Memento—Energy-Aware Memory Placement in Operating Systems*

Sven Köhler[1], Benedict Herzog[2], Henriette Hofmeier[2], Manuel Vögele[2], Lukas Wenzel[1]
[1]Hasso Plattner Institute, Potsdam [2]Ruhr University Bochum (RUB)
{firstname.lastname}@{hpi,rub}.de

ABSTRACT

Today's availability of new memory technologies requires radical re-thinking of memory management in general-purpose operating systems. Main-memory technologies (e.g. NVM), completely new cell types (e.g. PCRAM), and coherent interconnects (e.g. CXL) challenge existing programming and system abstractions (e.g. POSIX). At the same time, memory subsystems received much less attention from energy efficiency efforts, compared to compute resources, in part because of these outdated abstractions. We therefore need new interfaces at operating system level that not only communicate functional (e.g. persistent vs. volatile) and non-functional (e.g. latency bounds) memory properties to application developers, but also take energy efficiency into account.

We propose *Memento*, a new concept for efficient memory management at the operating system level. Memento addresses the shortcomings of the current state-of-the-art with methods for analysing program code at *development time*, its operational characteristics at *runtime*, along with characteristics of memory resources at system *setup time*. Figure 1 visualises the design of our approach. The core component is the *Memory Governor*, implemented within the operating system, which makes automatic and energy-efficient memory placement decisions. It handles *buffer-allocation requests* considering placement requirements, hardware characteristics, and administrative constraints with the memory resource promising the lowest energy demand for that workload.

Energy Efficiency. For energy considerations we implement a *memory energy model* for workloads and hardware components on the following inputs: (a) *memory access behaviour* of the workload, (b) *allocation granularity* (e.g. costs of large vs. small buffers in NVM vs. HBM memory), and (c) the actual *hardware*. Since it is not feasible to create energy models for every type of memory, we rely on models for general memory technologies, and only specialise in product-specific models where necessary. We utilise different machine-learning techniques—from simple linear or ensemble models up to sophisticated neural networks—to trade-off expressiveness and accuracy for training and execution costs.

Interfacing explicitly with the Memory Governor from a programmer's side, requires an API, which we build around the notion of *workload-sensitivity* weights. Workload sensitivity describes how domain experts expect their workloads to react to a memory resource with specific functional and non-functional properties. The implementation of workload sensitivity is based on scoped allocators valid for limited sections of the application source code and lifted as soon as the scope is left.

However, not all placement requirements can be explicitly stated—either due to the lack of domain experts and development time, or because the Memory Governor has to handle legacy software. In this case, our approach relies on an implicit deduction of a workload's requirements and sensitivities, based on *static code analysis* [2] ahead of time, as well as, *profiling/tracing* at runtime by means of of performance measurement counters (PMCs). The implicitly and explicitly collected knowledge about a workload is persisted and exchanged in the form of the *Memento Notation*. It can be embedded, for example, in ELF sections within the binary and later be refined and reused each time the application is started.

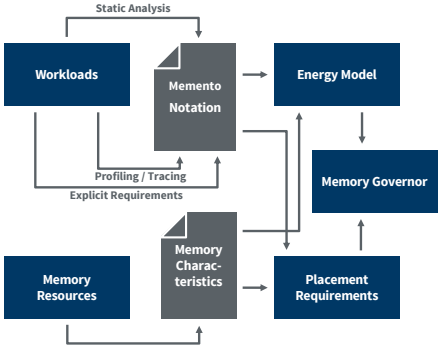


Figure 1: Design of the Memento approach for OS-level memory management. Core component is the *Memory Governor* for efficient placements based on hardware and workload characteristics, as well as, placement requirements.

Carbon Efficiency. The tools, notation, and microbenchmarks build as part of Memento can find also applicability in other fields. Prominently, despite energy savings being a worthwhile economical goal in itself, societal efforts are directed at limiting carbon emissions to limit global warming. Following the Software Carbon Intensity (SCI) specification [1], Memento may contribute to this objective in two ways: (i) by providing an estimate of the *operational carbon costs* by means of the energy model and energy grid's carbon footprint and (ii) by optimising placements with respect to total carbon costs (i.e. operational and embodied costs). Thereby, the embodied costs are costs for manufacturing, lifespan and wear and tear, and disposal. Considering both types of carbon costs leads to new trade-offs. For example, the most carbon-efficient placement for a 4 kB buffer with 10 s lifetime and few accesses depends on the memory technology and energy grid's carbon footprint (in this example DRAM/NVM and coal/wind). We assume 980 g CO_2e/kW for the carbon footprint of coal and 11 g CO_2e/kW for wind. In case of the wind-powered grid, the placement in NVM is more carbon efficient (0.08 ng and 0.13 ng). Whereas for the coal-powered grid, the placement in DRAM causes less carbon emissions (7.1 ng and 4.2 ng). With the tools developed in Memento, such decisions for energy or carbon efficiency can be made within the operating system and therefore made usable for many types of applications.

*This poster abstract was accepted at the 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI'23), Poster Session.
Parts of this work have been published at the 2nd Workshop on Sustainable Computer Systems (HotCarbon'23).

ACKNOWLEDGEMENTS

This work was partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 502228341 ("Memento") and 465958100 ("NEON") and from the Bundesministerium für Bildung and Forschung (BMBF, Federal Ministry of Education and Research) in Germany for the project AI-NET-ANTILLAS 16KIS1315.

REFERENCES

[1] Green Software Foundation. 2021. Software Carbon Intensity Standard. https://github.com/Green-Software-Foundation/sci/blob/main/Software_Carbon_Intensity/Software_Carbon_Intensity_Specification.md

[2] Julia Lawall and Gilles Muller. 2018. Coccinelle: 10 Years of Automated Evolution in the Linux Kernel. In *Annual Technical Conference (ATC'18)*. USENIX, 601–614.

Memento

Energy-Aware Memory Placement in Operating Systems

Sven Köhler[1*], Benedict Herzog[2*], Henriette Hofmeier[2], Manuel Vögele[2], Lukas Wenzel[1]
*both authors contributed equally to this work

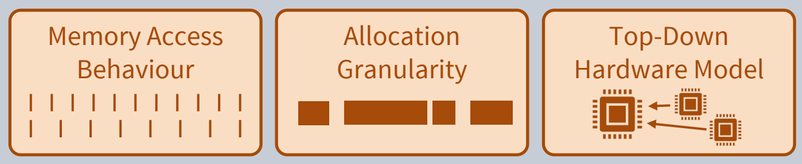
[1] Hasso Plattner Institute, Potsdam [2] Ruhr University Bochum

Motivation

- New memory technologies (NVM, HBM, …), cell-types (e.g. PCRAM), and coherent interconnects (e.g. CXL) challenge existing system and programming abstractions of a homogenous memory space
- Like functional and non-functional properties, all memory technologies highly differ in their energy demand
- The OS community needs to react and enable optimisation strategies, while leveraging energy saving potentials

Memory Energy Model

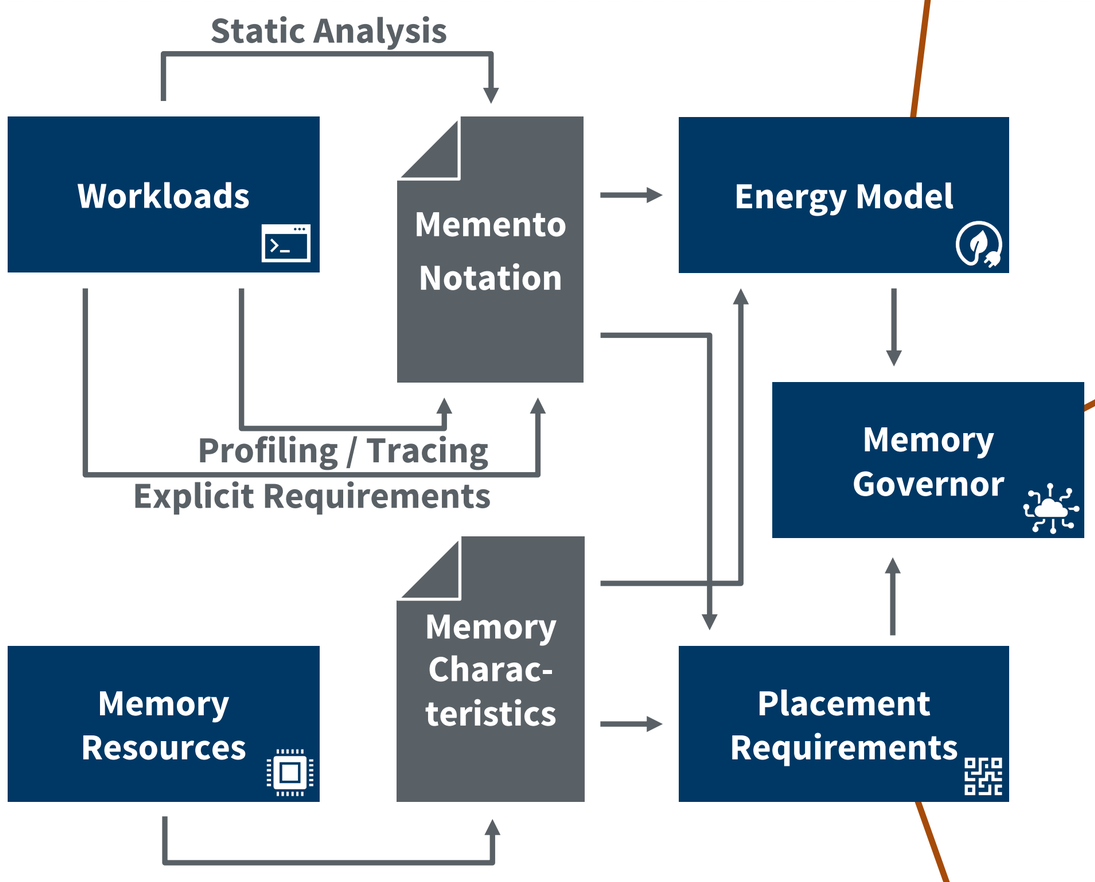
- Model of potential placement impact, based on:



- Approaches with ML techniques, i.e. linear, ensemble, neural networks
- We will evaluate expressiveness and accuracy for each, and additional costs for training and inference at runtime

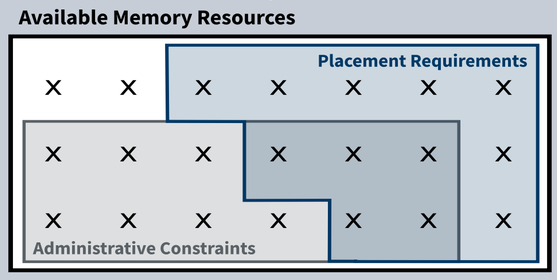
The Memento Approach

- The central **Memory Governor** matches allocation requests with the best available memory resource
- Using an energy model we try to minimize the entire system's energy demand at runtime
- We determine a workload's **sensitivity**, i.e. how it reacts **to** different **memory types**' bandwidth, latencies, or volatility
- Workload classification from code analysis, tailored micro benchmarks, and profiling at run-time [2]
- Deduced requirements are denoted in an exchangeable format, the **Memento Notation** (e.g. persisted across runs in dedicated ELF sections)

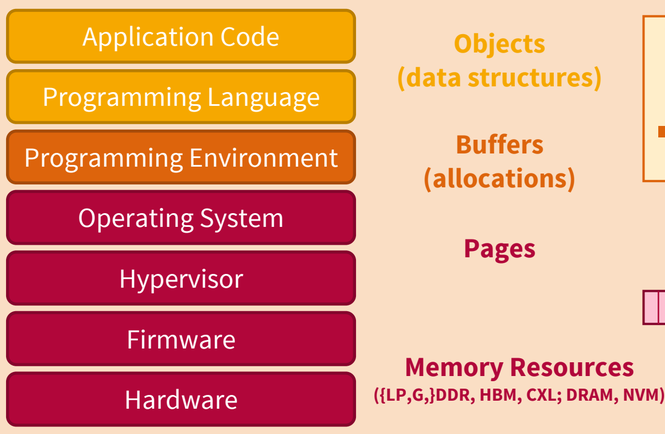


Memory Governor

- Part of the OS (service + module)
- Keeps track of available / free memory hardware resources
- Chooses most energy-efficient buffer location, considering availability, placement requirements, and administrative constraints
- Inspired by current frequency governors for compute devices
- Works on allocation granularity (not threads or processes)



Choosing The Right Granularity



- High-level applications structure memory in objects, rather than pages
- A buffer view on OS-level breaks the abstraction and allows for optimisation

Placement Requirements

- Developers are experts on the dynamics of their application, not on available system resources
- Legacy applications do not necessarily make (good) placement decisions for new memory resource types

- First, offer **explicit API** with manually annotated sensitivity as starting point, e.g. with scoped allocators
- For legacy workloads, use the **implicit** knowledge persisted in the Memento Notation

```
AllocCharacteristic LatencyNVSpec {
  // uninitialised fields default to 0
  weight_latency = 3.f,
  weight_randomAccess = 1.f,
  non_volatile = true
};
{
  // all allocations in this scope are
  // associated with LatencyNVSpec
  MemGuard guard(LatencyNVSpec);
  Index * jumpListA = new Index[4096];
}
```

Carbon-Efficient Placement

- Memento tools, notation and benchmarks also applicable in other fields, e.g. a carbon-efficient placement following the SCI specification [1]:

$$\textit{Software Carbon Intensity} = \frac{\textit{Operational Carbon} + \textit{Embodied Carbon}}{\textit{Unit of Work}}$$

- case study (gCO_2e/alloc) best highlighted

	10 µs allocation, 1 access, volatile			50 ns allocation, 100,000 acesses, persistent			10 s allocation, 5 acesses, volatile		
	DRAM	NVDIMM	Optane	DRAM	NVDIMM	Optane	DRAM	NVDIMM	Optane
Wind	1.3e-16	1.5e-16	1.6e-11		7.3e-13	1.6e-6	1.3e-10	1.5e-10	8.0e-11
Mix	1.6e-15	1.6e-15	5.4e-10		8.2e-12	5.4e-5	1.6e-9	1.6e-9	2.7e-9
Gas	2.0e-15	2.1e-15	6.9e-10		1.0e-11	6.9e-5	2.0e-9	2.1e-9	3.4e-9
Coal	4.2e-15	4.2e-15	1.4e-9		2.1e-11	1.4e-4	4.2e-9	4.2e-9	7.1e-9

References

1. Green Software Foundation. 2021. Software Carbon Intensity Standard. https://github.com/Green-Software-Foundation/sci/blob/main/Software_Carbon_Intensity/Software_Carbon_Intensity_Specification.md
2. Julia Lawall and Gilles Muller. 2018. Coccinelle: 10 Years of Automated Evolution in the Linux Kernel. In *Annual Technical Conference (ATC'18)*. USENIX, 601–614

Website
{firstname.lastname}@{hpi,rub}.de
informatik.rub.de/boss/research/memento

funded by the Federal Ministry of Education and Research in Germany – AI-NET-ANTILLAS 16KIS1315
SPP 2377
funded by the Deutsche Forschungsgemeinschaft (German Research Foundation) – 502228341, 465958100