Paper title: The Multikernel: A New OS Architecture for Scalable Multicore Systems
Paper authors: A. Baumann, P. Barham, P. Dagand, T. Harris, R. Isaacs, S. Peter, T. Roscoe, A. Schupbach, A. Singhania
Your name: Parth Kansara (115135130)

What problem does the paper address? How does it relate to and improve upon previous work in its domain?
The prime focus of the paper is to address the scalability issues of the operating system with advances in the hardware domain. OS optimizations are not portable across hardware and cannot easily adapt to developments in hardware. Further, the diversity in hardware is not only across machines - cores within a machine tend to be diverse. In this case, cores cannot share a single kernel instance since the tradeoffs in performance are different or the cores are inherently different in terms of their ISA. It is also evident from an experiment that cores using shared memory for updating a data structure spend extra cycles in the order of magnitude 3 waiting for an update. In contrast, message passing using asynchronous or pipelined RPC implementation avoids stalling the processors and frees them to perform other tasks. With increase in the number of cores, the traditional cache-coherence protocols also become excessively expensive and need to be replaced.

Multikernel architecture is based on several previous works in the domain. **Auspex** and **IBM System/360** had operating systems resembling distributed systems; multikernel improves the parallelism and deals with increased diversity of underlying machines. **Tornado and K42** presented clustered objects which used partitioning and replication, although they were based on shared data itself. **Microkernel** also employs message-passing, but maintains a shared memory kernel. Lastly, several works in the domain of **distributed operating systems** come close to the concept of multikernel. This paper showcases how multikernel leverages the best parts of all these systems to build a distributed system over a collection of cores.

What are the key contributions of the paper?
Based on the shortcomings of the traditional operating systems, this paper proposes a novel operating system that treats the machine as a network of cores that communicate through message passing and refrain from sharing memory. This idea is based on three principles - explicit communication among cores, hardware-independent design and using state replication over state sharing.

Firstly, except the messaging channels, no shared memory exists between the cores. Using explicit communication over implicit allows information about the shared state to be revealed. Further, this also allows the operating system to leverage common networking optimizations like pipelining or batching. Via message passing, operations can separate requests from responses and thus be productive while waiting for a response. Lastly, as they communicate through well-defined interfaces, there is scope for refining and tolerance to fault.

Secondly, decoupling OS from the hardware has several advantages. Adapting OS to new hardware becomes less cumbersome, while allowing communication algorithms to run independently of the underlying hardware implementation.

Lastly, state replication bolsters OS scalability by reducing load on the system interconnect, dispute for memory and the hassle of synchronizing. The paper also suggests an optimization based on limited sharing between a set of associated cores or threads.

Based on these design principles, the authors present **Barrelfish**, a multikernel OS implementation that proves the desired scalability and performance.

Briefly describe how the paper's experimental methodology supports the paper's conclusions.
The Barrelfish implements the multikernel OS by separating the CPU driver from the monitor components. The CPU driver runs in privileged mode while the monitor process runs in a distinguished user-mode. No state sharing policy allows CPU drivers to be event-driven, single threaded and event-driven. This in turn results in an easily developed small-sized kernel. In contrast to CPU drivers, monitors are independent of the processor and are schedulable. The Barrelfish also uses **System Knowledge Base (SKB)** to maintain knowledge of the underlying hardware which can be utilized for optimization.

Thus, the Barrelfish demonstrates satisfactory performance and meets all the goals set by the authors.

Write down one question you may want to ask the authors.

The authors consistently condemn memory sharing, yet suggest a limited amount of sharing as an optimization on top of message passing. What would be the specific use-cases where this kind of optimization could yield better performance?