Separating Translation from Protection in Address Spaces with Dynamic Remapping

Reto Achermann
ETH Zurich

Chris Dalton
Hewlett Packard Labs

Paolo Faraboschi
Hewlett Packard Labs

Moritz Hoffmann
ETH Zurich

Dejan Milojicic
Hewlett Packard Labs

Geoffrey Ndu
Hewlett Packard Labs

Alexander Richardson
University of Cambridge

Timothy Roscoe
ETH Zurich

Adrian L. Shaw
Hewlett Packard Labs

Robert N. M. Watson
University of Cambridge

Abstract

It is time to reconsider memory protection. The emergence of large non-volatile main memories, scalable interconnects, and rack-scale computers running large numbers of small "micro services" creates significant challenges for memory protection based solely on MMU mechanisms. Central to this is a tension between protection and translation: optimizing for translation performance often comes with a cost in protection flexibility.

We argue that a key-based memory protection scheme, complementary to but separate from regular page-level translation, is a better match for this new world. We present MaKC, a new architecture which combines two levels of capability-based protection to scale fine-grained memory protection at both user and kernel level to large numbers of protection domains without compromising efficiency at scale or ease of revocation.

1 INTRODUCTION

For the last few decades, memory protection in computer systems has been performed by the MMU alongside page-based translation. This has worked well in systems with 10s of cores and memory sizes of up to a few tens of gigabytes of – volatile – DRAM.

A new class of computer is emerging, however, which is very different. "Rack-scale" or "memory-centric" computer systems [3, 11, 21] have high core counts, extremely fast low-latency interconnects, and a large (petabytes) distributed "pool" of byte-addressable memory, most of which is persistent [11] (Fig. 1). Page-based MMUs are likely a poor match to these machines for translation or protection.

The problems with relying solely on the MMU for memory protection include:

(1) The very large physical address space (typically larger than the available virtual address space) may require changing translations without any change in the rights to access physical frames [10].
(2) Machines that soley rely on the MMU for protaction typically incorporate multiple levels of *physical* address translation between the MMU and memory [13, 18, 20] (see Figure 1) and/or remote memory copies such as RDMA. The MMU therefore lacks information about the eventual physical address itself, causing a disconnect between CPU and memory-side protection (and translation).
(3) Since memory, and the data therein, persists across process lifetimes and even reboots, so must the protection metadata for pages.
(4) Removing access rights to a page of data may become a challenge in a large machine with MMUs, since it requires identifying all page tables mapping the page and performing a distributed TLB shootdown.
(5) Since protection on each frame is applied in the MMU rather than close to memory, any code able to program the MMU is included in a single trusted computing base for the entire machine. For small machines this is not a problem,

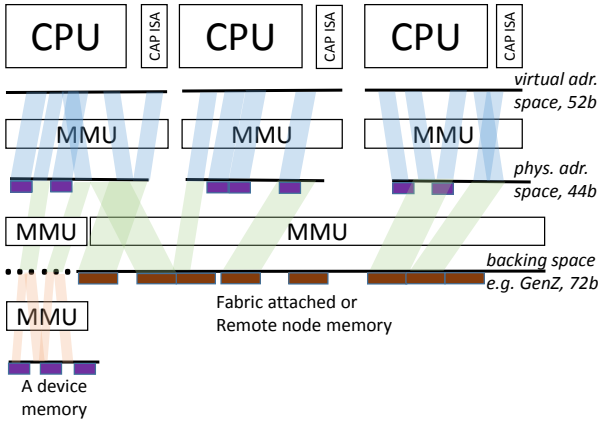


Figure 1: An of example of complex dynamically changing memory hierarchies

HotOS '17, Whistler, BC, Canada

978-1-4503-5068-6/17/05…$15.00
DOI: 10.1145/3102980.3103000

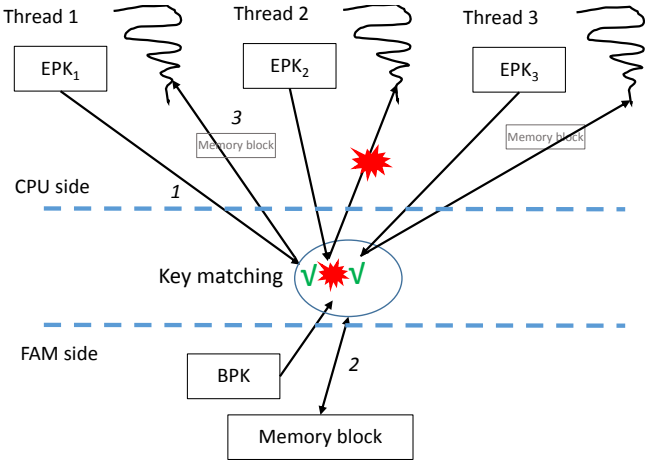


Figure 2: Each thread and each memory block have associated keys. EPK and BPK respectively. If there is a match, memory block is returned, otherwise a failure. Different thread keys can match the same block key.

for very large ones running multiple OSes and subject to transient partial hardware faults, it is a major concern.

This creates a dilemma. A single protection model covering the complete (executing and persisted) state of the machine is essential for correct programming and operation. Protection mechanisms close to cores, ideally "in front" of the MMU (as with CHERI [37]) are desirable for ensuring individual threads are protected and that runtime bugs are isolated. At the same time, protection mechanisms near memory itself are required for the scalability and assurance reasons described above.

Matching Key Capabilities or MaKC is a novel capability security model which resolves this dilemma. User processes hold keys authorizing access to blocks of memory (independent of page translation), while memory blocks themselves have an associated key. When a memory access occurs, the keys must "match" (for some suitable matching predicate) for the access to succeed. Since loading both kinds of keys is performed by different parts of the system, a variety of distributed trust models can be implemented both on the CPU side or on the memory side (see Figure 2).

MaKC is amendable to a number of different implementations (offering different levels of assurance), ranging from a software-only approach using existing MMUs to a (more practically useful) implementation with hardware support in processors and memory controllers.

2 USE CASES

MaKC has far-reaching potential to address many security concerns that arise from the complex memory architectures that are emerging in next-generation systems. In this section we highlight a few individual use cases to show the most promising directions where we envision MaKC could appear in the near future.

Fabric attached memory (FAM) is a recently proposed memory architecture that advocates sharing memory, not necessarily through a coherent protocol, across a rack-scale fabric [4, 17, 28]. Each node on the fabric can map part of the FAM on-demand into its physical address space, and then use it as regular memory. Physical-to-fabric memory mappings can change at any time, for example when permissions are revocated or memory reassigned. However, because end-user applications only understand virtual addresses, they may have kept pointers to FAM locations that have become stale, and can cause memory corruption, errors, or security compromises. MaKC solves these issues, by comparing the keys before de-referencing the pointer and hence detecting pointer validity in time to prevent dangerous accesses.

RDMA is a standard networking technique that provides direct access to the memory of a remote node, mediated by the network interface at the memory's home node. Memory regions need to be registered before they can be used to obtain a handle (*rkey*) that can be exchanged with the RDMA clients and used to prevent data corruption and accesses beyond the registered address range. Memory regions also need to be pinned (i.e., their virtual-to-physical mapping fixed and locked) so that an RDMA access cannot unintentionally (after a page swap) access another application data segment. Using MaKC extends the *rkey* mechanism by matching the keys and raising an exception if they do not match.

Distributed data access to services usually requires an agent that mediates access to the data, through a strict API and secure handles. While this approach works today it comes with the overhead of a software layer that may not be compatible with the tightly coupled high-performance hardware mechanisms that are appearing in rack-scale systems. Mechanisms such as fabric-attached memory or RDMA do not require a mediator agent, or the associated software overheads, because they directly expose memory. However, objects still need to be protected at fine granularity, and that cannot be achieved efficiently using MMU and page-level granularity. Like CHERI, using MaKC to protect distributed object handles allows fine-grain protection without compromising performance.

Micro-services are foundational building blocks that can be used to compose complex distributed applications and are only responsible for a small, well-defined, function of the overall system. To reduce the overhead of a full virtual machines, micro-services are typically deployed in containers, which requires making some security tradeoffs. For example, a malicious container that manages to compromise the underlying kernel would compromise the security of all the other containers running under the same kernel. Using MaKC, memory accesses can still be authenticated by hardware using the matching key, thus preventing access from unwanted threads, regardless of the state of the kernel.

Persistent active objects. Objects residing in non-volatile (or fabric-attached) memory require methods for accessing the data. Transferring execution and controlling access to those active objects requires flexible and efficient fine grained protection mechanisms. Page-based protection only works at large granularity, while accessing objects through supervisor calls or RPCs adds unnecessary performance overhead. Like CHERI, MaKC provides fine grained memory protection and defined entry points (call gates) to access the objects, and can also support remapping, if an additional level of translation is needed, as it is the case for fabric-attached memory.

General intra kernel protection. Running device drivers in the kernel is a practice that has been known to be vulnerable to many security threats [12, 14]. Split kernels [27] try to protect

memory of one kernel subsystem from another by running security critical functions in an inner kernel. Outer kernels then request services through a well defined call interface. This setup usually involves hypervisor calls and nested paging introducing runtime overheads. Split kernels can rely on MaKC for protection, eliminating the need of hypervisor calls and nested pages.

3 BACKGROUND

Many memory protections have been proposed over the years as alternatives to page-based translation: segments, bounds registers, hardware capabilities, memory keys MaKC borrows ideas from some of these, such as key-based access and capabilities.

3.1 MMU-based

MMU-based page protection coupled access rights of a physical frame to the structure of the virtual address space. The access principal is therefore the process, and any rights held by the process are deleted when the process exits. Protection is set up by the kernel, which must therefore be trusted.

Above this mechanism, a variety of high-level protection measures can be implemented in addition to the basic Unix model. Mechanism can be separated from policy [29]. Microkernels, for example, isolate some access authority in server processes. Systems like seL4 [23] and Barrelfish [5] implement partitioned capability-based protection in which the kernel limits which frames can be mapped into an address space. Whereas Chorus [32] and Amoeba [30] rely on sparsity and cryptography to make capabilities unforgeable. Even within kernel mode, software components can be isolated by giving them an address space overlay and controlling all privileged MMU update operations as done in ConspicuOS [9].

It is not even necessary to tightly couple translation to protection when virtualization hardware is present: nested paging can be used to modify translations without altering the protection rights on frames [6].

Protection lookaside buffers (PLB) [25] separate translation from protection information in TLBs. The PLB caches the protection information a domain has for a specific virtual memory page [24]. Many of the desired aspects of PLB's were already implemented in PA-RISC [36]. For a successful address translation the process ID bits in the PLB must match.

However, all MMU-based protection models suffer the problems identified in Section 1: enforcement for a given page occurs close to each core, is distributed throughout the machine, relies on trusted software on every core in the system, and requires an additional (unspecified) mechanism for persisting metadata.

3.2 Hardware Capabilities

Systems like CAP [31], and StarOS [22] provide instruction set extensions and special registers for object level protection. Recently, new systems were developed to revisit hardware capabilities. Systems without compatibility requirements (e.g. M-Machine [7]) and with the ability to run normal programs (e.g. CHERI [37]) provide architecture-supported capabilities. They are interpreted by hardware as bounded *virtual* address pointers into tagged memory which determines whether the value stored is a capability or regular data. CHERI capabilities can therefore provide efficient fine-grained protection within an address space. Revocation is not a problem when using the MMU, process exit removes all rights and in a single address space model, garbage collection can be used to invalidate tag bits. Since CHERI capabilities apply to a virtual address space they cannot be shared between different address spaces or persisted across process invocations in their current form. Moreover, they sit between software and the MMU and have the same issues as MMU-based protection.

However, CHERI does associate protection rights with individual threads (via capability registers) independent of page tables, and so demonstrates how conventional user-space code can carry authorization information while executing and pass it through to the system hardware. MaKC adapts this scheme to implement the user-facing aspect of its capability system.

CODOMs [34] is similar to our work in that it integrates keys with pages and combines keys with capabilities. Target architectures and goals are different. Our work is primarily focused on rack scale systems and persistent memory with capability enforcement close to the memory. CODOMs is focused on code-centric memory domains. They both enable simple and efficient capability revocation.

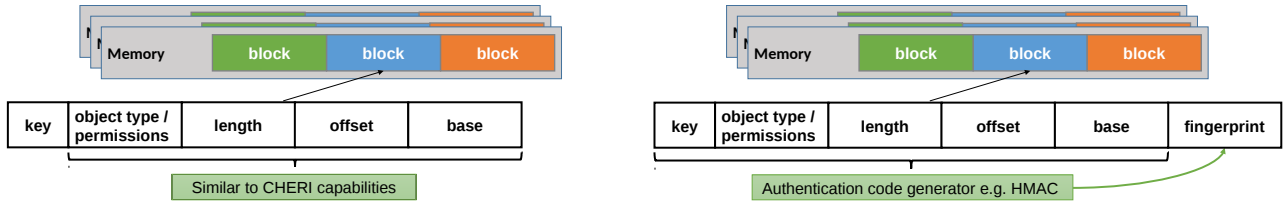
3.3 Memory keys

S/360 [2] is arguably the first system to introduce the concept of memory keys. It divided the physical address space into equal sized blocks each with a memory key. There is also another key that is part of the program status register. Access is granted to a block if the two keys match. This is similar to MaKC. Processors such as PA-RISC [16] and Itanium [19] employ keys to protect memory though they tend to apply keys to virtual memory and at the granularity of pages. Memory keys are becoming mainstream again, Intel is proposing adding them to future processors [8]

KeyKOS [15] and later EROS [33] use keys to refer to a fixed number of persistent pages and nodes which make up the entire state of the system. Pages of 4kB in size create segments which form the address space of domains holding data and code. Nodes cannot be directly accessed but provide an interface through key invocation. Similar to MaKC, the keys of KeyKOS consists of multiple fields that indicate the object type and value (address of the object). However, fine grained access to pages is not supported by KeyKOS.

4 MATCHING KEY CAPABILITIES (MAKC)

MaKC divides the last level of a complex memory hierarchy, such as the one shown in Figure 1, into equal sized blocks. Each block has an associated key (or capability) called the Block Protection Key (BPK). In addition, each execution hardware thread has at least one associated key termed the Execution Protection Key (EPK). EPKs are part of the processor's status registers, equivalent to capability registers in CHERI [37]. On each memory access, hardware automatically compares the BPK against the EPKs. Access is allowed on a match. Access is blocked and exception thrown on a mismatch. The software exception handler may check the missing key against a larger list of keys maintained by software. Note that our model is equally suited for variable sized blocks, however fixed sized blocks simplify management.



(a) MaKC in memory format. **(b) MaKC in-transit format.**

Figure 3: The MaKC formats

MaKC like CHERI is a hybrid capability model that blends conventional MMU-based virtual memory with a capability-system model. Keys can be modified from supervisor state but entering that state is not much more costly than entering a user state or crossing between user states as supervisor state are accessed via call gates just like in CHERI [35]. Keys enable both data access and execution control through a call gate model. Because there are both data and execution keys, it enables flexible protection models for both user and supervisor state as well as across different users.

4.1 Formats

MaKC has in-memory and in-transit formats. MaKC uses the in-memory format (Figure 3a) when the local node guarantees the integrity of capabilities. This is similar to how CHERI protects capabilities from manipulation by non-privileged entities. A key uniquely identifies a block or set of blocks. For a system with complex memory hierarchies as shown in Figure 1, a global memory manager/allocator is responsible for generating keys. The exact mechanism employed is implementation dependent, but may involve using a 64-bit counter, for example, which is incremented whenever a new key is needed. Note that the address of each block may be implicit especially if keys are closely integrated with a CPUfis paging mechanism.

Since MaKC is primarily designed for rack-scale systems, which are essentially distributed systems, it needs a format that prevents the manipulation of capabilities as they travel from one node to another over the interconnect. Figure 3b shows the in-transit format for keys. The main difference between the in-transit and the in-memory formats is the fingerprint field, which contains a cryptographically secure keyed-hash message authentication code (HMAC [26]). The authenticated hash can only be generated by authorized entities and protects the capability values when being transferred over untrusted channels. Such keys can be established (pre-shared) in a number of ways between components, for exampleas part of a trust establishment protocol when the system initializes. The HMAC ensures that a capability cannot be forged nor manipulated in transit. Once a key arrives at its destination, the in-transit format can be easily converted into in-memory format by stripping off the extra fields. Keys in MaKC are globally unique making them well suited for distributed systems with pooled memory.

4.2 Complex memory hierarchies

Figure 4 demonstrates with the aid of a simple example how MaKC can be used to manage and secure address spaces in a rack-scale system. The figure shows a node in rack-scale system with multiple

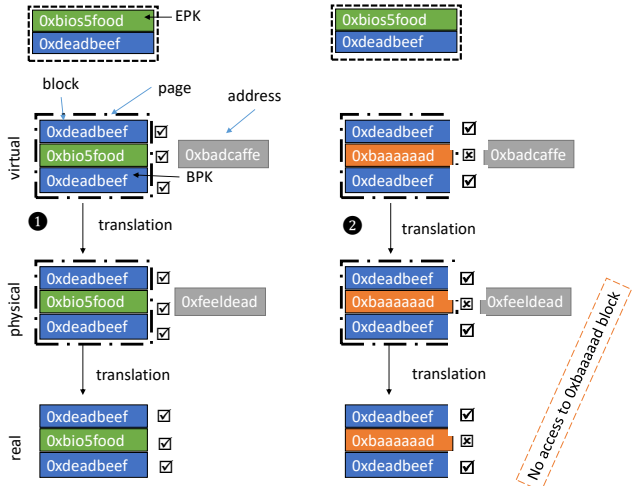


Figure 4: Managing memory hierarchies with MaKC.

address spaces. Like in Figure 1, the first (virtual address space) and the second (physical address space) levels of address space are local to the node. The third level of address space (real in Figure 4) is memory mapped from a global memory pool, hence it is globally unique i.e. has a unique tag. The figure only shows one node.

The real address space is divided into blocks with a BPK associated with each block and the node has a processor that supports matching key as described above. The blocks are mapped into node and then used for backing the physical address space. The virtual address space is then layered on top of the physical address space. A page frame in the physical address space framing a virtual page can be built from any combination of blocks. Notice that BPKs are propagated across the address spaces allowing any memory management system (e.g. MMU) to identify a block irrespective of the address space. For example, the local node can detect when the *0xbio5food* block is replaced by the *0xbaaaad* block as shown in Figure 4 even though the virtual and physical addresses are unchanged.

4.3 Supervisor state compartmentalization

Orthogonal to the call gate model described above, a local node (i.e. a processor) can designate a particular key as the master key. Keys can only be manipulated using special instructions which must be executed from a master key block. Code in blocks not

marked as master key blocks can access master key blocks by jumping/branching using special instructions to entry points on designated gateway blocks. The destination of a jump/branch must be marked by a gateway instruction. A gateway block can only be setup from a master key block. There are also special branch instruction(s) that allow jump/branching (and linking) from a master to a non-master block.

Essentially, our capability security model provides a low-cost means of de-privileging supervisor threads, so that they no longer have access to the entire memory. In addition to supervisor state, a hardware thread needs access to the master key for complete access to all memory blocks in a node. MaKC can be used to implement a split kernel [27] without relying on expensive hypercalls. The inner kernel mapped to key zero blocks and the guard blocks serves as controlled entry points (APIs) for outer kernels to request services from the inner kernel. Interrupts are initially received by the inner kernel and then dispatched to outer kernels.

4.4 Implementation feasibility

It is a challenge to evaluate the feasibility of a system that employs the MaKC without a concrete implementation. However, the key concepts in MaKC have been shown to have reasonable performance overhead in real systems.

Key matching could be implemented with protection tables that contain BPKs which the hardware can read and cache. This is similar to TLBs and hardware page walking. EPKs and BPKs can be dynamically checked in hardware after physical address computation, similarly to what happens in today's systems when they access memory region descriptors. This could be integrated with TLBs. For example, when the last-level TLB misses, hardware also walks the protection tables and retrieves the keys. If the check passes, the entry will be cached in the TLB and any subsequent access will just go through without additional overhead. Otherwise all TLBs would have to be invalidated.

CHERI showed that the cost and resources required for implementing a hybrid capability system is reasonable and furthermore maintaining the integrity of capabilities in the processor is feasible.

Most rack-scale systems have some sort of global memory manager so it is straight forward to generate capabilities with unique keys. Propagating MaKC over the interconnect would increase network traffic but we believe that the overhead is reasonable. 2 KiB fixed blocks and a 256-bit MakC protected by 64-bit time-stamp and 256-bit HMAC will introduce only 32 KiB of additional traffic per MiB of memory used. Note that it may be acceptable to use truncated hash to reduce overhead. We also performed experiments which showed that generating fingerprints for in-transit MaKC only adds 1776 cycles per operation on a Xeon E5v2 processor using OpenSSL 1.0.2g.

We believe future processors would provide hardware acceleration for HMAC (just as they now provide instructions for hashing) drastically reducing this overhead further. Alternatively, FPGA's coupled (via PCIe) to the main processor could be used to accelerate the generation of HMACs. Processors vendors may soon start multi-socket processors with an FPGA. Furthermore, some interconnects such as Gen-Z [1] can guarantee the integrity of packets eliminating the need for the in-transit format.

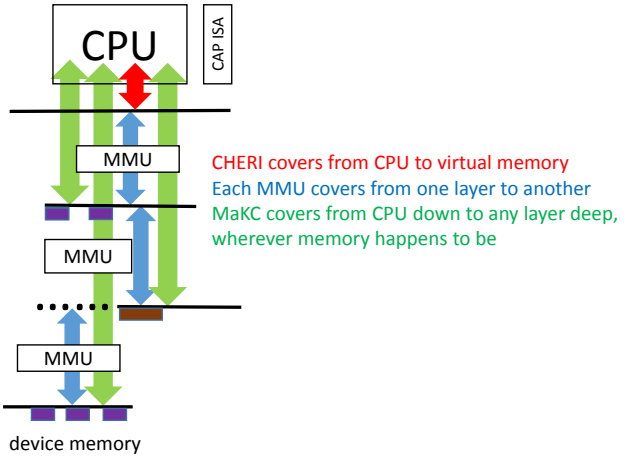


Figure 5: Comparison of approaches to protection

5 SUMMARY

MaKC is a capability based system to handle authorization and protection in complex memory hierarchies. Memory authentication using protection keys and predefined entry points using call gates allows efficient implementation of enclaves, kernel protection and virtual machines without the overhead of hypervisor or system calls. We show an early feasibility study of MaKC using memory and computation overhead analysis. We believe that MaKC provides the strong protection and isolation necessary to future rack-scalable systems, see Tables 1 and 2.

MaCK enables save remapping of memory at lower levels of the hierarchy by covering the full range from the CPU down to the memory cells (see Figure 5). Access control is empowered and

characteristics	approach to capability enforcement	
	memory-side	CPU-side
scaling	for larger mem	for smaller mem.
caching	complex	straightforward
mem. topol.	rack scale/hierar.	traditional arch.
mem-centric	better	agnostic
trust	data-centric (TOR)	node-, OS-centric
perform.	better for CC FAM	better for CC DRAM
reliability	easier containment	easier recovery
revocation	easier	distr. protocols

Table 1: Comparison of CPU-side vs memory-side

characteristics	approach to protection		
	MMUs	CHERI	MaKC
granularity	page	1B-AS size	1/N-N pages
hierarchy cover	2 layers	virtual AS	across layers
scale	node	single VAS	global
meta-data size	PTE	128b	256b
revocation	N/A	no support	supported
remapping	in 2 layers	no	yes

Table 2: Comparison of approaches to protection

enforced by memory-side capabilities and the matching-key checks. Revoking access to memory resources – a complex operation in capability systems – is made trivial by simply changing the key. Furthermore, the MaKC approach allows enabling huge pages without compromising security of small page sizes

Nevertheless, there are challenges that remain. Keys need to be stored and thus introduce space overhead which is a fraction of the data space. This can be optimized by reusing a single key for many blocks. The generation of fingerprints introduces a performance overhead which can be hidden by pipelining operations. The use of MaKC will add some complexity to software. However, CHERI faces a similar problem and showed that it is possible to abstract most of it inside libraries. MaKC uses cryptographic keys to ensure the authenticity of fingerprints and thus increases the complexity of security management – there is no free lunch but it can be optional enhancement

We plan to make MaKC real. First, we are exploring how MaKC matches the memory protection architecture of future processors. Once we understand the ISA implications, we will design the microarchitecture of the MaKC block itself. This includes enhancing simulators and exploring prototype FPGA implementations. In parallel, we are also exploring the needed modifications to OS and toolchains.

Besides the challenges stated above, there are still some open questions regarding the implementation of MaKC. We will need to make a design choice on whether to implement which features on the CPU-side or memory-side. With appropriate hardware support we have a choice between tagging bits or cryptographic keys for protecting capabilities. The block size could be fixed and possibly aggregated (e.g. cacheline < page < book) or it could be variable within a certain range. Last but not least, we need to think about the right abstractions for passing EPKs around as we envision a rack-scale, CHERI-like capability/pointer to memory resouces.

REFERENCES

[1] Gen-z draft core specification 2016, December 2016. Available online http://genzconsortium.org/draft-core-specification-december-2016/.

[2] Amdahl, G. M., Blaauw, G. A., and Brooks, F. P. Architecture of the IBM System/360. *IBM J. Res. Dev. 8*, 2 (Apr. 1964), 87–101.

[3] Asanovic, K. FireBox: A Hardware Building Block for 2020 Warehouse-Scale Computers. In *Proceedings of the 12th USENIX Conference on File and Storage Technologies (FASTfi14)* (Santa Clara, CA, USA, feb 2014), USENIX Association.

[4] Asanović, K. A hardware building block for 2020 warehouse-scale computers. In *FAST 14 keynote* (2014), USENIX.

[5] Baumann, A., Barham, P., Dagand, P.-E., Harris, T., Isaacs, R., Peter, S., Roscoe, T., Schüpbach, A., and Singhania, A. The Multikernel: A New OS Architecture for Scalable Multicore Systems. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles* (Big Sky, Montana, USA, 2009), SOSP '09, ACM, pp. 29–44.

[6] Belay, A., Bittau, A., Mashtizadeh, A., Terei, D., Mazières, D., and Kozyrakis, C. Dune: Safe User-level Access to Privileged CPU Features. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation* (Hollywood, CA, USA, 2012), OSDI'12, USENIX Association, pp. 335–348.

[7] Carter, N. P., Keckler, S. W., and Dally, W. J. Hardware Support for Fast Capability-based Addressing. In *Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems* (San Jose, California, USA, 1994), ASPLOS VI, ACM, pp. 319–327.

[8] Corbet, J. Memory protection keys. *LWN.net Online https://lwn.net/Articles/643797 May 13, 2015*.

[9] Dautenhahn, N., Kasampalis, T., Dietz, W., Criswell, J., and Adve, V. Nested Kernel: An Operating System Architecture for Intra-Kernel Privilege Separation. *SIGARCH Comput. Archit. News 43*, 1 (Mar. 2015), 191–206.

[10] El Hajj, I., Merritt, A., Zellweger, G., Milojicic, D., Achermann, R., Faraboschi, P., Hwu, W.-m., Roscoe, T., and Schwan, K. SpaceJMP: Programming with Multiple Virtual Address Spaces. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems* (Atlanta, Georgia, USA, 2016), ASPLOS '16, ACM, pp. 353–368.

[11] Faraboschi, P., Keeton, K., Marsland, T., and Milojicic, D. Beyond Processor-centric Operating Systems. In *Proceedings of the 15th USENIX Conference on Hot Topics in Operating Systems* (Switzerland, 2015), HOTOS'15, USENIX Association, pp. 17–17.

[12] Ganapathi, A., Ganapathi, V., and Patterson, D. Windows XP Kernel Crash Analysis. In *Proceedings of the 20th Conference on Large Installation System Administration* (Washington, DC, 2006), LISA '06, USENIX Association, pp. 12–12.

[13] Gerber, S., Zellweger, G., Achermann, R., Kourtis, K., Roscoe, T., and Milojicic, D. Not Your Parents' Physical Address Space. In *Proceedings of the 15th USENIX Conference on Hot Topics in Operating Systems* (Switzerland, 2015), HOTOS'15, USENIX Association, pp. 16–16.

[14] Glerum, K., Kinshumann, K., Greenberg, S., Aul, G., Orgovan, V., Nichols, G., Grant, D., Loihle, G., and Hunt, G. Debugging in the (Very) Large: Ten Years of Implementation and Experience. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles* (Big Sky, Montana, USA, 2009), SOSP '09, ACM, pp. 103–116.

[15] Hardy, N. Keykos architecture. *SIGOPS Oper. Syst. Rev. 19*, 4 (Oct. 1985), 8–25.

[16] HEWLETT-PACKARD COMPANY. PA-RISC 1.1 Architecture and Instruction Set Reference Manual HP Part Number: 09740-90039 Third Edition, February 1994. https://parisc.wiki.kernel.org/images-parisc/6/68/Pa11_acd.pdf.

[17] HP Labs. The Machine. http://www.hpl.hp.com/research/systems-research/themachine/, January 2015.

[18] Intel Corporation. Single-chip Cloud Computer. Online, 2009. Accessed 2017-01-09. http://www.intel.com/content/dam/www/public/us/en/documents/technology-briefs/intel-labs-single-chip-cloud-overview-paper.pdf.

[19] Intel Corporation. Intel Itanium Architecture Software Developerfis Manual Volume 2: System Architecture Revision 2.3, May 2010.

[20] Intel Corporation. *Intel Xeon Phi Coprocessor System Software Developers Guide*, March 2014. SKU 328207-003EN.

[21] Intel Corporation. Intel Rack Scale Design. Online, 2016. http://www.intel.com/content/www/us/en/architecture-and-technology/rack-scale-architecture/intel-rack-scale-architecture-resources.html.

[22] Jones, A. K., Chansler, Jr., R. J., Durham, I., Schwans, K., and Vegdahl, S. R. StarOS, a Multiprocessor Operating System for the Support of Task Forces. In *Proceedings of the Seventh ACM Symposium on Operating Systems Principles* (Pacific Grove, California, USA, 1979), SOSP '79, ACM, pp. 117–127.

[23] Klein, G., Elphinstone, K., Heiser, G., Andronick, J., Cock, D., Derrin, P., Elkaduwe, D., Engelhardt, K., Kolanski, R., Norrish, M., Sewell, T., Tuch, H., and Winwood, S. sel4: Formal verification of an os kernel. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles* (Big Sky, Montana, USA, 2009), SOSP '09, ACM, pp. 207–220.

[24] Koldinger, E. J., Chase, J. S., and Eggers, S. J. Architecture support for single address space operating systems. In *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (Boston, Massachusetts, USA, 1992), ASPLOS V, ACM, pp. 175–186.

[25] Koldinger, E. J., Levy, H. M., Chase, J. S., and Eggers, S. J. The Protection Lookaside Buffer: Efficient Protection for Single-Address Space Computers. Tech. Rep. 91-11-05, Department of Computer Science and Engineering, University of Washington, Seattle, Washington, Nov. 1991.

[26] Krawczyk, H., Bellare, M., and Canetti, R. *HMAC: Keyed-Hashing for Message Authentication.* Network Working Group, February 1997. RFC 2104.

[27] Kurmus, A., and Zippel, R. A Tale of Two Kernels: Towards Ending Kernel Hardening Wars with Split Kernel. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (Scottsdale, Arizona, USA, 2014), CCS '14, ACM, pp. 1366–1377.

[28] Kyathsandra, J., and Zhou, X. Rack Scale Architecture: Designing the Data Center of the Future. http://bit.ly/idf14-rsa, Intel IDF14 Shenzen, 2014.

[29] Levin, R., Cohen, E., Corwin, W., Pollack, F., and Wulf, W. Policy/Mechanism Separation in Hydra. In *Proceedings of the Fifth ACM Symposium on Operating Systems Principles* (Austin, Texas, USA, 1975), SOSP '75, ACM, pp. 132–140.

[30] Mullender, S., van Rossum, G., Tanenbaum, A., van Renesse, R., and van Staveren, H. Amoeba, A distributed operating system for the 1990s. *Computer 33*, 5 (May 1990), 44–53.

[31] Needham, R. M., and Walker, R. D. The Cambridge CAP Computer and Its Protection System. In *Proceedings of the Sixth ACM Symposium on Operating Systems Principles* (West Lafayette, Indiana, USA, 1977), SOSP '77, ACM, pp. 1–10.

[32] Rozier, M., Abrossimov, V., Armand, F., Boule, I., Gien, M., Guillemont, M., Herrmann, F., Kaiser, C., Langlois, S., Léonard, P., et al. Overview of the chorus distributed operating systems. In *Computing Systems* (1991), Citeseer.

[33] Shapiro, J. S., Smith, J. M., and Farber, D. J. EROS: A Fast Capability System. In *Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles* (Charleston, South Carolina, USA, 1999), SOSP '99, ACM, pp. 170–185.

[34] Vilanova, L., Ben-Yehuda, M., Navarro, N., Etsion, Y., and Valero, M. Codoms: Protecting software with code-centric memory domains. In *2014*

ACM/IEEE 41st International Symposium on Computer Architecture (ISCA) (June 2014), pp. 469–480.

[35] Watson, R. N. M., Woodruff, J., Neumann, P. G., Moore, S. W., Anderson, J., Chisnall, D., Dave, N., Davis, B., Gudka, K., Laurie, B., Murdoch, S. J., Norton, R., Roe, M., Son, S., and Vadera, M. CHERI: A Hybrid Capability-System Architecture for Scalable Software Compartmentalization. In *2015 IEEE Symposium on Security and Privacy* (May 2015), pp. 20–37.

[36] Wilkes, J., and Sears, B. A comparison of Protection Lookaside Buffers and the PA-RISC Protection Architecture. Technical Report HPL-92-55, Computer Systems Laboratory, Hewlett-Packard Laboratories, Palo Alto, CA, USA, March 1992.

[37] Woodruff, J., Watson, R. N., Chisnall, D., Moore, S. W., Anderson, J., Davis, B., Laurie, B., Neumann, P. G., Norton, R., and Roe, M. The CHERI Capability Model: Revisiting RISC in an Age of Risk. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture* (Minneapolis, Minnesota, USA, 2014), ISCA '14, IEEE Press, pp. 457–468.