SOC Design for HPC: Technology Analysis & Requirements

Peter M. Kogge

McCourtney Prof. of CS & Engr.

University of Notre Dame

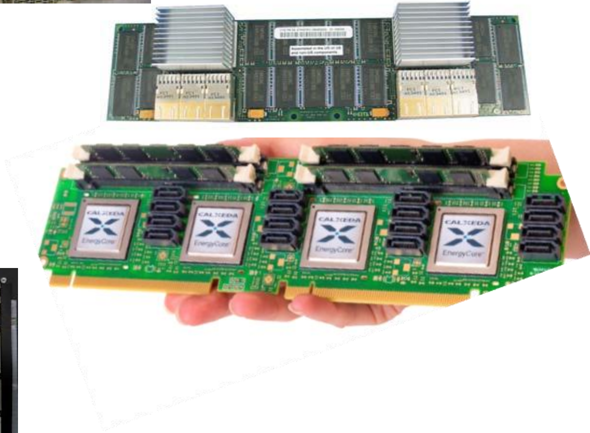
Acknowledgement: This work was funded in part by the US Dept. of Energy, Sandia National Labs, as part of their Xcaliber and XGC projects.

Thesis

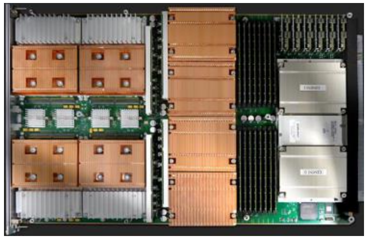
- Today’s COTS design typically “inward” focus
- For HPC, “outward” is far more crucial
  - Memory, esp. random access
  - Off-chip bandwidth
- This talk
  - Take-aways from TOP500
  - Take-aways from a Big Data problem
  - Energy discussion
- The biggest gains seem to come from rethinking system architecture
- SOC, if done right, seems to be right direction

Today's Architecture Classes

- **Heavyweight**: traditional 100+W multi-core

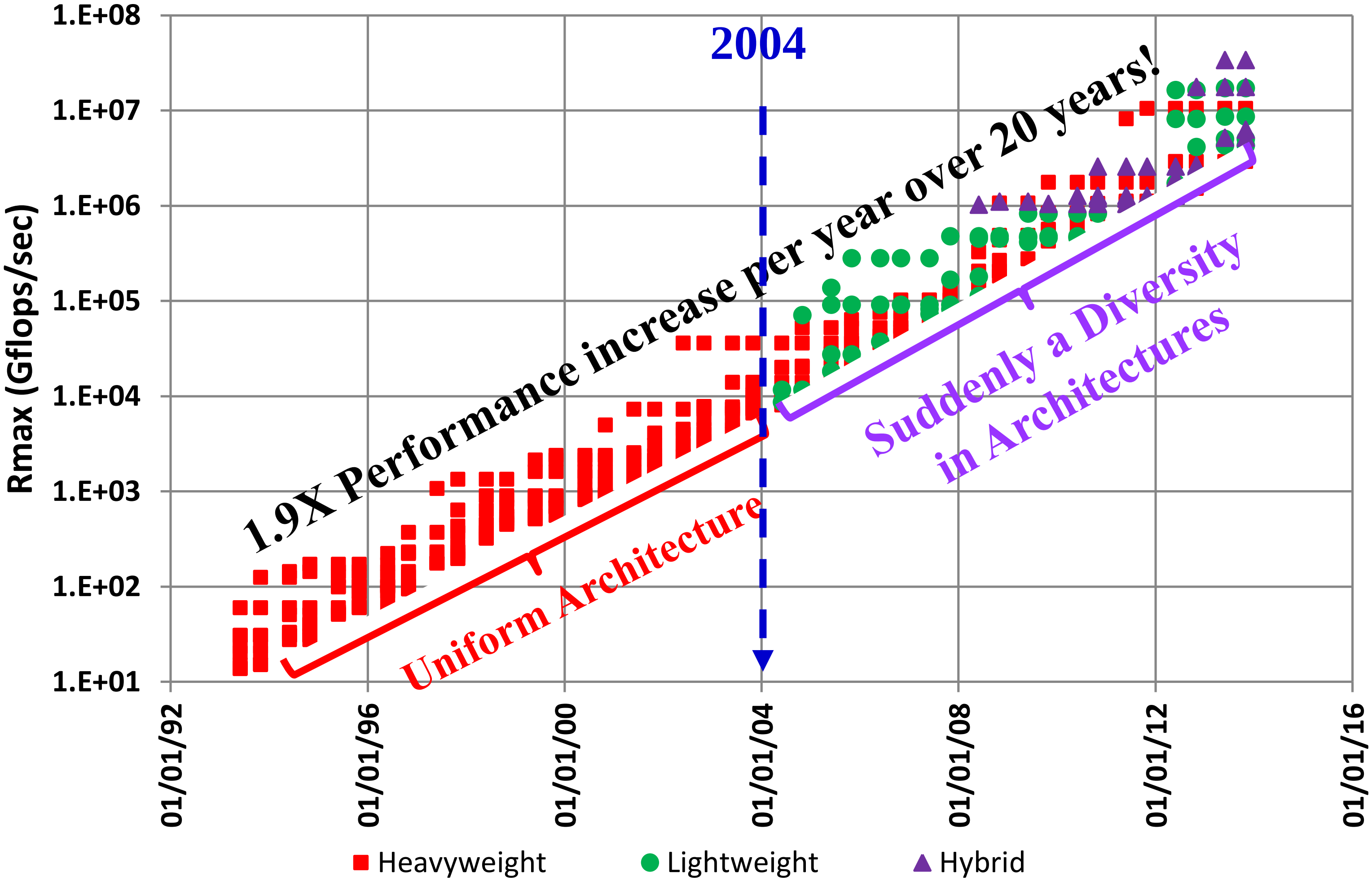


- **Lightweight**: lower power single chip system

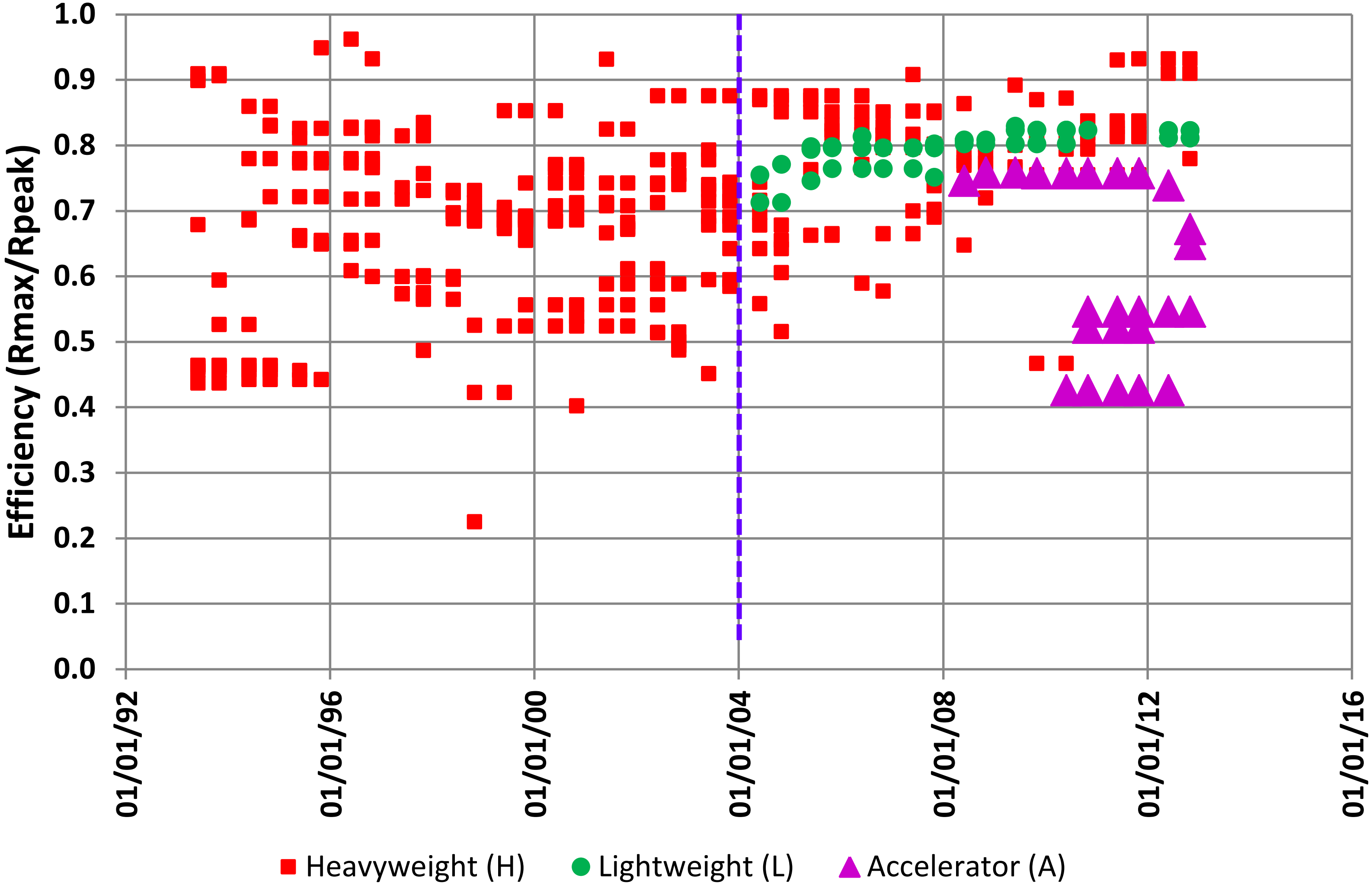


- **Hybrid/Heterogeneous**: Heavyweight/GPU combination
- **Big/Little**: Same ISA, different microarchitectures
- **Other**: XMT, Convey

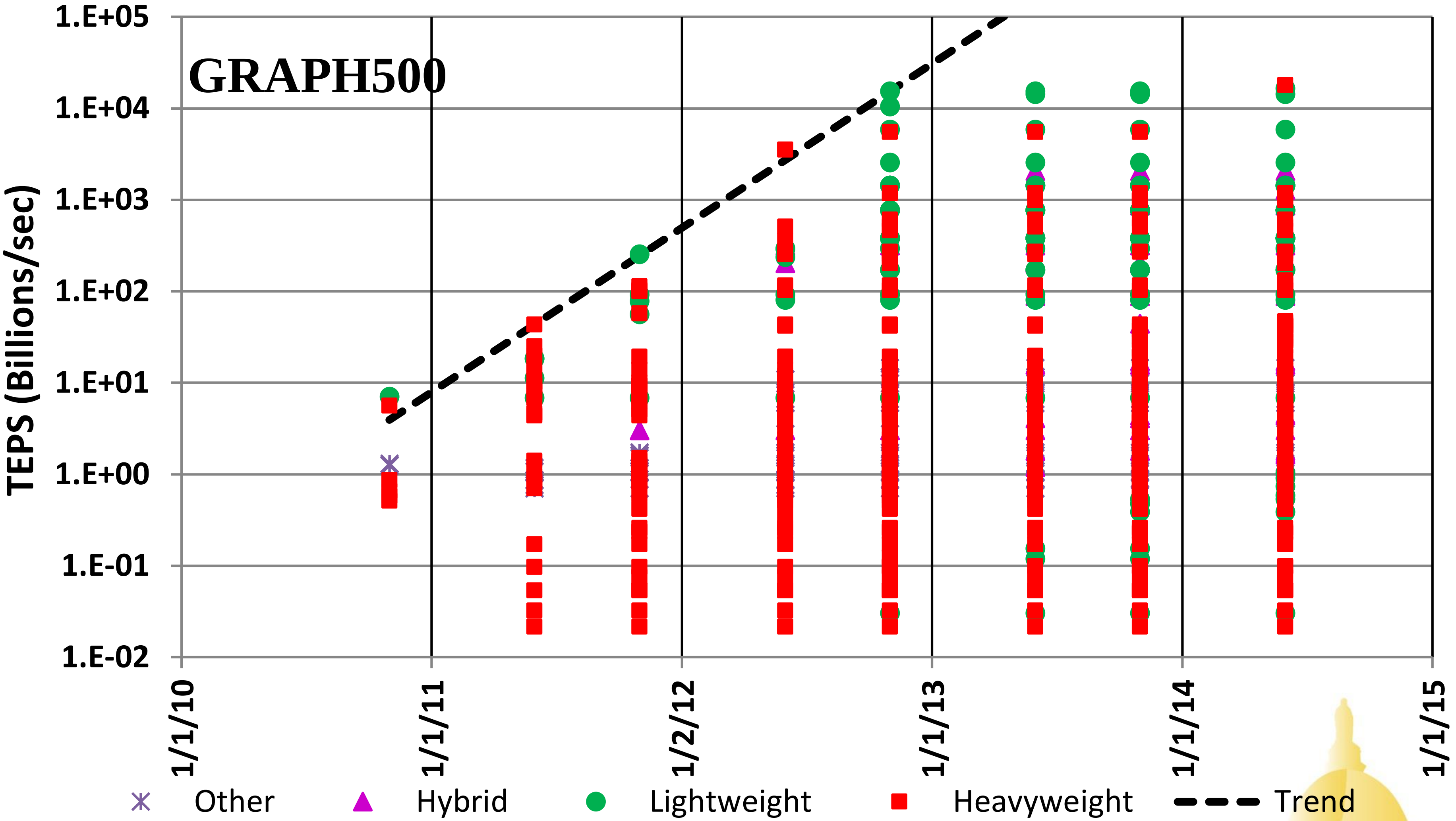
We All Know The Story: Unbroken Growth in TOP500 Rmax



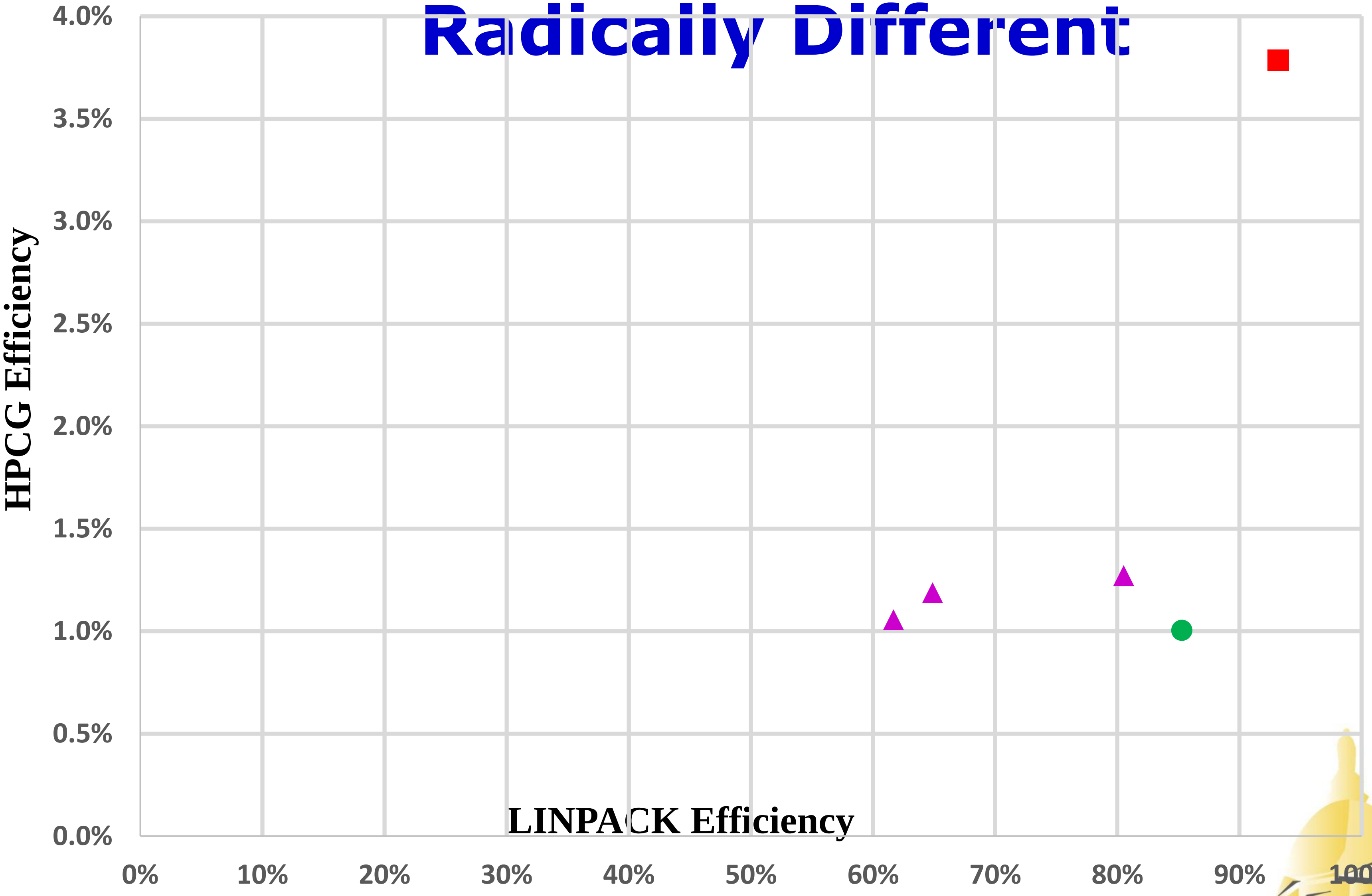
Floating Point Efficiency Remains High for Linpack



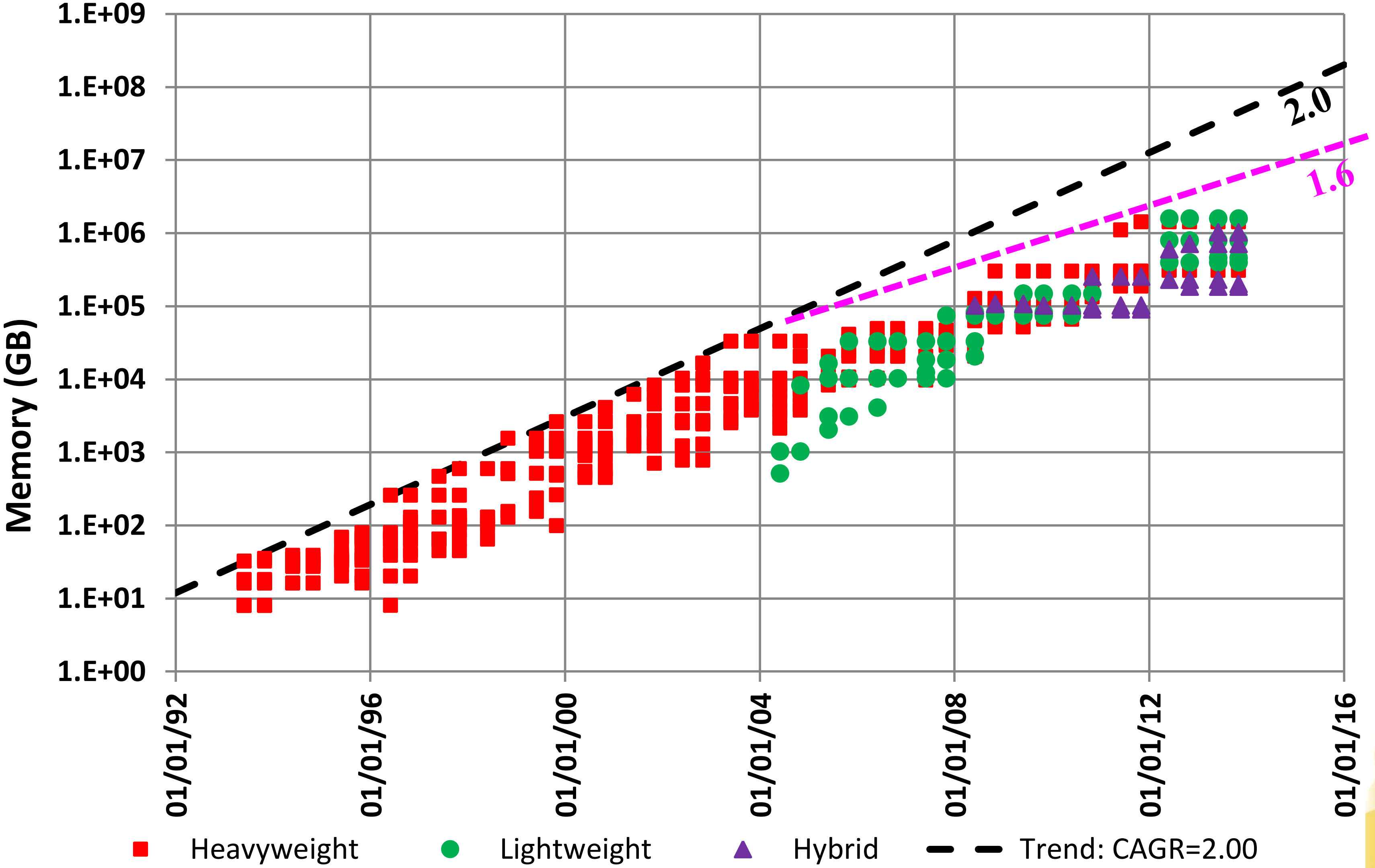
But Not All Benchmarks Double/Year



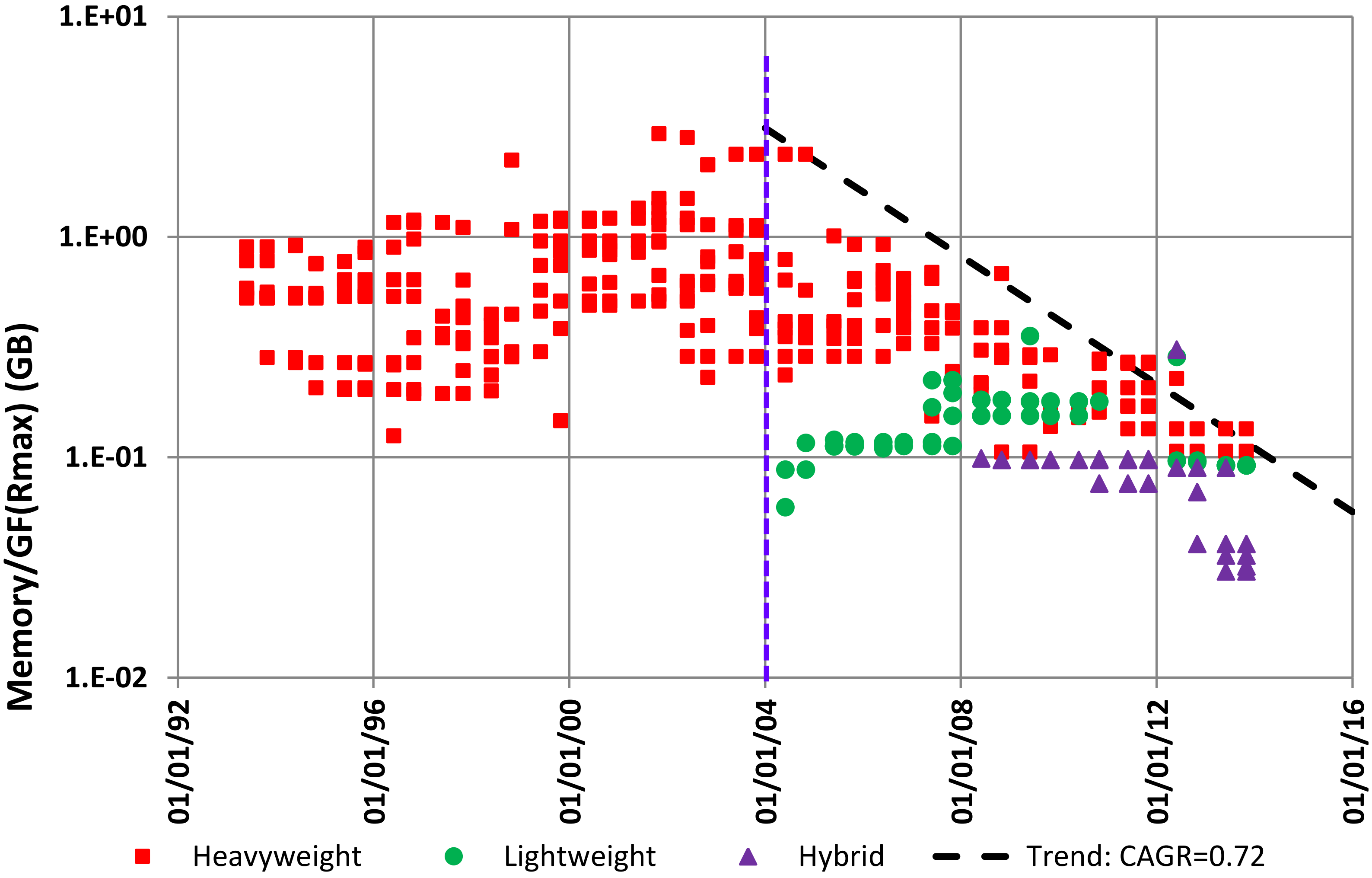
Even Newer Scientific Codes Are Radically Different



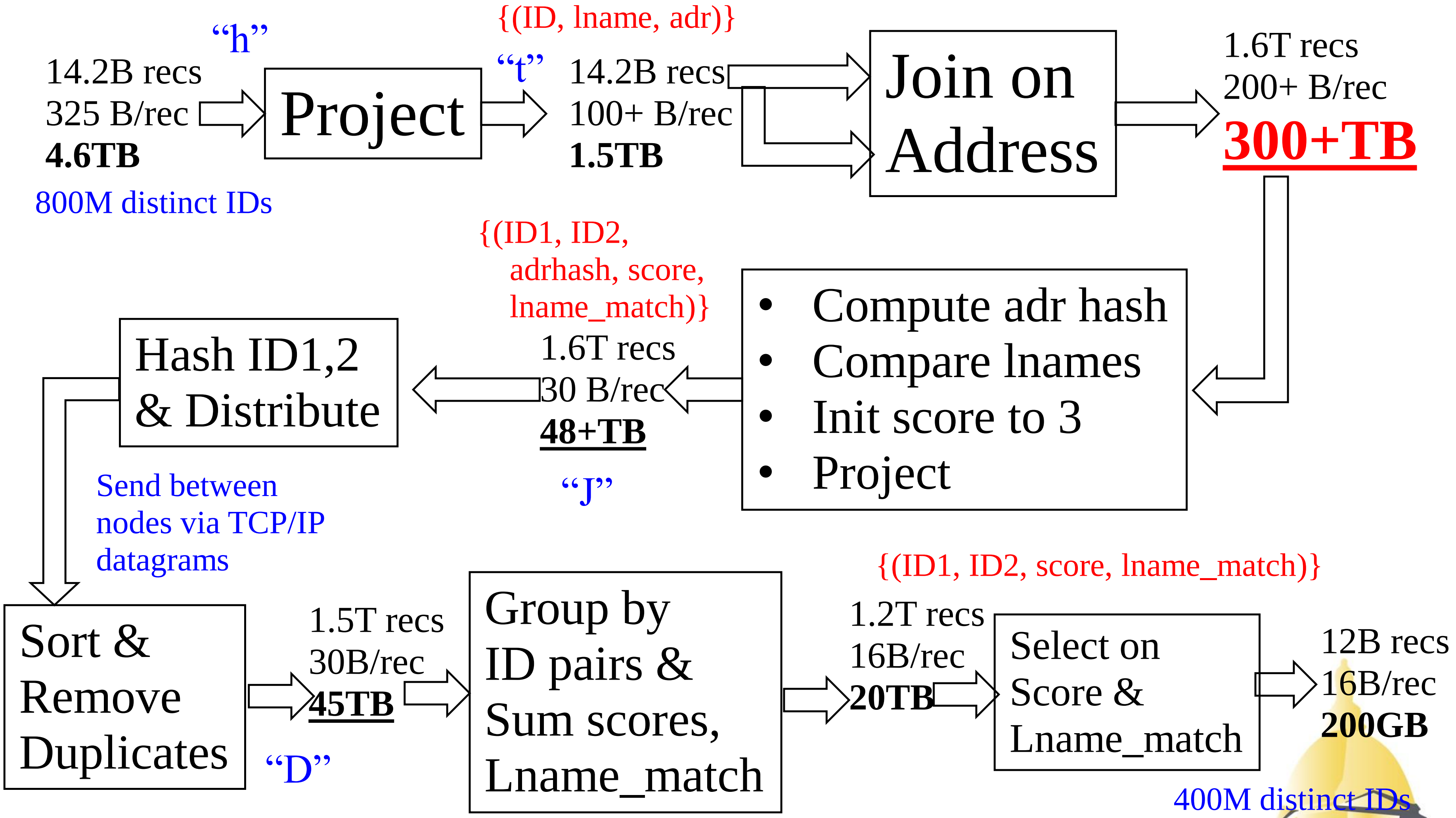
Memory Growth Has Slowed



And Memory per Flop/s Is Dropping!



A Real-World Big Data Problem



14.2B recs
325 B/rec
4.6TB

800M distinct IDs

"h"

Project

"t"

{(ID, lname, adr)}

14.2B recs
100+ B/rec
1.5TB

Join on Address

1.6T recs
200+ B/rec
300+TB

- Compute adr hash
- Compare lnames
- Init score to 3
- Project

{(ID1, ID2, adrhash, score, lname_match)}

1.6T recs
30 B/rec
48+TB

"J"

Hash ID1,2 & Distribute

Send between nodes via TCP/IP datagrams

Sort & Remove Duplicates

"D"

1.5T recs
30B/rec
45TB

Group by ID pairs & Sum scores, Lname_match

{(ID1, ID2, score, lname_match)}

1.2T recs
16B/rec
20TB

Select on Score & Lname_match

12B recs
16B/rec
200GB

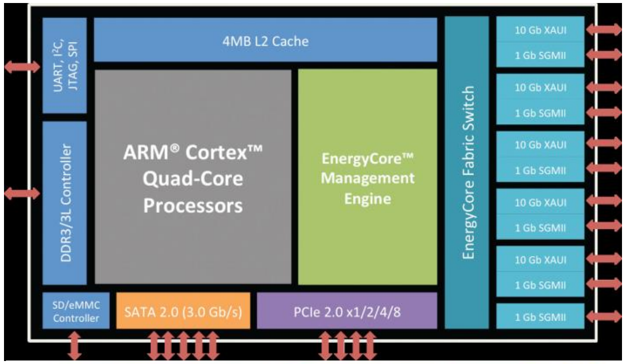
400M distinct IDs

Configurations

- Baseline: Lexis Nexis HPCC Configuration
  - 100 4-node Blades in 10 racks
- Memory Rich Configuration
  - Same as above but with maxed DRAM for RAM Disk
- 2015 Configuration
  - 4X cores/socket, DRAM, switched Infiniband
- 2015 Configuration with DRAM for RAM Disk
- **Lightweight Configurations**
  - **2 racks of Calxeda-like ARM-based SOCs**
- **Xcaliber: Memory Stack-Based**
- **Xcaliber with all computing at bottom**

Possible “Lightweight” System

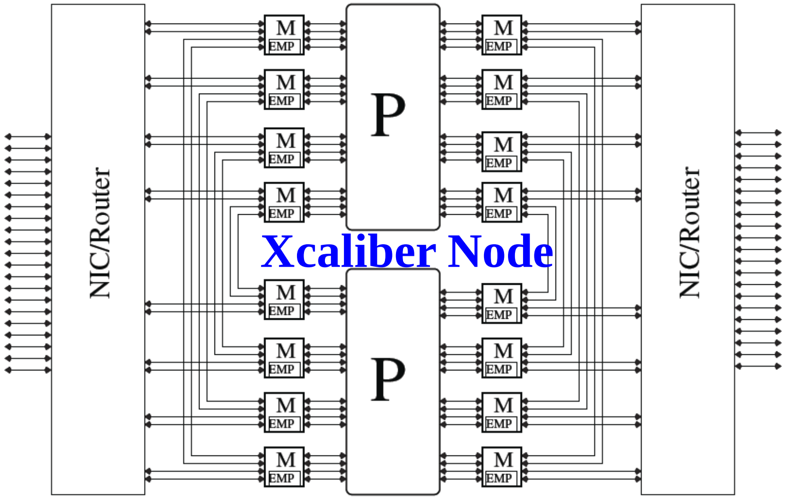
- Assume Calxeda System on a Chip
  - 4 1-1.4GHz ARM A9 cores w’FPU
  - Single DDR3 2 rank controller
  - Networking: GigE, XAU
  - Supports up to 5 SATA
  - Fabric: 8x8 crossbar, 10Gbps links
    - 3 internal, 5 external
- Calxeda Reference card:
  - 4 SOCs + 4 VLP DDR3 DIMMs (max 4GB each)
  - 4 SATA sockets/SOC for disk connections
  - 8 interfaces for off-card fabric
- 2U Blade (based on Boston Viridis Chassis)
  - 12 reference cards + up to 24 SATA
- Assumed Configuration of 40 blades, 2 racks

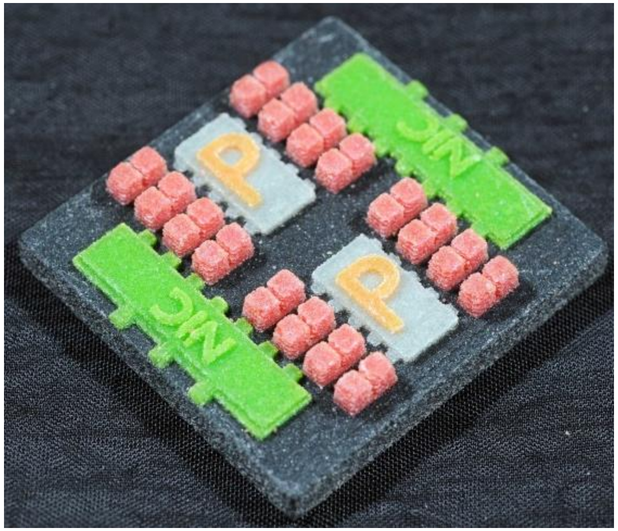


Images from www.calxeda.com 6/2/12

http://www.boston.co.uk/solutions/viridis/viridis-2u.aspx/

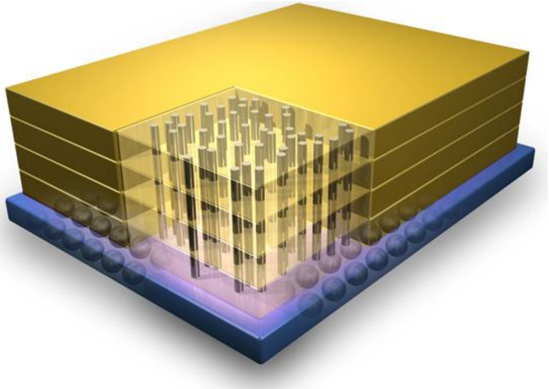
X-Caliber-like Architecture





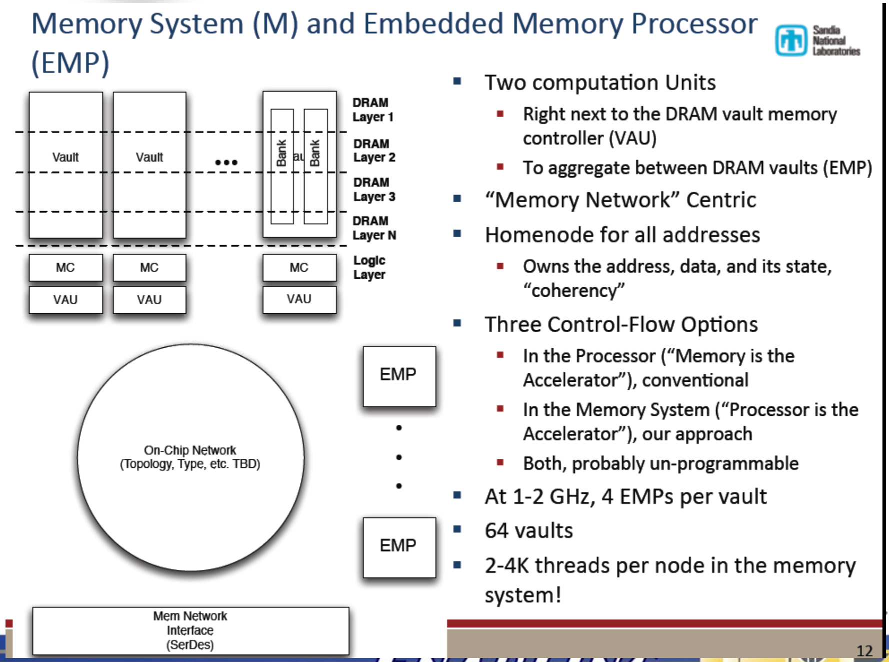
(b) X-caliber Node Mockup

M's built from 3D stacks of memory



Each Stack

- 32 GB DRAM
- 256GB PCM
- Logic chip at bottom
- 64 0.5GB "Vaults"
- 8 full-duplex links
  - 32 GB/s each dir

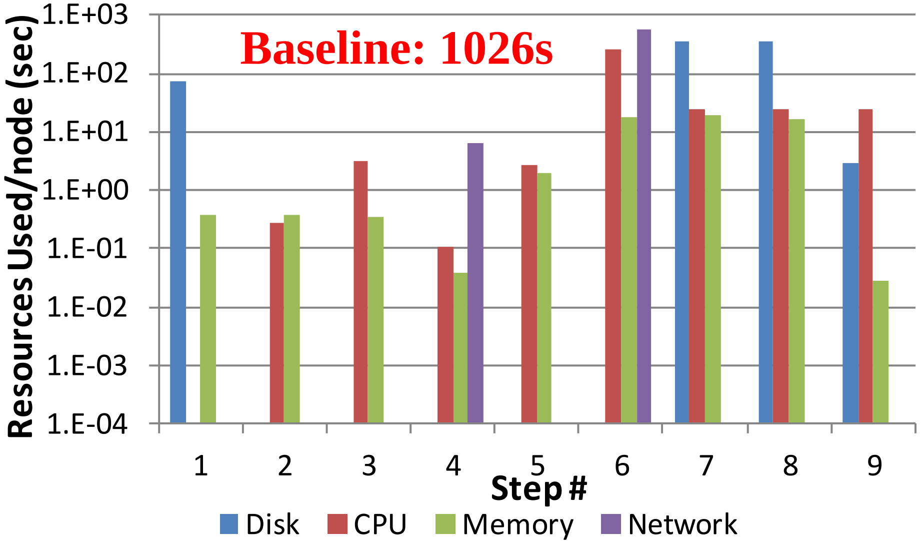


Memory System (M) and Embedded Memory Processor (EMP)

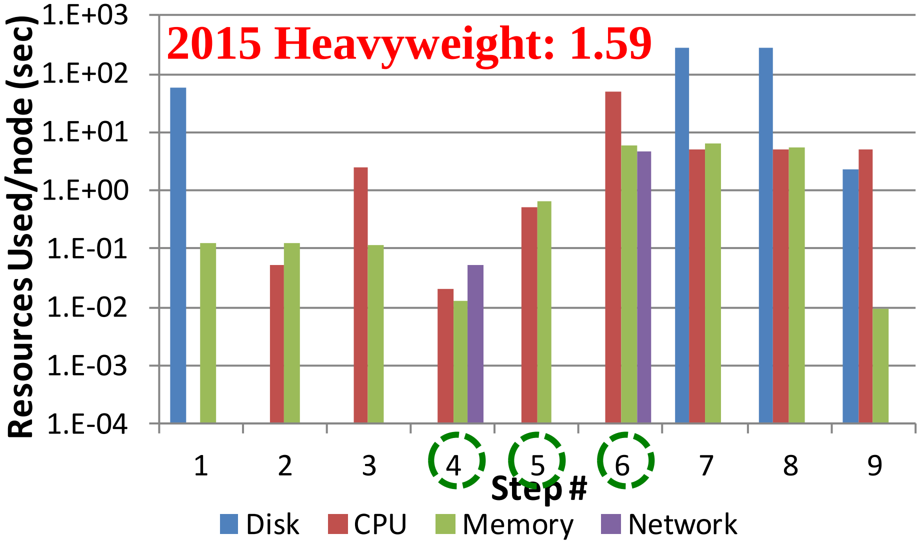
Sandia National Laboratories
Vault
Vault
•••
Bank
au
Bank
DRAM Layer 1
DRAM Layer 2
DRAM Layer 3
DRAM Layer N
Logic Layer
MC
MC
MC
VAU
VAU
VAU
On-Chip Network (Topology, Type, etc. TBD)
EMP
EMP
Mem Network Interface (SerDes)
12

- Two computation Units
  - Right next to the DRAM vault memory controller (VAU)
  - To aggregate between DRAM vaults (EMP)
- "Memory Network" Centric
- Homenode for all addresses
  - Owns the address, data, and its state, "coherency"
- Three Control-Flow Options
  - In the Processor ("Memory is the Accelerator"), conventional
  - In the Memory System ("Processor is the Accelerator"), our approach
  - Both, probably un-programmable
- At 1-2 GHz, 4 EMPs per vault
- 64 vaults
- 2-4K threads per node in the memory system!

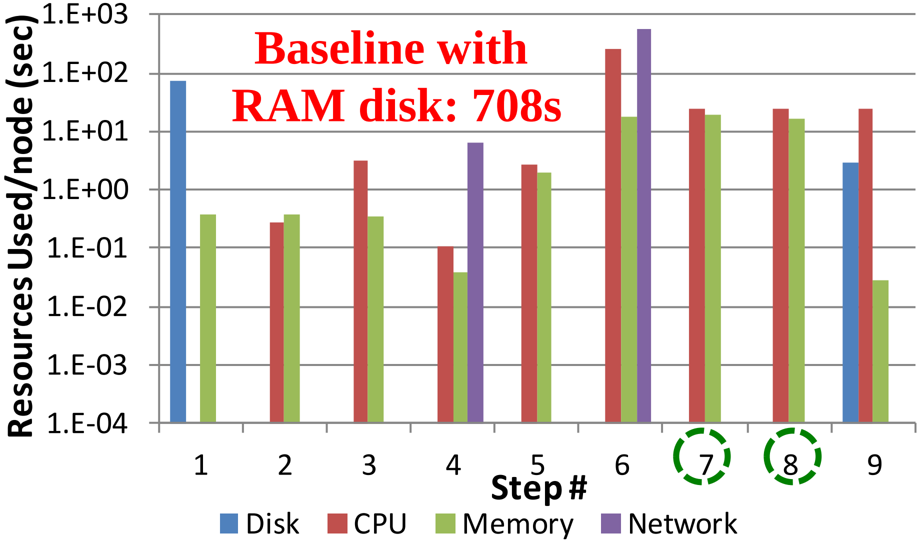
Details: Heavyweight Alternatives



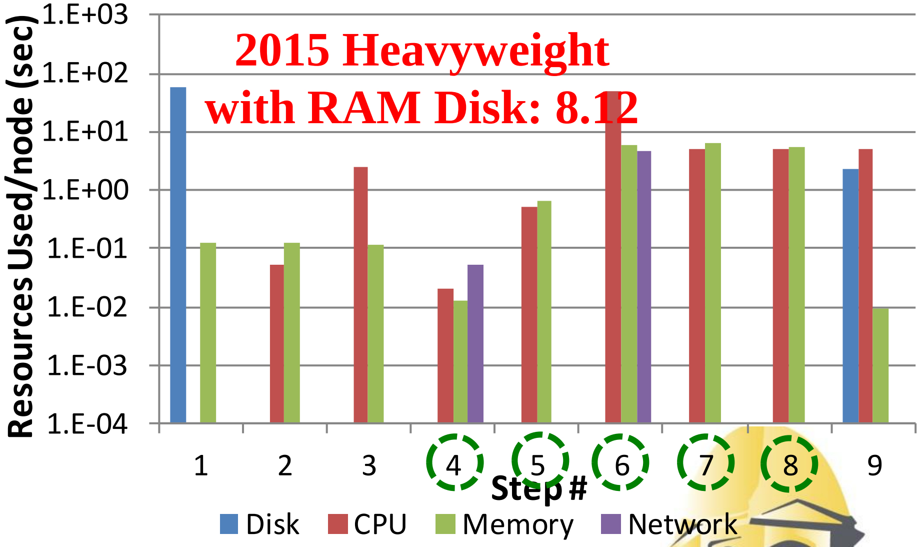
Baseline: 1026s



2015 Heavyweight: 1.59

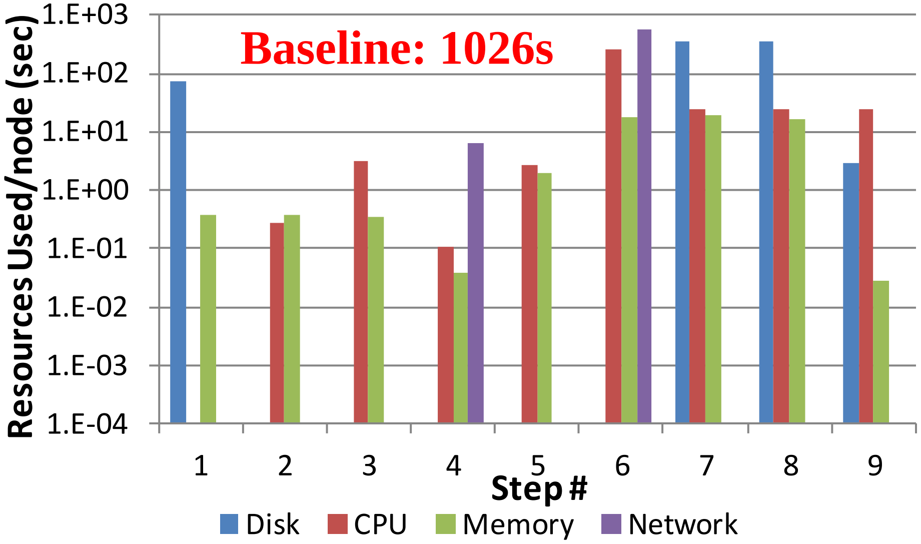


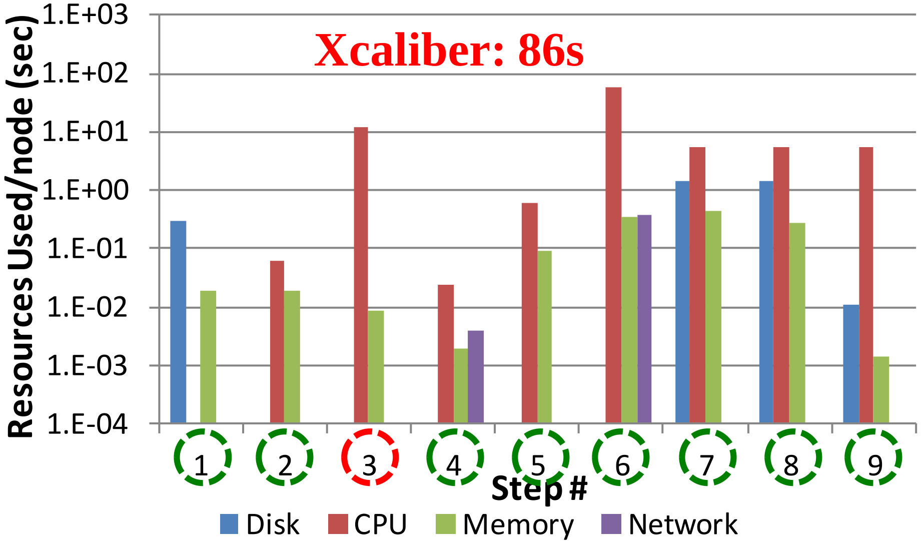
Baseline with RAM disk: 708s

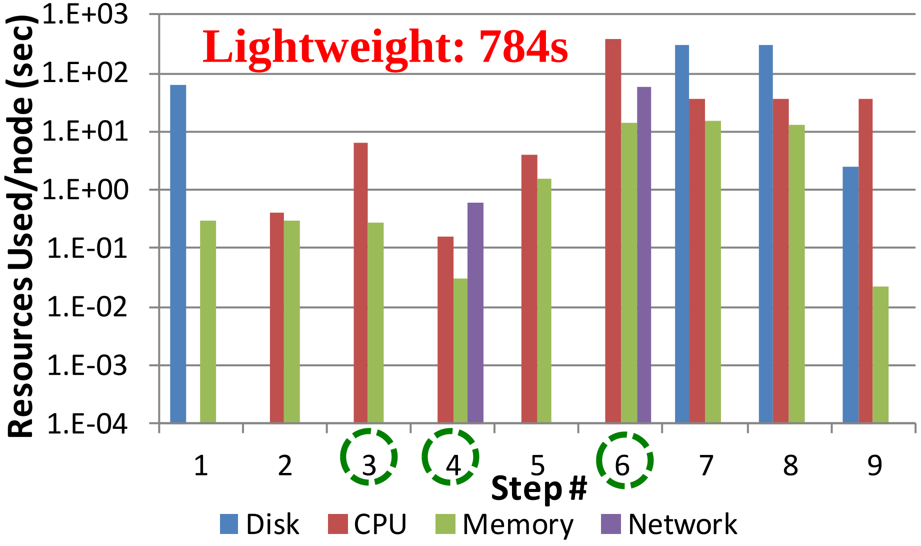


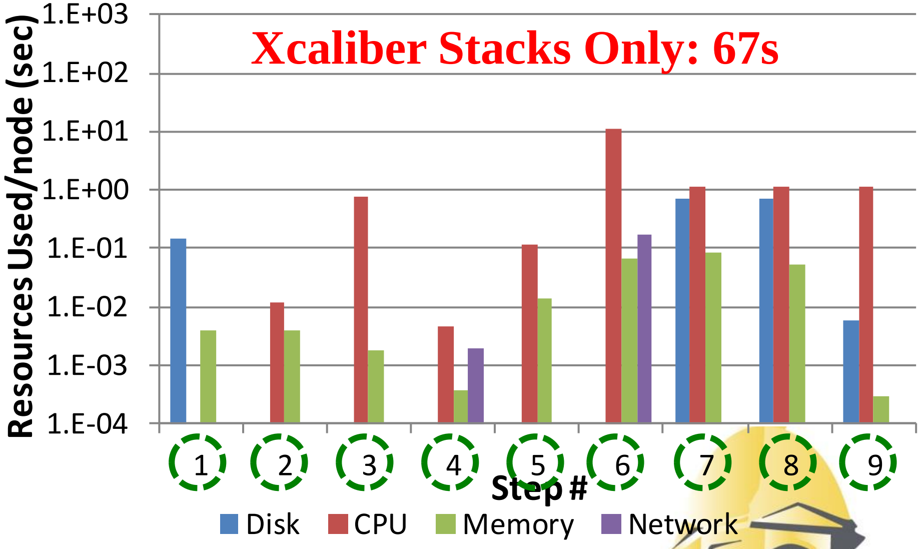
2015 Heavyweight with RAM Disk: 8.12

Non-Heavyweights

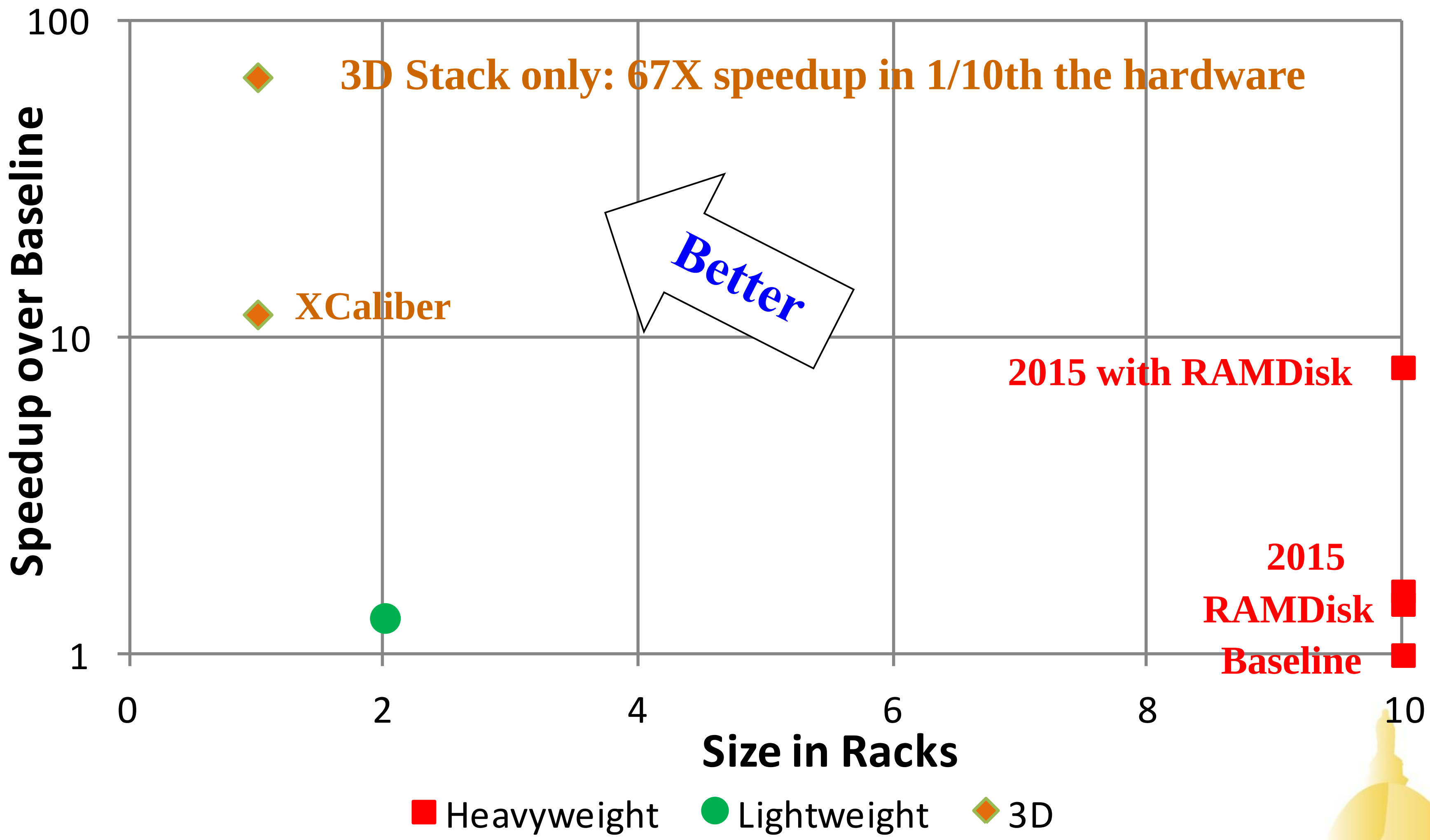




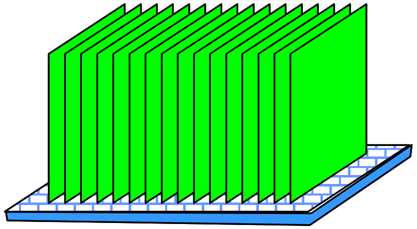


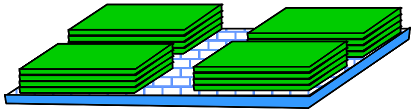


Comparison

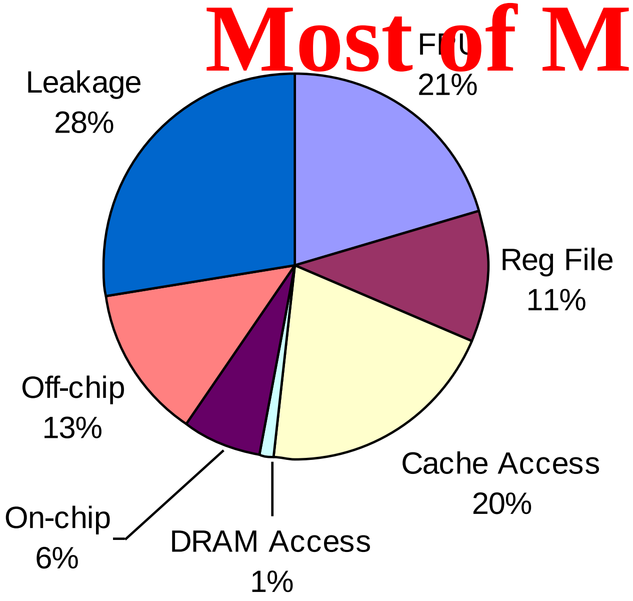

100
10
1
0
2
4
6
8
10
Speedup over Baseline
Size in Racks
3D Stack only: 67X speedup in 1/10th the hardware
XCaliber
Better
2015 with RAMDisk
2015
RAMDisk
Baseline
🟥 Heavyweight
● Lightweight
◆ 3D

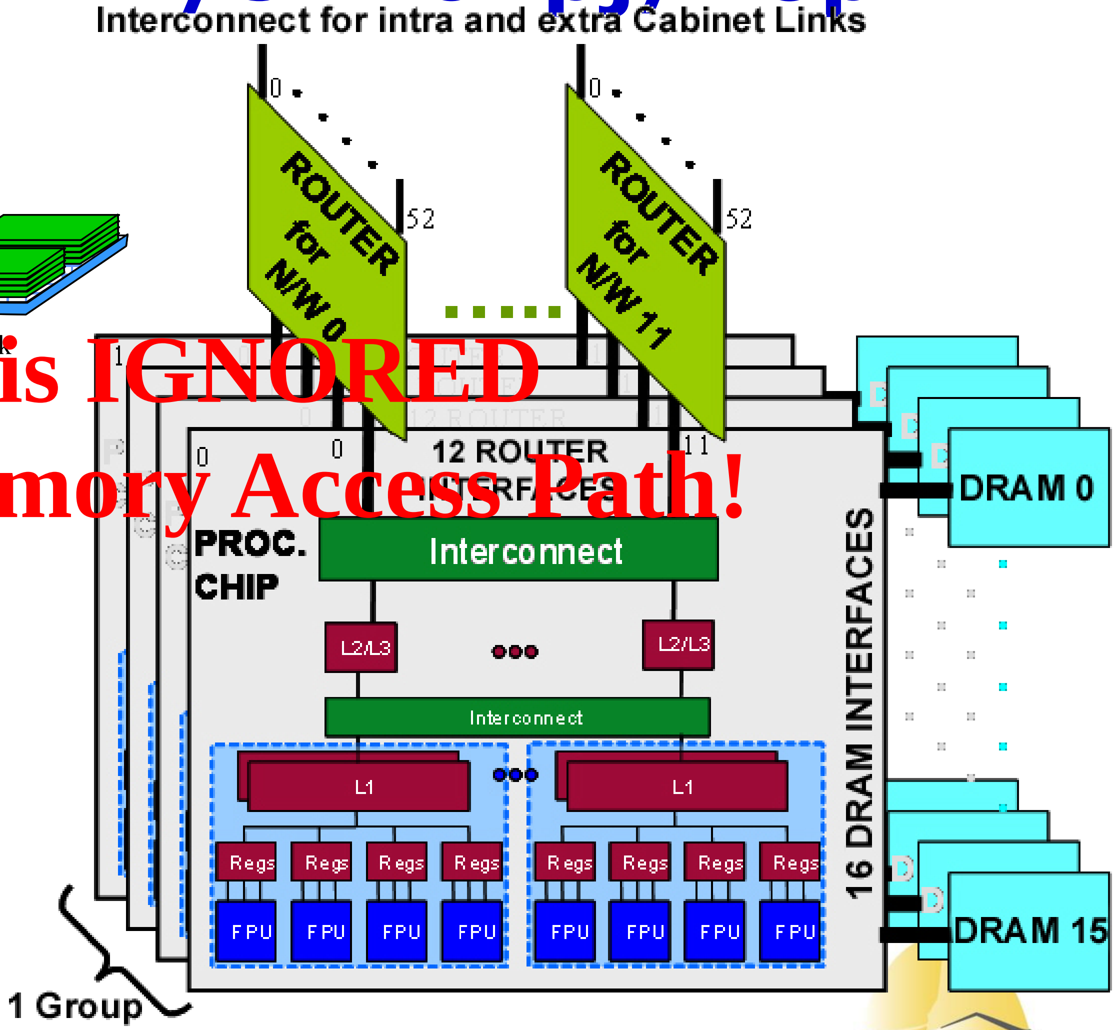
The Exascale Study Analysis: 67MW for 1EF/s = 67pj/flop


(a) Quilt Packaging


(b) Thru via chip stack

But This IGNORED Most of Memory Access Path!

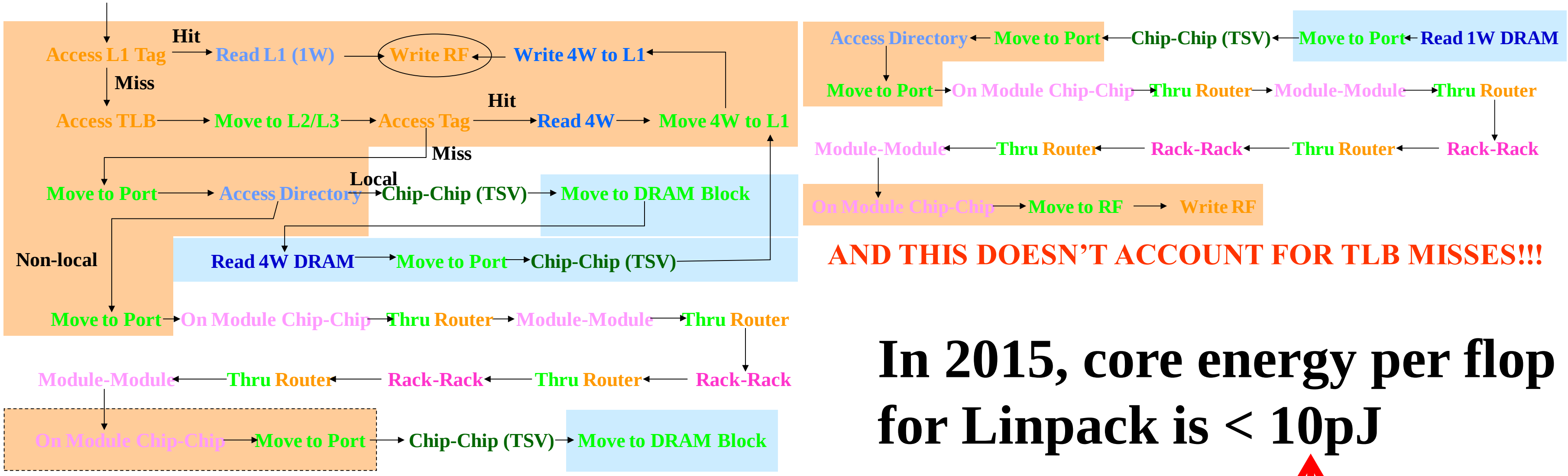




Sample Path – Off Module Access

1. Check local L1 (miss)
2. Go thru TLB to remote L3 (miss)
3. Across chip to correct port (thru routing table RAM)
4. Off-chip to router chip
5. 3 times thru router and out
6. Across microprocessor chip to correct DRAM I/F
7. Off-chip to get to correct DRAM chip
8. Cross DRAM chip to correct array block
9. Access DRAM Array
10. Return data to correct I/R
11. Off-chip to return data to microprocessor
12. Across chip to Routre Table
13. Across microprocessor to correct I/O port
14. Off-chip to correct router chip
15. 3 times thru router and out
16. Across microprocessor to correct core
17. Save in L2, L1 as required
18. Into Register File

Relook at Exascale Strawman



Access L1 Tag → Hit → Read L1 (1W) → Write RF ← Write 4W to L1

Miss

Access TLB → Move to L2/L3 → Access Tag → Hit → Read 4W → Move 4W to L1

Miss

Move to Port → Access Directory → Local → Chip-Chip (TSV) → Move to DRAM Block

Non-local

Read 4W DRAM → Move to Port → Chip-Chip (TSV)

Move to Port → On Module Chip-Chip → Thru Router → Module-Module → Thru Router

Module-Module ← Thru Router ← Rack-Rack ← Thru Router ← Rack-Rack

On Module Chip-Chip → Move to Port → Chip-Chip (TSV) → Move to DRAM Block

Access Directory ← Move to Port ← Chip-Chip (TSV) ← Move to Port ← Read 1W DRAM

Move to Port → On Module Chip-Chip → Thru Router → Module-Module → Thru Router

Module-Module ← Thru Router ← Rack-Rack ← Thru Router ← Rack-Rack

On Module Chip-Chip → Move to RF → Write RF

AND THIS DOESN'T ACCOUNT FOR TLB MISSES!!!

In 2015, core energy per flop for Linpack is < 10pJ

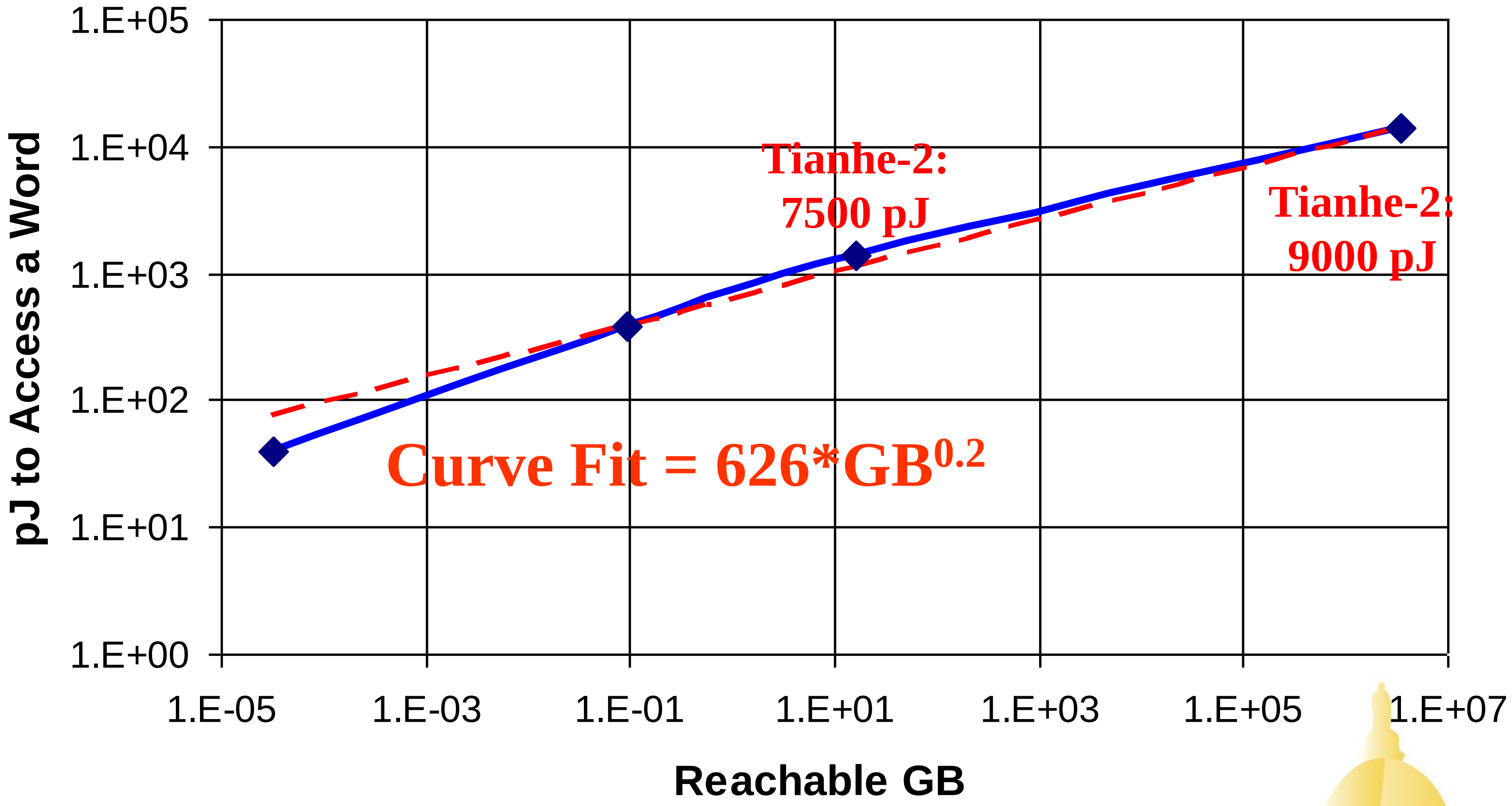
Operation	Energy (pJ/bit)
Register File Access	0.16
SRAM Access	0.23
DRAM Access	1
On-chip movement	0.0187
Thru Silicon Vias (TSV)	0.011
Chip-to-Board	2
Chip-to-optical	10
Router on-chip	2

Step	Target	pJ	#Occurrances	Total pJ	% of Total
Read Alphas	Remote	13,819	4	55,276	16.5%
Read pivot row	Remote	13,819	4	55,276	16.5%
Read 1st Y[i]	Local	1,380	88	121,[illegible]	[illegible]3%
Read Other Y[i]s	L1	39	264	10[illegible]	[illegible]%
Write Y's	L1	39	352	13,900	4.2%
Flush Y's	Local	891	88	78,880	23.4%
Total				334,656	
Ave per Flop				475	

50X

If this is true, 1 EF/s = 0.5 GW!

Access vs Reach



What Does This Tell Us?

- Cannot afford ***ANY*** memory references
- Many more energy sinks than you think
- Cost of Interconnect ***Dominates***
- Must design for on-board or stacked DRAM
- Need to redesign the entire access path:
  - Alternative memory technologies – reduce access cost
  - Alternative packaging costs – reduce bit movement cost
  - Alternative transport protocols – reduce # bits moved
  - Alternative execution models – reduce # of movements

AND IT GETS *MUCH WORSE*
FOR CACHE UNFRIENDLY PROBLEMS