Memory disaggregation: why now and what are the challenges

Marcos K. Aguilera[1], *Emmanuel Amaro*[1], *Nadav Amit*[1], *Erika Hunhoff*[2], *Anil Yelam*[3], *Gerd Zellweger*[1]
[1]*VMware Research* [2]*University of Colorado, Boulder* [3]*UC San Diego*

Abstract

Hardware disaggregation has emerged as one of the most fundamental shifts in how we build computer systems over the past decades. While disaggregation has been successful for several types of resources (storage, power, and others), memory disaggregation has yet to happen. We make the case that the time for memory disaggregation has arrived. We look at past successful disaggregation stories and learn that their success depended on two requirements: addressing a burning issue and being technically feasible. We examine memory disaggregation through this lens and find that both requirements are finally met. Once available, memory disaggregation will require software support to be used effectively. We discuss some of the challenges of designing an operating system that can utilize disaggregated memory for itself and its applications.

1 Introduction

Hardware disaggregation, or simply *disaggregation*, means separating hardware resources (e.g., disks, GPU, memory) that have been traditionally combined in a single server enclosure. Disaggregation has physical and logical implications. Physically, the disaggregated resource is placed in a separate box or chassis, increasing its distance from other resources. Logically, the disaggregated resource becomes less coupled with the rest of the system.

Figure 1 illustrates memory disaggregation: the memory moves from the confines of a host into a memory pool, which can then be accessed by multiple servers. This provides two benefits: servers can access large amounts of memory in the pool (more than a server can have locally) and servers can use the disaggregated memory to efficiently share data. Note one need not disaggregate all memory: some of it remains local memory in servers.

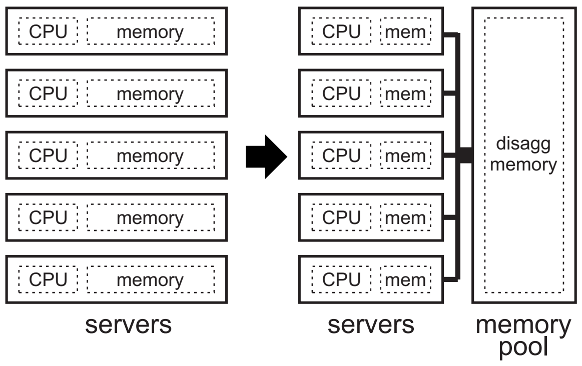

Figure 1: With traditional servers (left), we have CPU, memory, disks, and network adapters on a single enclosure. With memory disaggregation (right), we move some of that memory to a pool. Servers still have some local memory but they can also access the memory in the pool to obtain additional memory or share data.

Memory disaggregation originated from ideas in the 1990s [11], and Intel proposed a commercial architecture in 2013 [16]. Nevertheless, memory disaggregation has yet to happen in production systems.

To understand why, we look at history to study other forms of successful disaggregation: mainframe computers, storage, power and cooling, and GPUs (Section 2). From these efforts, we learn that disaggregation succeeds based on two requirements: the need to solve a burning issue and the availability of a feasible technology.

We argue that these two requirements are now met for memory disaggregation as big data increasingly moves to memory, and memory becomes an expensive resource; meanwhile, the emergence of Compute eXpress Link (CXL) provides technical feasibility (Section 3).

Once available, disaggregated memory can significantly benefit data-intensive systems, such as machine learning data pipelines, distributed computing frameworks, and database systems. These systems can leverage the two main capabilities of disaggregated memory (larger memories and data sharing across servers), but they will require appropriate operating system support. We discuss the challenges of building an operating system that can leverage disaggregated memory to benefit both applications and the operating system itself (Section 4).

2 Disaggregation: the past

Disaggregation brings different benefits and trade-offs that depend on the disaggregated resource. In the rest of this section, we discuss past successful disaggregation efforts and the lessons we can learn from them.

2.1 From mainframes to clusters

The transition from large mainframe computers to computer clusters in the 1970s and 1980s can be seen as an extreme form of disaggregation where many resources were simultaneously disaggregated: a monolithic box was separated into multiple minicomputers that worked together. Initially, these minicomputers were tightly coupled to provide the illusion of a single system (e.g., in a VAXcluster, one could uniformly handle processes, users, and disks from any computer in the cluster). Over time, minicomputers became less and less coupled with each other, losing their single-system transparency in favor of larger systems deployed across broader geographic locations (e.g., Unix clusters). This trend eventually led to systems that connect minicomputers in different organizations (often universities or national laboratories) over long-distance lines using modems at first, or dedicated lines later, to exchange email and access usenet newsgroups.

Here, disaggregation was motivated initially by cost and scale (mainframes were expensive) and later by accessibility (users around the country wanted access to computers, but mainframes were only available in a few locations, while the smaller computers could be spread around). Ultimately, mainframe disaggregation led to the creation of the Internet.

2.2 Storage disaggregation

Storage disaggregation followed a gradual path starting in the early 1990s. It started with the realization that if a computer crashes, all data in its locally attached disks become inaccessible. To address this problem and improve data availability, the industry introduced dual-ported SCSI disks, which could be connected to two computers simultaneously. This idea evolved into storage area networks (SANs), where disk arrays are attached to a network fabric connected to many computers which can access any disk. The network fabric was initially dedicated for storage (Fibre Channel) but later proposals used general-purpose networks (iSCSI, NVMe over Fabrics).

Although storage disaggregation was initially motivated by improved data availability, it later brought other benefits: manageability (it is easier to manage a central pool of storage than storage spread over all computers), reliability (the central storage pool became sophisticated fault-tolerant disk arrays), and disaster tolerance (disk arrays could copy data to other disk arrays across wide area networks).

2.3 Power and cooling disaggregation

Power supplies and fans are clunky components that tend to fail often. Thus, for reliability, servers tend to have two sets of power supplies and fans, but this is a significant waste of space and money: a rack with 40 servers would require 80 power supplies and fans. It makes much more sense to disaggregate the power supplies and fans so that many servers can share a few. Blade servers accomplished this in the 2000s, where a single chassis with its own power supplies and fans housed multiple servers. This idea has motivated the rack designs in the Open Compute Project [25] and Open19 [26], which are used in today's hyperscalers: each rack has power supplies and fans, and rack servers take DC power as input.

Power and cooling disaggregation was initially motivated by density and cost in blade servers, and it eventually led to new data center designs.

2.4 GPU disaggregation

Graphics Processing Units, or GPUs, started as graphics cards that produced a computer's video output. Later, GPUs gained general-purpose acceleration support for Single Instruction Multiple Data computations (e.g., vector and matrix operations), often used in machine learning algorithms. Modern GPUs are extremely powerful and have a much higher degree of parallelism than processors by several orders of magnitude. This power comes at a high cost in price and power consumption. If many servers need access to GPUs, it is expensive to provision each server with its own GPU. If those GPUs are used at different times, their utilization will be low. With

GPU disaggregation, the GPUs in a pool are accessible to many servers, so one can provision a smaller number of GPUs and increase their utilization. This idea was first implemented in software in the mid-2010s by redirecting CUDA calls to a remote server housing the GPU (e.g., Bitfusion [6]). Later, composable infrastructure systems gained GPU disaggregation support (e.g., HPE Synergy, Dell PowerEdge MX, Supermicro SuperCloud Composer, Lenovo ThinkAgile CP).

GPU disaggregation was motivated by utilization and cost, and we believe the future will bring other benefits as the idea gains further adoption.

2.5 Lessons

Looking at the past success of hardware disaggregation, we identify some common patterns. We observe that there are two requirements for disaggregation of a resource to succeed. First, it needs to address a *burning issue*; that is, it needs a powerful, compelling motivation: cost and scalability for mainframe disaggregation, data availability for storage disaggregation, density and cost for power and cooling disaggregation, and utilization and cost for GPU disaggregation. A burning issue is required because disaggregation needs significant and simultaneous investment from many parties: chip makers, device makers, systems integrators, and sometimes operating system and application vendors. These parties must overcome significant inertia to build solutions that work well together, and this is only possible with a strong use case. The second requirement to disaggregate a resource is a *feasible technology*, which often requires addressing nontrivial technical problems at many levels of the system stack: cluster systems, remote storage systems, blades, composable hardware, etc.

Another lesson we learn from the past is that, once disaggregation happens, it brings a much broader impact than originally anticipated. Mainframe disaggregation led to a revolution in distributed systems; storage disaggregation created an entire industry of storage appliances; power and cooling disaggregation led to new data center designs; and GPU disaggregation is still ongoing so the jury is still out, but it will possibly create new computing paradigms for processing of data pipelines.

3 Memory disaggregation: why now?

Memory disaggregation originated with paging to remote memory in the early 1990s [11]. The network had become fast enough that it was more efficient to page to remote memory using the network than to a local disk. These initial efforts provided a rudimentary form of memory disaggregation: a processor accessing disaggregated memory incurs a page fault, bringing the data to local memory before the processor can resume execution. Full-fledged memory disaggregation was first proposed in [21], where a processor can issue loads and stores directly to disaggregated memory, without incurring page faults, thus improving performance.

While memory disaggregation is an old idea, it has not yet been commercially successful. As we mentioned before, Intel proposed a Rack Scale Architecture in 2013 [16] that included memory disaggregation. However, the proposal did not succeed because it was based on the technology of silicon photonics [32], which has not yet prevailed.

That begs the question: why is the right time for memory disaggregation now? We next explain why we believe that is the case, by arguing that memory now satisfies the two requirements for successful disaggregation, as described in Section 2.5: addressing a burning issue and the existence of feasible technology.

Burning issue. Memory is increasingly becoming a problematic resource in computing systems, due to four reasons. First, the need for memory is surging as big data moves to memory for faster processing in a broad range of systems and applications, such as in-memory databases, data analytics, streaming analytics, and machine learning. Second, the memory capacity of individual servers is constrained due to physical limitations in the density of memory DIMMs and the number of DIMMs that can be connected to a processor. Third, over time, applications need more memory but upgrading memory is notoriously challenging due to the stringent memory population rules for servers [1]—for example, all memory controllers should have the same number and sizes of DIMMs, and all memory channels should have the same capacity. Failure to follow these rules results in poor memory bandwidth. Fourth, the cost of memory is growing due to the proliferation of new use cases: smart vehicles, 5G phones, gaming consoles, and data centers. Meanwhile, supply remains restricted by an oligopoly of only three memory companies (Samsung, SK Hynix, Micron). As a result, memory can dominate the cost of the system. In fact, cloud operators report that memory can constitute 50% of server cost [20] and 37% of total cost of ownership [23].

Feasible technology. The technology that enables disaggregated memory has significantly advanced. Commercially, Remote Direct Memory Access (RDMA) has become commodity with technologies such as RDMA over Converged Ethernet (RoCE), which provides RDMA on Ethernet networks. Furthermore, standards for disaggregated memory are emerging, initially through the GenZ Consortium [12] and more recently through Compute eXpress Link (CXL [9]). CXL allows devices to provide cache-coherent memory on the PCIe bus; version 1 of CXL enables local memory expansion cards while subsequent versions will enable memory disaggregation via cards connected to memory pools. Research prototypes that leverage these technologies, including The Machine [18, 34] based on Gen-Z, and DirectCXL [13] and Pond [20] based on CXL, demonstrate the feasibility of disaggregated memory. Moreover, CXL version 1 is already supported by the latest server processors by Intel and AMD, while device vendors gradually introduce local memory expansion cards. We anticipate CXL memory disaggregation to come soon after (say, in 3–5 years) following a similar trajectory as storage disaggregation: beginning with multi-ported memory connected to a few hosts (say, 2–8), followed by CXL switches on dedicated memory fabrics connecting memory to a rack of servers, and ultimately reaching convergence of the network and memory fabric (e.g., CXL over Ethernet or IP). A noteworthy feature of CXL is that it will allow the memory pool to use different memory technology from ones locally attached to servers. Consequently, as servers evolve to use the newer DDR5 memory, the pool can be provisioned with older DDR3 and DDR4 memory that is incompatible or too slow be used in the server.

4 An operating system for disaggregated memory

We are developing a new operating system (OS) that offers first-class support for disaggregated memory. Our goal is to allow the OS and applications to take full advantage of the capabilities of disaggregated memory: greater memory capacity and the ability to share data efficiently across servers. We refer to the union of the servers and the disaggregated memory as the *cluster*. This new OS differs from traditional distributed OSes in two key ways. First, we want the OS to conveniently expose disaggregated memory to applications so they can benefit from it. Second, the OS itself can use disaggregated memory to improve its design—for example, by sharing kernel data structures across servers.

Disaggregated memory allows the OS to provide the illusion of a single system across the cluster to applications. In particular, the OS will allow processes to have threads running on different servers, and these threads can access a unified file system. In addition, as processes can span multiple servers, we need a communication stack that can provide a unified view across the cluster, so an external client can reach application processes regardless of which servers are running them.

In terms of disaggregated memory management, the OS will offer both transparent and targeted memory allocations. A transparent allocation can be served from either local or disaggregated memory (i.e., the process does not care), while a targeted allocation chooses between local memory or the memory pool. Transparent allocations allow disaggregated memory to be used as an extension of local memory. In contrast, a targeted allocation provides faster memory (if allocating locally) or shareable memory (if allocating from the pool).

Providing first-class support for shareable disaggregated memory will benefit data-intensive distributed applications, which often incur significant overheads when sharing data because they must rely on traditional network stacks (e.g., RPC, TCP) and pay the costs of serialization, deserialization, and copying [17]. Concretely, we target data processing systems (e.g., machine learning pipelines [15]), distributed computing frameworks (e.g., Spark [37], Ray [35]), cluster schedulers (e.g., Kubernetes), and some traditional applications such as database systems (SQL and NoSQL), web servers, and file servers. For example, a distributed computing framework can optimize object movement between tasks by using disaggregated memory; a database system doing distributed query processing may create shared intermediate data and materialized views in disaggregated memory; a cluster scheduler also needs to share executables, job inputs, and job outputs; a web server or file server can share caches; and all of these systems can benefit from a larger memory capacity that disaggregated memory can provide.

4.1 Challenges

Building an OS for disaggregated memory raises a number of challenges.

Memory allocation. When a process allocates memory, the OS must decide from where. If the application does not care if the memory is local or disaggregated, the OS must pick a type based on memory availability and locality. This presents a memory tiering problem, where two types of memory offer different trade-offs: local memory is faster and private, while disaggregated memory is larger and shared. Even if the application makes a targeted request for local or disaggregated memory, the OS needs appropriate mechanisms to organize the memory. Traditional schemes such as buddy allocators and slab allocators need to be revisited since allocation requests will now be distributed across servers and must scale to large memory sizes [22] and a large number of processes, while keeping fragmentation under control. A solution is to handle all allocations centrally at one of the servers; another solution is to allocate memory in a distributed fashion, which requires efficient coordination across servers. These solutions trade off simplicity, global optimality, allocation latency, and fragmentation. Memory migration presents a related problem: the OS can move physical memory (while keeping virtual addresses) to remove fragmentation and improve locality (e.g., a process has freed some local memory so that data in disaggregated memory can be migrated locally for faster access). Memory migration requires policies of when and what to migrate, analogous to non-uniform memory access (NUMA) migration policies.

Scheduling. Where should processes and their threads execute? This decision depends on several factors. First, the servers eligible to schedule a thread depend on their resource availability (CPU, local memory, etc.). Second, processes that share buffers in disaggregated memory can benefit from cache locality if they are scheduled on the same server or a small number of servers; this affinity needs to be considered. Third, the OS should provide a balanced consumption of per-server CPU and memory; to do so, the scheduler and memory allocator must collaborate. Fourth, the memory pool may have memory regions with different distances to a server, and it may have regions that are only accessible to some of the servers due to the topology of the memory fabric; this is another case where the scheduler and allocator must coordinate. A scheduler design that weighs these factors can be centralized (as we can centralize the allocator), but a distributed design can scale better and provide lower latency.

Addressing. Disaggregated memory will have a pool memory controller that connects the pool memory to the servers. Servers will map virtual addresses to physical addresses, and the pool controller will map physical addresses to pool addresses. Translating addresses in the pool controller allows the system to easily move memory within the pool (without it, all servers using a shared buffer would have to fix their mappings when memory is moved), which in turn is useful for maintenance (e.g., adding or removing DIMMs to the pool) and optimization (e.g., for pools that have regions with different memory types). This creates many questions: what should the granularity of these mappings be, what is the underlying mechanism used to maintain these mappings, and who will maintain the mappings. Answering these questions will require a co-design of the pool controller (which performs the actual translations) and the OS (which provides the functional requirements). Another related challenge is how to correctly share pointers to shared buffers across servers. A simple solution is to map the buffers to the same virtual address at all servers, but this solution may not scale well as it requires memory allocations to return unique virtual addresses across processes in the cluster (if virtual addresses are not unique, buffers cannot be shared between processes).

OS state. Operating systems keep many types of internal state (e.g., process tables, device state, network connections), which raises three challenges for a disaggregated memory OS. First, we must identify the state needed locally at each server and the state that must be shared across servers. For example, some memory allocation metadata should be shared because we would like different processes to communicate using disaggregated memory; similarly, some OS scheduler state should be shared to support efficient process synchronization mechanisms (e.g., wait queues for locks being held). On the other hand, device state is local because we do not wish to expose devices across servers. An exception for this is the storage and network devices, which we address below.

The second challenge is efficiently coordinating access by different servers. Although this problem exists in traditional operating systems, it is more severe when using disaggregated memory since the coordination cost is higher (disaggregated memory is slower than local memory), and more parties can coordinate (thousands of cores across all CPUs in all servers). In theory, RCU, wait-free data structures, or fine-grained locking mechanisms can be used to tackle this challenge. However, with much higher memory latencies and number of cores, even the most sophisticated approaches eventually succumb under contention. Therefore, we need radically new approaches to manage OS state that scales well for reads and writes across machines. Partitioning (for writes) and replication (for reads) are often employed strategies to solve these challenges. Replication trades off extra memory consumption for more performance so ideally it can be applied in a dynamic way, using only memory the OS can currently spare.

The third challenge is that the larger capacity of disaggregated memory will result in much larger kernel data structures, to the point that they will require their own memory management mechanisms (e.g., memory migration). These mechanisms are normally used on user memory and they can be hard to apply to kernel memory (e.g., migration of kernel memory can cause deadlocks as kernel pages get write-protected and subsequently cause a page fault).

File system. To provide a single file system view across all servers, a simple solution is to mount a network file system (e.g., NFS, CIFS) on all servers. However, this approach misses an opportunity to leverage disaggregated memory. File systems can use the larger capacity of disaggregated memory for caching (buffer cache, page cache, inode cache, etc.), and they can use the sharing ability of disaggregated memory to share these caches across servers to improve cache utilization and reduce disk IO—if a server reads a file, another server can find its contents in the disaggregated memory cache. Designing a disaggregated memory file system raises the question of how to keep the various in-memory file system data structures, how to synchronize access to these structures, and how to schedule IO on disk. The result will be a type of distributed file system that is uniquely designed for disaggregated memory.

External communication. In addition to local network endpoints for each server, we believe the OS should provide network endpoints for the entire cluster. That is, the cluster should have an external IP address (or a few IP addresses) where communication with that IP on a given port is transparently mapped to a server in the cluster. According to application needs, the target server can be fixed or it can be chosen from a number of servers that balance load among themselves. This functionality is useful for implementing scalable and highly available services, such as web servers, file servers, etc. To implement this feature, the OS leverages disaggregated memory to maintain shared network state (e.g., which processes are bound to which ports) and an efficient way to coordinate shared access.

Failures. Disaggregated memory can experience partial or even total failures as the pool crashes or loses connectivity to the servers. The OS needs to handle such failures gracefully by containing the problem to the parts of the system that used the failed memory. It may be necessary to kill processes that lost data, but not necessarily take down the entire OS. Moreover, the OS will support the notion of *optional memory*, which is memory that is not vital for the execution of the process, such as caches or reconstructible data. If a failure affects only optional memory of a process, the process receives an exception and continues running. Finally, the OS needs to cleanly decouple the data structures that it keeps in disaggregated memory to minimize the blast radius. In some cases, it could make sense to keep critical kernel data structures entirely in local memory.

Security. We need to find a suitable trust model for the cluster. In a typical OS, the kernel is trusted and processes are isolated from each other. Our OS can offer defense in depth by providing isolation of access to the (shared) disaggregated memory; in particular, there should be isolation between processes in different servers and perhaps even between kernels in different servers. A related issue is to provide address space isolation in disaggregated memory for locations that are not being shared across servers. Here, again, is an opportunity for a co-design of the disaggregated memory controller and OS, where the controller provides isolation mechanisms (e.g., access control for servers) while the OS sets the policy. Enclaves play an important role as they can provide a thin trusted computing base across servers, which can be used to control the hardware mechanisms. Disaggregated memory encryption is another interesting capability and even more relevant than encryption of local memory, as we expect hot-swap of disaggregated memory similar to hot-swap of disks, where it is desirable to protect data of memory being taken out (DRAM chips keep data residues for some time after they are powered down, which can lead to known attacks).

5 Other related work

LegoOS [31] is an OS designed for hardware disaggregation of all resources (not just memory) based on the notion of loosely coupled monitors. Our goal is different, as we are focused on disaggregated memory, specifically on how to build an OS that allows applications to best leverage shareable disaggregated memory. The notion of a single system image across a cluster is provided by cluster OSes (e.g., VMS [27] for VAXclusters); we believe it is worth revisiting these design with disaggregated memory and modern hardware in mind.

Firebox [3] and dReDBox [5] provide hardware platforms that disaggregate multiple resources, which further motivates OSs designed for disaggregation. Work on HPE's The Machine included not just the hardware for persistent disaggregated memory, but also software support [18] and OS challenges [10], which overlap some of the challenges we describe but with a different perspective (e.g., due to the fact that disaggregated memory is persistent or due to the specifics of their hardware design). Unfortunately, a detailed write-up of their OS design is not yet available.

Memory vendors are providing toolkits (e.g., [33]) to utilize CXL memory in existing OSes. This effort is adequate for CXL memory expansion cards—which provide additional local memory—but we believe shareable disaggregated memory will require a new OS or significant OS modifications.

Remote memory access using RDMA has become widely available with RoCE [29] and have enabled systems that provide disaggregated memory through paging [1, 14], user libraries [30, 38], or by leveraging cache coherence mechanisms [8]. These systems provide one benefit of disaggregated memory (larger memory capacity) but does not allow processes in different servers to share memory. Some of these systems use software-based solutions that are likely to underperform the hardware-based disaggregation that is enabled by CXL.

Recent memory tiering systems have focused on mechanisms akin to NUMA migration with the goal of maintaining hot subsets of memory in fast DRAM while migrating cold subsets to slower locally-attached memory (e.g., NVM) [23, 28, 36]. Therefore, these systems neither present disaggregated memory to applications, nor do they consider sharing of such memory across processes in multiple servers.

Disaggregated memory can be seen as a form of distributed shared memory [2, 4, 7, 19, 24], which has been extensively studied decades ago and recently. There is an important lesson to learn from this body of work: cache coherence is extremely hard to scale. We thus believe cache-coherent disaggregated memory systems will be limited to a rack or a few racks of servers.

6 Conclusion

Memory disaggregation will finally become a reality, enabled by the emergence of CXL and its adoption by suppliers of memory, chipsets, controllers, servers, operating systems, and software. This industry alignment is driven by the memory problems faced by distributed data-intensive applications. Once available, memory disaggregation will introduce many OS challenges that must be addressed to best use this technology for memory expansion and memory sharing across servers.

References

[1] Emmanuel Amaro, Christopher Branner-Augmon, Zhihong Luo, Amy Ousterhout, Marcos K Aguilera, Aurojit Panda, Sylvia Ratnasamy, and Scott Shenker. Can far memory improve job throughput? In *European Conference on Computer Systems*, pages 1–16, April 2020.

[2] Cristiana Amza, Alan L. Cox, Shandya Dwarkadas, Pete Keleher, Honghui Lu, Ramakrishnan Rajamony, Weimin Yu, and Willy Zwaenepoel. TreadMarks: Shared memory computing on networks of workstations. *IEEE Computer*, 29(2):18–28, February 1996.

[3] Krste Asanović. FireBox: A hardware building block for 2020 Warehouse-Scale computers. In *USENIX Conference on File and Storage Technologies*, February 2014. Keynote talk.

[4] J. K. Bennett, J. B. Carter, and W. Zwaenepoel. Munin: Distributed shared memory based on type-specific memory coherence. In *ACM Symposium on Principles and Practice of Parallel Programming*, pages 168–176, March 1990.

[5] Maciej Bielski, Ilias Syrigos, Kostas Katrinis, Dimitris Syrivelis, Andrea Reale, Dimitris Theodoropoulos, Nikolaos Alachiotis, Dionisios N. Pnevmatikatos, Evert H. Pap, Georgios Zervas, Vaibhawa Mishra, Arsalan Saljoghei, Alvise Rigo, Jose Fernando Zazo, Sergio Lopez-Buedo, Martí Torrents, Ferad Zyulkyarov, Michael Enrico, and Oscar Gonzalez de Dios. dReDBox: Materializing a full-stack rack-scale system prototype of a next-generation disaggregated datacenter. In *Design, Automation & Test in Europe Conference & Exhibition*, pages 1093–1098, March 2018.

[6] VMware Bitfusion. https://core.vmware.com/bitfusion.

[7] Qingchao Cai, Wentian Guo, Hao Zhang, Divyakant Agrawal, Gang Chen, Beng Chin Ooi, Kian-Lee Tan, Yong Meng Teo, and Sheng Wang. Efficient distributed memory management with RDMA and caching. *Proceedings of the VLDB Endowment*, 11(11):1604–1617, July 2018.

[8] Irina Calciu, M. Talha Imran, Ivan Puddu, Sanidhya Kashyap, Hasan Al Maruf, Onur Mutlu, and Aasheesh Kolli. Rethinking software runtimes for disaggregated memory. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, April 2021.

[9] Compute eXpress Link. https://www.computeexpresslink.org.

[10] Paolo Faraboschi, Kimberly Keeton, Tim Marsland, and Dejan Milojicic. Beyond processor-centric operating systems. In *Workshop on Hot Topics in Operating Systems*, May 2015.

[11] E. Felten and J. Zahorjan. Issues in the implementation of a remote memory paging system. Technical Report CSE TR 91-03-09, University of Washington, March 1991.

[12] Gen-Z consortium. https://en.wikipedia.org/wiki/Gen-Z_(consortium).

[13] Donghyun Gouk, Sangwon Lee, Miryeong Kwon, and Myoungsoo Jung. Direct access, high-performance memory disaggregation with DirectCXL. In *USENIX Annual Technical Conference*, pages 287–294, June 2022.

[14] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G Shin. Efficient memory disaggregation with Infiniswap. In *Symposium on Networked Systems Design and Implementation*, pages 649–667, March 2017.

[15] Hannes Hapke and Catherine Nelson. *Building Machine Learning Pipelines*. O'Reilly Media, Inc, July 2020.

[16] Intel rack scale architecture. https://www-conf.slac.stanford.edu/xldb2016/talks/published/Tues_6_Mohan-Kumar-Rack-Scale-XLDB-Updated.pdf.

[17] Svilen Kanev, Juan Pablo Darago, Kim Hazelwood, Parthasarathy Ranganathan, Tipp Moseley, Gu-Yeon Wei, and David Brooks. Profiling a warehouse-scale computer. In *International Symposium on Computer Architecture*, pages 158–169, June 2015.

[18] Kimberly Keeton. Memory driven computing. In *USENIX Conference on File and Storage Technologies*, February 2017. Keynote presentation.

[19] Seung-seob Lee, Yanpeng Yu, Yupeng Tang, Anurag Khandelwal, Lin Zhong, and Abhishek Bhattacharjee. MIND: In-network memory management for disaggregated data centers. In *ACM Symposium on Operating Systems Principles*, pages 488–504, October 2021.

[20] Huaicheng Li, Daniel S. Berger, Stanko Novakovic, Lisa Hsu, Dan Ernst, Pantea Zardoshti, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, Mark D. Hill, Marcus Fontoura, and Ricardo Bianchini. Pond: CXL-based memory pooling systems for cloud platforms. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, March 2023.

[21] Kevin Lim, Jichuan Chang, Trevor Mudge, Parthasarathy Ranganathan, Steven K Reinhardt, and Thomas F Wenisch. Disaggregated memory for expansion and sharing in blade servers. *ACM SIGARCH Computer Architecture News*, 37(3):267–278, June 2009.

[22] Mark Mansi and Michael M. Swift. 0sim: Preparing system software for a world with terabyte-scale memories. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, page 267–282, March 2020.

[23] Hasan Al Maruf, Hao Wang, Abhishek Dhanotia, Johannes Weiner, Niket Agarwal, Pallab Bhattacharya, Chris Petersen, Mosharaf Chowdhury, Shobhit Kanaujia, and Prakash Chauhan. TPP: Transparent page placement for CXL-enabled tiered-memory. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, page 742–755, March 2023.

[24] Jacob Nelson, Brandon Holt, Brandon Myers, Preston Briggs, Luis Ceze, Simon Kahan, and Mark Oskin. Latency-tolerant software distributed shared memory. In *USENIX Annual Technical Conference*, pages 291–305, July 2015.

[25] Open compute project. https://www.opencompute.org.

[26] Open19. https://www.open19.org.

[27] OpenVMS. https://en.wikipedia.org/wiki/OpenVMS.

[28] Amanda Raybuck, Tim Stamler, Wei Zhang, Mattan Erez, and Simon Peter. HeMem: Scalable tiered memory management for big data applications and real NVM. In *ACM Symposium on Operating Systems Principles*, pages 392–407, October 2021.

[29] RDMA over Converged Ethernet. https://en.wikipedia.org/wiki/RDMA_over_Converged_Ethernet.

[30] Zhenyuan Ruan, Malte Schwarzkopf, Marcos K Aguilera, and Adam Belay. AIFM: High-performance, application-integrated far memory. In *Symposium on Operating Systems Design and Implementation*, pages 315–332, November 2020.

[31] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. LegoOS: A disseminated, distributed os for hardware resource disaggregation. In *Symposium on Operating Systems Design and Implementation*, pages 69–87, October 2018.

[32] Silicon photonics. https://en.wikipedia.org/wiki/Silicon_photonics.

[33] Scalable memory development kit. https://github.com/OpenMPDK/SMDK.

[34] Paul Teich. HPE powers up The Machine architecture, January 2017. https://www.nextplatform.com/2017/01/09/hpe-powers-machine-architecture.

[35] Stephanie Wang, Eric Liang, Edward Oakes, Benjamin Hindman, Frank Sifei Luan, Audrey Cheng, and Ion Stoica. Ownership: A distributed futures system for fine-grained tasks. In *Symposium on Networked Systems Design and Implementation*, pages 671–686, April 2021.

[36] Zi Yan, Daniel Lustig, David Nellans, and Abhishek Bhattacharjee. Nimble page management for tiered memory systems. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 331–345, April 2019.

[37] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *Workshop on Hot Topics in Cloud Computing*, June 2010.

[38] Yang Zhou, Hassan MG Wassel, Sihang Liu, Jiaqi Gao, James Mickens, Minlan Yu, Chris Kennelly, Paul Turner, David E Culler, Henry M Levy, et al. Carbink: Fault-tolerant far memory. In *Symposium on Operating Systems Design and Implementation*, pages 55–71, July 2022.