BRIAN D. MARSH, THOMAS J. LEBLANC,
MICHAEL L. SCOTT, EVANGELOS P. MARKATOS

FIRST-CLASS USER-LEVEL THREADS

> IT IS OFTEN DESIRABLE, FOR REASONS OF CLARITY, PORTABILITY, AND EFFICIENCY, TO WRITE PARALLEL PROGRAMS IN WHICH THE NUMBER OF USER PROCESSES IS INDEPENDENT OF THE NUMBER OF AVAILABLE PROCESSORS.
>
> Brian D. Marsh, et al.

WHAT'S WRONG WITH KERNEL-LEVEL THREADS?

- semantic inflexibility
- poor performance

WHAT'S WRONG WITH USER-LEVEL THREADS?

- Blocking denies service to other threads
- Lack of coordination between scheduling & synchronization
- Lack of conventions for data sharing between thread packages

User-level threads are not recognized or supported by the kernel.

GRANTING FIRST-CLASS STATUS TO USER-LEVEL THREADS

- Blocking operations without denying service to their peers
- Threads in different, but overlapping, address spaces are able to synchronize access to shared data
- Provide user-level code with the same scheduling information available to the kernel

Most operations reasonable in the kernel will become reasonable in user space.

PREMISE

- Most thread operations can be performed in user space
- Fine-grained user scheduler
- Coarse-grained kernel scheduler

User space and kernel space schedulers need to communicate.

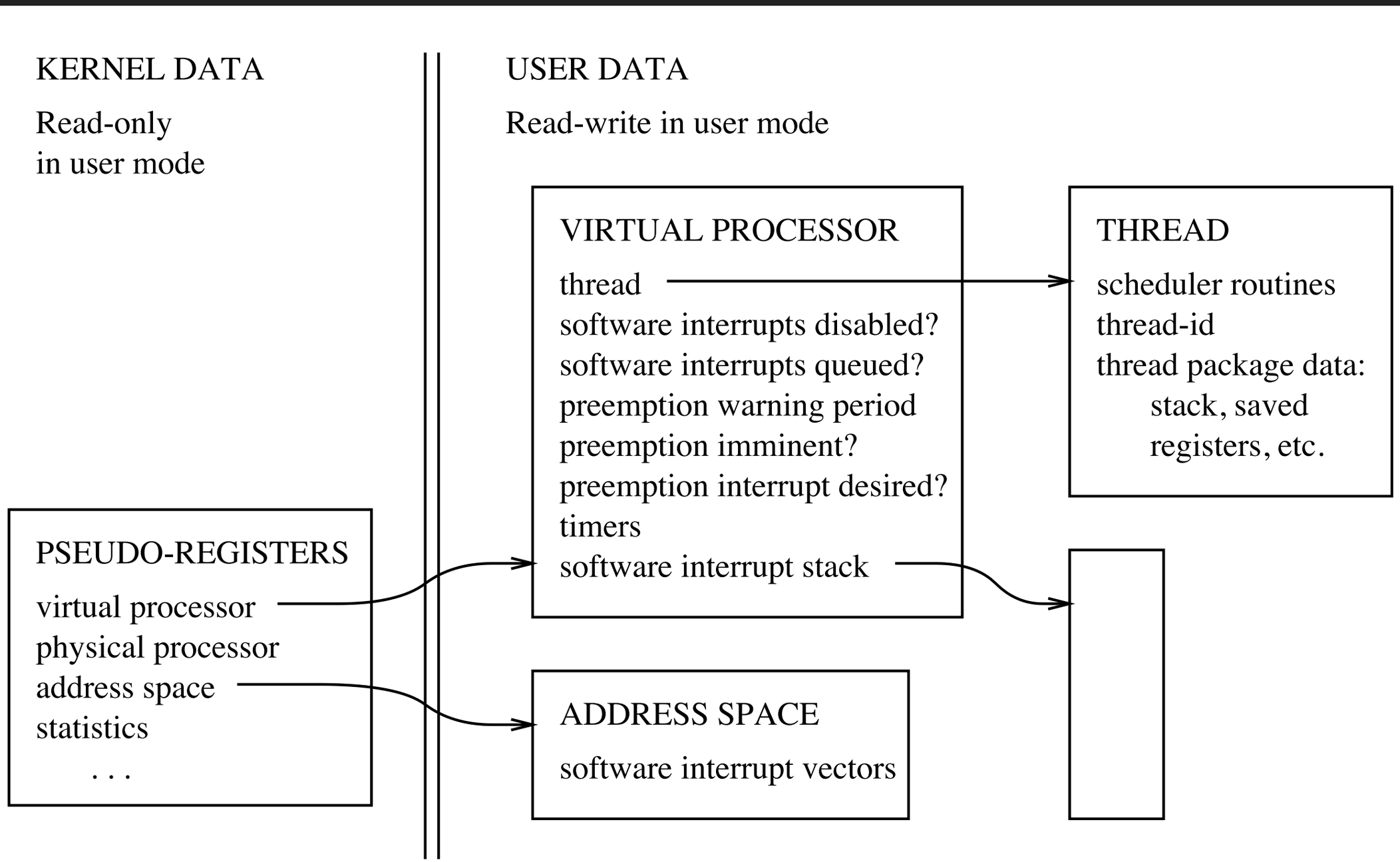
OBSERVATIONS

- Thread state information accessible to kernel
- Thread package accepts scheduling interrupts from the kernel
- Thread package schedulers provide a standard interface

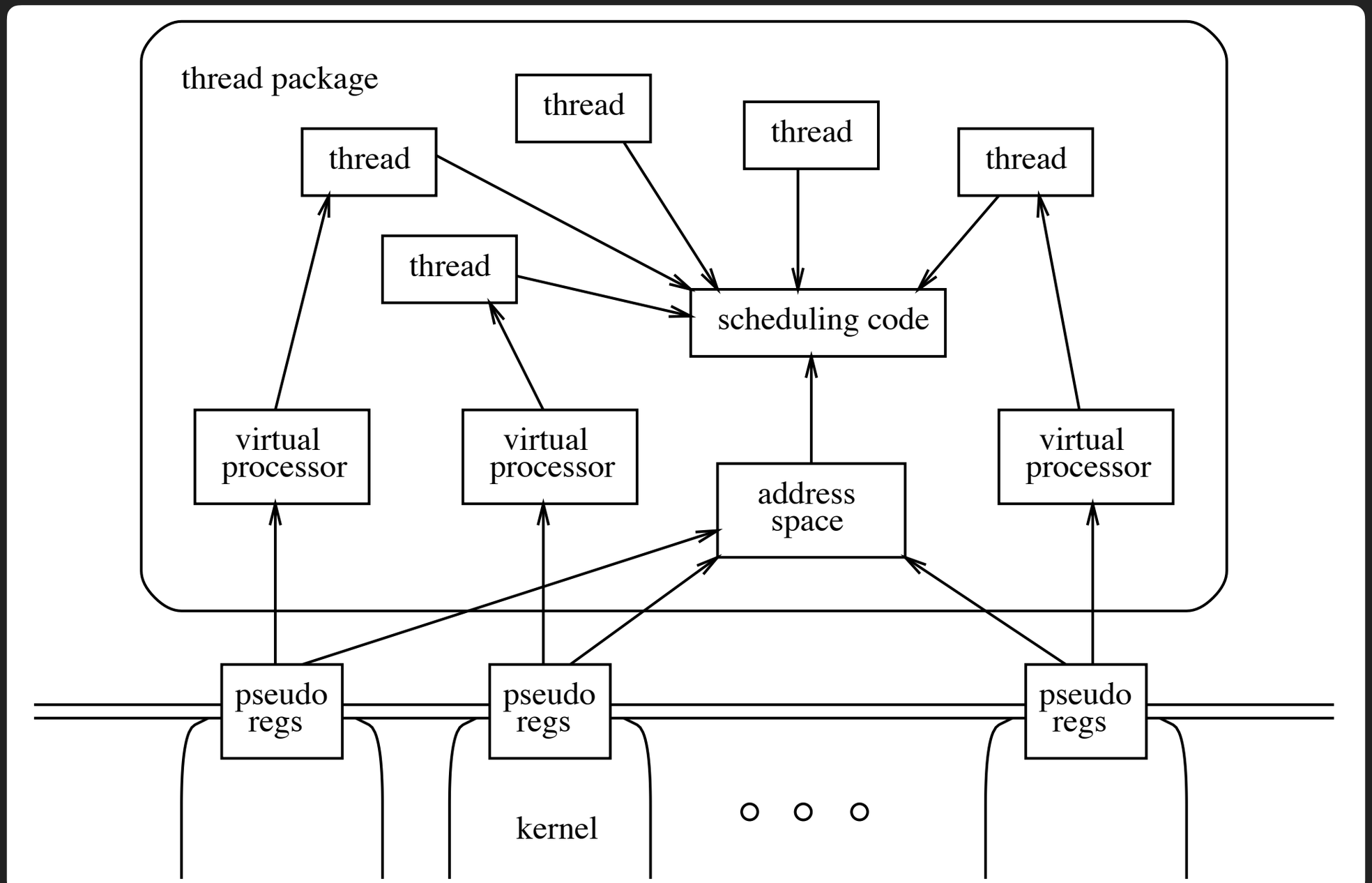
MECHANISMS

- Shared data structures
- Software interrupts (signals, upcalls)
- Standard interface for user level schedulers

SHARED KERNEL/USER DATA STRUCTURES



A TYPICAL PSYCHE THREAD PACKAGE



COMPARISON WITH KERNEL THREADS

Application	Kernel-level vCPU	User-level Thread
Gaussian elimination	33.8s	22.0s
Parallel sort	44.3s	27.1s

COMPARISON WITH KERNEL THREADS

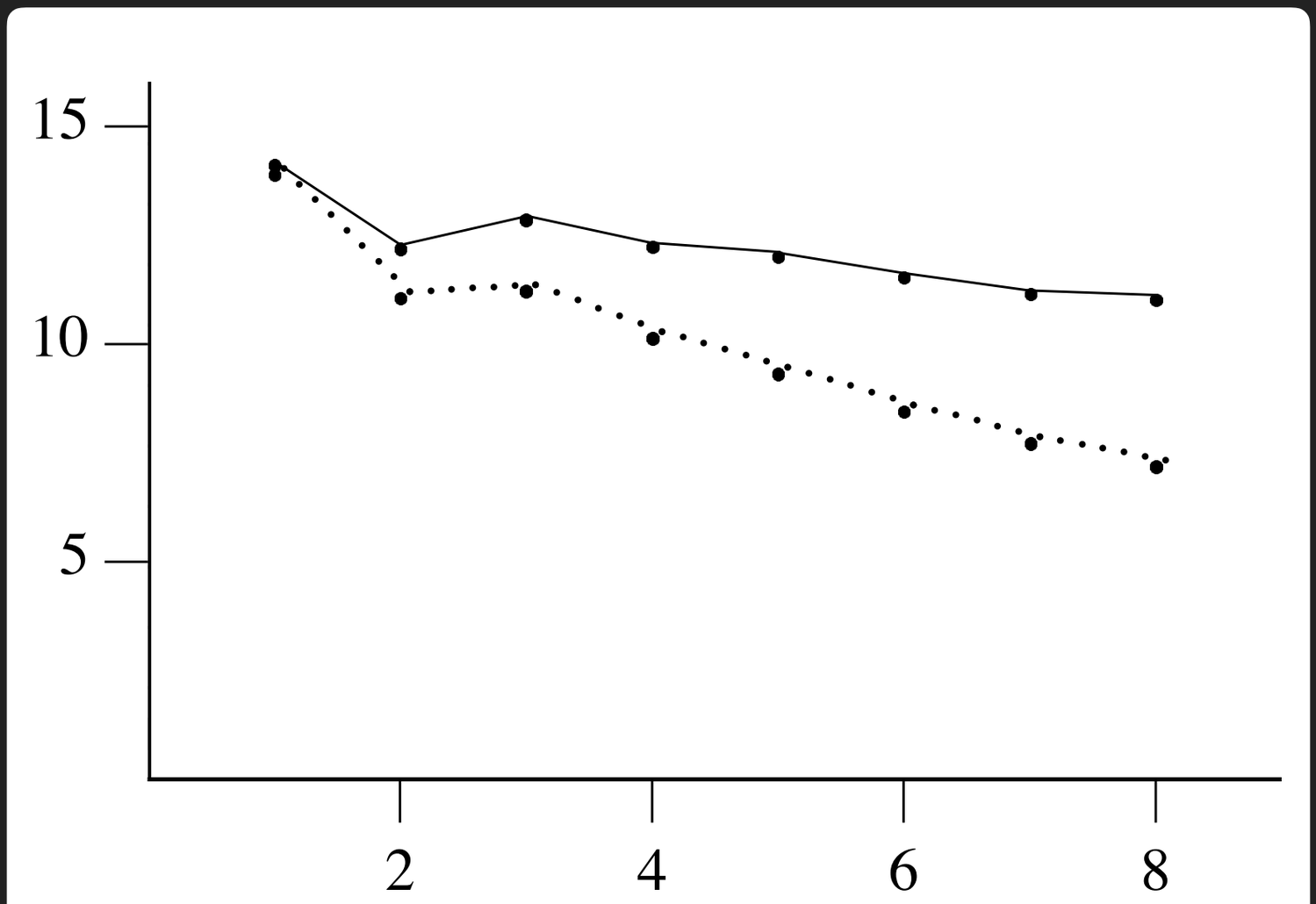


Figure 3: Speedup on 16 physical processors as a function of the number of kernel processes (dotted) or user threads (solid) per processor.

COMPARISON TO CONVENTIONAL THREAD PACKAGES

Multi-Programming Level	Two-minute warning	
	Disabled	Enabled
1	8.5s	8.58s
2	40.1s	16.0s
3	61.6s	23.8s

CONCLUSION

- Kernel threads are inflexible and expensive for fine-grained operations
- User-level threads experience performance losses when blocking in the kernel or are preempted in critical sections
- Accord first-class status to user-level threads

DISCUSSION

- Explicit user-level threading vs. run time-managed parallism
- Validity of micro-benchmarks
- Usefulness for real-time