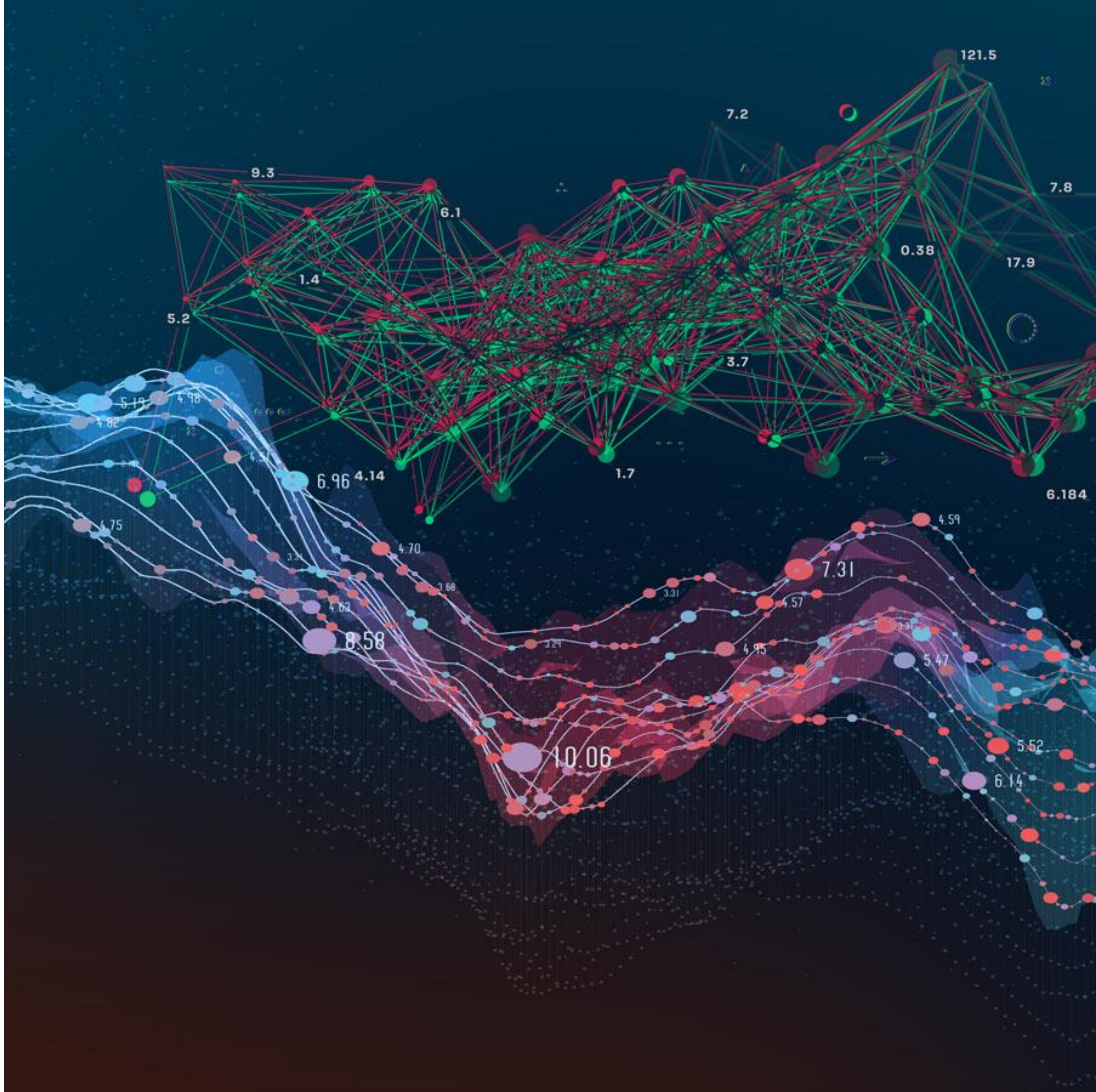
A Scalable, Thread-safe Programming Environment for Streaming Edge Analytics

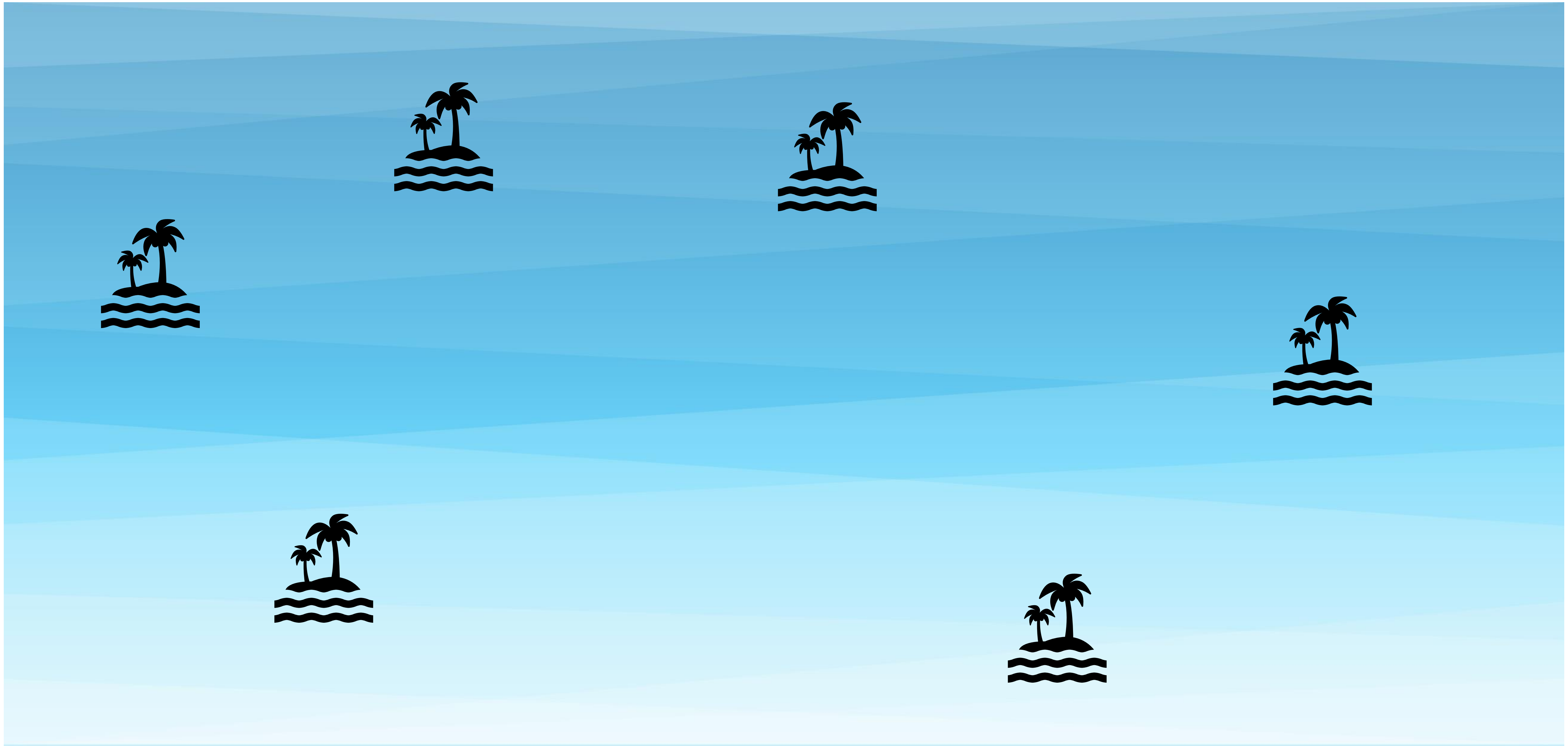
Vito Giovanni Castellana



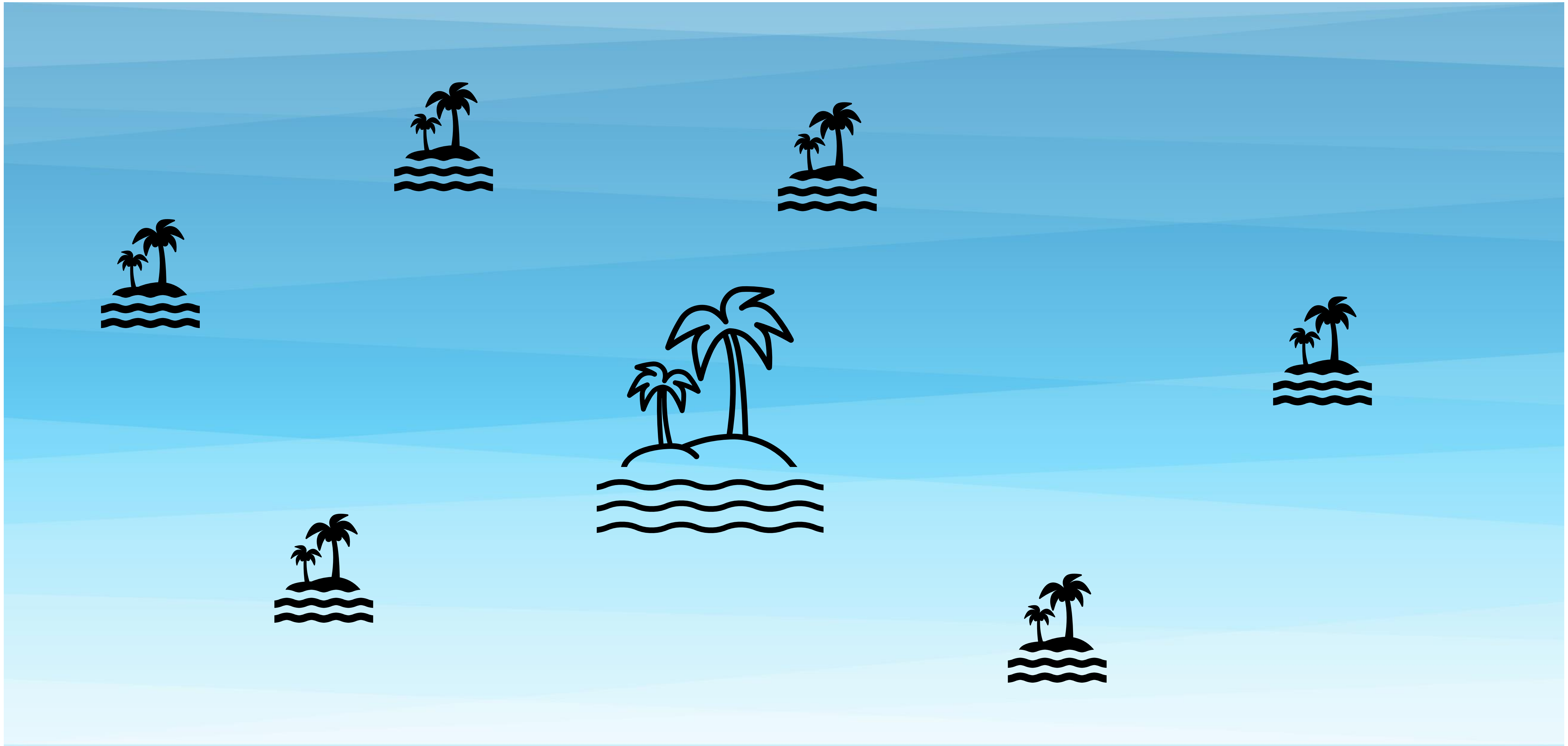
PNNL is operated by Battelle for the U.S. Department of Energy

A Sea of Data

A Lot of Little Islands



A Big Island



A LOT of Messages in Bottles

A Sea of Complexity and Heterogeneity

- Endpoints can scale in numbers from tens to millions (and beyond)

- Each system has different endpoints characteristics
  - They can perform local computation
    - ✓ Image recognition, classification, etc
    - ✓ Data preprocessing, filtering
  - They may have limited memory and compute
    - ✓ Data is collected and sent to the big island

- Computing at the edge is cool and increasingly ubiquitous, but…

We Still Need the Big Island (or an Archipelago)

- Several analytics applications work on data gathered from multiple or all endpoints

- Compute can be offloaded to a server and/or distributed across endpoints

- The server can be distributed itself

- Server features
  - Capacity – high data volumes
  - Scalability – system size (e.g. number of endpoints)
  - Performance – latency and throughput, at scale

Flexibility: The Hidden Feature

- Distributed Analytics Systems are complex
- Programming and using them is as complex

We need flexible software ecosystems to facilitate both the development and use of analytics systems

- … while satisfying the performance/scalability constraints

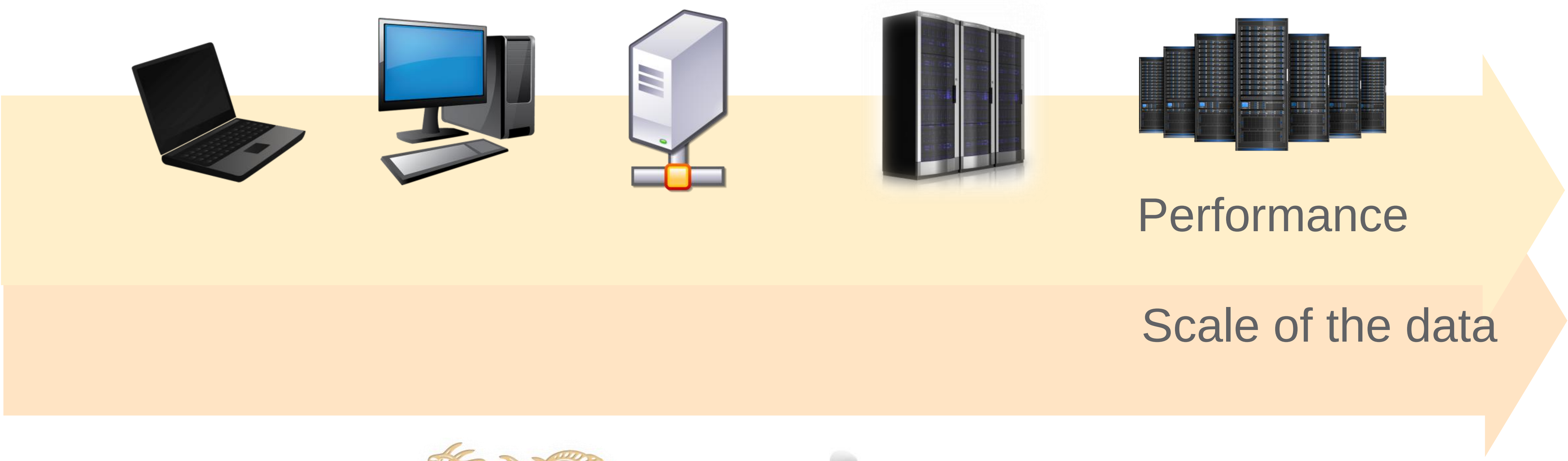
Flexibility: Not Much Hidden After All

- **Custom** Software solutions
  - Often **tailored** for **specific** hardware
  - Good performance, but…
    - Applications AND data (including models and abstractions) change/evolve
    - Most or the full software infrastructure may need to be re-written

- **Custom** Hardware/software solutions
  - **Custom** applications on **custom** hardware
    - Best performance, but…
  - Very high development effort, very high costs, very low portability

- Flexibility also connects to productivity, and cascades to costs/maintainability

Performance, Portability, and Productivity



Pacific Northwest
NATIONAL LABORATORY

Our Solution

SHAD

SCALABLE
HIGH-PERFORMANCE
ALGORITHMS &
DATA-STRUCTURES

What is SHAD?

- A fish  And a *portable* one too!

- The C++ library of Scalable Algorithms and Data-Structures
  - General Purpose Building Blocks (something like oneTBB, but on steroids)
  - High-Level, “custom” methods and utilities
    - ✓ New features are and will be added based on user requirements

- A playground for research in
  - Parallel Programming Models
  - Runtime systems and their application
  - New programming abstractions
    - ✓ focus on distributed, possibly heterogeneous systems
    - ✓ Goal: influence the community and possibly the standards

Features and Design Goals

- Flexibility
  - Rich set of general purpose data-structures
    - Can be used to implement a variety of applications in different domains
  - Data structures support efficiently both
    - Read only operations
      - Ingest & process applications
    - Frequent updates
      - **Streaming**

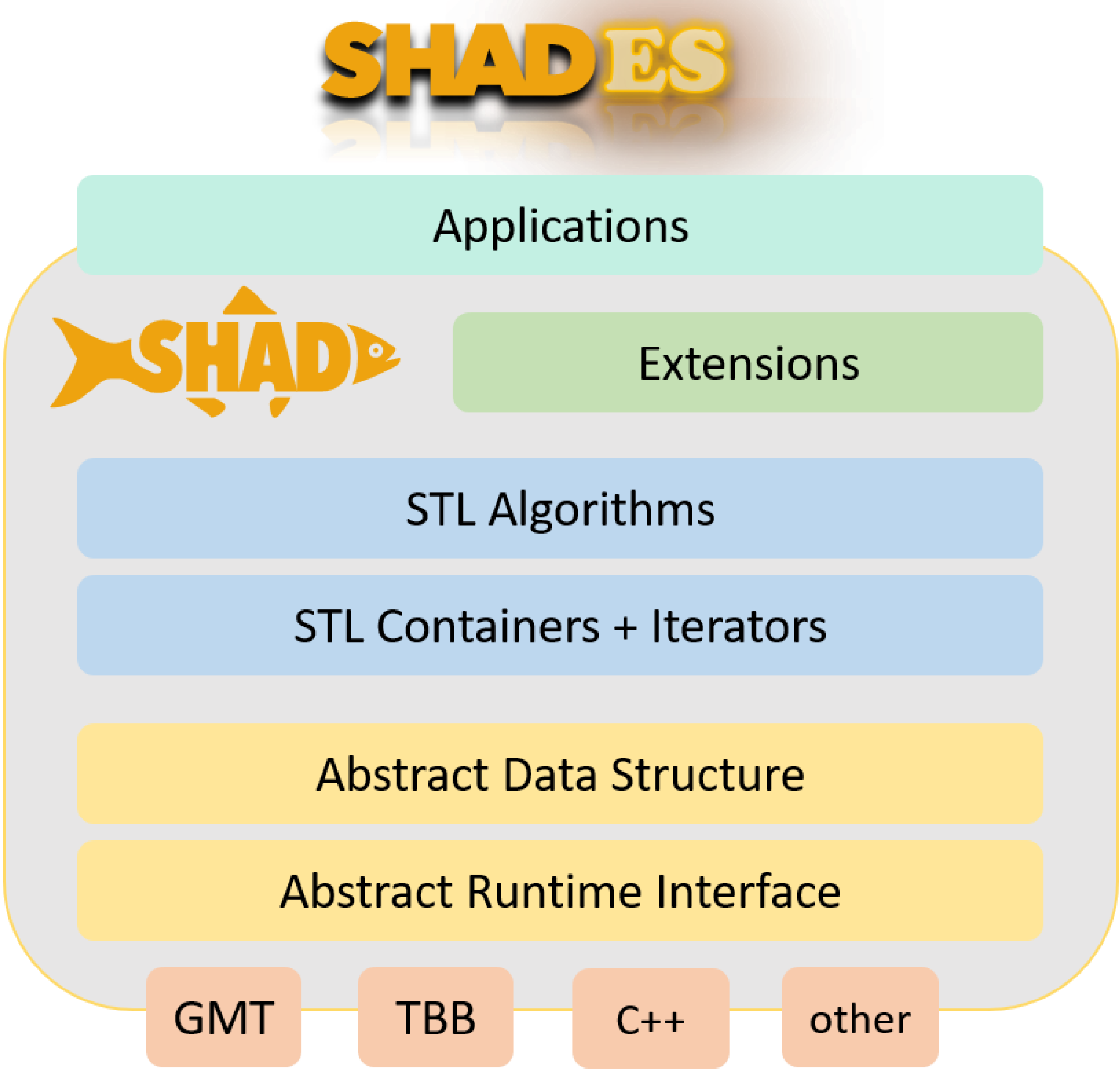
- Scalability and performance
  - Data structures can store, update and process TB scale data
  - Distributed on several nodes of a cluster, parallel access and update

Features and Design Goals

- Productivity
  - User-friendly STL-inspired interfaces -> improved user productivity
    - Easier porting of existing application
  - Most low level details (architecture, system configuration) are hidden

- Portability
  - Abstraction of underlying hardware and runtime system
    - Facilitates supporting multiple platforms/environments
  - Limited set of software dependencies
    - E.g. compiler support for C++ 17

High-level Design Overview



Abstract Runtime Interface: Main Concepts

Machine Abstraction

- **Locality**
  - Entity in which memory is directly accessible
  - Examples: node in a cluster, core, NUMA domain
  - Same abstraction can be extended to edge devices
- **Task**
  - Basic unit of computation
  - Can be executed on any locality
  - Can be asynchronous
- “***Handles***”
  - Identifiers for spawning activities
    - Multiple tasks may be associated to the same handle
  - Used to check for task completion

Runtime Interface API (extract)

- **[async]ExecuteAt**
  - [asynchronously] execute a function on a given locality
- **[async]ExecuteAtWithRet**
  - [asynchronously] execute a function on a given locality and returns data back
- **[async]ExecuteOnAll**
  - [asynchronously] execute a function on all localities
- **[async]ForEachAt**
  - [asynchronously] execute a parallel loop on a given locality
- **[async]ForEachOnAll**
  - [asynchronously] execute a parallel loop on all localities
- **[async]dma**
  - [asynchronously] copy data to/from a [remote] memory location
- **waitForCompletion**
  - wait for the completion of asynchronous tasks

Runtime Interface Mappings

- Plain C++
  - For fast prototyping and playing around
- PNNL's Global Memory and Threading (GMT) library
  - Targets distributed systems
  - Available at **https://github.com/pnnl/gmt**
- Intel' Threading Building Blocks (oneTBB)
  - Targets shared memory systems
    - … these may include your laptop ☺
- First version of an HPX backend is also available
  - **https://github.com/STEllAR-GROUP/hpx**

Programming Model

- **Shared Memory Programming Model**
  - Also on distributed setting
  - Non-SPMD
- Standard C++ STL and “STL-like” APIs
  - Data structure interfaces, iterators, algorithms, execution policies, etc

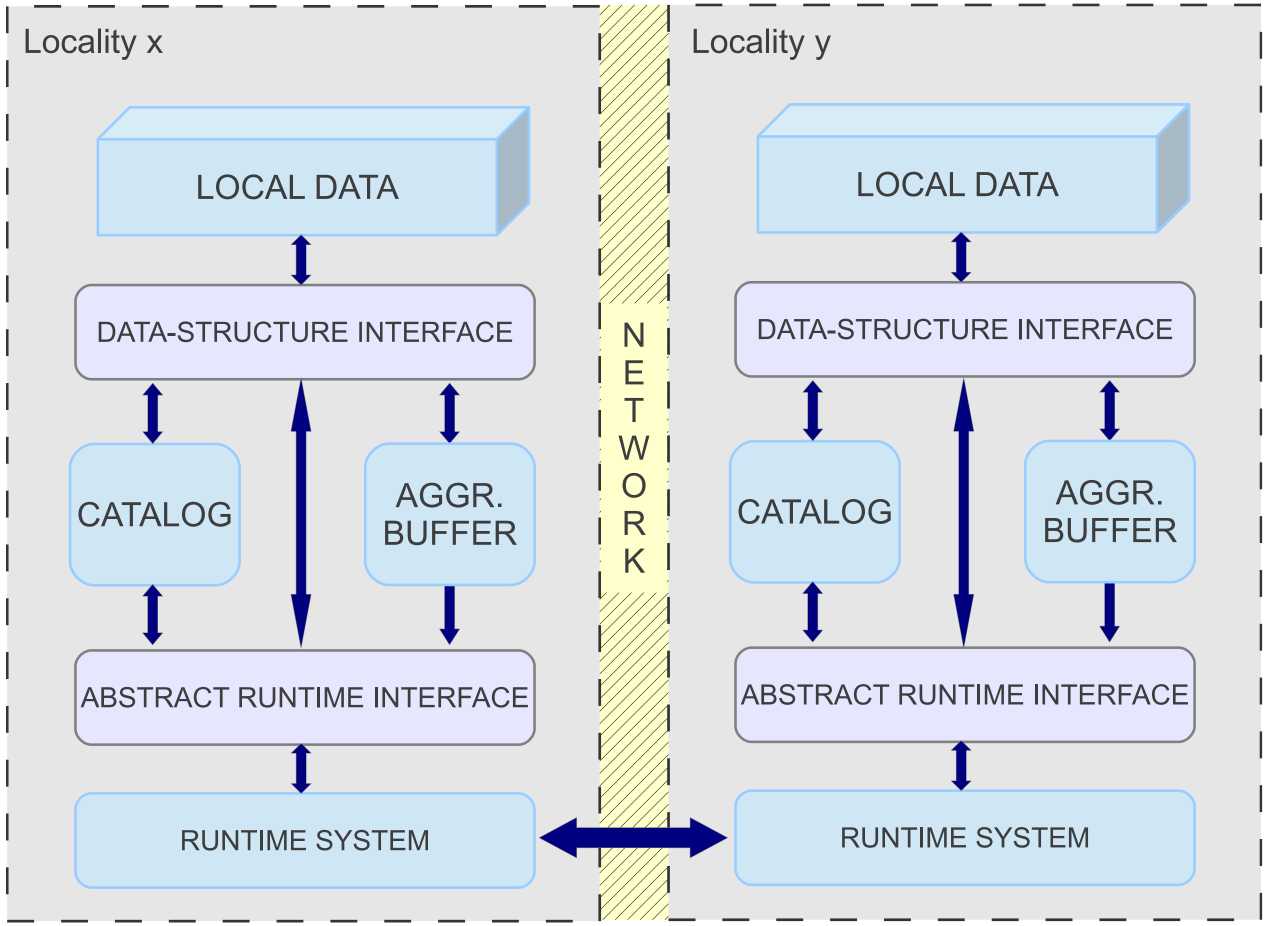
```
price_t max_price(shad::array<option_t, n> &a) {
 shad::array<price_t, n_options> p;
 shad::transform(shad::execution::par, a.begin(), a.end(),
                      p.begin(), blck_schls);
 return *shad::max_element(shad::execution::par, p.begin(), p.end());
}
```

SHAD-powered Distributed STL

General Purpose Data Structures and Algorithms

- Include: array, vector, unordered set, map and multimap
- They “look like” STL, but they
  - Can be distributed on several localities
    - High capacity (TB+ scale data)
  - **Are thread safe**
  - Can be modified and accessed in parallel
    - High performance
  - Automatically manage synchronization and data-movements

Abstract Data Structure



SHAD Arrays

- STL compliant with iterators
- Distributed evenly across locales
  - Data distribution can be changed
- Single and multiple element get and put operations
- Bulk puts/gets with DMA support
- `shad::array<type>`

- SHAD also includes two variants of vector
  - Legacy implementation
    - Round robin dynamic memory allocation, support for push_back
  - New implementation
    - Analogous to Array, but allows resizing

Unordered Maps and Sets

- STL compliant with iterators
- Keys hashed to locales
  - Local data is stored in an unordered map/set, with the same API
  - Local map/set is a vector of linked lists (and it is **thread safe** too!)
  - Nodes in the lists are dynamically allocated
- Multiple readers, single writer per bucket
  - Inserts only block access to the updated and following entries in the list
    - Previous entries can be accessed
  - Updates don't block any access
- Insert, delete, update, and apply are **atomic**
- Deletes swap the deleted entry with a valid one
- `shad::unordered_hmap<ktype, vtype, key_compare, insert_policy>`

Identical keys in different structures mapped to the same locale

Needed for streaming data

Multiple field keys

Way cool

Pacific Northwest NATIONAL LABORATORY

Multimaps and Atomics

Multimaps

- STL compliant with iterators
- Same structure as unordered_map
- Key differences
  - Each key may have multiple values, stored in a std::vector
  - Writes lock the bucket
- `shad::unordered_multimap<ktype, vtype, key_compare, insert_policy>`

Atomics

- Atomic objects are globally accessible, but the data is stored in one locale
- Supported atomic operations defined on std::atomic, plus
- Customizable operations (via user defined operators)
- `shad::atomic<type>`

Inserters

- Inserters are cool
- Inserters are functors which define how the insert operation behaves
  - Default inserters simply update the entry value
  - They can be complex classes, with attributes and their own additional methods
  - They can even NOT insert!
- Regardless the operation(s) they actually perform, inserters have the same **atomic properties** of regular writes
- Maps store a main inserter at creation, of the specialized type (defaulted to Overwriter)
- Insert methods can use any different custom inserter

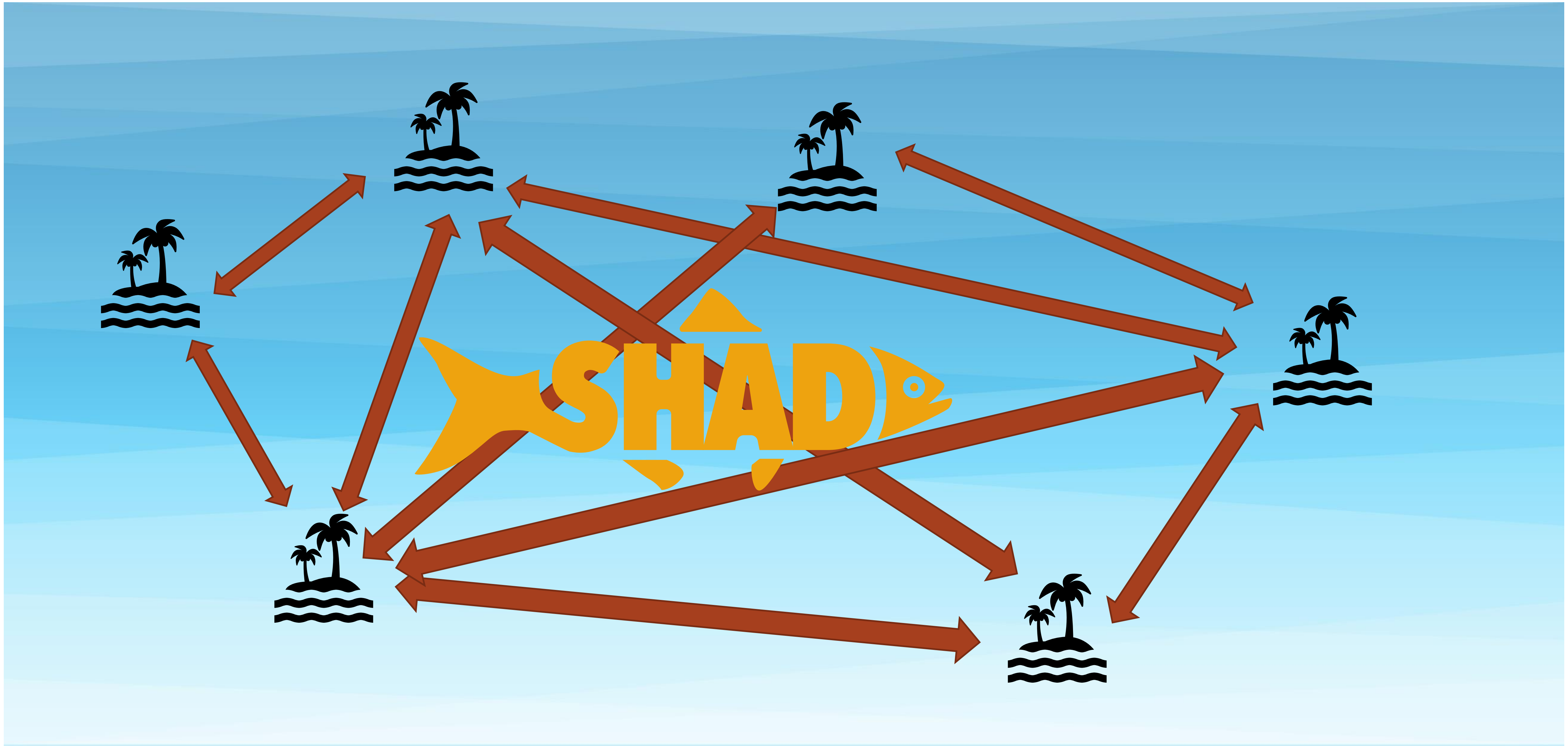
Reactive Analytics

- Inserters can be used for a number of different applications

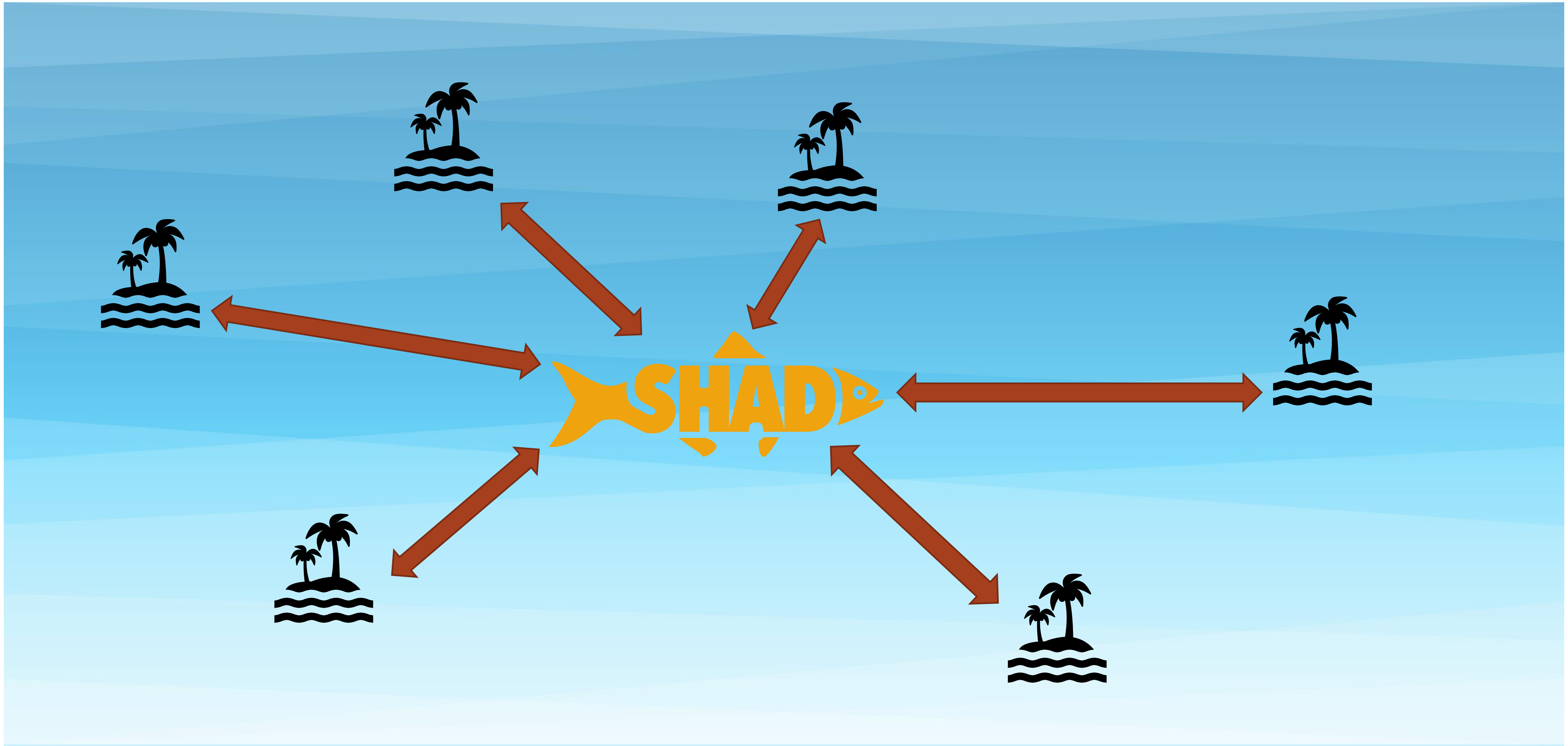
- Examples
  - Cascaded inserts and data filtering
    - Can be used for access control, multi-level security
  - Compute statistics
    - E.g. count same-key insertions, aggregate value properties, etc
  - Trigger computation
    - Distributed ID dictionary creation
    - Alerting systems
    - Action Graphs

SHAD-powered Systems for Streaming Edge Analytics

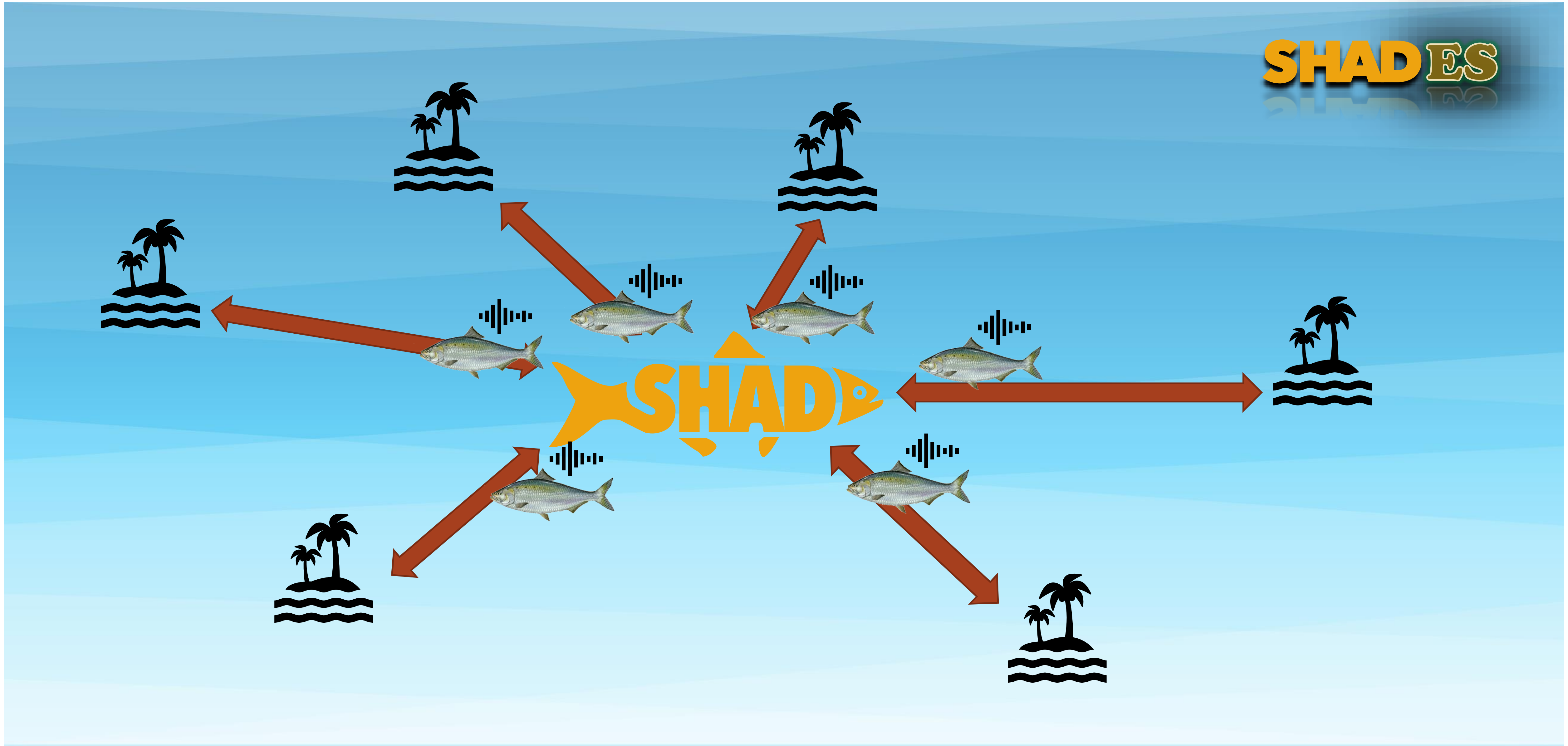
OPT1: Data/Computation is fully distributed



OPT2: Data/Computation is Offloaded



OPT3: Client-Server Model

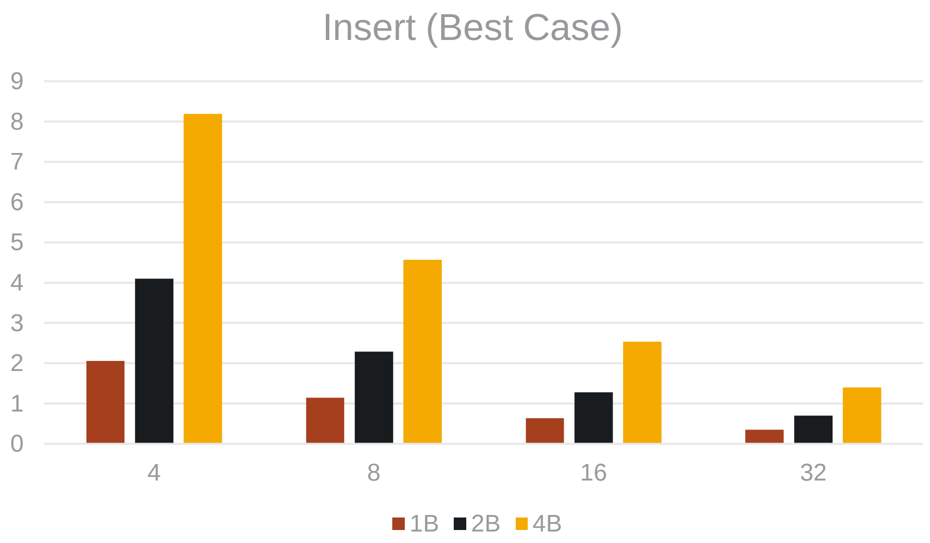


Throughput Analysis of SHAD Data-Structures

Experimental Setup

- GMT Mapping
- SHAD/GMT compiled with GCC 8 and OpenMPI
  - We are using tcMalloc
- Platform: commodity cluster
  - Intel Xeon dual socket processors @2.80GHz
    - 10 cores per socket
    - Used up to 320 cores
- Machine abstraction: 1 Locality per socket
  - Up to 32 localities
- Data elements are of type uint

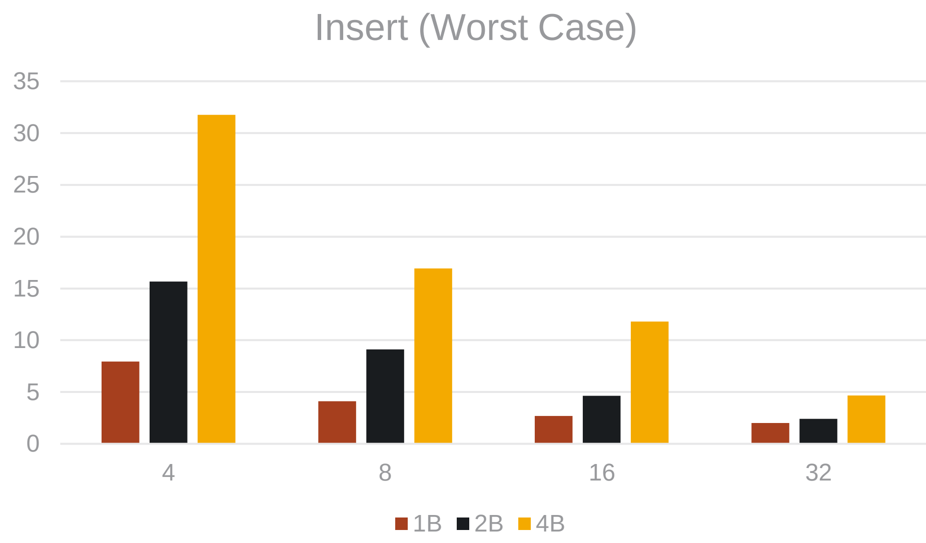
Array

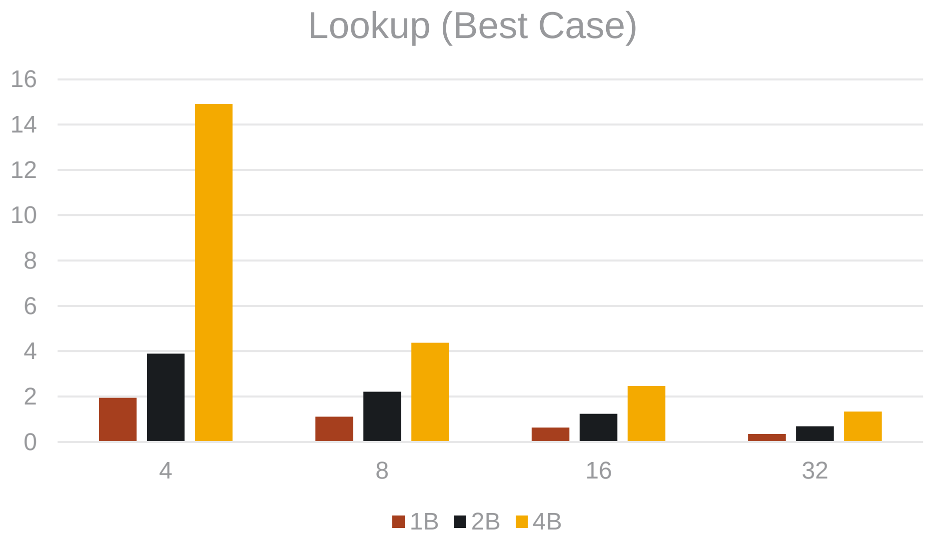


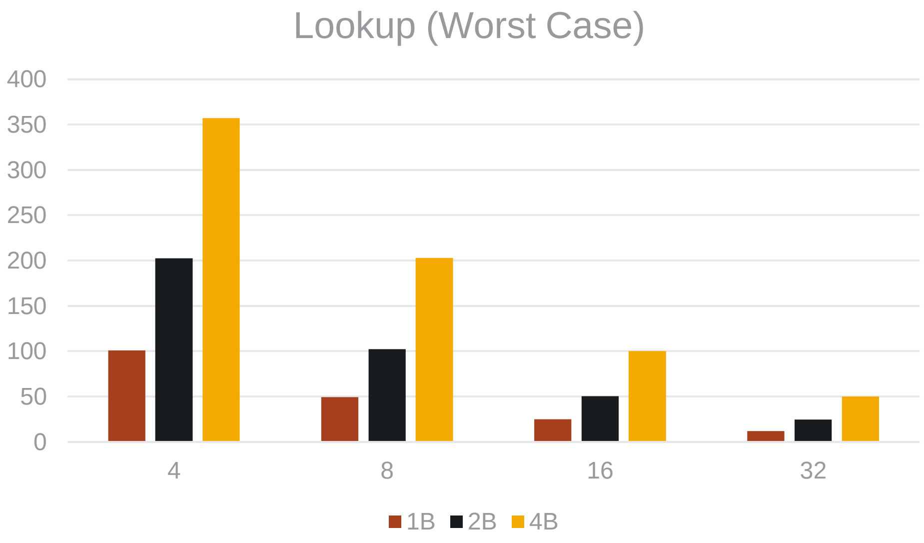
Linear throughput up to 1T

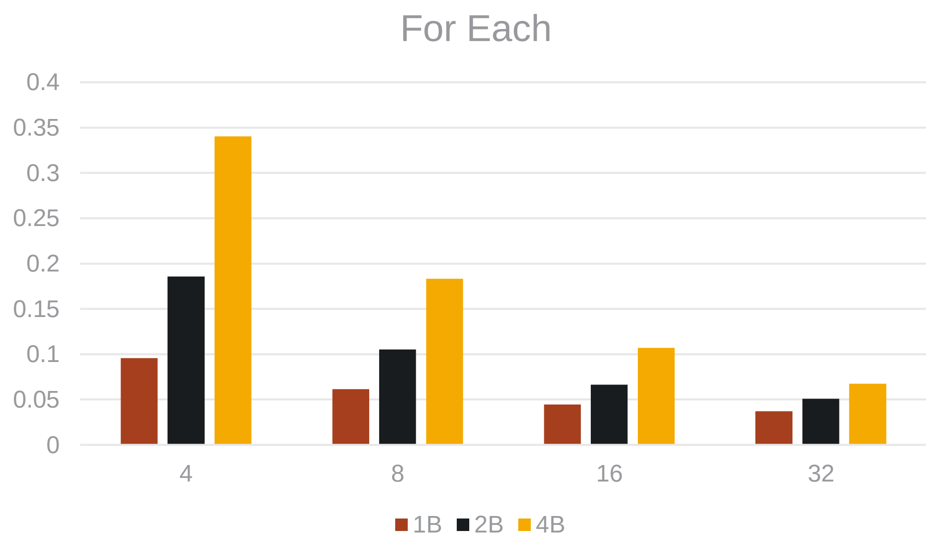
- Ins/LkUp: ~**3B** ops/sec
- ForEach: ~**80B** ops/sec

Note: arrays support DMA transfers, not used here

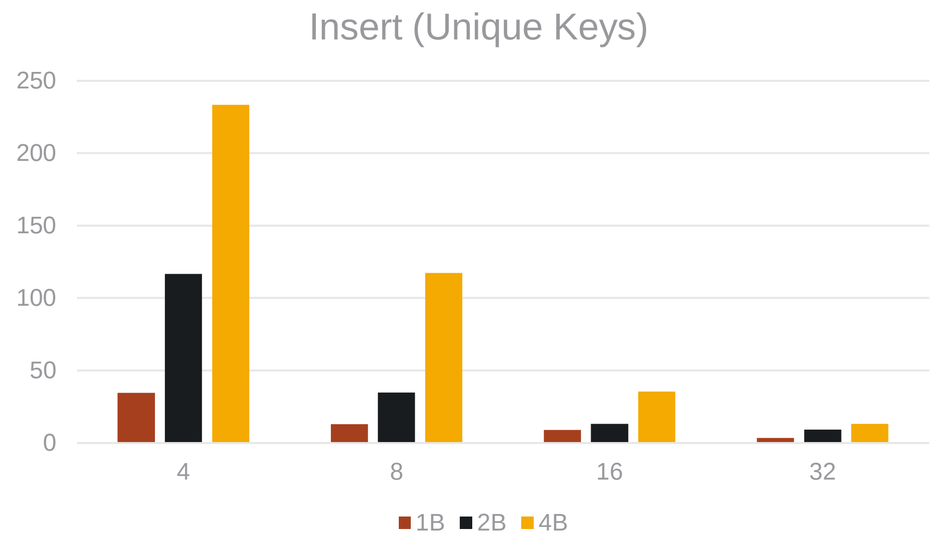






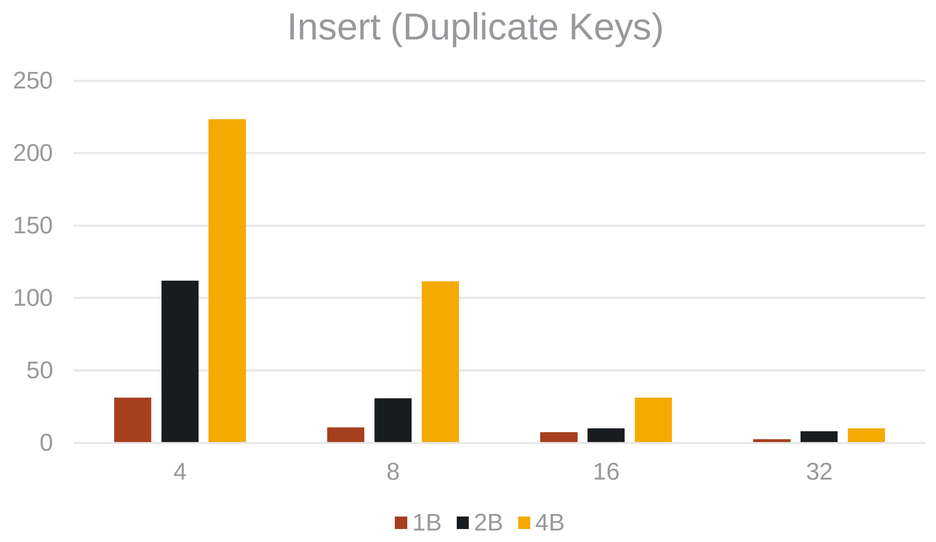


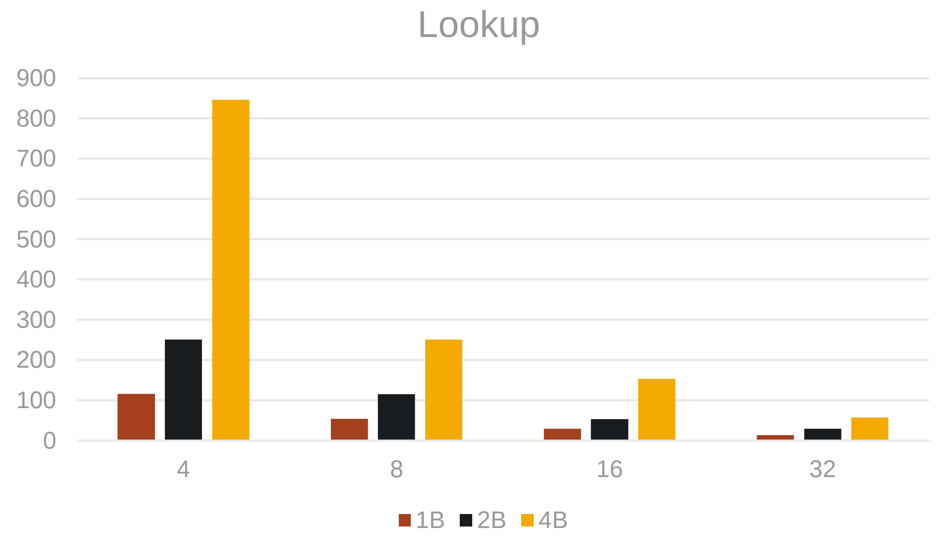
Unordered Map

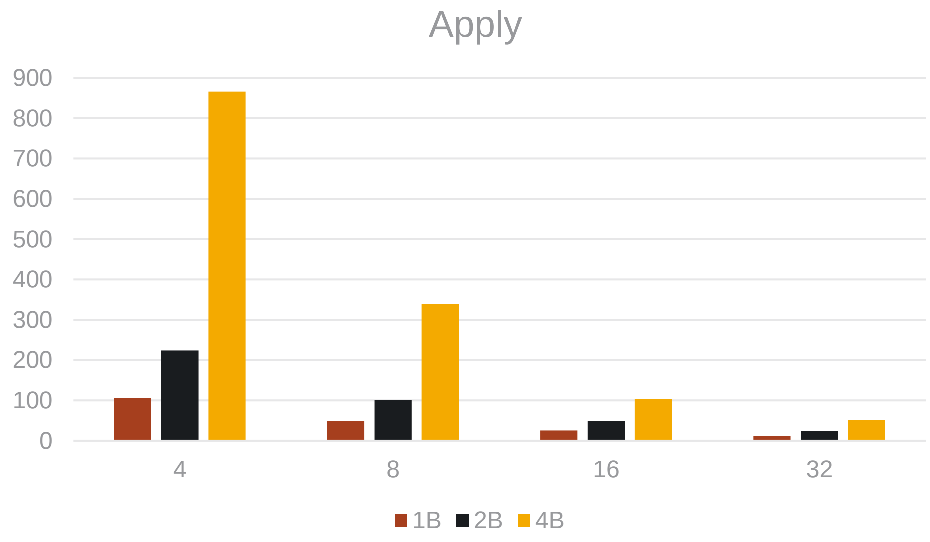


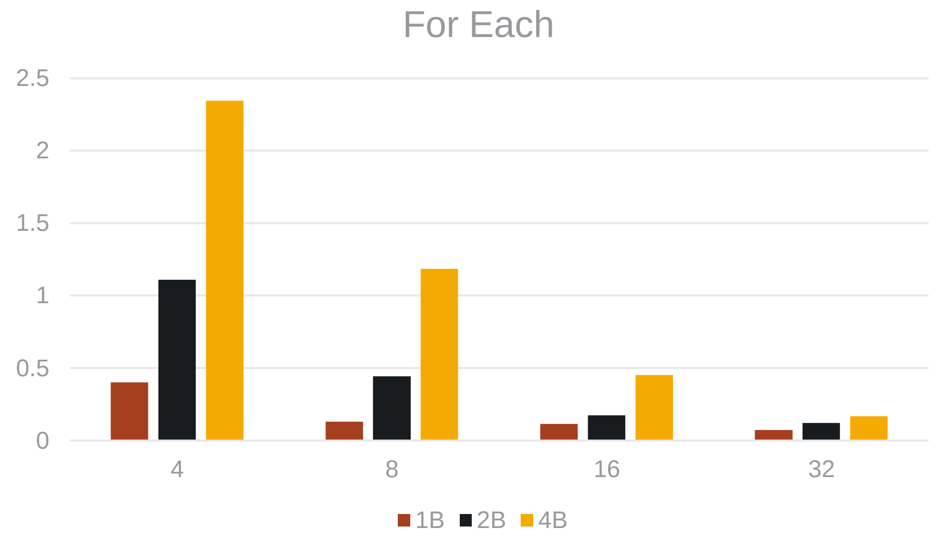
Peak @ 4B

- Insert: ~**307M** ops/sec
- LkUp/Apply: ~**75M** ops/sec
- ForEach: ~**25B** ops/sec

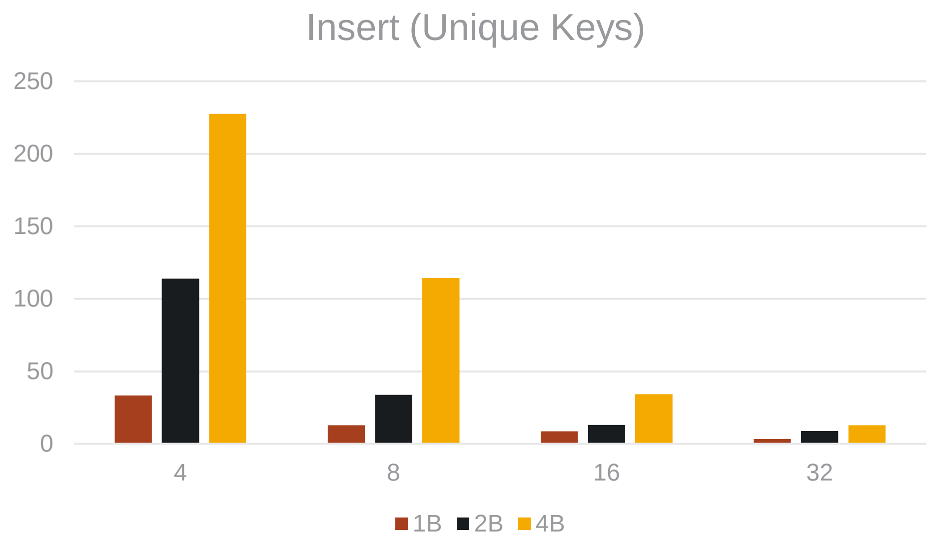


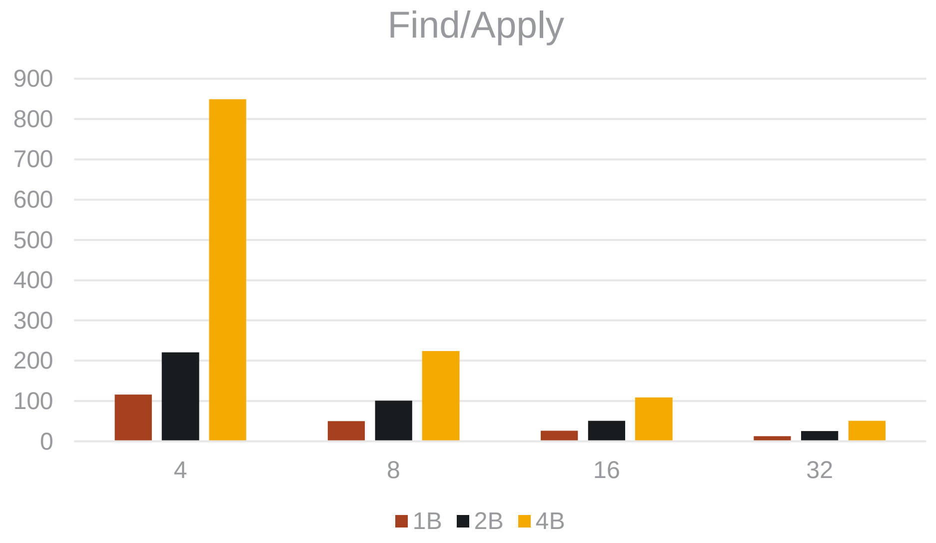






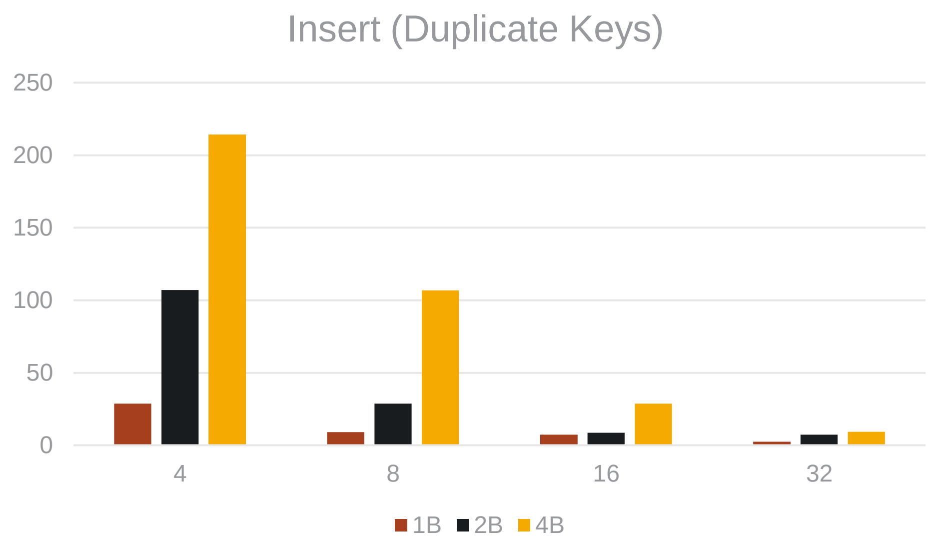
Unordered Set

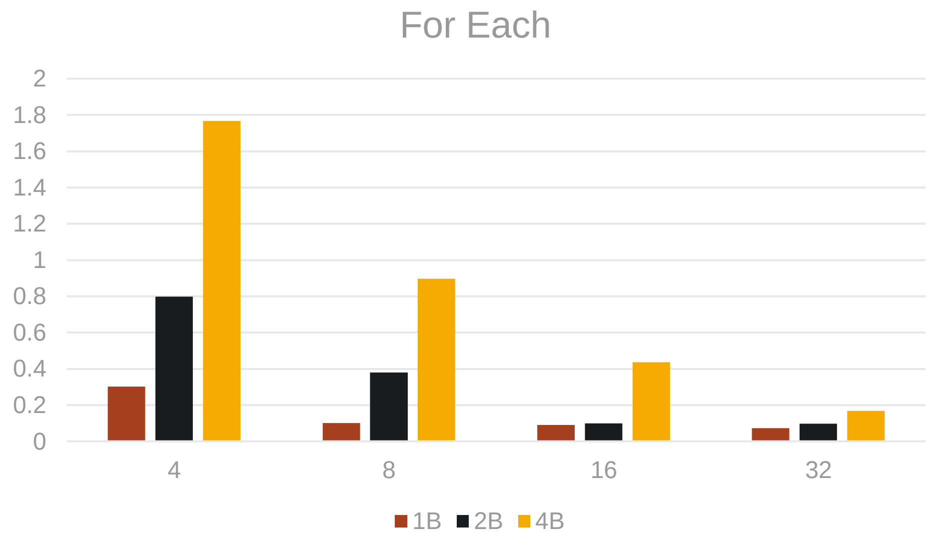




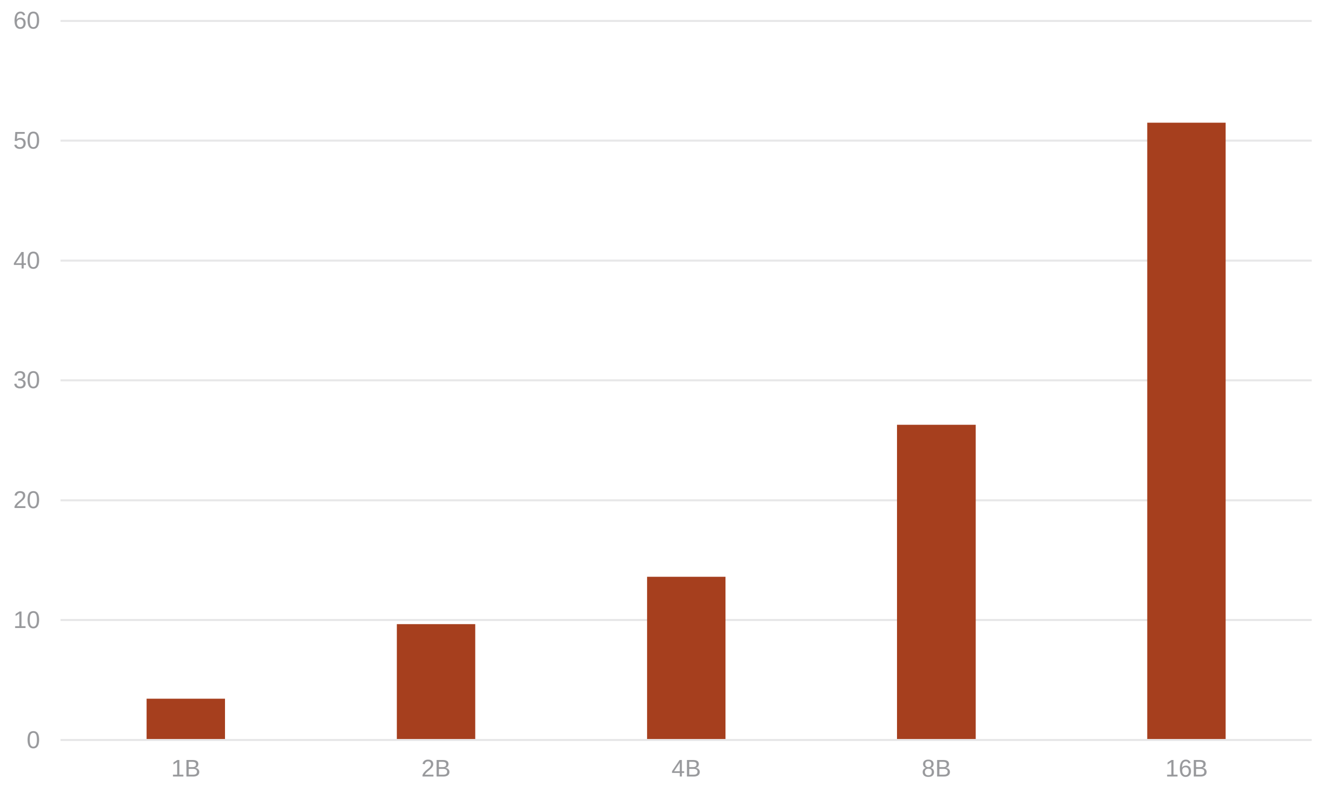
Peak @ 4B

- Insert: **~315M** ops/sec
- Find/Apply: **~80M** ops/sec
- ForEach: **~25B** ops/sec



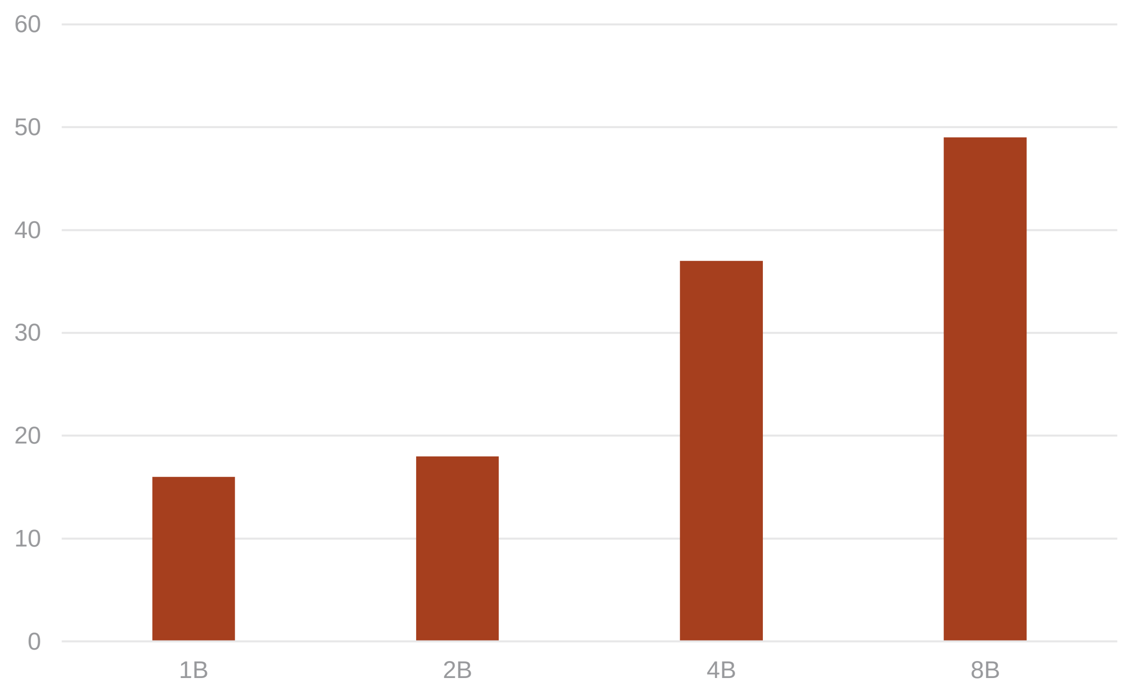


Scaling the Number of Endpoints



- Endpoints scaled from 2 to 64 **per locality**
  - max: **2048** endpoints
- **~Same performance regardless the number of endpoints**
- **Peak @ 16B**
  - Insert: ~**310M** ops/sec

“Cascaded” Insertions



- Insertions in a map, triggering an insertion in a set (unique keys)
- Each insertion in the set is done atomically wrt to the insertion in the map
- **Peak @ 8B, 32 locales**
  - Cascaded Insert: ~**163M** ops/sec

Encore: Ongoing Research

Pacific Northwest NATIONAL LABORATORY

Extend the Concept of Locality

- Current limitation: data/computation is distributed over **homogenous** sets of localities
  - ✓Example: CPUs VS GPUs (experimental)
    - Black Scholes on CPUs
      - ~706.7 millions options/sec @16 locales
      - ~**82.5x speedup** vs plain STL
    - Black Scholes on GPUs (NV Tesla)
      - ~5 billions options/sec @4 locales
      - ~**585x speedup** vs plain STL (CPUs)

- GOAL: Fully exploit heterogeneity, while maintaining high-level, portable interfaces
  - FPGAs, GPUs, custom accelerators including Edge Devices

Pacific Northwest NATIONAL LABORATORY

Build Complex Analytics Workflows

- We are using SHAD as the software infrastructure to define and build complex analytics applications

- Mix of different computational and memory access patterns
    - ✓ Graph Analytics + Machine Learning

- Workflows have **streaming** variants

- More info @

https://www.iarpa.gov/research-programs/agile

Pacific Northwest
NATIONAL LABORATORY

Thanks!!

SHAD

https://github.com/pnnl/SHAD

Vito Giovanni Castellana
vitoGiovanni.castellana@pnnl.gov