

VELOC: Very Low Overhead Checkpointing System

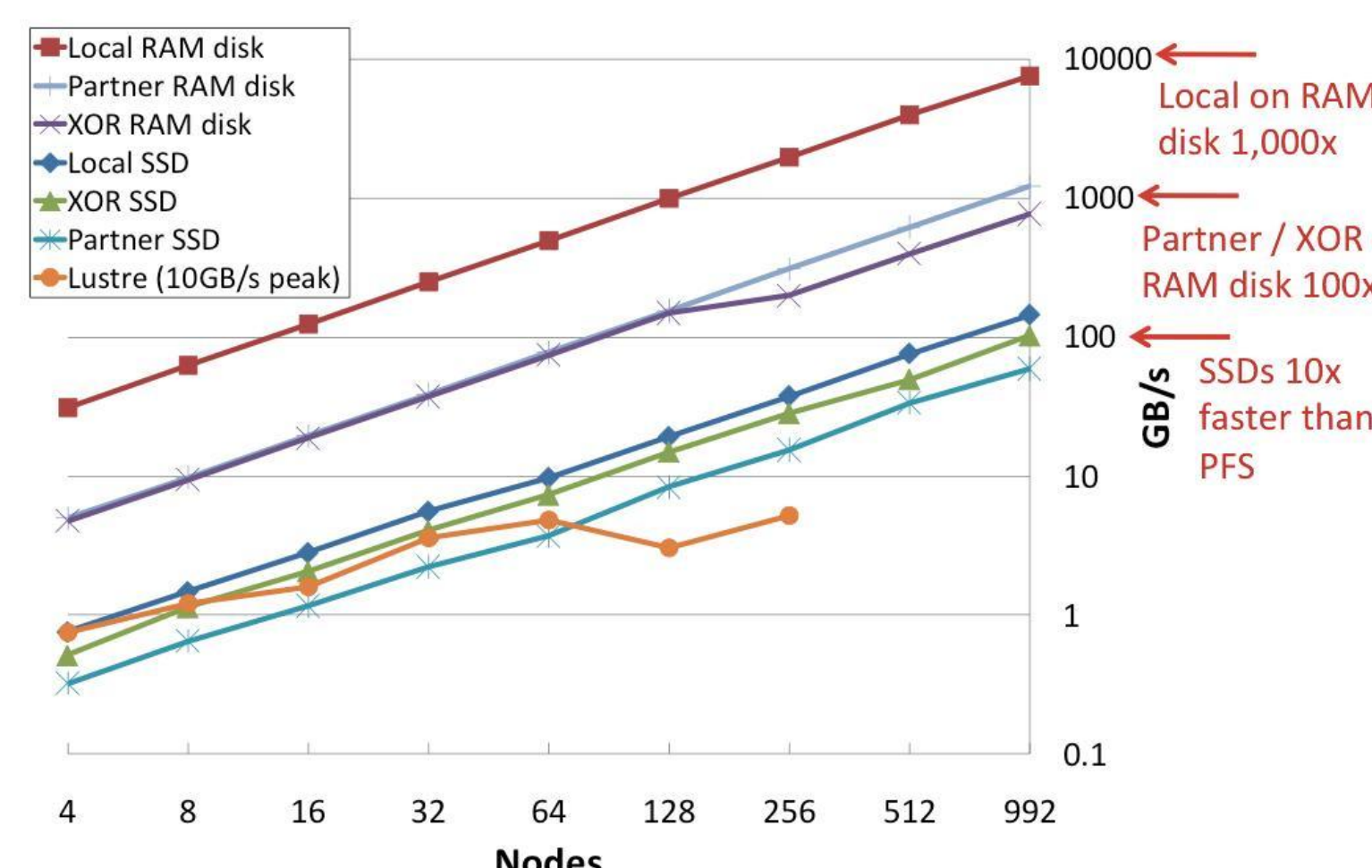
ANL: Bogdan Nicolae, Franck Cappello
LLNL: Adam Moody, Elsa Gonsiorowski, Kathryn Mohror



Description

Problem Statement: Extreme scale simulations need to checkpoint periodically for fault-tolerance. But overhead of checkpoint/restart is increasing beyond acceptable and requires non-trivial optimizations

Goal: Provide a **multi-level checkpoint/restart environment** that delivers high performance and scalability for complex heterogeneous storage hierarchies without sacrificing ease of use and flexibility

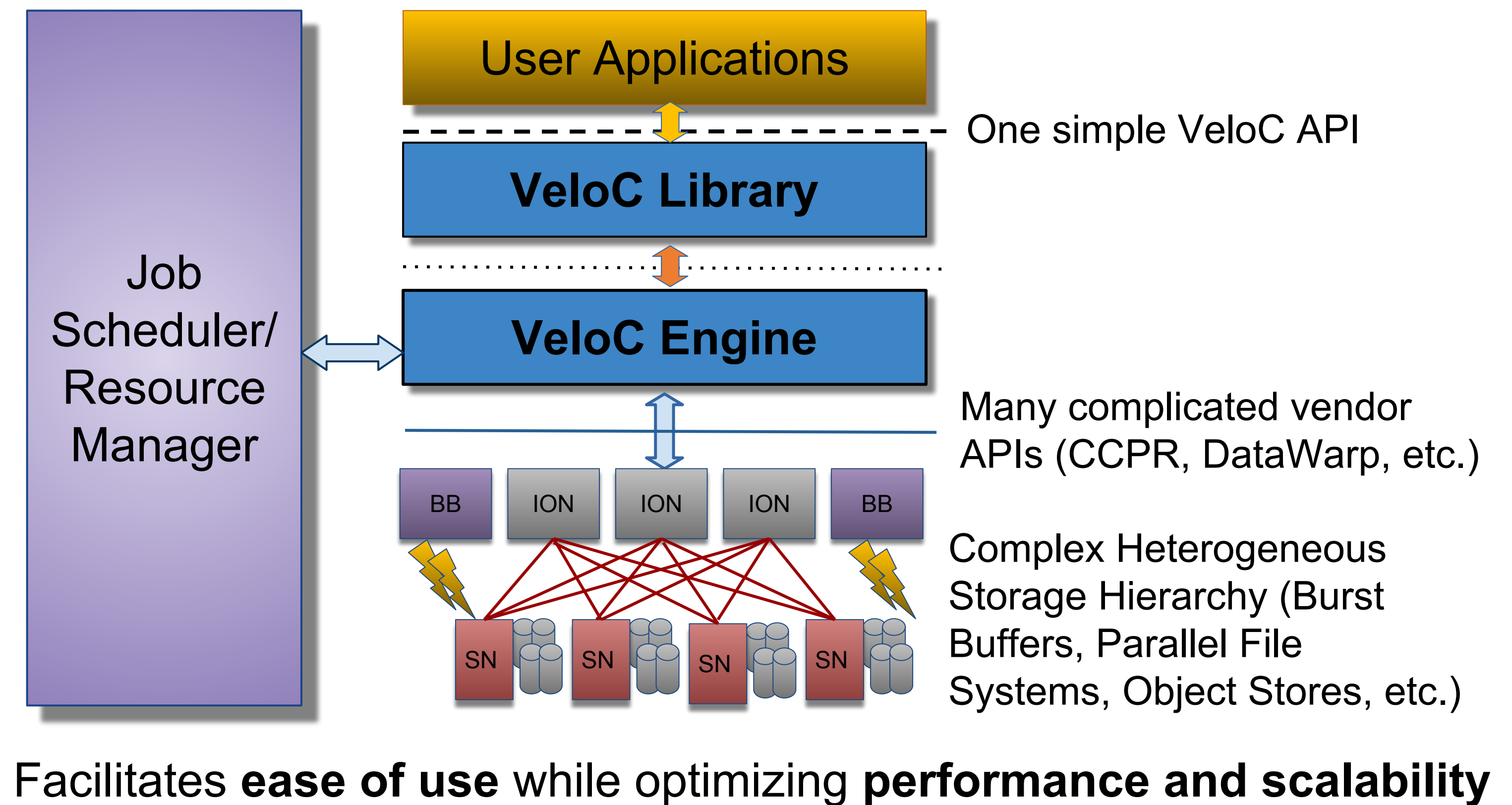


VeloC in a nutshell:

- Provides a simple API to checkpoint and restart based on data structures declared as important (protected) and captured automatically or files directly managed by the application
- Hides the complexity of interacting with the heterogeneous storage hierarchy (burst buffers, etc.) of current and future CORAL and ECP systems
- Based on a modular design that facilitates flexibility in choosing a resilience strategy and operation mode (synchronous or asynchronous), while being highly customizable with additional post-processing modules
- Many other use cases beyond fault tolerance: suspend-resume jobs to extend over multiple reservations, revisit previous states (e.g. adjoint computations), etc.
- Initial stress tests show high performance and scalability

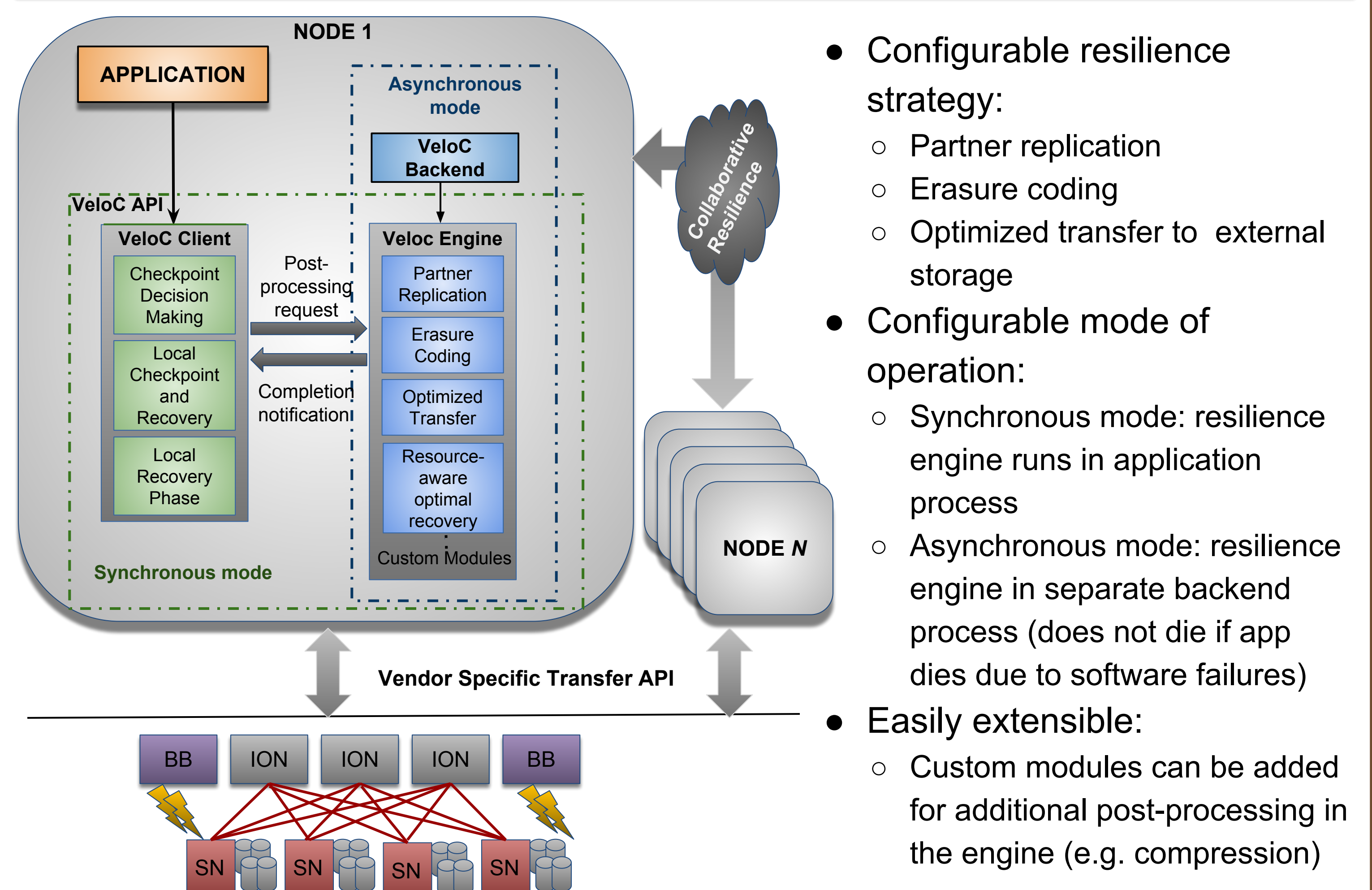
<http://veloc.readthedocs.io>

Hidden Complexity of Storage Interaction



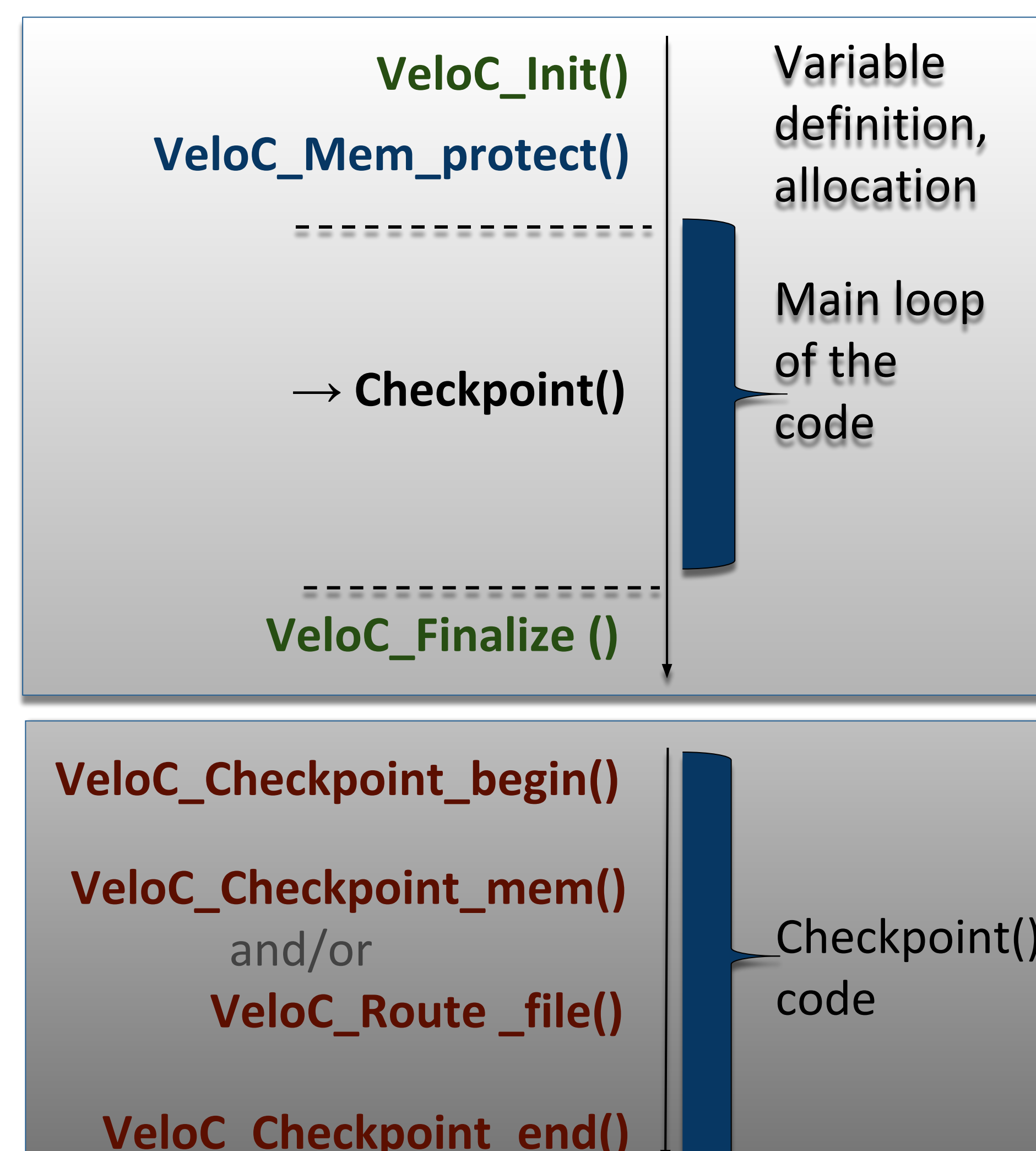
Facilitates **ease of use** while optimizing **performance and scalability**

Flexibility through Modular Design

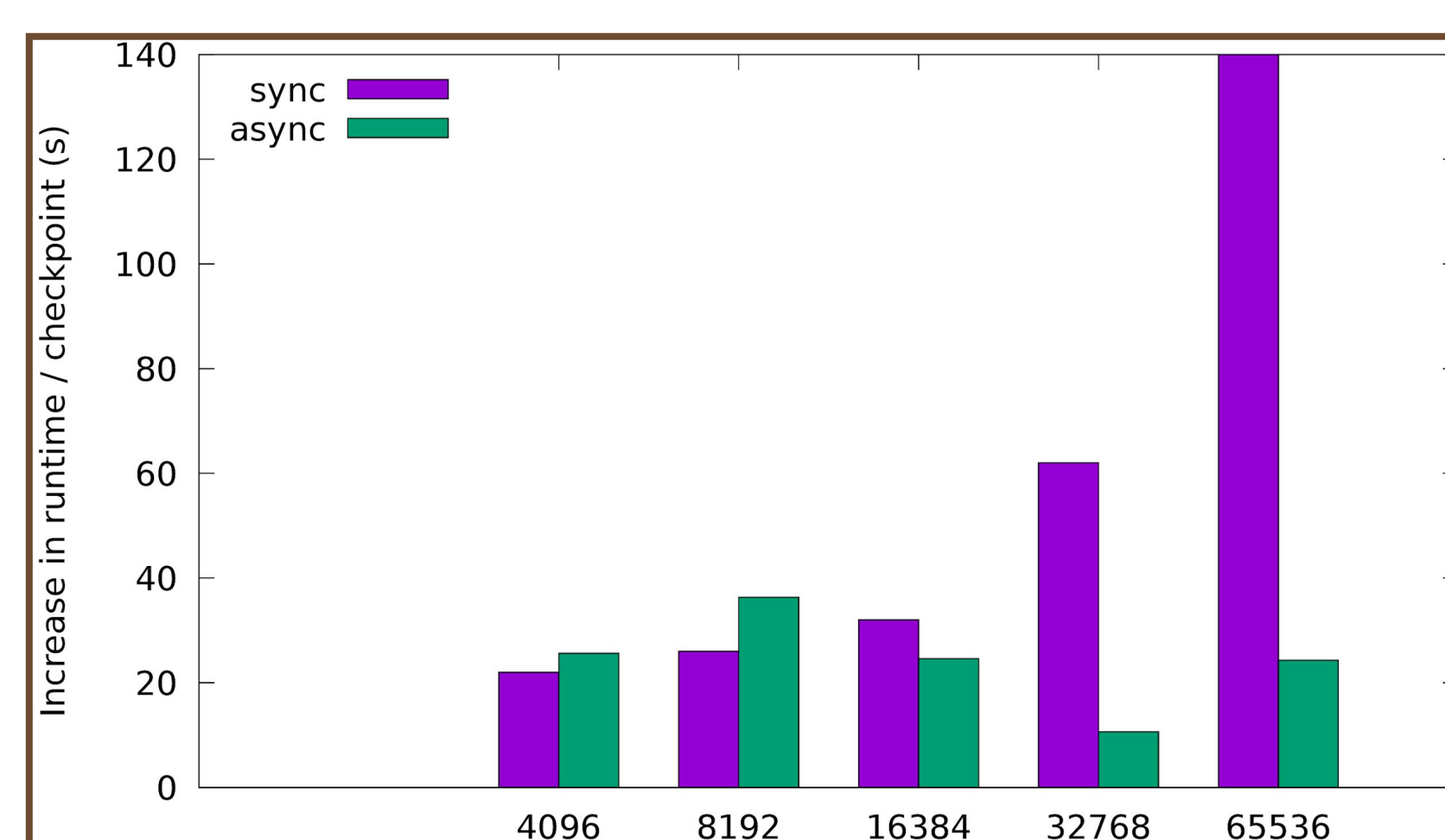


VeloC API

- Application-level checkpoint and restart API
- Minimizes code changes in applications
- Two possible modes:
 - File-oriented API: Manually write files and tell VeloC about them
 - Memory-oriented API: Declare and capture memory regions automatically
- Fire-and-forget: VeloC operates in the background
- Waiting for checkpoints is optional; a primitive is used to check progress



High Performance and Scalability



- Initial stress tests show promising results
- Application: Heat Distribution
- Platform: ANL Theta (KNL, Local SSD, Lustre)

- Scenario: 512 PEs/node, each checkpoints 256 MB (64 GB total/node)
- **Sync**: blocking writes to Lustre
- **Async**: writes to SSD followed by async flush to Lustre
- Metric: Increase in runtime due to checkpointing
- Conclusions:
 - Sync flush to PFS shows I/O bottlenecks even at small scale
 - ASync mode hides PFS latency: (reduced overhead up to 80%, improved scalability)