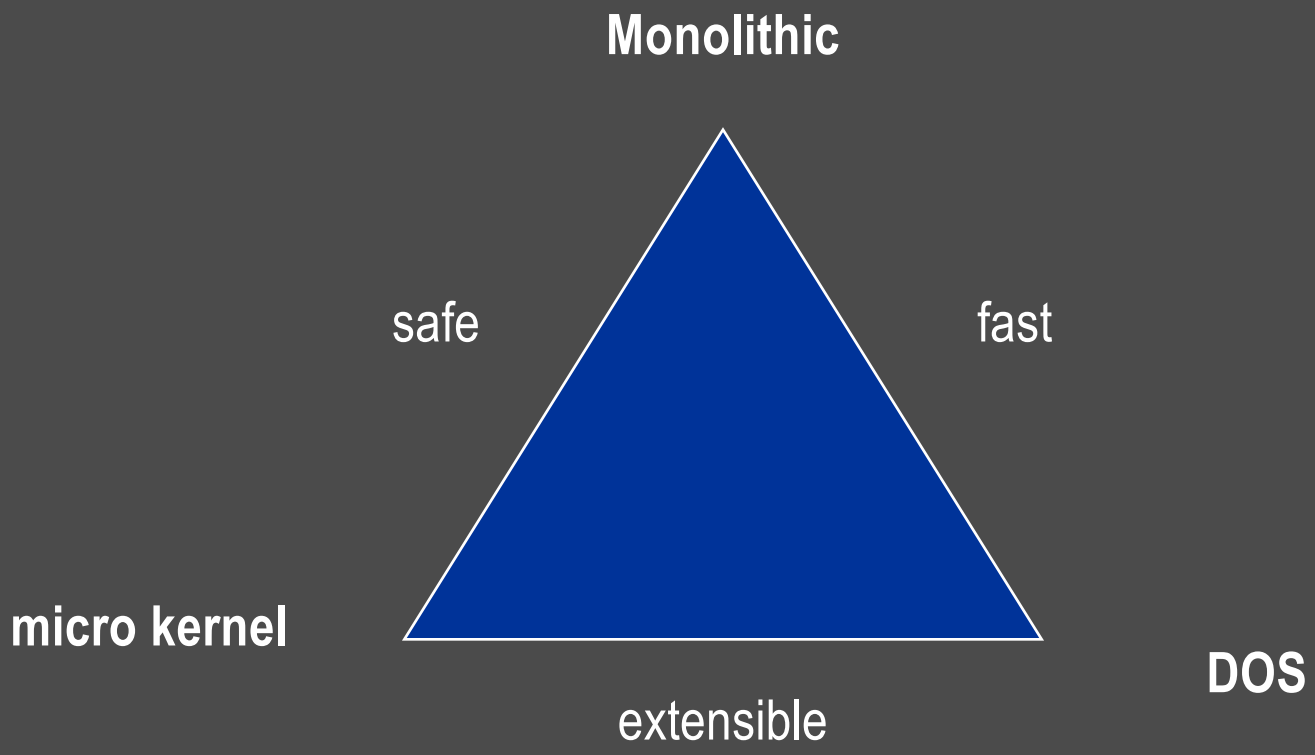
OS Extensibility: Spin, Exo-kernel and L4

Some slides from Hakim Weatherspoon

More simply



Need for Extensibility

- Buffer Pool Management In DBs (*)
  - LRU, prefetch (locality Vs suggestion), flush (commit)
- Shared Virtual Memory (+)
  - use a page fault to retrieve page from disk / another processor

Examples (cont.)

- Concurrent Checkpointing (+)
  - Overlap checkpointing and program being checkpointed
  - Change rights to R-only on dirty pages
  - Copy each page and reset rights
  - Allow reads; Use write faults to {copy, reset rights, restart}

* OS Support for Database Management (Stonebraker)

+ Virtual Memory Primitives for User Programs (Andrew W. Appel and Kai Li)

Examples (cont.)

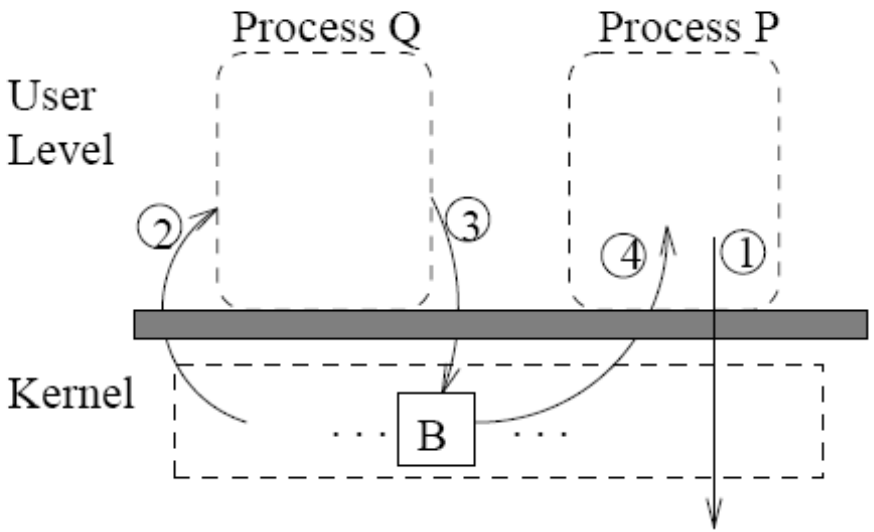


Figure 1: Interaction between kernel and user processes in two-level replacement: (1) P misses; (2) kernel consults Q for replacement; (3) Q decides to give up page B; (4) kernel reallocates B to P.

Feedback for file cache block replacement

[Implementation and Performance of Application-Controlled File Caching – Pei Cao, et al.]

Extensible Kernels

- SPIN (SOSP 1995): kernel extensions (imported) safely specialize OS services
  - Extensions dynamically linked into OS kernel
  - Safety ensured by Programming Language facilities

- Exokernel (SOSP 1995): safely exports machine resources
  - Kernel only multiplexes hardware resources (Aegis)
  - Higher-level abstractions in Library OS (ExOS)
  - Secure binding, Visible resource revocation, Abort
  - Apps link with the LibOS of their choice

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

SPIN MECHANISMS/TOOLBOX

Logical protection domains

- Modula-3 safety and encapsulation mechanisms
  - Type safety, automatic storage management
  - Objects, threads, exceptions and generic interfaces
- Fine-grained protection of objects using capabilities. An object can be:
  - Hardware resources (e.g., page frames)
  - Interfaces (e.g., page allocation module)
  - Collection of interfaces (e.g., full VM)
- Capabilities are language supported pointers

Logical protection domains -- mechanisms

- Create:
  - Initialize with
- Resolve:
  - Names are res
    - Once resolve
- Combine
  - To create an aggregate domain
- This is the key to spin – protection, extensibility and performance

```
INTERFACE Domain;

TYPE T <: REFANY;   (* Domain.T is opaque *)

PROCEDURE Create(coff:CoffFile.T):T;
(* Returns a domain created from the specified object
   file (``coff'' is a standard object file format). *)

PROCEDURE CreateFromModule():T;
(* Create a domain containing interfaces defined by the
   calling module.  This function allows modules to
   name and export themselves at runtime. *)

PROCEDURE Resolve(source,target: T);
(* Resolve any undefined symbols in the target domain
   against any exported symbols from the source.*)

PROCEDURE Combine(d1, d2: T):T;
(* Create a new aggregate domain that exports the
   interfaces of the given domains. *)

END Domain.
```

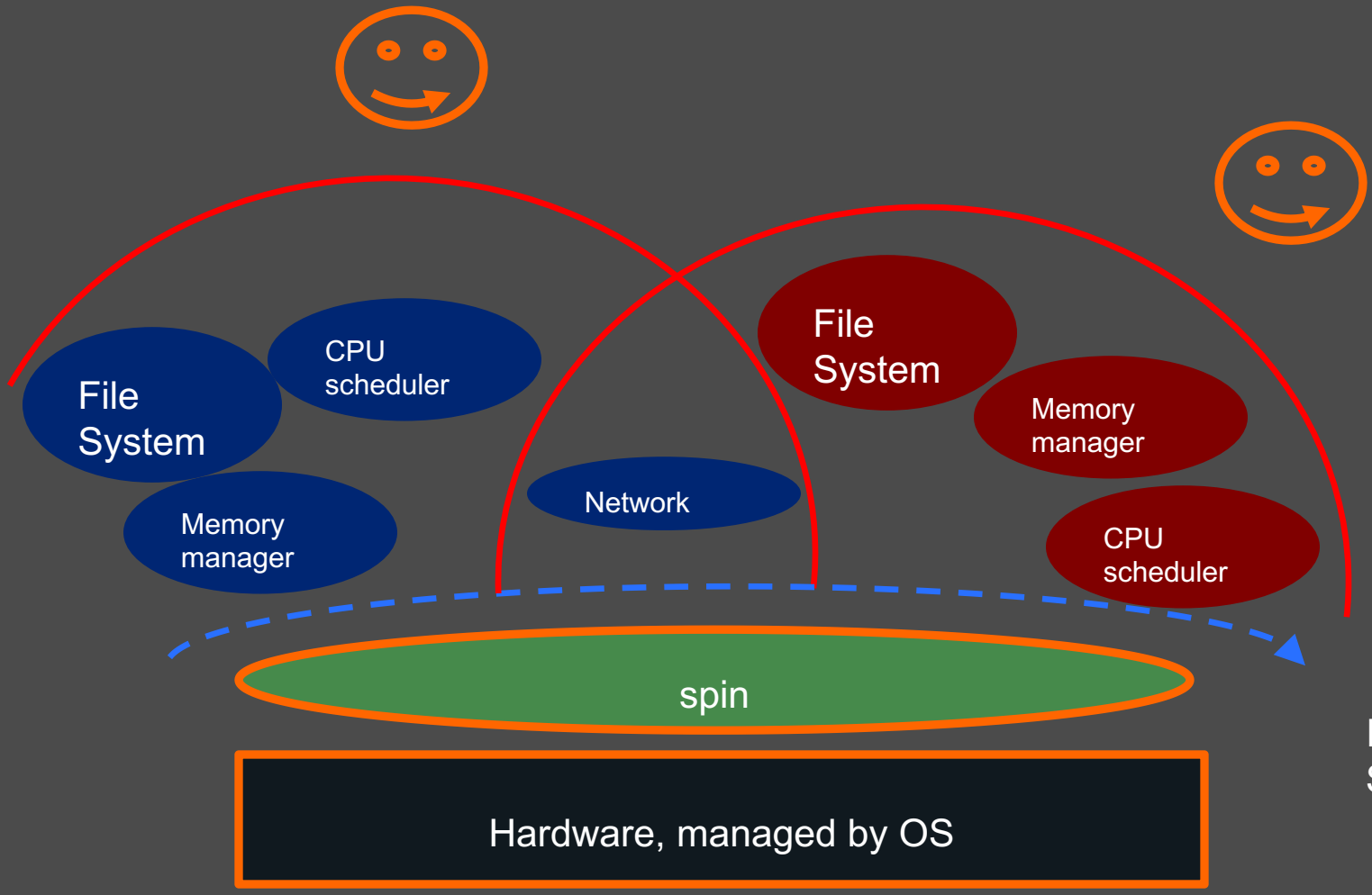
Protection Model (I)

- All kernel resources are referenced by *capabilities* [tickets]
- SPIN implements capabilities directly through the use of pointers
- Compiler prevents pointers to be forged or dereferenced in a way inconsistent with its type at *compile time*:
  - No run time overhead for using a pointer

Protection Model (II)

- A pointer can be passed to a user-level application through an *externalized reference:*
  - Index into a per-application table of safe references to kernel data structures
- Protection domains define the set of names accessible to a given execution context

Spin

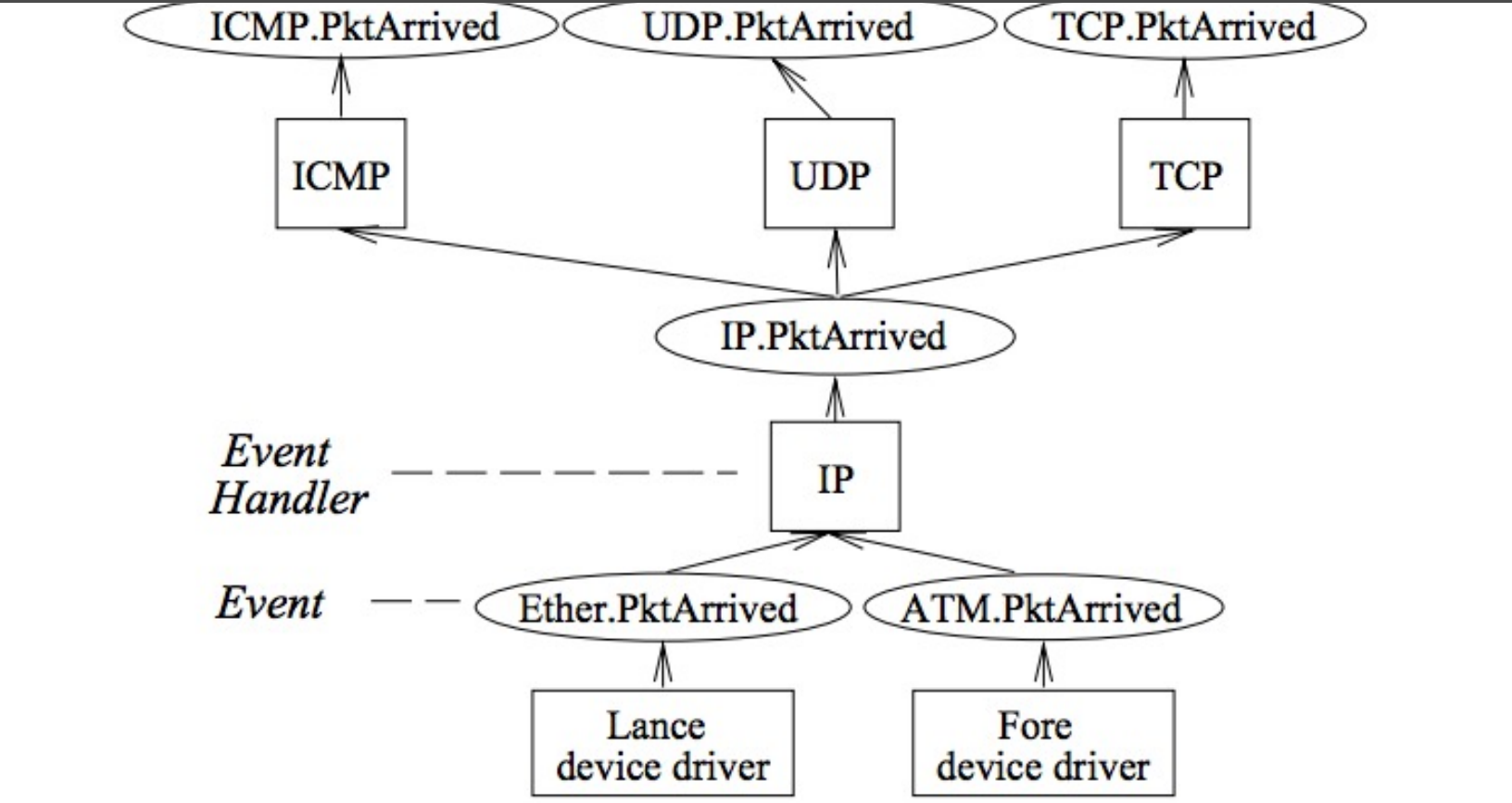


IPC, Address Spaces, …

Spin Mechanisms for Events

- Spin extension model is based on events and handlers
  - Which provide for communication between the base and the extensions
- Events are routed by the Spin Dispatcher to handlers
  - Handlers are typically extension code called as a procedure by the dispatcher
  - One-to-one, one-to-many or many-to-one
    - All handlers registered to an event are invoked
      - Guards may be used to control which handler is used

Event example



PUTTING IT ALL TOGETHER

Default Core services in SPIN

```
INTERFACE PhysAddr;

TYPE T <: REFANY;  (* PhysAddr.T is opaque *)

PROCEDURE Allocate(size: Size; attrib: Attrib): T;
(* Allocate some physical memory with
   particular attributes. *)

PROCEDURE Deallocate(p: T);

PROCEDURE Reclaim(candidate: T): T;
(* Request to reclaim a candidate page.
   Clients may handle this event to
   nominate alternative candidates. *)

END PhysAddr.
```

```
INTERFACE VirtAddr;

TYPE T <: REFANY;  (* VirtAddr.T is opaque *)

PROCEDURE Allocate(size: Size; attrib: Attrib): T;
PROCEDURE Deallocate(v: T);
END VirtAddr.
```

```
INTERFACE Translation;
IMPORT PhysAddr, VirtAddr;

TYPE T <: REFANY;   (* Translation.T is opaque *)

PROCEDURE Create(): T;
PROCEDURE Destroy(context: T);
(* Create or destroy an addressing context *)

PROCEDURE AddMapping(context: T; v: VirtAddr.T;
          p: PhysAddr.T; prot: Protection);
(* Add [v,p] into the named translation context
   with the specified protection. *)

PROCEDURE RemoveMapping(context: T; v: VirtAddr.T);

PROCEDURE ExamineMapping(context: T;
          v: VirtAddr.T): Protection;

(* A few events raised during *)
(* illegal translations *)
PROCEDURE PageNotPresent(v: T);
PROCEDURE BadAddress(v: T);
PROCEDURE ProtectionFault(v: T);

END Translation.
```

Figure 3: *The interfaces for managing physical addresses, virtual addresses, and translations.*

- Event handlers
  - Page fault, access fault, bad address

- Sp
  - S
- Eve
  - E me
- Sp
  - Interacts with extension threads package

```
INTERFACE Strand;

TYPE T <: REFANY;   (* Strand.T is opaque *)

PROCEDURE Block(s:T);
(* Signal to a scheduler that s is not runnable. *)

PROCEDURE Unblock(s: T);
(* Signal to a scheduler that s is runnable. *)

PROCEDURE Checkpoint(s: T);
(* Signal that s is being descheduled and that it
   should save any processor state required for
   subsequent rescheduling. *)

PROCEDURE Resume(s: T);
(* Signal that s is being placed on a processor and
   that it should reestablish any state saved during
   a prior call to Checkpoint. *)

END Strand.
```

Figure 4: *The Strand Interface. This interface describes the scheduling events affecting control flow that can be raised within the kernel. Application-specific schedulers and thread packages install handlers on these events, which are raised on behalf of particular strands. A trusted thread package and scheduler provide default implementations of these operations, and ensure that extensions do not install handlers on strands for which they do not possess a capability.*

Experiments

- Don't worry, I wont go through them
- In the OS community, you have to demonstrate what you are proposing
  - They built SPIN, extensions and applications that use them
  - Focus on performance and size
    - Reasonable size, and substantial performance advantages even relative to a mature monolithic kernel

Conclusions

- Extensibility, protection and performance
  - compiler features and run-time checks
  - Instead of hardware address spaces
  - …which gives us performance—no border crossing
- Who are we trusting? Consider application and Spin
  - How does this compare to Exo-kernel?
- Concern about resource partitioning?
  - Each extension must be given its resources
  - No longer dynamically shared (easily)
  - Parallels to Virtualization?

EXOKERNEL

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

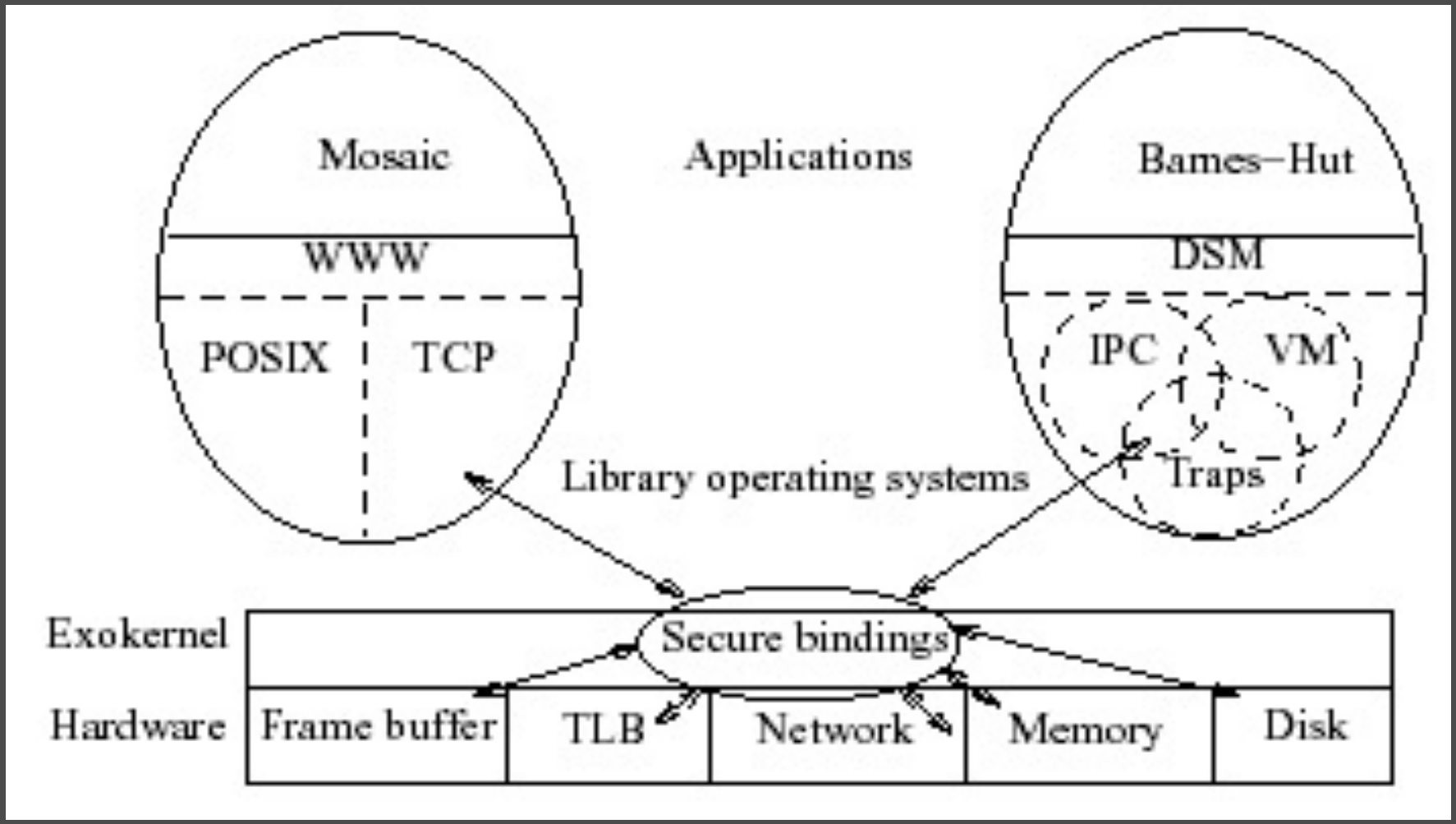
Exokernel Philosophy

- 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



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

How have such designs influenced current OS?

- Kernel modules
- Virtualization
- Containers
- Specialized OS

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

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..)

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

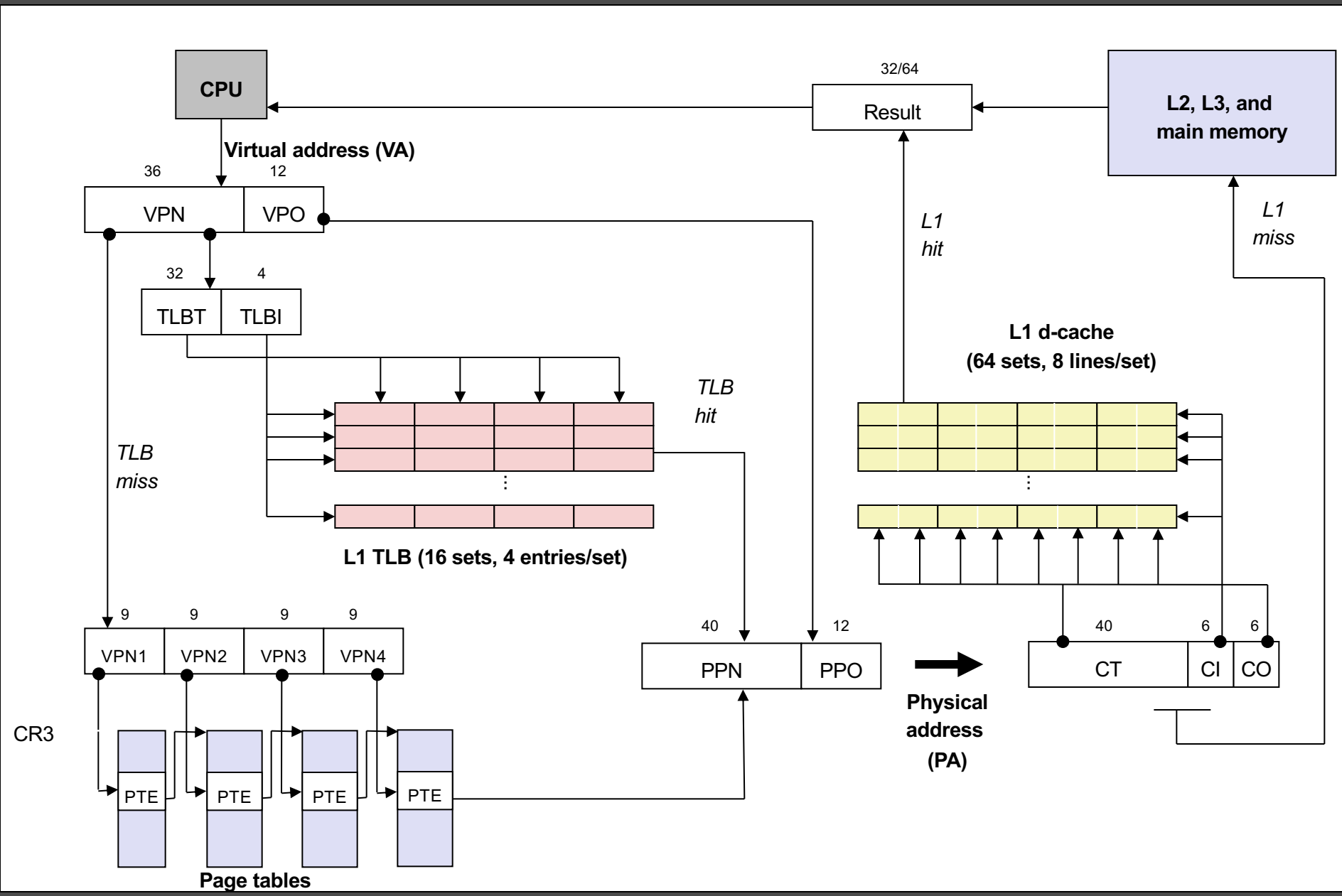
Mode switches

- 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

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

Review: End-to-end Core i7 Address Translation



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

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