

Bridging the Gap between MPC and Information Systems

Kert Tali

Jun 10, 2025

Taxonomy of MPC software

Application

[The MPC component in your information system]

Platform

*e.g.
Sharemind MPC application server,
Carbyne Stack, Roseman Labs*

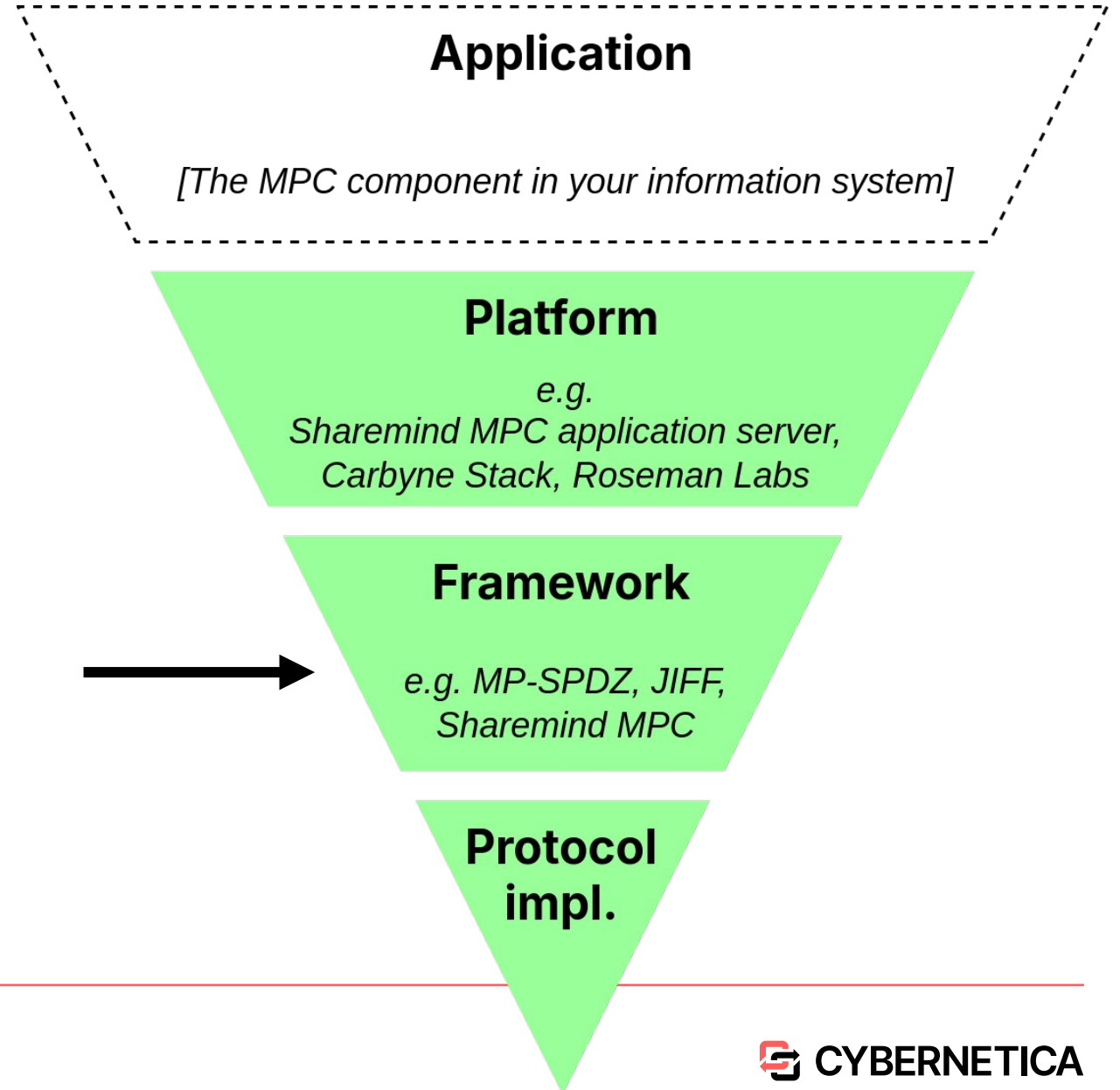
Framework

*e.g. MP-SPDZ, JIFF,
Sharemind MPC*

Protocol impl.

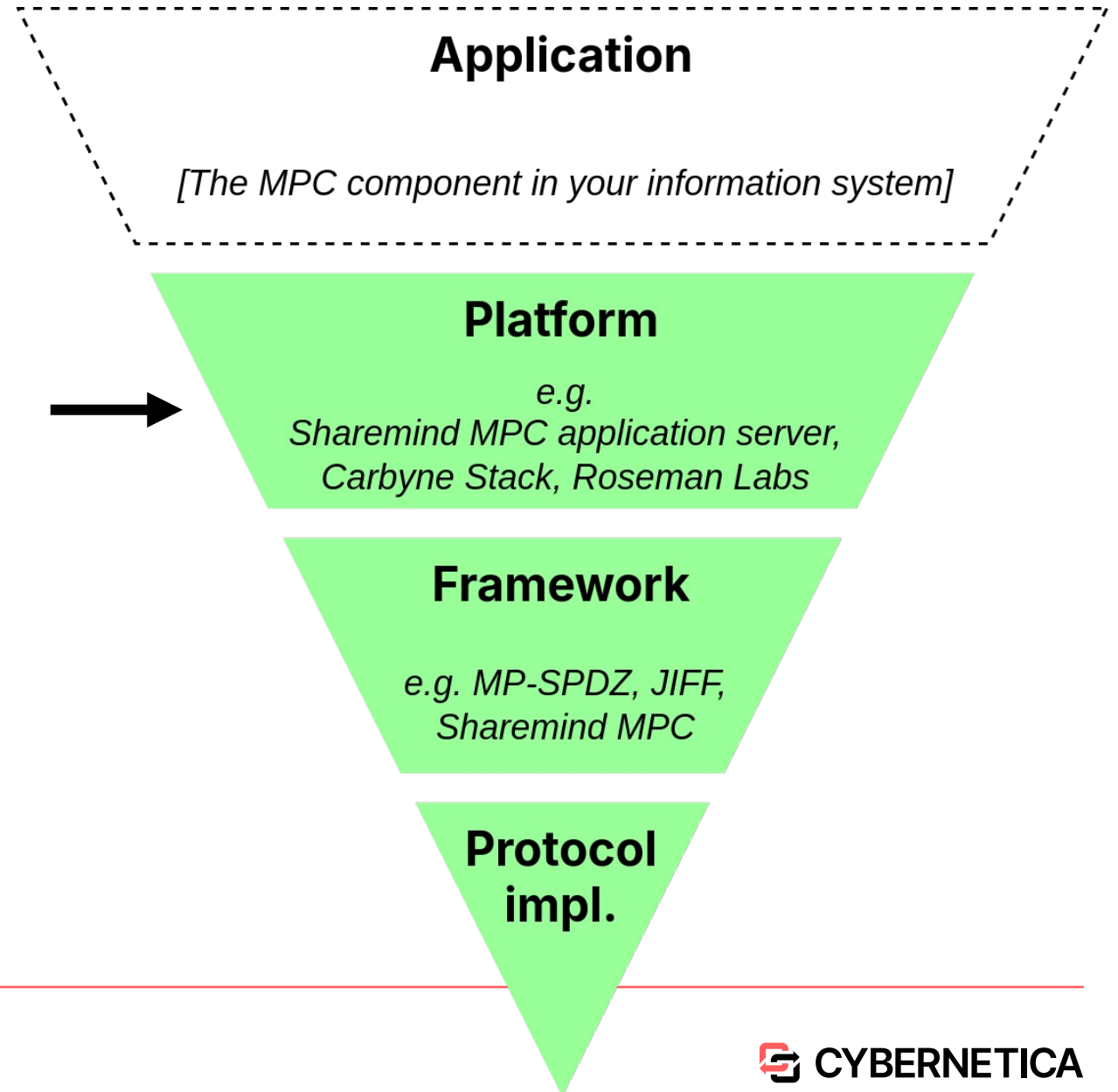
Framework

- Development toolkits
- Any spectrum of capabilities
- Many to choose from

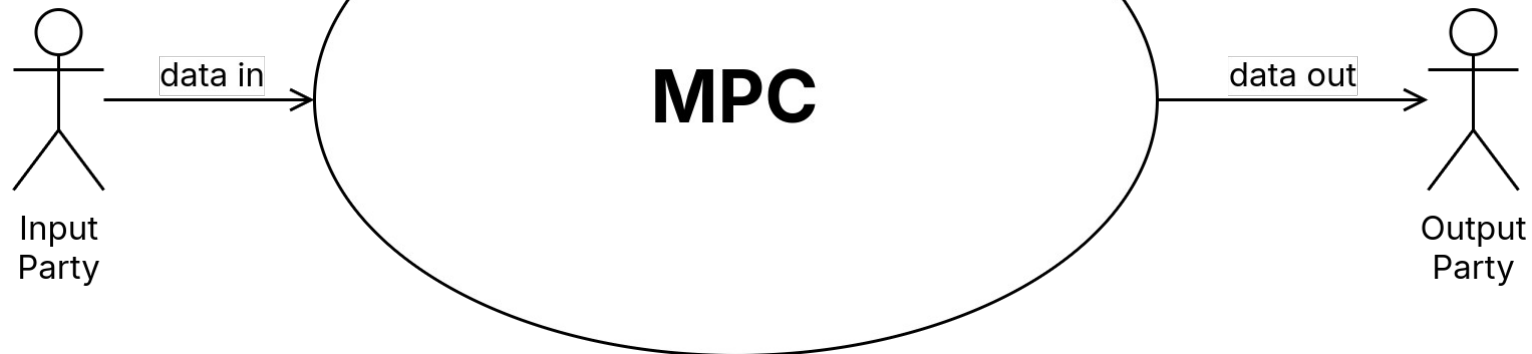


Platform

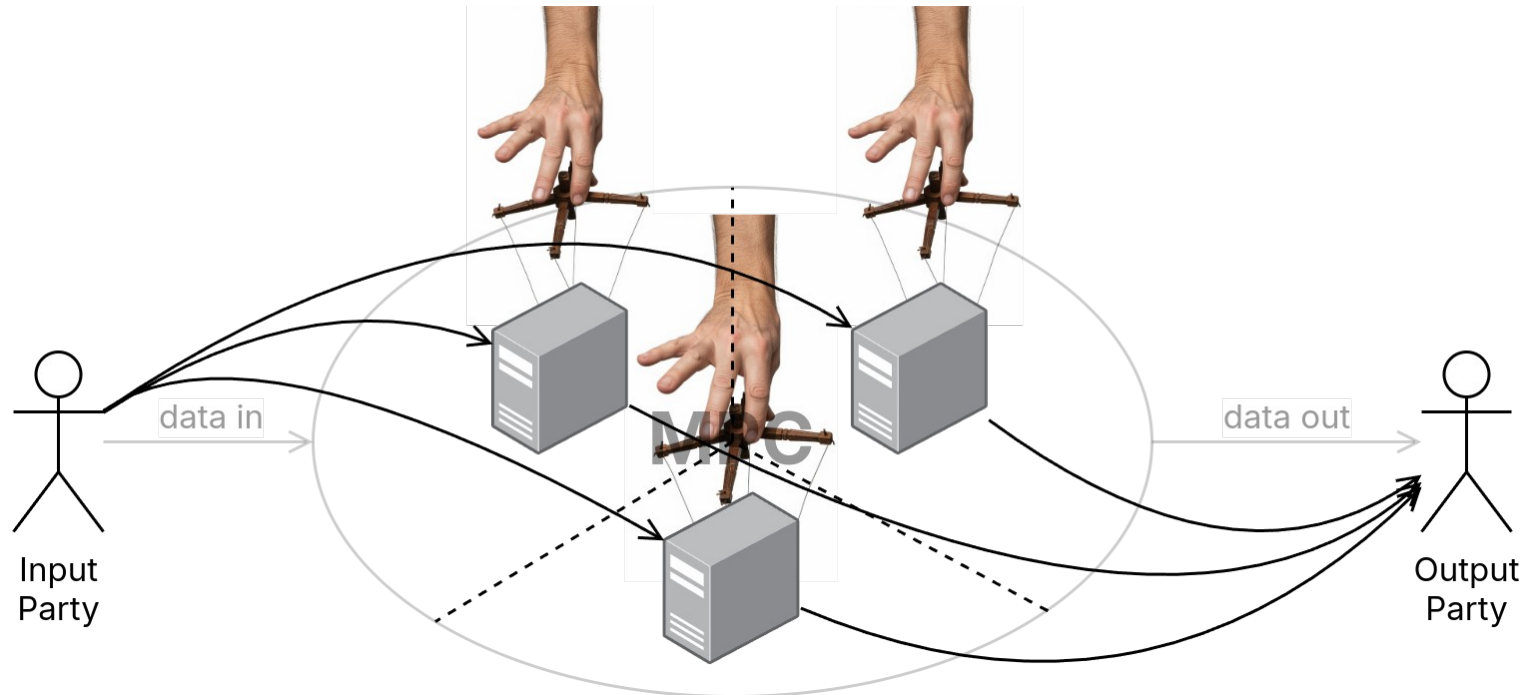
- Facade for the MPC distributed system
- Prescribes **control measures**
- For specific application scenarios



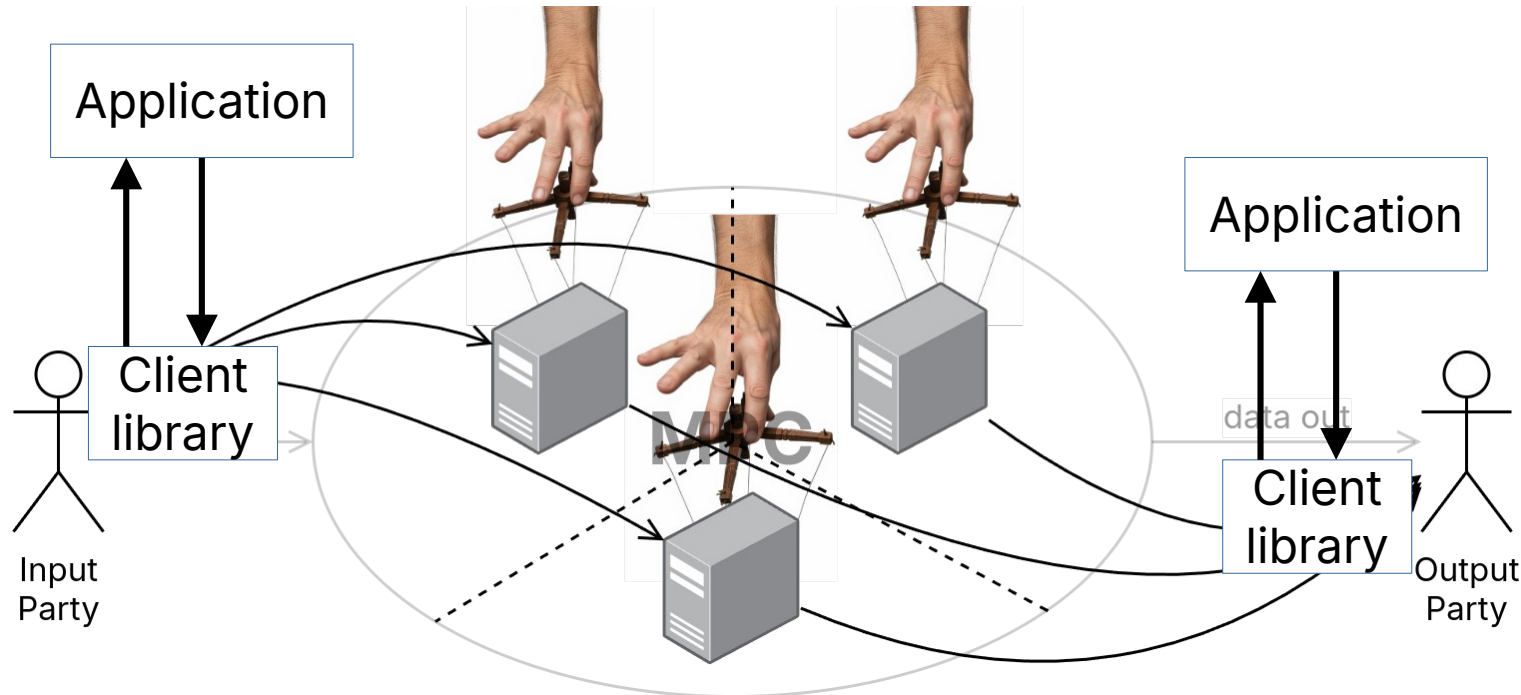
Platform control measures



Platform control measures



Platform + Application



Observation

- Frameworks don't give you obvious components to deploy
- Platforms tend to be self contained systems
- What is the better building block abstraction for MPC?

DOTADIW* for MPC

** Do One Thing And Do It Well*

MPC Engine



- instance level abstraction
- deployed at a single site
- one-shot

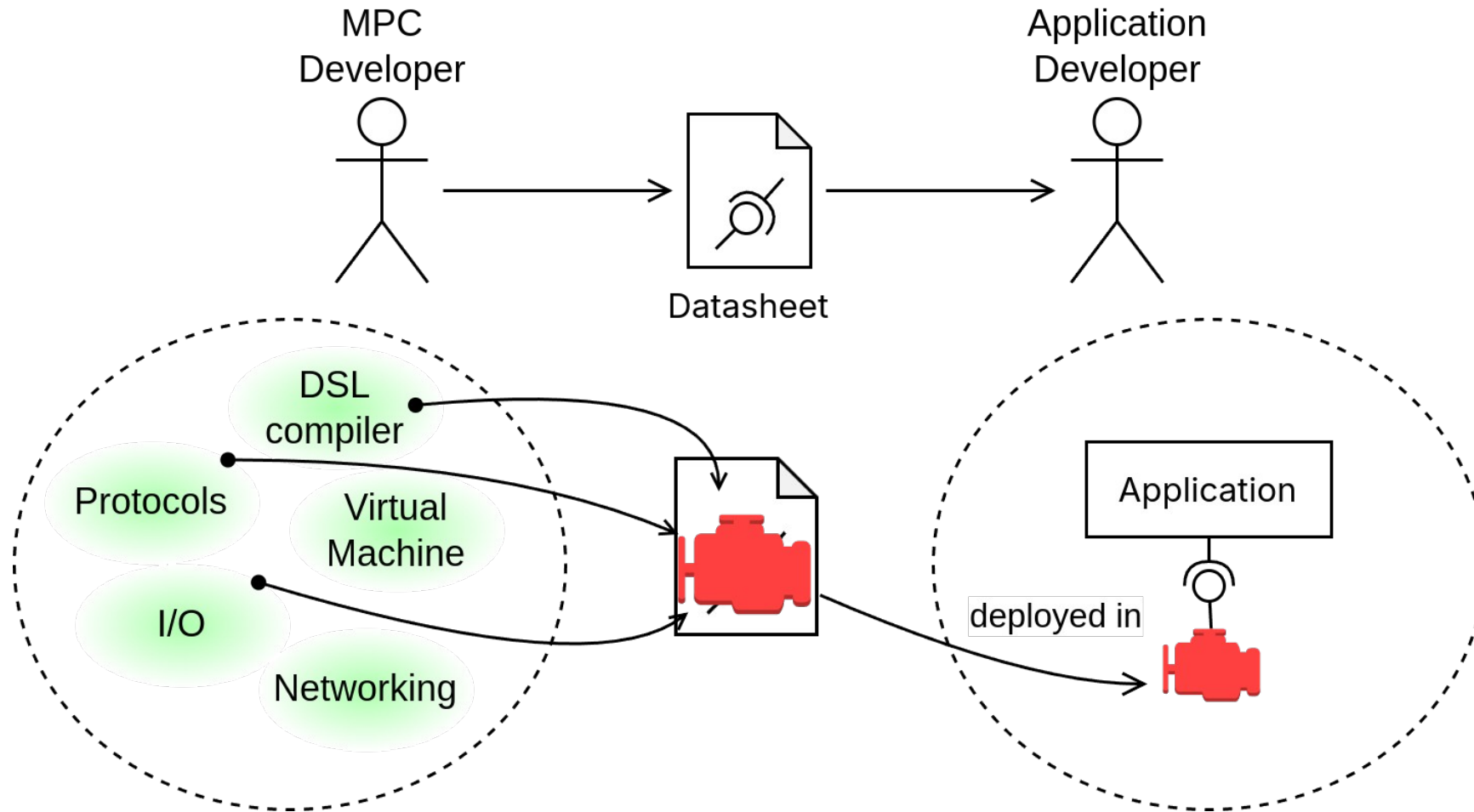


```
alice@PC ~ % ./mpc --party-b "1.2.3.4:5000"  
Success  
alice@PC ~ %
```



```
bob@PC ~ % ./mpc --party-a "5.6.7.8:5000"  
Success  
bob@PC ~ %
```

Building an MPC Engine

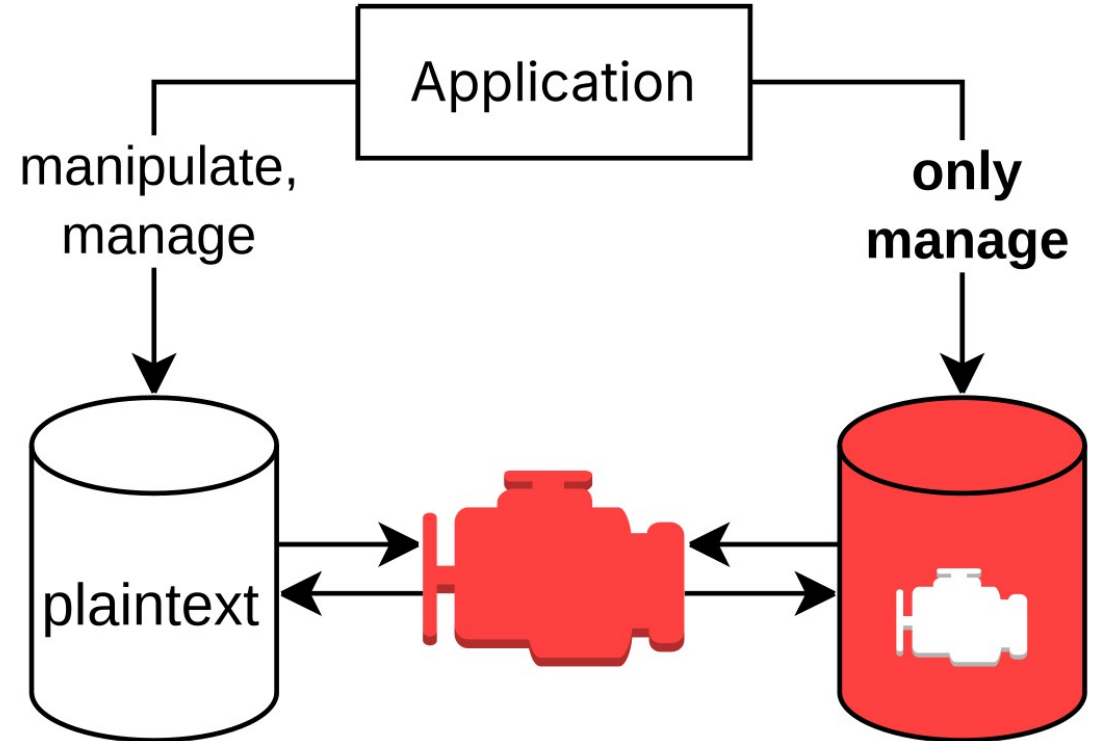


Create many MPC Engines

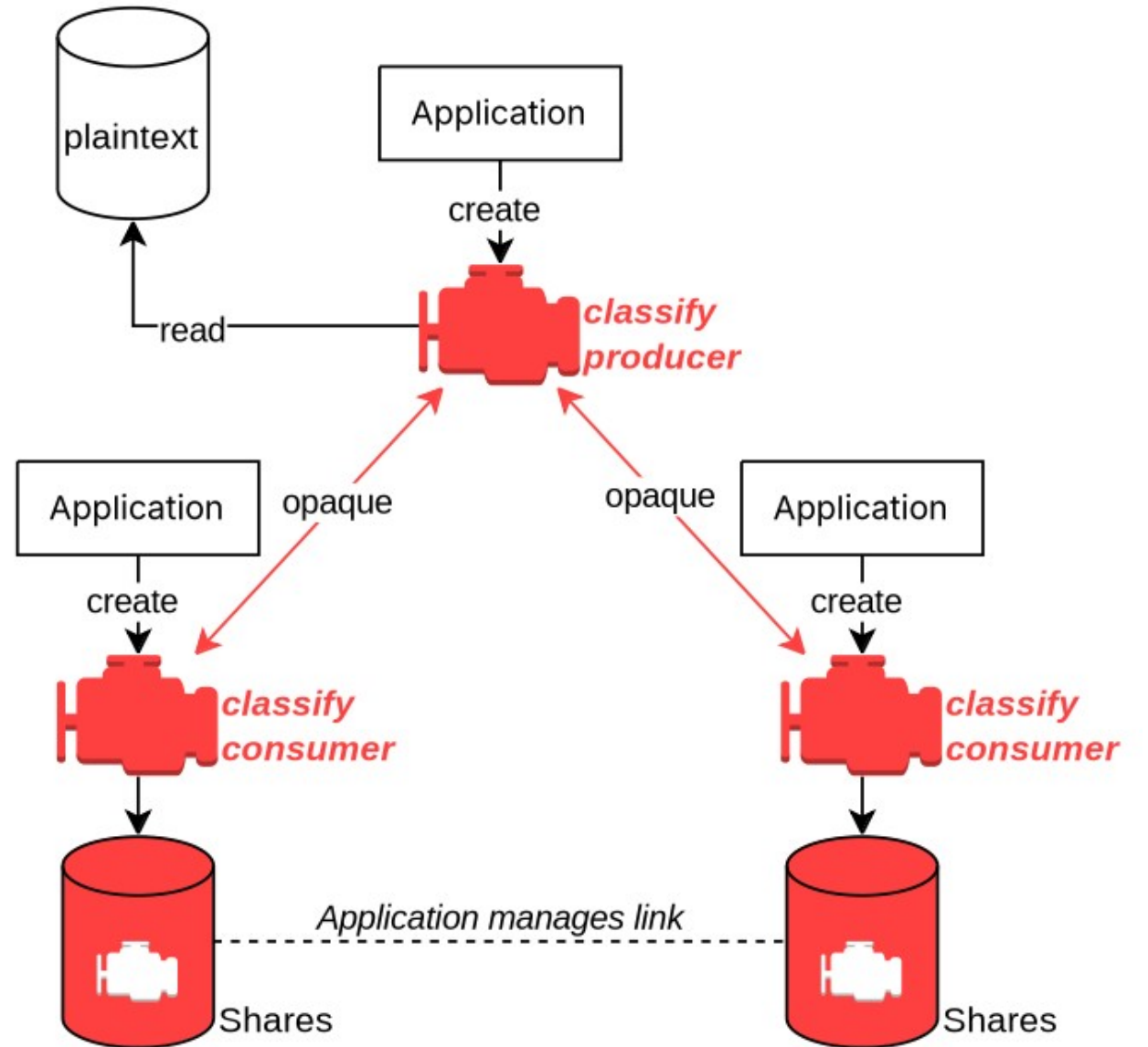
- For each functionality that should be physically separated;
- for each protocol family;
- Make note of cross-compatibility

Two kinds of data

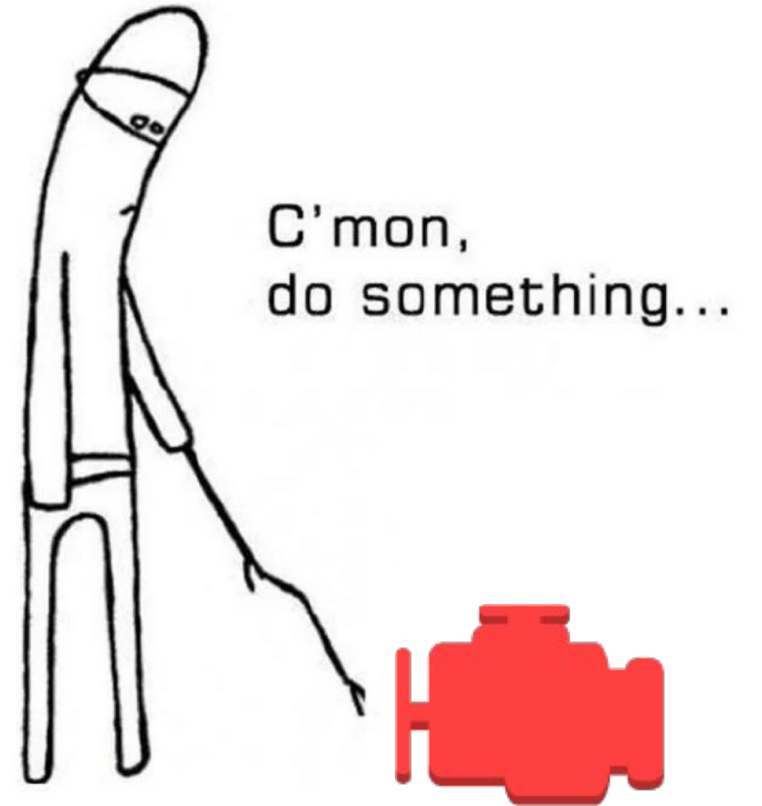
- MPC Engine creates data in some internal representation
 - meant to be ingested by another instance
 - e.g. protocol messages, persisted shares, stored correlated randomness
- MPC Engine should provide an interface to convert legible data
 - e.g. plaintext \leftrightarrow secret sharing
- The Application manages the data state



Secret sharing example



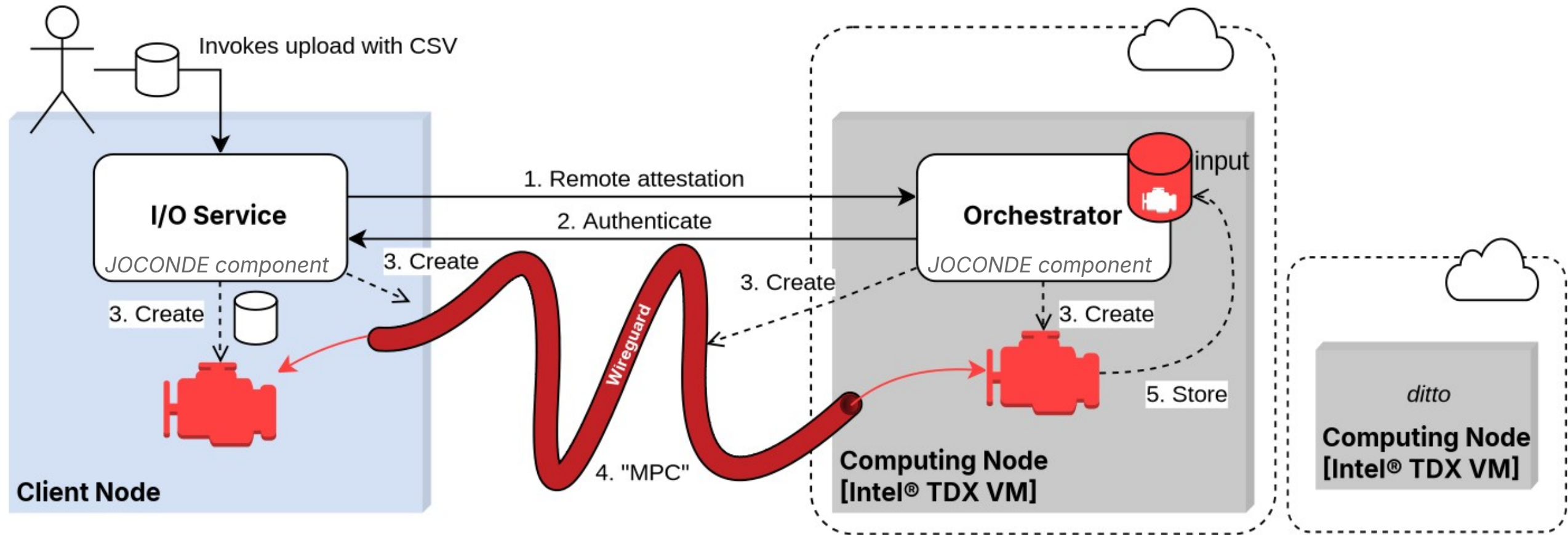
Application Level



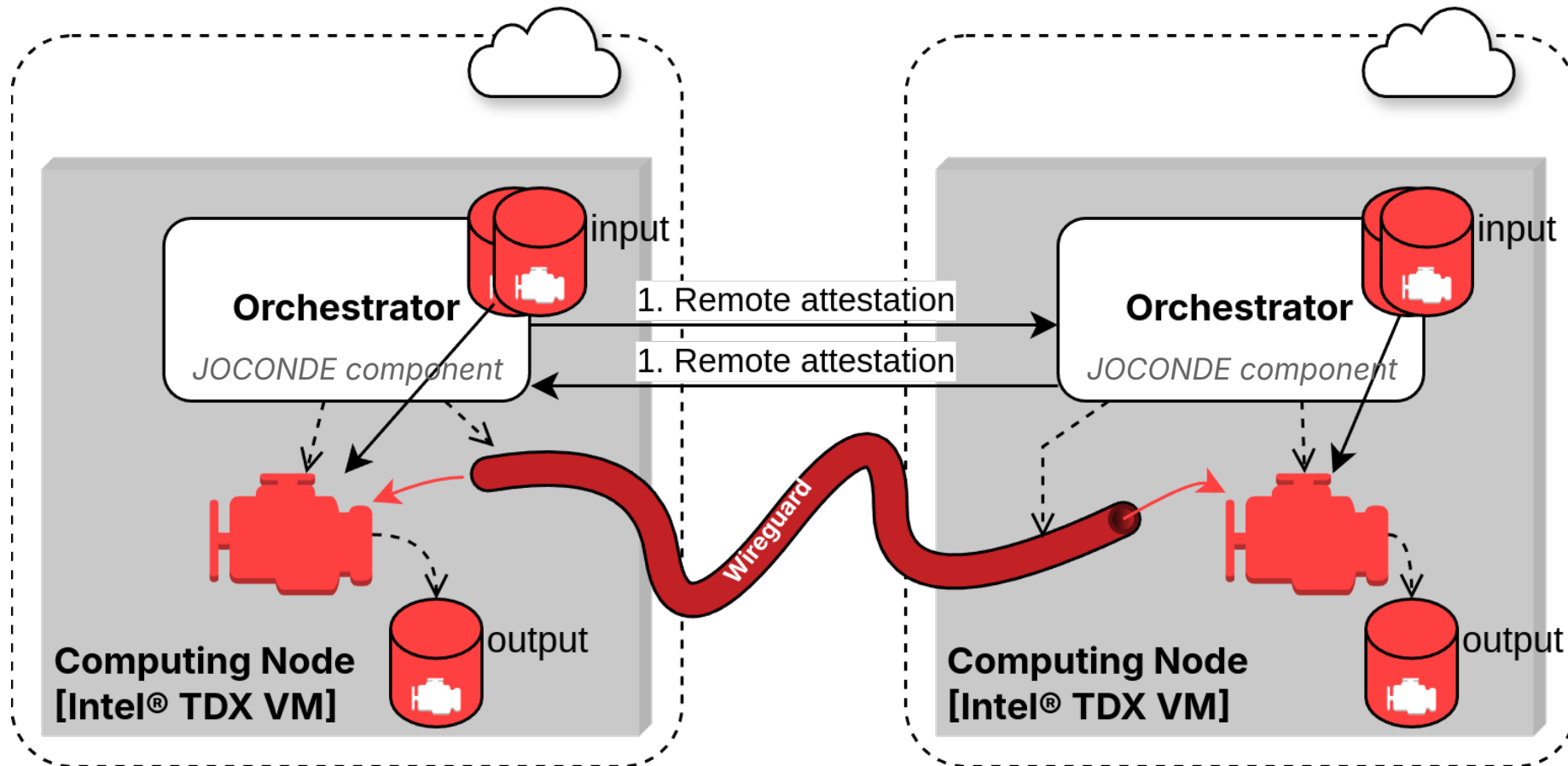
Example: JOCONDE

- MPC as a Service system tailored for official statistics production
- *Requirement: MPC-over-TEE*
 - *Solution: A custom system rather than a platform wrapper*
- *Requirement: Modular MPC (no vendor lock-in)*
 - *Soluton: A JOCONDE MPC Engine interface*

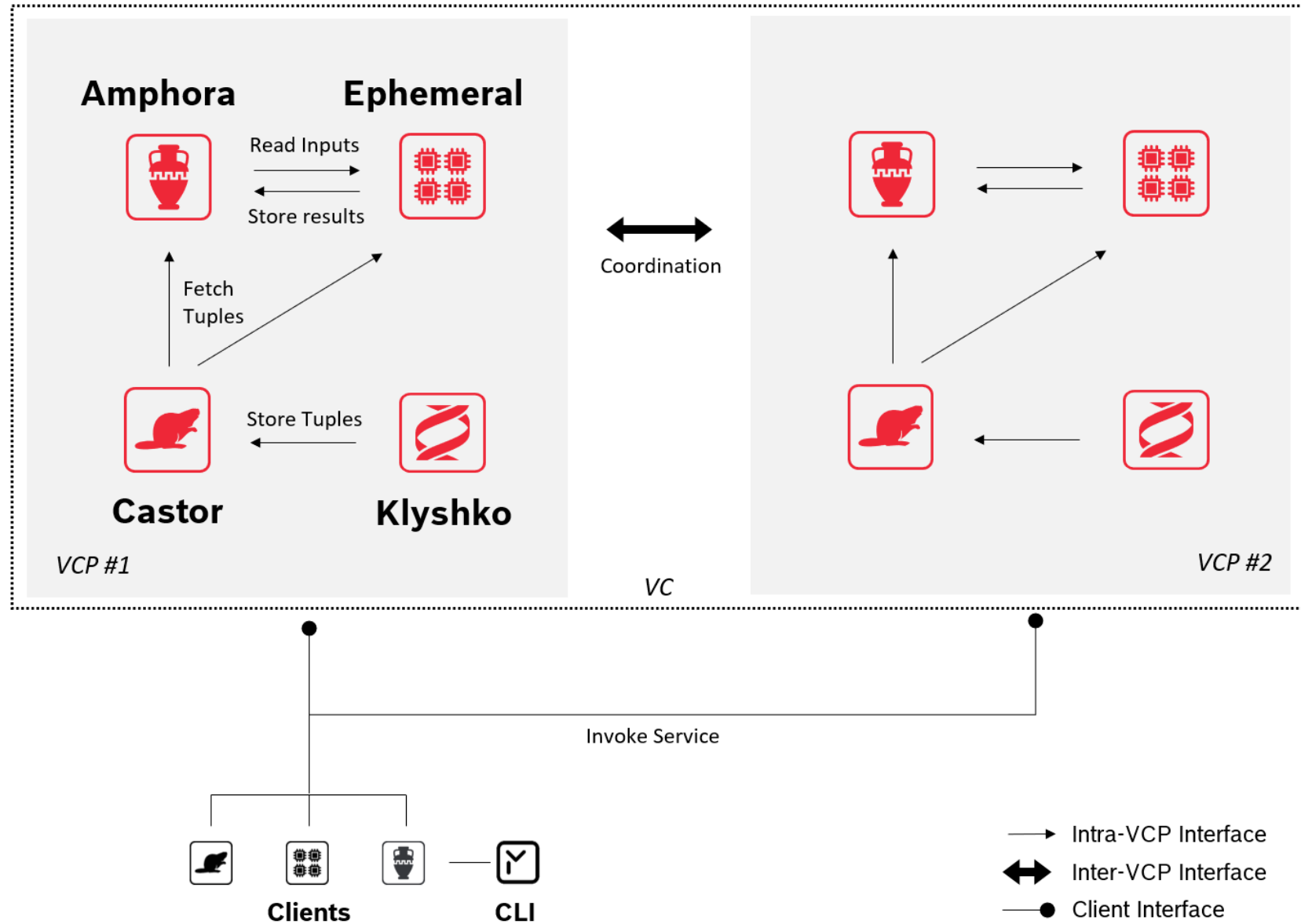
Example: JOCONDE Input Upload



Example: JOCONDE Computation



Example: Carbyne Stack



Klyshko Integration Interface (KII)

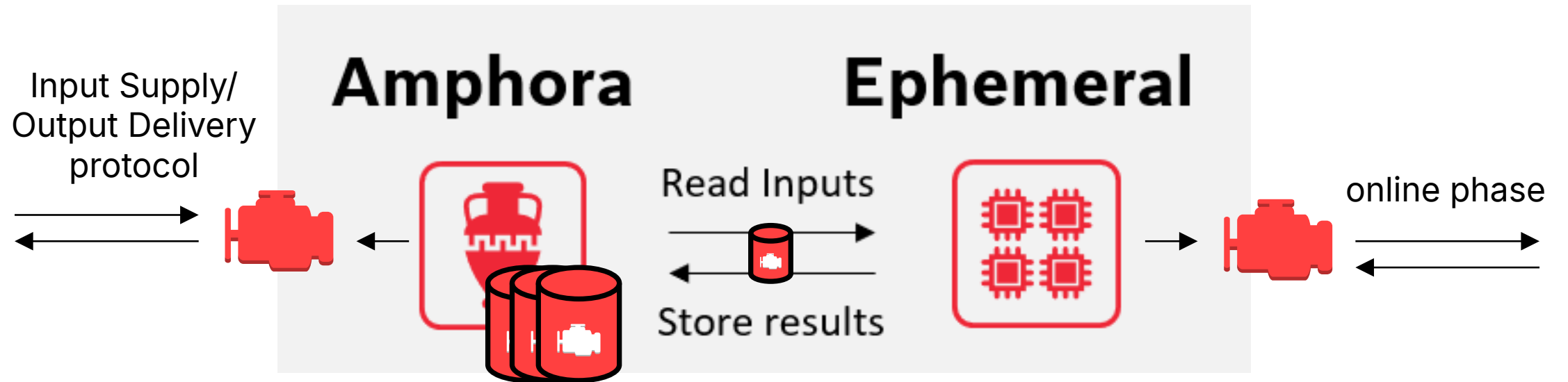
IMPORTANT: This is an initial incomplete version of the KII that is subject to change without notice. For the time being it is very much influenced by the CRGs provided as part of the [MP-SPDZ](#) project.

Klyshko has been designed to allow for easy integration of different *Correlated Randomness Generators* (CRGs). Integration is done by means of providing a docker image containing the CRG that implements the *Klyshko Integration Interface* (KII). The parameters required by the CRG are provided using a mix of environment variables and files made available to the container during execution. See below for a detailed description.



MPC Engine

EII and AII









Conclusion

- MPC Engine
 - flexible but simple component abstraction
 - only a matter of wrapping existing MPC tools
 - possible solution to modularity

Thank you!

Learn more about JOCONDE:
<https://cros.ec.europa.eu/joconde>

-  <https://cyber.ee/>
-  info@cyber.ee
-  [cybernetica](https://twitter.com/cybernetica)
-  [CyberneticaAS](https://www.facebook.com/CyberneticaAS)
-  [cybernetica_ee](https://www.instagram.com/cybernetica_ee)
-  [Cybernetica](https://www.linkedin.com/company/Cybernetica)