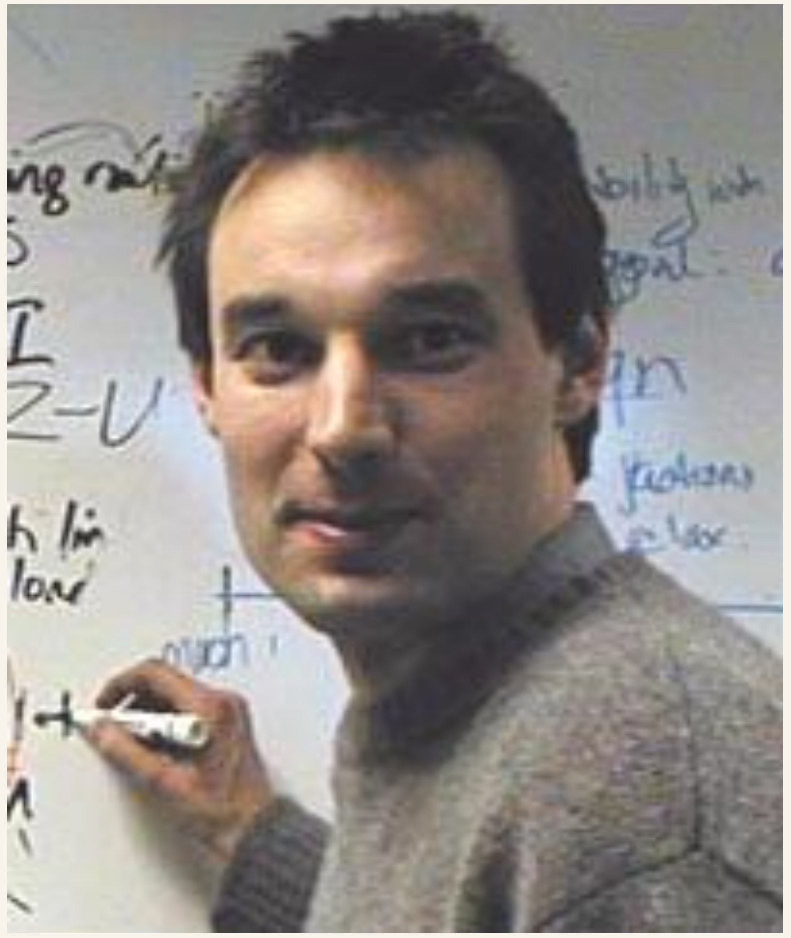
Extensible Kernels

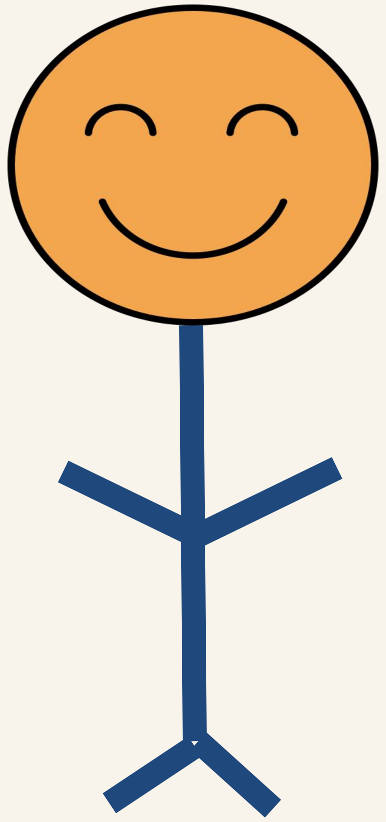
Presentation by Lindsey Bowen

Meet the Authors

Dawson Engler



M. Frans Kaashoek

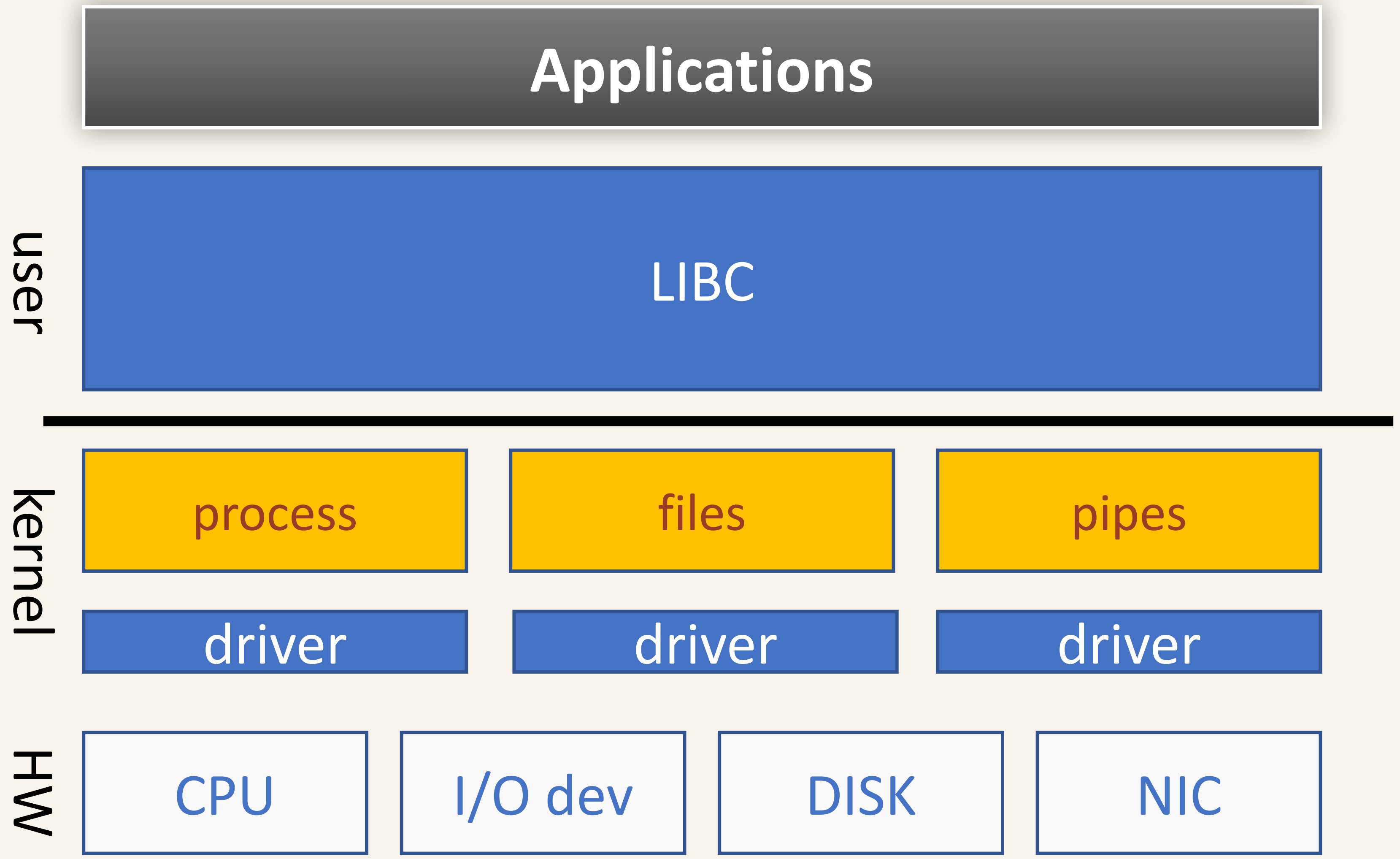


James O’Toole Jr.

Where are we Now?

Monolithic Kernels

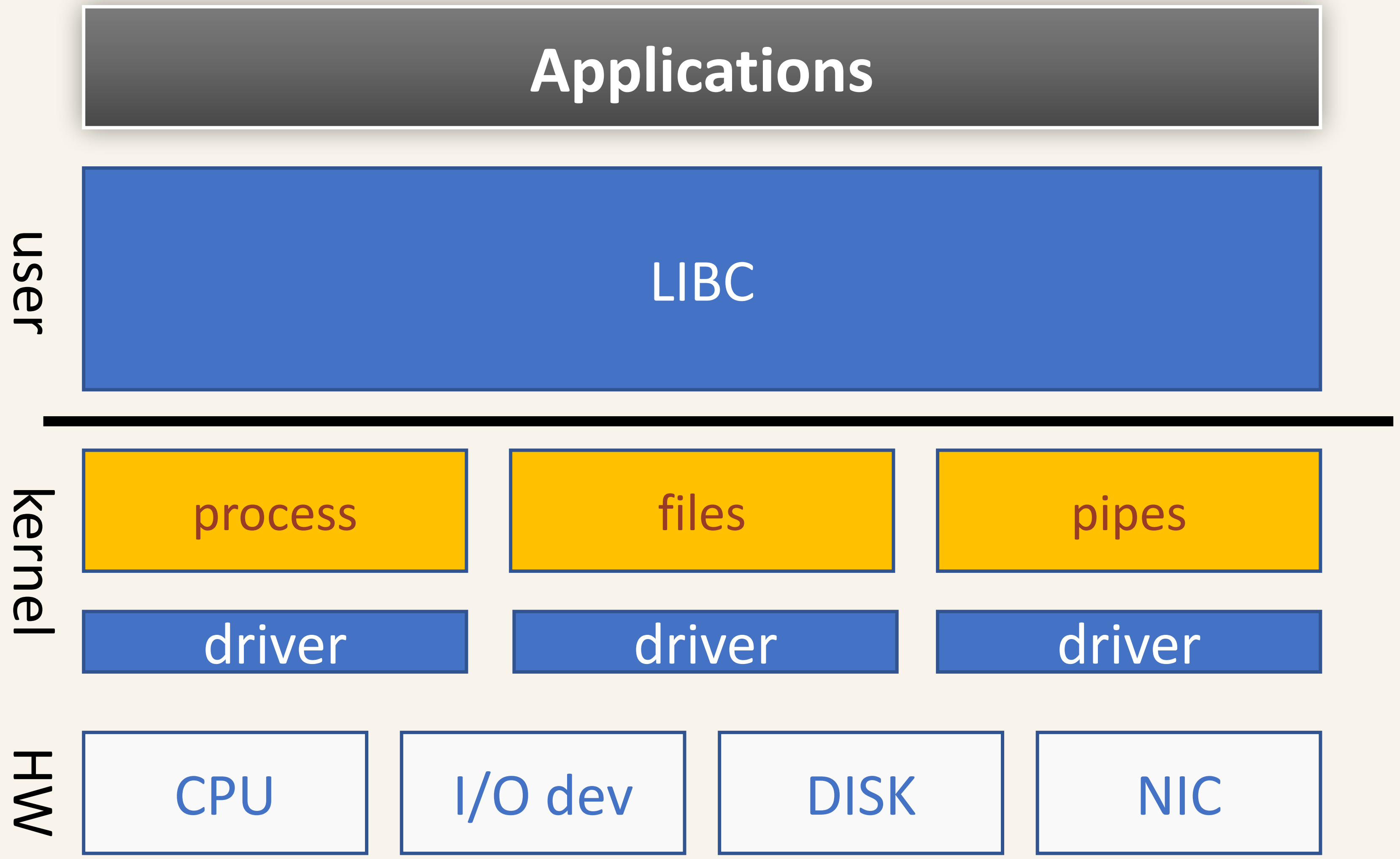
- All applications share a single OS
- OS manages and secures system resources through high level abstractions
- So awesome! Everything is all set to go. ☺



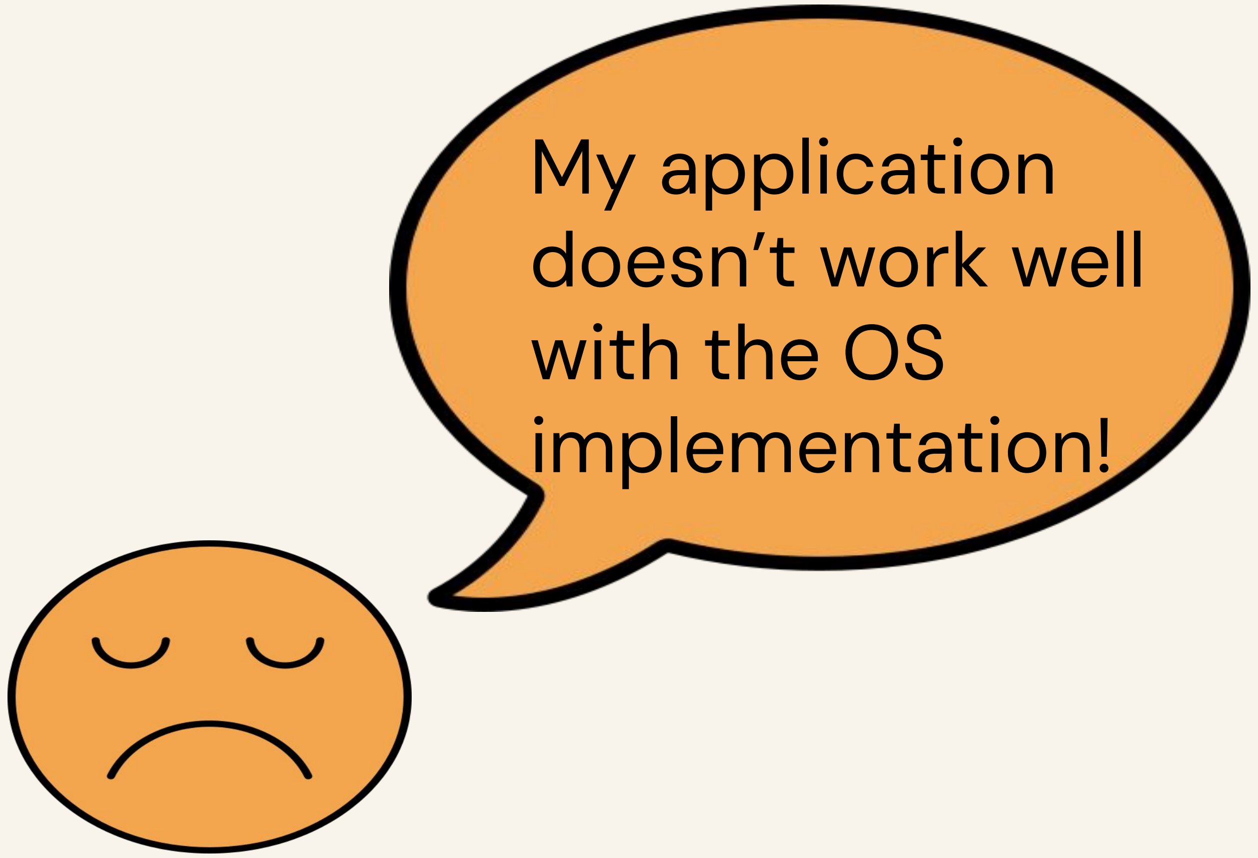
Where are we Now?

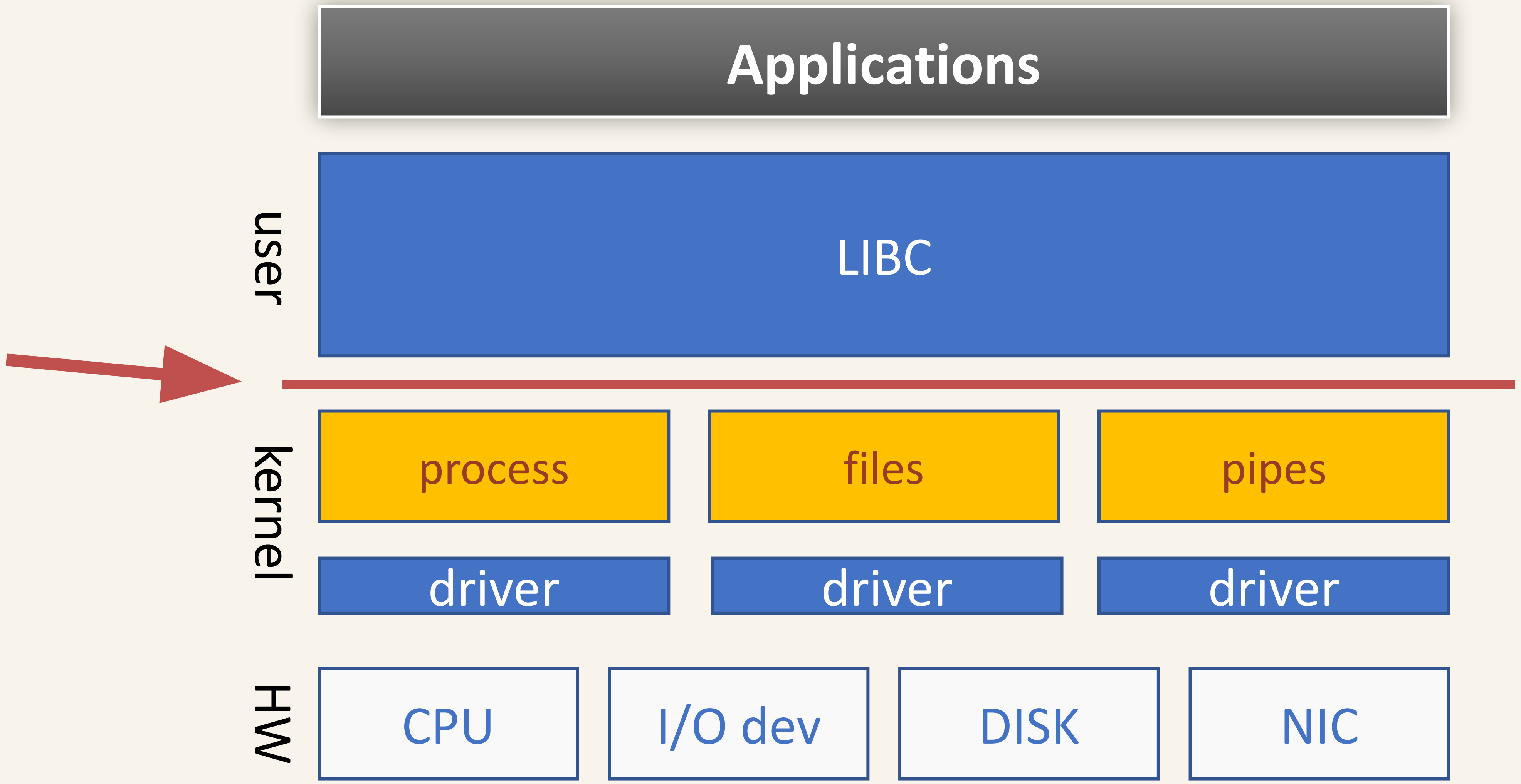
Monolithic Implementation

- Try to optimize for a wide variety of workloads
- Unchangeable from the application layer
  - Applications are untrusted
- Guess an application's future move by using heuristics.



What’s the Problem?


My application doesn’t work well with the OS implementation!



Applications

user

LIBC

kernel

process

files

pipes

driver

driver

driver

HW

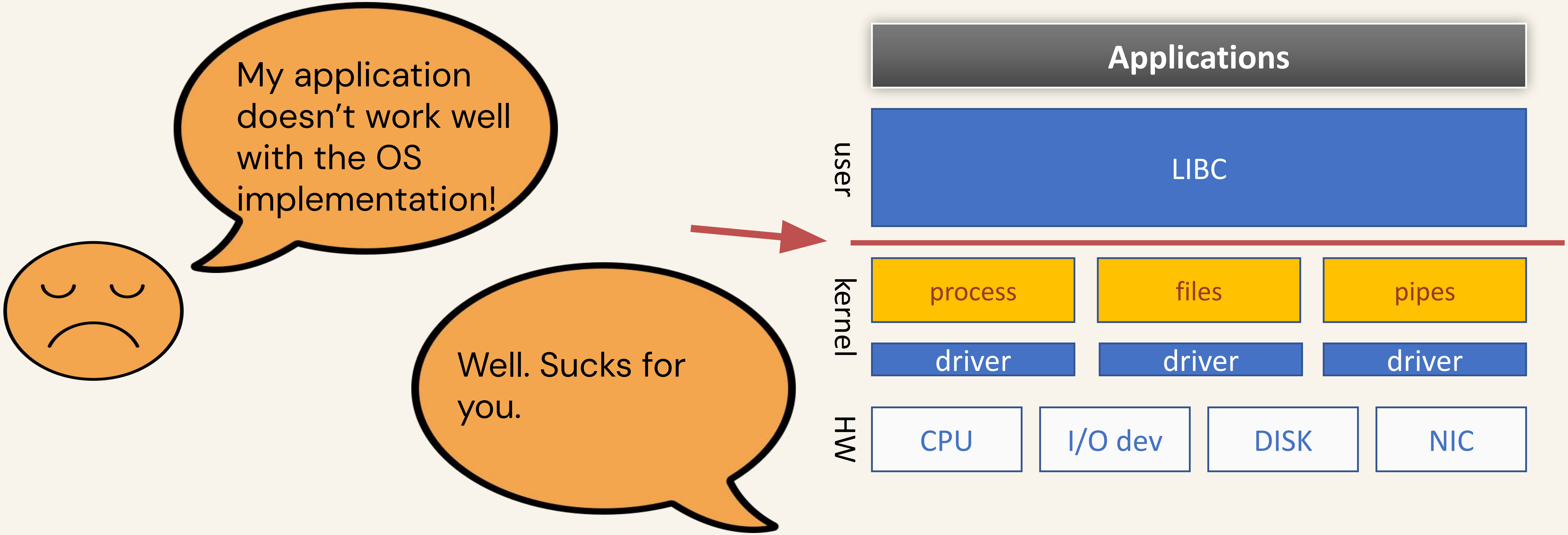
CPU

I/O dev

DISK

NIC

What’s the Problem?



My application doesn’t work well with the OS implementation!

Well. Sucks for you.

Applications

user

LIBC

kernel

process

files

pipes

driver

driver

driver

HW

CPU

I/O dev

DISK

NIC

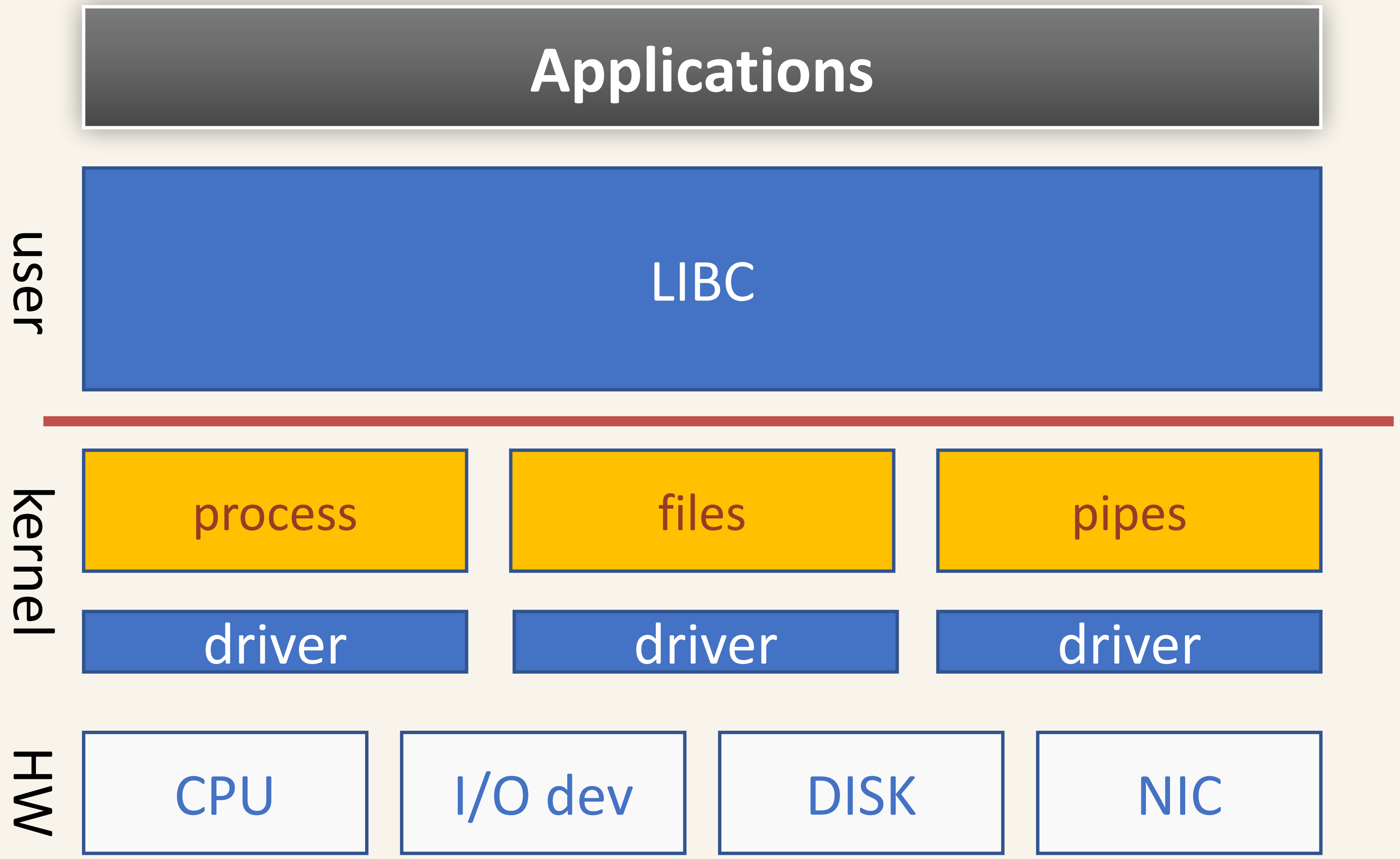
Who Cares?

Michael Stonebraker (~1980)

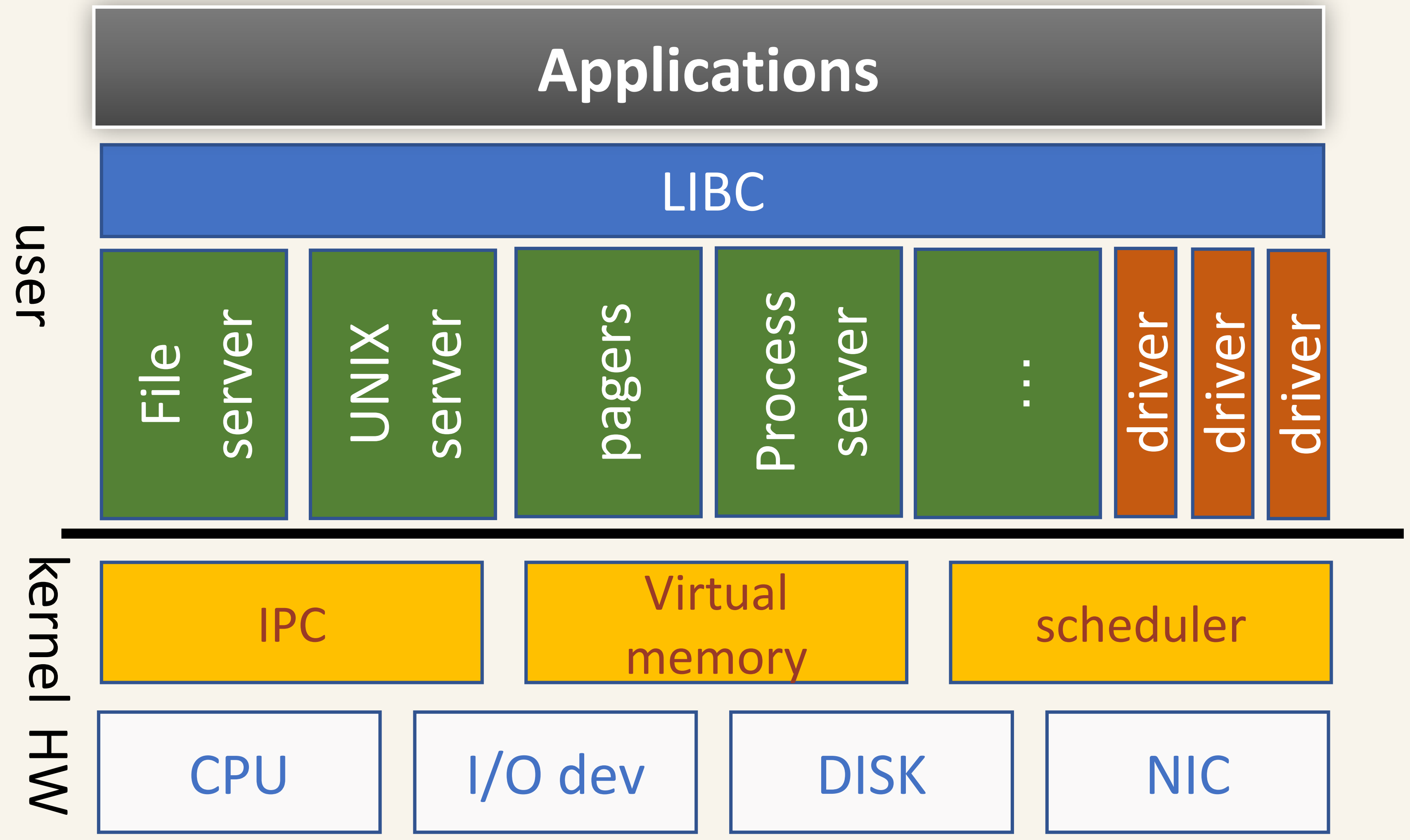
- Michael says current OS services are not suitable for database systems!
  - File buffer cache LRU replacement strategy is bad for non-rereferenced blocks.
  - The DBMS has to re-implement the buffer cache to provide the correct access pattern
- **Sound familiar?**

The End-to-End Argument

- **At which level should abstractions be exposed?**
- “General purpose implementations of abstractions force applications that do not need a given feature to pay substantial overhead costs”
- “The lower level a primitive, the more efficiently it can be implemented”
- Do you buy the end-to-end argument?

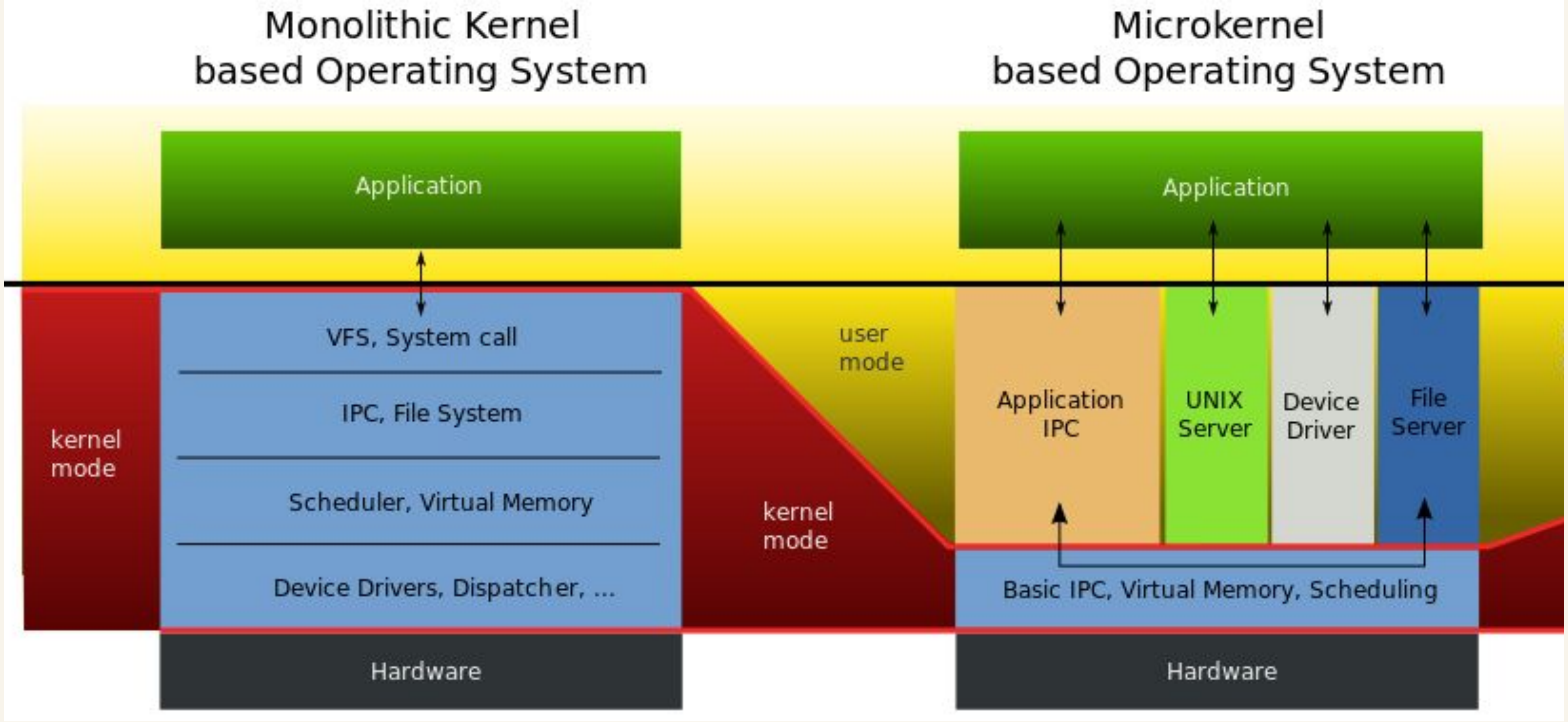


Microkernels

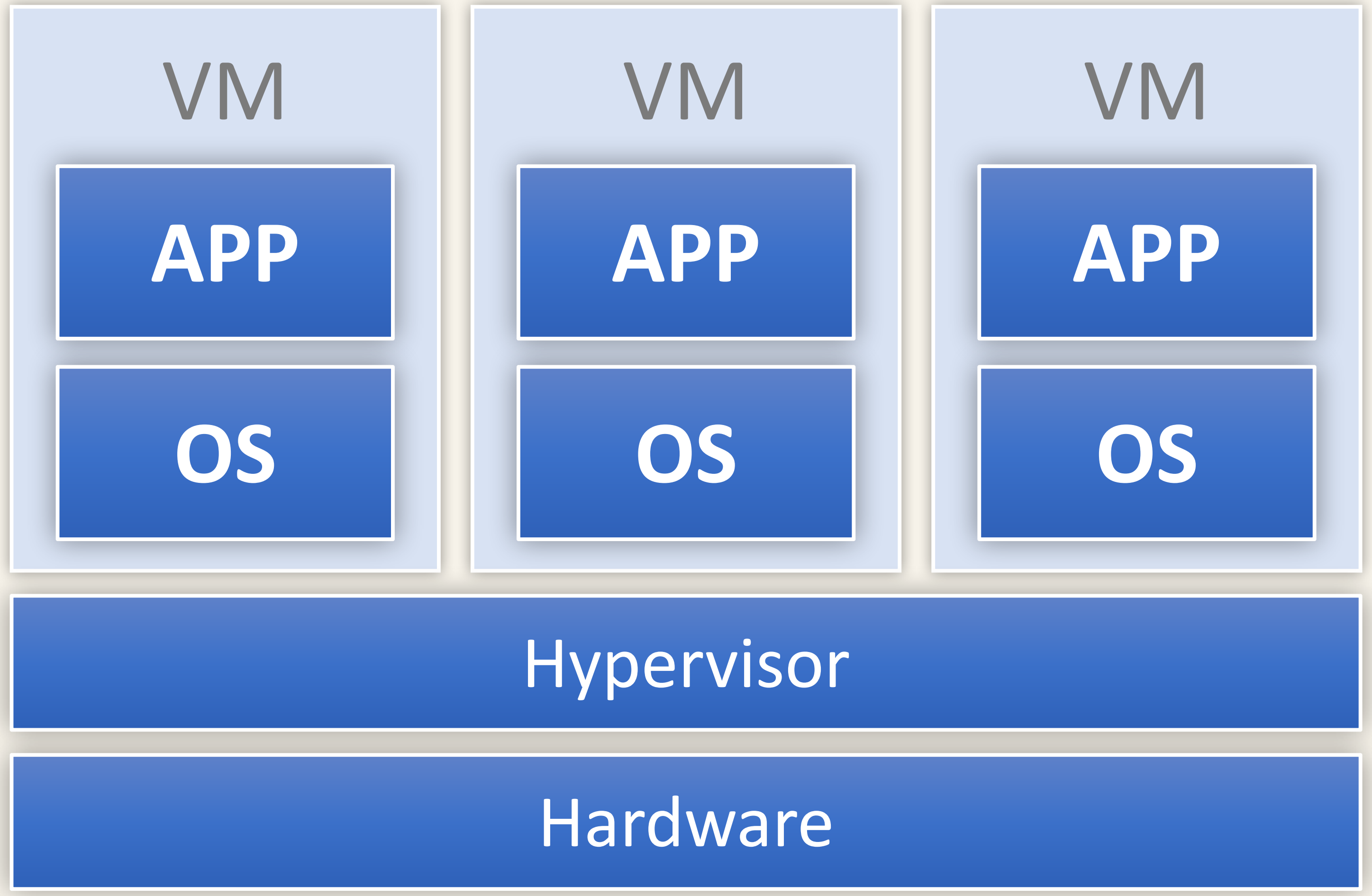


- Minimize what's provided by the OS
- Move abstractions to user space
- **Problems?**
  - Slow (kernel crossings)
  - Extensibility still limited

Monolithic v. Microkernels



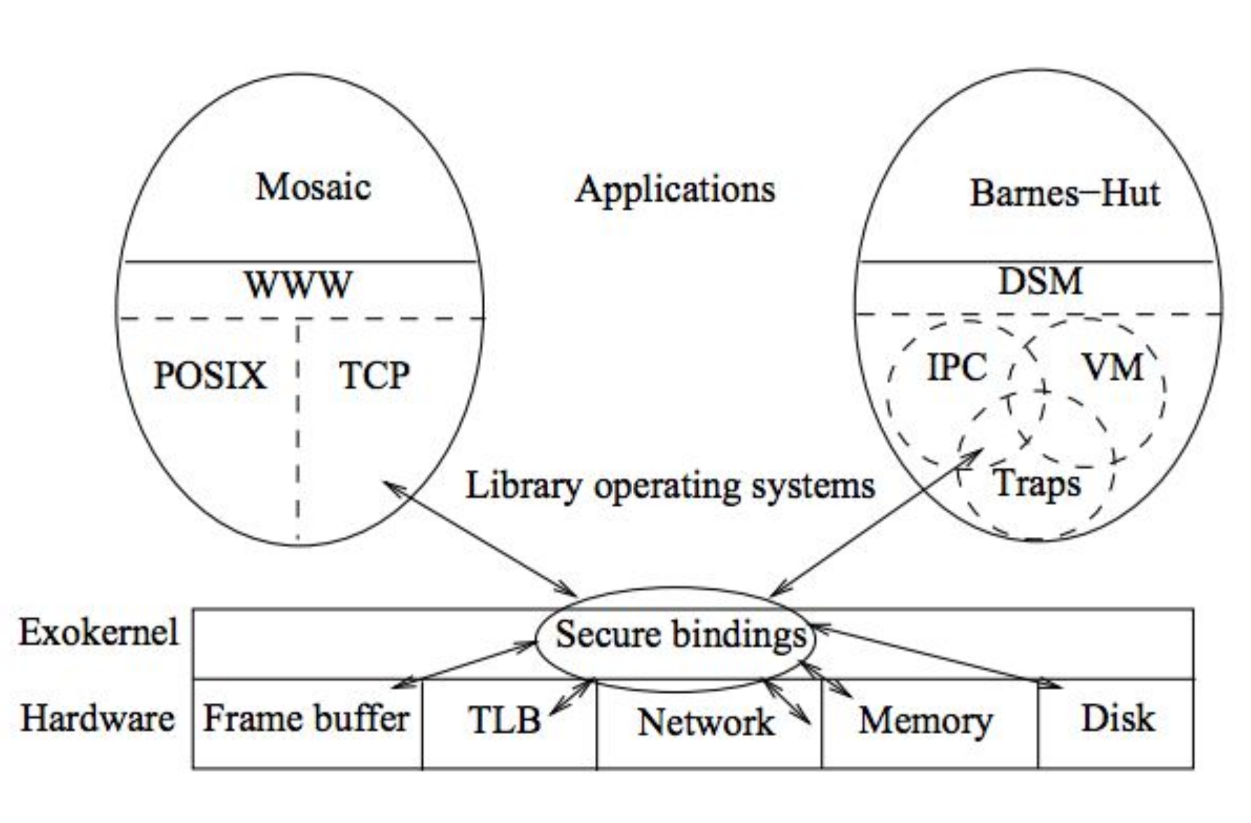
Virtual Machines



- Ok fine you can run whatever OS you want
- Yay! Our hypervisor interface is very low-level
- **Problems?**
  - Extensible?
  - Scalable?

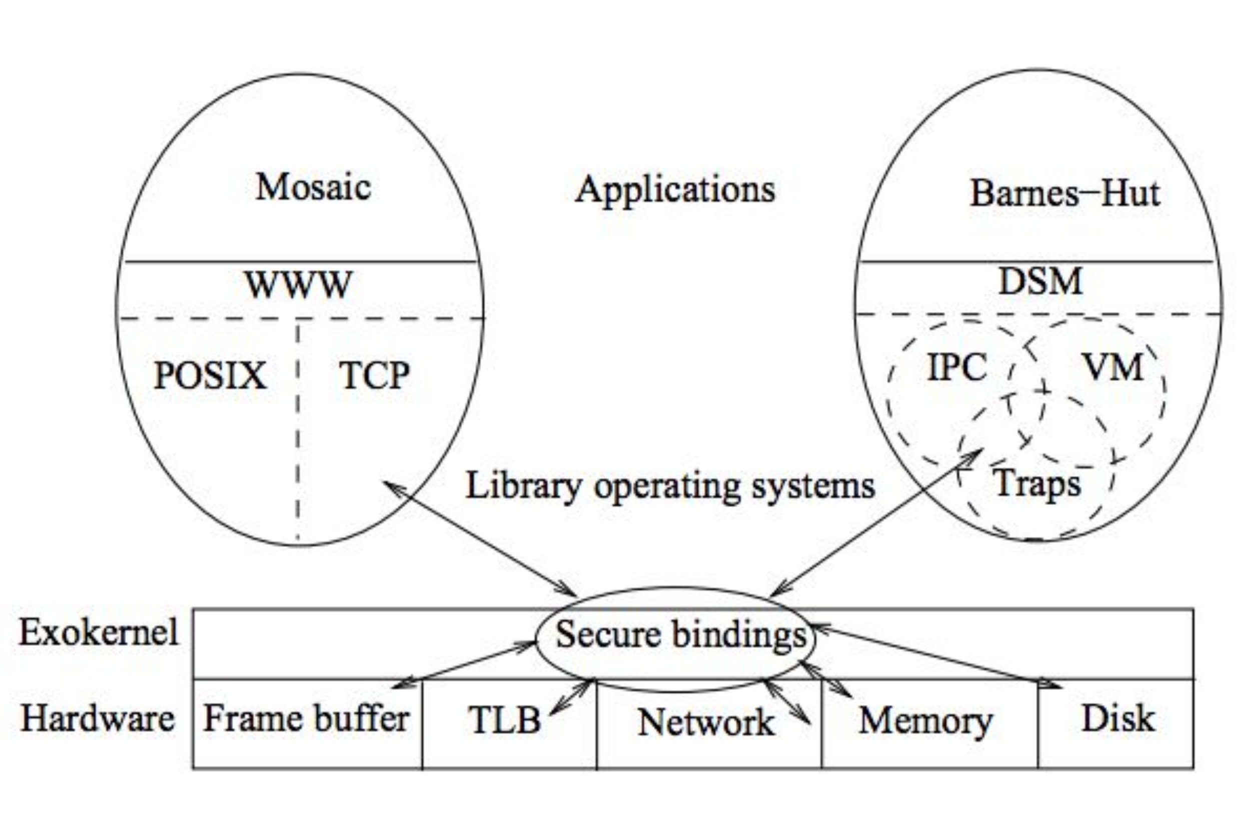
Exokernel Hypothesis

- **Low level** multiplexing is more efficient
- Traditional OS abstractions can be **implemented more efficiently at the application level**
- **Special purpose implementations** for these abstractions will allow applications to gain efficiency in resource usage.



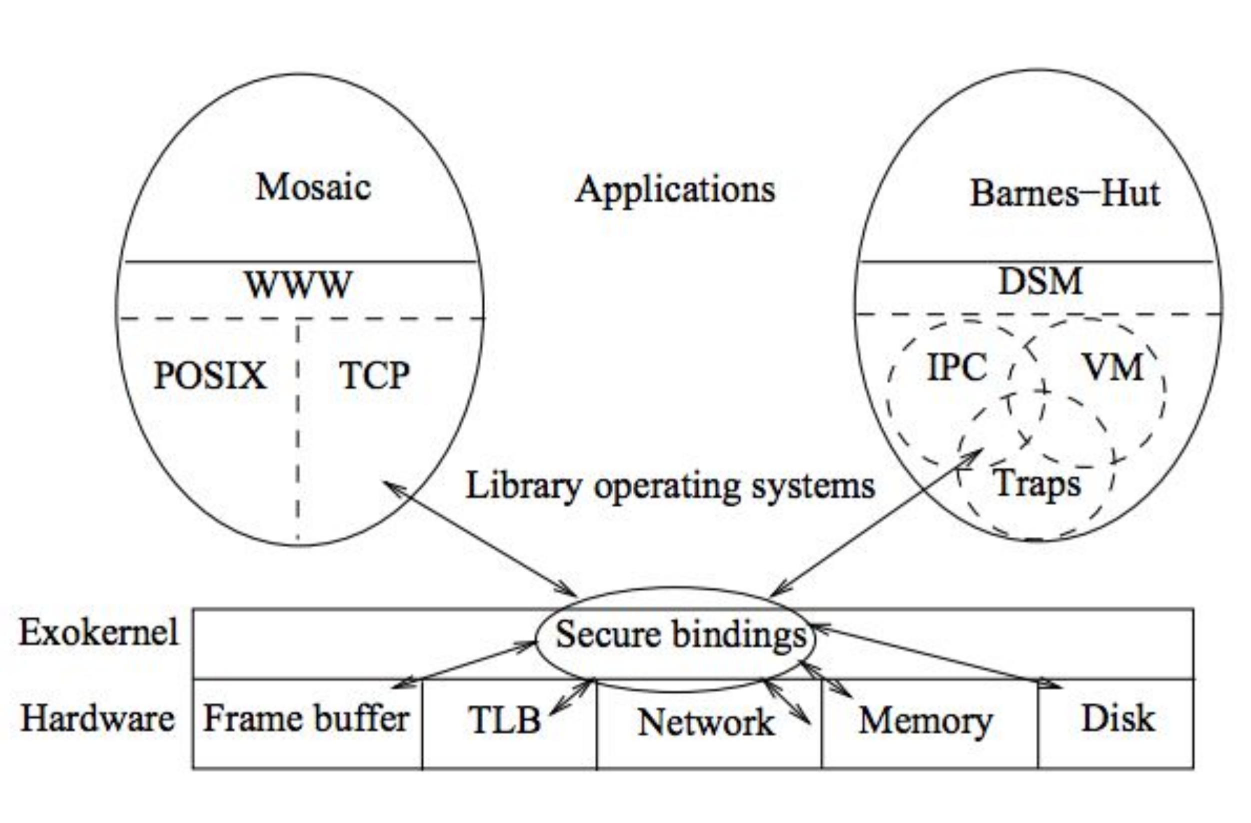
Exokernel Policy

- **Separate resource protection from management.**
  - Securely multiplex resources, but leave management to the user level.
- Allow applications to choose the implementations that work best for their use case.



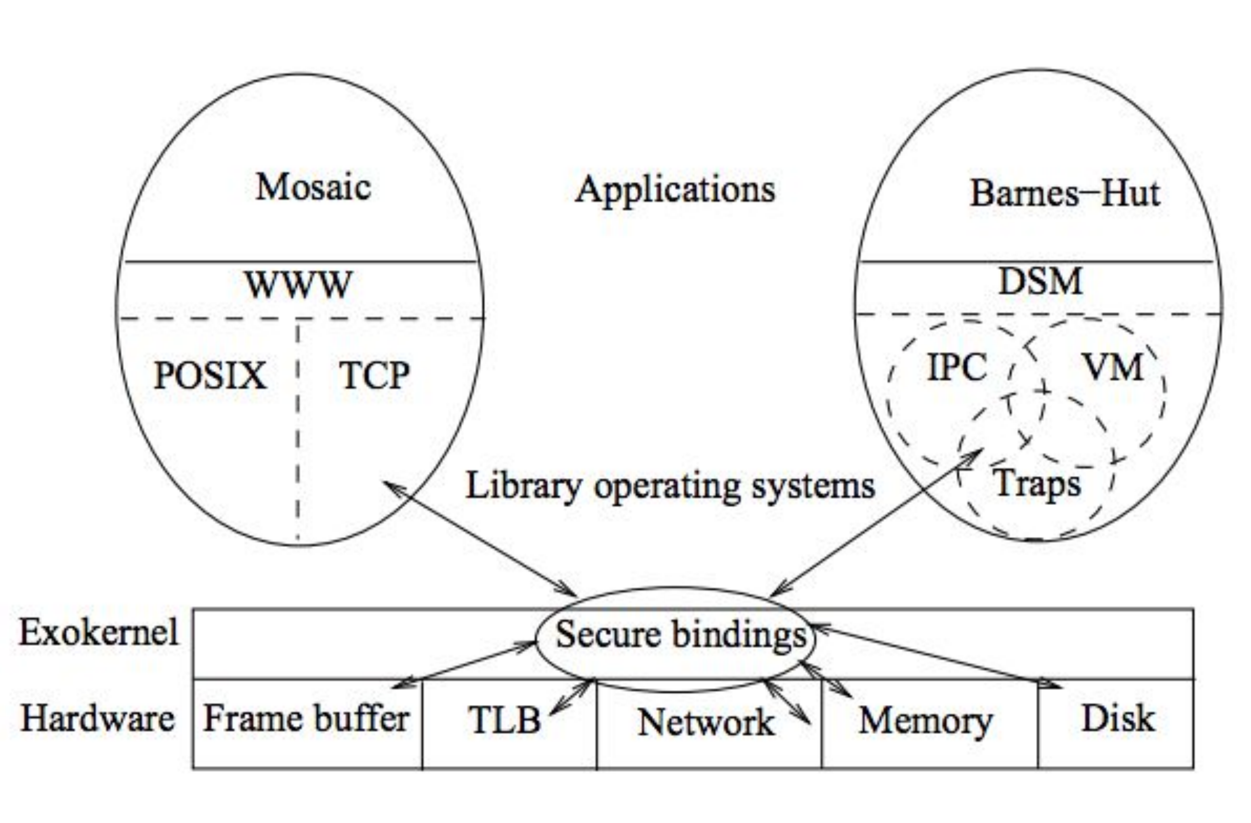
LibOS Policy

- **Portability**
  - Implement POSIX compliant calls
  - Or don't!
- **Security**
  - LibOS not depended on by other applications
  - Library can trust the application all it wants!

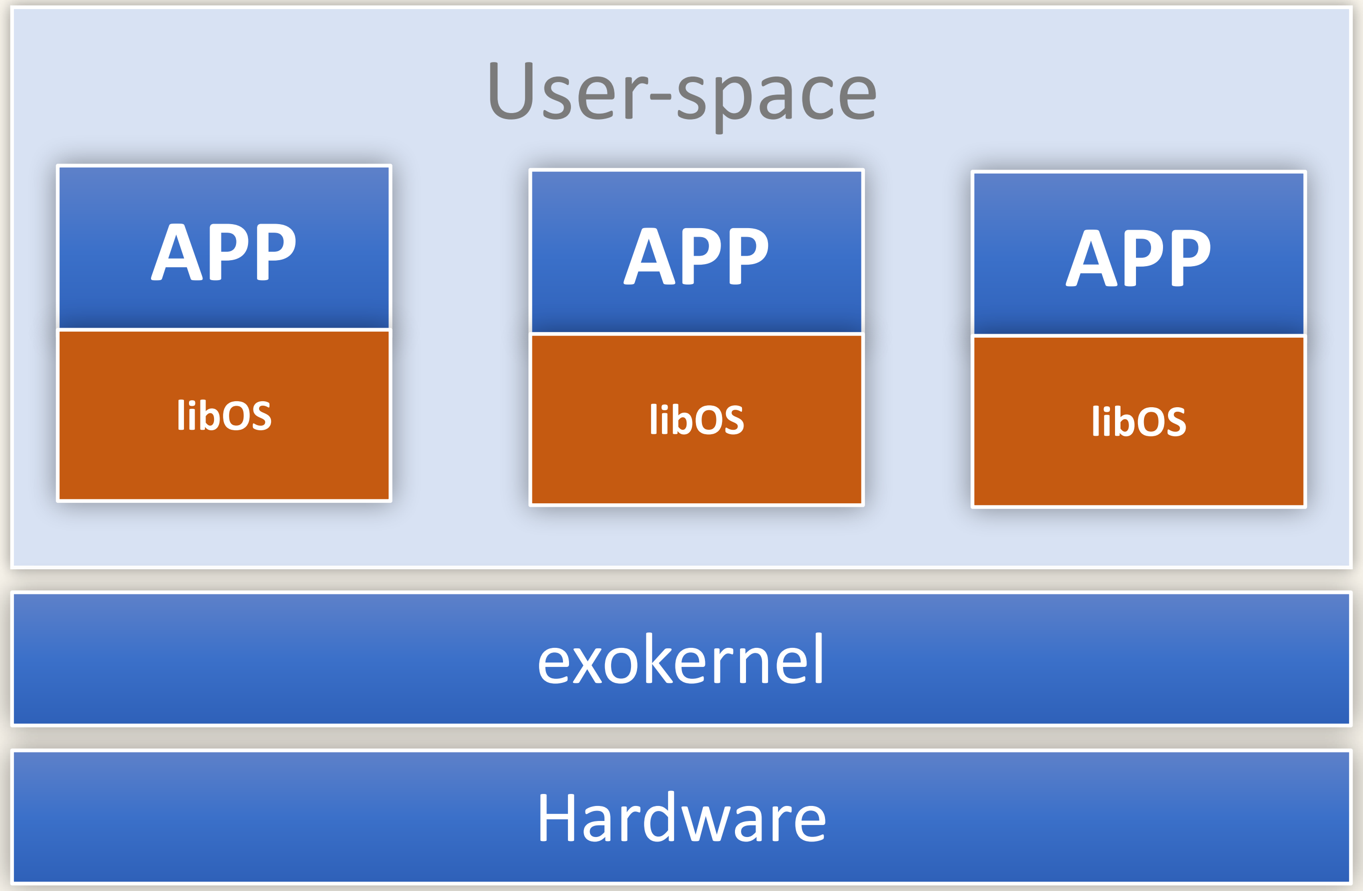


Discussion

- What are the benefits of this design over a monolithic OS?
- Which OS services might have the most trouble separating protection from management?
- Is the exokernel doing enough to be useful?

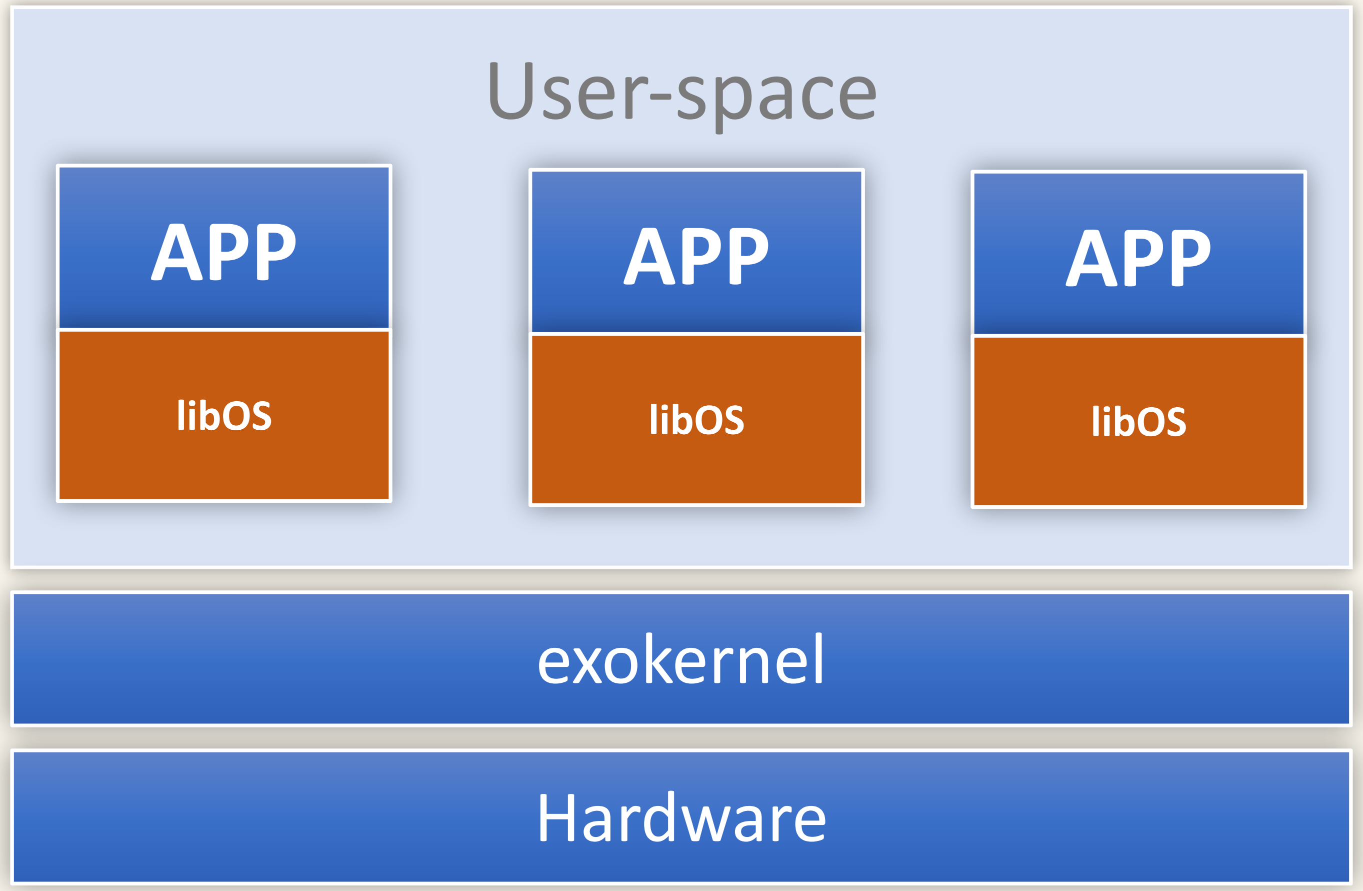


Exokernel Mechanisms



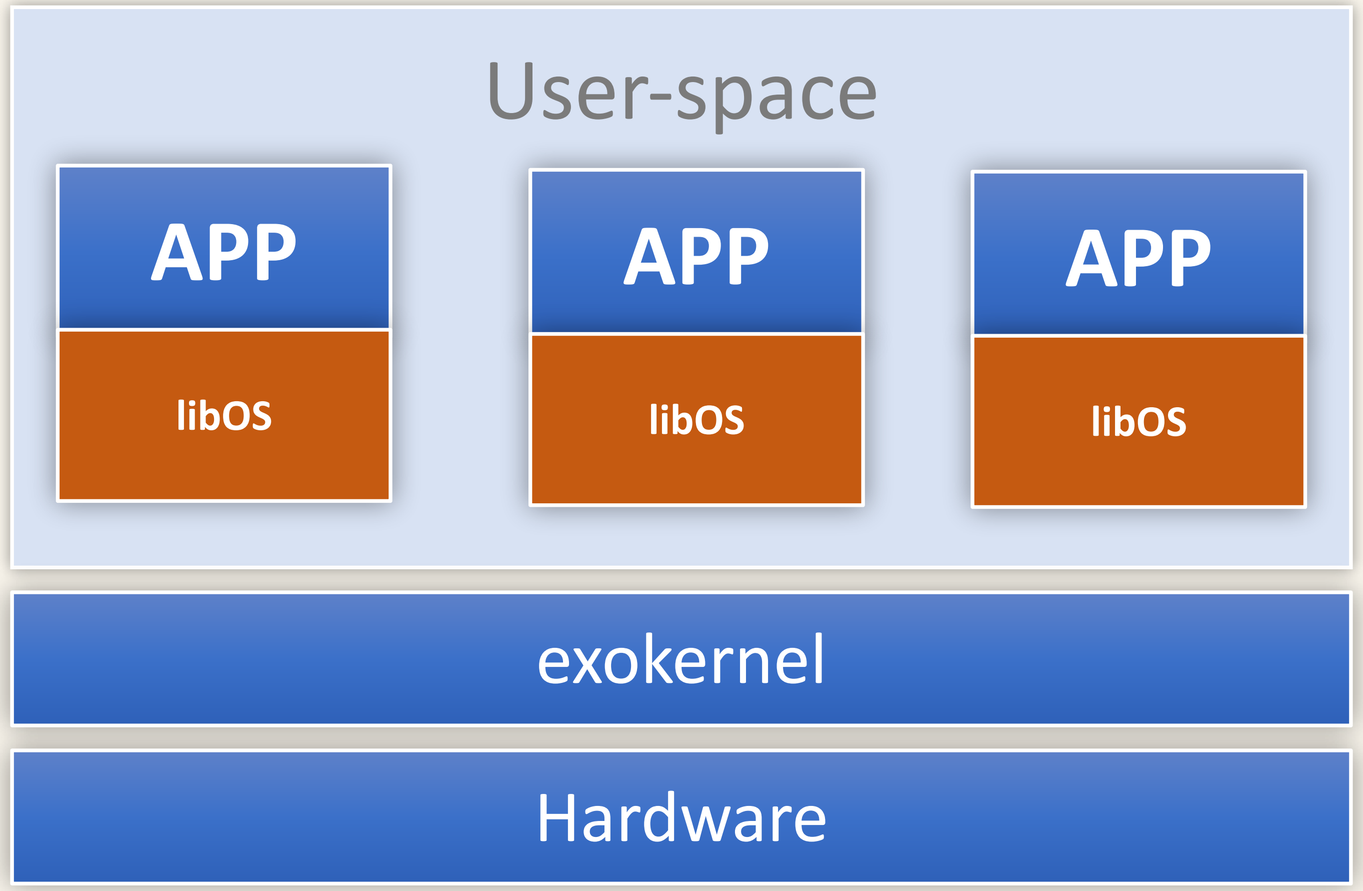
- Track ownership, guard usage, revoke access
- Export freelists, disk arm positions, cached TLB entries, etc.
- Secure bindings
- Visible revocation
- Abort protocol

Secure Bindings



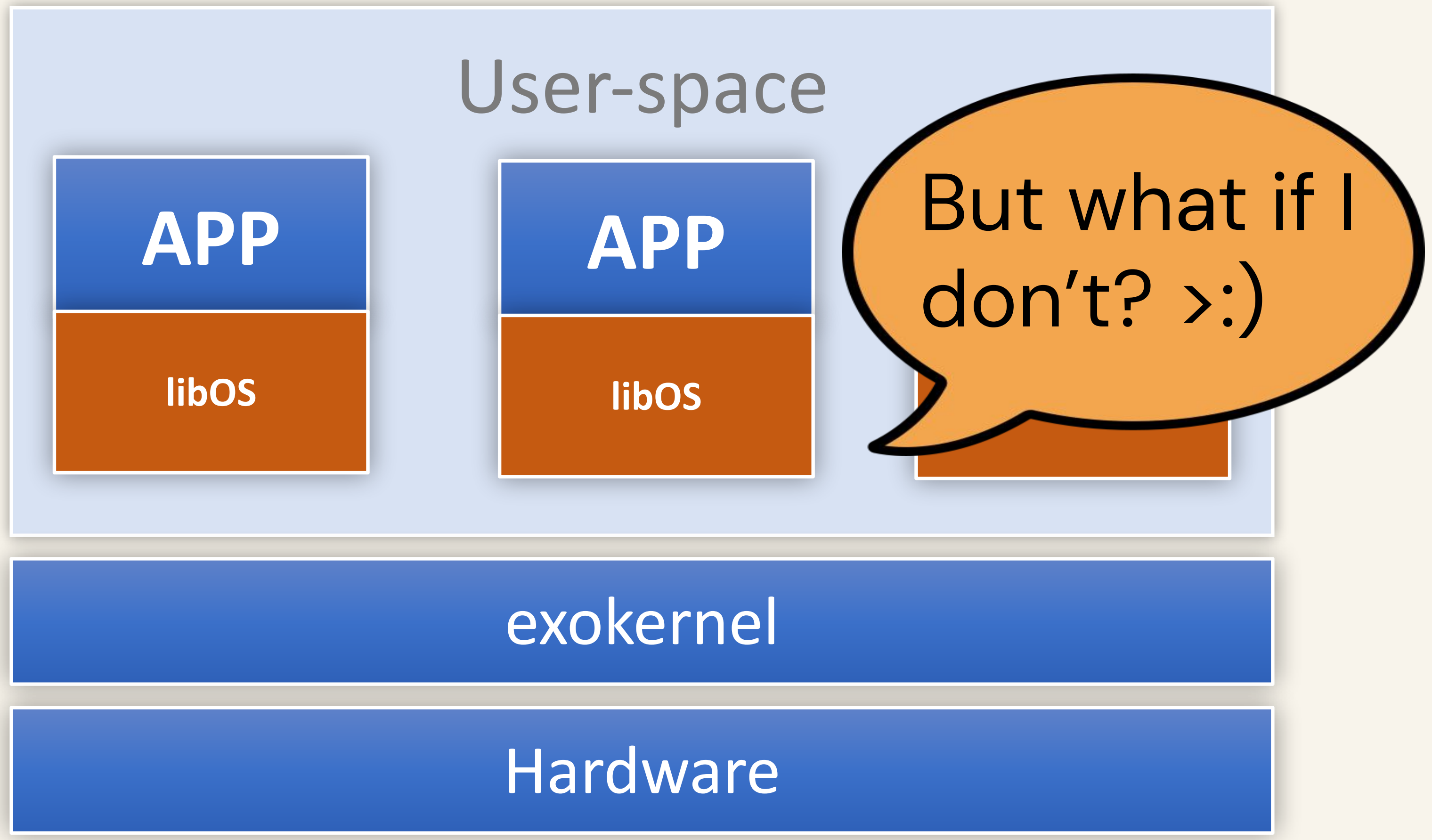
- Bind at large granularity, access at small granularity
  - Check access at bind time not access time
  - Use capabilities to share resources
  - Ex: Check TLB entry at load time for the page, not during address translation
- Protect resources without understanding them

Visible Revocation



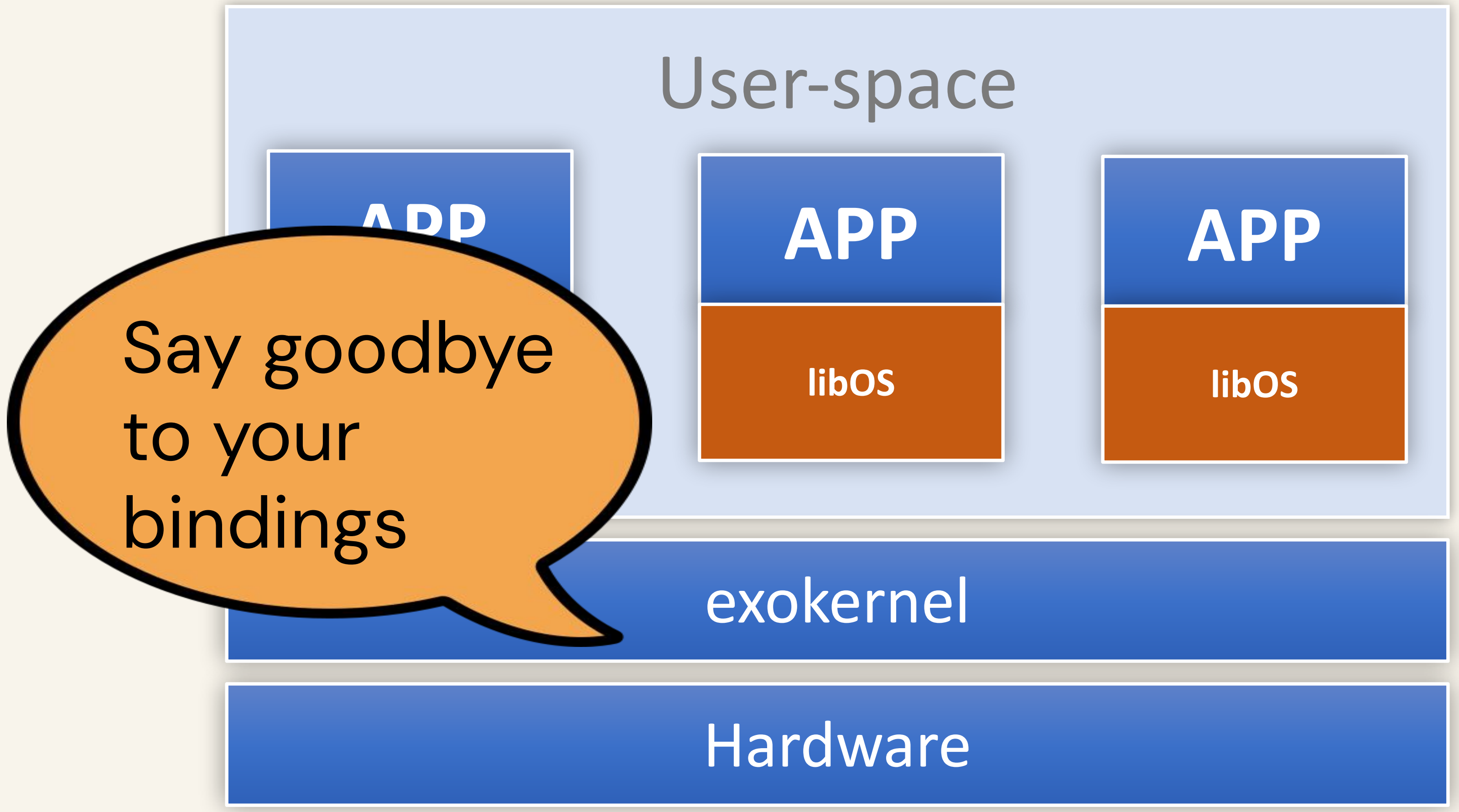
- **Before**: OS can take back whatever resource it wants **without** informing the application
- **Now:** Exokernel asks libOS to give back a resource
  - libOS can decide which resource to give up.

Visible Revocation



- **Before**: OS can take back whatever resource it wants **without** informing the application
- **Now:** Exokernel asks libOS to give back a resource
  - libOS can decide which resource to give up.

Abort Protocol



- If the libOS does not comply
  - Threaten with imperative (you have 5 μs!)
  - Break all secure bindings and **inform the libOS**
- Where should I store vital information that can't be revoked?
  - Arbitrary number of guaranteed pages.

Downloading into the Kernel

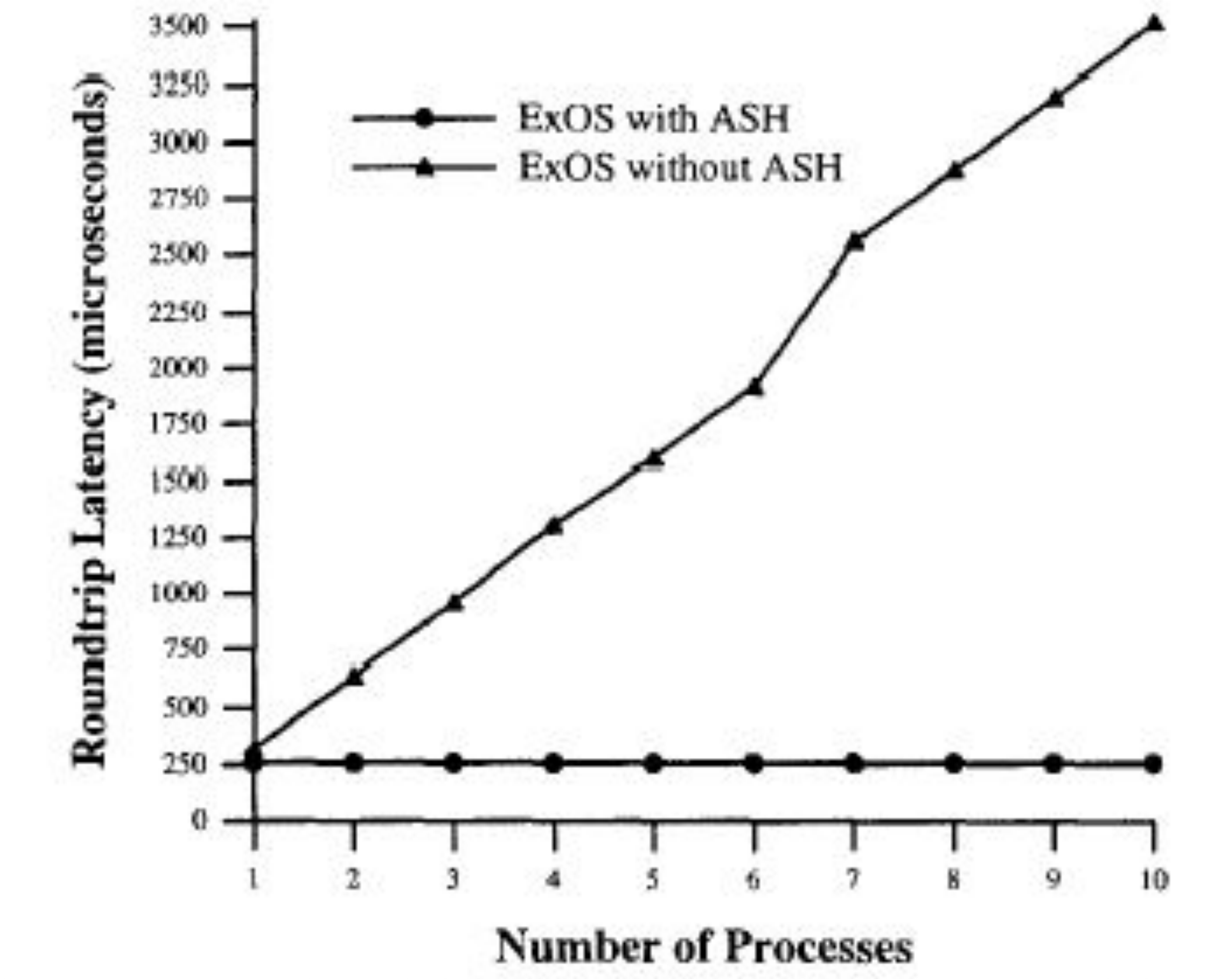


Figure 2: Average roundtrip latency with increasing number of active processes on receiver.

- How should we efficiently multiplex the network?
  - Load handlers for application specific messages into the kernel
  - Written in safe language: check for loops, memory references, etc.
- Now we don't need to context switch to respond!

So cool, right?

Downloading into the Kernel

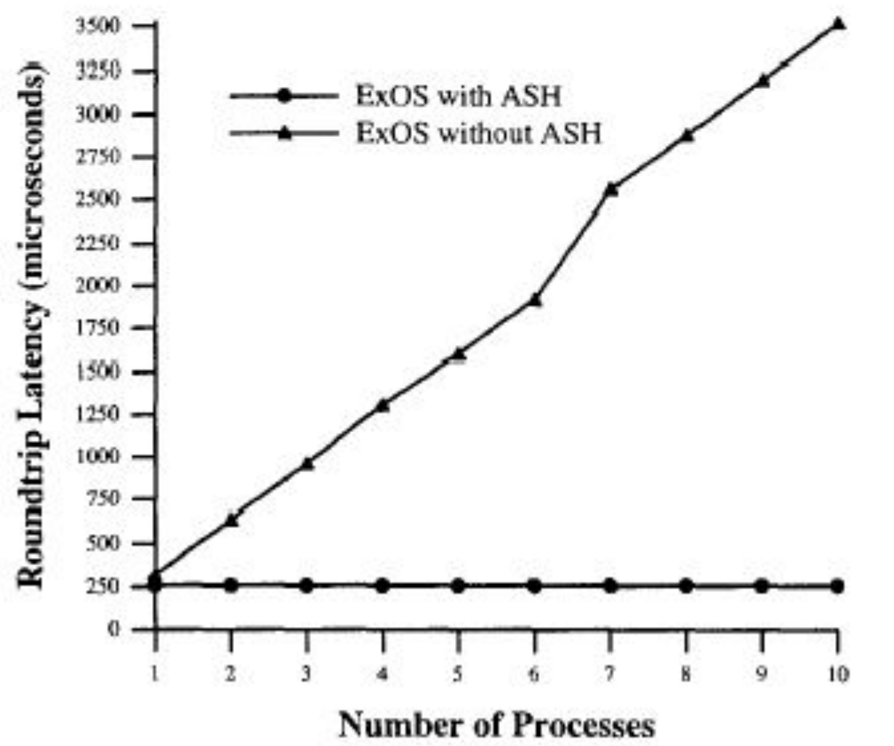
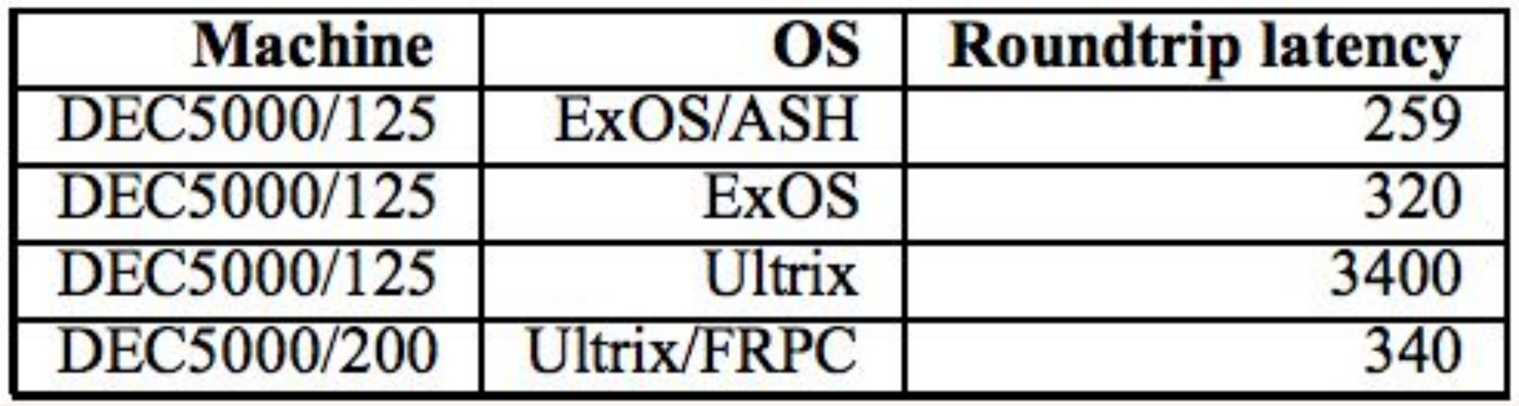


Figure 2: Average roundtrip latency with increasing number of active processes on receiver.

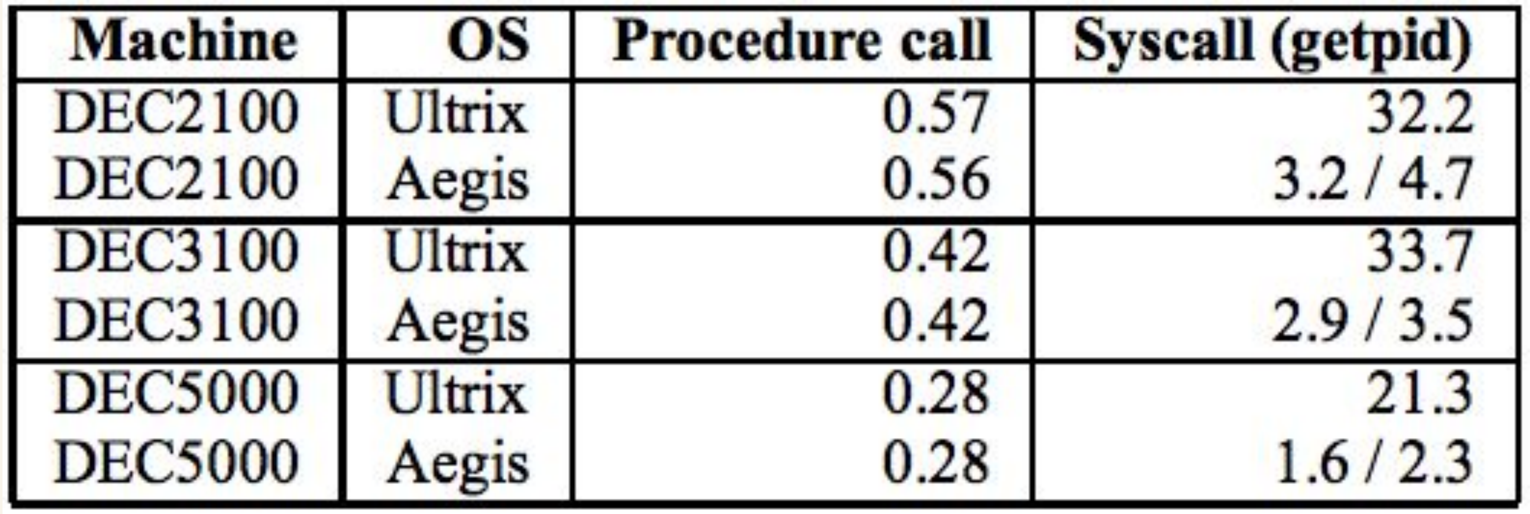


Machine	OS	Roundtrip latency
DEC5000/125	ExOS/ASH	259
DEC5000/125	ExOS	320
DEC5000/125	Ultrix	3400
DEC5000/200	Ultrix/FRPC	340

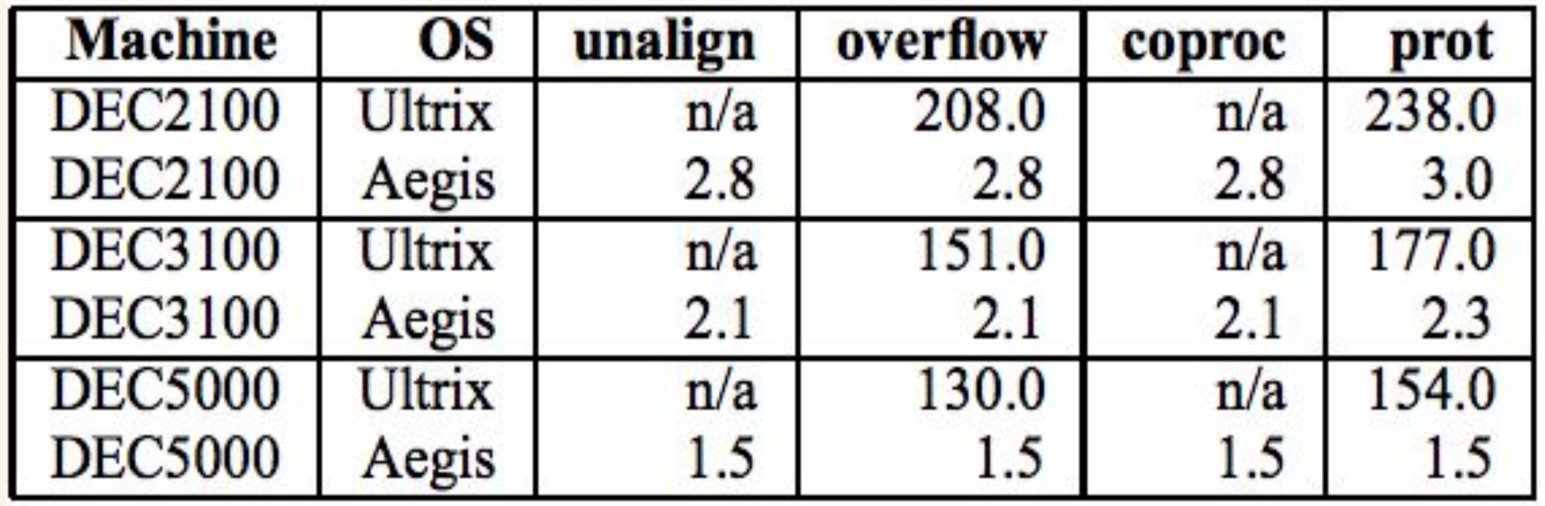
- What if the packet filter lies and claims a packet when it belongs to someone else?
  - Assume no one lies :D
- What would happen if we didn’t have the ASH?

Evaluation

- Run benchmarks multiple times to warm up cache.
- Take the best run of Ultrix. Take the median of 3 runs for exokernel.
- Are these fair benchmarks? Why or why not?



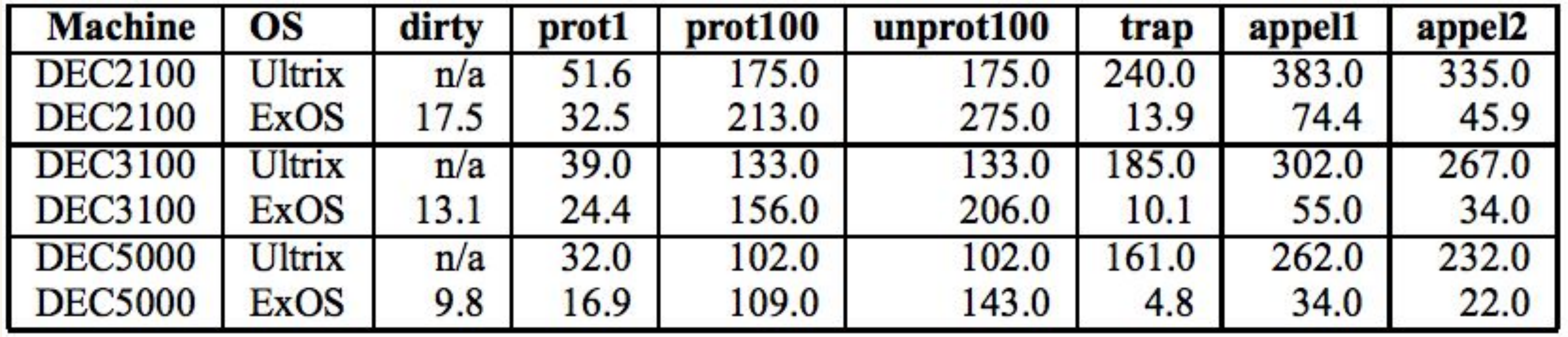
Machine	OS	Procedure call	Syscall (getpid)
DEC2100	Ultrix	0.57	32.2
DEC2100	Aegis	0.56	3.2 / 4.7
DEC3100	Ultrix	0.42	33.7
DEC3100	Aegis	0.42	2.9 / 3.5
DEC5000	Ultrix	0.28	21.3
DEC5000	Aegis	0.28	1.6 / 2.3



Machine	OS	unalign	overflow	coproc	prot
DEC2100	Ultrix	n/a	208.0	n/a	238.0
DEC2100	Aegis	2.8	2.8	2.8	3.0
DEC3100	Ultrix	n/a	151.0	n/a	177.0
DEC3100	Aegis	2.1	2.1	2.1	2.3
DEC5000	Ultrix	n/a	130.0	n/a	154.0
DEC5000	Aegis	1.5	1.5	1.5	1.5

Exception dispatch time (µs)

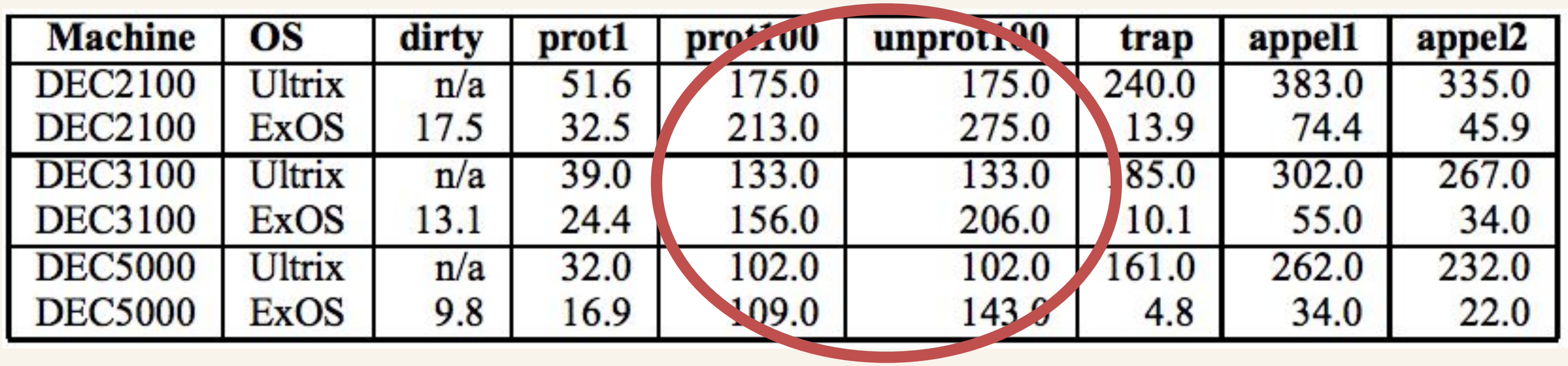
Evaluation



Machine	OS	dirty	prot1	prot100	unprot100	trap	appel1	appel2
DEC2100	Ultrix	n/a	51.6	175.0	175.0	240.0	383.0	335.0
DEC2100	ExOS	17.5	32.5	213.0	275.0	13.9	74.4	45.9
DEC3100	Ultrix	n/a	39.0	133.0	133.0	185.0	302.0	267.0
DEC3100	ExOS	13.1	24.4	156.0	206.0	10.1	55.0	34.0
DEC5000	Ultrix	n/a	32.0	102.0	102.0	161.0	262.0	232.0
DEC5000	ExOS	9.8	16.9	109.0	143.0	4.8	34.0	22.0

- Faster in ExOS because we are operating all in user space!
- Anything unexpected?

Evaluation

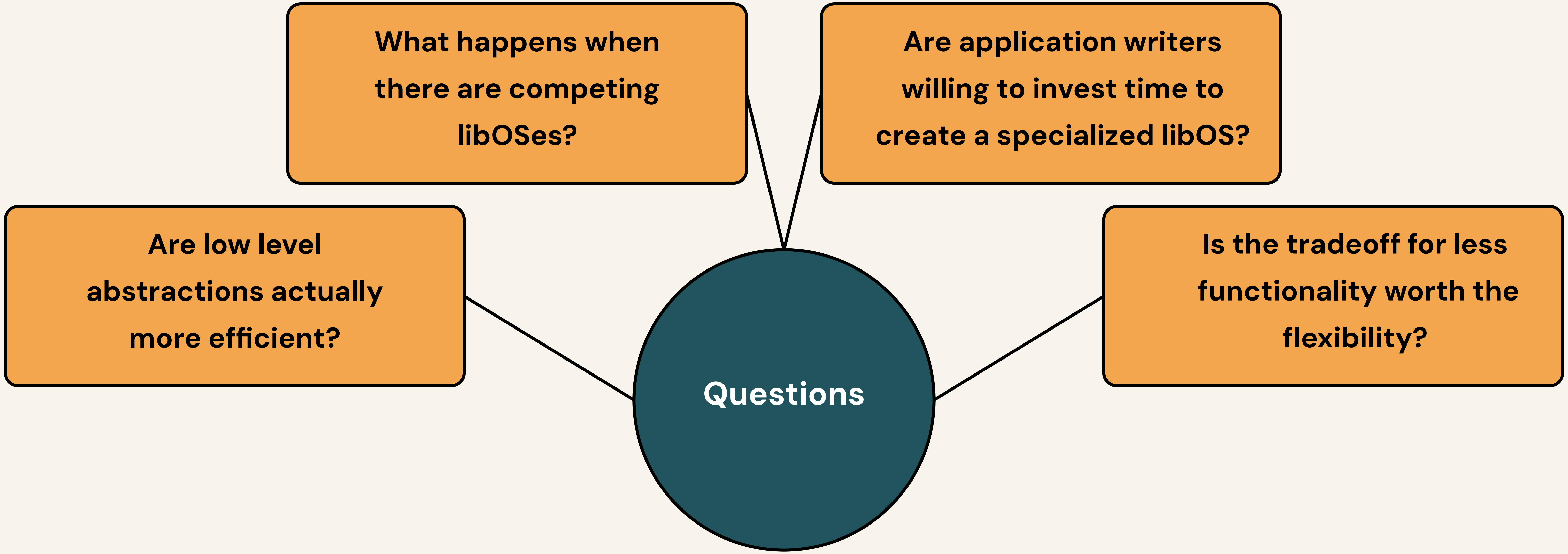


Machine	OS	dirty	prot1	prot100	unprot100	trap	appel1	appel2
DEC2100	Ultrix	n/a	51.6	175.0	175.0	240.0	383.0	335.0
DEC2100	ExOS	17.5	32.5	213.0	275.0	13.9	74.4	45.9
DEC3100	Ultrix	n/a	39.0	133.0	133.0	185.0	302.0	267.0
DEC3100	ExOS	13.1	24.4	156.0	206.0	10.1	55.0	34.0
DEC5000	Ultrix	n/a	32.0	102.0	102.0	161.0	262.0	232.0
DEC5000	ExOS	9.8	16.9	109.0	143.0	4.8	34.0	22.0

- Faster in ExOS because we are operating all in user space!
- Anything unexpected?
  - Why is prot100 and unprot100 so slow in comparison?

Where is the file system?

- It’s really hard to build a filesystem ☹
- Exokernel filesystem went through 4 redesigns
- How do we give all libOSes control of the filesystem when they all have to share it?
- What would you do?



Questions

What happens when there are competing libOSes?

Are application writers willing to invest time to create a specialized libOS?

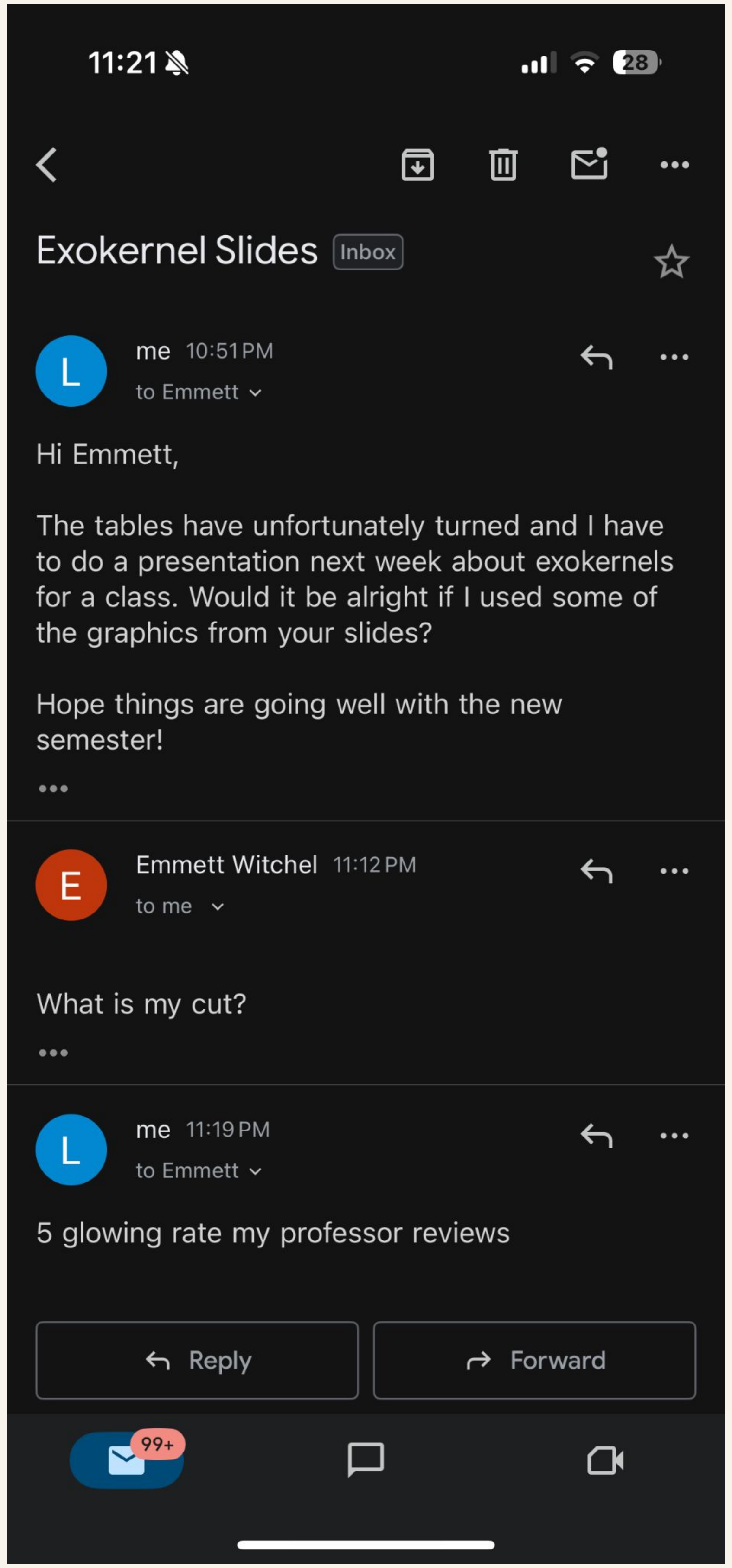
Are low level abstractions actually more efficient?

Is the tradeoff for less functionality worth the flexibility?

Summary

- Lower level abstractions in an OS can lead to better performance.
- Trade-off since we are losing functionality
- A more elegant idea than the monolithic kernel, but is it in actuality?

Diagram credits :-)



Exokernel Slides

me 10:51 PM
to Emmett

Hi Emmett,

The tables have unfortunately turned and I have to do a presentation next week about exokernels for a class. Would it be alright if I used some of the graphics from your slides?

Hope things are going well with the new semester!

Emmett Witchel 11:12 PM
to me

What is my cut?

me 11:19 PM
to Emmett

5 glowing rate my professor reviews