Dawson R. Engler, M. Frans Kaashoek, and James O'Toole Jr.

1. What is the overriding goal of exokernel? How is this different than what previous extensible systems did (Nucleus, Mach, HYDRA)? How is this different from a VMM?

- separate protection from management
- lowest level primitives possible:
  - more efficient
  - more flexibility
  - export hardware resources
  - similar goals, but more extreme
  - compare - don't try to abstract resources
    - or virtualize them (like VMM)
  - not even "mechanism"
2. What are exokernel's design principles?

- securely expose hw
  - instr.
  - resources
    (avoid resource management)
  let libOS do

- expose allocation
  let apps request physical resources
  (don't make implicit)
  - this app?
  - this page?

- expose name

- expose revocation
  - if something taken away, let app know
3. Where does policy belong? How does exokernel handle conflicts/competing applications?

`in libOS`

`if managed by same libOS → fine`

`otherwise ??`

"must contain policy to arbitrate between competing libOSes"
4. What is the purpose of a **secure binding**? Why can secure bindings achieve good performance? How can secure bindings be used to multiplex physical memory? To multiplex the network?

- Guard resources efficiently
- Decouple authorization from use

1) Primitives for setting up should be a good match with how (e.g., kernel)

2) Only need to be done at bind, not access

"Mem: check capability to access page on TLB miss (fast access)"

"Network:"

- Multiplex network regimes: interpret incoming msgs
- Download "safe" code into kernel to do this interpretation"
5. What is an Application-Specific Safe Handler (ASH) and why is it useful?

- Code run in kernel on behalf of app
- Faster with switches out
6. Why is resource revocation **visible** to applications using an exokernel?

- Traditional OS: invisible – just take time slice or page away

- ExO: believe app should be able to respond (save regs on context switch) (page out io/phys. mem)
7. Why is an *abort protocol* needed?

*If libDS doesn't revoke as desired*

1) *Could kill libDS*

2) *Tell libDS gone*  
   + guarantee some minimal resources
8. How did the authors demonstrate the flexibility of the exokernel architecture?

1) Aegis is efficient w/ basic costs

Section 7: ExOS

- extensible RPC (who saves reply)
- inverted page table (instead of linear)
- extensible scheduler
  - stride
9. How can various resources be “multiplexed” without policy decisions?

2) \[
\begin{array}{c}
A_1 \\
\text{buffercode}
\end{array}
\]
should they get to share?

1) \[
\begin{array}{c}
\text{libos} \\
\text{libos}
\end{array}
\]  \\
1 cpu

who, when? policy!

3) \[
\begin{array}{c}
A_1 \rightarrow A_2 \\
\text{network}
\end{array}
\]
A_3

have to download code

point of mux is decision point!
10. Conclusions?

Other problems?
- large executables
- portability??

What was going on in OS world?
- OS was viewed as couldn't be changed, very complex
  - HP-UX, VMS, Solaris, Windows...

- just let interested developers change source code directly
  - Linux?