**Paper title:** Exokernel: An Operating System Architecture for Application-Level Resource Management
**Paper authors:** Dawson R. Engler, M. Frans Kaashoek, and James O'Toole Jr.
**Your name:** Parth Kansara (115135130)

**What problem does the paper address? How does it relate to and improve upon previous work in its domain?**
This paper looks at enhancing the flexibility and performance of applications by replacing the way traditional operating systems abstract the hardware resources. This high-level abstraction is immutable which hampers the ability of the applications to optimize, inhibits them from modifying the implementation of existing abstractions and limits their flexibility. Discouraging domain-specific implementation of abstractions results in applications paying overhead costs rooting from the trade-offs that are decided by the OS without considering the requirements of the application. This also results in a limited view of the system information to the applications thereby making it difficult for them to apply any customized resource management abstractions.

Previously, several endeavors have been made into developing flexible kernels. ***Hydra*** separated kernel policy and mechanism - a principle extended by exokernel which eliminates mechanism altogether. ***VM/370*** offers virtualization of the base-machine - an expensive and complicated affair that still doesn't address the lack of control for the applications. ***Microkernels*** provided some flexibility over monolithic kernels by distributing the functionality of the kernel over several sub-processes known as servers, running in the user space. However, these microkernels restricted applications from modifying the high-level abstractions which again raises the same kind of limitations as discussed above. ***Cache Kernel*** provides a low-level kernel supporting concurrent application-level OS, however it focuses on reliability and thus limits the extent of concurrency. Exokernel circumvents this by safely exporting the hardware resources and providing abundant freedom to untrusted applications.

**What are the key contributions of the paper?**
The paper designs an exokernel architecture which pushes the OS interface very close to the hardware and transfers all hardware resources to library operating systems. For this, the exokernel aims to isolate protection from management using 3 mechanisms - secure bindings, visible resource revocations and abort protocol.

***Secure binding*** aims to protect individual resources by enforcing authorization at bind time and then implementing access checks at access time. This mechanism is implemented using - hardware mechanisms, software caching and downloading application code. ***Visible resource revocation*** allows the involvement of the applications and notifies the application when the resources are drained. This allows the library OS to save state information or relocate its physical names, thus enhancing performance. ***Abort Protocol*** is a second stage of the revocation protocol to forcefully break the secure bindings and free up resources, which is recorded in a *repossession vector*. This information is passed to the library OS so that it can update the mappings of the lost resource. Exokernel can preserve the state information of the revoked resource by writing it to a memory resource - one which can be pre-specified by the library OS itself.

The paper offers two software implementations - ***Aegis***, an exokernel and ***ExOS***, a library OS. These prototypes demonstrate the four hypotheses - exokernels are very efficient, secure low-level multiplexing of hardware resources can be implemented efficiently, traditional OS abstractions can be implemented efficiently at the application-level & applications can create special-purpose implementations of these abstractions.

**Briefly describe how the paper's experimental methodology supports the paper's conclusions.**
Aegis showcases good performance by monitoring ownership, maintaining a small kernel, caching secure bindings and downloading packet filters along with dynamic code generation for efficient secure binding to the network. It maintains the simplicity of basic primitives, which beats the general primitives of Ultrix in performance, along with superior implementation of exceptions dispatch and control transfer primitives, efficient multiplexing of the processor, memory and the network.

ExOS is able to efficiently implement interprocess communication, virtual memory and remote communication at the application level. It also creates special purpose implementations of these abstractions, which offer improvements in functionality and performance.

**Write down one question you may want to ask the authors.**
While the concept of exokernels is seemingly better than any of the kernels at that time, what could be some limitations preventing its mass adoption that the authors could have foreseen? This is intriguing to me, because unlike most research papers, this one does not discuss any scenarios where exokernels might be at a loss.