CS 202: Advanced Operating Systems

Extensible Operating Systems

Extensibility

- What do we mean by extensibility?
  - Flexible to add new features/functionalities
  - Good efficiency
  - Good security

- Can you give a few examples?
  - Device drivers
  - Browser plugins/extensions

Existing Approaches

- Directly insert code modules
  - E.g., Loadable kernel module
  - Good efficiency
  - Bad security
- Put into a new process
  - E.g., User-mode driver (e.g., FUSE)
  - E.g., Microsoft puts browser plugin into a new process
  - Good security
  - Bad efficiency (context switch/mode switch)

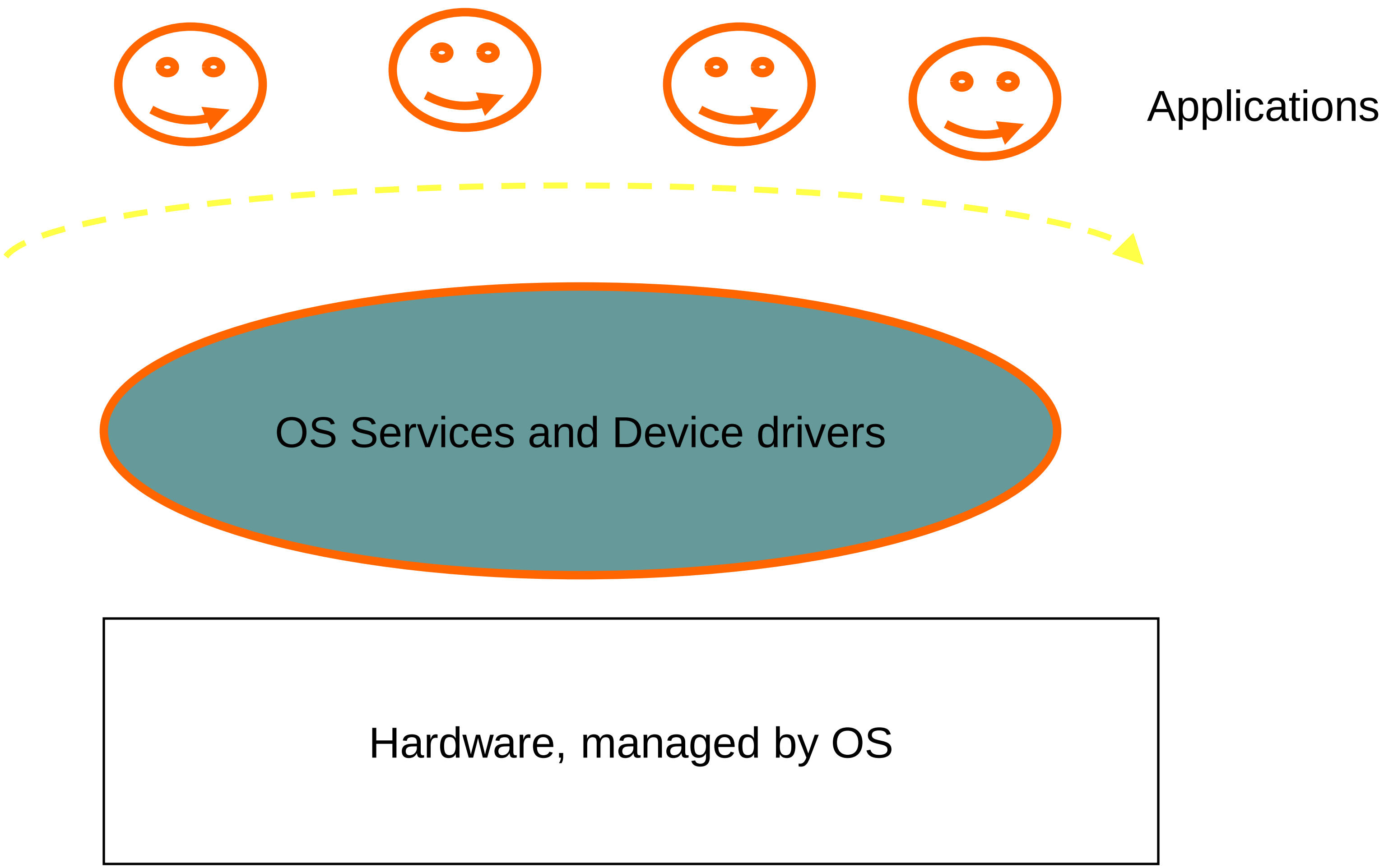
How expensive are border crossings?

- Procedure call: save some general-purpose registers and jump
- Mode switch:
  - Trap or call gate overhead
    - Nowadays syscall/sysreturn
  - Switch to kernel stack
  - Switch some segment registers
- Context switch?
  - Change address space
  - This could be expensive; flush TLB, …

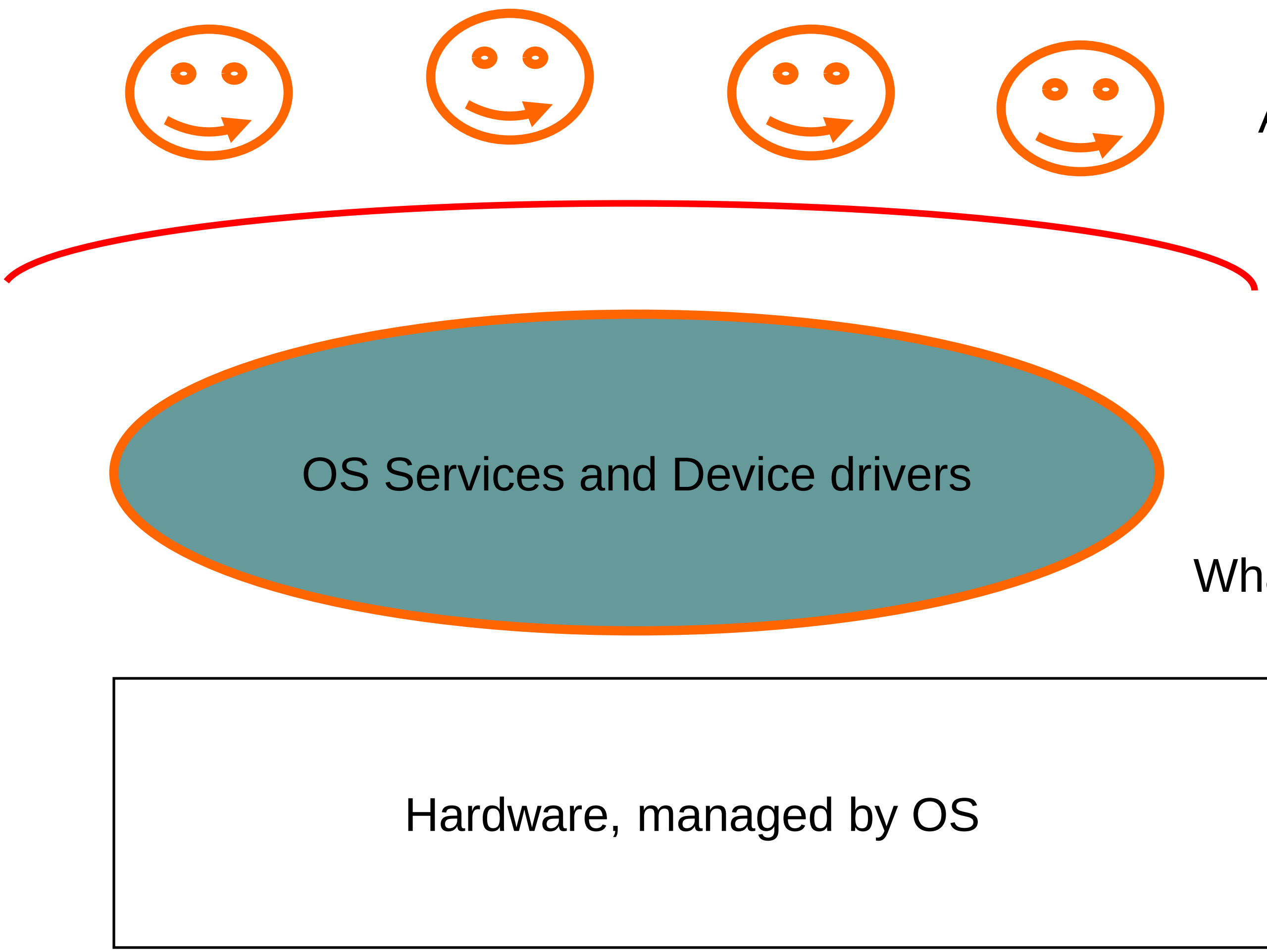
OS design models

- Library OS
- Monolithic Kernel
- Micro Kernel

OS as library (DOS-like)



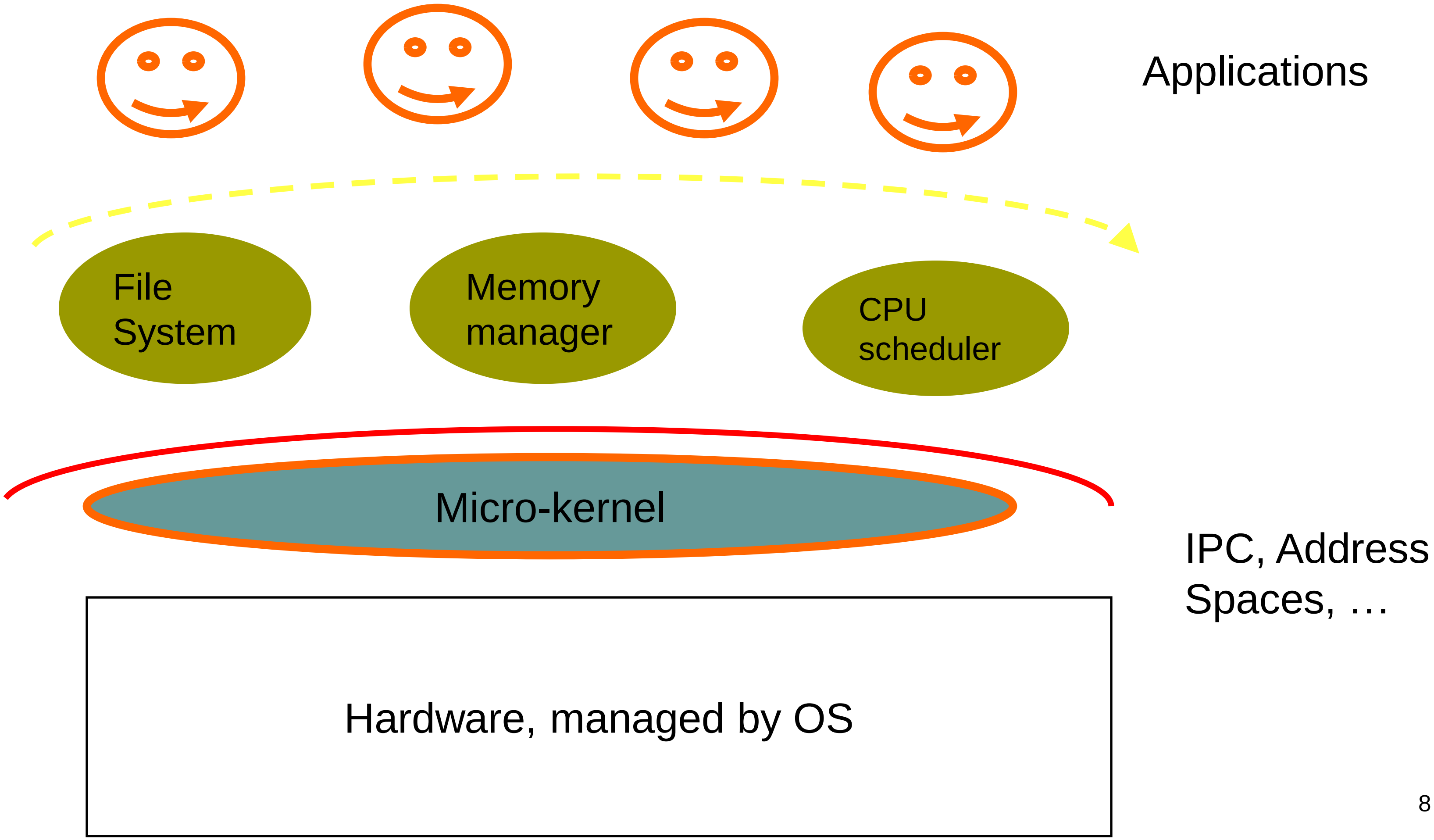
Monolithic Kernel



Applications

What is the difference

Micro-kernel



Summary

- DOS-like structure:
  - good performance and extensibility
  - Bad protection
- Monolithic kernels:
  - Good performance and protection
  - Bad extensibility
- Microkernels
  - Very good protection
  - Good extensibility
  - Bad performance!

Existing Approaches (cont'ed)

- Language Runtime
  - JavaScript for Browser
  - SPIN for OS
  - Good efficiency
  - Good security

- Software Fault Isolation (not required)
  - E.g., Google NativeClient

What should an extensible OS do?

- It should be thin, like a micro-kernel
  - Only mechanisms (or even less?)
  - no policies; they are defined by extensions
- Fast access to resources, like DOS
  - Eliminate border crossings
- Flexibility without sacrificing protection or performance
- Basically, fast, protected and flexible

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

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 obje
- Resolve:
  - Names are resolv
    - Once resolved, a
- Combine
  - To create an aggr

```
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.
```

- This is the key to spin – protection, extensibility and performance

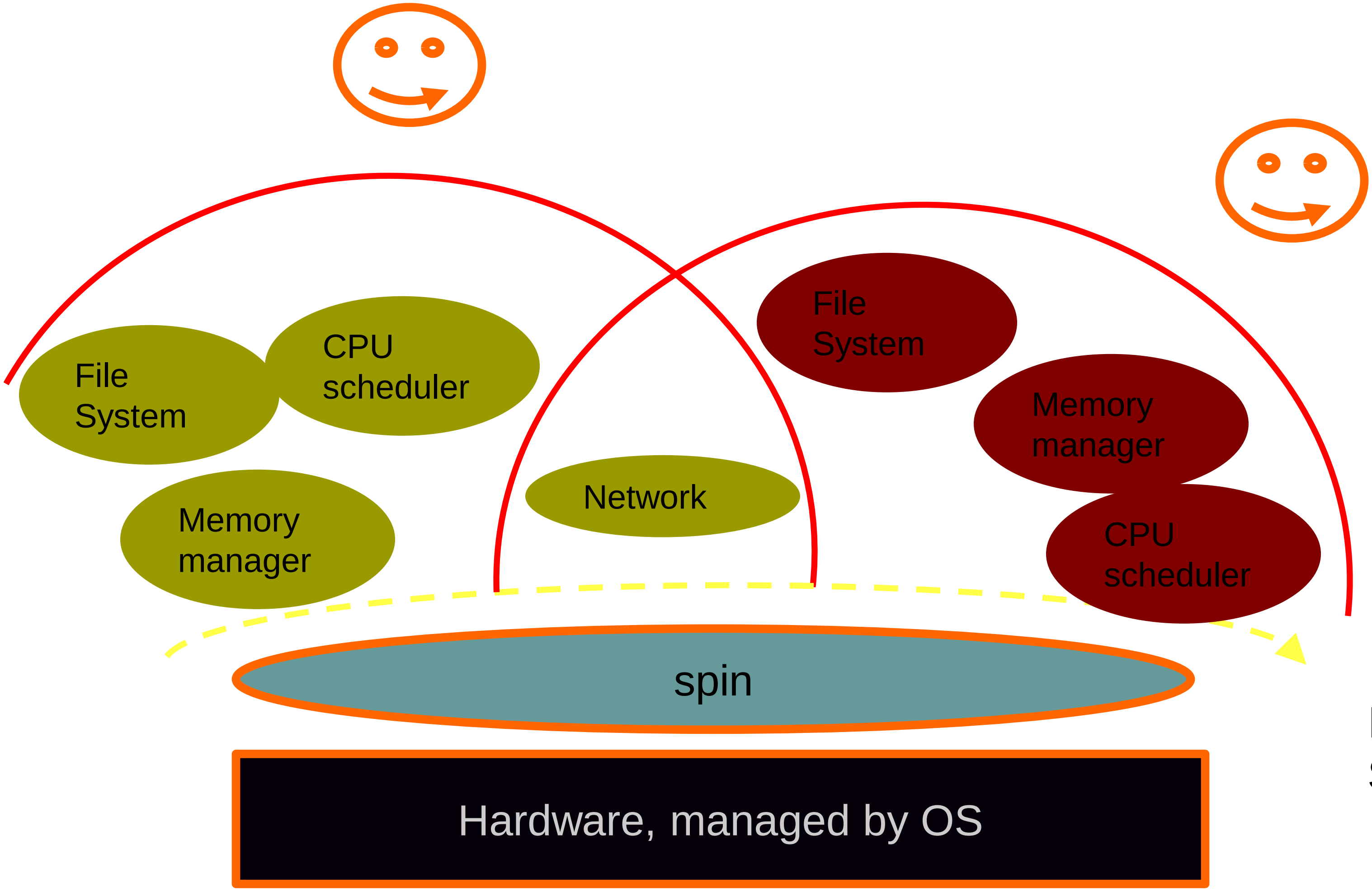
Protection Model (I)

- 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

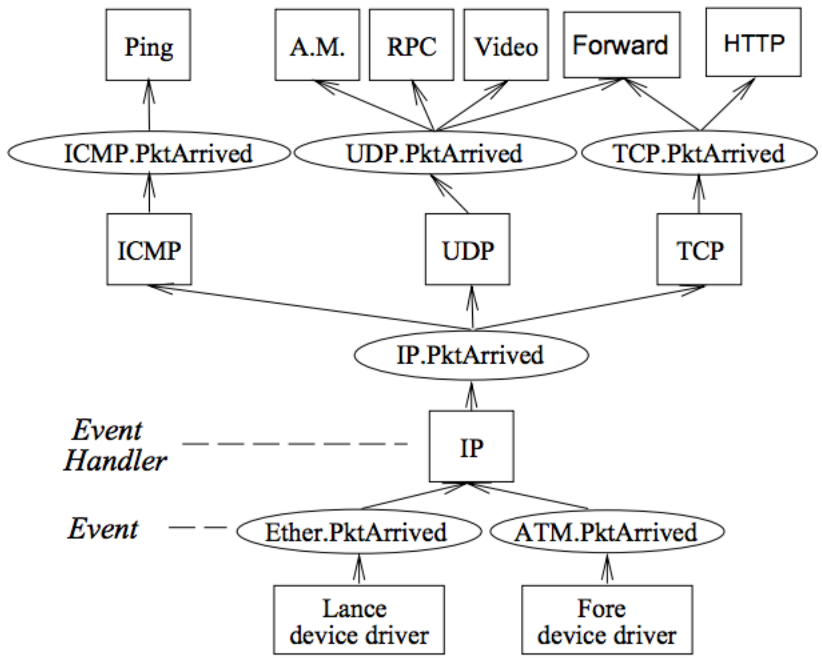


Figure 5: *This figure shows a protocol stack that routes incoming network packets to application-specific endpoints within the kernel. Ovals represent events raised to route control to handlers, which are represented by boxes. Handlers implement the protocol corresponding to their label.*

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.*

- Page fault, access fault, bad address

CPU Scheduling

- Spin
  - Se
- Eve
  - Bl
- Spin
  - Int … age

```
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

- System Components

Component	Source size		Text size		Data size	
	lines	%	bytes	%	bytes	%
sys	1646	2.5	42182	5.2	22397	5.0
core	10866	16.5	170380	21.0	89586	20.0
rt	14216	21.7	176171	21.8	104738	23.4
lib	1234	1.9	10752	1.3	3294	.8
sal	37690	57.4	411065	50.7	227259	50.8
Total kernel	65652	100	810550	100	447274	100

- Microbenchmarks
  - Protected communication
  - Thread management
  - Virtual memory
- Networking
- End-to- end Performance

Conclusions

- Extensibility, protection and performance
- Extensibility and protection provided by language/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?