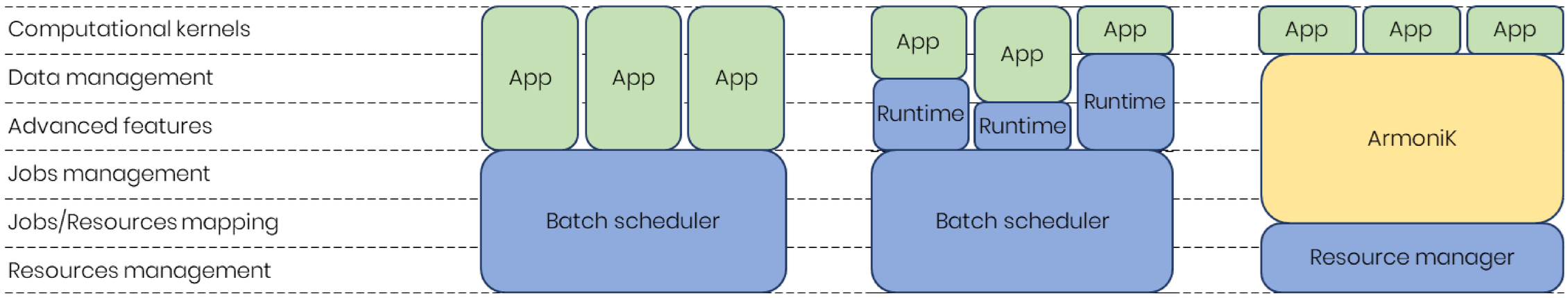
ArmoniK : une solution open source pour l'orchestration et la distribution de calculs

Jérôme Gurhem -
Wilfried Kirschenmann - Aneo, Boulogne-Billancourt, France

ArmoniK Positionning in HPC



A Serverless Many-Task Computing Platform

Serverless

Cloud service category in which the customer can use different cloud capabilities types without the customer having to provision, deploy and manage either hardware or software resources, other than providing customer application code or providing customer data.
— ISO 22123-2:2023

Many-Task Computing

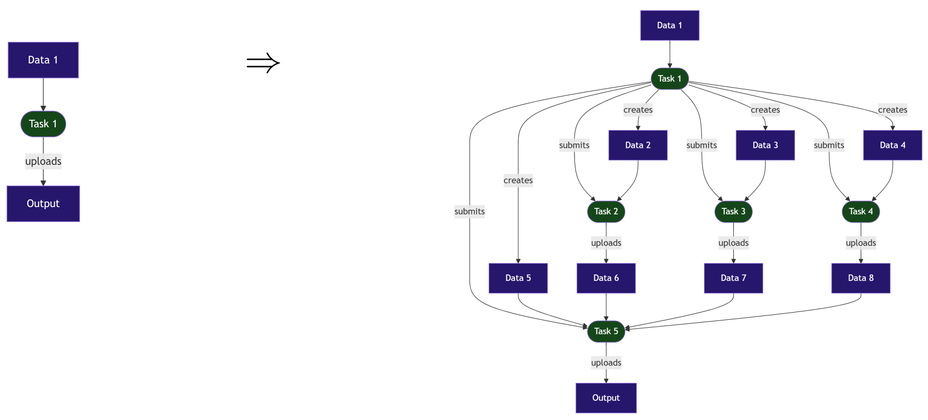
Approach that aims to bridge the gap between High-Perfomance Computing and High-Throughput Computing.
— Wikipedia

Task Definition in ArmoniK

```python
from pymonik import Pymonik, Task

if __name__ == "__main__":
    with Pymonik(endpoint="localhost:5001", partition="pymonik"):
        my_task = Task(lambda a, b: a+b, func_name="add")
        result = my_task.invoke(1, 2).wait().get()
        print(f"Result of add task: {result}")
```

Dynamic Graph

- Dependency graph is not fully known when scheduling starts
- Submissions can happen anytime
- Tasks can submit new tasks
- Tasks can delegate the production of their output to their new tasks



Dynamic Graph Example

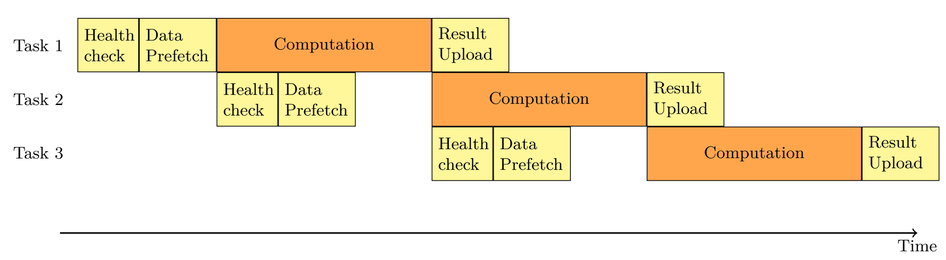
```python
from pymonik import Pymonik, task

@task
def add_one(a:int) -> int:
    return a + 1

@task
def add(a: int, b: int) -> int:
    if a <= 0:
        return b
    b_plus_one = add_one.invoke(b)
    return add.invoke(a-1, b_plus_one, delegate=True)
```

Task-based Programming in ArmoniK

- Expression of complex data-driven dependencies
- ArmoniK's **distributed scheduler** responsible for:
  - Task distribution and load balancing
  - Dependency resolution
  - Tasks execution
  - Data management (overlapping, prefetching, and checkpointing)

Computations/Comm Overlapping

- ArmoniK is responsible for tasks input and output data management
- Allows for automatic communication + scheduling/task execution overlapping
- Automatic Uncoordinated Checkpointing
- Data communications through global storage



Main features

- **Observability**: Understanding of the internal state of a system based solely on its external outputs
  - GUIs, CLIs, monitoring APIs, metrics, logs, and traces
- **Portability**: Effort to transfer an application from one environment to another
  - Officially supported languages: C#, C++, Python, Rust, Java, and JavaScript
  - Tasks on different architectures (x86, ARM, GPU, Linux, Windows), applications, environments
- **Fault Tolerance**: Ability to continue functioning without interruption when one or more nodes fail
  - Allow support for preemptible computing resources
  - Error management at the task level
- **Malleability**: Dynamic reconfiguration of the number of allocated resources during execution without interruption
- **Resource Sharing**: Share resources between applications to execute as many as possible at the same
- **Modularity**: Modules can be swapped without modifying ArmoniK's code to suit user neeeds and constraints