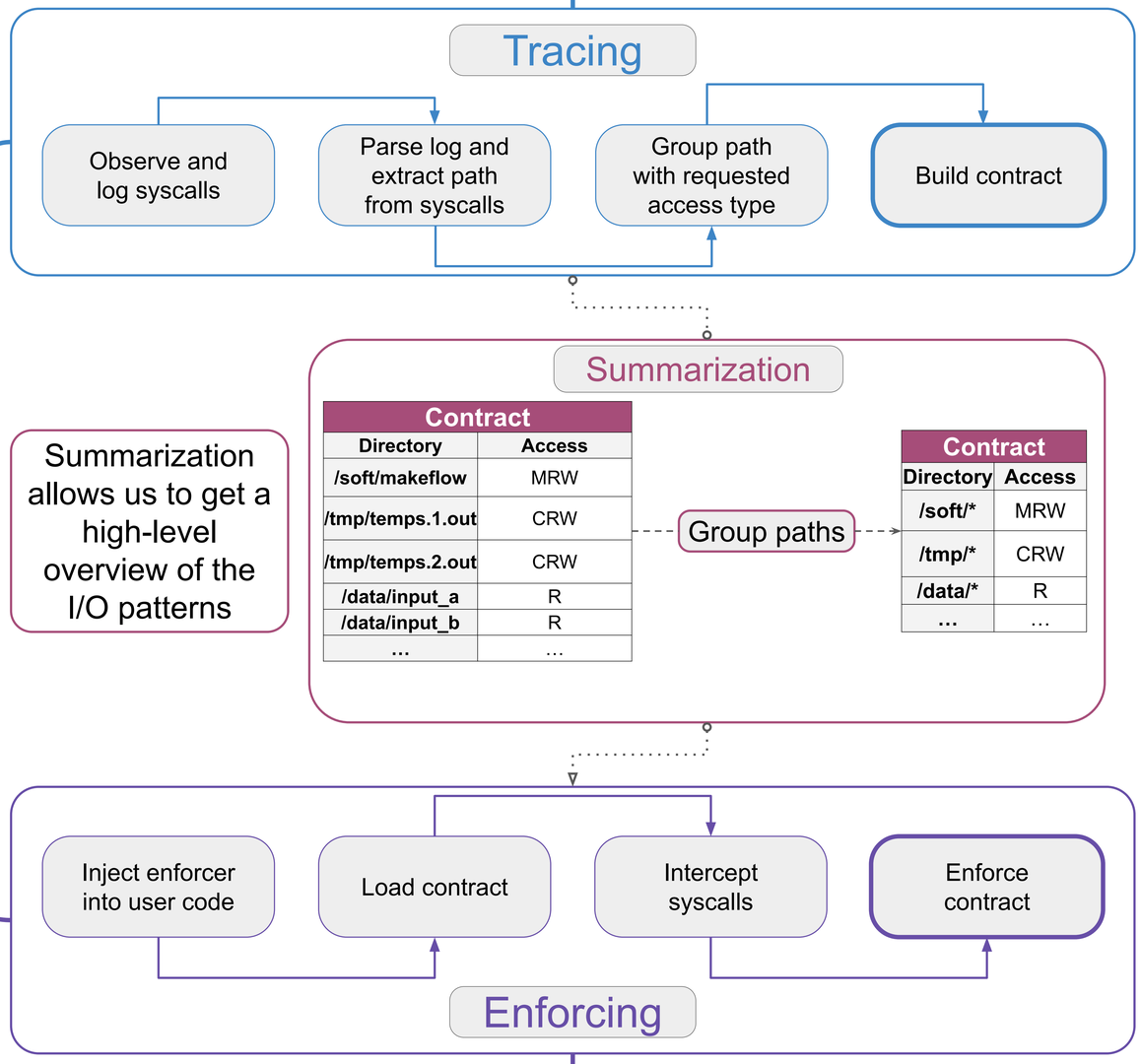
If the **applications** intentions are declared upfront, we can take full advantage of the internal storage and **I/O** capacity of the cluster. These intentions are expressed with "**consistency contracts**", for which we provide a "**tracer**" tool, to generate **contracts** for end users.



Contracts allow us to exploit the **I/O** patterns of scientific **applications** by simplifying the process of caching, buffering and distribution.
An "**enforcer**" wrapper is provided to ensure the application respects the declared **consistency contract**, and it informs users when a violation occurs.

A relatively simple **application** still depends on more **files**, **operations** and **syscalls** than initially claimed. Furthermore, complex **applications**, tend to do unexpected things, like loading configuration files for shared libraries at runtime **before** main.

Tracing a user process is done by observing and logging **syscalls**. The paths used in said **syscalls** allow us to start building a description of the **I/O patterns** of a program. We group each path with all the requested **access types** for it and generate the **contract**. **Contracts** can also be provided by the user.



The **enforcer** is injected at runtime, it loads the **contract** and wraps key **syscalls** to **intercept** and **determine** if the path that was attempted to be **accessed** is in our **contract**, and if the **access type** matches the request. If the **validation** fails, the call is blocked. The contract **is** the description of an applications **I/O patterns**, so the behaviour must match.