Lucata Technical Overview

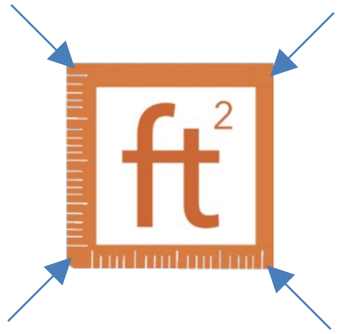
Marty Deneroff (mdeneroff@lucata.com)

WHAT DOES LUCATA DO?

Lucata's goal is to provide a computer that is effective when executing algorithms that work poorly on mainstream computers:
Those that require random access to very large, sparse datasets.

Lucata has developed a unique architecture / computing system called Pathfinder that supports massive parallelism, near-linear scaling, and fine grain concurrency that is well matched to the data types and sizes of today and the future. It efficiently performs analytics on large-scale datasets in real- time, enabling applications that are inefficient or intractable today.

Pathfinder delivers unparalleled CAPEX and OPEX economics …better performance per dollar, per watt, and per square foot.

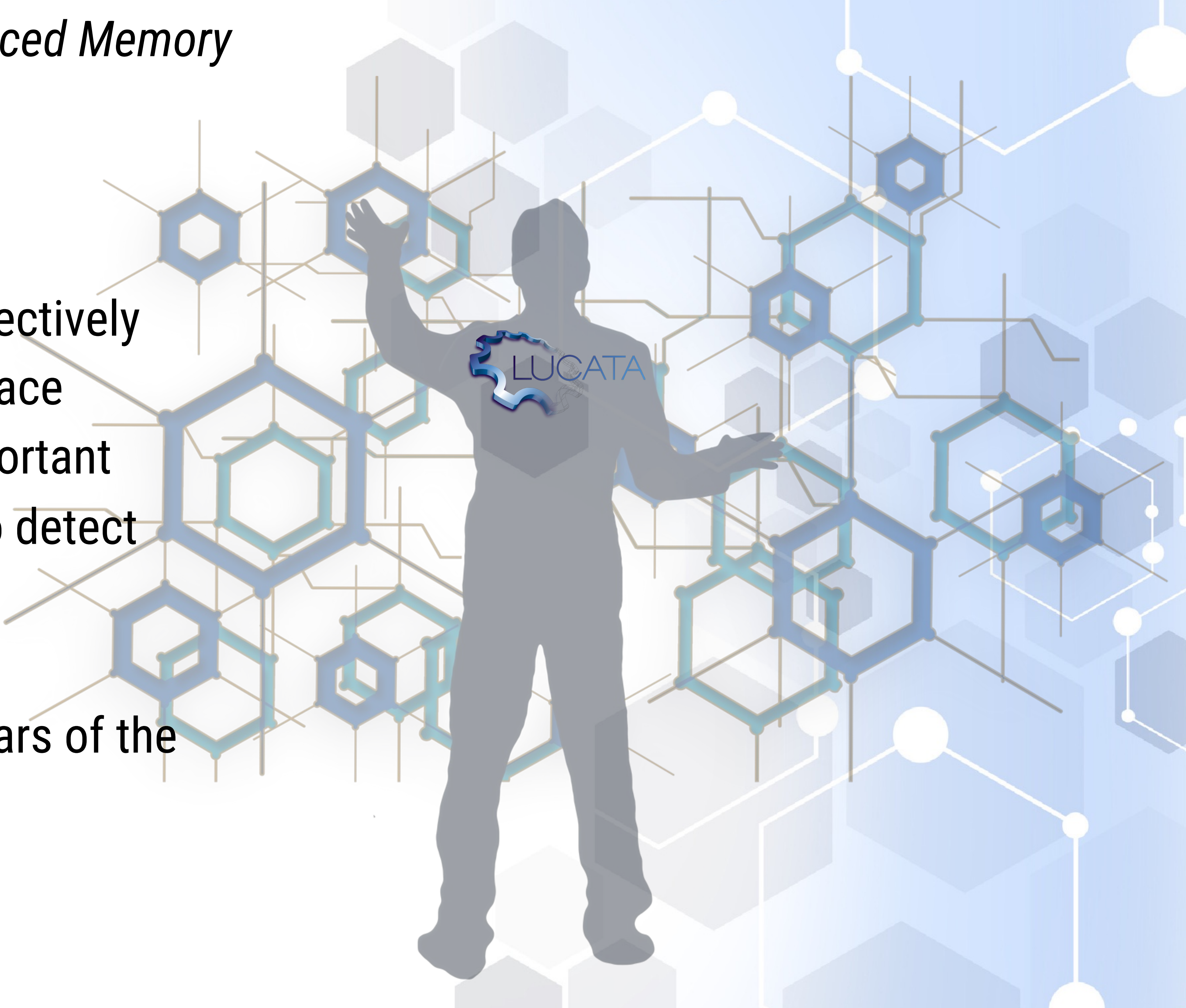


LUCATA HISTORY

Lucata (originally called **EMU Technology,** an acronym *for Enhanced Memory Utilization*) was founded in 2010
by Drs. Peter Kogge, Jay Brockman, and Ed Upchurch.

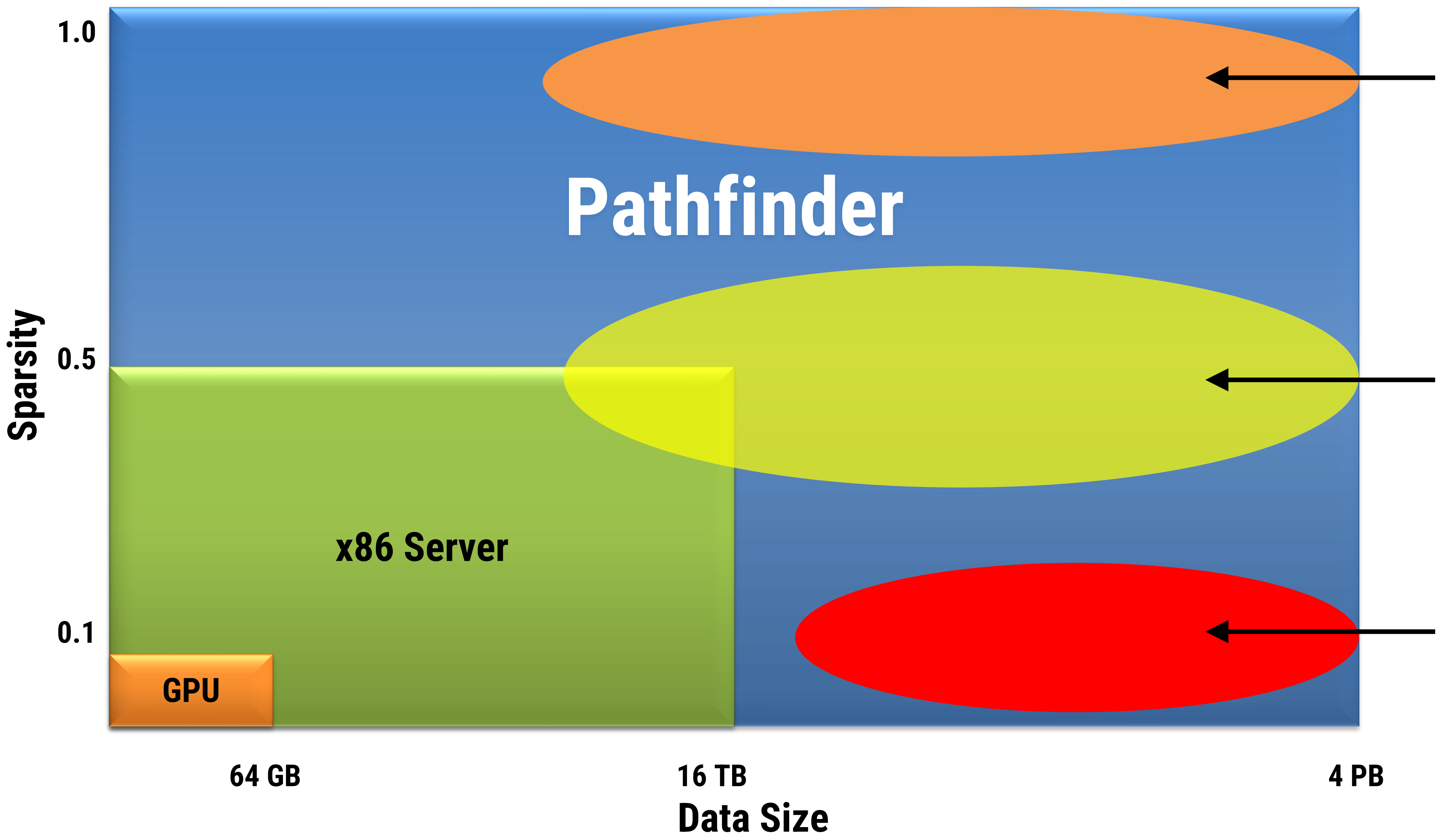
Emu's initial goal was to develop a computer that could work effectively with sparse data in enormous memory systems. The problem space centered on finding the **"Needle in a Haystack"** – the bits of important information embedded in gigantic collections of raw data, and to detect **Non-Obvious Relationships** among pieces of this data.

This work was largely funded by the NSA through the first ten years of the company's existence.



PRIMARY APPLICATION DOMAINS

Datasets that are hundreds of TB in size, high in sparsity and require concurrent queries without performance loss.



Sparsity = Number of Values / Number of Indices; Max. possible = 1.0

Large in-memory graph databases
Large knowledge graphs
Retrieval-Augmented Generation
Sparse models of large physical systems

Large in-memory Relational Databases

Very large (brain-scale) AI models

LUCATA PATHFINDER DATA CENTRIC PLATFORM

- **Migrating Thread** technology allows Processing In Memory as it moves computation to the memory system containing the data being processed and performs the processing at that memory.
- **More memory** per rack at a **lower** overall **cost;** handles multiple **petabytes** datasets.
- **Global Shared Memory** can span thousands of cabinets.
- Maintains **linear performance** as the dataset grows.
- Support for real-time **streaming ingest** at enormous rates from **hundreds** of **high-speed network interfaces**.
- Unprecedented **energy efficiency**, with a **compact** system that minimizes use of datacenter space.
- **Millions of concurrent threads** managed by hardware can be dedicated to work on a single complex task or a very large number of concurrent tasks.
- **Vastly simplified programming** using familiar programming languages such as C++ and Python, and the ubiquitous Linux operating system.

MEASURABLE CONCURRENCY AND SCALABILITY

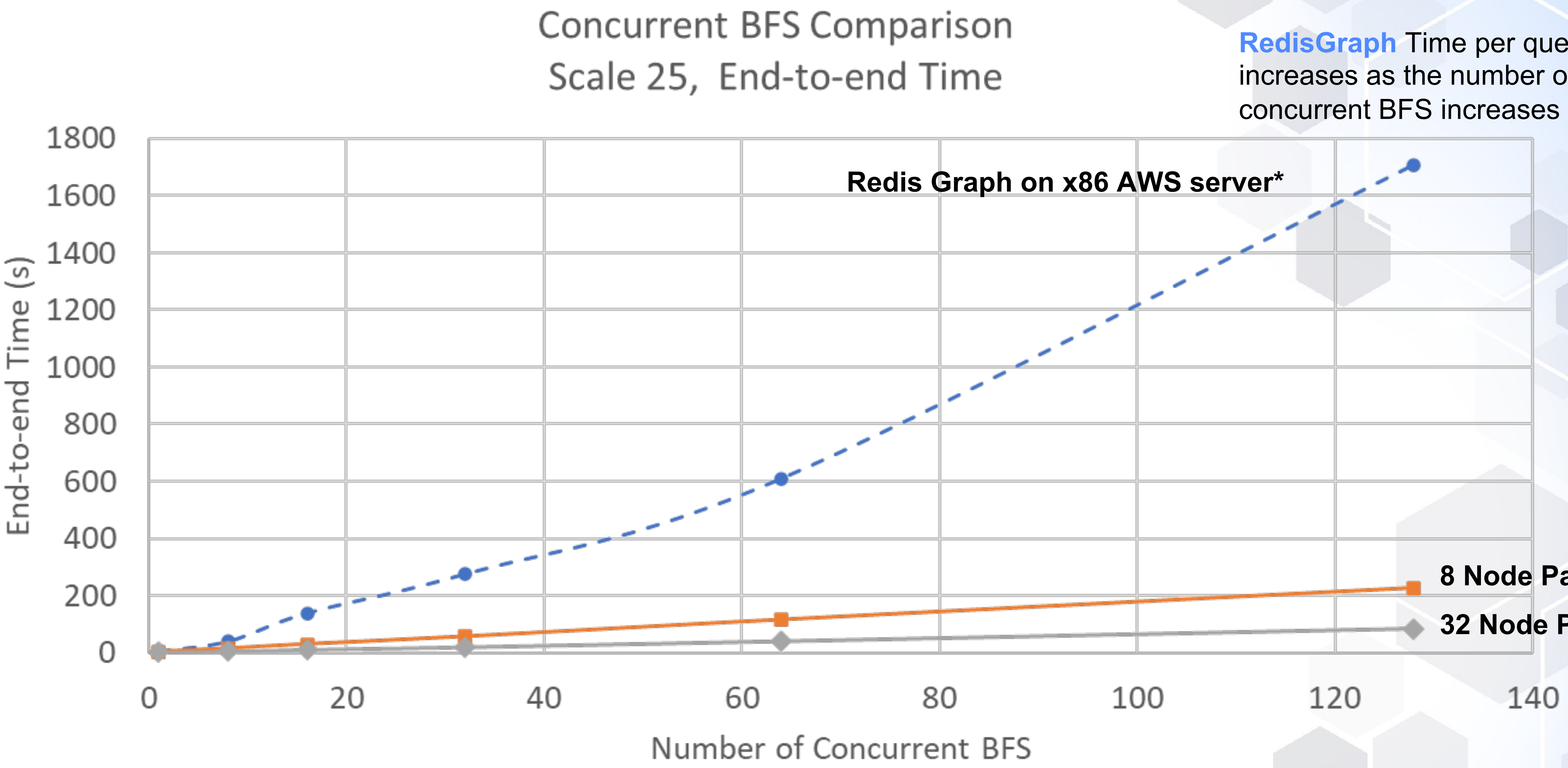
Common Graph benchmarks measure performance of a **single** query. But real-world applications need a high rate of **simultaneous** queries and updates on a **shared graph**.

Lucata Pathfinder supports **multiple concurrent queries**, running the same or disparate algorithms, without losing performance.

Breadth-First Search (BFS) is a key algorithm used to search a tree or graph data structure

Lower End-to-End Time is Better

ASIC Based Pathfinder will deliver a 10-25x performance improvement



RedisGraph Time per query increases as the number of concurrent BFS increases

Pathfinder: Time per query decreases from 1 to 8 queries, then stays almost constant for increasing concurrency.

*Large AWS Server - Intel(R) Xeon(R) CPU E7-8880 v3, 2.30GHz, 128 vCPUs (64 cores, 2 threads per core), 4TB memory

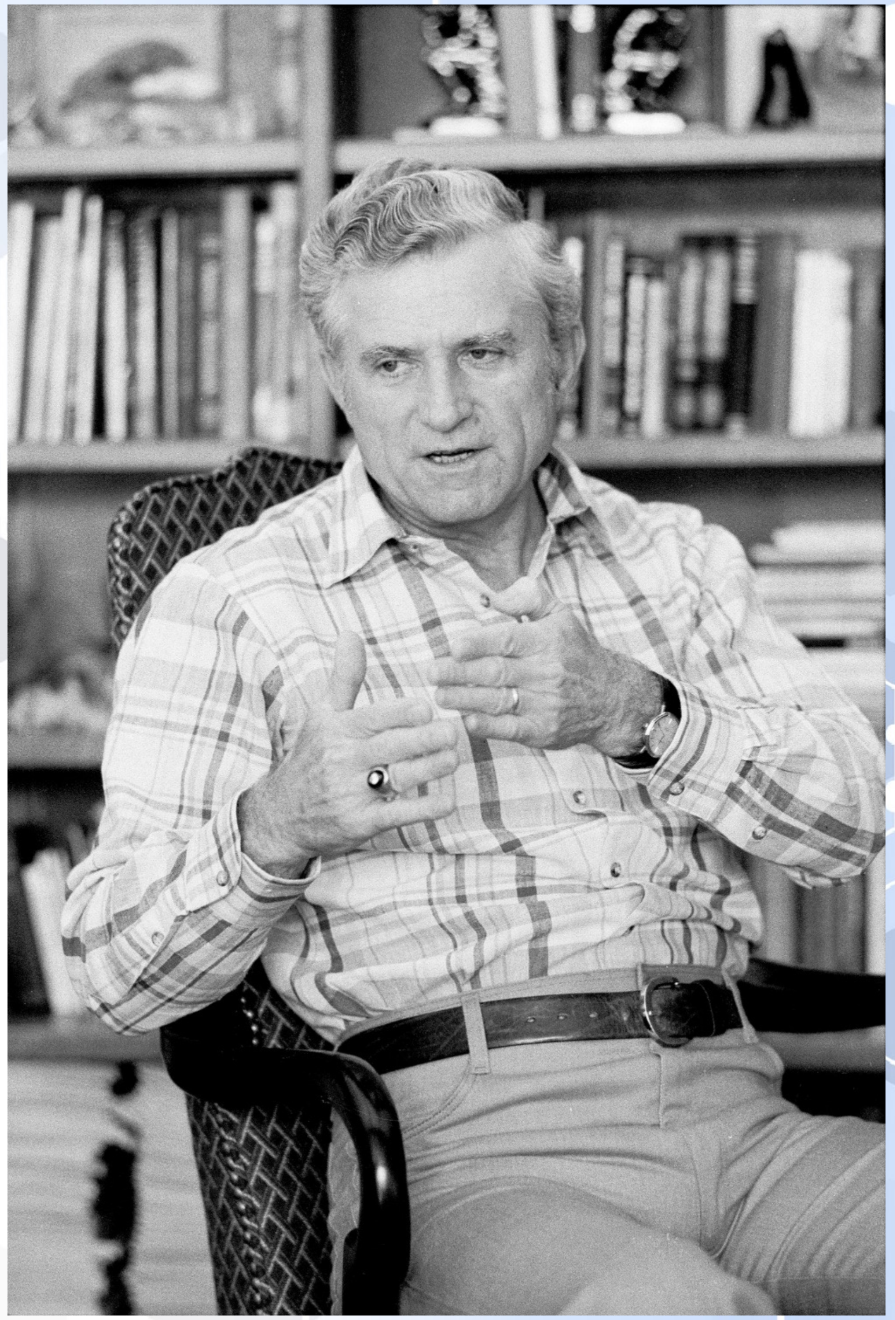
Scalability: *Gene Amdahl was a smart guy!*

All scaling in parallel computers is limited by Amdahl's Law:

$T_{execution} = T_{serial} + (T_{parallel} / Nthreads)$

Is there more to it than that? **NO!** … But portions of both T_{serial} and $T_{parallel}$ are often overlooked. Pathfinder minimizes or eliminates these hidden costs.

Components of T_{serial}	Conventional Computers	Pathfinder	Components of $T_{parallel}$	Conventional Computers	Pathfinder
Serial part of user Program	Same		Parallel Part of User Program	same	
Cache-Coherent Write Access	Coherency Messages – 10s to 100s of cycles	NA	Message Transmit / receive driver code	100s of instructions	NA
Message System Initialization	100s of instructions	NA			
Locks / critical regions	100s of instructions	10s of instructions			
Thread startup / synchronization	100s of instructions in OS / Runtime	10 - 40 cycles			



The Pathfinder Architecture revolves around minimizing / eliminating the non-program portions of T_{serial} through customized hardware, resulting in excellent scaling to thousands of Nodes and millions of threads!

WHAT IS LUCATA PATHFINDER?

A specialized computer designed to efficiently perform analytics on large datasets in real time.

- Shared memory multiprocessor that scales from 1 to 64K nodes (powers of 2)
- No Data Caches, No TLB
- Performance scales near-linearly over a range of system sizes
- No software modifications needed as system size grows
- Supports over 64k concurrent threads per Rack
- Up to 8 TB DRAM / Rack in current generation, 128 TB / Rack by end of year
- Memory-Side Processing of atomic memory operations supporting synchronization and fine grain access control
- Reduces Communications Bandwidth requirements by using *Migratory Threads*
  - Thread moves its execution site to where the data resides
  - Results in dramatic reduction in interconnect bandwidth used
- Tightly integrated with x86 (Xeon / Epyc) Host server
  - Runs operating system (RHEL)
  - Dense / Sequential portions of applications can execute on Host

ADVANTAGES OF LUCATA PATHFINDER

Specialized analytics hardware platform, designed to efficiently perform analytics on large datasets in real time.

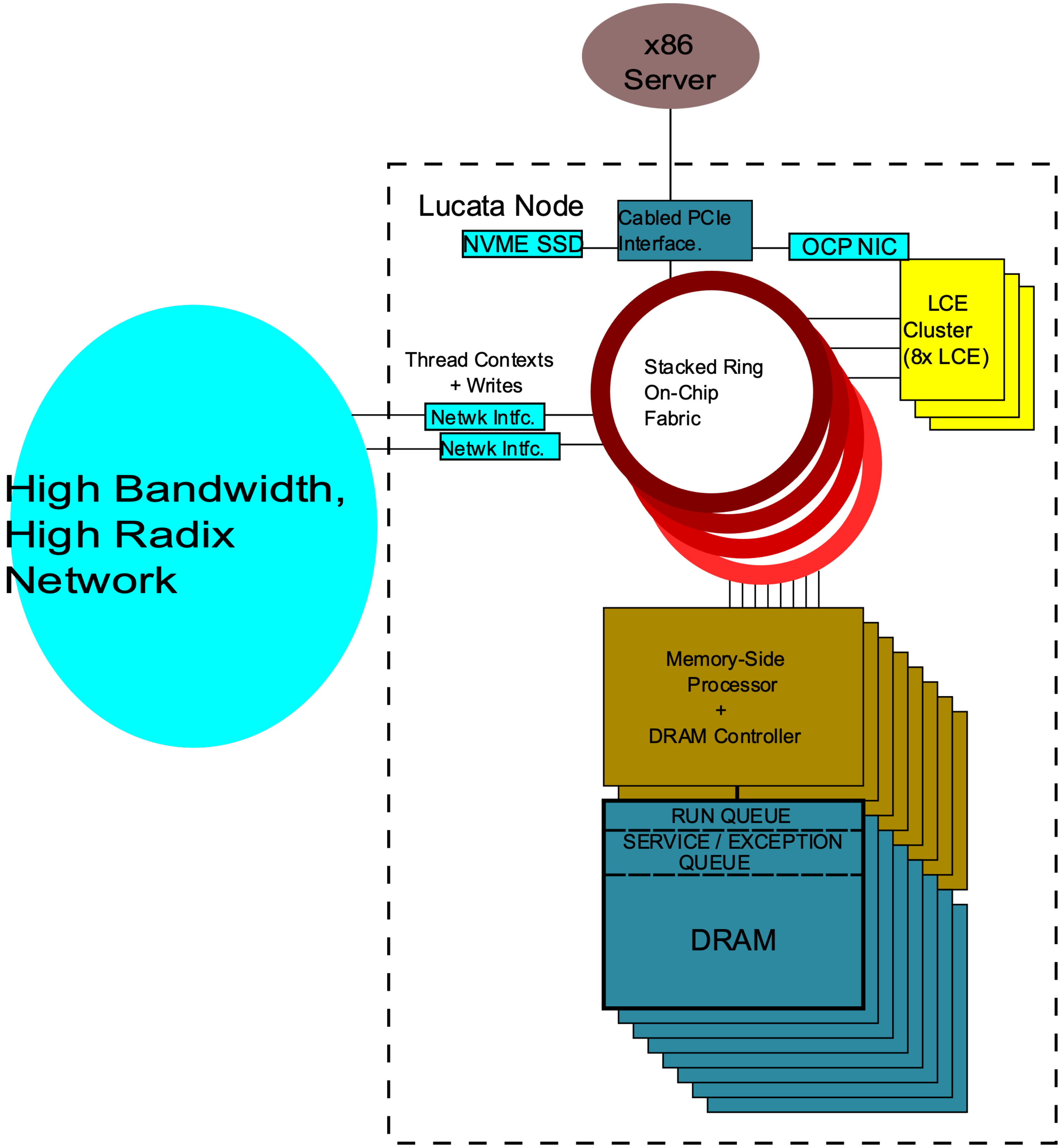
- **Moves computation to the memory** system containing the data being processed and performs the processing at that memory. Multiple memory channels per node, each with its own Memory-Side Processor, maximize parallelism.
- **Global Shared memory** handles datasets up to multiple **petabytes** in size..
- Petabyte-scale **Global Shared Memory** that can span hundreds of cabinets and can support **millions of concurrent threads**.
- **Eliminates** most of the hidden **serial component of applications** by replacing networking stack with hardware message forming and delivery.
- **Barrel Processors hide** nearly all **stalls**, maintaining nearly perfect IPC (Instructions Per Clock) > 0.9. Enough Barrel Processors are provisioned to keep the memory system (the most expensive component) fully utilized.
- Maintains **linear performance scaling** in both dataset and system size.
- Support for real-time **streaming ingest** from **hundreds** of high-speed network interfaces.
- **Millions of concurrent threads** managed by hardware can be dedicated to work on a single complex task or a large number of concurrent tasks.
- **Simplified programming** using familiar programming languages such as C / C++ and the ubiquitous Linux operating system in a shared memory environment. Properly written programs can scale across all system sizes without modification.

Pathfinder System Architecture

PATHFINDER Bread (Migratory Threads) & Butter (Memory-Side Processing)

- **Migratory Threads** *push* the compute to the data (instead of pulling data to the core doing the computation)
  - Thread moves its execution site to where the data resides
  - Results in *dramatic reduction in interconnect bandwidth* used
  - *Reduces latency* by making all memory reads local
  - *Improves core utilization* by vacating *HART* (**HAR**dware **T**hread – register set holding the context of an executing thread) on starting core as soon as remote access triggers migration, allowing a new thread to begin execution
- **Memory-Side Processing** performs computation at the memory controller instead of in processing cores
  - *Reduces data movement* between memory controller and core
  - Implements *fine grain atomic updates* of shared memory locations
  - *Avoids need for Cache Coherency*
  - *Increases concurrency* by allowing thread to launch many atomic updates and synchronize on Acknowledges
  - Operations: CAS, SWAP, ADD, AND, OR, XOR, MIN, MAX, FP-ADD on 8, 16, 32 and 64-bit values (FP on 32 and 64)
  - Upcoming: MemCopy, Vector Ops, Dyadic Ops, Indirect Ops, String Search-and-Match

Pathfinder Node Architecture



Pathfinder – A

- Agilex-7 FPGA
- 1 Node / Board
- COTS Board from established manufacturer
- Up to 1 TB / Node (4x DDR4 DIMMs)
- Up to 1 TB / Node CXL Mem (2x modules - option)
- 250 MHz Core Clock (est.)
- 16 LC "Barrel" Processors / Node
- 12-16 MSPs feeding mix of DDR and CXL Channels
  - Atomic ADD, AND, OR, XOR, MIN, MAX, FP_ADD
  - ~ 250 M atomics / s / MSP
  - New Vector and Dyadic Operations
- 200 G Enet Port to System Network
- Edge Finger PCIe G5 SC Connection
- Uses SC's IO device slots

Massive Concurrency thru ‘Barrel’ Processing Cores

Each Lucata Core (LC) is a “Barrel Processor”, executing from 64 different threads at once

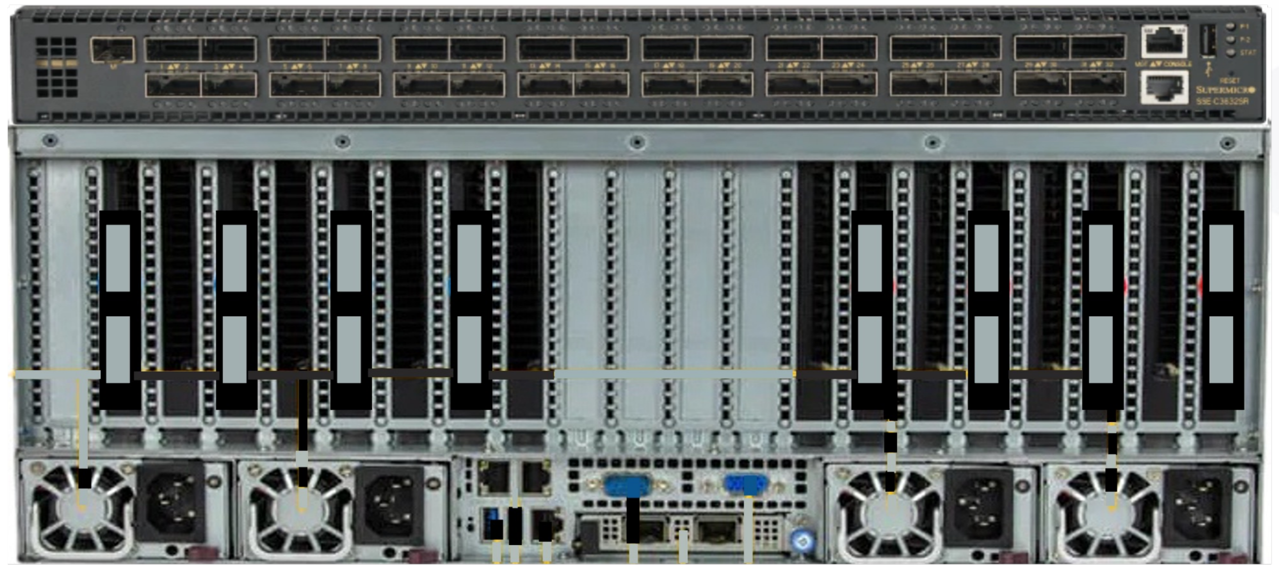
- Register file holds 64 contexts (*HARTs = HARdware Threads)* at once
  - Extra Register File Read and Write Ports support thread unload / load without stealing execution cycles
- Hardware Scheduler chooses a different HART to execute every cycle from among HARTs that are *Ready*
  - Upcoming *Priority Mode* selects among Ready High Priority threads when such threads exist – improves performance in critical regions involving locks
  - Migrating / Completed threads unloaded from HARTs and new threads loaded without interfering with executing threads
- *Hides latency* of local memory access
- Thread alternation *eliminates need for Hazard Detection and Bypass logic*
- Shares one set of power-hungry instruction processing and execution logic among 64 HARTs
- Icache shared among all HARTs – gets high hit ratio because generally have many concurrent threads executing the same program
  - Hardware prefetcher has good success filling Icache before program reaches a missing line

16 LCs per node can generate over one memory access per clock (on average, assuming ~7% of instructions reference memory), thus maintaining *full memory utilization*

LUCATA PATHFINDER-A

Programming Model Options

- ✓ GraphBLAS
- ✓ C/C++
- ✓ Python on SC
- ✓ Lucata Libraries & Utilities



8 Lucata Nodes in 5U Rack Space:
- Execute 8K Concurrent Threads
- No D-Cache, No TLB
- 128 64-way multi-threaded Cores
- 32 DDR4 Memory Channels
- 8 100G NICs – up to 2 Tb/s BW

16 TB memory (8 TB DDR + 8 TB CXL)

Up to 16 NVME SSDs

Pathfinder Attributes

- State-of-the-art FPGA allows instruction set flexibility
- Parallel load capability
- Scale capabilities with: 16 -> 64K Nodes
- Multiple, Parallel queries
- Dynamic Update / Addition of new data
- Uniform Shared Memory to 2^{54} Bytes
- Atomic Memory Operations on all memory, at full memory access rate
- Incremental Updates to Database with concurrent queries
- Can act as Intelligent Multi-server Memory Pool
- ~ **24 KW** / Rack, Air Cooled – **No Liquid Cooling Required**

Pathfinder-A: COTS Agilex Boards in COTS Server

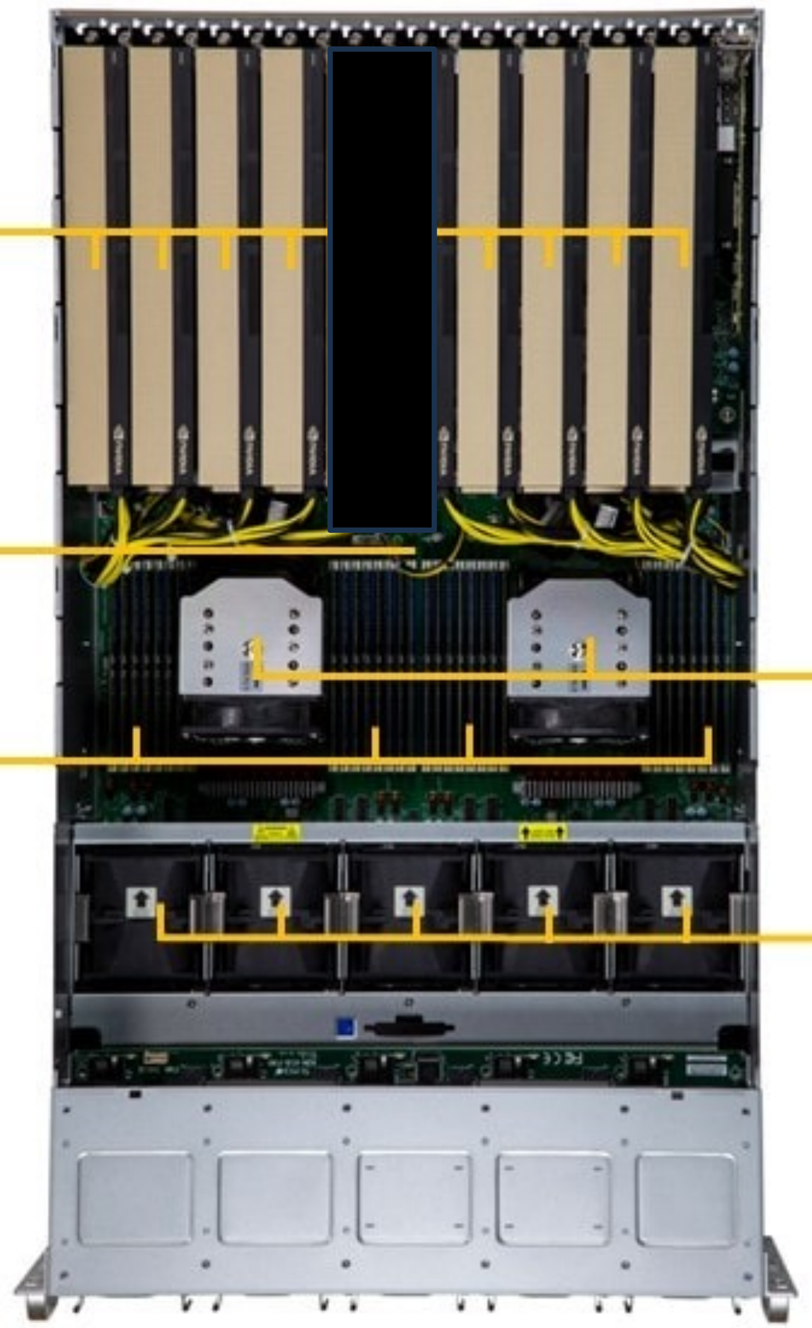
8x SSD

8 or 16x CXL Memory Modules

8x Lucata Nodes plus Hot standby Nodes or optional GPUs

2x NVMe M.2 (on motherboard)

32 DIMM Slots DDR5



Dual 4th Gen Intel® Xeon® Scalable Processors up to 350W TDP

5 Hot-plug System Fans

Software Environment

Software Environment

- Low-level Programming Model
  - C/C++/Cilk
  - Distributed Data Structures
  - C/C++ Parallel Programming Abstractions
  - Architecture Specific Features (e.g. Atomic and Remote Operations)
- Domain-specific Libraries
  - Beedrill Graph Algorithms
  - Lucata Graph BLAS/LAGraph Algorithms
- Program Execution Model and Software Stack
- Application Development and Porting Process
- Example Applications
  - Beedrill Graph Data Structures and Page Rank Algorithm
  - Concurrent Graph Algorithms
  - Pattern Matching Workflow
- Development Environment
  - Compiler Toolchain and Libraries
  - Simulator and Profiling Tools
  - Hardware Performance Counters and Profiling Tools

Programming Model

High Productivity Computing

IDC predicts a shortfall of four million developers by 2025 and Gartner lists talent shortage as one of the top five emerging risks for companies.

Lucata Programming Models

Increasing performance & flexibility

Commercial Graph Databases and analytic software packages

Such as Trovares, RedisGraph, Neo4j

Lowest programming effort for user, requires porting

Domain specific libraries and APIs

(GraphBLAS/LAGraph today, others in the future)

Many algorithms available in Open-Source Community. Supports interfaces like **Arkouda**

C/C++ with Lucata libraries and parallel extensions

Highest performance, greatest flexibility

Requires porting effort by user, but few new concepts for skilled programmers

Increasing level of abstraction

Open-Source Software, No vendor or platform lock-in

Cilk/C/C++ Programming Model

- Dynamic parallelism using OpenCilk language (extensions to C/C++)
  - **cilk_spawn**: spawn a thread (locally or remotely)
  - **cilk_for**: distribute iterations of a loop among parallel threads
  - **cilk_sync**: synchronize all the threads spawned in a function

  * Thread scheduling handled efficiently in hardware, not via software runtime

- Distributed data structures
  - **Distributed arrays** – Striped across the nodes in the system with various block sizes
  - **Replicated data** – Data allocated at same offset at each node; Accesses are to local instance to eliminate migrations for frequently used data

- C/C++ Parallel abstractions
  - **apply/for_each** – applies worker function to each iteration of loop, distributes work among parallel threads based on scheduling policy
  - **reduce** – parallel reduction over array
  - **fill** – parallel array fill

Specialized Memory Operations

Memory operations performed at the Memory Side Processor (MSP)

- Migrating Atomic Memory Operations for lock-free or fine grain locking algorithms
  - **Integer**: MIN, MAX, ADD, AND, OR, XOR, SWAP, CAS
  - **Double Precision Floating Point**: ADD, SUB, SUBreverse
  - Thread migrates if data is remote
- Non-blocking Remote Atomic Memory Operations
  - **Integer:** MIN, MAX, ADD, AND, OR, XOR
  - **Double Precision Floating Point**: ADD, SUB, SUBreverse
  - Fire and forget
  - Do Not cause thread migration
  - Do Not return result
  - Do return acknowledgement
  - **FENCE** used to ensure all ACKs have returned
- *Under Development:* Stationary Loads and Atomic Memory Operations
  - Same set of operations as migrating atomics + remote read
  - Thread does not migrate – sends a remote operation and waits for result to be returned (e.g. remote read or remote atomic add)

Lucata GraphBLAS Library

- GraphBLAS: library of standard building blocks for graph algorithms in the language of linear algebra
  - GraphBLAS API specification https://graphblas.org
  - Suite Sparse GraphBLAS implementation http://faculty.cse.tamu.edu/davis/GraphBLAS.html
  - Lucata GraphBLAS (LGB) implementation for the Lucata architecture (GraphBLAS v2.0)

Basic Functionality (Matrix/Vector/Scalar)
- Multiplication
- Element-wise operations
- Reduction to vector/scalar
- Apply unary/select operator
- Submatrix assignment/extraction
- Transpose
- New, build, get, set, clear, free…

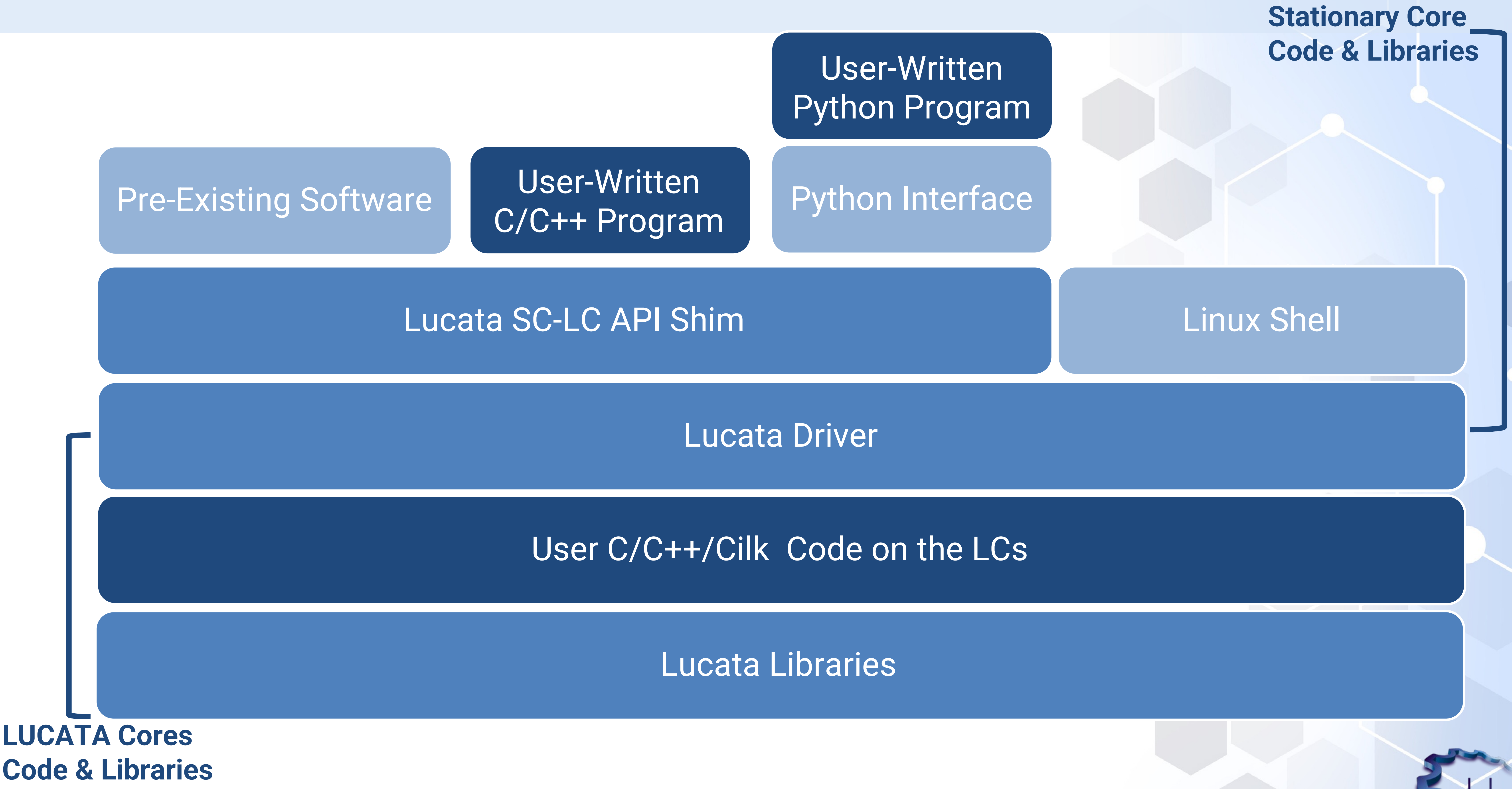
Descriptors modify behavior
- Accumulate
- Mask
- Transpose
- Replace

Component	Examples
Type	int8/16/32/64, Boolean, …
Monoid	Plus, times, min, max, …
Semiring	Plus_times, Min_plus, …
Unary operation	Identity, inverse, negation, …
Binary operation	Equal, greater than, …

LAGraph Library

- LAGraph is a library of algorithms built using GraphBLAS
- “Vanilla” algorithms target only functionality defined in the v2.0 specification (no Suite Sparse extensions)
  - Breadth first search
  - Triangle count
  - Betweenness centrality
  - Single source shortest path
  - Connected components
  - Page rank

Programming Stack



Program Execution Model

- **Stand-alone model**: main program runs on the Lucata Pathfinder cores (LCs)
  - Main LC program
    - Parallel program written entirely in C/C++/Cilk with Lucata libraries
    - Compiled with the Lucata OpenCilk Compiler
  - Linux runs on the standard x86 host (SC)
    - Driver code on the SC loads the program to the LCs and launches the initial thread running main() to the LCs
    - Initial main thread can spawn/sync additional threads and migrate through the system as needed
    - Upon completion, main thread returns to the SC driver
    - SC runtime provides file I/O, system calls, and exception handling

- **Accelerator model:** main program runs on the host (SC) and issues calls via the SC-LC API to run data-intensive kernels in parallel on the LCs
  - SC-LC API library provides interface, similar to OpenCL or CUDA
    - Allocate memory on the LCs
    - Move data back-and-forth between the SC and LCs
    - Invoke kernels (threads) that run on the LCs
  - SC program
    - Written in C/C++/Python or other language that can call C functions
    - Compiled with standard tools
    - Runs on SC and uses SC-LC API library to make calls to specialized kernels in LC program running on LCs
  - LC program written in C/C++/Cilk with Lucata libraries and compiled with Lucata compiler

Program Development Characteristics

- Traditional distributed system
  - Program written to take advantage of cache
  - May require mixed programming model – e.g. OpenMP + MPI
  - Explicit distribution of data and definition of communication pattern
  - Ineffective for irregular data access patterns
  - May need to rewrite communication patterns for changes in system configuration
- Lucata
  - Program does not require cache-based locality or explicit communication for correctness
  - Cyclic distribution typically provides first level load balance
  - Programs with little locality (e.g. pointer chasing) system will perform better than a traditional cache-based/distributed system
  - Can explicitly distribute data to take advantage of node-level locality when it exists
  - Program scales with size of system

Application Development Process

- Identify execution patterns
  - How computation touches data
  - Where parallelism can be extracted

- Distribute data/computation using distributed data structures
  - Cyclic or random distribution will be correct
  - Take advantage of node level locality where available
  - Balance workload/avoid hotspots where possible*

- Implement parallelism using cilk_spawn and/or parallel abstractions
  - Target enough threads to fill HARTs
  - Choose threading model
    - Parallel – uniform workload, so each thread is given the same number of iterations to process
    - Dynamic – work per element may be unbalanced, so each thread atomically grabs work from a local work queue

* Simulator, profiling and visualization tools to understand program execution characteristics

Application Porting

Programming Model	Porting Effort	Comments
Parallel, shared memory (e.g. OpenMP, XMT)	Low effort	• Existing parallel constructs typically map to Lucata parallel abstractions • Often need to extract additional levels of parallelism to support more threads • MTA/XMT codes port most easily – highly multithreaded with randomized memory • Map data structures to distributed data structures
Sequential	Moderate effort, most flexibility	• Define parallel algorithms and data distribution from scratch
Distributed memory (e.g. MPI)	Moderate effort	• Explicit communication doesn't map as directly to multithreaded programming model (e.g. mpi_send/receive) • New stationary, synchronous remote operations will map more closely

Summary

- Lucata Pathfinder is a differentiated system that performs better per $, Per watt, and Per sq. foot for Large-scale critical workloads in graph analytics, Big Data, and AI/ML.
- Massive business opportunity to deliver Pathfinder platform in enterprise-scale deployments across commercial, government, and defense spaces.
- Lucata requires a partner to provide marketing / distribution clout to make this opportunity a reality.

THANK YOU!

Marty Deneroff, CEO mdeneroff@lucata.com

Jim Light, VP Business Development jlight@lucata.com