



Flash Memory Summit

# Flash As Memory

Maher Amer

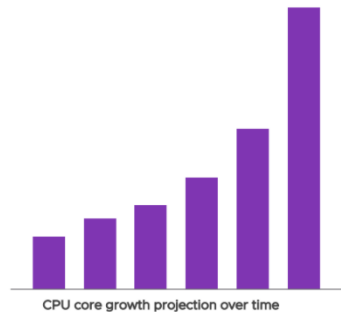
Co-Founder: Thikra Technology

# The Challenge: DRAM is not playing nice

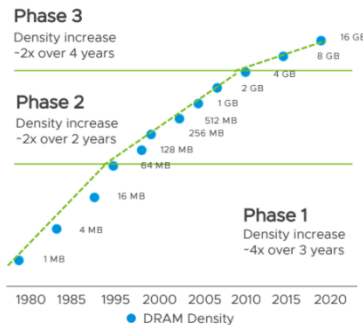
- All components in servers are scaling nicely except memory
- Cost of DRAM (\$/GB) is not scaling as well

## Compute, Memory & Data

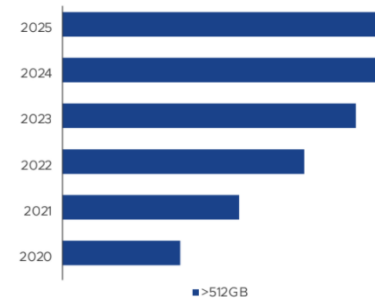
Compute demand accelerating



DRAM not scaling  
Compound annual growth rate in data<sup>2</sup>



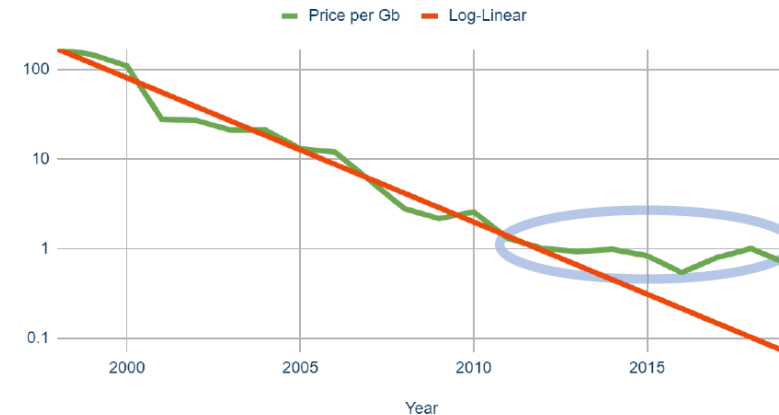
Large-memory systems growing in prevalence



Data-intensive workloads need hot data in memory, but DRAM is expensive and has limited capacity  
Enterprise data is vulnerable, but security and encryption can compromise performance

## Increasing Memory Cost and Power

Price per Gb (Log Scale)

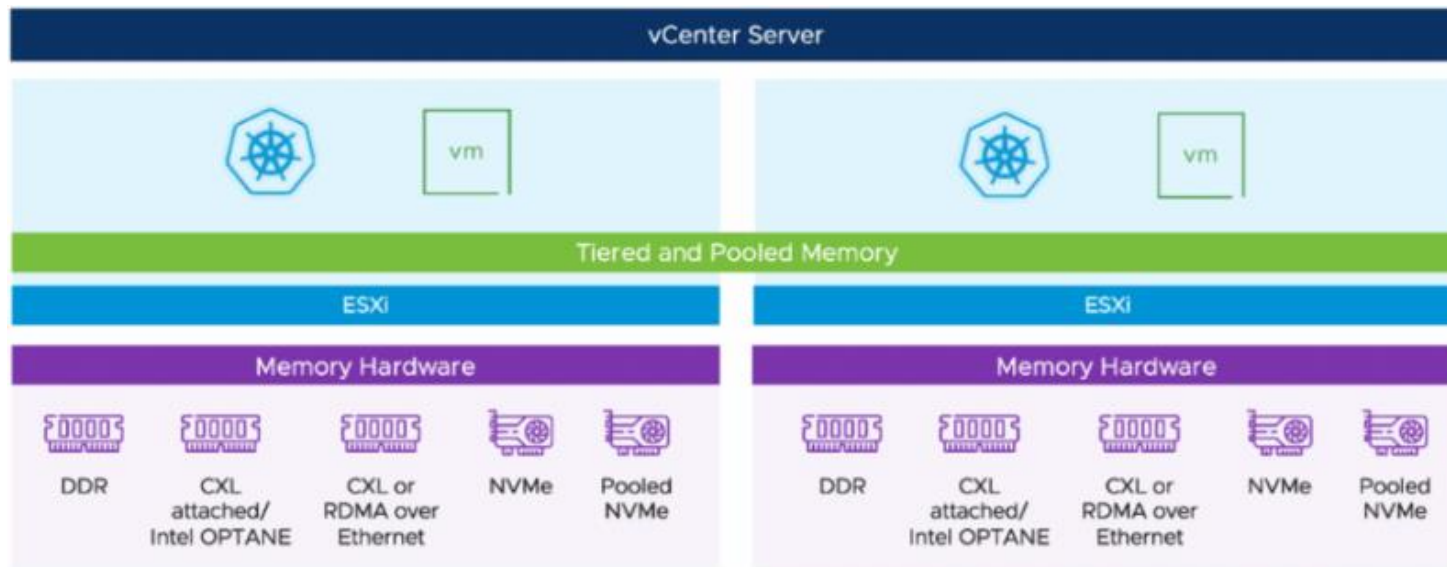


### Memory an increasing % of system power and cost

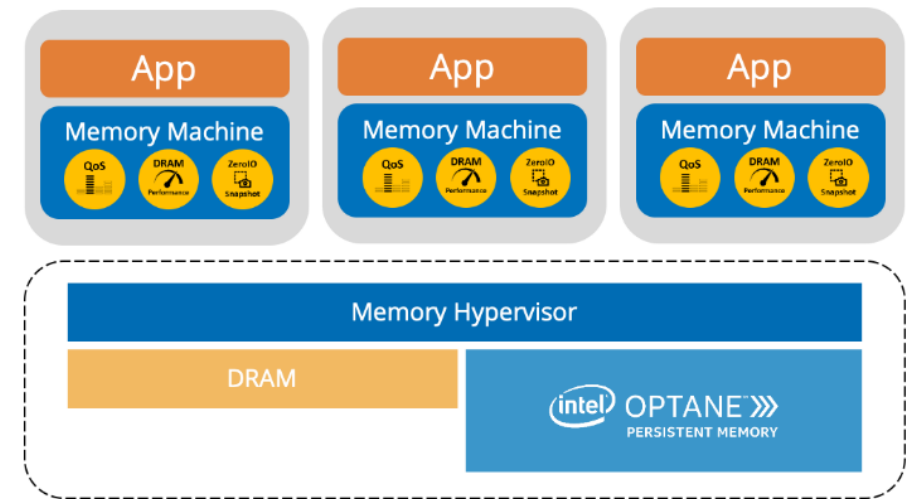
- Memory price (cost/bit) flat due to scaling challenges
- Memory power scaling with speed

# This is NOT Memory Abstraction

- Memory Abstraction is very well-established technology
- What happens when flash is being used as one of memory tiers



\*VMware Project Capitola



<https://memverge.com/memory-machine-data-sheet/>

# Why Flash As Memory: CXL Use Cases

Use Case	Description	Value Proposition	Availability
Persisting memory	Use Persistent memory technologies to host memory data	Avoid expensive data-persisting operations: sync, snapshotting, etc	Applications need to be modified. Long term availability
Memory Sharing	Avoid having to cache memory in different HW components: NIC, Storage, GPU, etc	Save in memory utilization and cost	Applications/FW need to be modified to leverage this value prop. Long term availability
Memory Expansion	Increase memory footprint and provide configurability	??***	Once CXL HW is available

- The only use case that can immediately leverage CXL is memory expansion, but its value proposition is not clear due to lack of TCO story
- Need to leverage economic and dense available technologies: **FLASH**

# Tiered Memory is already here

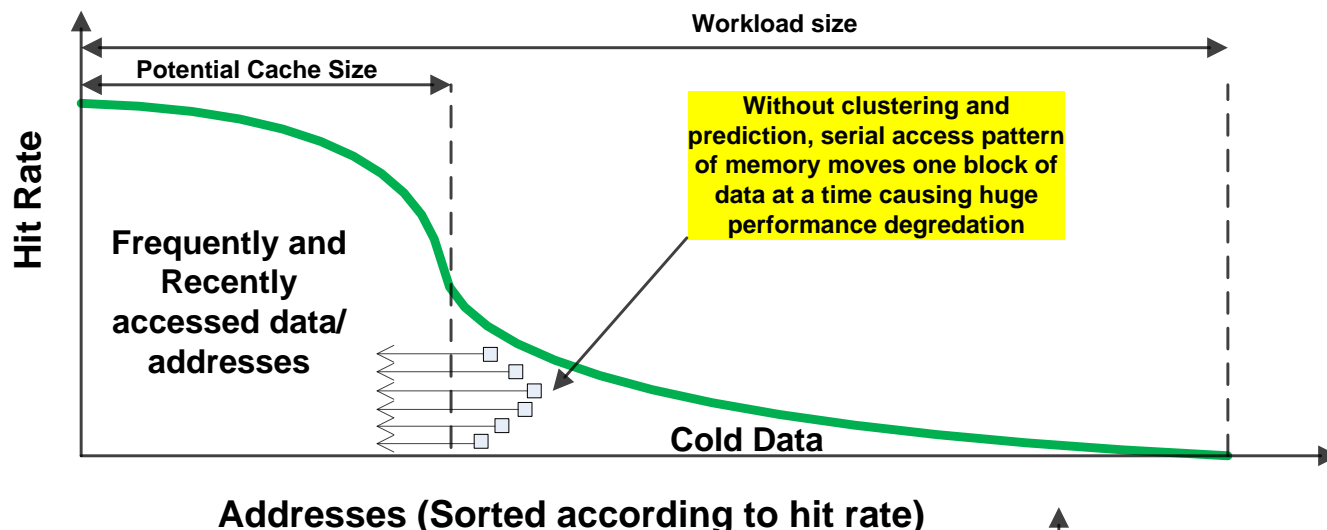
"Our characterizations show that datacenter applications can benefit from tiered memory systems as there exist opportunities for offloading colder pages to slower memory tiers. **Without efficient memory management, however, such systems can significantly degrade performance.**"

Analysis done by Mata: <https://arxiv.org/abs/2206.02878>

# Are Traditional Cache Management Techniques Good Enough?



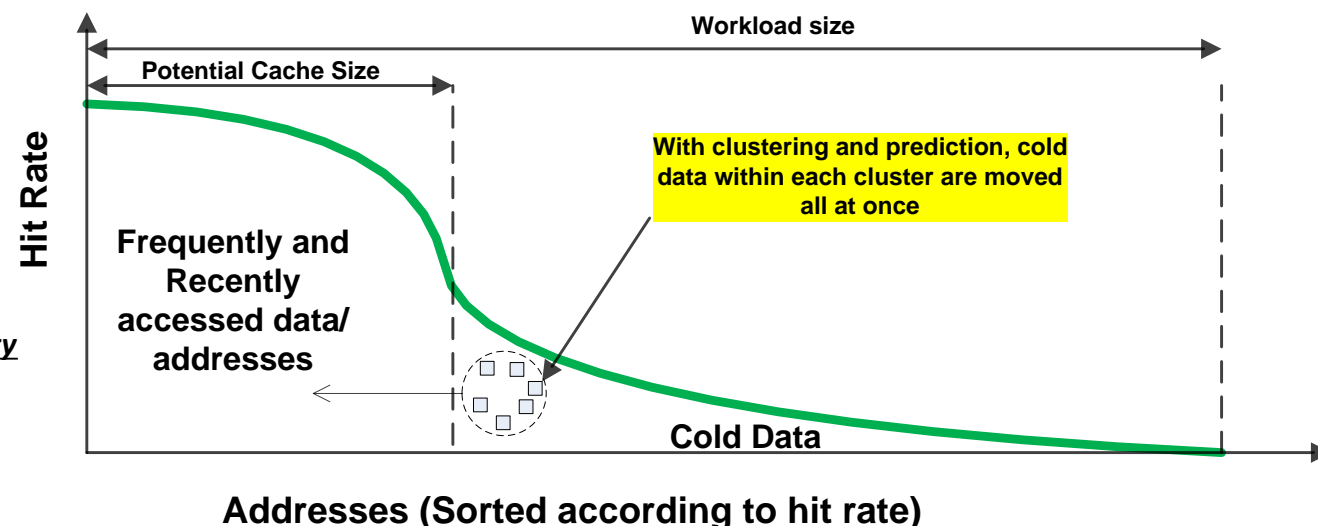
Flash Memory Summit



□ Cold Data

- Leveraging AI to detect patterns and create clusters amongst cold data allows for the parallelization access of cold data. As a result, the flash latency is amortized across each cluster size.
- Cluster learning and pattern detection addresses the long tail distribution of cold data and significantly reduces the impact of latency differential of multi tier memory system

- Traditional Cache Management algorithms are capable of placing frequently and recently accessed data into the fast tier memory and placing cold data into the slow tier memory
- Single thread memory access is a serial process (accesses one address at a time). As a result, if the latency differential between the fast and slow tier is large (like the case between DRAM and Flash), the cache performance will significantly suffer due to the long tail of cold data.
- Traditional cache management algorithms don't have a solution to long tail access distribution problem



□ Cold Data

# Challenges of Detecting Memory Access Patterns



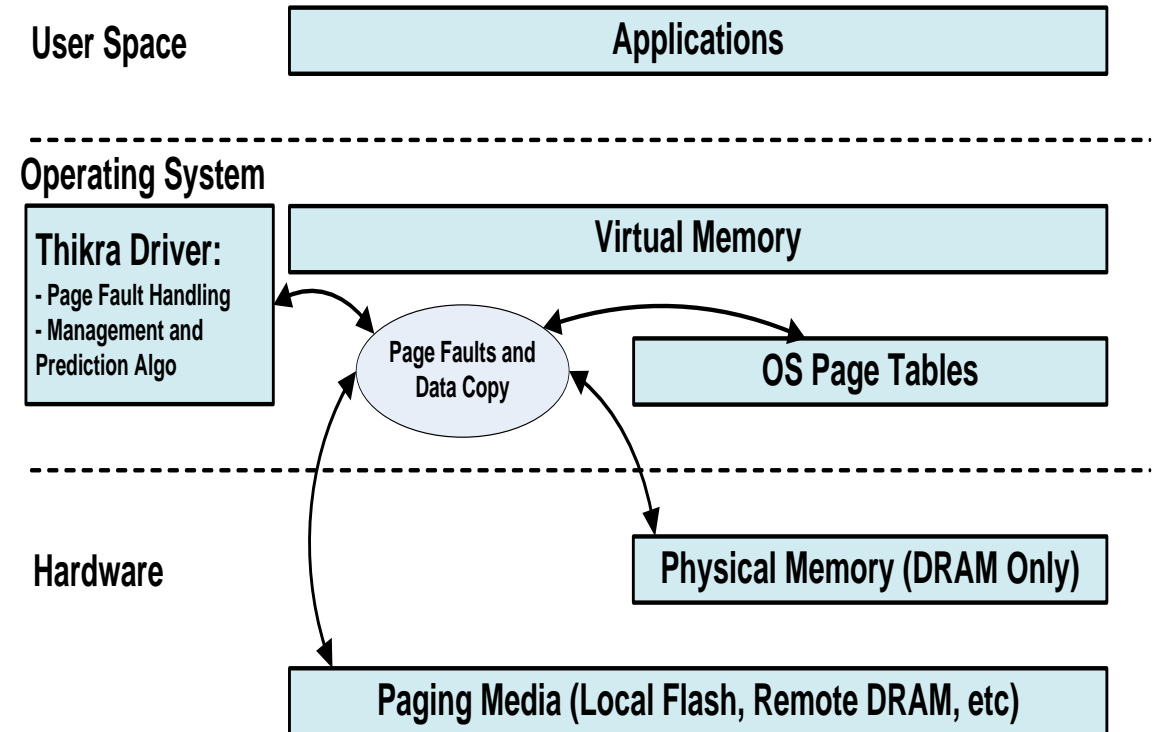
Flash Memory Summit

- Number of unique addresses is in the millions making collecting statistical information per address unrealistic/impossible
- Memory access patterns change so fast making it impossible to buffer the patterns and perform traditional clustering algorithms that relies on data being relatively static
- In multi-threaded environment it is impossible to isolate patterns from one thread to another (at the HW level). As a result, traditional pattern detection algorithms can't perform data cleaning phase to isolate relevant data from irrelevant one

# SW Prototype: THRIVER

- THRIVER is configured to keep a fixed number of concurrently mapped pages (MAX\_MAP/Cache size)
- In case of a FAULT, THRIVER:
  1. Un-map required number of pages to maintain (MAX\_MAP)
  2. Write unmapped pages to paging device
  3. Make prediction based on current FAULT
  4. Read missing page along with predicted pages from paging device
  5. Map missing and predicted pages in the PID page table
  6. Resume the application

\*Steps 2) and 4) can be skipped to test algorithm only





# Some Stats

#	Workload	WL Size (#4k Pages)	Cache Size (#4k Pages)	Clustered Pages	Cluster Size Avg (#4k Pages)	Prediction Accuracy (%)
1	Machine Learning (Neural Network Training)	360k	90k (25%)	300k (83%)	12	93%
2	HPC (Matrix Operations)	490k	70k (14%)	460k (94%)	53	>99%
3	Data Analytics (TPCH on SQLITE)	1.5M	300k (20%)	1.35M (90%)	50	96%
4	Random Access (synthetic benchmark)	400k	70k (18%)	800 (<1%)	0	NA
5	(1) + (2) + (4)	1.28M	210k (18%)	560k (60%)*	14	>99%

- ❑ Prediction Algorithm can achieve 93%-99% accuracy with various workloads
- ❑ Prediction Algorithm can associate 60%-94% of the workload size into clusters of average size between 12-53 pages per cluster
- ❑ Prediction Algorithm can detect random workload and correctly makes no associations between data structures
- ❑ Prediction Algorithm can handle mixed workload (random and predictable)

# Performance Analysis

Workload	DRAM /Flash	Faults Reduction	Prediction Rate	Prediction Accuracy	Performance Slow Down No Prediction	Performance Slow Down With Prediction	Performance Gap Reduction (Runtime)
OpenFoam (HPC)	50/50	5.7x	82%	96%	1.8x	1.1x	6.8x
OpenFoam (HPC)	25/75	10.2x	90%	95%	3.8x	1.5x	5.1x
OPM (Oil and Gas) (HPC)	50/50	8.9x	92%	98%	2.3x	1.3x	4.5x
ML Training	35/65	5.8x	80%	95%	3.4x	1.7x	3.4x
SQLite TPCH (Data Analytics)	50/50	10.1x	90%	93%	6.9x	1.6x	9.1x

Fault Reduction

= #Faults without Prediction / #Faults with Prediction

Prediction Rate

= #Predictions / (#Faults + #Predictions )

Performance Slow Down No Prediction

= No Prediction Runtime / Baseline Runtime\*

Performance Slow Down With Prediction

= With Prediction Runtime / Baseline Runtime\*

Performance Gap Reduction

= (No Prediction - Baseline) / (With Prediction - Baseline)

\*All data is in DRAM



THANK YOU