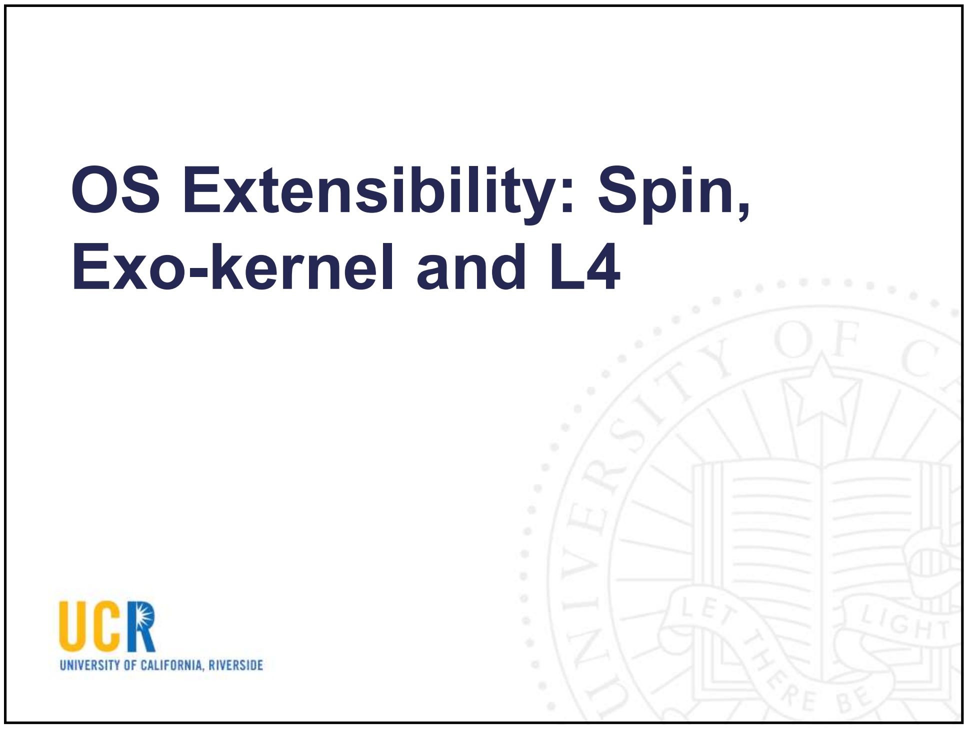
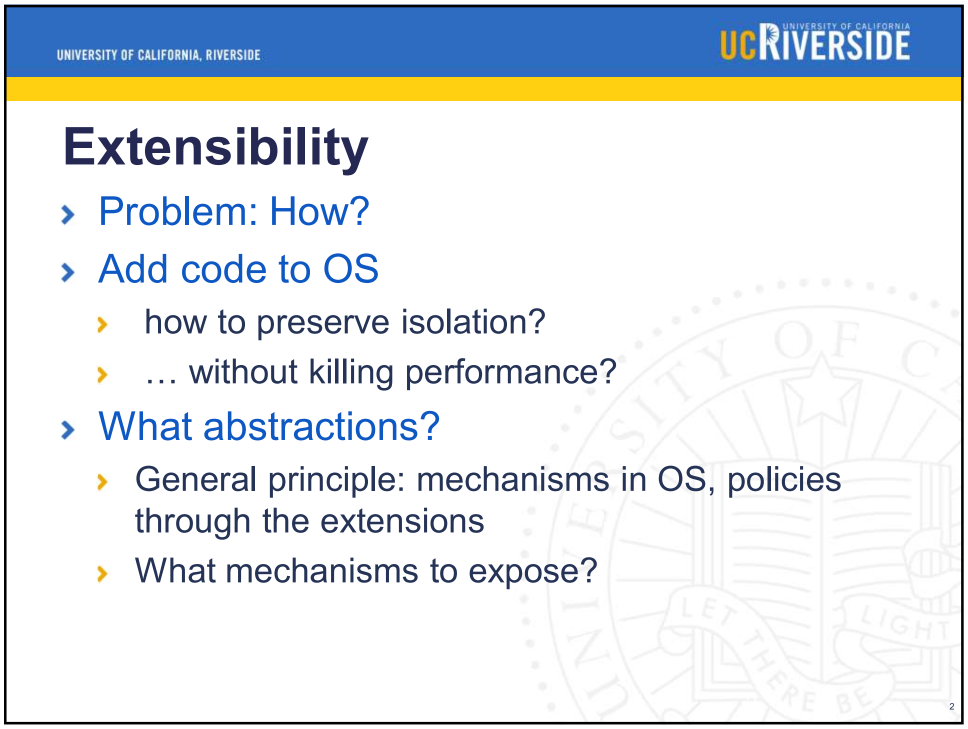


# OS Extensibility: Spin, Exo-kernel and L4

UNIVERSITY OF CALIFORNIA, RIVERSIDE



## Extensibility

- Problem: How?
- Add code to OS
  - how to preserve isolation?
  - … without killing performance?
- What abstractions?
  - General principle: mechanisms in OS, policies through the extensions
  - What mechanisms to expose?

UNIVERSITY OF CALIFORNIA, RIVERSIDE

UC RIVERSIDE

## Spin Approach to extensibility

- Co-location of kernel and extension
  - Avoid border crossings
  - But what about protection?
- Language/compiler forced protection
  - Strongly typed language
    - Protection by compiler and run-time
    - Cannot cheat using pointers
  - Logical protection domains
    - No longer rely on hardware address spaces to enforce protection – no boarder crossings
- Dynamic call binding for extensibility

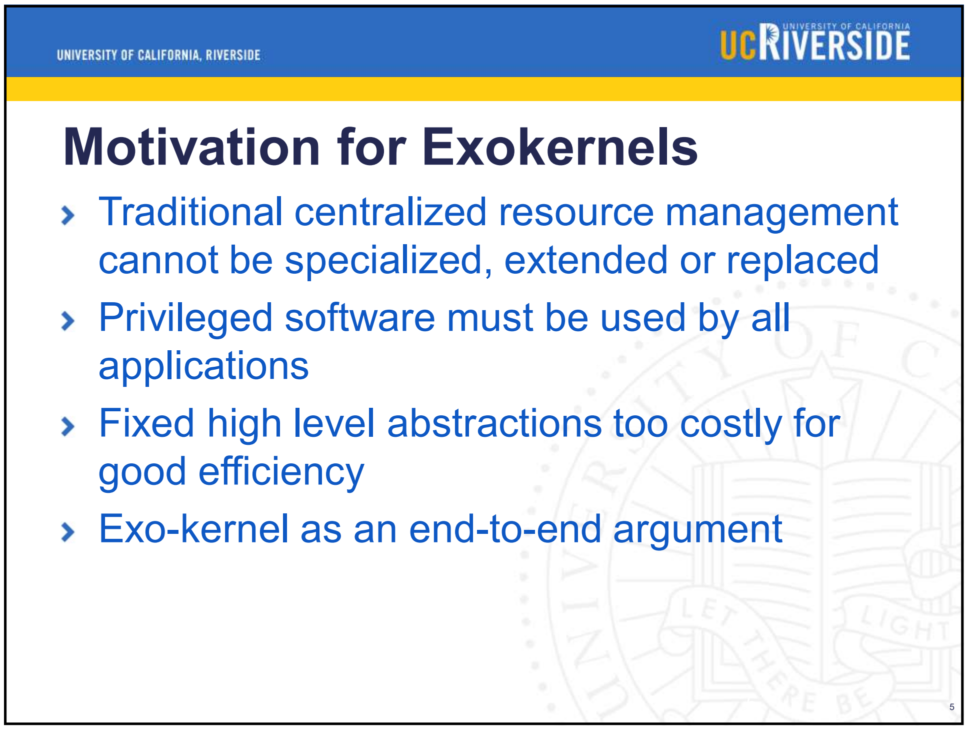
3

UNIVERSITY OF CALIFORNIA, RIVERSIDE

UC RIVERSIDE

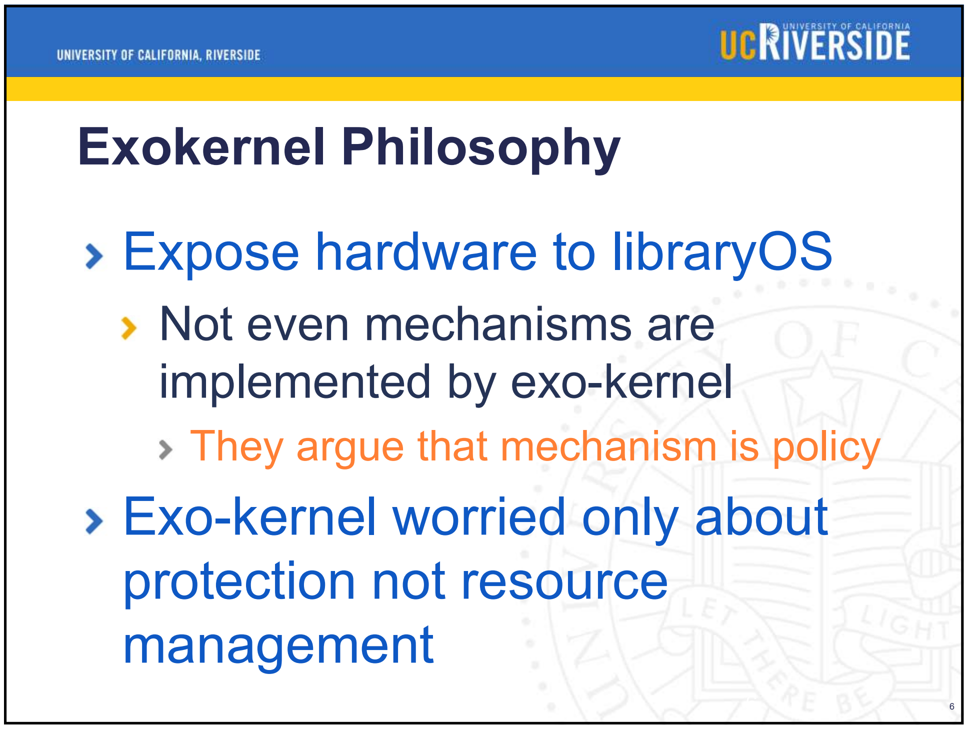
## ExoKernel

4



## Motivation for Exokernels

- Traditional centralized resource management cannot be specialized, extended or replaced
- Privileged software must be used by all applications
- Fixed high level abstractions too costly for good efficiency
- Exo-kernel as an end-to-end argument



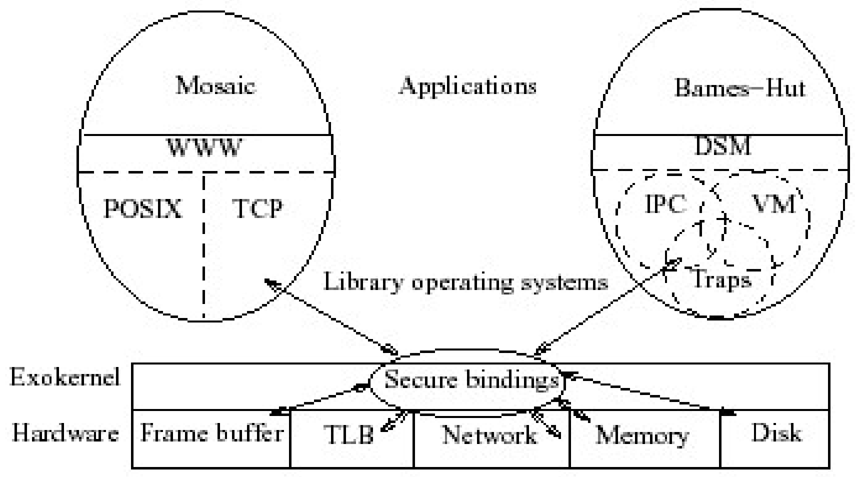
## Exokernel Philosophy

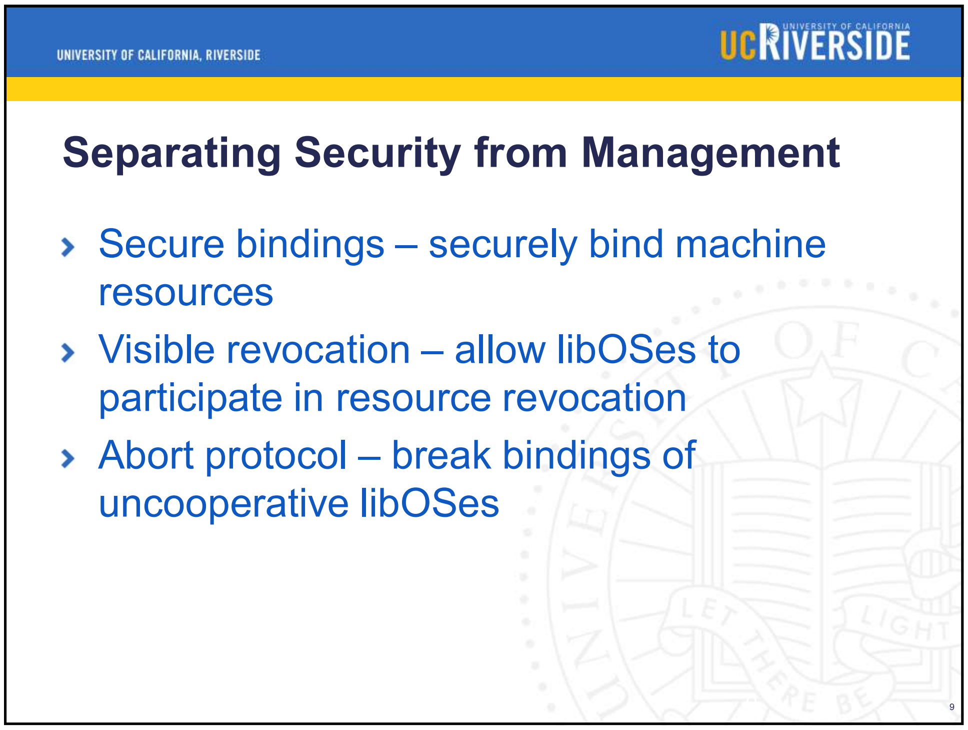
- Expose hardware to libraryOS
  - Not even mechanisms are implemented by exo-kernel
    - They argue that mechanism is policy
- Exo-kernel worried only about protection not resource management

## Design Principles

- Track resource ownership
- Ensure protection by guarding resource usage
- Revoke access to resources
- Expose hardware, allocation, names and revocation
- Basically validate binding, then let library manage the resource

## Exokernel Architecture



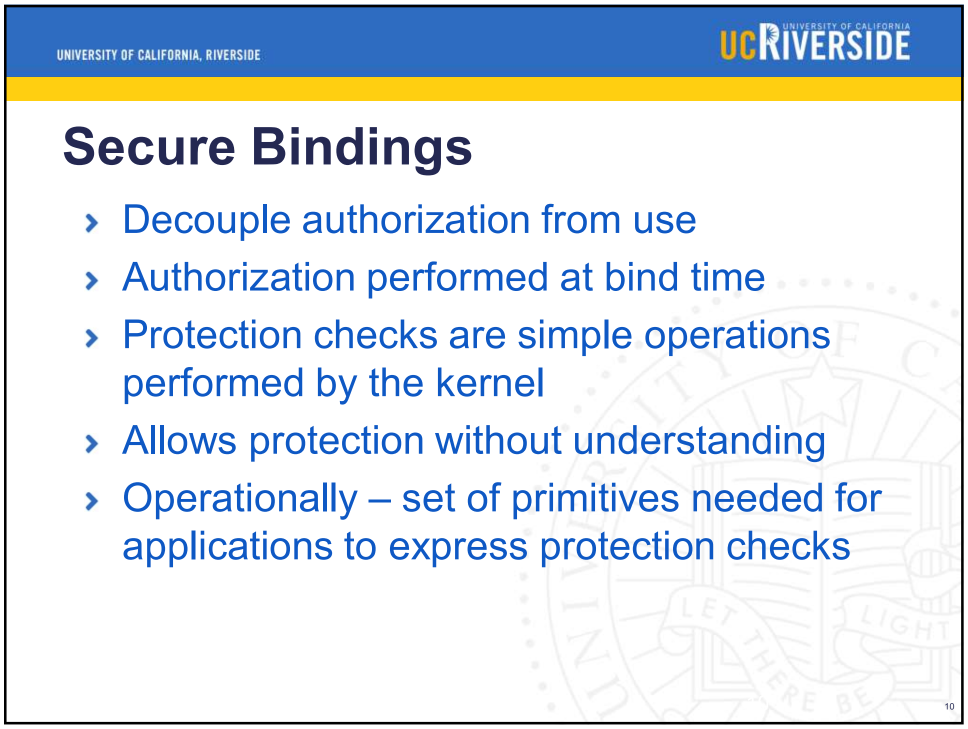

UNIVERSITY OF CALIFORNIA, RIVERSIDE

UC RIVERSIDE

## Separating Security from Management

- Secure bindings – securely bind machine resources
- Visible revocation – allow libOSes to participate in resource revocation
- Abort protocol – break bindings of uncooperative libOSes

9

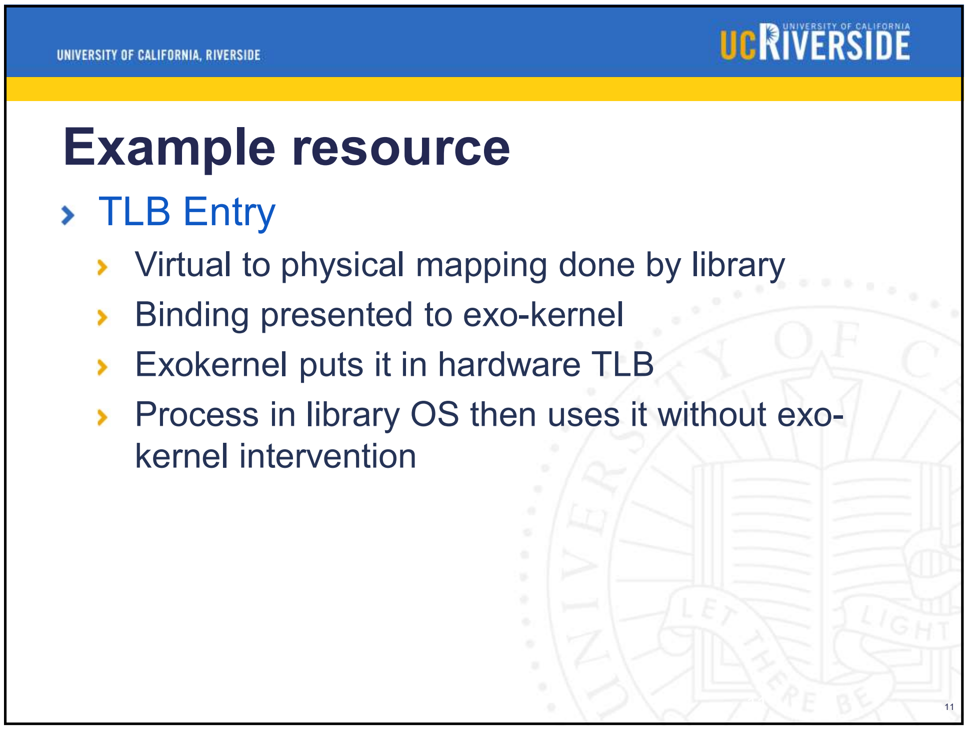

UNIVERSITY OF CALIFORNIA, RIVERSIDE

UC RIVERSIDE

## Secure Bindings

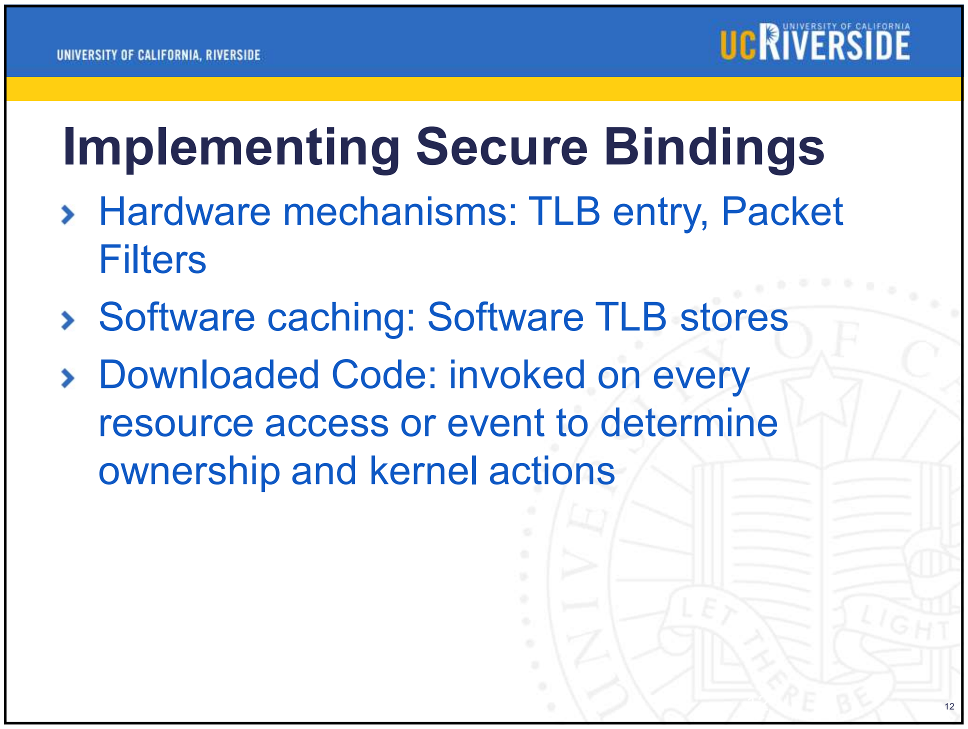
- Decouple authorization from use
- Authorization performed at bind time
- Protection checks are simple operations performed by the kernel
- Allows protection without understanding
- Operationally – set of primitives needed for applications to express protection checks

10



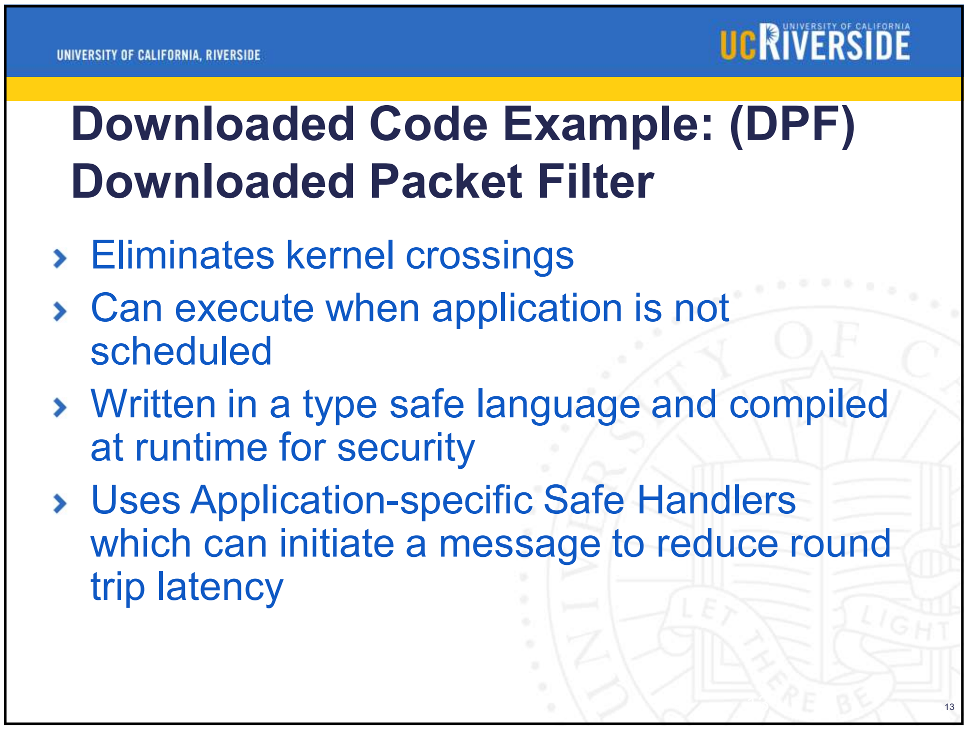
## Example resource

- TLB Entry
  - Virtual to physical mapping done by library
  - Binding presented to exo-kernel
  - Exokernel puts it in hardware TLB
  - Process in library OS then uses it without exo-kernel intervention



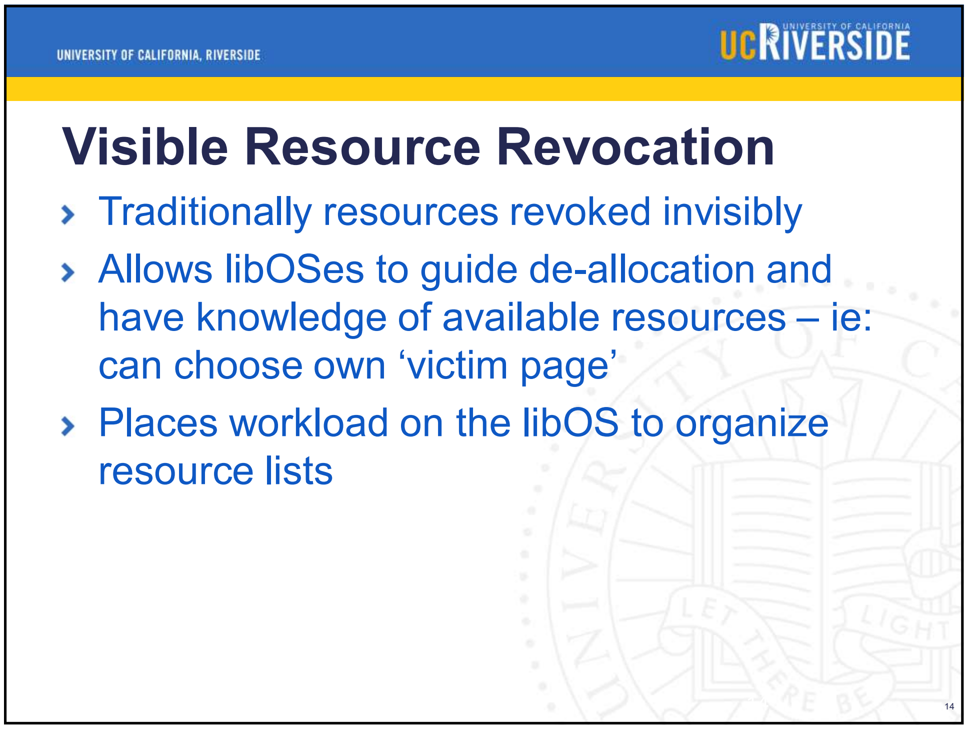
## Implementing Secure Bindings

- Hardware mechanisms: TLB entry, Packet Filters
- Software caching: Software TLB stores
- Downloaded Code: invoked on every resource access or event to determine ownership and kernel actions



## Downloaded Code Example: (DPF) Downloaded Packet Filter

- Eliminates kernel crossings
- Can execute when application is not scheduled
- Written in a type safe language and compiled at runtime for security
- Uses Application-specific Safe Handlers which can initiate a message to reduce round trip latency



## Visible Resource Revocation

- Traditionally resources revoked invisibly
- Allows libOSes to guide de-allocation and have knowledge of available resources – ie: can choose own ‘victim page’
- Places workload on the libOS to organize resource lists

UNIVERSITY OF CALIFORNIA, RIVERSIDE

## Abort Protocol

- Forced resource revocation
- Uses 'repossession vector'
- Raises a repossession exception
- Possible relocation depending on state of resource

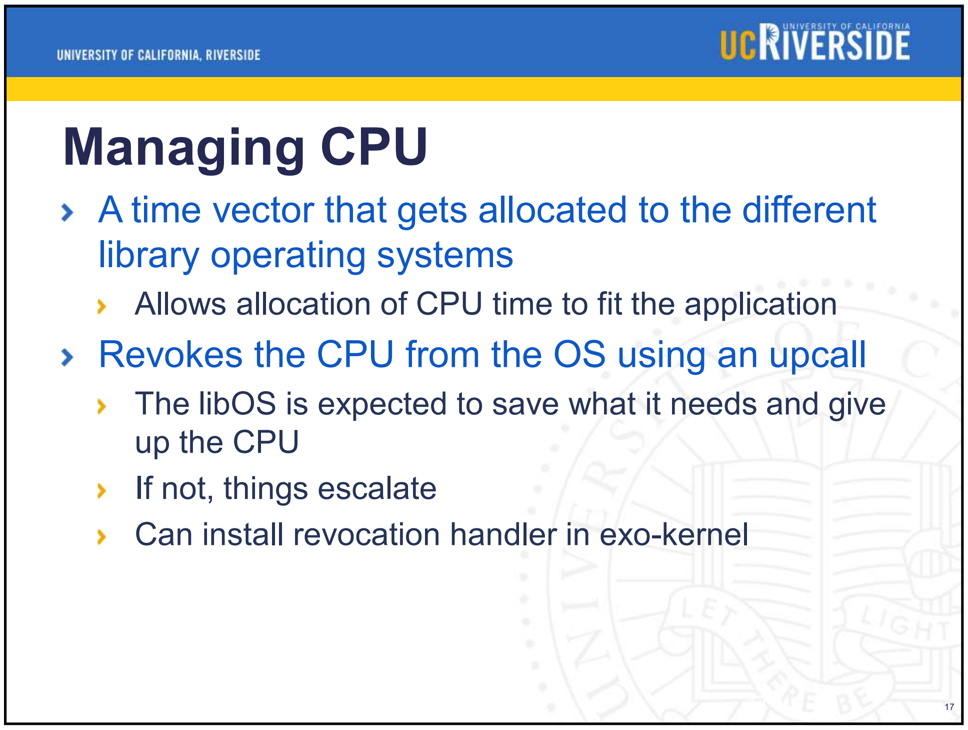
15

UNIVERSITY OF CALIFORNIA, RIVERSIDE

## Managing core services

- Virtual memory:
  - Page fault generates an upcall to the library OS via a registered handler
  - LibOS handles the allocation, then presents a mapping to be installed into the TLB providing a capability
  - Exo-kernel installs the mapping
  - Software TLBs

16



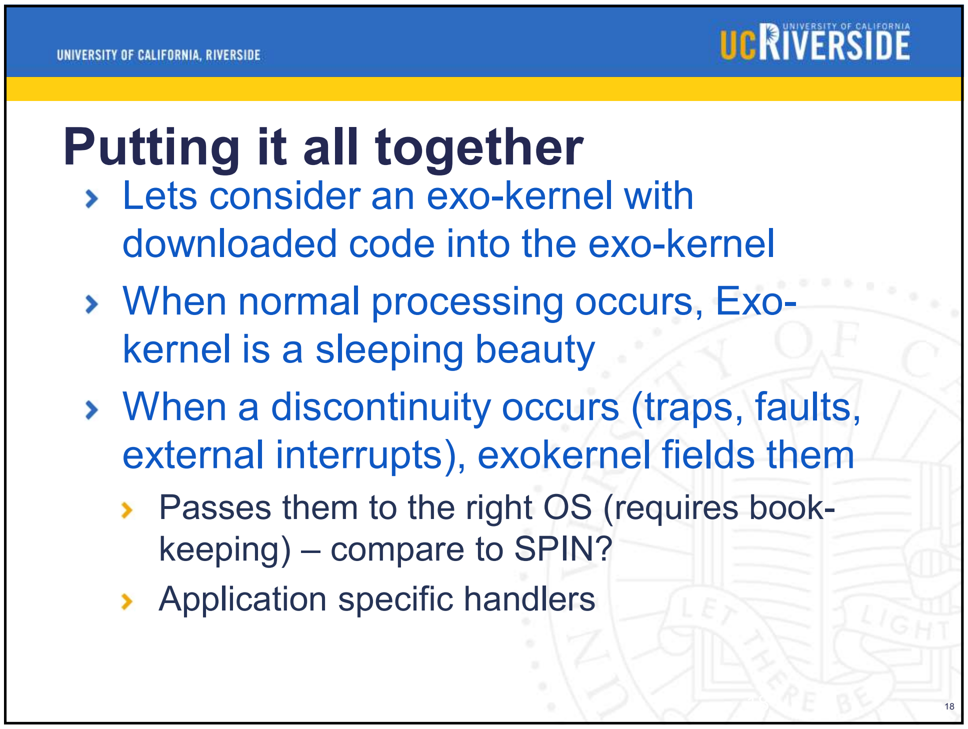
UNIVERSITY OF CALIFORNIA, RIVERSIDE

UC RIVERSIDE

## Managing CPU

- A time vector that gets allocated to the different library operating systems
  - Allows allocation of CPU time to fit the application
- Revokes the CPU from the OS using an upcall
  - The libOS is expected to save what it needs and give up the CPU
  - If not, things escalate
  - Can install revocation handler in exo-kernel

17



UNIVERSITY OF CALIFORNIA, RIVERSIDE

UC RIVERSIDE

## Putting it all together

- Lets consider an exo-kernel with downloaded code into the exo-kernel
- When normal processing occurs, Exo-kernel is a sleeping beauty
- When a discontinuity occurs (traps, faults, external interrupts), exokernel fields them
  - Passes them to the right OS (requires book-keeping) – compare to SPIN?
  - Application specific handlers

18

## Evaluation

- Again, a full implementation
- How to make sense from the quantitative results?
  - Absolute numbers are typically meaningless given that we are part of a bigger system
    - Trends are what matter
- Again, emphasis is on space and time
  - Key takeaway→ at least as good as a monolithic kernel

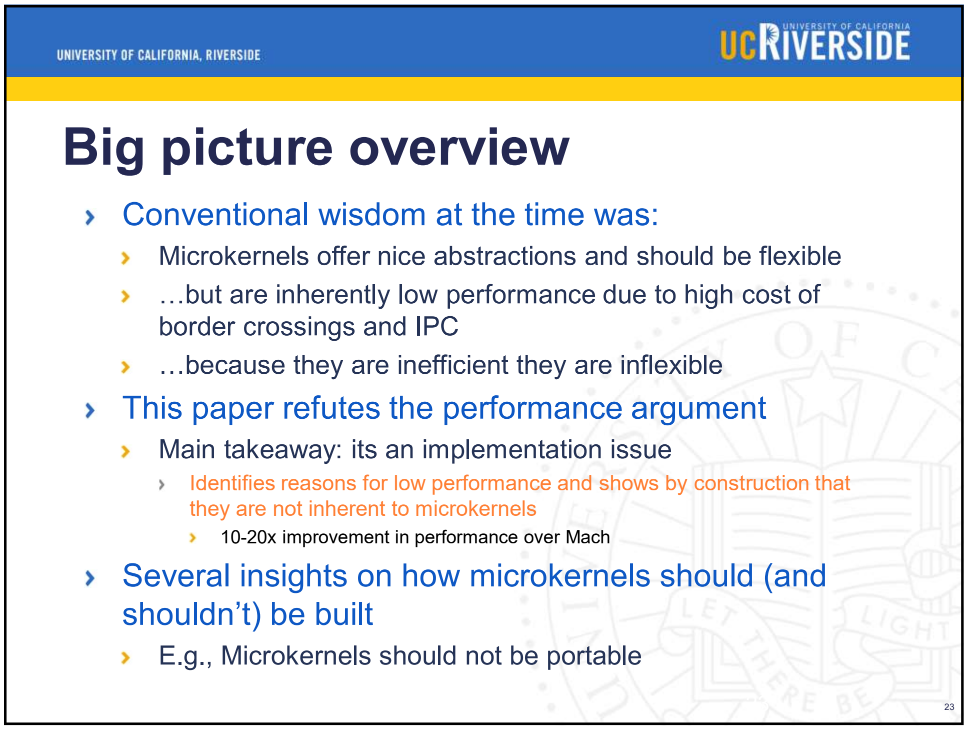
## Questions and conclusions

- Downloaded code – security?
  - Some mention of SFI and little languages
  - SPIN is better here?
- SPIN vs. Exokernel
  - Spin—extend mechanisms; some abstractions still exist
  - Exo-kernel: securely expose low-level primitives (primitive vs. mechanism?)
- Microkernel vs. exo-kernel
  - Much lower interfaces exported
  - Argue they lead to better performance
  - Of course, less border crossing due to downloadable code

## On Microkernel construction (L3/4)

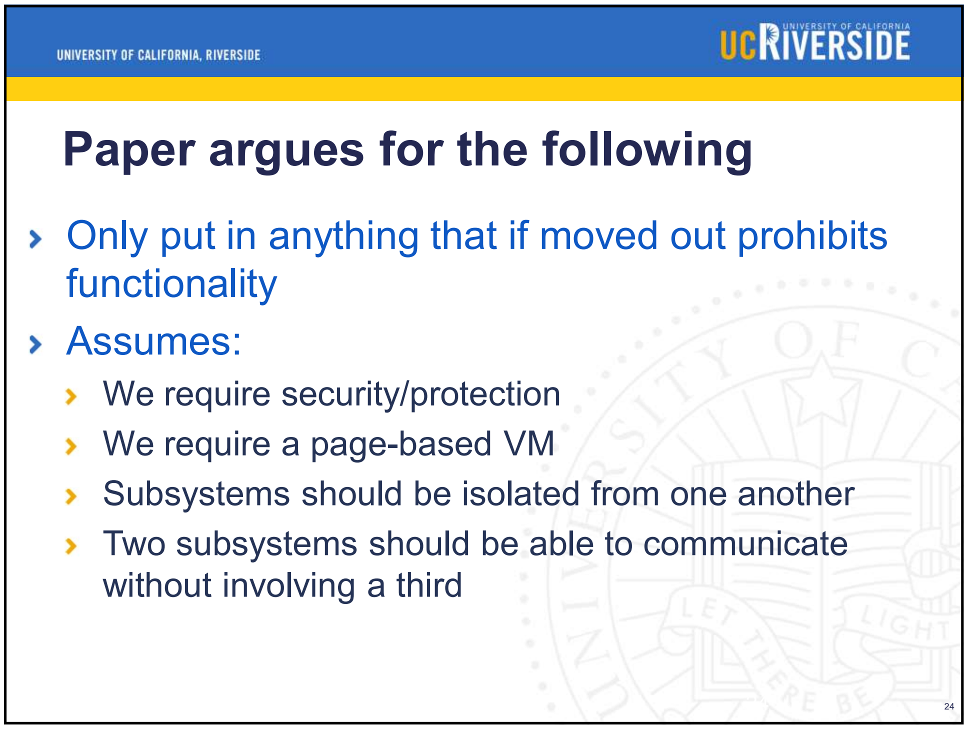
## L4 microkernel family

- Successful OS with different offshoot distributions
  - Commercially successful
    - OKLabs OKL4 shipped over 1.5 billion installations by 2012
      - Mostly qualcomm wireless modems
      - But also player in automative and airborne entertainment systems
    - Used in the secure enclave processor on Apple's A7 chips
      - All iOS devices have it! 100s of millions



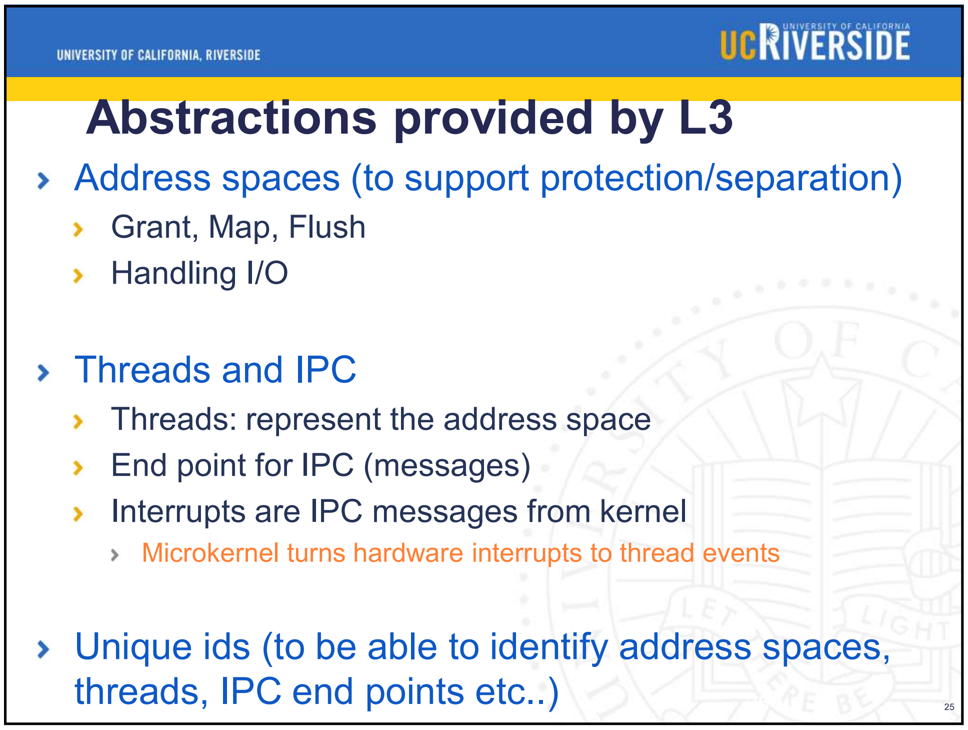
## Big picture overview

- Conventional wisdom at the time was:
  - Microkernels offer nice abstractions and should be flexible
  - …but are inherently low performance due to high cost of border crossings and IPC
  - …because they are inefficient they are inflexible
- This paper refutes the performance argument
  - Main takeaway: its an implementation issue
    - Identifies reasons for low performance and shows by construction that they are not inherent to microkernels
      - 10-20x improvement in performance over Mach
- Several insights on how microkernels should (and shouldn't) be built
  - E.g., Microkernels should not be portable



## Paper argues for the following

- Only put in anything that if moved out prohibits functionality
- Assumes:
  - We require security/protection
  - We require a page-based VM
  - Subsystems should be isolated from one another
  - Two subsystems should be able to communicate without involving a third

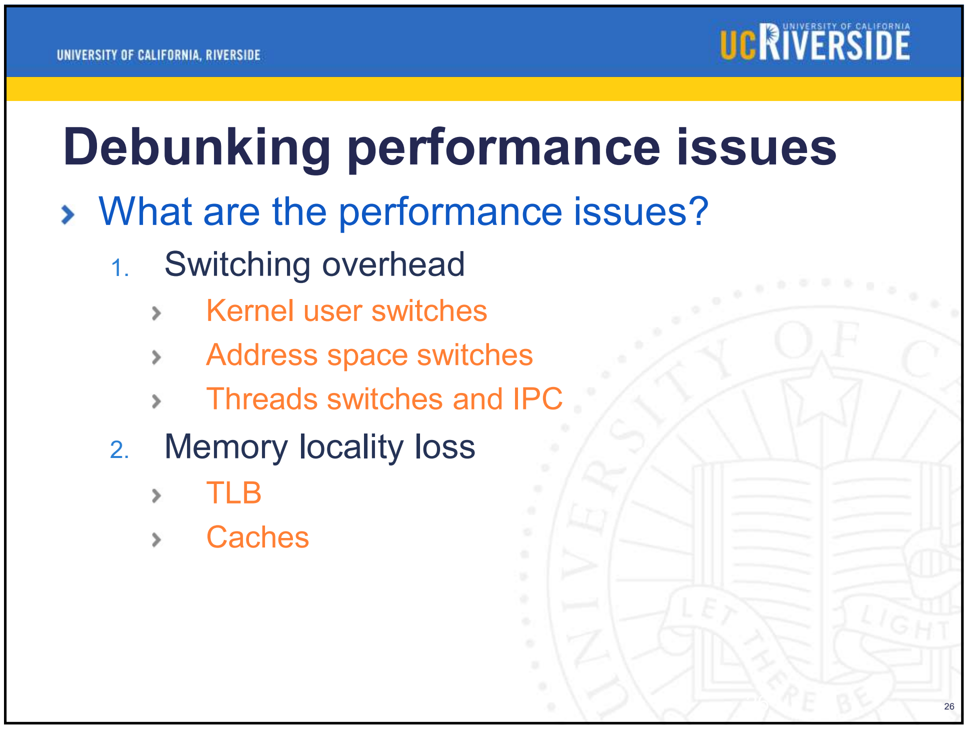


UNIVERSITY OF CALIFORNIA, RIVERSIDE

## Abstractions provided by L3

- Address spaces (to support protection/separation)
  - Grant, Map, Flush
  - Handling I/O
- Threads and IPC
  - Threads: represent the address space
  - End point for IPC (messages)
  - Interrupts are IPC messages from kernel
    - Microkernel turns hardware interrupts to thread events
- Unique ids (to be able to identify address spaces, threads, IPC end points etc..)

25

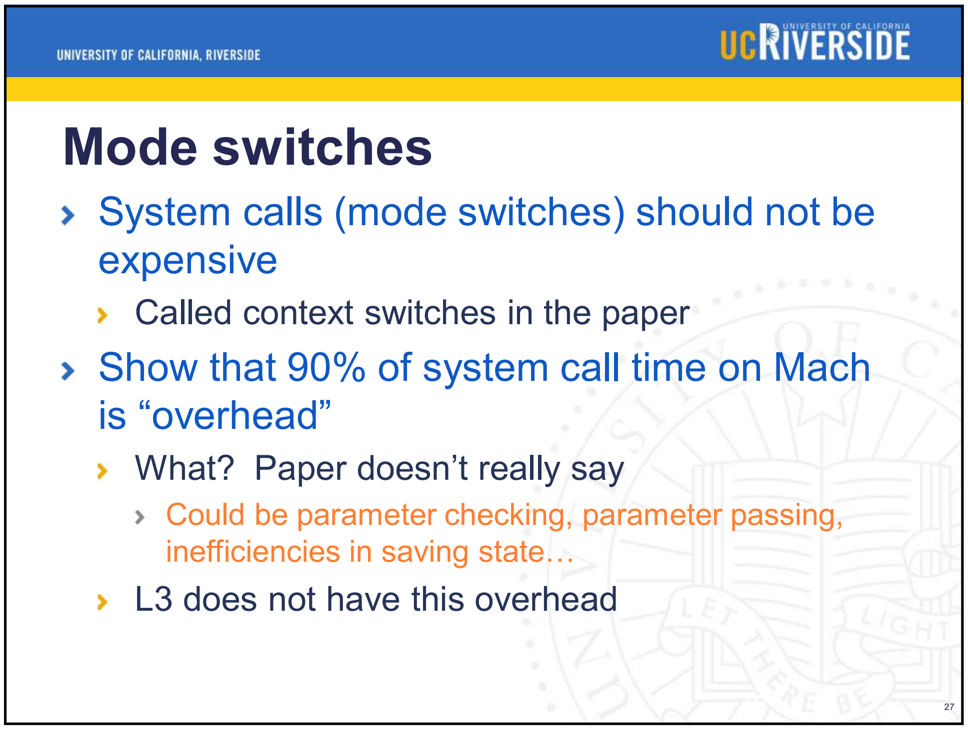


UNIVERSITY OF CALIFORNIA, RIVERSIDE

## Debunking performance issues

- What are the performance issues?
  1. Switching overhead
     - Kernel user switches
     - Address space switches
     - Threads switches and IPC
  2. Memory locality loss
     - TLB
     - Caches

26



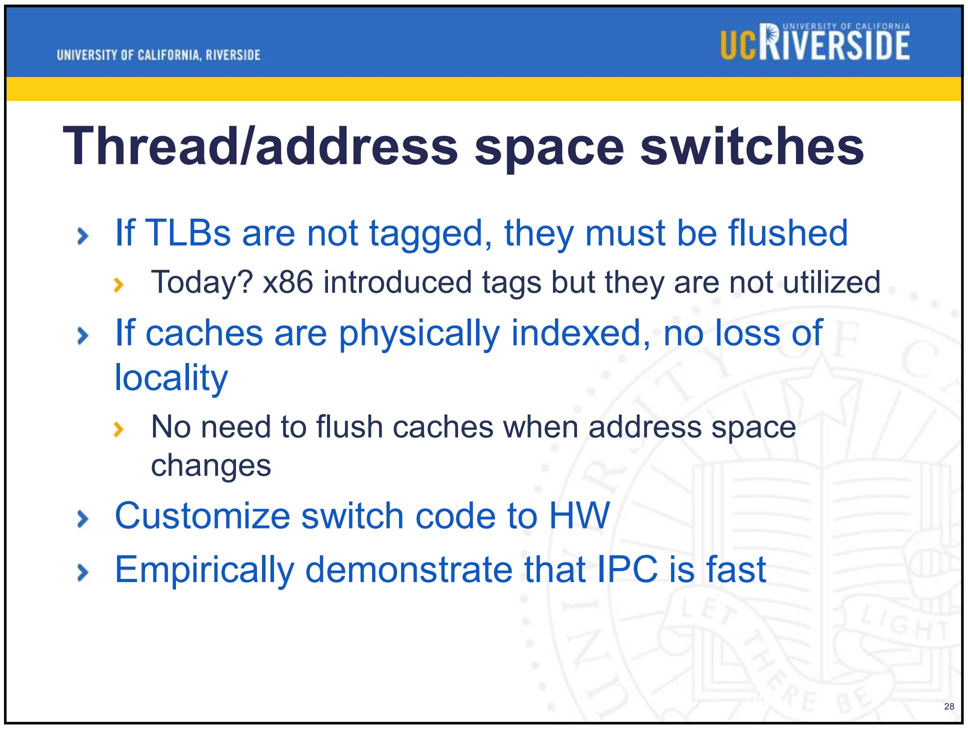
UNIVERSITY OF CALIFORNIA, RIVERSIDE

UC RIVERSIDE

## Mode switches

- System calls (mode switches) should not be expensive
  - Called context switches in the paper
- Show that 90% of system call time on Mach is “overhead”
  - What? Paper doesn’t really say
    - Could be parameter checking, parameter passing, inefficiencies in saving state…
  - L3 does not have this overhead

27



UNIVERSITY OF CALIFORNIA, RIVERSIDE

UC RIVERSIDE

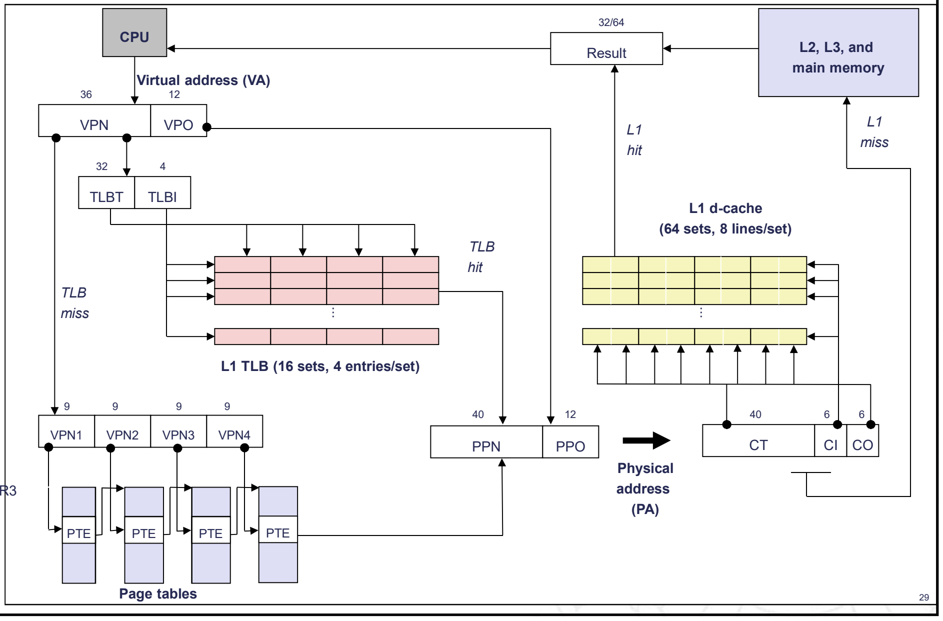
## Thread/address space switches

- If TLBs are not tagged, they must be flushed
  - Today? x86 introduced tags but they are not utilized
- If caches are physically indexed, no loss of locality
  - No need to flush caches when address space changes
- Customize switch code to HW
- Empirically demonstrate that IPC is fast

28

UNIVERSITY OF CALIFORNIA, RIVERSIDE

## Review: End-to-end Core i7 Address Translation



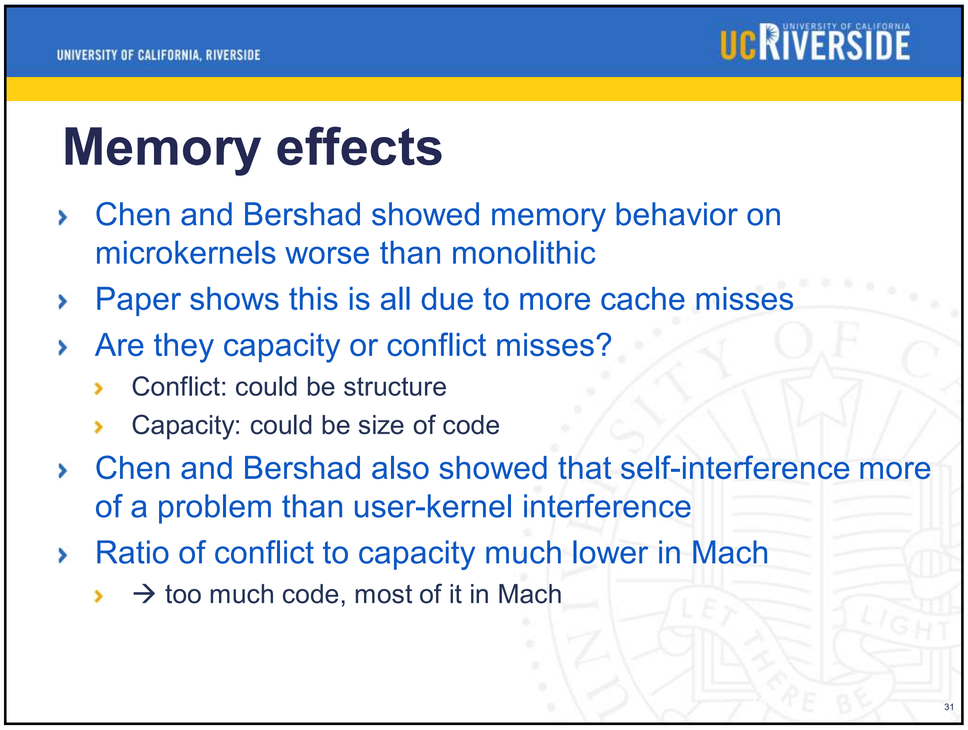
29

UNIVERSITY OF CALIFORNIA, RIVERSIDE

## Tricks to reduce the effect

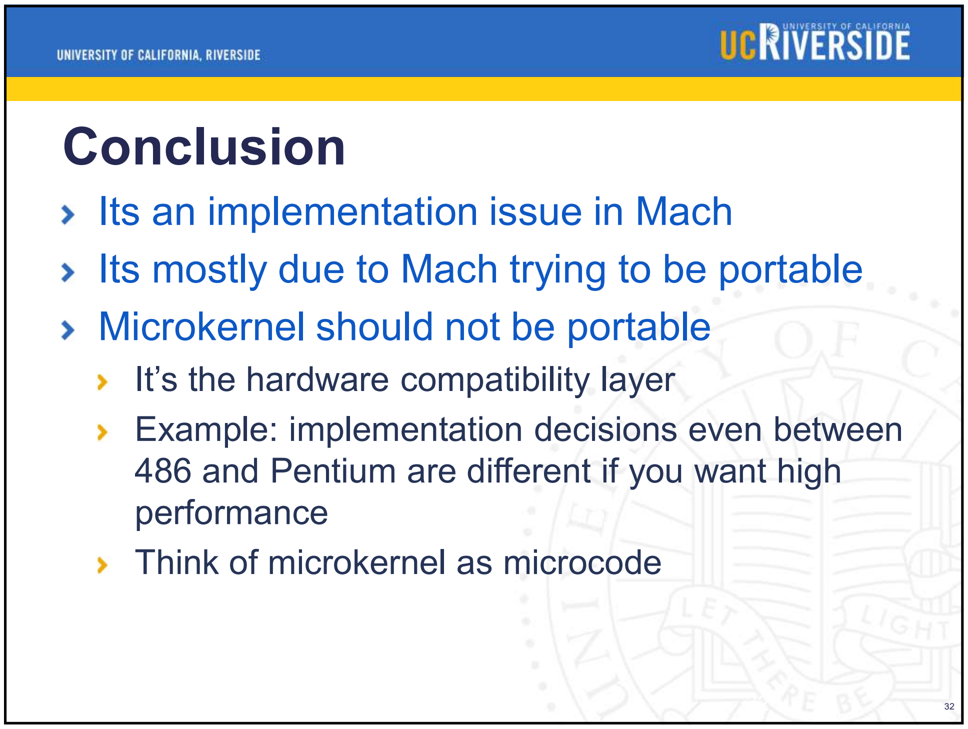
- TLB flushes due to AS switch could be very expensive
  - Since microkernel increases AS switches, this is a problem
  - Tagged TLB? If you have them
  - Tricks with segments to provide isolation between small address spaces
    - Remap them as segments within one address space
    - Avoid TLB flushes

30



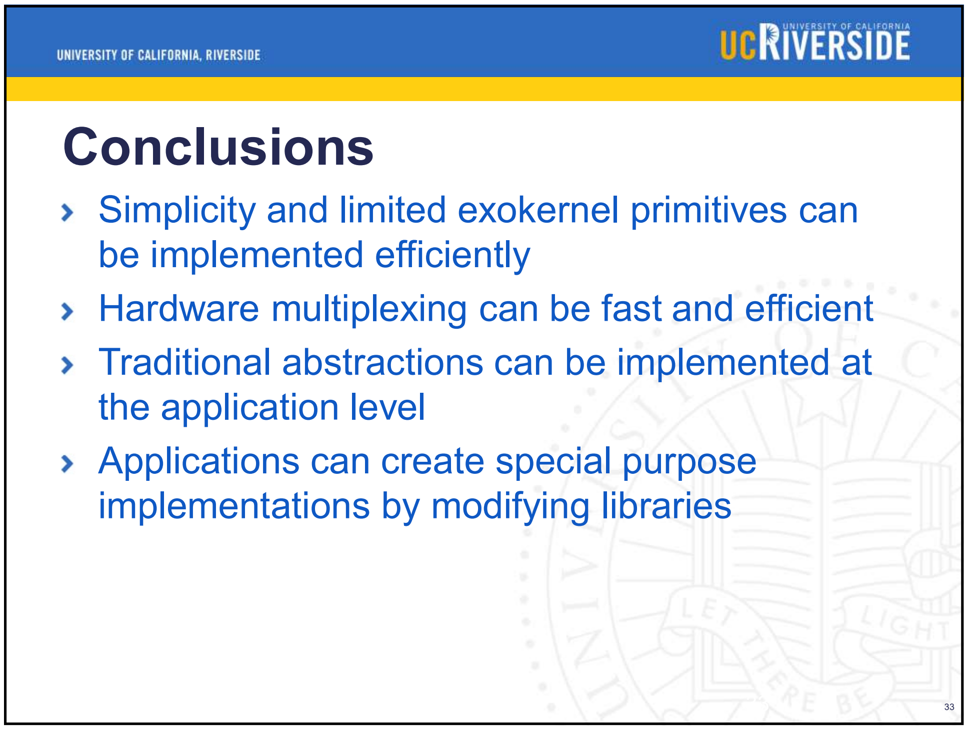
## Memory effects

- Chen and Bershad showed memory behavior on microkernels worse than monolithic
- Paper shows this is all due to more cache misses
- Are they capacity or conflict misses?
  - Conflict: could be structure
  - Capacity: could be size of code
- Chen and Bershad also showed that self-interference more of a problem than user-kernel interference
- Ratio of conflict to capacity much lower in Mach
  - → too much code, most of it in Mach



## Conclusion

- Its an implementation issue in Mach
- Its mostly due to Mach trying to be portable
- Microkernel should not be portable
  - It’s the hardware compatibility layer
  - Example: implementation decisions even between 486 and Pentium are different if you want high performance
  - Think of microkernel as microcode



## Conclusions

- Simplicity and limited exokernel primitives can be implemented efficiently
- Hardware multiplexing can be fast and efficient
- Traditional abstractions can be implemented at the application level
- Applications can create special purpose implementations by modifying libraries