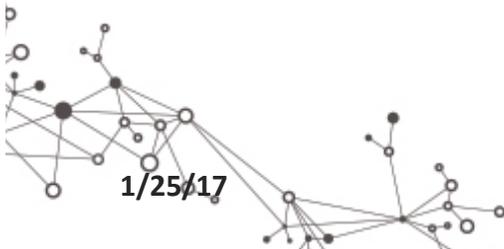
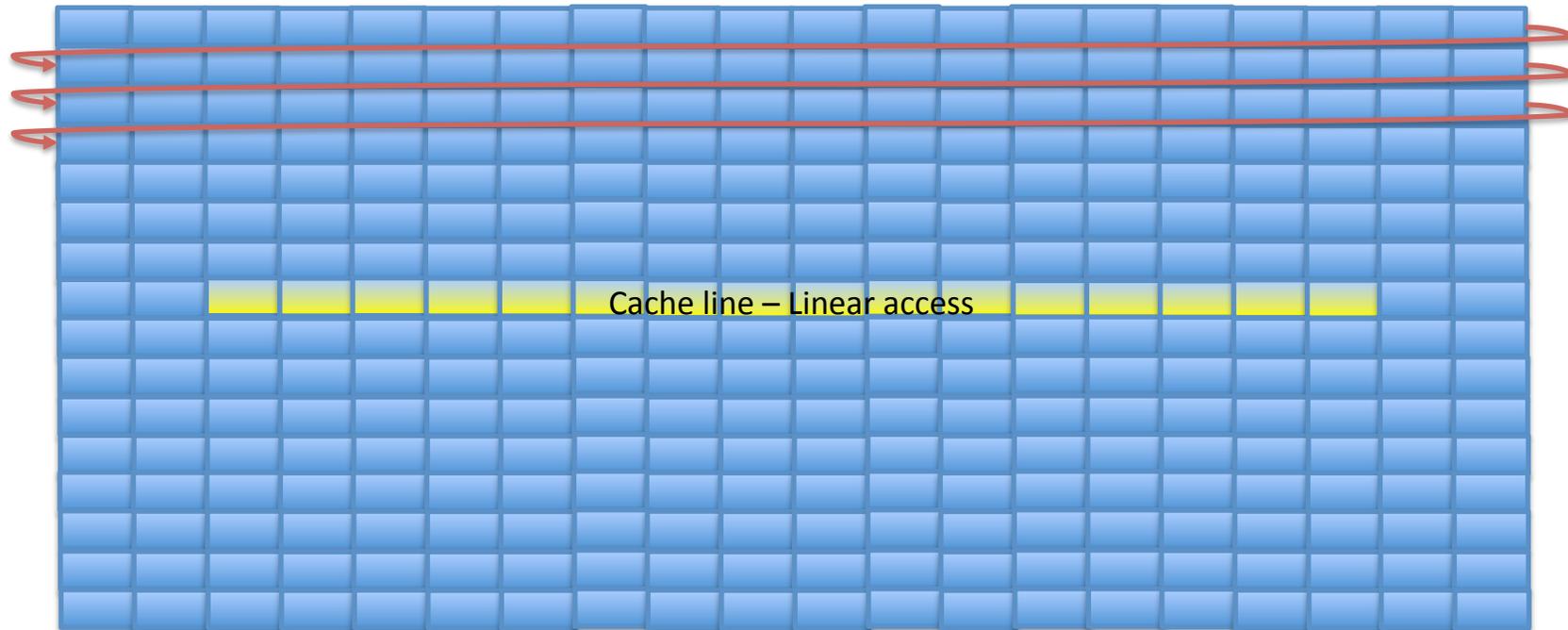


AMS Seminar Series
NASA Ames Research Center
December 15, 2016

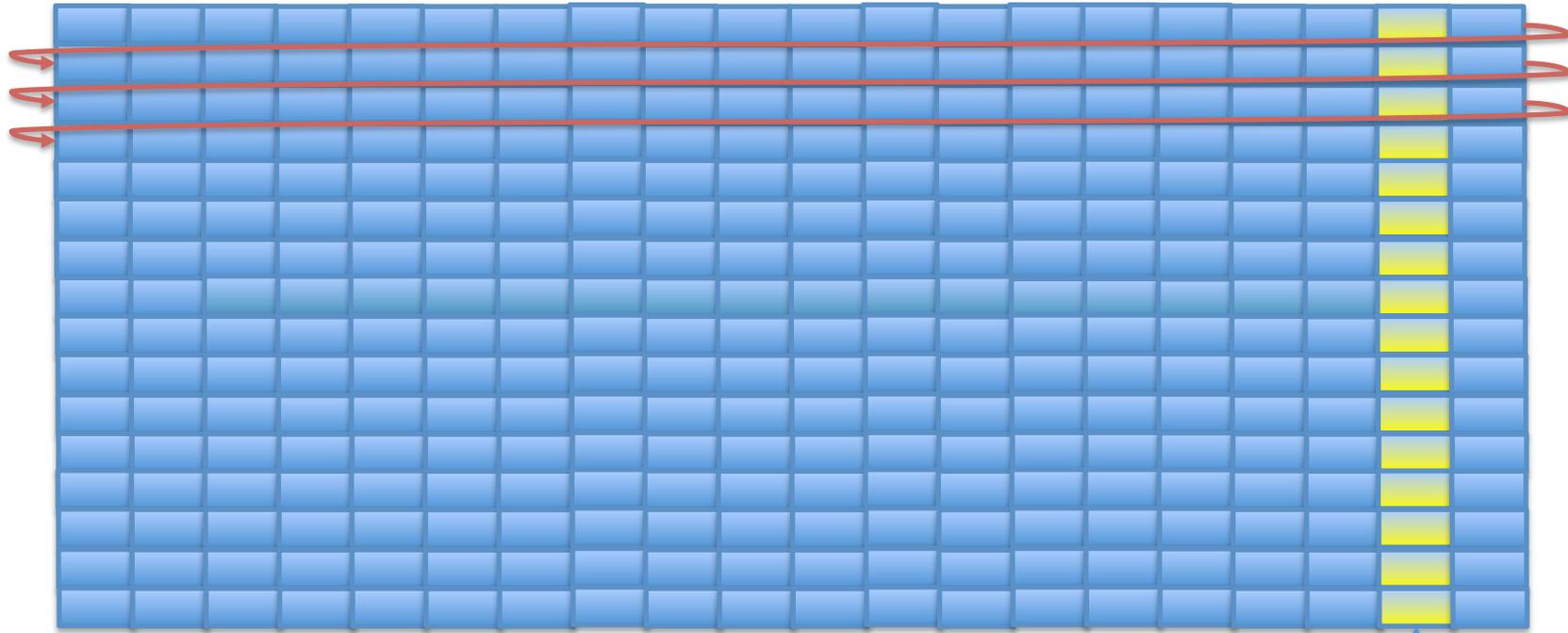
Emu TECHNOLOGY
READY FOR SOMETHING REALLY NEW?

Performance Advantages of a Data Centric Computer Architecture

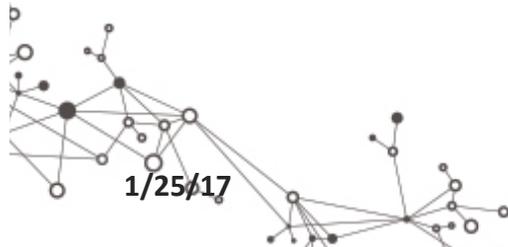
Current Memory



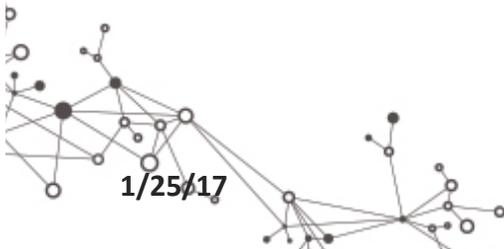
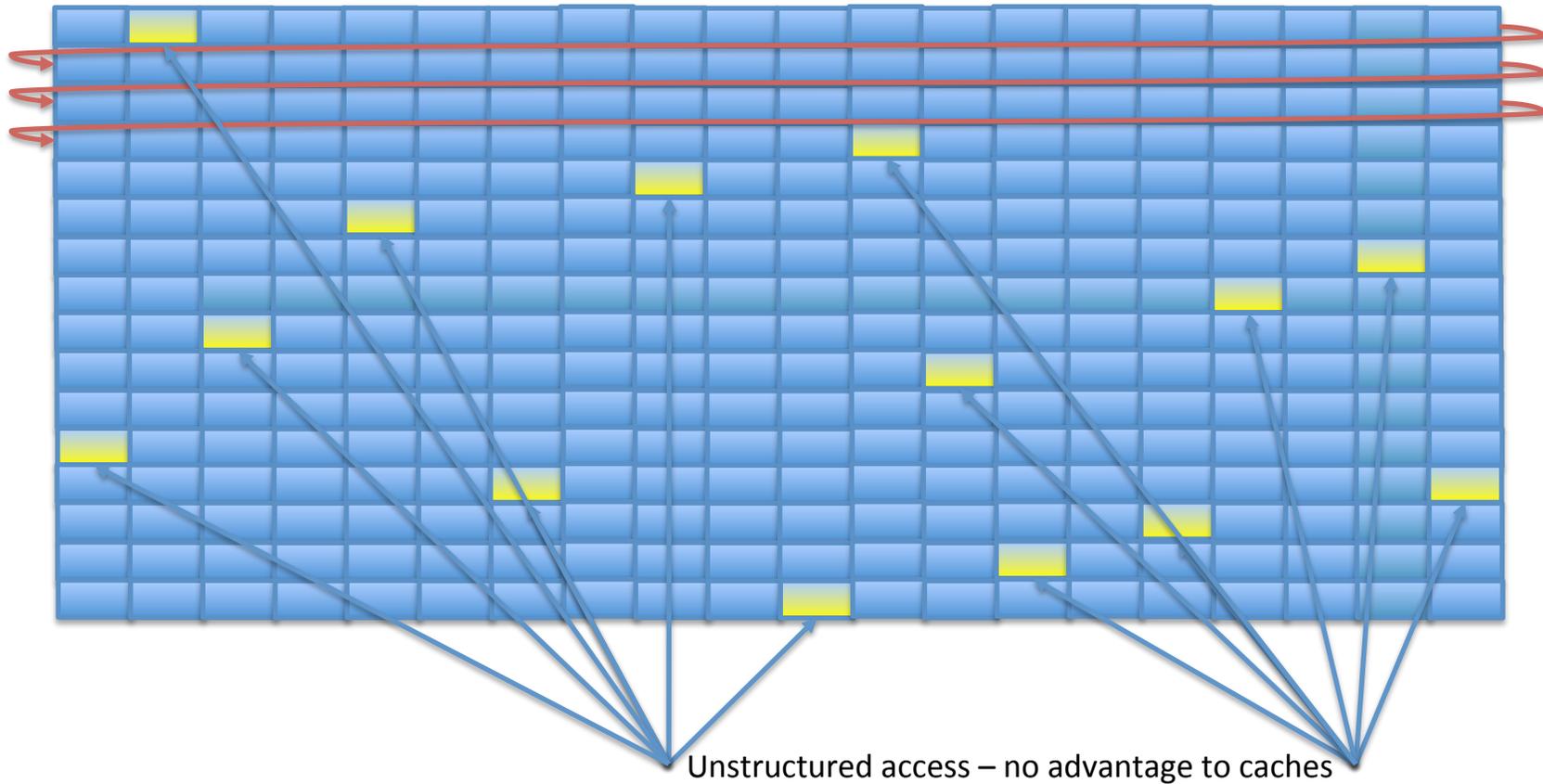
Current Memory



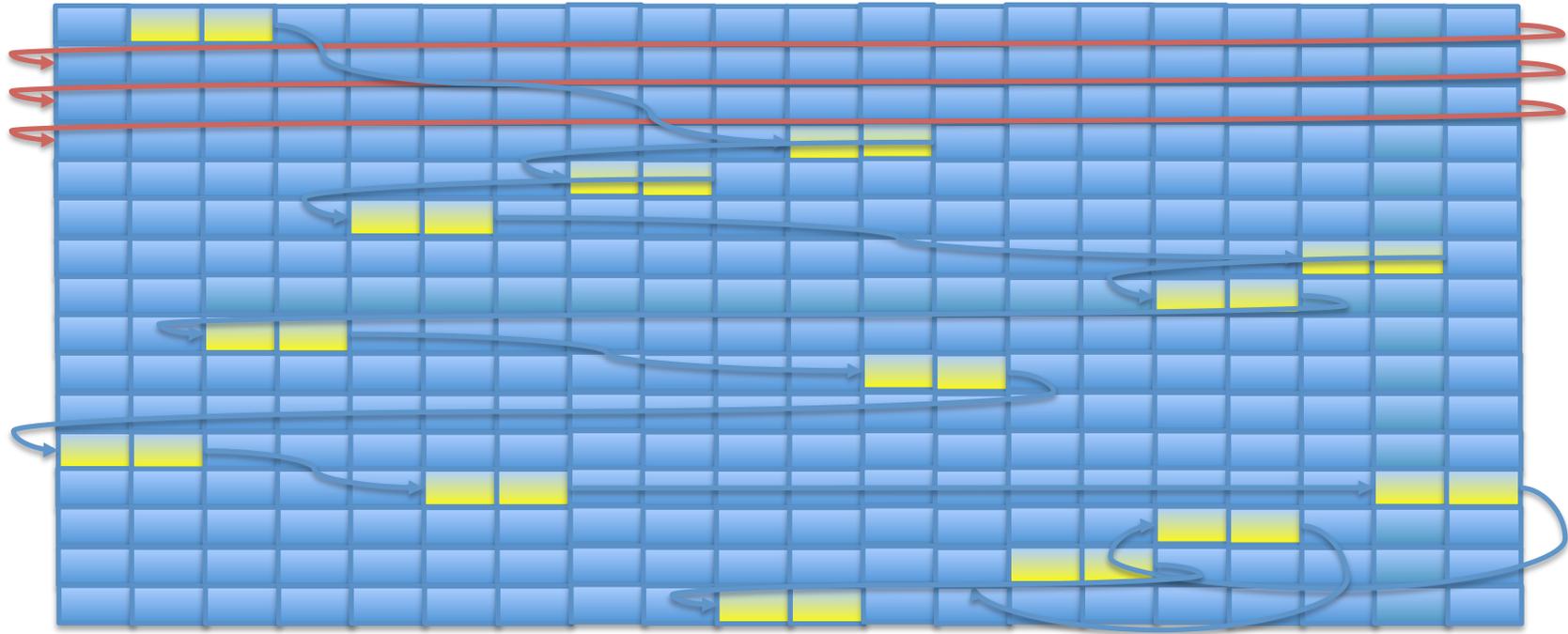
Strided access – no advantage to caches



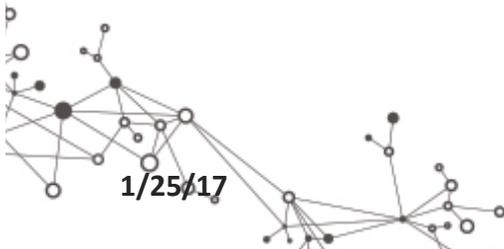
Current Memory



Current Memory

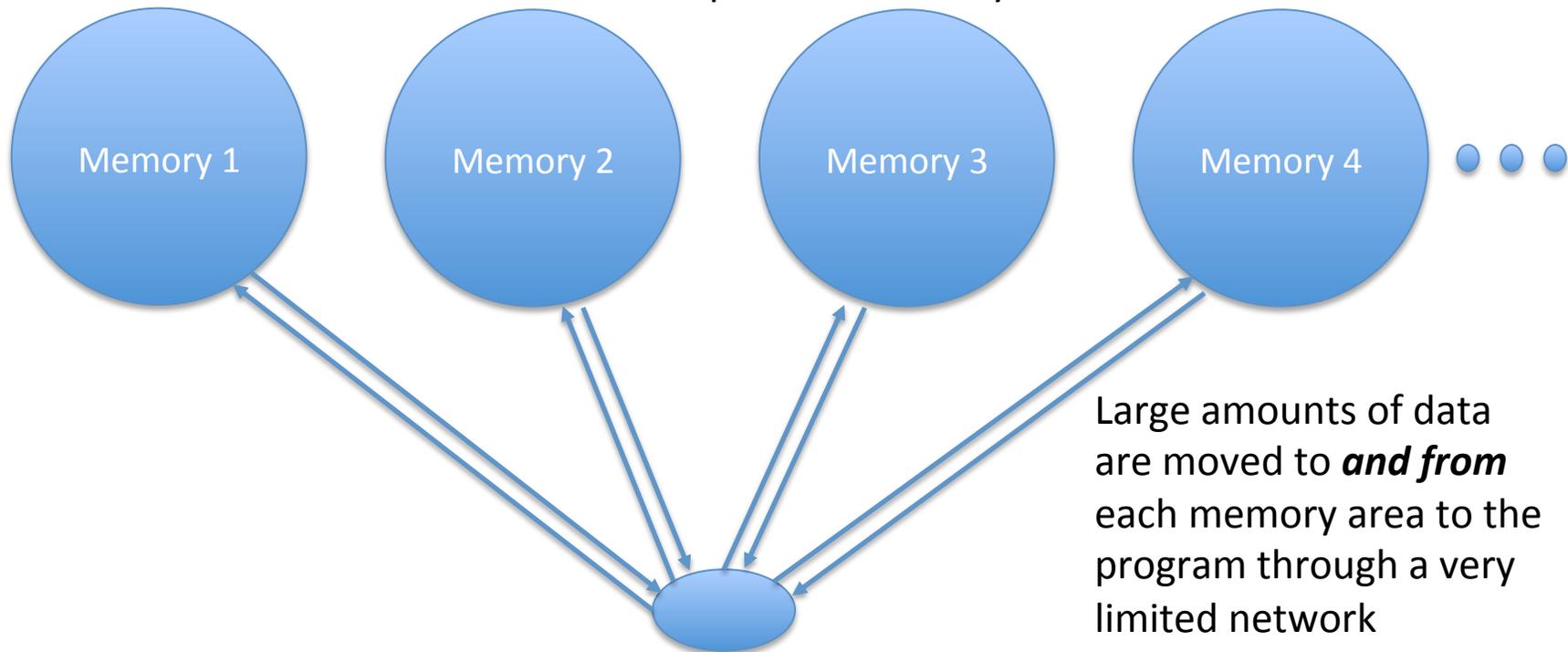


Forward looking linked list access – no advantage to caches

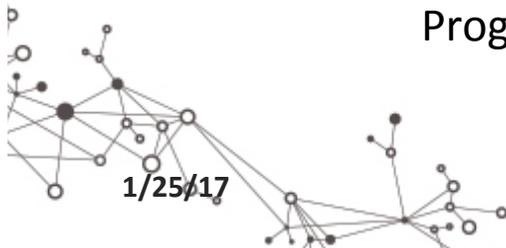


Current Memory

Massive amounts of data spread over many memories

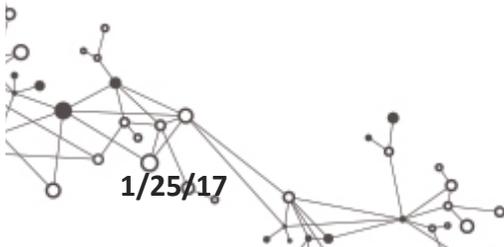


Program (very, very small compared to data)



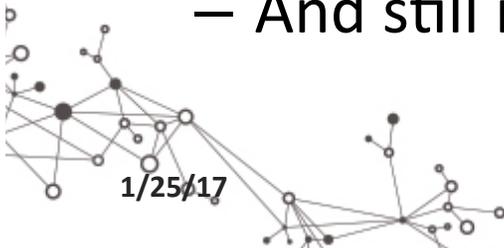
Introduction

- Issues with Today's Architectures
- Emu Migratory Near Memory Processing Architecture
- Programming Emu using Cilk
- Initial Emu products



Motivation

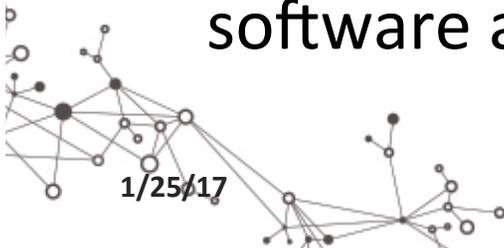
- Today's architectures optimized for highly local memory reference patterns with significant reuse
 - Large, deep cache hierarchies; wide memory paths
- New applications like Big Data are different
 - Very large unstructured data structures
 - Often with non-traditional searches: relationships
- Result: Big mismatch!
 - Scalable performance requires very complex programs
 - And still result in low efficiencies



Introducing the Emu Architecture

Designed for Irregular Data!! (The bigger the problem, the better)

- Designed from ground up for little or no locality
- Fully random access to individual 64-bit words at full efficiency
- Multithreaded cores hide nearly all latency
- Lower energy – less data moved shorter distances
- High radix network supports fast system-wide PGAS
- Compute, memory capacity, memory bandwidth and software all scale simultaneously



Going to the Mountain

If the mountain won't come to you, you must go to the mountain - ancient proverb

Instead of moving blocks of data across the network, Emu moves (“*Migrates*”) the program context to the locale of the data accessed

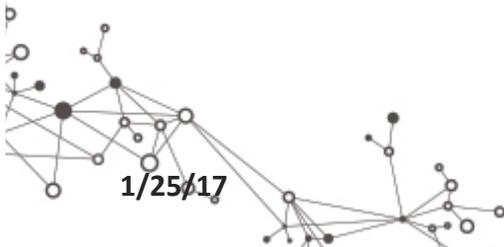
- Move program counter, working registers, thread status word
 - Application code replicated on each nodelet, never moves
- One-way trip – no return needed
- Invisible to programmer – triggered by reference to non-local address
- Latency completely hidden if sufficient active threads
- Writes transmitted on network without migrating, unless programmer explicitly requests migration



Emu: The “Put - Only” Architecture

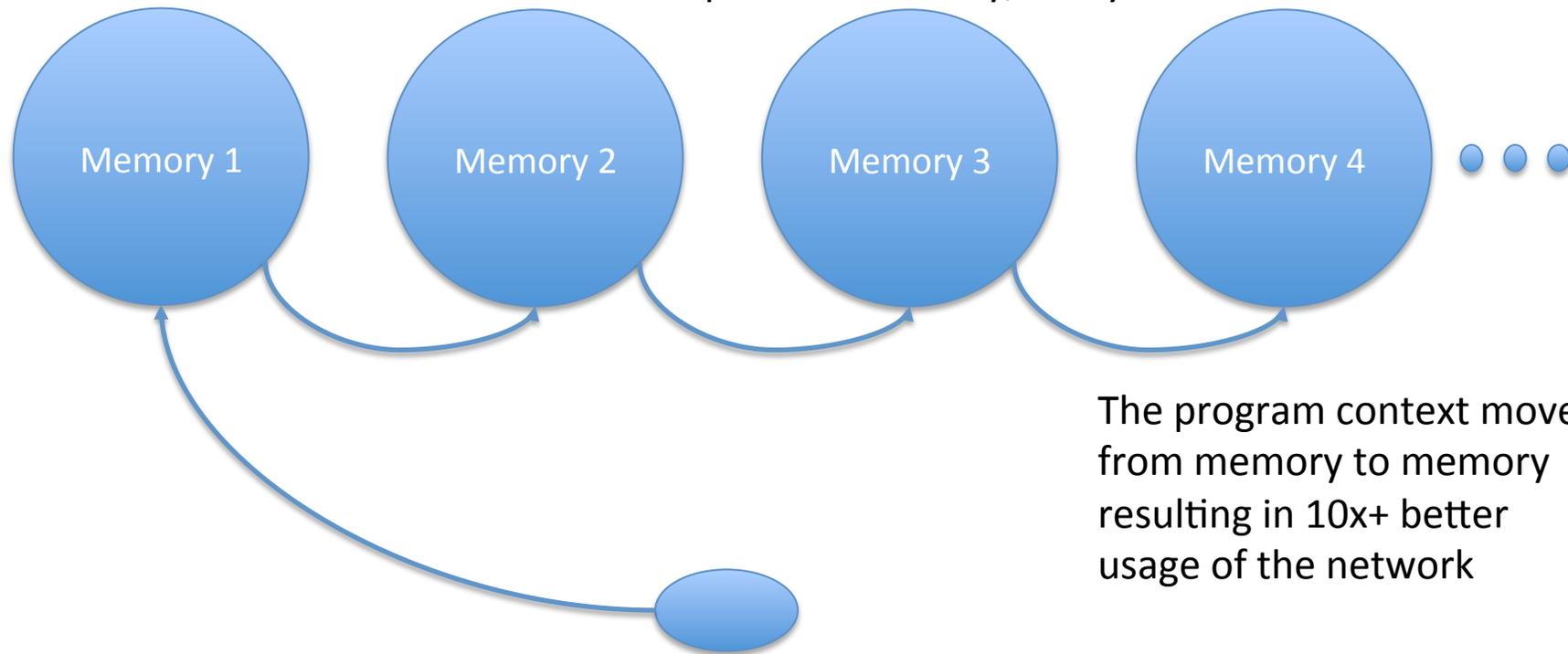
Reading a memory location on a different Emu node causes the context to move to that node, instead of sending a read request.

- This *wins big* especially when program pattern is series of often brief “visits” to widely dispersed data
- Processor utilization improved
 - Processors never stall for long periods waiting for remote reads
- Network is simplified
 - Doesn’t need to support round trip (read / response) messages
- Remote Writes can be performed directly or via migrations, under programmer (compiler) control.



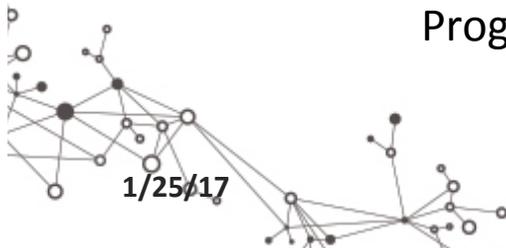
How the Emu Memory Works

Modest amounts of data spread over many, many memories



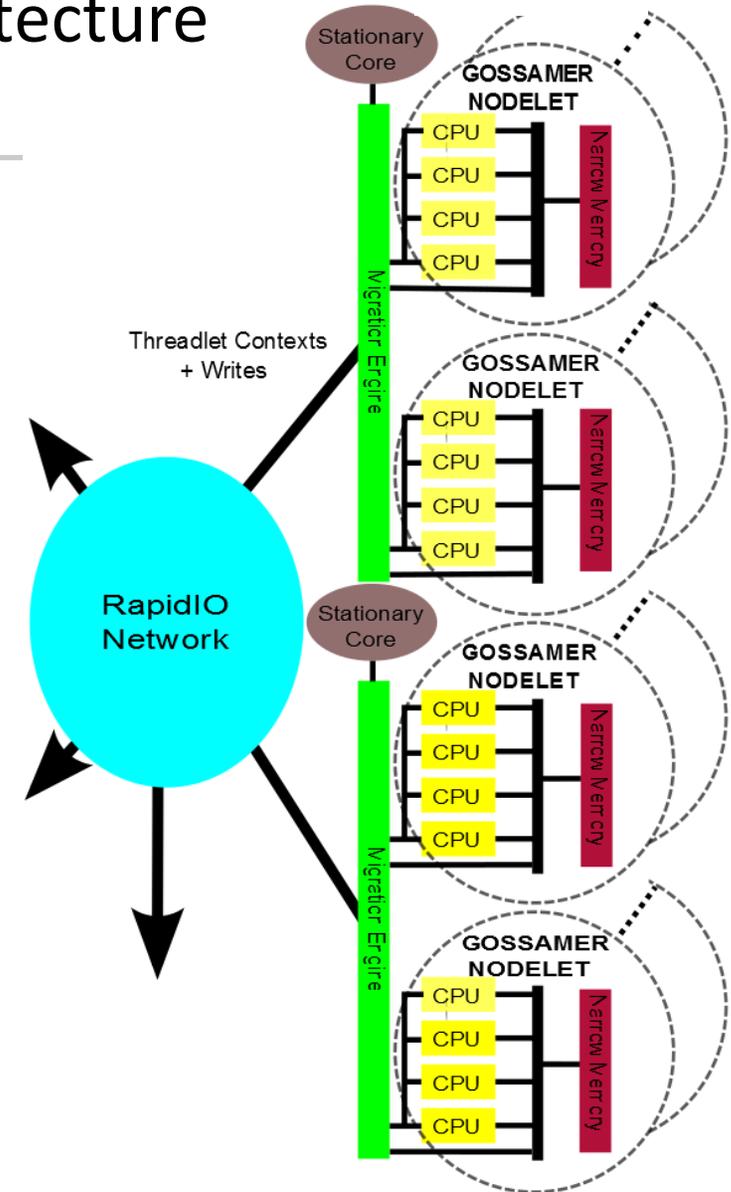
The program context moves from memory to memory resulting in 10x+ better usage of the network

Program (very, very small compared to data)



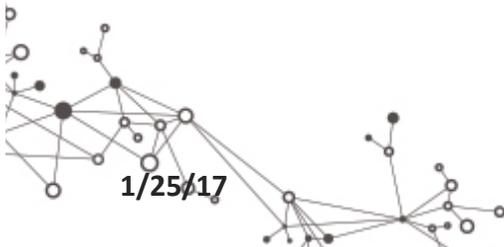
Memory Web Migrating Thread Architecture

- ➔ • **Single System-wide Address Space**
- **Stationary Cores (SCs):**
 - Execute Operating System
 - Manage IO / File System
 - Call or Spawn **Gossamer Threads**
- **Gossamer Cores (GCs) execute Gossamer Threads at *Nodelets***
 - Perform local computations & memory references (inc. atomics)
 - ➔ – **Migrate to other Nodelets w/o software involvement**
 - ➔ – **Spawn new Threads**
 - Call System Services on SCs
- **Programmed in Cilk**

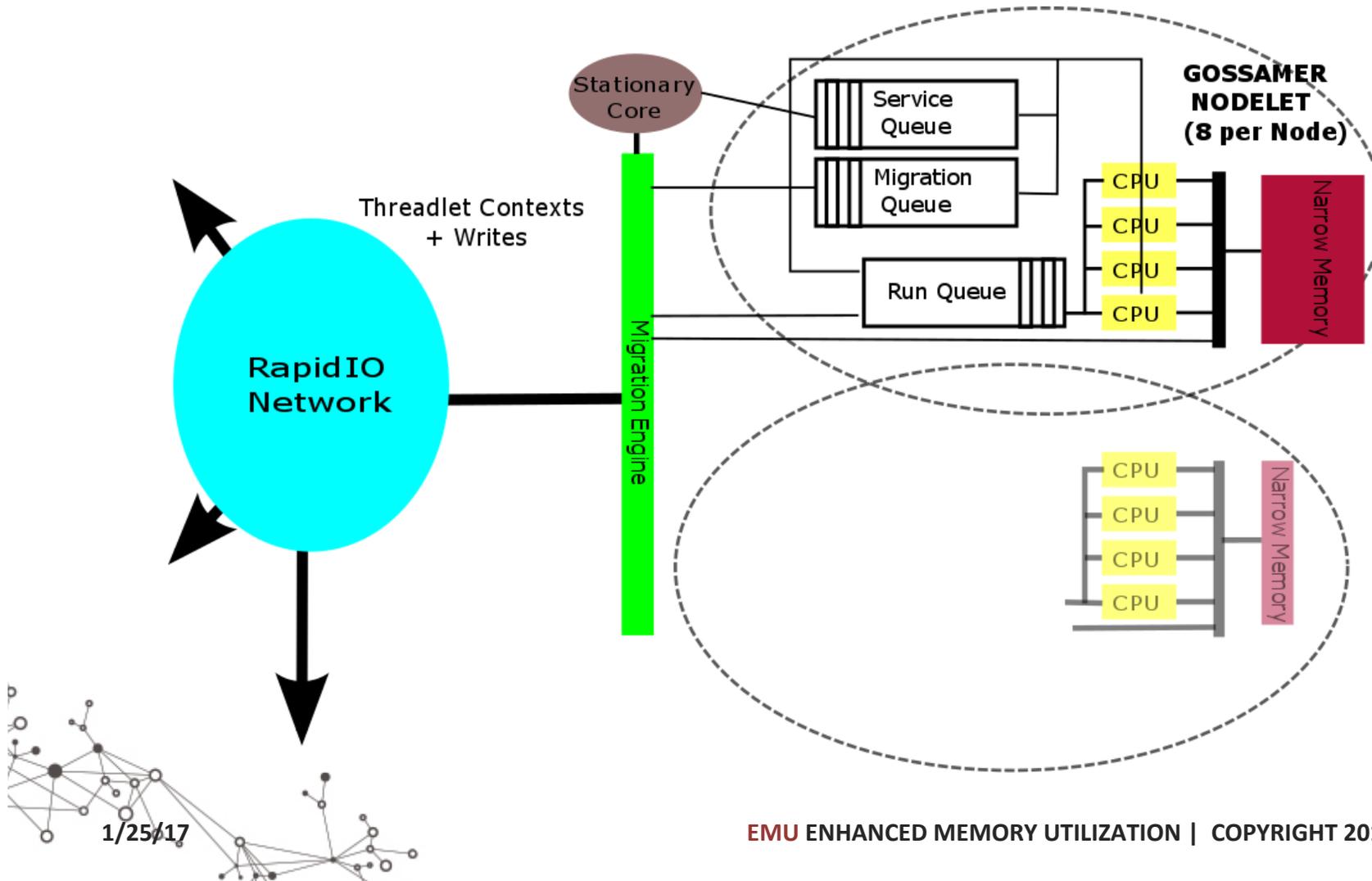


Gossamer Core Architecture

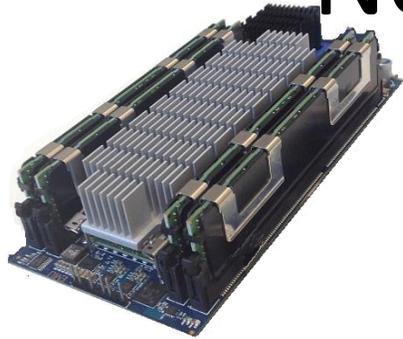
- Deeply pipelined, 64-way multithreaded core
- Accumulator + 16 general registers
- Rich suite of Atomic Memory Operations
- Single *SPAWN* instruction to create a new thread
- Thread scheduling is automatic and performed by hardware
- *RELEASE* instruction places context in a *Service Queue* for processing by Stationary Core



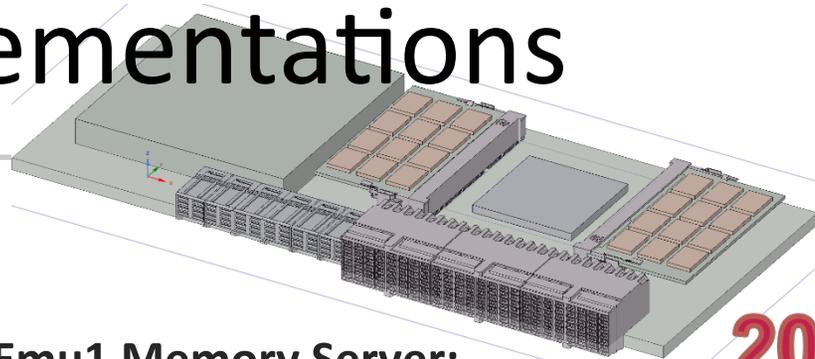
Node Architecture



Node Implementations



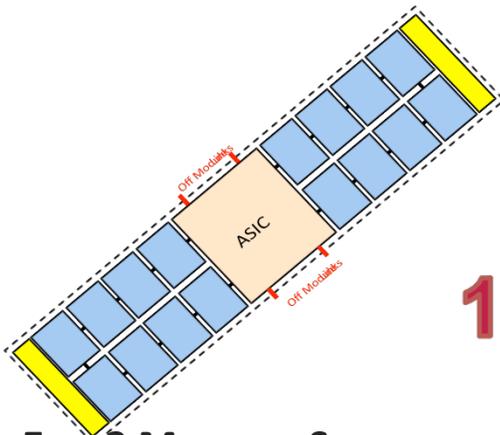
Proof of Concept



2016

Emu1 Memory Server:

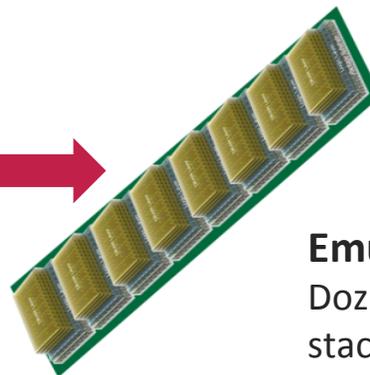
- Practical system for large data problems
- 8 Narrow Channels utilize all available bandwidth
- Solid State Disk for bulk storage
- Highly efficient, with moderate power density
- Scales to multiple racks
- High bandwidth Gossamer cores usable in future versions



**~2018
10x MW1**

Emu2 Memory Server:

Denser memory for higher bandwidth, ASIC for Compute node can utilize all available bandwidth



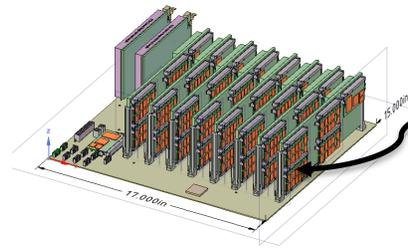
**~2020
>10x MW2**

Emu3 Memory Server:

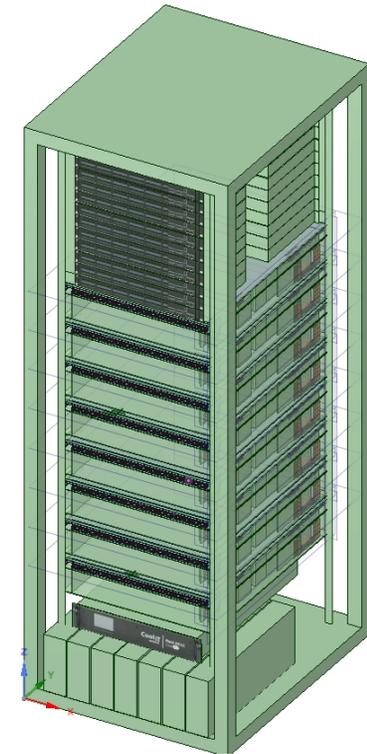
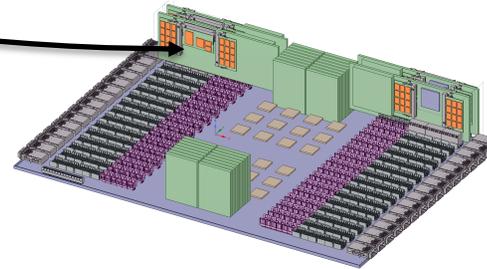
Dozens of nodelets in custom memory stack



Emu Product Initial Deliverables



The node boards are the same!

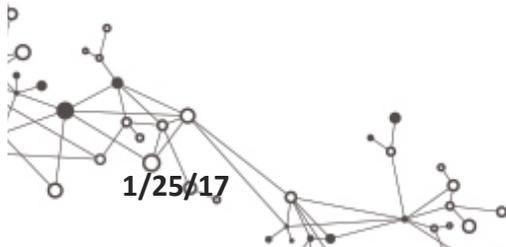


Emu Chick

- 2.2 GUPs performance
- Copy room environment
- FCS Q3'16

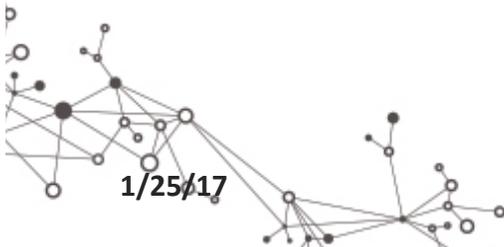
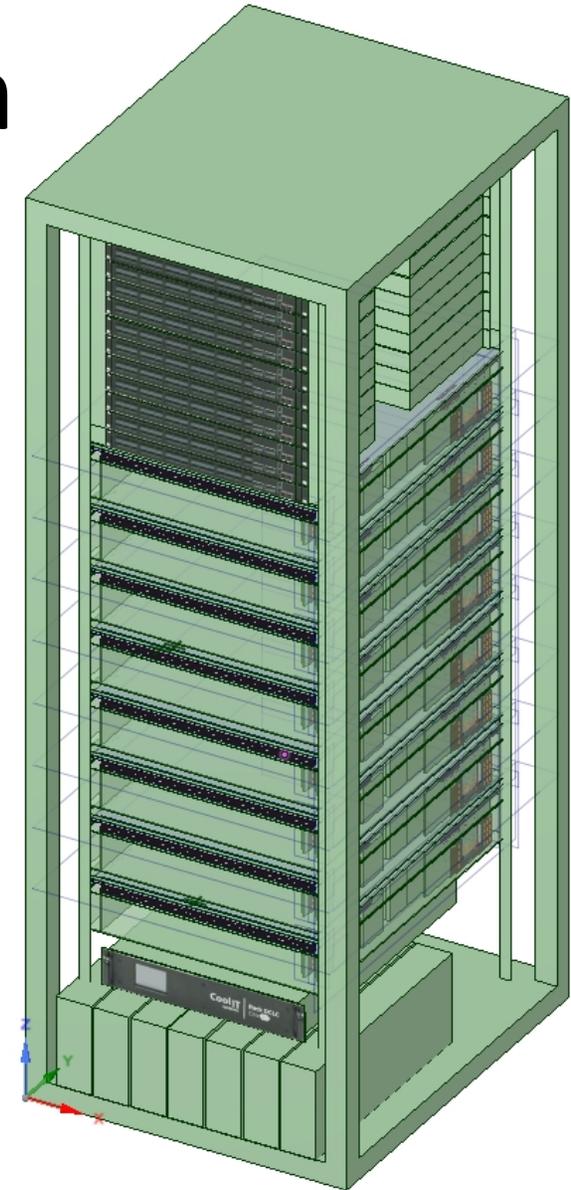
Emu1 Memory Server

- 70 GUPs performance
- Server room environment
- FCS Q3'17



Emu1 Rack Configuration

- **8 Chassis / Motherboards**
 - 256 Nodes (32 / Tray)
 - 8 8-lane PCIe Gen3 slots (2 / Tray)
- **24 ProDrive RapidIO Switches**
 - 96 SRIO 3.1 (6.25 GB/s) links between Supergroups and Central Switch
- **48 VDC Power Distribution**
- **N+1 Redundant Front-end Power Converter**
- **Air or Building Water cooling options**
- **Approximately 40 KW @ 440 VAC TriPhase**



Emu System Characteristics

Up to 64K Nodes / System

(256 Racks, 256 Nodes/rack, 8+ nodelets/node)

Air or building water cooled, ~40 KW/rack

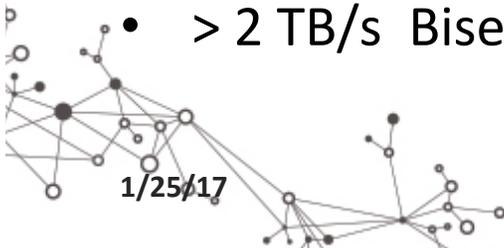
High radix RapidIO Network

Emu1 Rack

- 16 TB DDR4 DRAM
- 256 TB Solid State Disk
- 2048 Mem. Channels @ 2.4 GB/s
- > 2 TB/s Bisection BW/rack

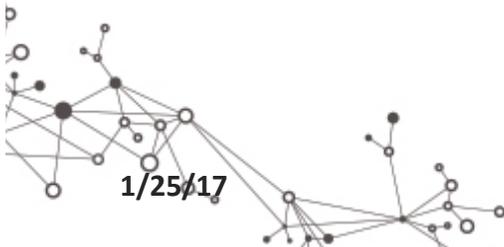
Emu2 Rack

- ~16 TB HMC Memory
- ~256 TB Solid State Disk
- 2048 Mem. Channels @ 20 GB/s
- >2 TB/s Bisection BW/rack



Programming Emu

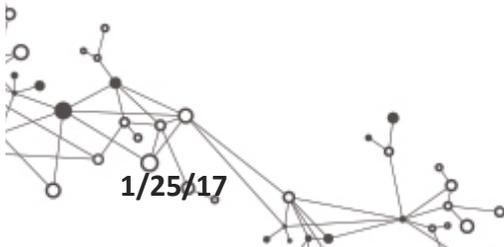
- Programmed in Cilk, a truly parallel extension of C
 - Emu code generator coupled to LLVM front end
 - Inlining of many library functions to leverage Emu instructions
- Added capability to define memory *Views* and place data in those Views
 - Shared data items are *Dynamic Shared Objects (DSOs)* that may be defined outside the programs that manipulate them and persist beyond program completion
 - Private automatic variables are declared normally in Cilk.



The Emu Cilk Language

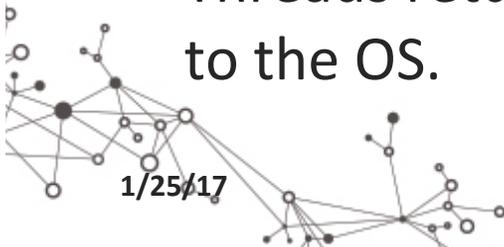
Emu Cilk extends C with a few new keywords

- **Cilk_spawn** [**<var> =**] **<function>** creates new “child” thread running **<function>** while parent thread continues asynchronously
- **Cilk_sync** causes current function to wait until all its children have completed
- **Cilk_for** (**<iterator>**) {**<iteration_code>**} [**grainsize = <grainsize>**] creates a group of new threads (from CilkPlus)
- Termination of a function always performs an implicit sync
- No support (at present) for **INLET**, CilkPlus vector operations, C++
 - Eventually plan to add C++ functionality



System Software

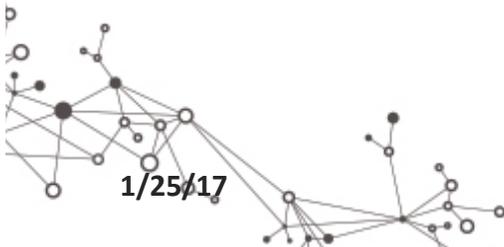
- LINUX runs on the Stationary Cores (SCs).
- OS launches main() user program on a Gossamer Core (GC)
- main() spawns descendants that execute in parallel and migrate throughout system as needed.
- Runtime executes primarily on the SCs and handles service requests from the threads running on the GCs.
 - Memory allocation and release, I/O, exception handling, and performance monitoring.
 - A few special system threads run on Gossamer Cores to provide real-time system management, such as distribution of credits
- Threads return to main() upon completion, which then returns to the OS.



Where does Emu shine?

- **Big Data:** the bigger the better. Data that lacks regular structure works especially well
- **Graph Algorithms:** examples include Non-Obvious Relationships, Particle Swarm and Network Optimization
- **Searching, Sorting, and Pattern Matching Problems:** such as Gene Sequence matching, Protein-Ligand Docking and Machine Learning
- **Sparse Matrix Problems:** examples include the HPCG benchmark and Linear Programming

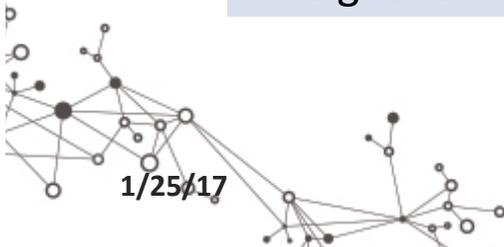
These will all perform far better on Emu than on conventional computers



Market Application Areas:

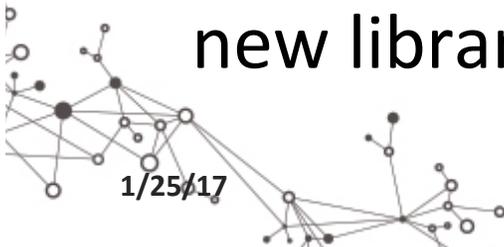
These are areas where the Emu Architecture will shine

Numerical	Search
<ul style="list-style-type: none">• Computational Fluid Dynamics	<ul style="list-style-type: none">• Graph Analysis
<ul style="list-style-type: none">• Reservoir Modeling	<ul style="list-style-type: none">• Network Optimization
<ul style="list-style-type: none">• Weather & Climate Modeling	<ul style="list-style-type: none">• Regression Analysis
<ul style="list-style-type: none">• Linear & Dynamic Programming	<ul style="list-style-type: none">• Risk & Fraud Analysis
<ul style="list-style-type: none">• Quantum Physics	<ul style="list-style-type: none">• Map/Reduce Acceleration
<ul style="list-style-type: none">• Quantum Chemistry	<ul style="list-style-type: none">• Record Management & Statistics
<ul style="list-style-type: none">• Molecular Dynamics	<ul style="list-style-type: none">• Machine Learning
<ul style="list-style-type: none">• Many-body Problems	<ul style="list-style-type: none">• Image Understanding
<ul style="list-style-type: none">• Signal & Image Processing	<ul style="list-style-type: none">• Gene Sequence Analysis

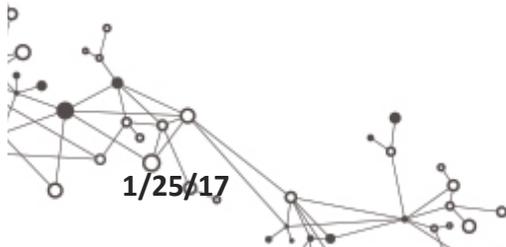


Current Status

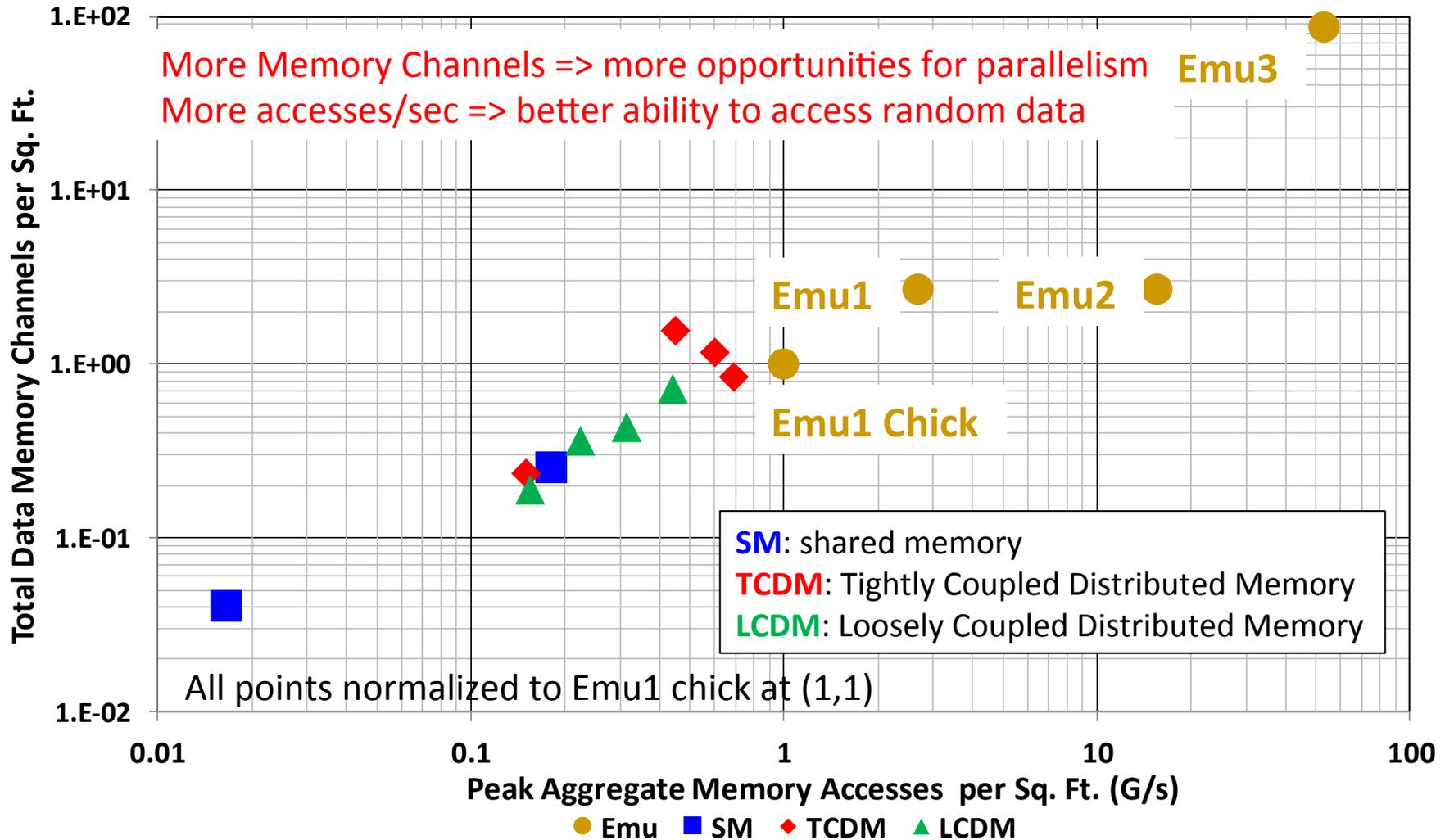
- Will deliver first Emu “Chicks” in Fall 2016
- Design ports directly to rack level & above
- Significantly better scalability than commercial systems on GUPS, BFS, and similar applications
- Ongoing development of libraries for comprehensive programming environment
- Expanding application base
- Interested in research partnerships to explore new library and application areas



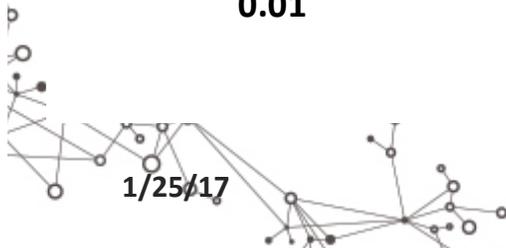
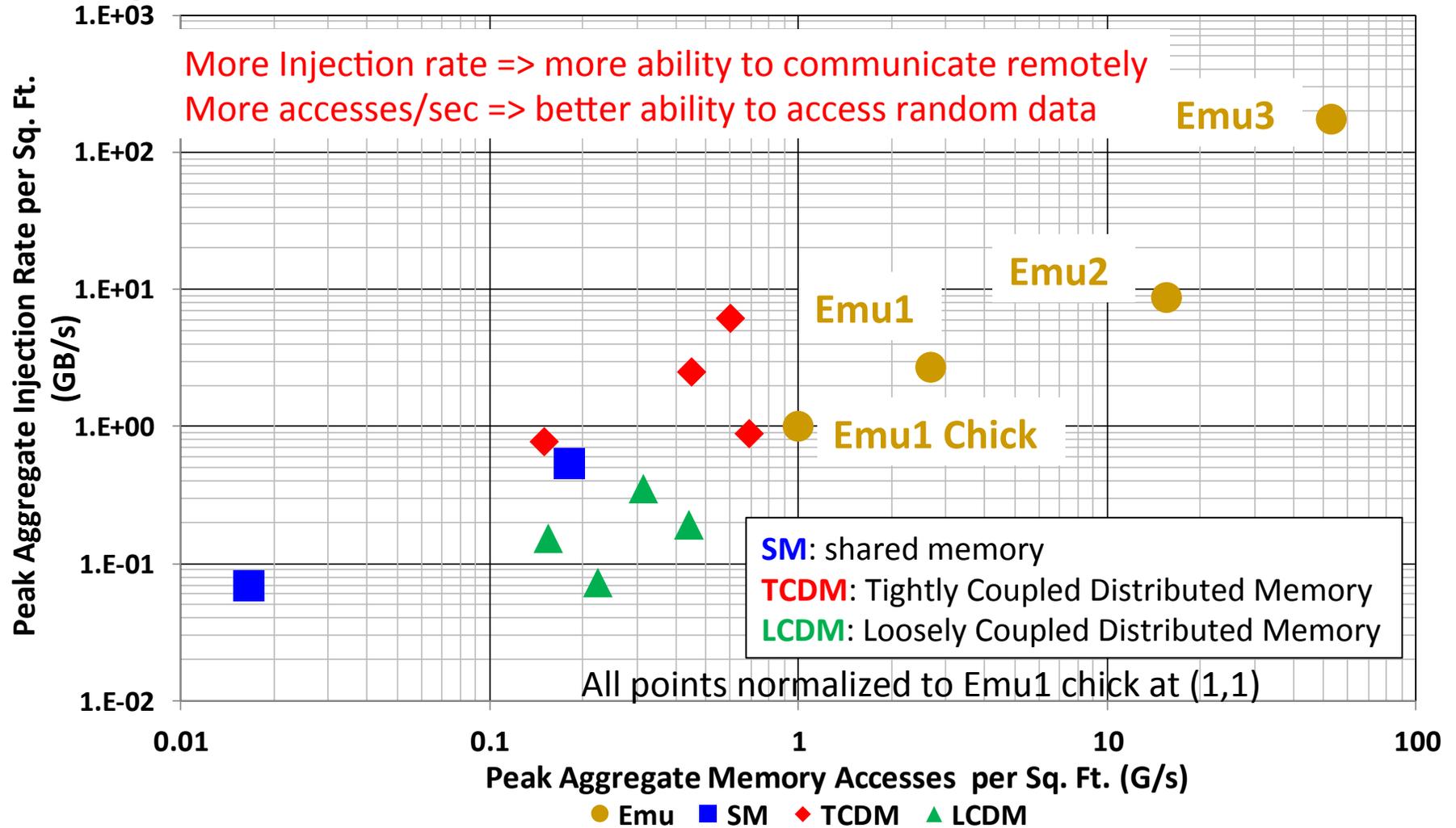
Some Early Comparisons



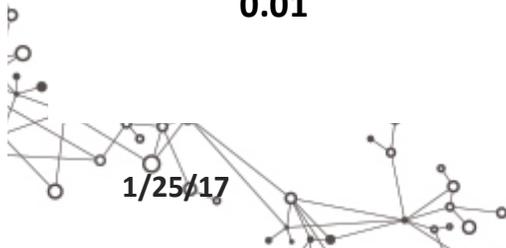
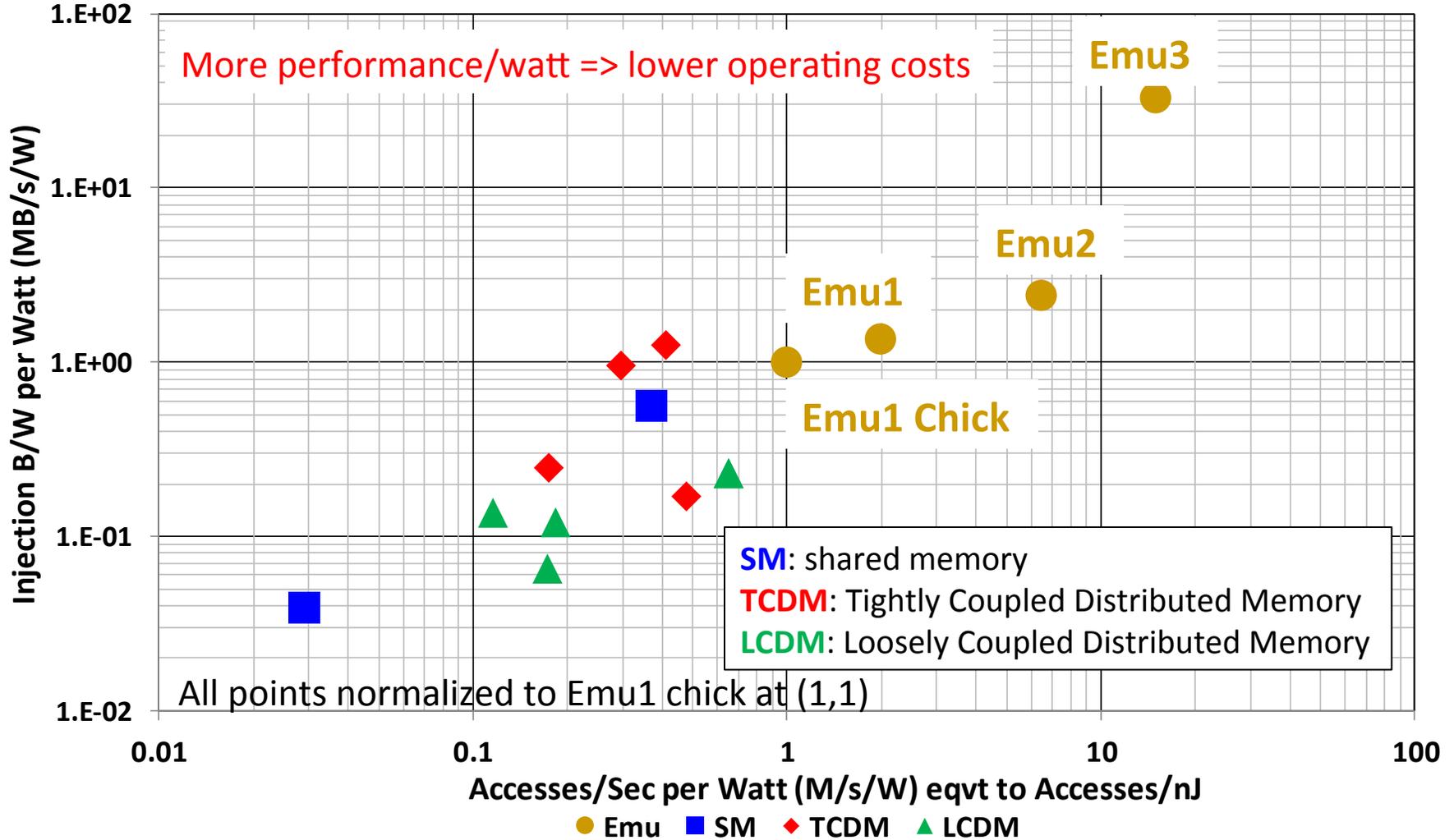
Memory Channel Comparison



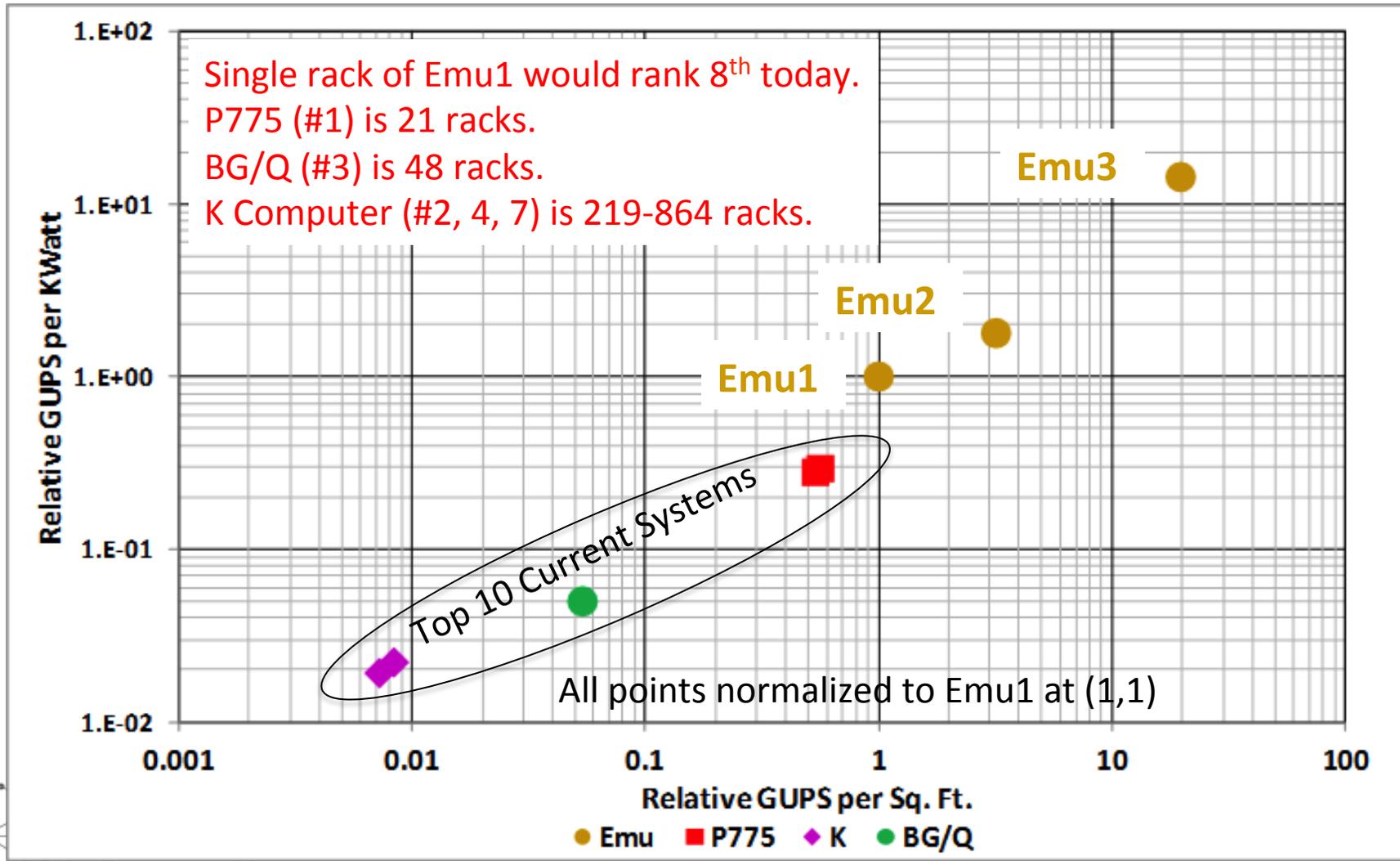
Injection Bandwidth & Access Rate



System Per Watt Comparison

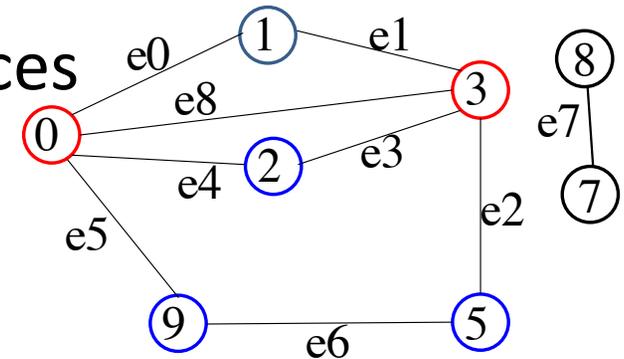


GUPS: Global Updates per Second



Breadth First Search

- Core of GRAPH 500 rankings
- Start with a root, find reachable vertices
- Metric: **TEPS**: Traversed Edges/sec
- Performance issues
 - Massive data sets
 - Very irregular access
 - Very challenging load balancing

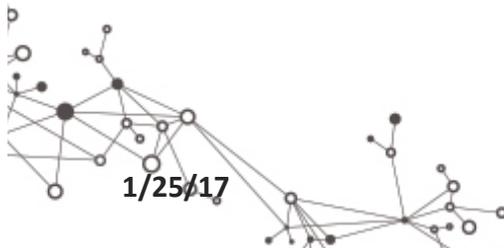


Starting at 1: 1, 0, 3, 2, 9, 5

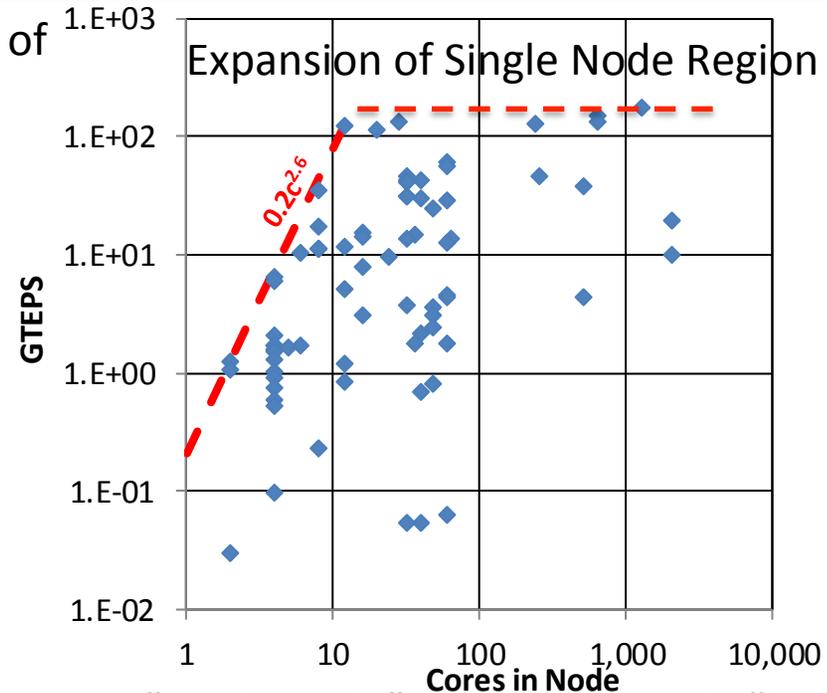
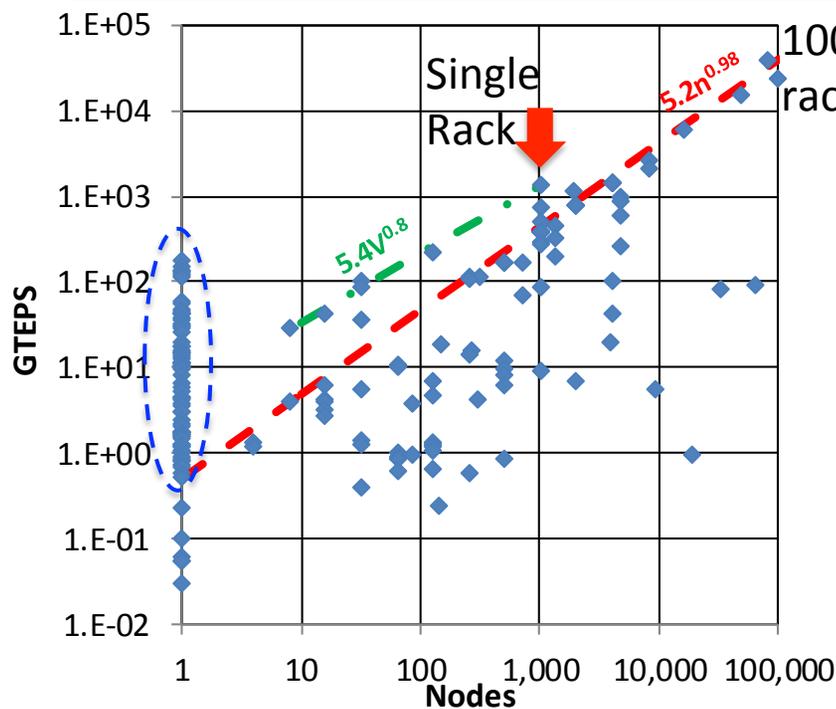
GRAPH500 Graph Sizes

Level	Scale	Size	Vertices (Billion)	TB	Bytes /Vertex
10	26	Toy	0.1	0.02	281.8048
11	29	Mini	0.5	0.14	281.3952
12	32	Small	4.3	1.1	281.472
13	36	Medium	68.7	17.6	281.4752
14	39	Large	549.8	141	281.475
15	42	Huge	4398.0	1,126	281.475
				Average	281.5162

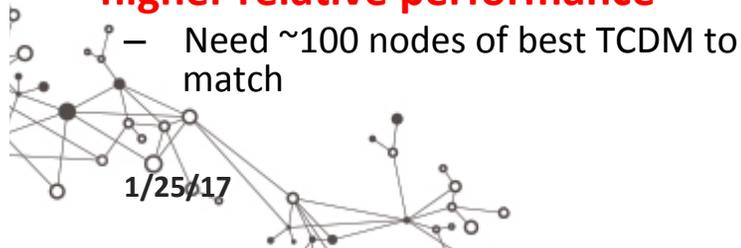
$$\text{Scale} = \log_2(\# \text{ vertices})$$



Most Recent GRAPH500 Data



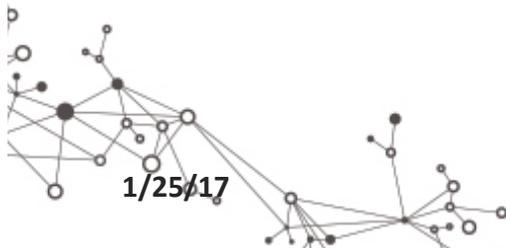
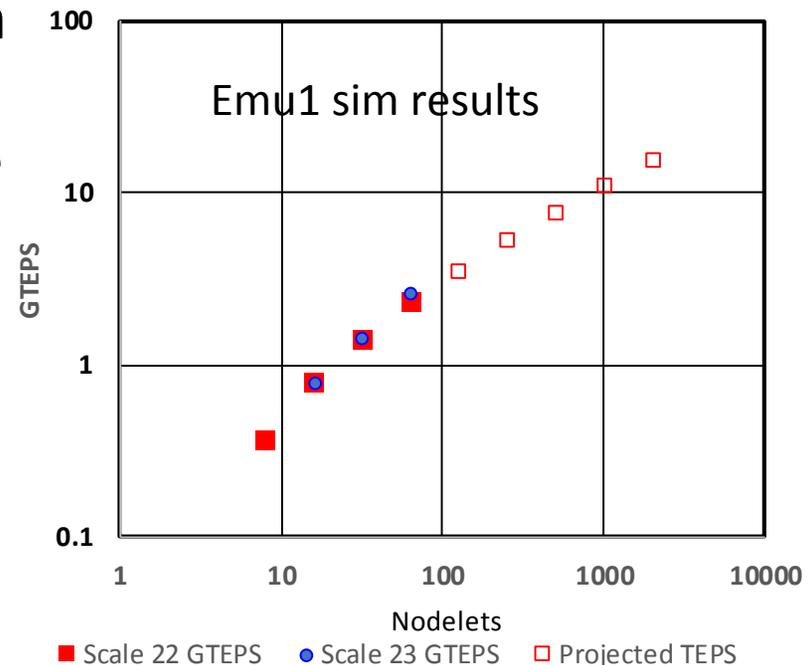
- Good scalability for multi-node **TCDM** systems
- **But single node SM systems far higher relative performance**



- “Single node” systems actually “**Shared Memory**” multi-core systems
- But performance **flat** after ~ 10 cores
- Today’s Shared Memory systems inherently more efficient but don’t scale
- Extra “cores” add memory, not performance

Early Projections for Emu

- Emu *is* a shared memory system
- But with **FAR better** scaling than today's SM systems
- And this is with *very small* (scale 22) problems
- More realistic sizes (scale 30 for chick) could increase GTEPS by almost 10X
 - Initial scale 23 results point in this direction
- More complete simulations in progress



Stay Tuned!

Real World Commercial Challenge Problem (From Lexis Nexis)

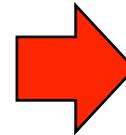
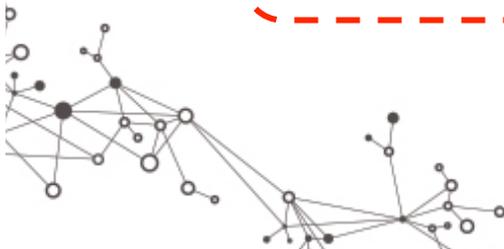
Auto Insurance Co: "Tell me about giving auto policy to Jane Doe" in < 0.1sec

- 40+ TB of Raw Data
- Periodically clean up & combine to 4-7 TB
- Weekly "Boil the Ocean" to precompute answers to all standard queries

- Does X have financial difficulties?
- Does X have legal problems?
- Has X had significant driving problems?

- Who has shared addresses with Z?
- Who has shared property ownership with Z?

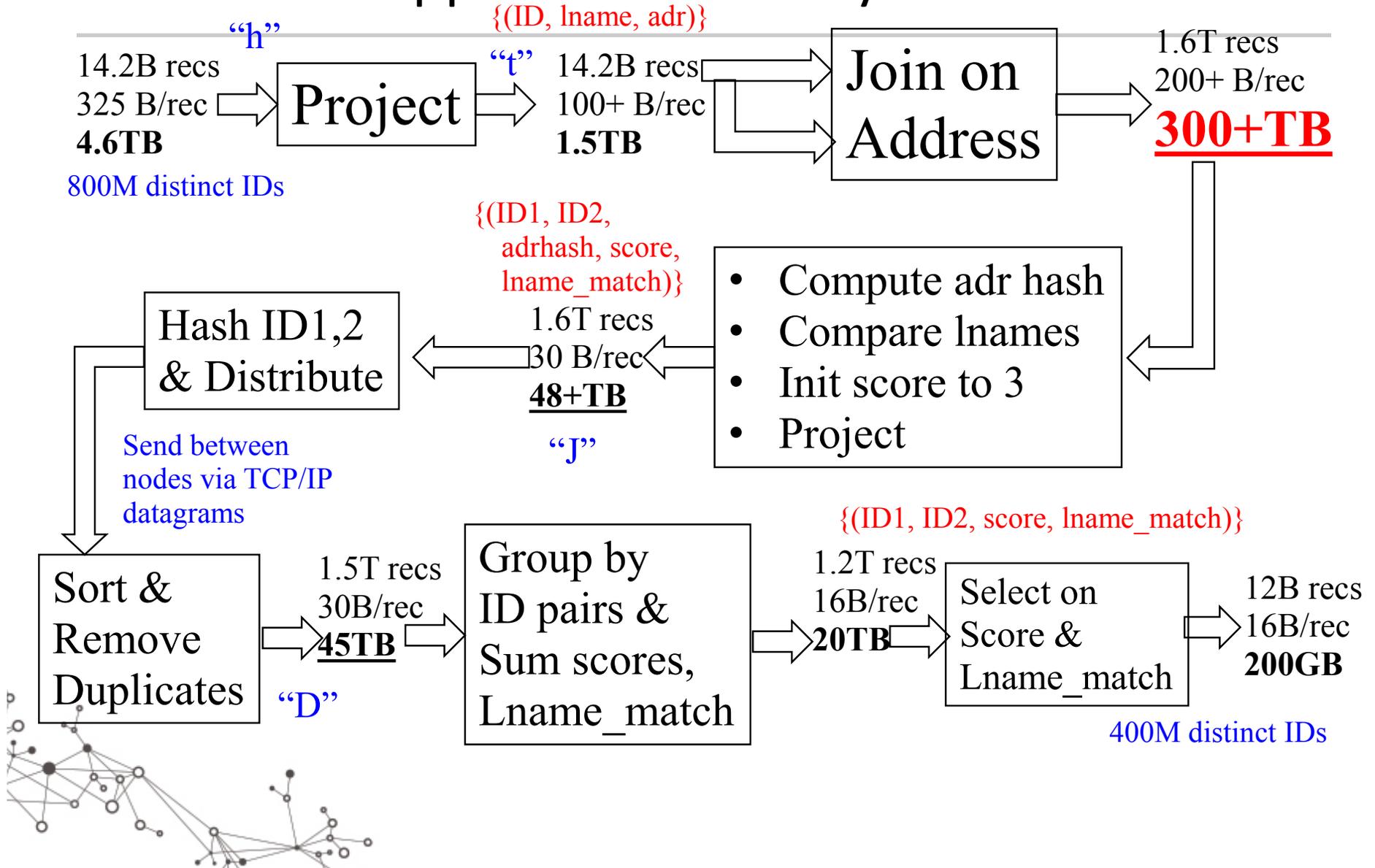
Relationships



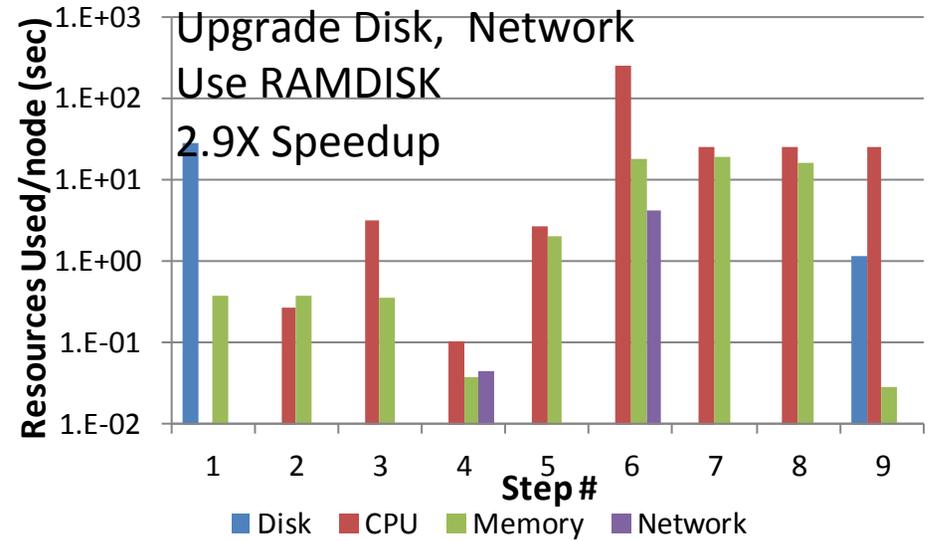
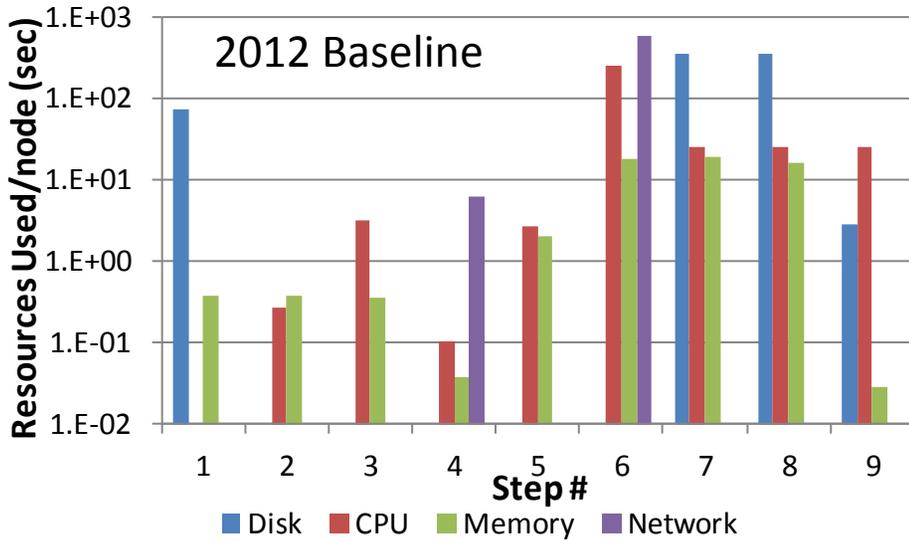
Look up answers to precomputed queries for "Jane Doe" and combine

"Jane Doe has no indicators
But
she has shared multiple addresses with Joe Scofflaw
Who has the following negative indicators"

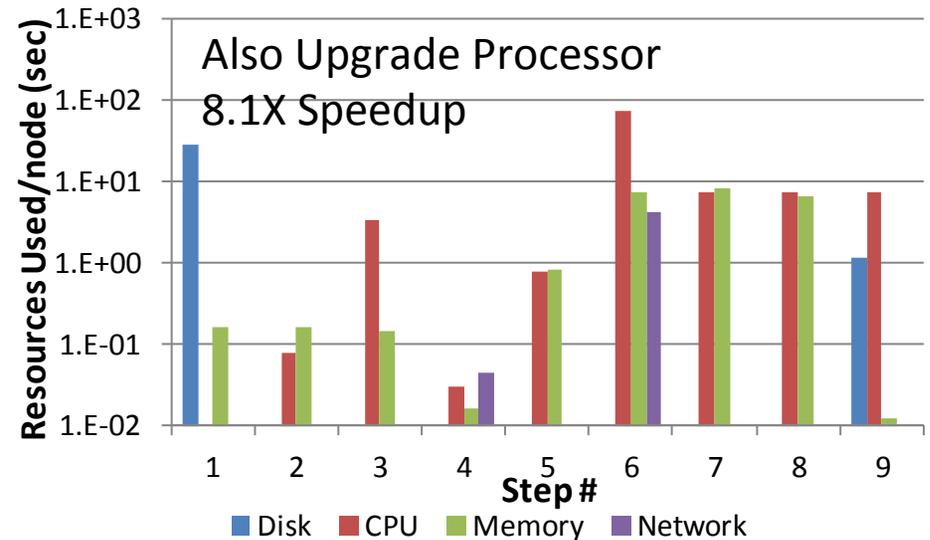
Traditional Approach: Runaway Intermediate Data



Heavyweight Upgrades

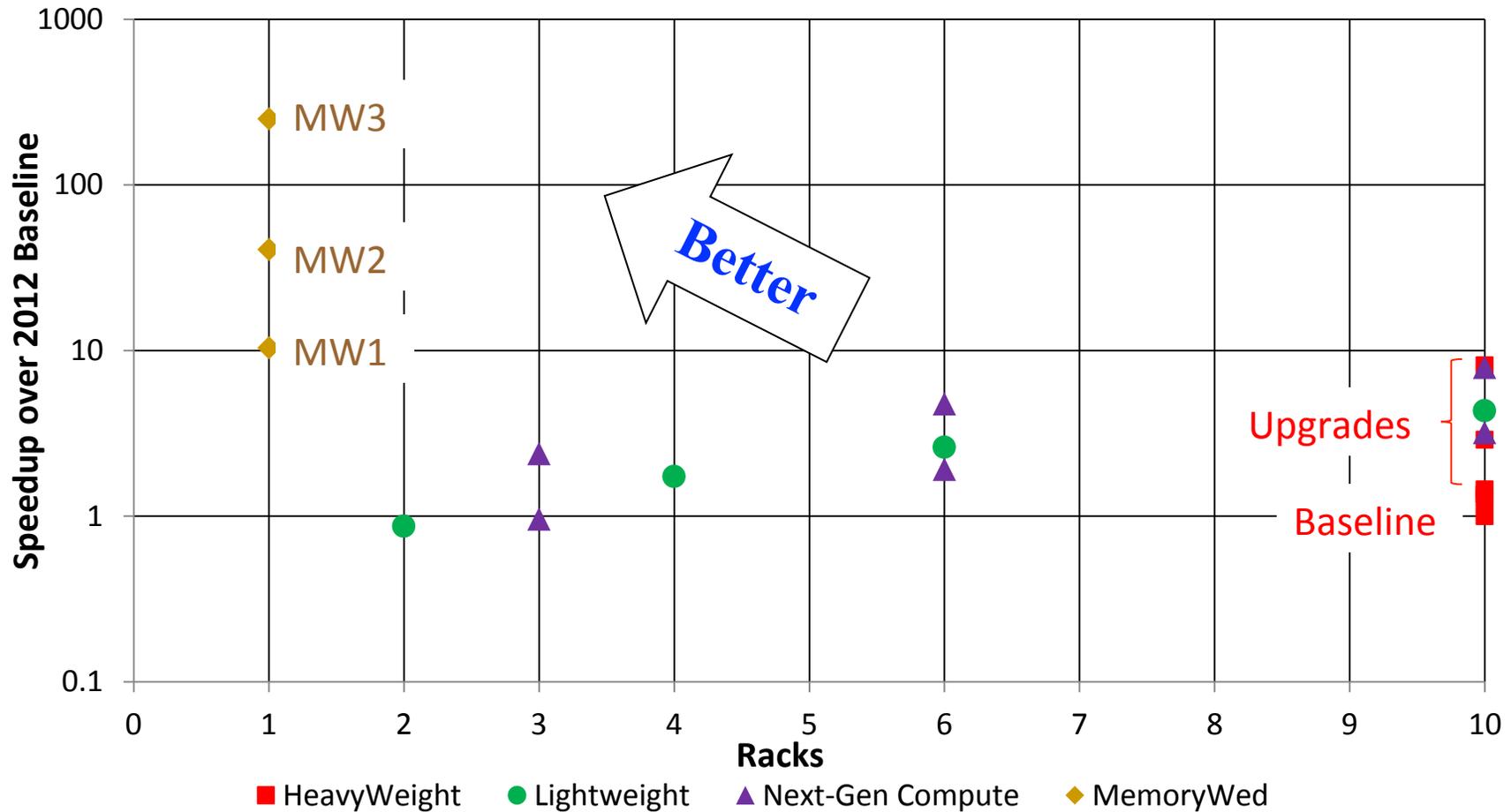


Enhancements				Time per Step									Total Time	Speedup
2x IB FDR	SATA SSD	RAMDISK	22 Cores, 4xDDR4	1	2	3	4	5	6	7	8	9		
2012 Baseline				75.2	75.2	75.2	75.2	75.2	580.5	580.5	360.0	362.9	1025.8	1.00
X				75.2	75.2	75.2	75.2	75.2	250.2	360.0	360.0	362.9	805.3	1.27
	X			28.8	28.8	28.8	28.8	28.8	580.5	580.5	144.0	145.2	757.4	1.35
		X		75.2	75.2	75.2	75.2	75.2	580.5	580.5	25.0	50.0	707.7	1.45
			X	75.2	75.2	75.2	75.2	75.2	580.5	580.5	360.0	362.9	1025.8	1.00
X	X	X		28.8	28.8	28.8	28.8	28.8	250.2	275.2	25.0	50.0	355.1	2.89
X	X	X	X	28.8	28.8	28.8	28.8	28.8	74.4	81.9	7.4	14.9	126.3	8.12



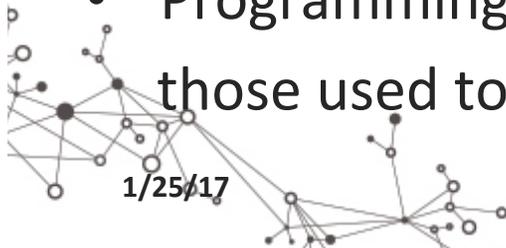
All are still 10 Rack Systems

Comparison Based on Analytic Model



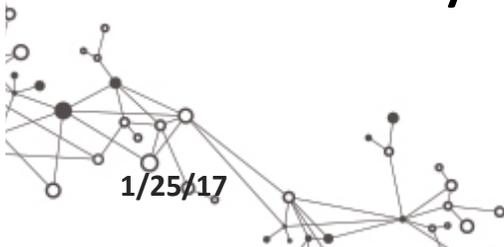
Summary

- Supports enormous problems in-memory
- Supports high-rate real-time updates to the database
- Execute cache-breaker apps far more effectively than today
- Makes better use of system bandwidth than conventional computers, resulting in higher performance and lower energy utilization
- Offers unprecedented levels of scalable parallelism in a structure that can be understood and exploited
- Programming environment is effective and reasonably familiar to those used to standard systems



Relevance to Space

- Growth in sparse & irregular in on board apps
- Natural integration into 3D
- Lower energy per op
- Spawns can be extended to interface with specialized accelerator cores
- Cilk provides simpler parallel programming
- Memory ops can be architected to include RAID-like redundancy
- Address-based migrations enable simple failure recovery



Thank You

Emu

