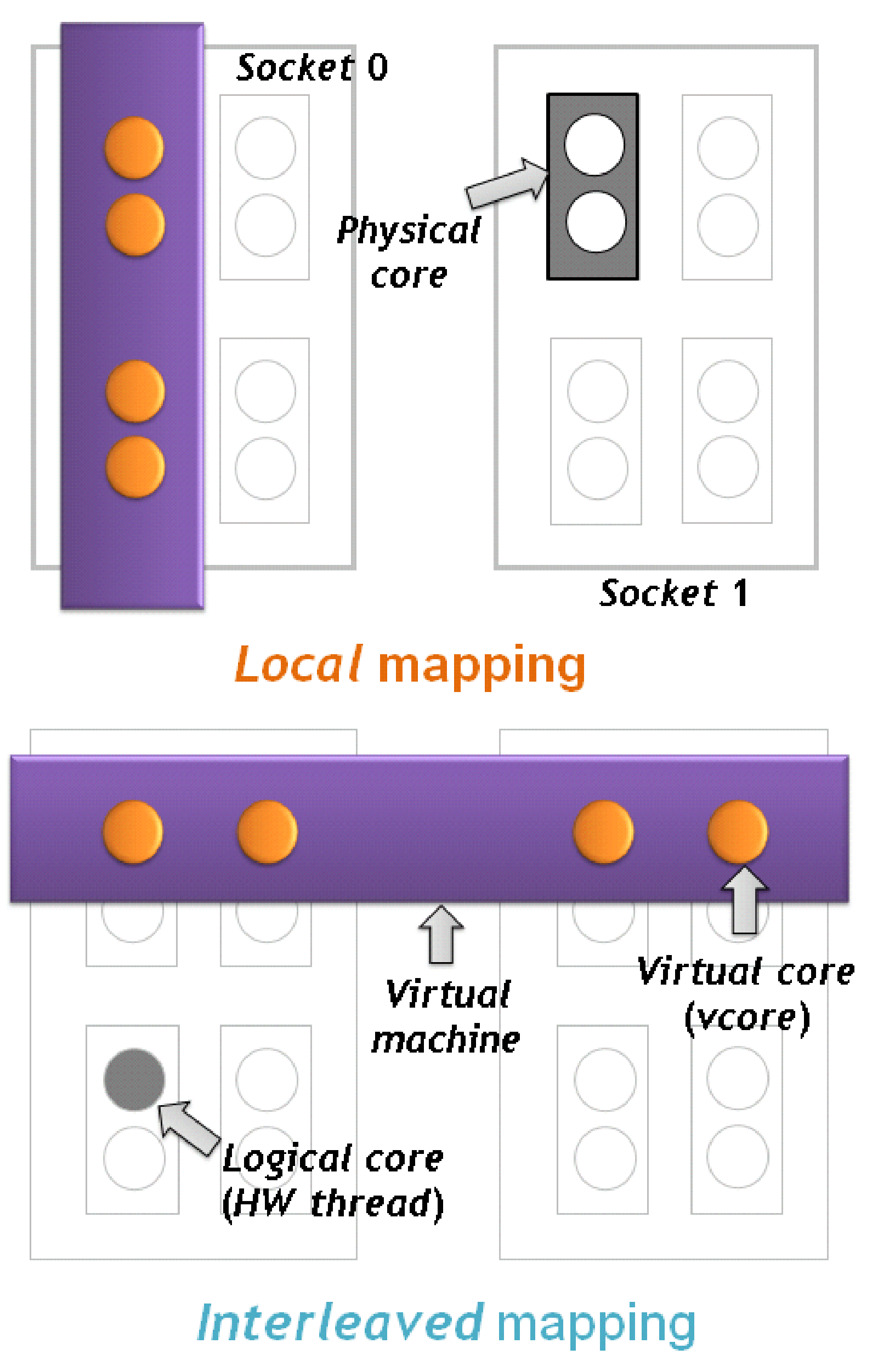
Dynamic Adaptive Virtual Core Mapping to Improve Power, Energy, and Performance in Multi-socket Multicores

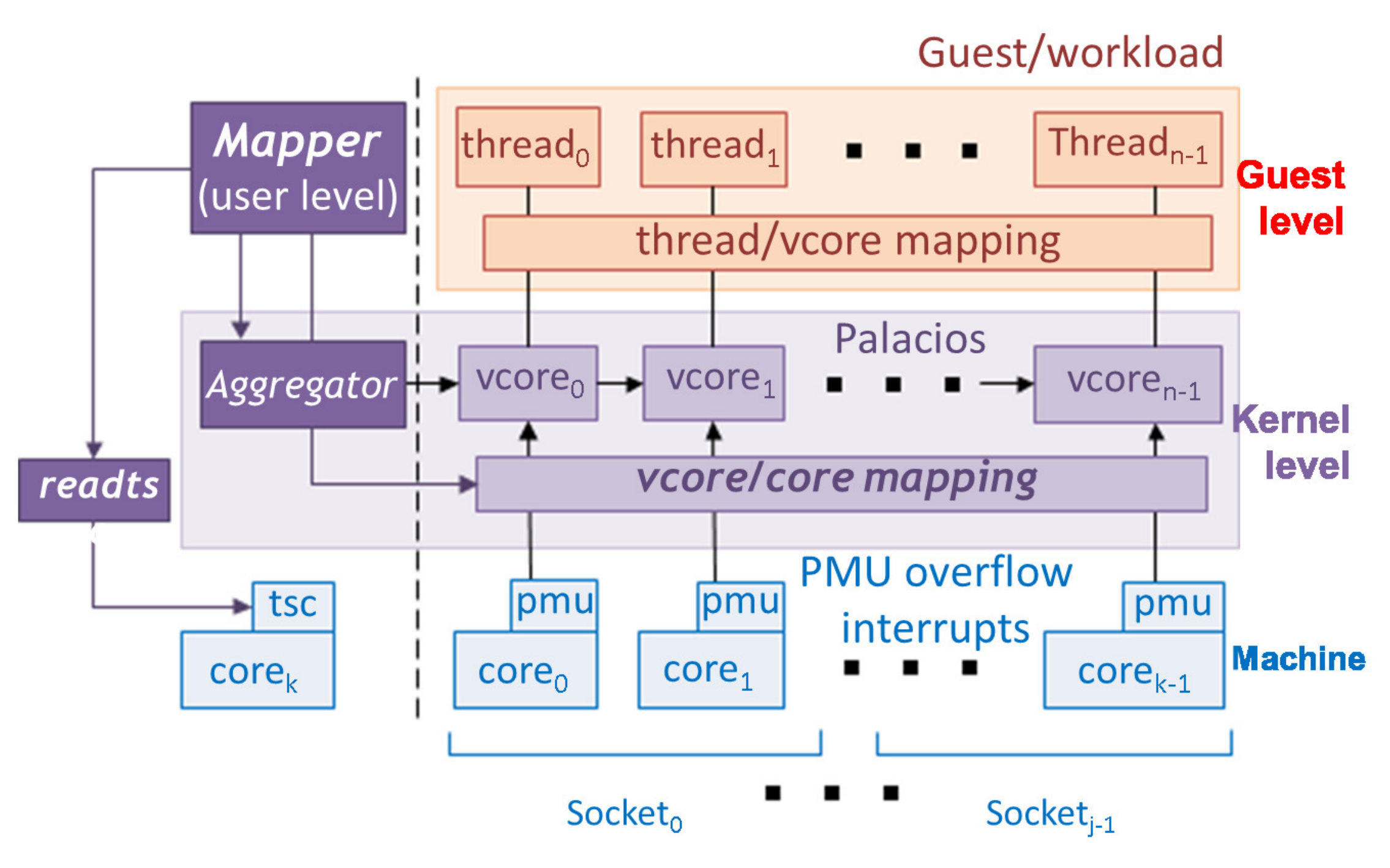
Chang Bae, Lei Xia, Peter Dinda, John Lange
{cbae@u.,lxia@,pdinda@}northwestern.edu, jacklange@cs.pitt.edu

Virtual Core Mapping Prob.

Consider a multithreaded parallel application running inside a multicore VM context that is itself hosted on a multi-socket multicore physical machine. How should the VMM map virtual cores (vcores) to physical cores? We compare a *local* mapping, which compacts virtual cores to processor sockets, and an *interleaved* mapping, which spreads them over the sockets.



System Overview



Three key components of the system: Mapper, Aggregator, and vcore/pcore mapping. Vcore/pcore mapping provides the core mechanism. The Aggregator and mapping components are controlled and called by the Mapper component.

References

[1] LANGE et al. *An introduction to the palacios virtual machine monitor release 1.3.*, Tech. Rep. NWU-EECS-11-10, Depart. of EECS, Northwestern Univ. (2011).

[2] LANGE et al. *A. Minimal-overhead virtualization of a large scale supercomputer*, In Proceedings of the 2011 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (2011).

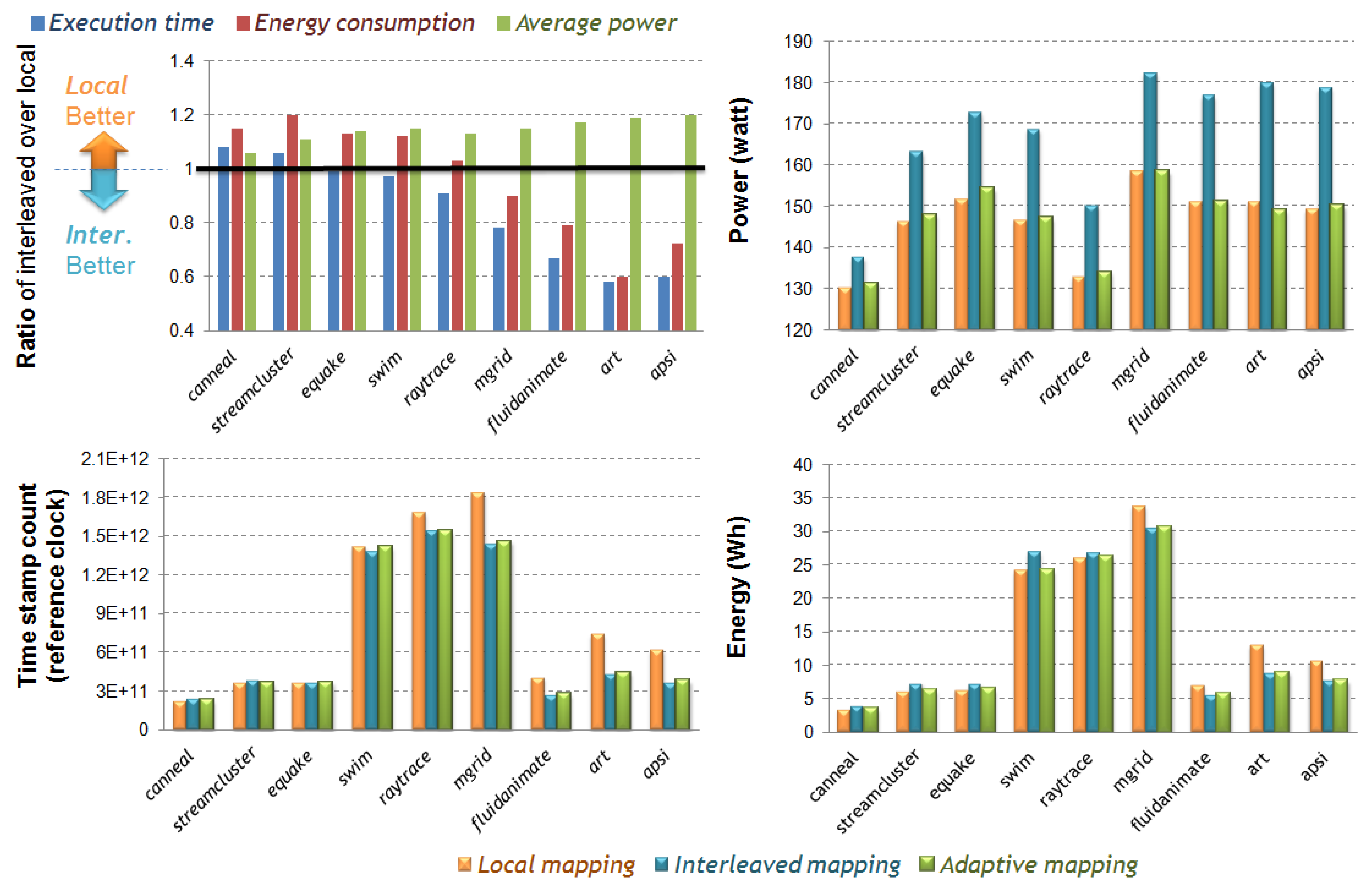
[3] LANGE et al. *Palacios and kitten: New high performance operating systems for scalable virtualized and native supercomputing.*. In Proceedings of the 24th IEEE International Parallel and Distributed Processing Symposium (2010).

Acknowledgements

This project is made possible by support from the National Science Foundation (NSF) via grant CNS-0709168 and the Department of Energy (DOE) via grant DE-SC0005343.

Dynamic Adaptive Vcore Mapping for Various Objectives

We have demonstrated the opportunity for optimizing for performance, power, and energy presented by being able to simply choose between *local* and *interleaved* mappings of virtual cores to physical cores. This opportunity is leveraged in an automatic adaptive system that chooses between these two mappings. We implemented and evaluated in the context of the Palacios VMM [1, 2, 3] to do this. We demonstrate that the performance of SPEC and PARSEC benchmarks can be increased by as much as 66%, energy reduced by as much as 31%, and power reduced by as much as 17%, depending on the optimization objective. The overhead of system is concentrated in page table scanning and vcore remapping; the worst case we observed takes 4.6 ms and 5.3 ms for each. The overall overhead in one execution is clearly negligible even in the worst case that has less than 0.05 % overhead.

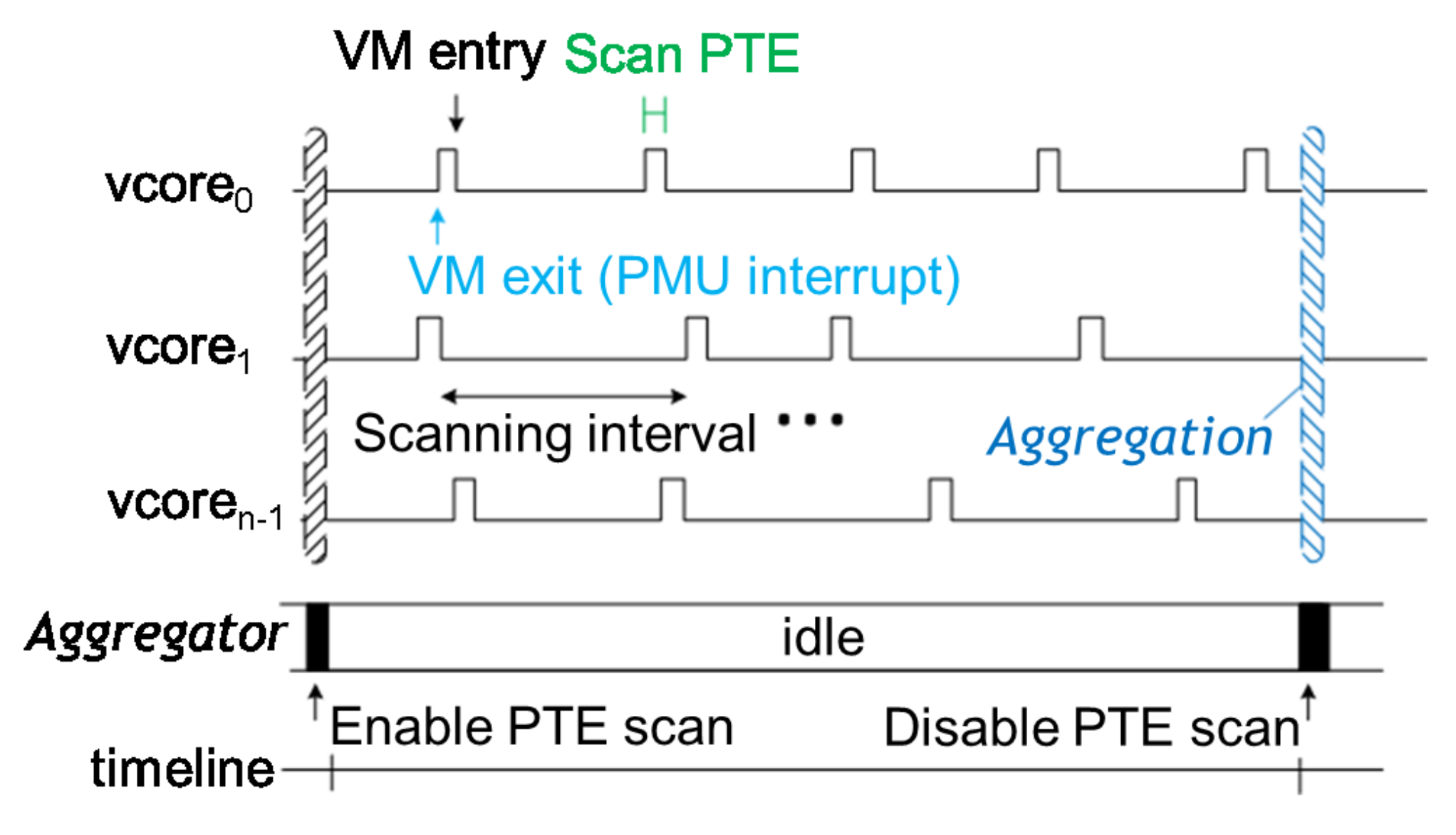


VMM-base Measurement

We took several measures to arrive at our set of metrics. First, we used architecture-level analysis. Secondly, we considered only metrics that could be quickly captured in a VMM, which generally means operating at the page granularity. Finally, we selected a minimally correlated set.

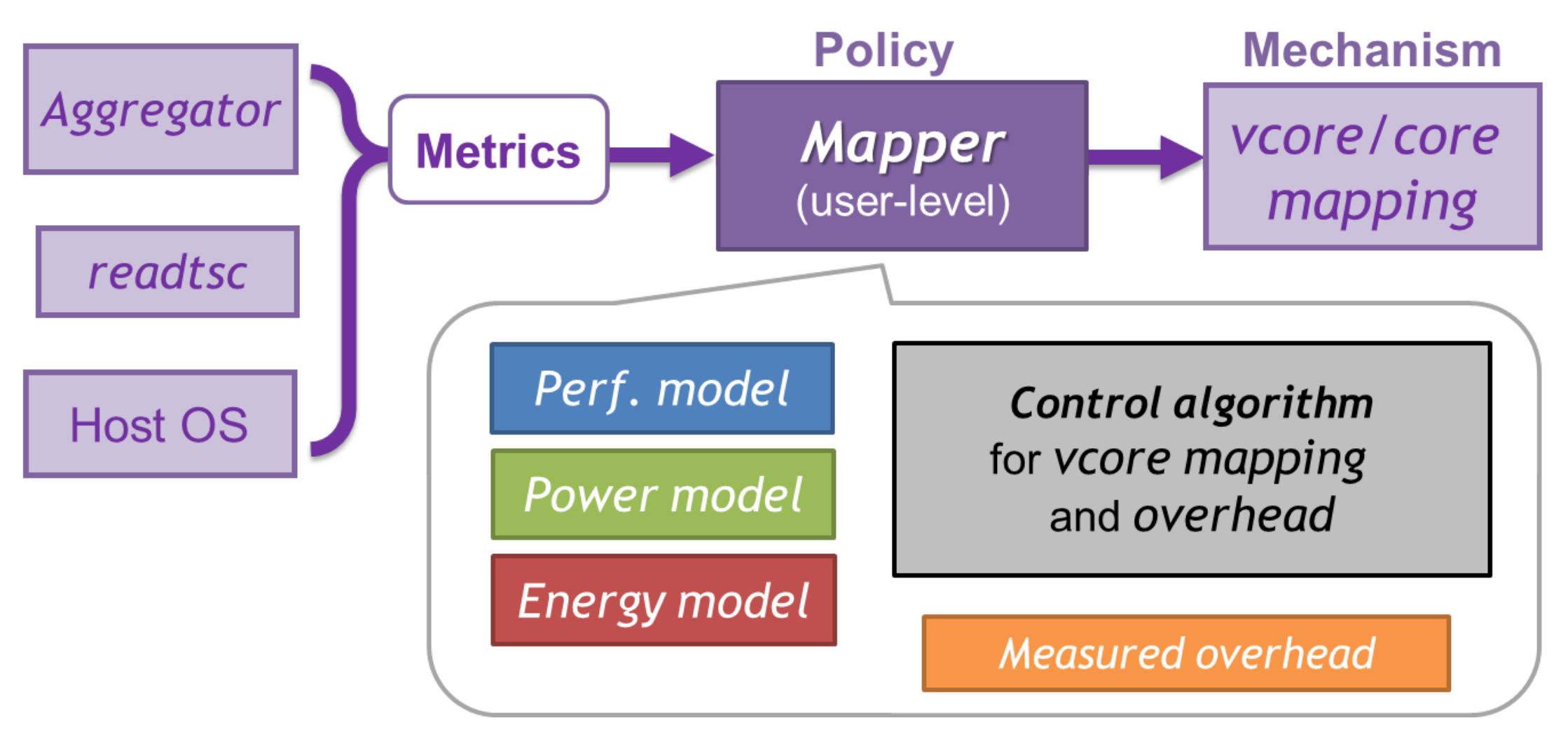
- Pages with memory load from all vcores
  - Average page access rate *per memory* or *write op*
- Pages with memory store from all vcores
  - Average page write rate *per memory* or *write op*
- Degree of read or write sharing
  - Shared page access ratio *per memory* or *write op*
- Degree of write sharing
  - Shared page write ratio *per memory* or *write op*

Flow of measurement mechanism: In each vcore, (1) Hardware triggers the interval after a specified number of memory ops or write ops. (2) VMM scans page table to find page access or page writes generating bitmap(s). (3) Aggregator collects bitmaps across vcores, and computes the metrics



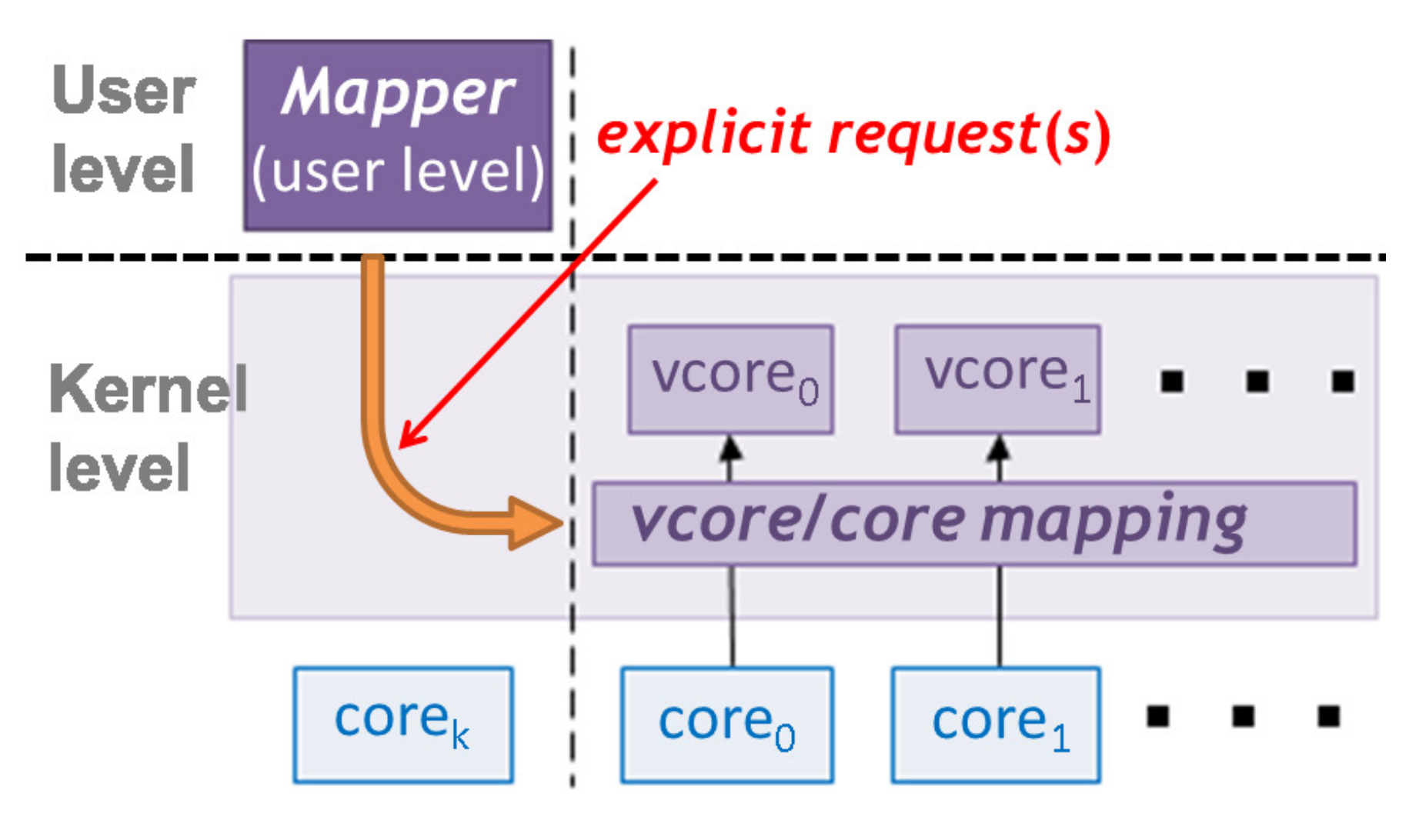
Vcore Mapping Policy

Our approach is based on modeling, in which we run diverse workloads on the machine. As the machine runs, we continue to collect the metrics, and use their values, plus the models, to make predictions of the relative utility of the two mappings, deciding between them in pursuit of the currently chosen goal.



Migration Mechanism

vcore mapping in Palacios is changed only on explicit request(s) from *Mapper*. Steps in the request: 1) Forces all vcores to exit. 2) Rebinds host kernel threads, with virtualization states to the new locations. 3) Synchronizes threads and reenters the guest.


User level
Mapper (user level)
explicit request(s)
Kernel level
vcore0
vcore1
vcore/core mapping
core k
core0
core1