Data Flow under the von Neumann Computer Architecture
冯·诺依曼体系结构下的数据流动

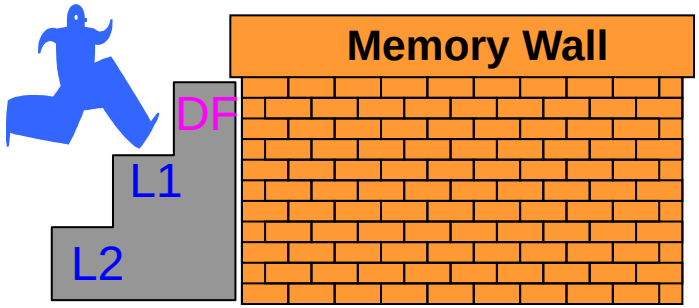
Xian-He Sun
Illinois Institute of Technology
sun@iit.edu

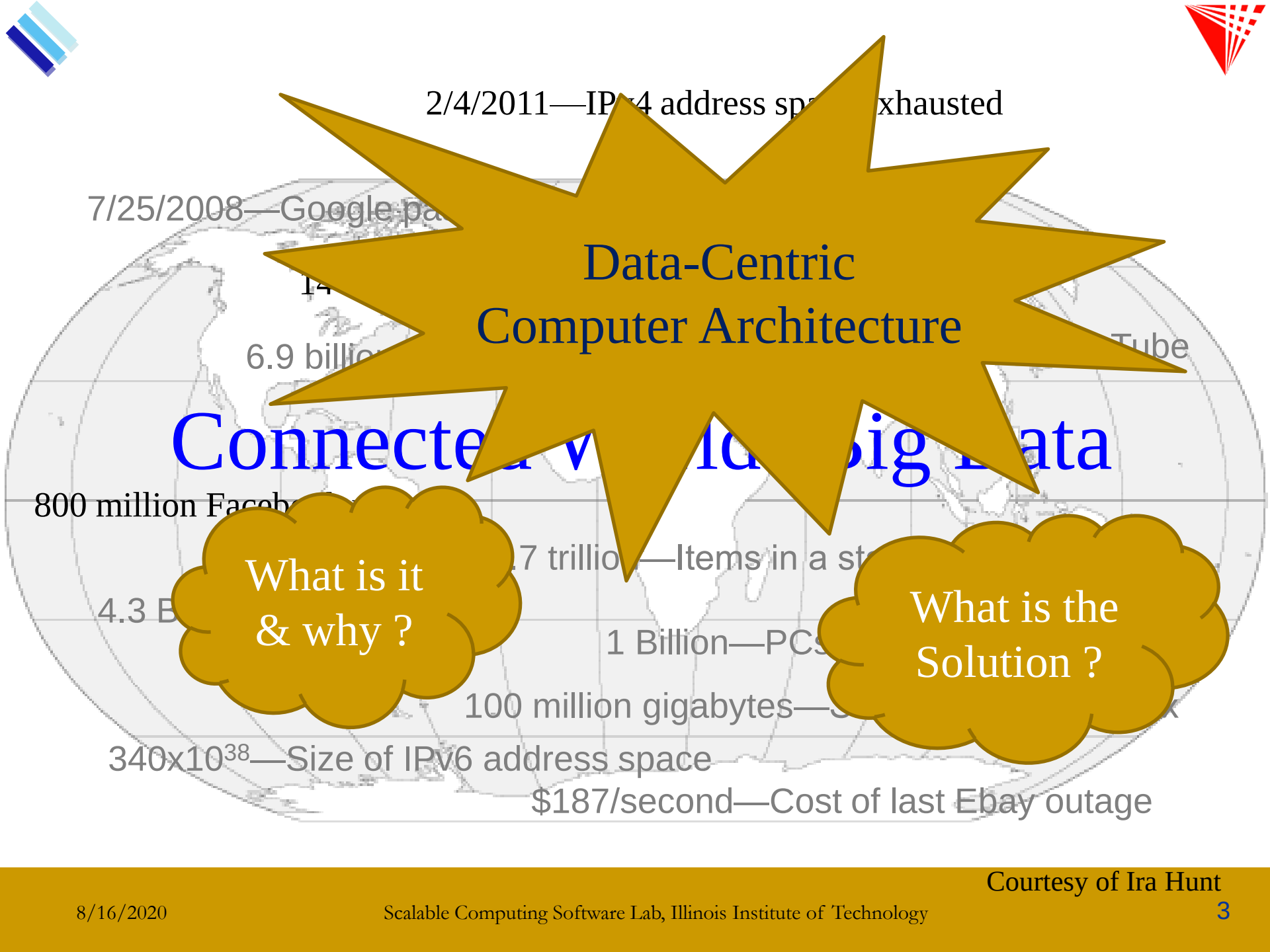
Keynote at CCF ACA 2020
August 13, 2020

A Fundamental Issue

Data-Centric Computer Architecture: Architecture in the Big Data Era

- Computational Thinking and Data-Centric Thinking
- Models and Solutions
- Application and implementation

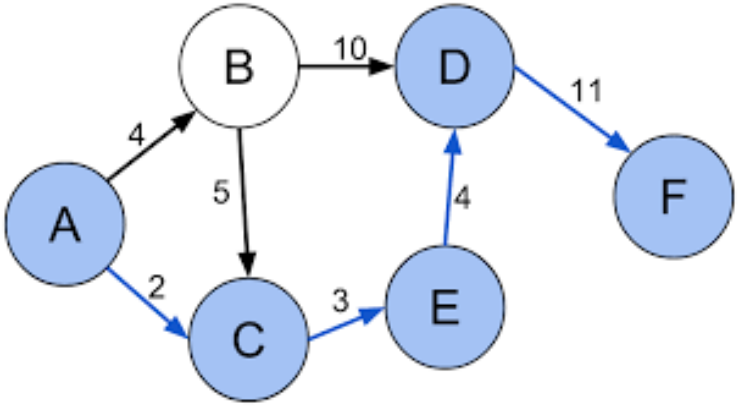




Courtesy of Ira Hunt

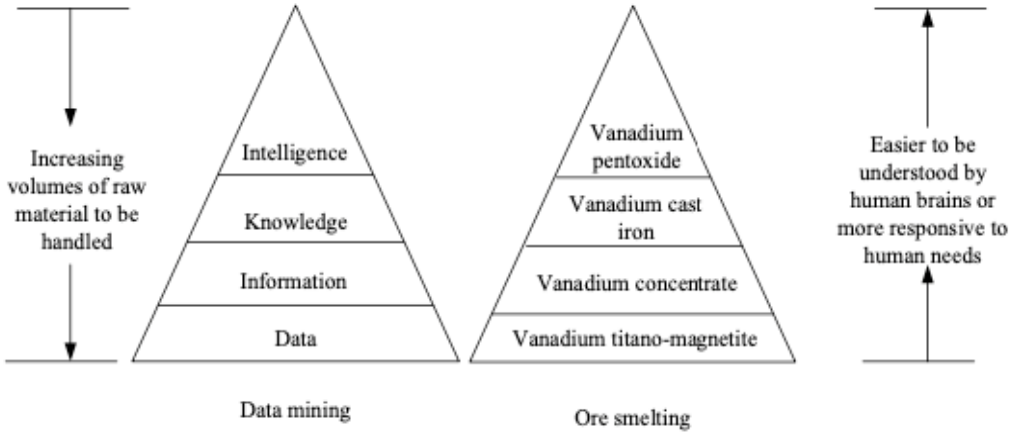
Computational Thinking

- Computer is for computing
  - Mathematical Modeling
  - CPU performance: *speed*, *flops*
- Scientific computing
  - Mathematical Equations: PDE, ODE equations
- Non-Scientific Computing
  - Graph based discrete computing
- Example
  - Travelling salesman problem



Data-Centric Thinking

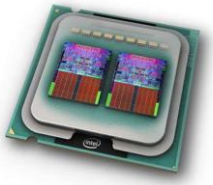
- Solving problem via data
  - Discover relations from (massive) data
  - Memory and storage system performance: *Bandwidth? Miss?*
- Deep learning and other methods
  - AI, generic, probability, statistical, index, graph, etc.
- Internet and database applications
- Example
  - flu epidemic
  - Travelling salesman problem



Y. Liu, X.-H. Sun, Y. Wang, and Y. Bao, "HCDA: From Computational Thinking to a Generalized Thinking Paradigm," Communication of ACM, accepted to appear

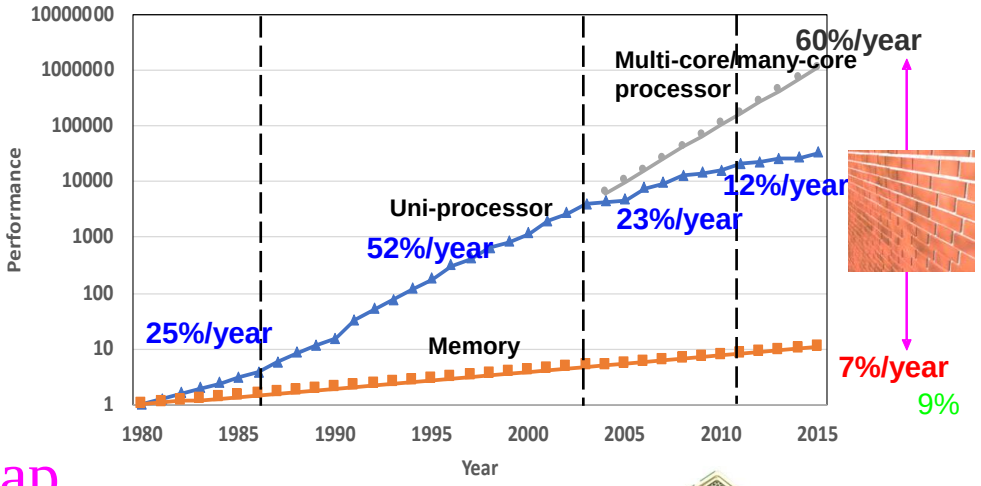
Why Data Centric ? The Memory-wall Problem

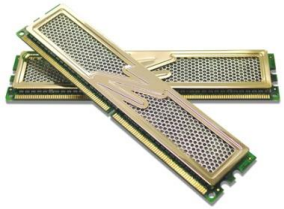
- Processor performance increases rapidly
  - Uni-processor: ~52% until 2004
  - Aggregate multi-core/many-core processor performance even higher since 2004
- Memory: ~7% per year
  - Storage: ~6% per year
- Processor-memory speed gap keeps increasing



Source: Intel

Leiserson, Charles E., et al. "There's plenty of room at the Top: What will drive computer performance after Moore's law?." *Science* 368.6495 (2020).





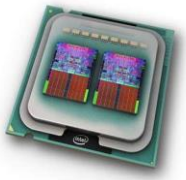
Source: OCZ

Memory-bounded speedup (1990), Memory wall problem (1994)

X.-H. Sun, and L. Ni, "Another View of Parallel Speedup," Proc. of ACM-IEEE Supercomputing'90, NY, Nov. 1990

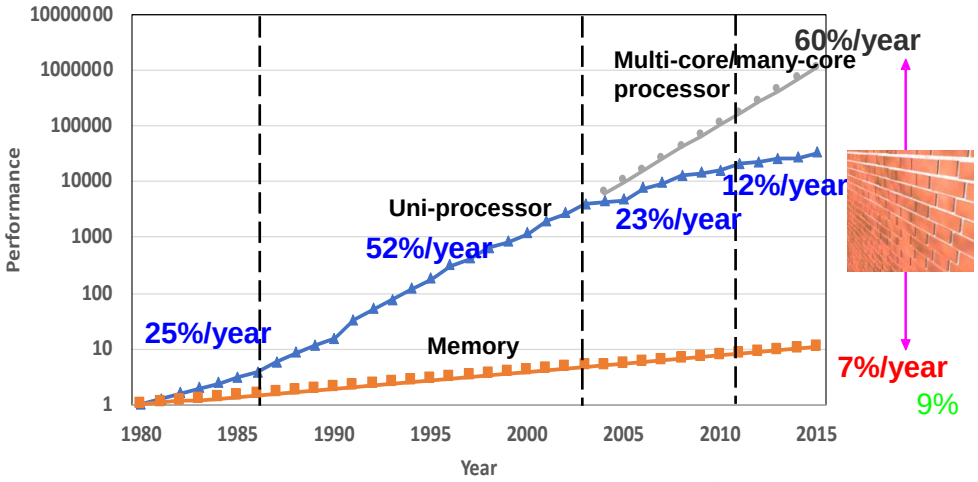
Why Data Centric ? The Memory-wall Problem

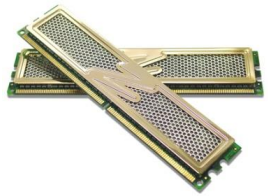
- Can we make a big memory?
  - No
- Can we make a big cache?
  - No
- Power consumption
  - 80% of memory are not used during 80% of its lifetime
- Memory & cache become small per processor/core



Source: Intel

Leiserson, Charles E., et al. "There's plenty of room at the Top: What will drive computer performance after Moore's law?." *Science* 368.6495 (2020).





Source: OCZ

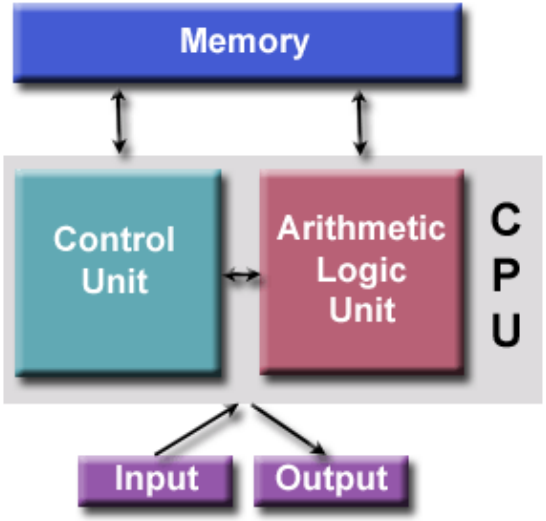
Memory-bounded speedup (1990), Memory wall problem (1994)

X.-H. Sun, Y. Chen. "Reevaluating Amdahl's Law in the Multicore Era," Journal of Parallel and Distributed Computing, Vol. 70, No. 2, pp183-188, 2010

The Von Neumann Architecture

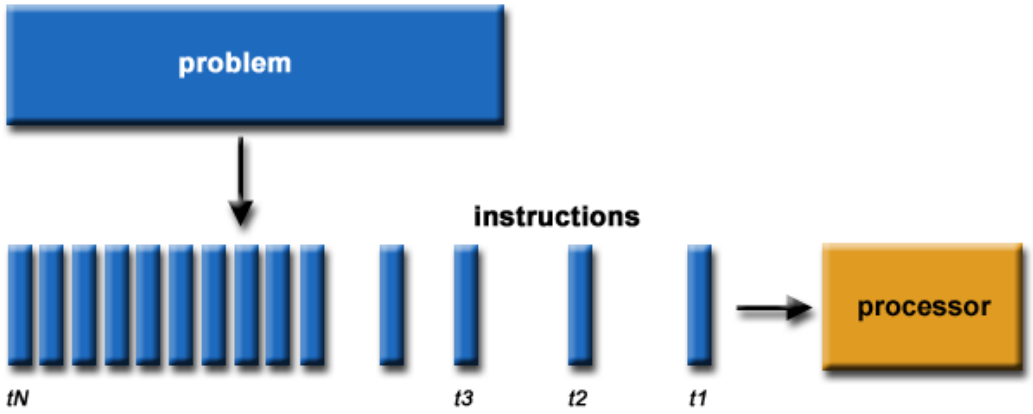
The Classical

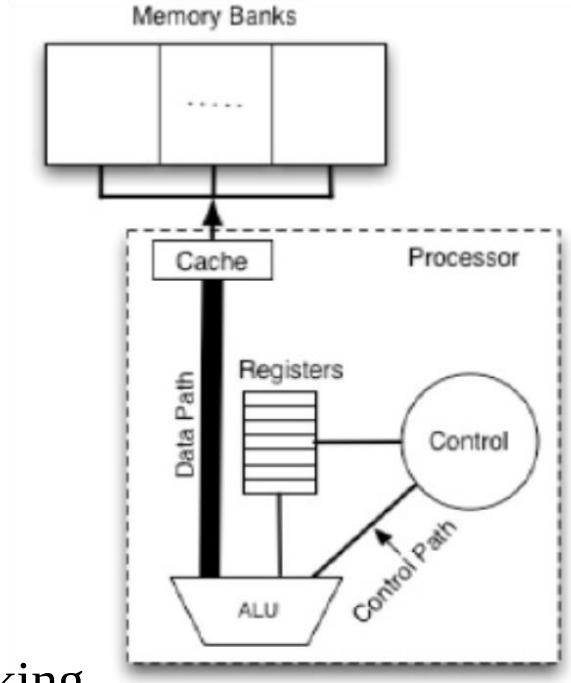
- John von Neumann first authored the general requirements for an electronic computer in 1945
- Aka “stored-program computer”
  - Both program inst. and data are kept in electronic memory
- Since then, all computers have followed this basic design
- Four main components: ALU, control unit, memory, I/O



von Neumann Implementation

- Why it is **Compute Centric ?**
  - A problem is broken into a discrete series of instructions
  - Instructions are executed by CPU
  - Fetch data if a computing need it
  - CPU utilization, CPU scheduling





Also called Control Flow Architecture

Computational thinking

Other Computer Architectures

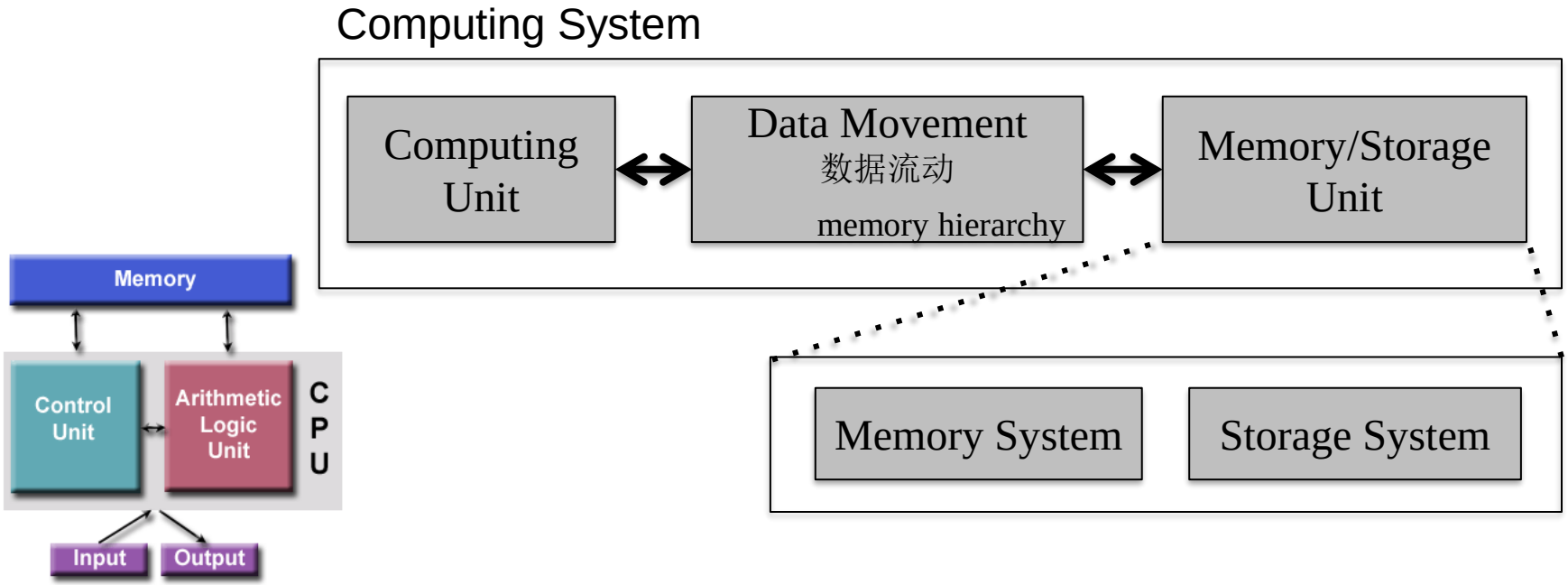
- The **Dataflow** Computer Architecture
  - Instruction execution is solely determined by the availability of data
  - Dataflow is designed to explore parallelism and maximize parallel computing (data move to make computing happen)

- **Quantum Computing**
  - Quantum computers are using existing electrical technologies for persisted memory and storage
  - Quantum supremacy is built based on "applications" which do not need inputting data

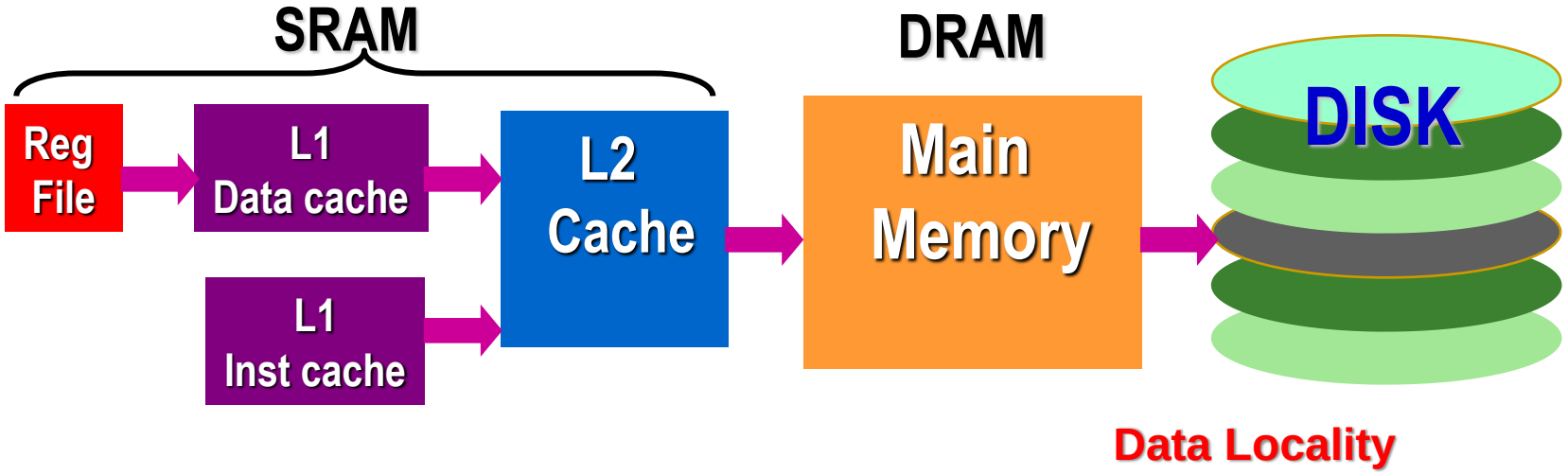
They are not data-centric

Re-examine the von Neumann Arch.

- Can we make von Neumann more data centric or compute and data equal ?
- **Yes**: focus on data and data access delay
- **How**: *Advance current memory-wall solutions*



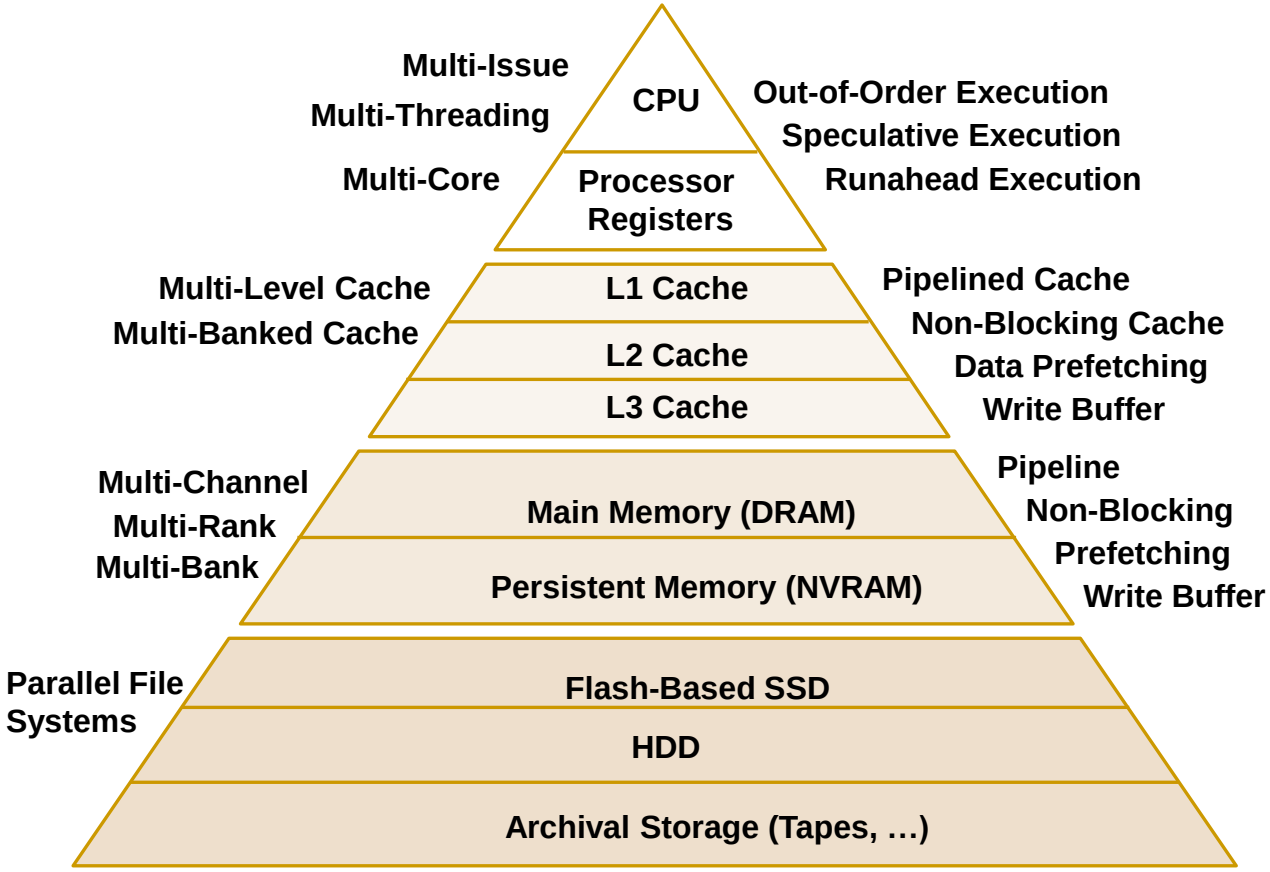
Memory-wall Solution: Memory Hierarchy



Advanced Solution: Deep Hierarchy & Concurrency

Assumptions

- Memory Hierarchy: Locality
- Concurrence: Data access pattern
  - Data stream



Deep Memory-Storage Hierarchy with Concurrence

Advanced Solution: ASIC from CPU side

- GPU, DSP, AI Chip
  - GPU is a chip tailored to graphics processing, DSP is for signal processing, and AI chip is designed to do AI tasks
- Limited data-centric
  - Assume data are on the chip
- Limited application
  - Accelerator
  - Data intensive?

New Solution: PIM chip

- PIM
  - Processing in memory (also called processor in memory) is the integration of a processor with RAM on a single chip.
  - NDP (Near-memory Data Processing)
  - ISP (In-Storage Processing)
- Computer power is weak
  - A full kitchen needs a refrigerator
- A helper/mitigator

Basic Idea: Separate CPU & Memory Under von Neumann

Memory Stall Time (MST)

$$\text{CPU.time} = \text{IC} \times (\text{CPI}_{\text{exe}} + \textit{Memory stall time}) \times \text{Cycle.time}$$

- Let reducing **MST** be the **final goal** of memory systems
  - Reduce data access delay
  - Separate the concern of memory systems
- The measurement of memory system performance
  - AMAT (Average Memory Access Time)

    AMAT = Hit time + MR × AMP
  - C-AMAT (Concurrent-AMAT)
  - APC (Access Per memory active Cycle) = 1/C-AMAT

D. Wang & X.-H. Sun, "APC: A Novel Memory Metric and Measurement Methodology for Modern Memory System," in *IEEE Transactions on Computers*, vol. 63, no. 7, pp. 1626-1639, July 2014

Reduce Memory Stall Time

The Traditional AMAT model

$$CPU.time = IC \times (CPI_{exe} + f_{mem} \times AMAT) \times Cycle.time$$

Memory stall time

The New C-AMAT model

$$CPU.time = IC \times (CPI_{exe} + f_{mem} \times C\text{-}AMAT \times (1 - overlapRatio_{c\text{-}m})) \times Cycle.time$$

Memory stall time

- Reducing **MST** becomes reducing C-AMAT

Reduce C-AMAT

- C-AMAT is Recursive

Where

$$C\text{-}AMAT_1 = \frac{H_1}{C_{H_1}} + MR_1 \times \kappa_1 \times C\text{-}AMAT_2$$

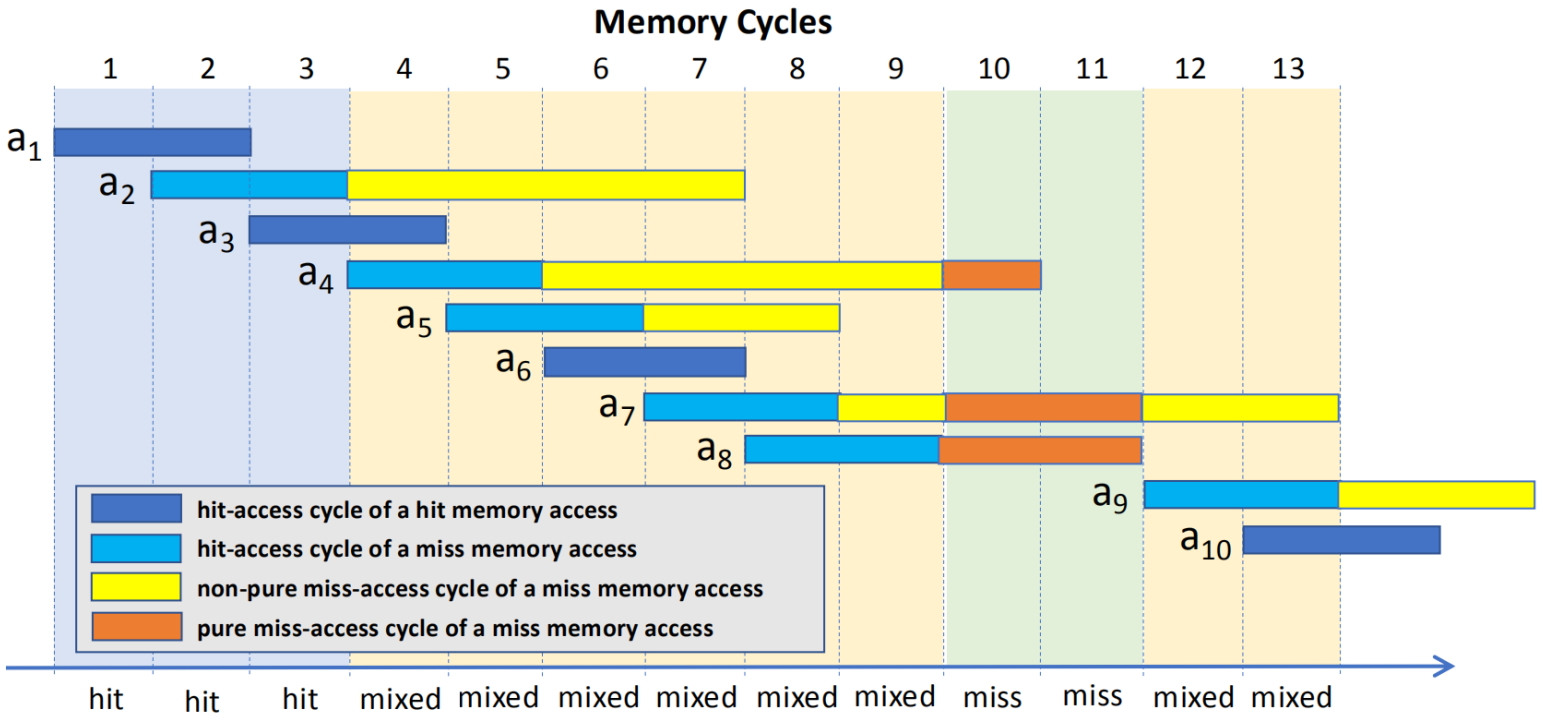
$$C - AMAT_2 = \frac{H_2}{C_{H_2}} + MR_2 \times \kappa_2 \times C - AMAT_3 \qquad \kappa_1 = \frac{pMR_1}{MR_1} \times \frac{pAMP_1}{AMP_1} \times \frac{C_{m\ 1}}{C_{M\ 1}}$$

- H is hit time
- MR is the miss ratio
- C_H is the hit concurrency
- κ is the overlapping ratio (pure miss cycles over miss cycles)
- A pure miss cycle is a miss cycle with no hit

With Clear Physical Meaning

C-AMAT : Four Types Cycle Analysis

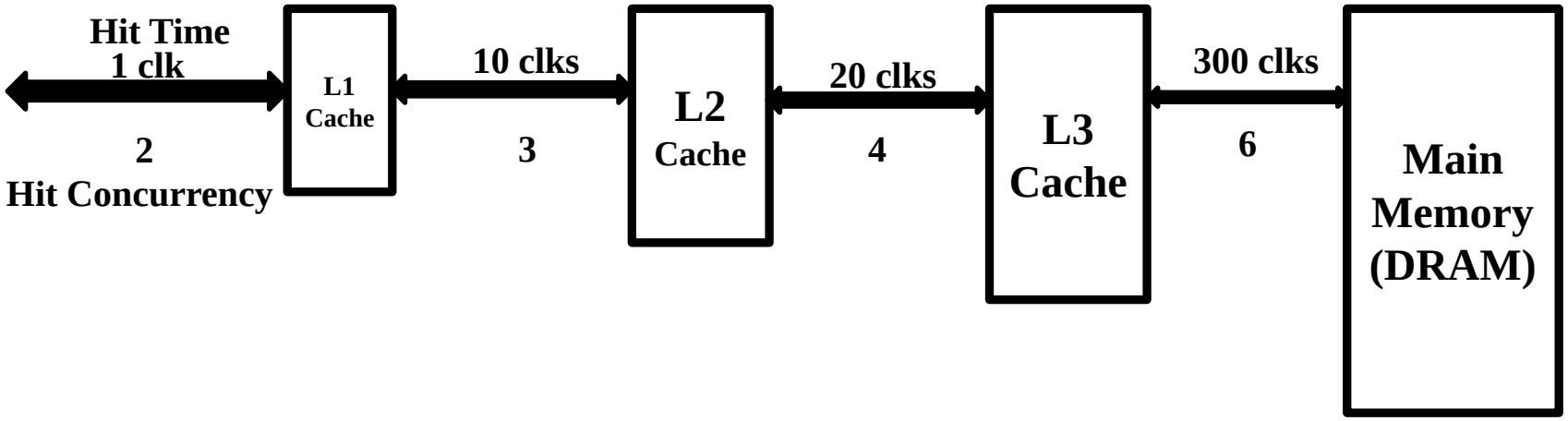
- Data (memory) centric analysis: memory cycles
- Memory cycles can see the overlapping



c	1	2	2	3	3	4	5	4	3	3	2	2	3
$c^{(h)}$	1	2	2	2	2	2	2	2	1	0	0	1	2
$c^{(m)}$	0	0	0	1	1	2	3	2	2	3	2	1	1

J. Liu, P. Espina, & X.-H. Sun, "A Study on Modeling and Optimization of Memory Systems," submitted for publication

C-AMAT is Recursive: Data Access Time



- **Concurrent Average Memory Access Time (C-AMAT)**

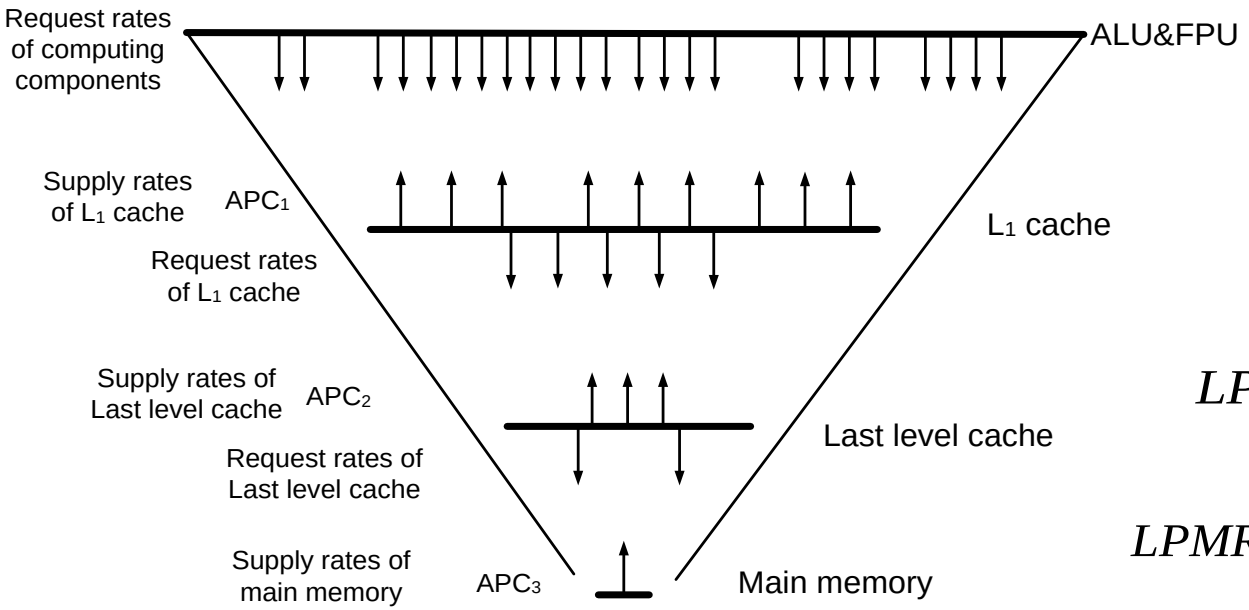
$$= \frac{H_1}{C_{H_1}} + MR_1 \times \kappa_1 \times \left(\frac{H_2}{C_{H_2}} + MR_2 \times \kappa_2 \times \left(\frac{H_3}{C_{H_3}} + MR_3 \times \kappa_3 \times \frac{H_{Mem}}{C_{H_{Mem}}} \right) \right)$$

- **Example**
  - *Miss Rate: L1=10%, L2=5%, L3=1%*
  - *κ: L1=0.56, L2=0.6, L3=0.8*
  - *C-AMAT≈0.696*

pMR, pAMP, AMP, C_M, C_m: L1=7%, 10, 10, 5, 4
L2=3%, 60, 40, 9, 6
L3=0.8%, 400, 300, 16, 12

Optimization: Layered Performance Matching

LPM with C-AMAT



$$LPMR_1 = \frac{IPC_{exe} \times f_{mem}}{APC_1}$$

$$LPMR_2 = \frac{IPC_{exe} \times f_{mem} \times MR_1}{APC_2}$$

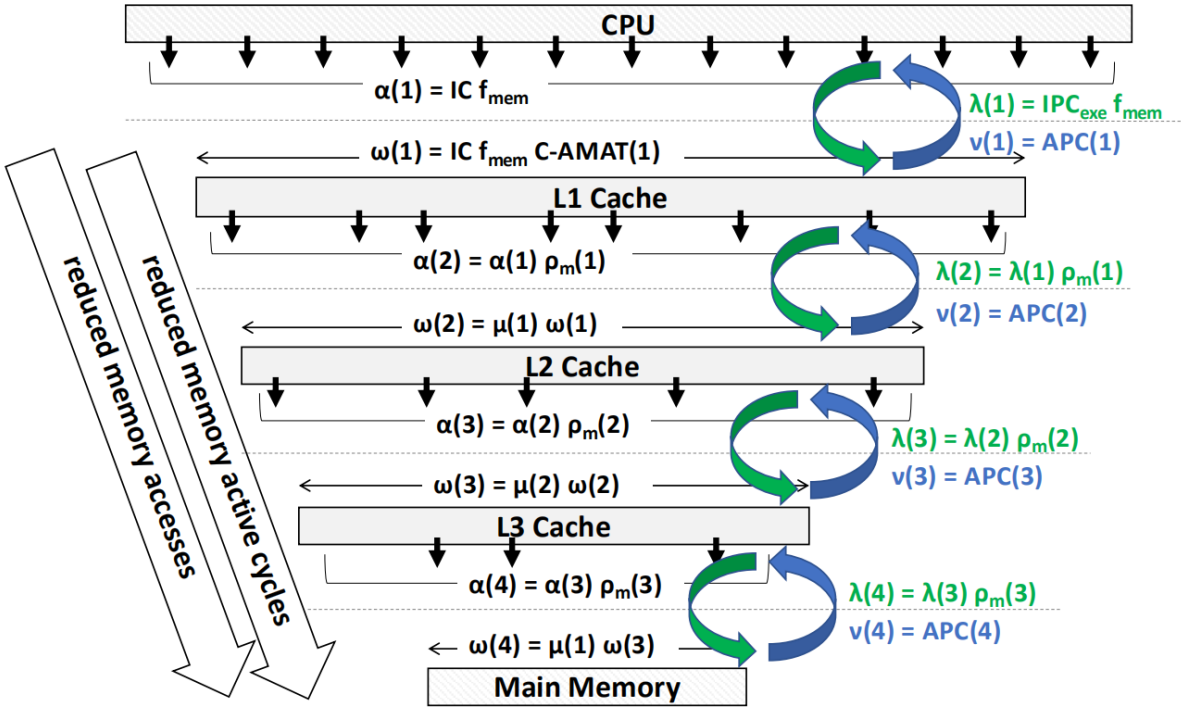
$$LPMR_3 = \frac{IPC_{exe} \times f_{mem} \times MR_1 \times MR_2}{APC_3}$$

- Match the data request and supply at each layer
- C-AMAT can increase supply with effective concurrency and locality

Y. Liu, X.-H. Sun. "LPM: A Systematic Methodology for Concurrent Data Access Pattern Optimization from a Matching Perspective," IEEE TPDS, vol. 30, no. 11, pp. 2478-2493, 1 Nov. 2019

Layered Performance Matching (LPM)

- The *Matching* ratio values of request and supply at each layer are given and the matching process is well designed & analyzed

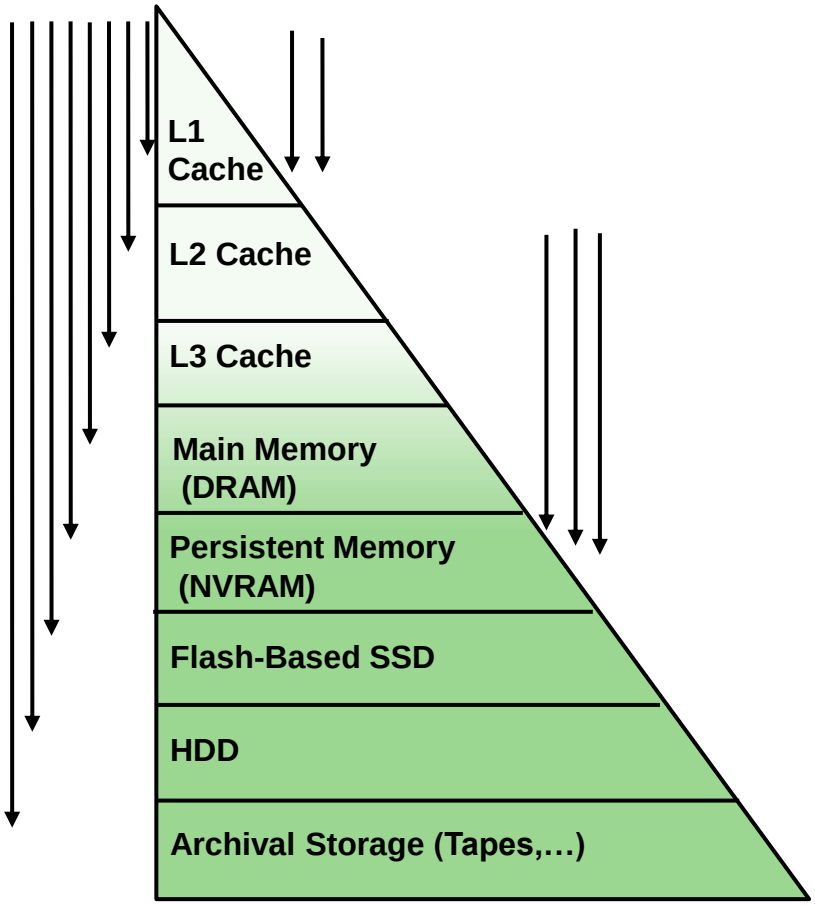


Simulatable → Measurable → Controllable → Optimizable

J. Liu, P. Espina, & X.-H. Sun, "A Study on Modeling and Optimization of Memory Systems," submitted for publication

Deep Memory-Storage Hierarchy: a general match

- Do we need to use all layers every time?
  NO
- Flexible tier selection with no inclusive
- Concurrent accesses now can concurrently access on different tiers
- Tier: memory device with different performance
- Layer: memory hierarchy with data inclusiveness
- A general match in Deep Memory-Storage Hierarchy (DMSH)



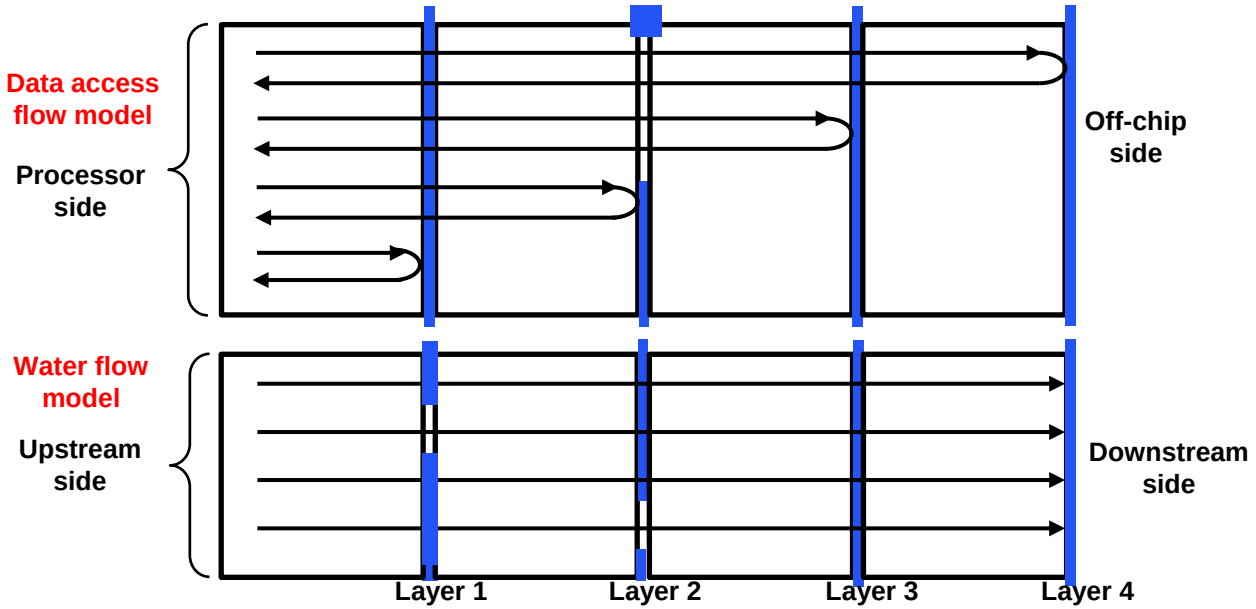
Persistent memory blurs memory & storage

DMSH with Concurrence

Y. Liu and X.-H. Sun, "CaL: Extending Data Locality to Consider Concurrency for Performance Optimization," IEEE Transactions on Big Data, vol. 5, no. 2, pp. 273-288, June 2018

Memory Sluice Gate Theory

***Sluice Gate Theorem**: If a memory system can match an application's data access requirement for any matching parameter $T_1 > 0$, then this memory system has removed the memory wall effect for this application.*



X.-H. Sun and Y.-H. Liu, "Utilizing Concurrency Data Access: A New Theory," in Proc. of LCPC2016, Sept. 2016, New York, USA

The Next Step: Reduce Data Movement

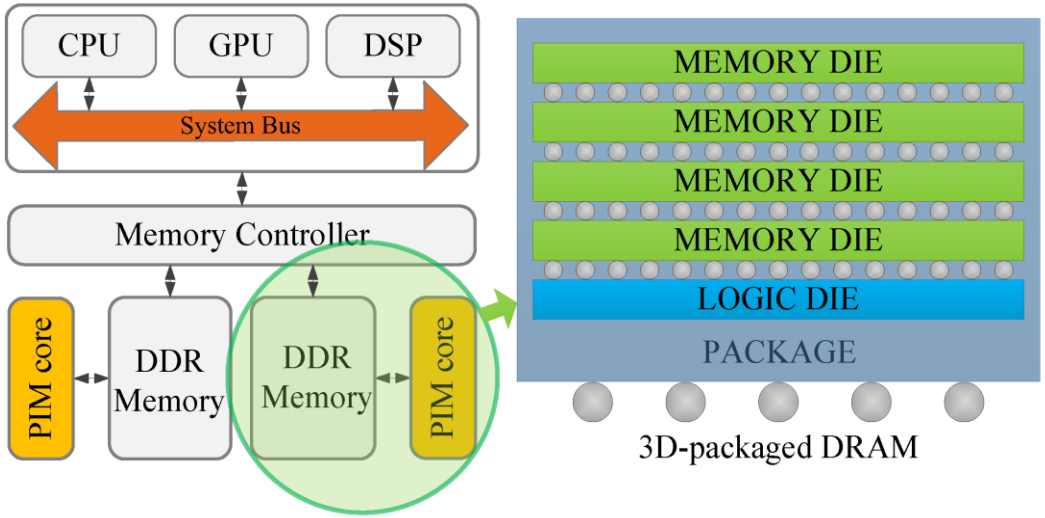
Memory Stall Time (MST)

$$\text{CPU.time} = \text{IC} \times (\text{CPI}_{\text{exe}} + \textit{Memory stall time}) \times \text{Cycle.time}$$

- Separating Compute and Memory is NOT data centric, reducing data movement is
- Next step: Reducing (CPI_{exe} + **MST**) by reducing data movement
- How?
  - Add Processing in Memory (PIM) into the matching

PIM: A Part of the Sluice Gate Consideration

- PIM conduct computing at the memory side and reduce data movement
- PIM is less powerful and when can help is an issue
- When to use PIM is part of the Sluice Gate theory
- Similar discussions for NDP and ISP



Next Step: Include PIM into the Picture

Memory Stall Time (MST)

$$\text{CPU.time} = \text{IC}_{\text{exe}} \times (\text{CPI}_{\text{exe}} + \textit{Memory stall time}\ ) \times \text{Cycle.time} + \text{IC}_{\text{pim}} \times \text{CPI}_{\text{pim}} \times \text{Cycle.time}_{\text{pim}}$$

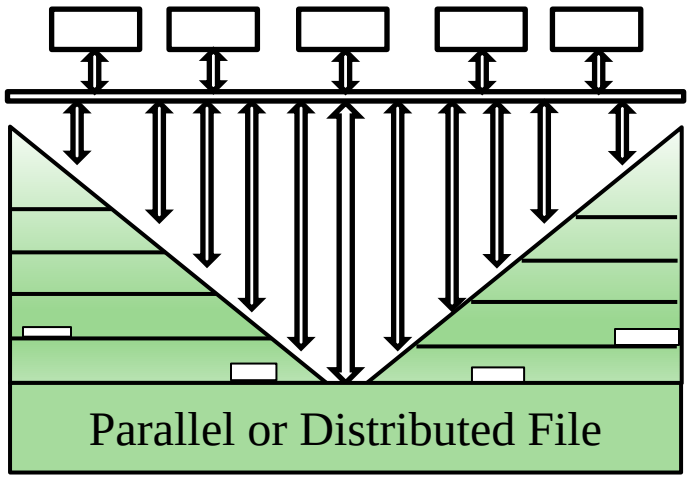
- Add PIM into the performance formulation
- PIM is a way to reduce request and is a trade-off of computing and MST
- Result: data movement cost decides where to do the computing

Data Flow Sluice Gate Control

- Two classes of computing devises, powerful CPU (multicore, GPU, XPU. Etc.) and less powerful PIM (NDP, ISP， etc.)
- Sluice gates decide which data are processed on PIM
- (rest) Data flows from memory in a rhythmic, concurrent matching fashion, passing through sluice gates (layers) before reach a CPU, then return to memory
- A general structure:
  - Fin-in, fin-out, branch,
  - More than one PIM/NDP/ISP and more than one CPU/GPU/XPU
  - Staged execution
  - Storage is the last layer of the data movement hierarchy

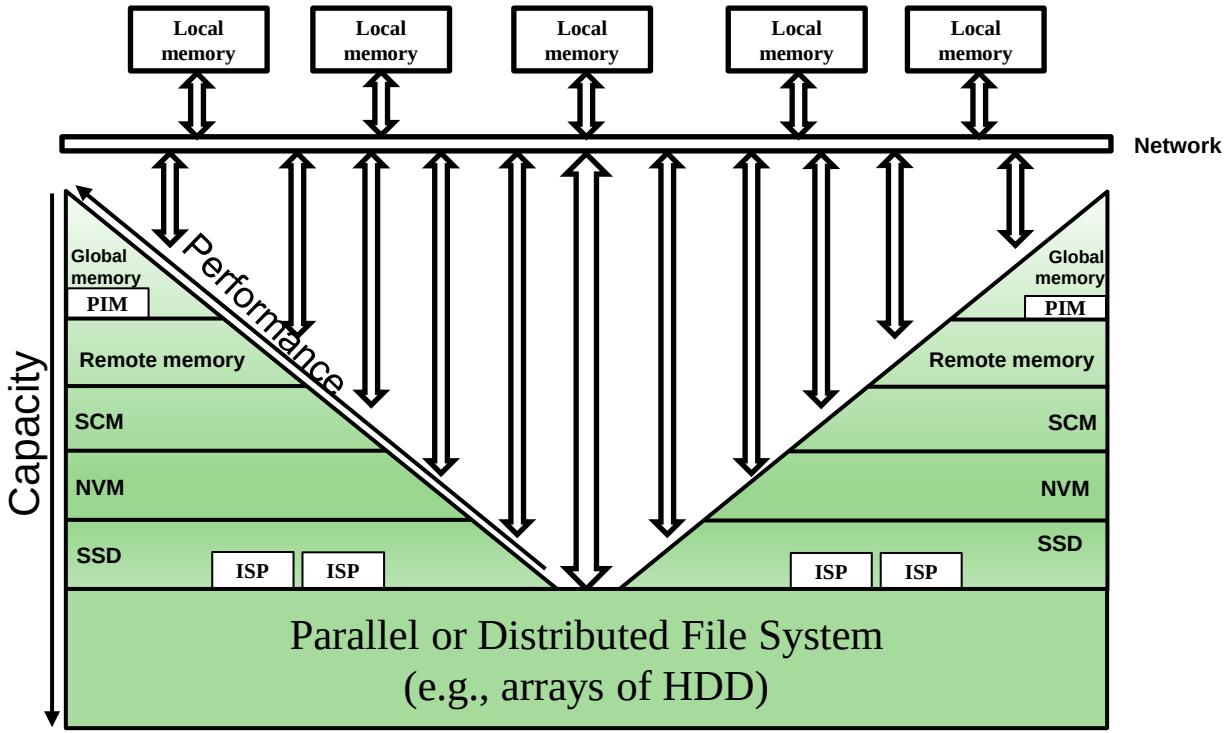
Data Flow under von Neumann

- Memory hierarchy with PIM (von Neumann)
- *Optimize 【compute + data access】 via Sluice Gate theory*
- Data flow from memory to CPU with minimum MST and conduct processing in memory when necessary
- $Dataflow_v$



Data Centric Sluice Gate Theory

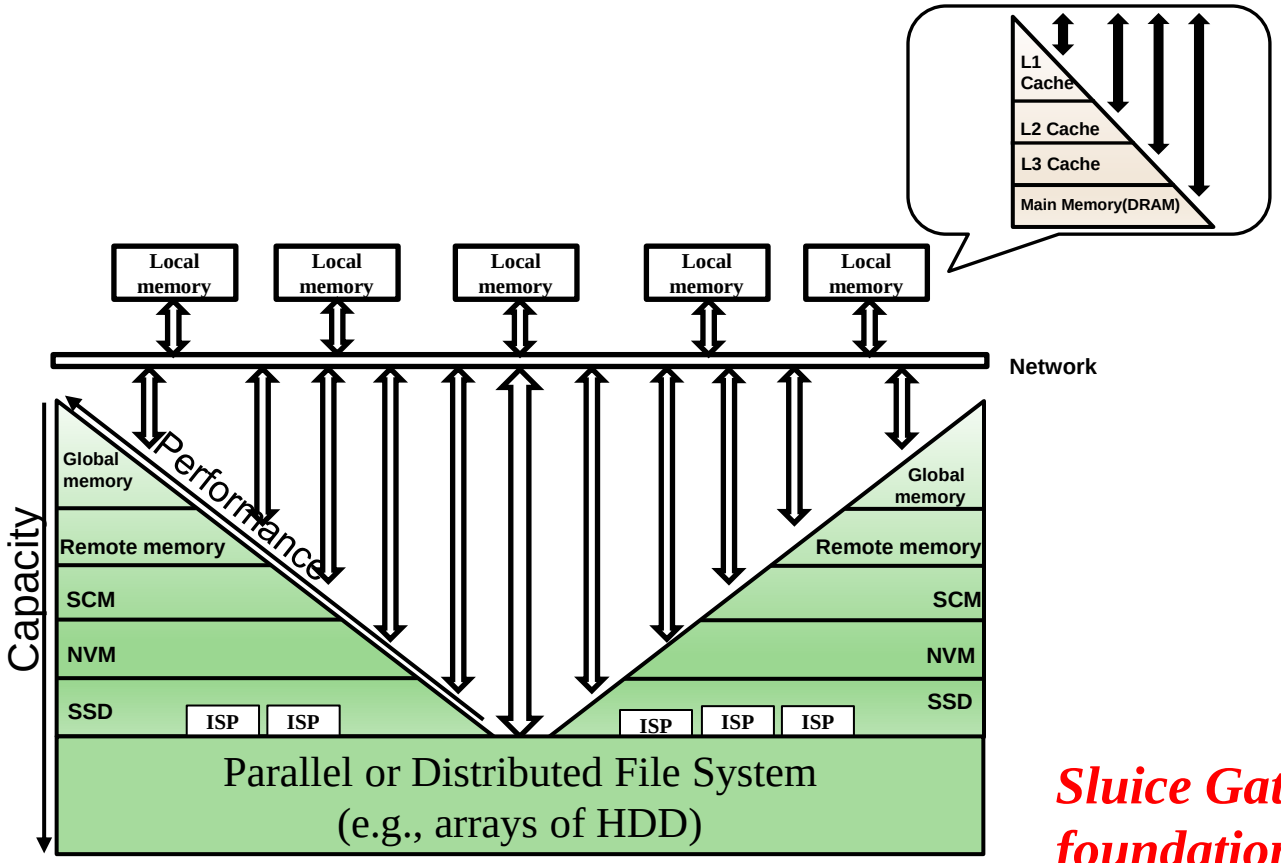
- Sluice Gate theory with in-place-computing
  - Concurrent access at each tier, concurrent access at different tiers
- Data movement cost is as important as computing power



Sluice Gate Theory is the foundation of $Dataflow_v$

Data Centric Sluice Gate Theory

- Sluice Gate theory with in-place-computing
- Data movement cost is as important as computing power



Sluice Gate Theory is the foundation of $Dataflow_v$

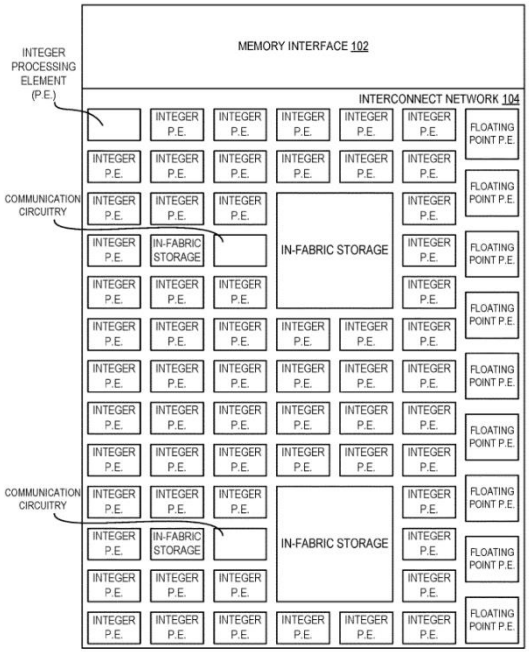
Dataflow$_v$ Summary

- Where to do computing is determine by *computing and data movement cost*
- Data flow through memory hierarchy via *matching the request and supply* at hierarchy layer interfaces
- Utilize newly proposed PIM (NDP, ISP, etc.) technology to reduce data movements
- Support advanced computer architectures, such as multicore, disaggregated memory, deep memory-storage hierarchy, etc.
- Selective cache/buffer with multi-tiers concurrent accesses
- *A data-centric implementation* of the von Neumann computer architecture
- *PIM and Deep Memory-Storage Hierarchy*

$Dataflow_v$ Application

Cache Level

- GPU, AI, XPU
  - Challenge: CPU/Accelerator data movement
- FPGA, RISC-V
  - Application: Guideline
- ***General purpose multicore***
  - ***Challenge: Fast elastic***



Intel CSA

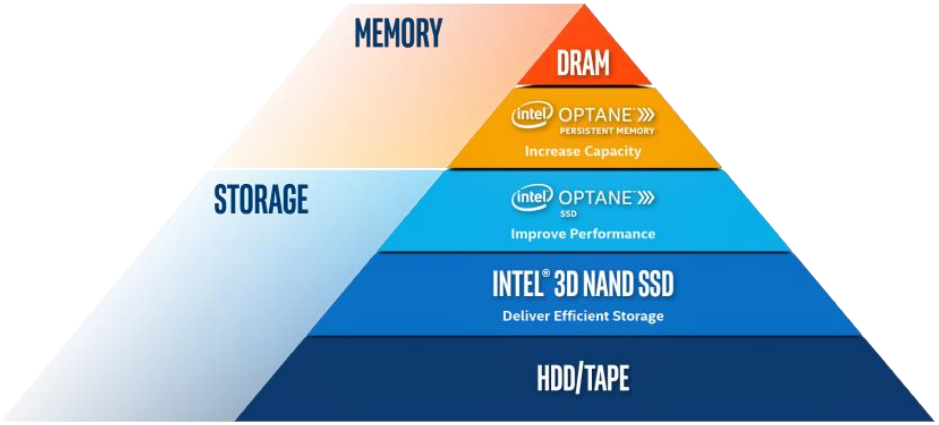
Accelerator-based Microarchitecture ?

N. Zhang, C. Jiang, X.-H. Sun, S. Song, "Evaluating GPGPU Memory Performance Through the C-AMAT Model," in Proc. of ACM SIGHPC Workshop on Memory Centric Programming for HPC, with SC'17, Denver, USA, Nov. 2017

$Dataflow_v$ Application

Memory Level

- Quite open with new opportunities
- Challenges
  - Interface with cache and storage
  - Measurement and Simulator
  - Controllers
  - OS
- ***Infrastructure ?***



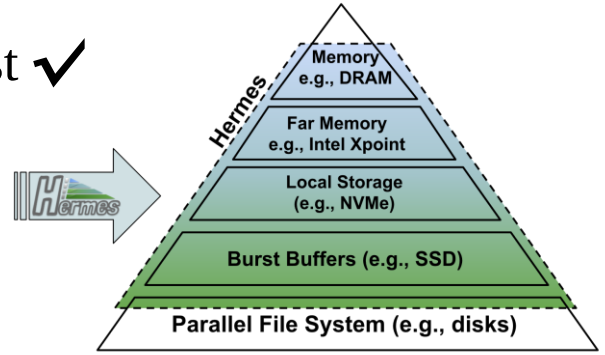
N. Zhang, B. Toonen, X-H. Sun, B. Allcock, "Performance Modeling and Evaluation of a Production Disaggregated Memory System," International Symposium on Memory Systems (MEMSYS'20), Sept. 2020 (accepted to appear)

$Dataflow_v$ Implementation

I/O level

- Storage is the last level of the memory hierarchy (DMSH)√
- Start at where the data is
- Advantage
  - Can be implemented and verified
- ***Challenges***
  - Data management ✓
  - Network impact √
  - Passing operation demands with data request ✓

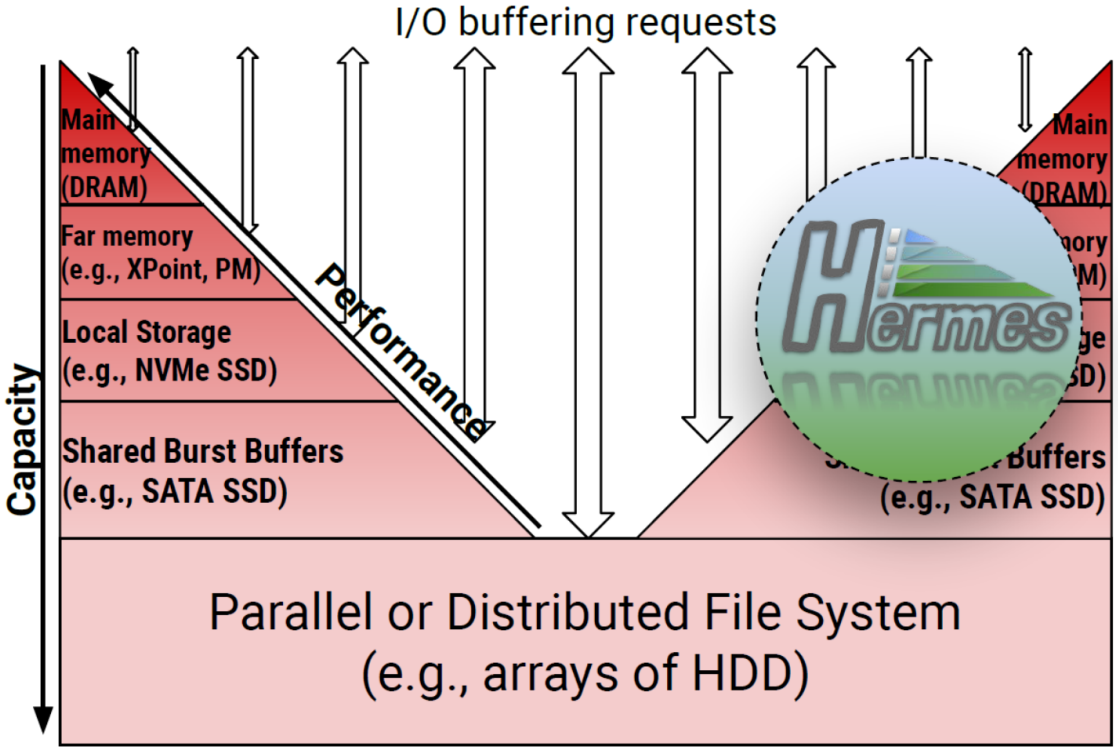
Let us do it ?



Anthony Kougkas, Hariharan Devarajan, and Xian-He Sun. "Hermes: a heterogeneous-aware multi-tiered distributed I/O buffering system," ACM, HPDC18, Tempe, Arizona, USA, June 2018

Hermes: A Multi-tiered I/O Buffering System

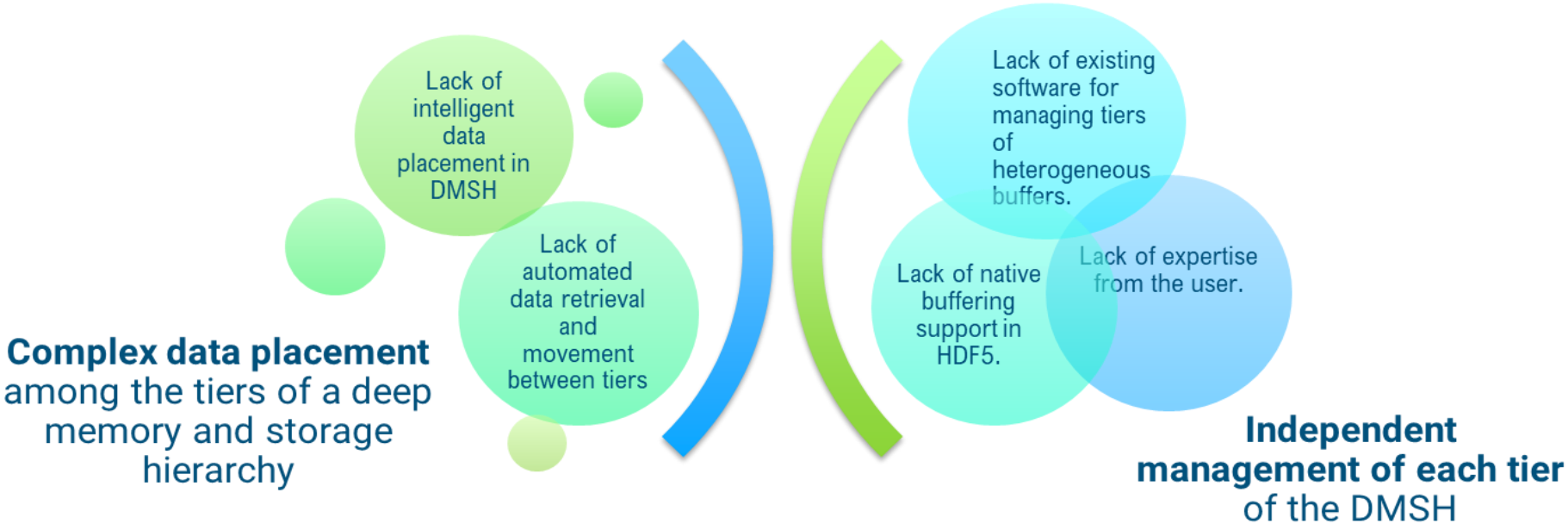
- Selective cache, concurrent, matching
- Independent management of each tier



A. Kougkas, H. Devarajan, and X.-H. Sun, "I/O Acceleration via Multi-Tiered Data Buffering and Prefetching," Journal of Computer Science and Technology, vol. 35, no. 1, pp. 92-120, Jan. 2020

Hermes: A Multi-tiered I/O Buffering System

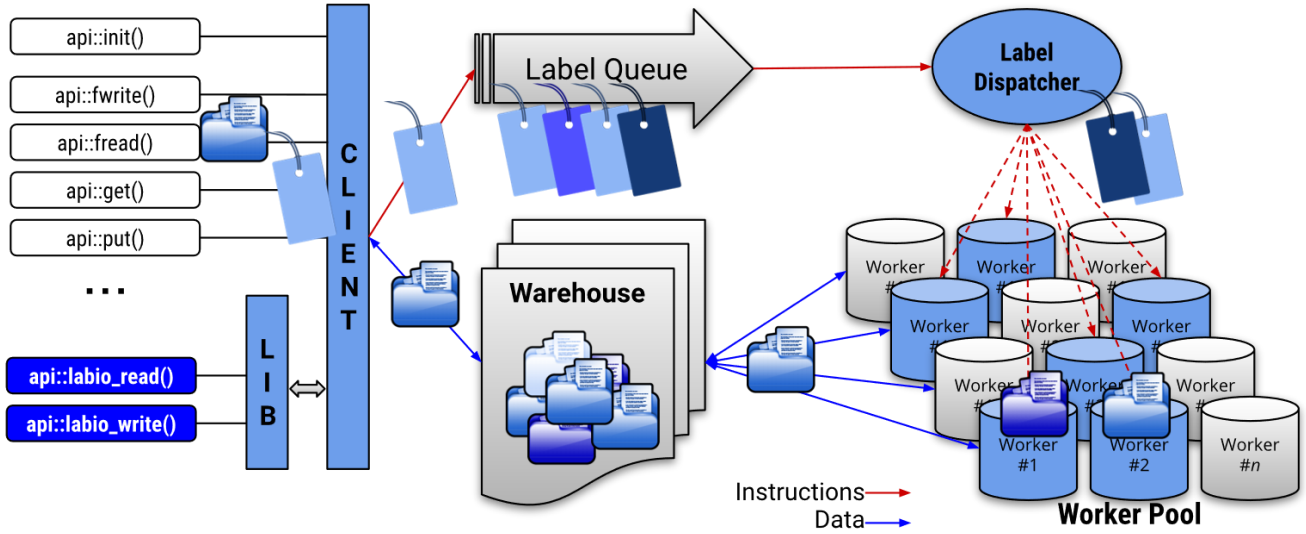
- Application-aware multi-tier matching
- Start at the log file
- An example of memory/storage integration
- An implementation of the $Dataflow_v$ concept



A. Kougkas, H. Devarajan, and X.-H. Sun, "Bridging Storage Semantics using Data Labels and Asynchronous I/O," ACM Transactions on Storage, accept to appear

dLabel: Data Operation with Label

- Data requests are transformed into (data) ***Label*** units
  - A label is a tuple of an operation and a pointer to the data
- A dispatcher distributes labels to the workers
- Workers execute labels independently (i.e., fully decoupled)



A. Kougkas, H. Devarajan, J. Lofstead, X.-H. Sun; "*LABIOS: A Distributed Label-Based I/O System", i*n Proceedings of ACM HPDC '19 (Best Paper Award)

Take Home Messages

- What is data-centric architecture ?
  - Reduce data movement and memory stall time (in addition to better device)
  - Design an architecture to do so
- How to do it ?
  - Start at where the data is
  - Computing side ? Storage side ?
- Storage side key technologies
  - Modeling
  - Deep memory/storage hierarchy, labeled data, ChronoLog, etc.
  - Hardware/software co-design
- Many things need to do
  - **The good part**: Any progress is an improvement of current systems

A. Kougkas, H. Devarajan, K. Bateman, J. Cernuda, N. Rajesh and X.-H. Sun, "ChronoLog: A Distributed Shared Tiered Log Store with Time-based Data Ordering," Proceedings of the 36th International Conference on Massive Storage Systems and Technology (MSST 2020), Oct. 2020, (accepted to appear).

Conclusion

- Make von Neumann architecture more data centric
  - $Dataflow_v$
- The I/O implementation & other applications
  - Hermes: integration of calculation & storage
- The concept of matching and Sluice Gate Theory
  - In place computing & deep memory/storage hierarchy

Thank you
Any questions?

Data Flow under the von Neumann Architecture

Xian-He Sun
The SCS laboratory at the Illinois Institute of Technology

sun@iit.edu
www.cs.iit.edu/~scs

We would like to thank our sponsors the
National Science Foundation