0. The Problem
- OSes are in a constant state of flux
- Adapting to new hardware
- Meeting the demands of new software
- Patching bugs and regressions
- Current systems, however, do not optimize for the continuous redesign all systems must undergo

1a. OS Researchers Want...
- A flexible OS that can be extended easily to experiment with new hardware and algorithms
- The ability to radically redesign subsystems without rewriting most of the OS
- A small trusted code base, with a short learning curve

1b. Application Developers Want...
- To be free from the hindrance of forced OS abstractions
- Why use files or sockets? Can we do better?
- Control over performance critical paths
- To specialize code using application-specific knowledge
  - E.g., customize network access

1c. OS Educators Want...
- A simple and real OS that can illustrate core concepts
- To be able to demonstrate various OS architectures
- A ‘real world’ system (not a simulation) that can be used by students for everyday tasks

2. Rethinking OS Evaluation
- Kaashoek’s Law: “There are plenty of good reasons to design a new OS, but performance is not one of them”
- Ganger’s Corollary: “Anything you can do, they can do”
- Focusing on Flexibility allows a system to efficiently achieve any performance goal or feature set

3. Flexibility through Statelessness
- Flexibility (for our purposes) means components are in userspace and untrusted
- A stateless kernel simply operates on data for processes
- Permission is granted by the resource mapping (an implicit capability)
- We extend microkernel and exokernel migration of OS abstractions to userspace
  - use IOMMU to place untrusted drivers in userspace
  - use self-virtualizing devices (e.g., NICs)
  - one driver per application

4. Userspace Resource Management
- The XOmB kernel manages all resources as contiguous regions of virtual memory (segments)
- heap, files, shared-memory, memory-mapped devices, address spaces of child and parent processes
- Segments are subtrees of the page table
- kernel sets R/W/X permissions on resource mappings
- Segments can be shared among processes, with differing permissions, by editing a single Page Table Entry

5. Userspace CPU and RAM Allocation
- CPU can be scheduled entirely in userspace
- Our approach is inspired by CPU Inheritance Scheduling and Scheduler Activations
- Non-blocking system calls and a lack of kernel managed CPU context simplify our dispatch mechanism
- Removing scheduling from the kernel is key to enabling our stateless, segment based kernel interface
- Using the same process hierarchy as CPU allocation we can also flexibly allocate memory in userspace (DRAM and Storage Class Memory)

6. Benefits of Statelessness
- XOmB’s stateless design and small code base has allowed the implementation of kernel Hot-Swap
- Upgrades kernel without rebooting or patch modifications
- Implemented by an undergrad without prior exposure to XOmB in 2.5 months

7. Conclusion
- XOmB is a stateless kernel providing maximum flexibility
- XOmB is practical for industry/research AND simple enough for education
- Flexibility is a first-class OS feature