CLARISSE: Reforming the I/O stack of high-performance computing platforms

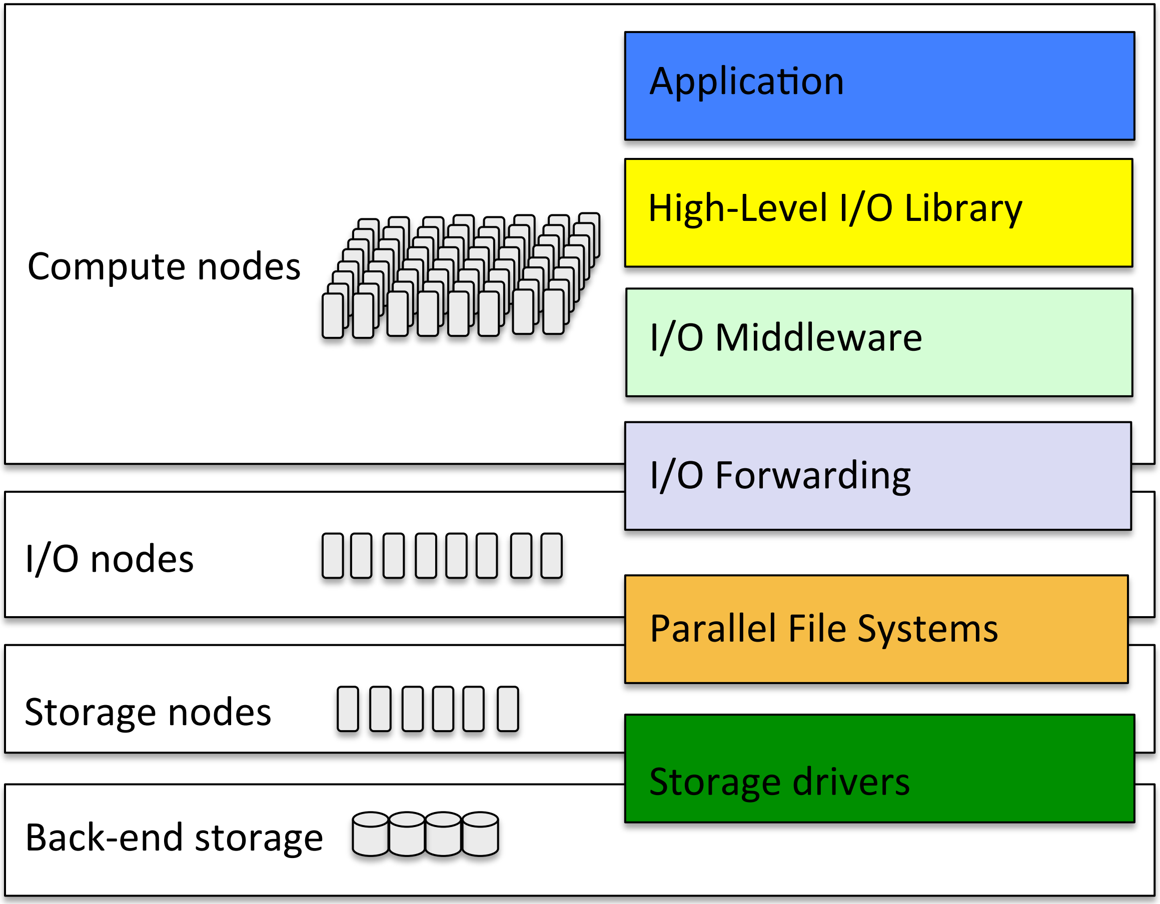
Florin Isaila

ANL & University Carlos III

Introduction

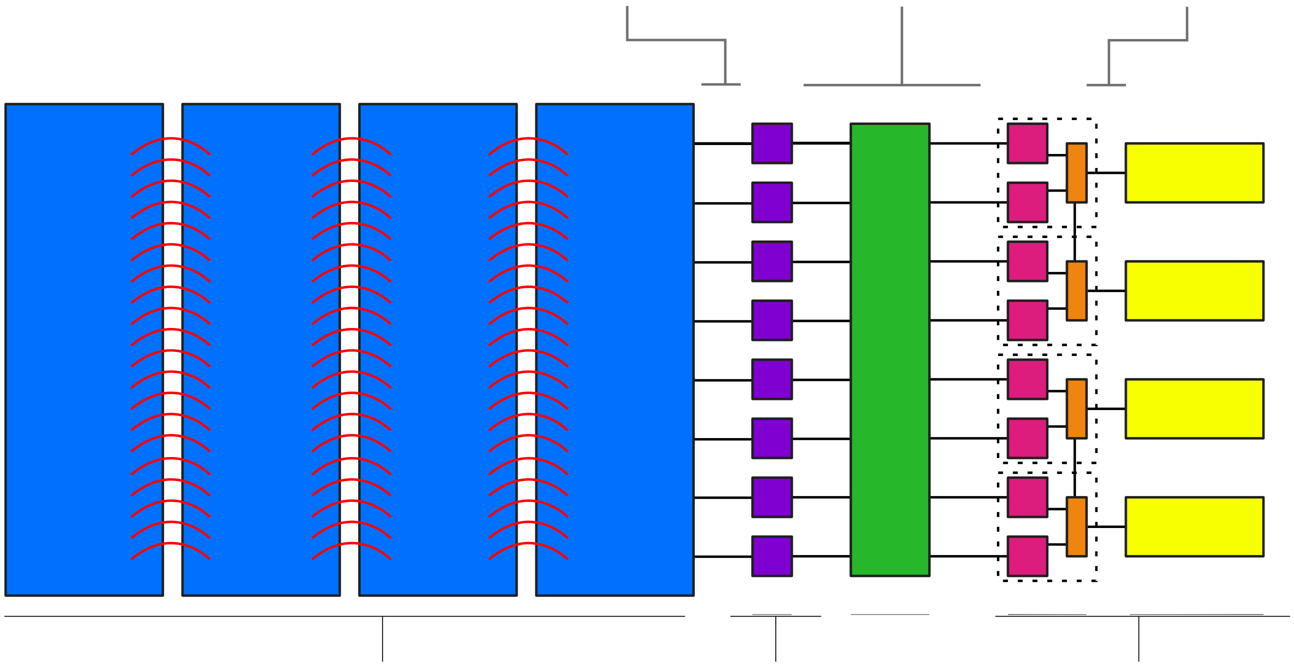
- Scientific applications (climate, genomics, high energy physics, astronomy etc.) process increasingly larger data sets
- Future high scale supercomputers need to deal efficiently with huge amounts of data
- Current I/O software stack needs to evolve in order to meet the oncoming scalability challenges

The storage I/O stack



- Maps application abstractions onto storage abstractions (e.g.: HDF5, ParallelNetCDF)
- Reduces the number of file system calls by optimizations like collective I/O (e.g.: MPI-IO)
- Offloads I/O functionality from compute nodes (e.g.: Mercury, IOFSL)
- Offer a global name space and high performance storage access (e.g.: GPFS, Lustre, PVFS)
- Block and storage object devices

Blue Gene/Q system at ANL



Compute nodes 1.6 GHz 16 cores 16GB RAM

5D torus network interconnecting compute nodes

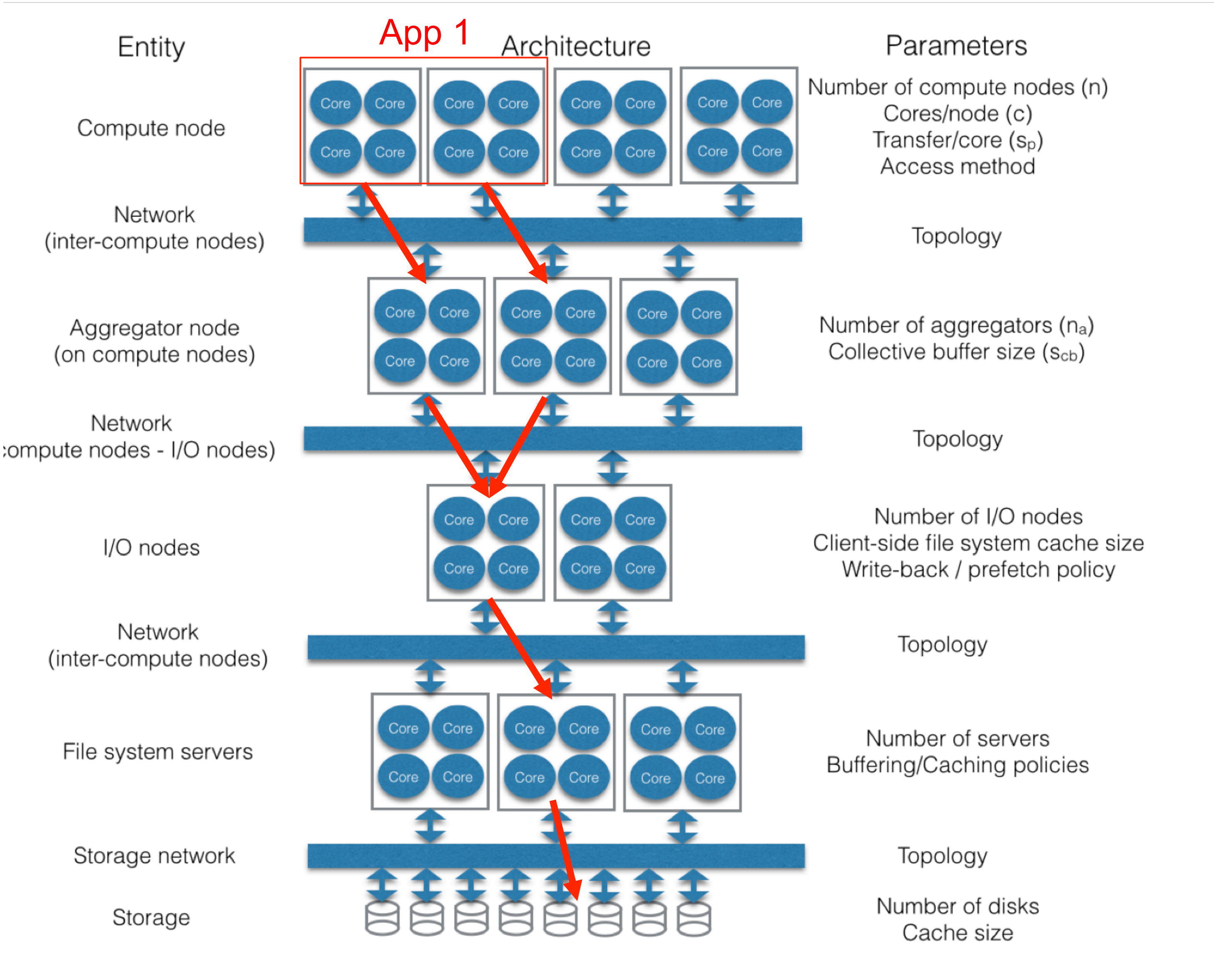
1 I/O node per 128 compute nodes

Client (I/O node) cache: 4GB

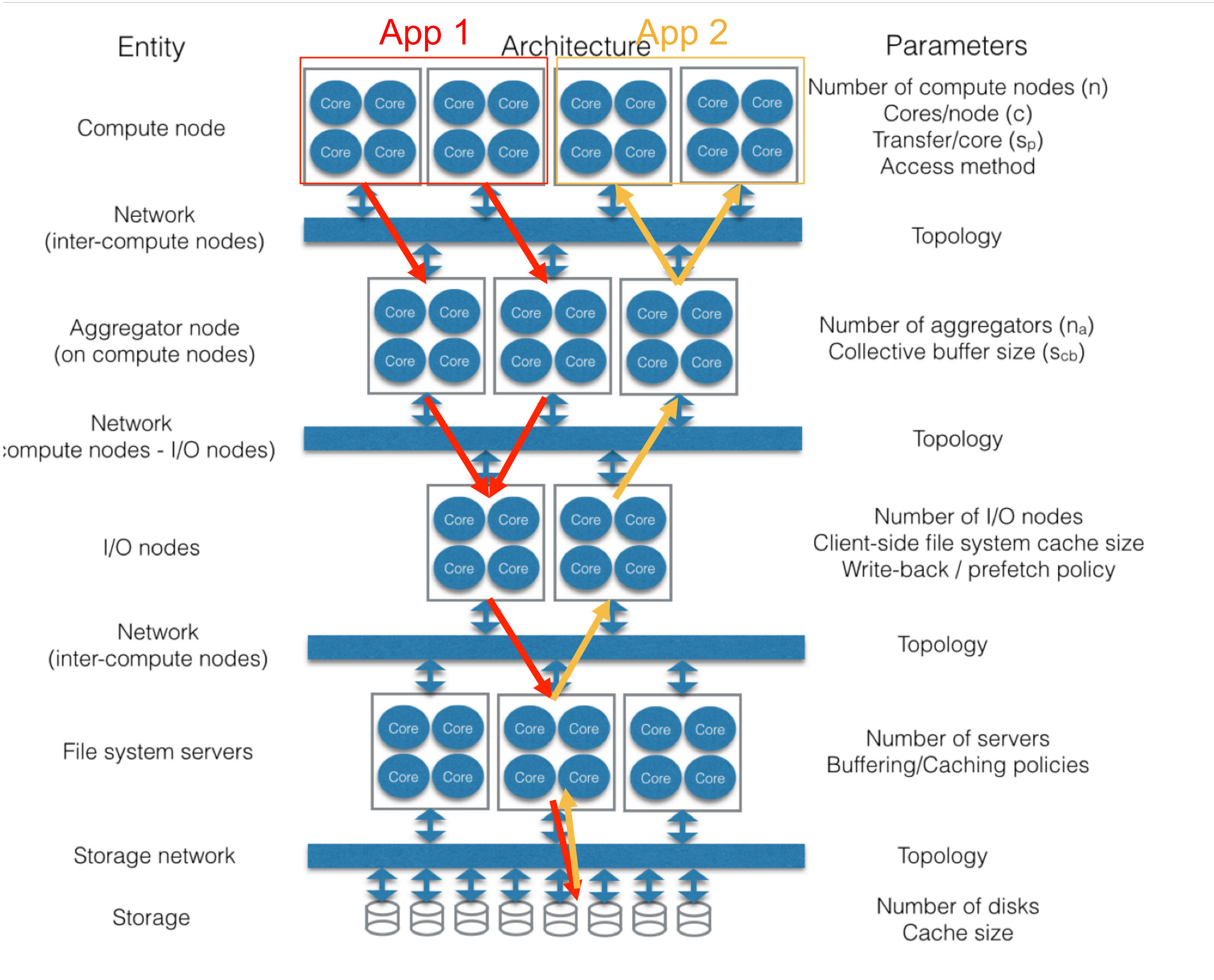
GPFS 3.5

- Data: 40 NSD SATA drives
- Max disk throughput: 250 MB/s (cache 1000 MB/s)
- Block size: 8MB

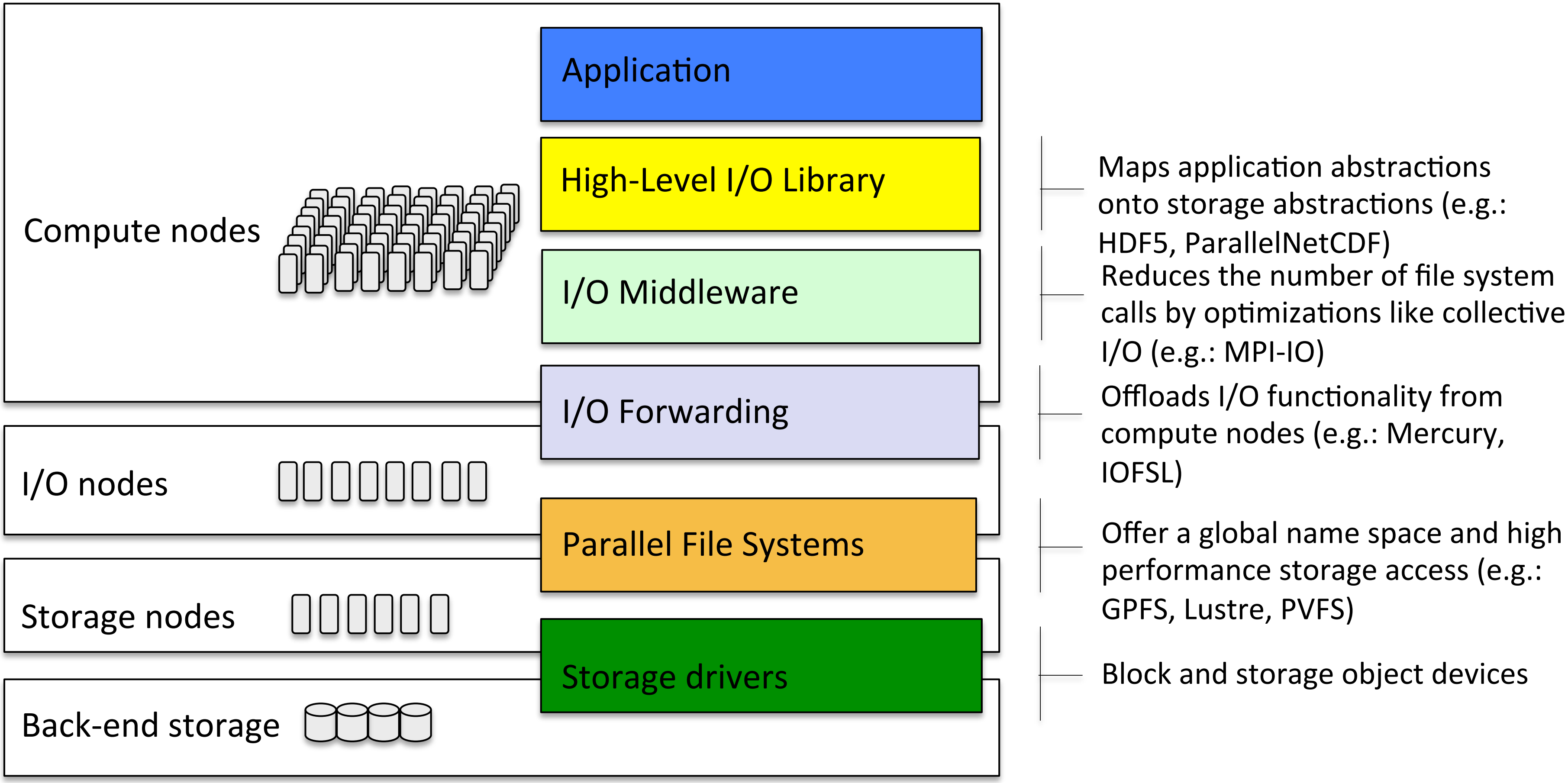
Data flow in Blue Gene/Q



Data flow in Blue Gene/Q



Current problems of storage I/O stack



- Long path from compute nodes to final storage impacts performance (latency, throughput)
- Storage I/O optimizations are local: Difficult to perform global optimizations
- Cross-layer control mechanisms are not available (e.g., for data staging, dynamic load balancing, resilience)

Outline

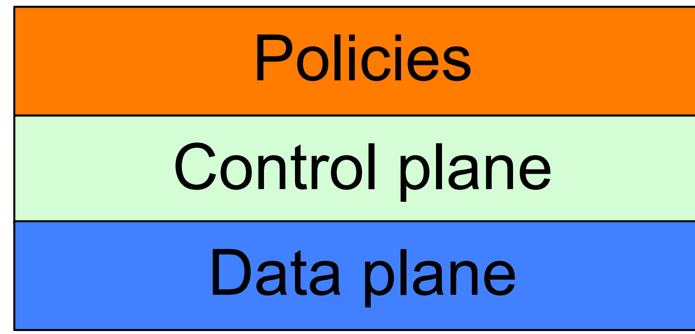
- **CLARISSE overview**
- CLARISSE abstractions
- Deployment
- Status
- Conclusions
- Future work

CLARISSE overview

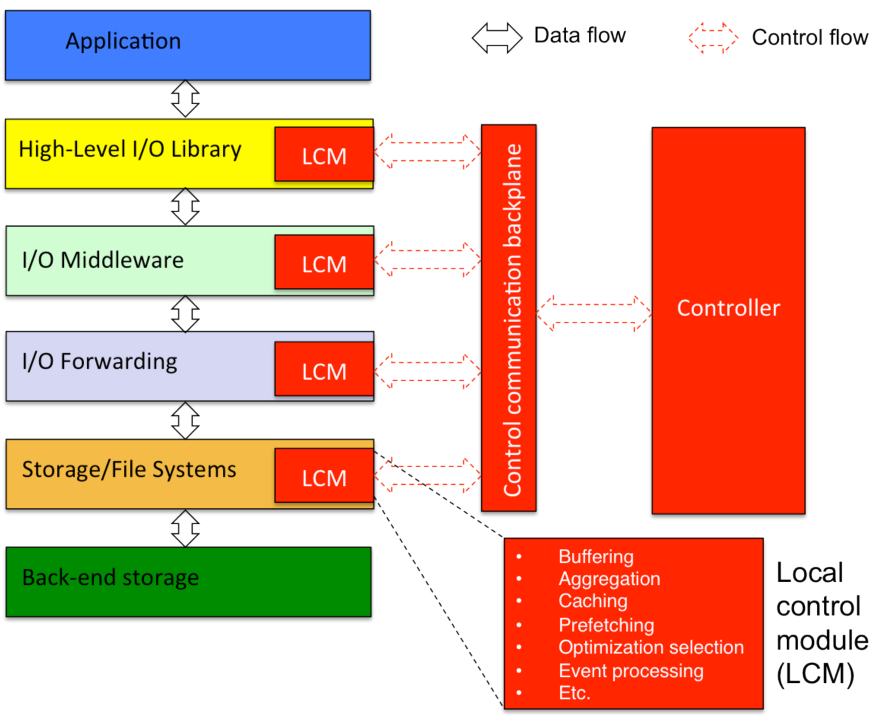
- Cross-Layer Abstractions and Runtime for I/O Software Stack (CLARISSE)
  - A 3-year project started October 2013
  - European “Marie Curie” International Outgoing Fellowship
  - Collaboration between ANL and UC3M (Spain)
- Goals
  - Enable global optimizations of the software I/O stack
  - Improve programmability
  - Facilitate extendability

CLARISSE overview

- Decouple the data and control planes
  - Data plane
  - Control plane
  - Policy
- Cross-layer abstractions and run-time
  - Facilitate the flow of control and data across the I/O stack
- Global logical view of optimizations space
  - Distributed application of global optimization
  - Facilitate the combination of local optimizations based on a global view



CLARISSE overview



- Data plane (this talk)
  - Design novel cross layer control abstractions and mechanisms for supporting data flow optimizations
    - Data aggregation (e.g., collective I/O)
    - buffering / caching, data staging
    - load balance
    - data locality (e.g. in-situ and in-transit data processing)
  - Parallel data-flows based on the these abstractions

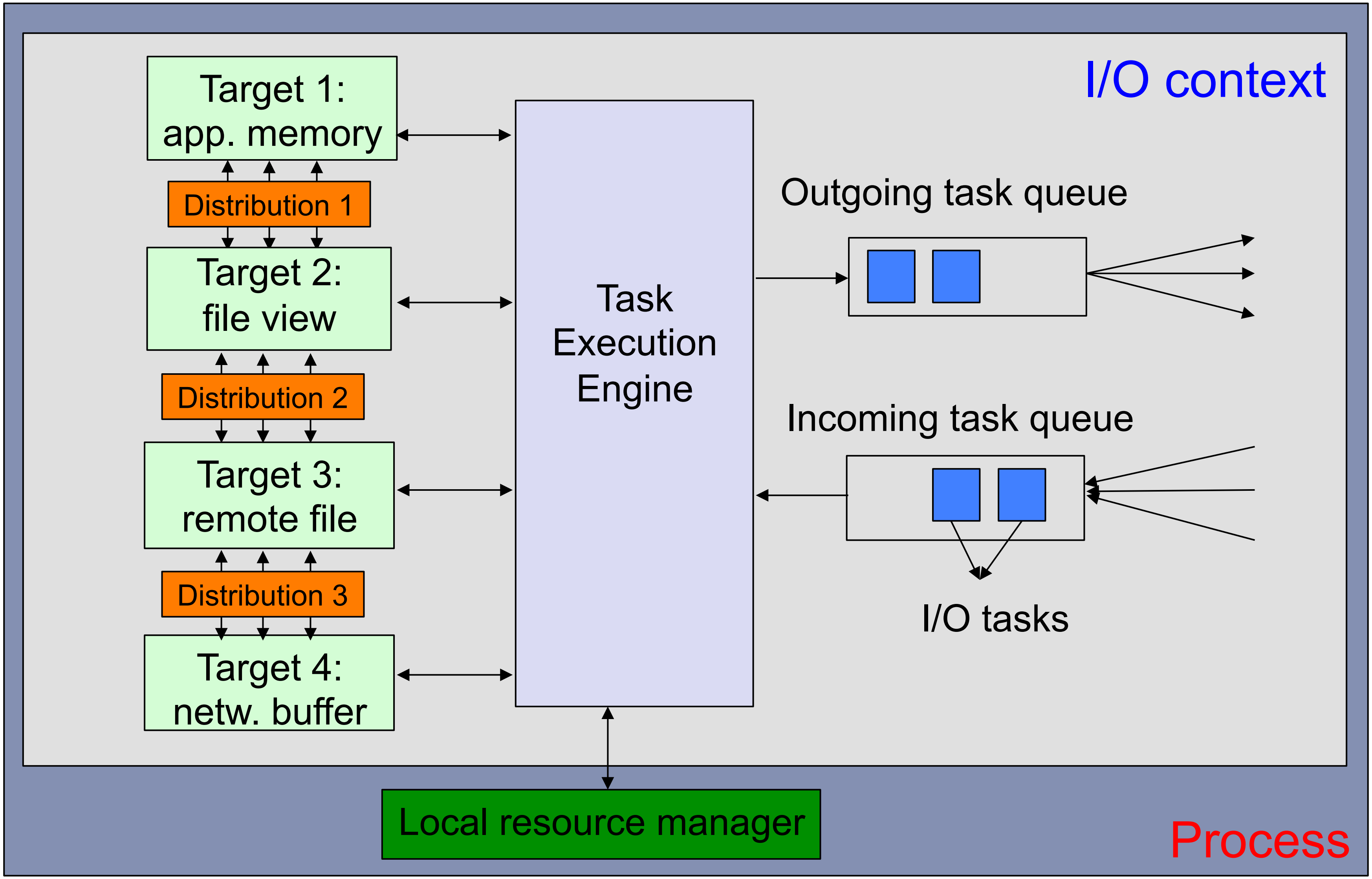
Outline

- CLARISSE overview
- **CLARISSE abstractions**
- Deployment
- Status
- Conclusions
- Future work

CLARISSE Abstractions

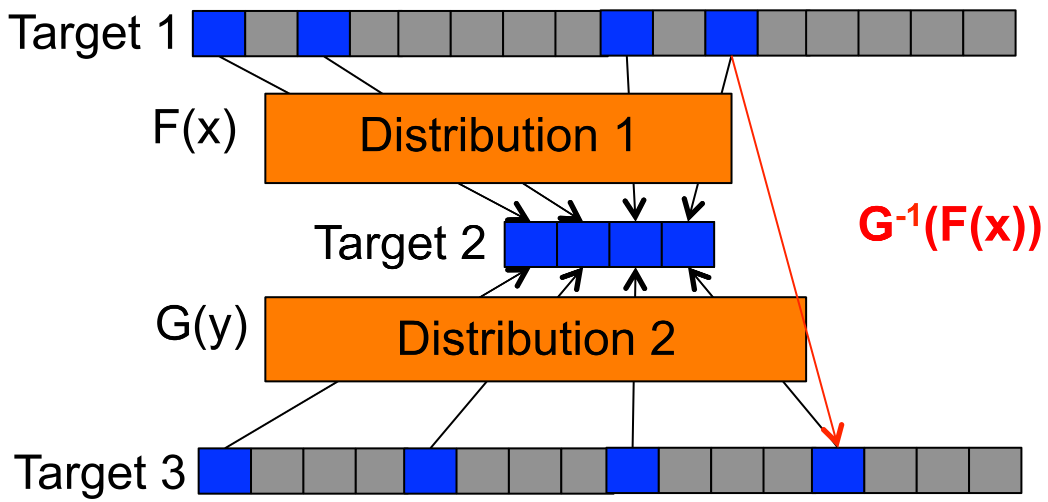
- Five main abstractions
  - Targets
  - Distributions
  - I/O contexts
  - I/O tasks
  - Task queues
- Objectives
  - Represent the storage I/O activity in terms of these abstractions
  - Offer a logically centralized view of these abstractions
- Probably not realistic to expect to have these abstractions present at all stack layers

CLARISSE abstractions



Abstractions for the I/O stack

- Targets
  - Virtual linear spaces
  - Memory, files, storage objects, network buffers
  - put/get interface
- Distributions
  - Mappings between targets
  - Mapping functions from non-contiguous regions to a contiguous region
  - Composing the functions for arbitrary non-contiguous to non-contiguous mappings

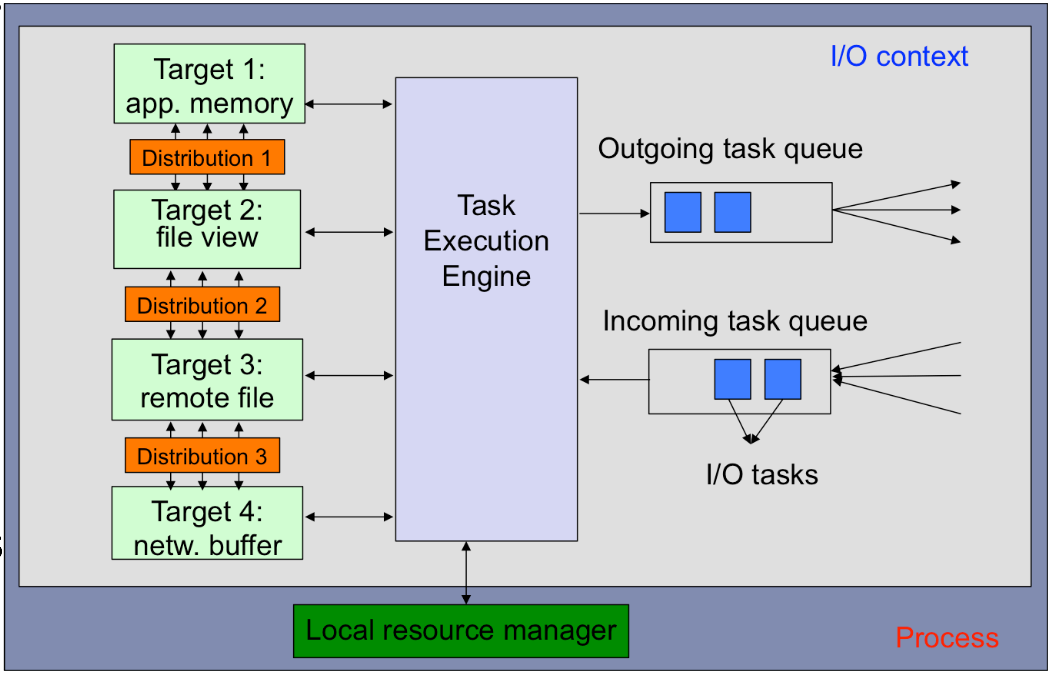


I/O tasks and task queues

- I/O task
  - I/O related set of actions between a local I/O context and a remote I/O context
  - Example of I/O tasks
    - Redistribute data between memory application and network buffer
    - Run custom defined computation (e.g. Code-on-Demand of EVPath)
    - Send/receive data from remote nodes, file/storage systems
- Task queues
  - Queues containing the tasks to be processed by the task engine
  - Control exposed to the control backplane

I/O contexts

- Local to a node
- Link between one local target and N remote targets
  - Local representation of vertices in the data flow graph
- Two I/O task queues
  - Incoming
  - Outgoing
- Assigned system resources
  - E.g. memory, cores
- Task execution engine
  - Execute task actions
  - Enforce an I/O scheduling policy



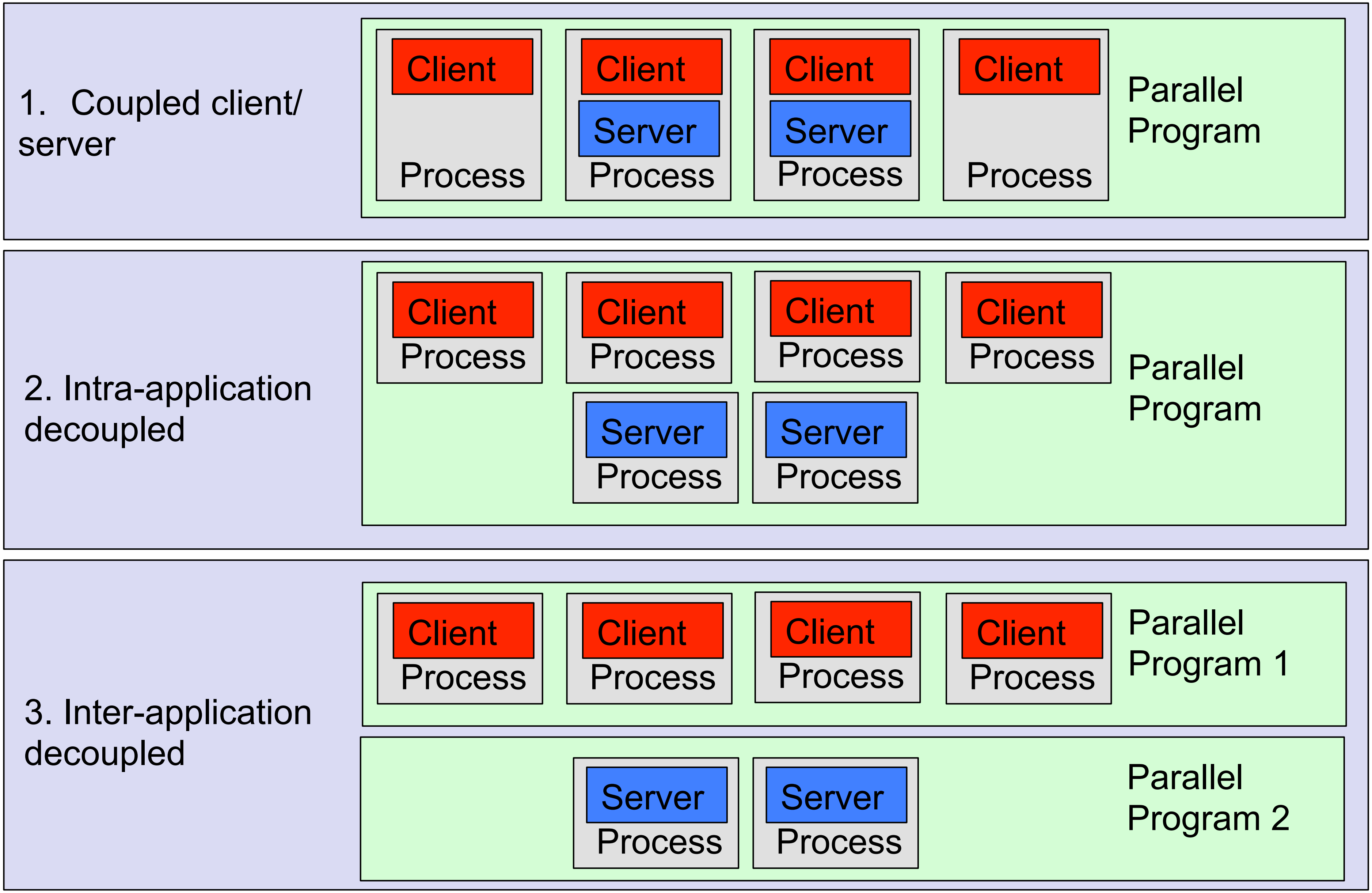
Outline

- CLARISSE overview
- CLARISSE abstractions
- **Deployment**
- Status
- Conclusions
- Future work

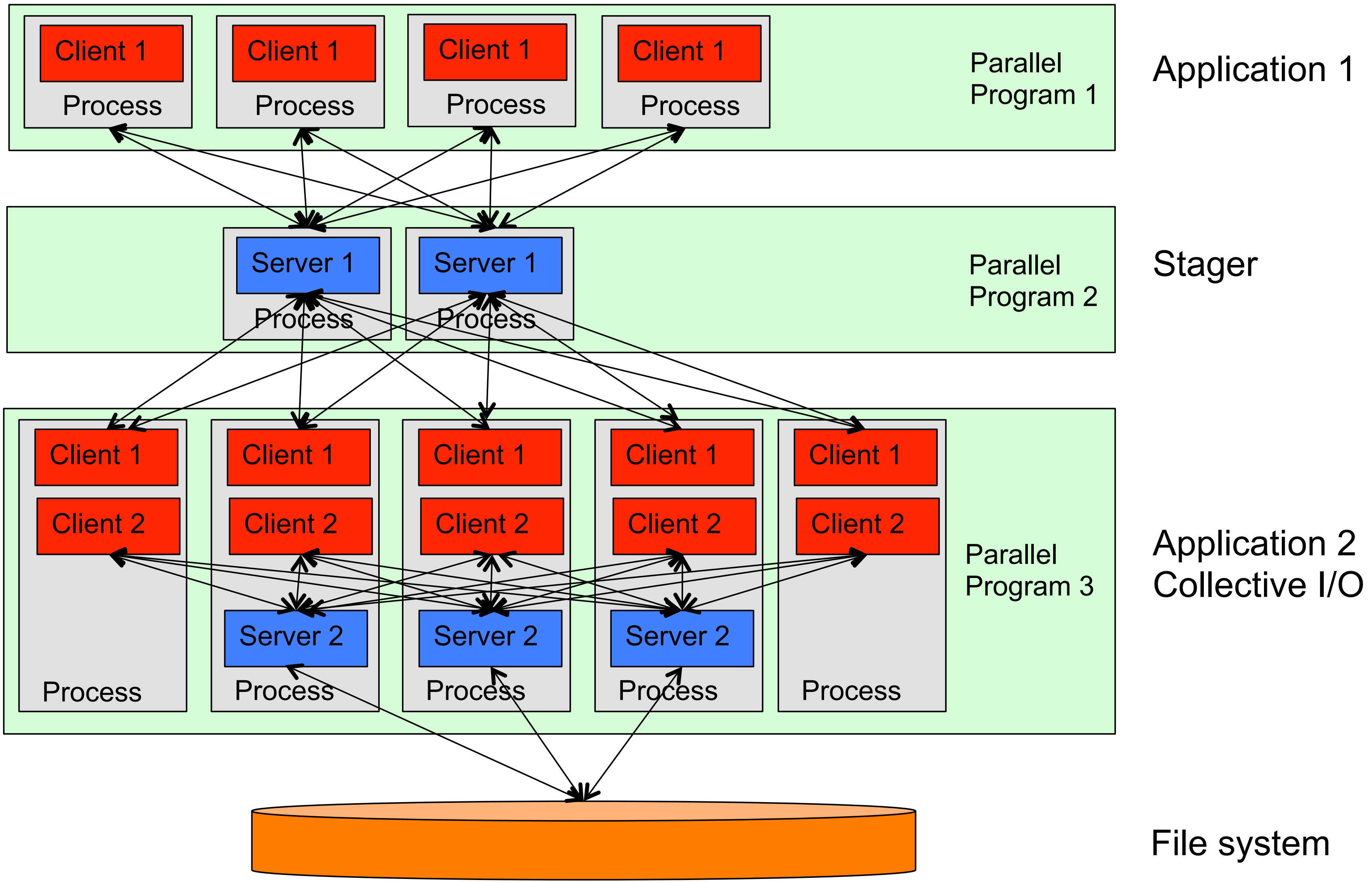
Deployment

- A process can be:
  - A CLARISSE server (a process that serves remotely data access calls)
  - A CLARISSE client (a process that issues data access calls)
  - Both a server and a client
- Three combinations of parallel servers / parallel clients
  1. Coupled client-server
     - Client and server run in the same process (multi-threaded or not)
     - Locally shared memory
  2. Intra-application decoupled
     - One parallel application
     - Separate client and server processes
  3. Inter-application decoupled
     - Connect different parallel applications or applications to parallel storage systems
     - Construct parallel workflows

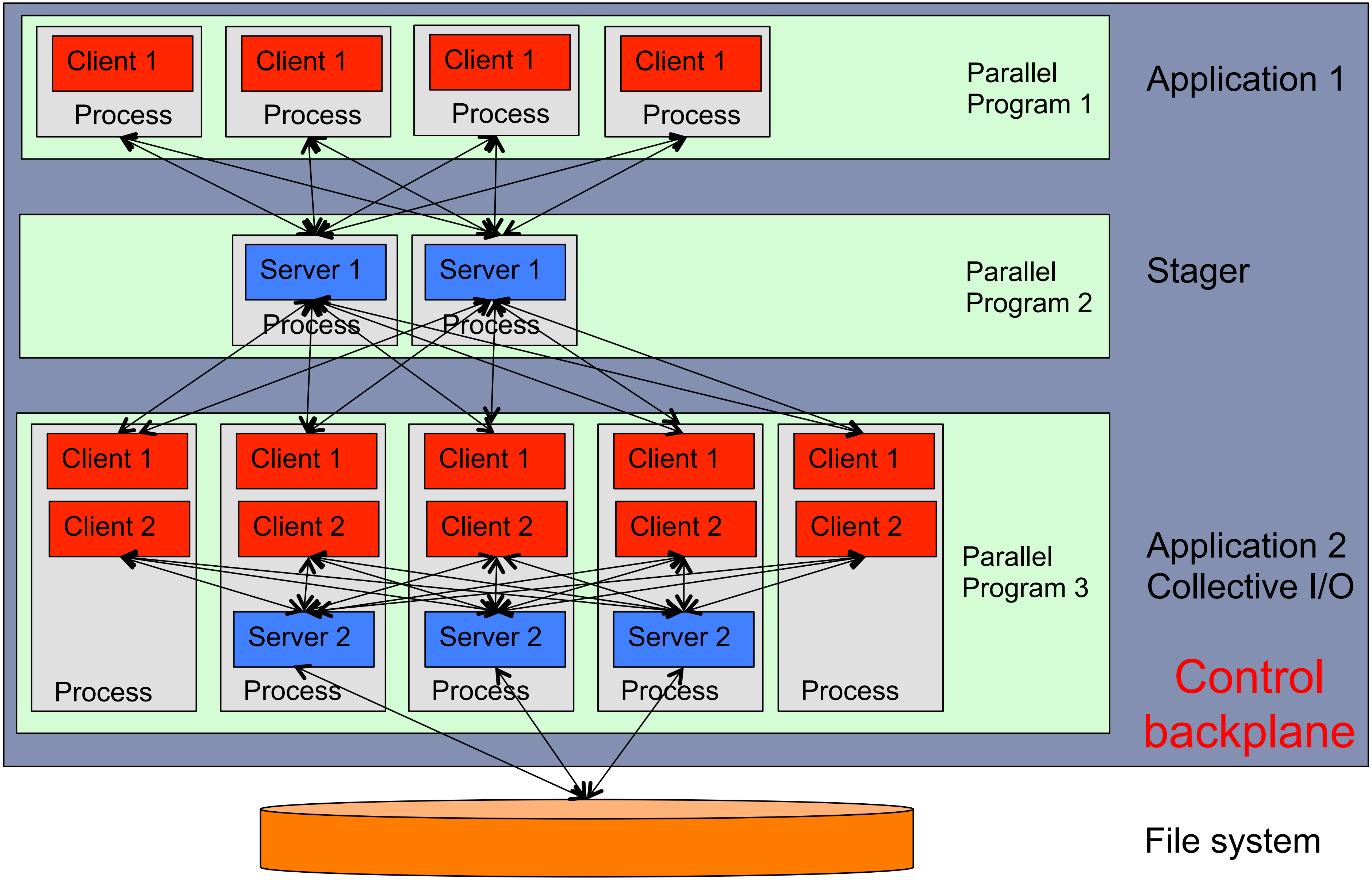
Clarisse deployment



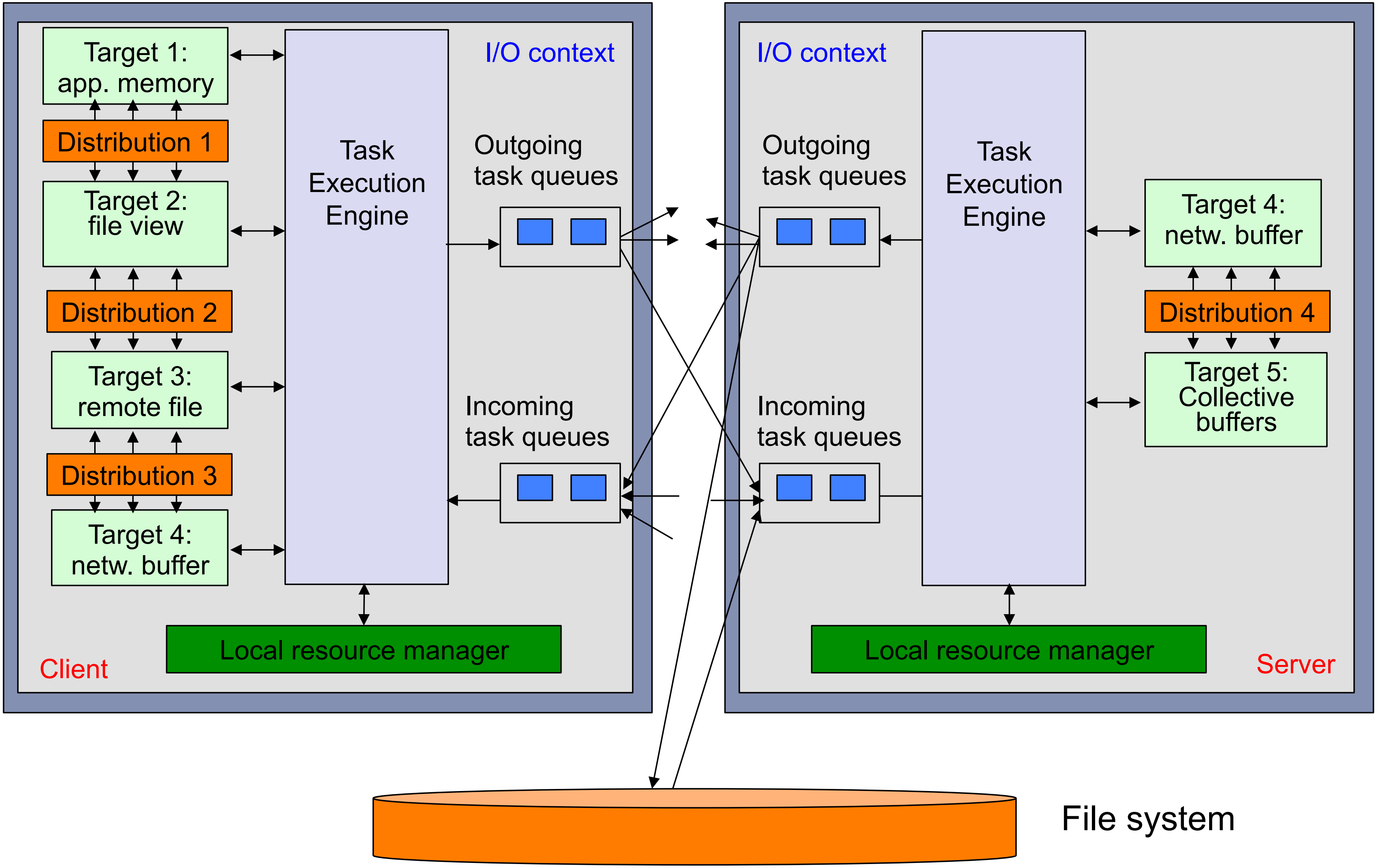
Clarisse parallel data flow example



Clarisse parallel data flow example



Collective I/O implementation



Mechanisms and policies

- Data aggregation:
  - An I/O context can associate N remote targets to a local target: data from the N targets can be aggregated in the local target
  - Collective I/O
- Buffering / Caching
  - An I/O context can be dynamically assigned buffers from a local resource manager
  - Local caching policy for each I/O context
- Data staging:
  - Staging policies can be defined for each context
  - Cross context coordination through global controller
- Exploitation of the data locality by in-situ and in-transit data processing.
  - On-demand code deployed to an I/O context
- Workflows of parallel applications

Status

- One implementation of all abstractions
- Deployments
  1. Coupled client-server: a fully implemented collective I/O method
  2. Intra-application decoupled: most functionality implemented
  3. Inter-application decoupled: most functionality implemented
- Communication
  - MPI (implemented)
  - EvPath (near future)

Related work

- Software Defined Networking (e.g. Open Flow): global control based on separation of control and data flow
- I/O Flow (Microsoft Research): A Software Defined Storage Architecture for virtualized data centers
- Fast Forward (Intel et al.): redesign of the storage I/O stack
- Argo (Argonne et al.), Hobbes (Sandia et al.): system software for exascale based on an OS/Run-time environment
- Decaf (Argonne): decoupling of tightly coupled workflows
- In-situ and in-transit processing: Data Spaces (Rutgers), Flexpath (Georgia Tech), FlowVR (INRIA), Damaris (INRIA), Glean (Argonne)

Conclusions

- CLARISSE project approach of redesigning the I/O stack aiming to facilitate global optimizations, programmability, and extendability.
- A set of novel abstractions for enabling global optimizations for
  - data aggregation
  - buffering
  - staging
  - exploitation of the data locality by in-situ and in-transit data processing
- Three types of deployments targeting to efficiently support both
  - independent parallel applications
  - ensembles of parallel applications

Ongoing and future work

- Fully implement and evaluate the three types of deployments and combinations of them
  - Evaluate application workflows (e.g. HACC)
- Decaf
  - Alternative implementation of Decaf primitives: aggregator, buffering, pipelining, selectors
  - Leverage distributions in Decaf
- Control plane
  - Argo control plane (Beacon/Exposé)
- Data flow optimizations
  - Coordinated data staging
- Additional communication support
  - EvPath, Mercury

Thank you

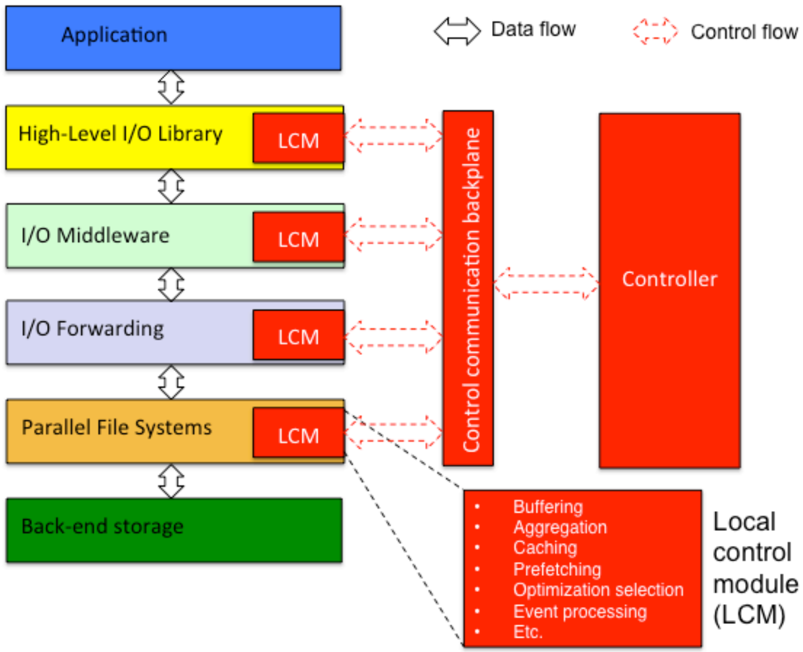
CLARISSE is NOT:

- A full redesign of the I/O stack
  - These abstractions should work as well:
    - With some current layers of I/O stack
    - On top of existing functionality of the I/O stack
- A library for the end user
  - Rather a set of abstractions for facilitating global optimizations when writing system software

- Contributions over other works
  - Separation data flow –control flow
  - Reimplementation of optimizations at various levels (e.g. collective I/O in middleware)
  - Enable global optimizations
  - Arbitrary redistributions

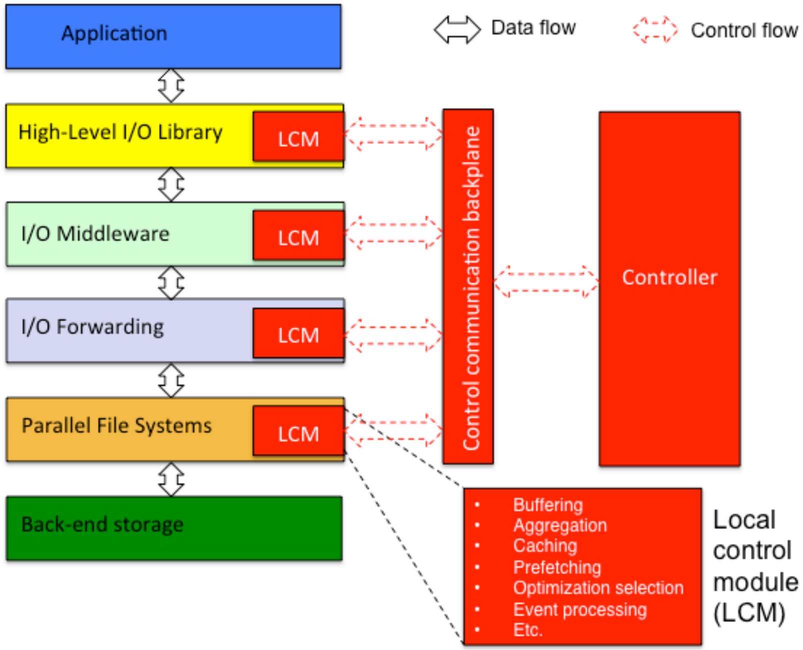
Controller

- Cross-layer logic
- Gathers information
  - Data related attributes
  - System characteristics
  - Available optimizations
  - Run-time events and statistics
- Global optimization inference
- Distributes the control decisions to LCMs
- API for defining control policies
  - e.g. prediction, load-aware, data locality-aware



Local control module (LCM)

- Enforces control at I/O software stack layers
- Selects local optimizations
- Event processing
- Collects run-time information and forwards them to controller
- Optional
  - Not all layers have to provide it
  - Each layer can use a default policy



Communication backplane

- Coordination between controller and I/O software stack layers
- Generic
- Event-based: react to user-defined criteria
- Low overhead: low interference with the system workings
- Potential candidates
  - CIFTS: A Coordinated Infrastructure for Fault-Tolerant Systems
  - Beacon: The communication backplane of ARGO, an exascale operating system