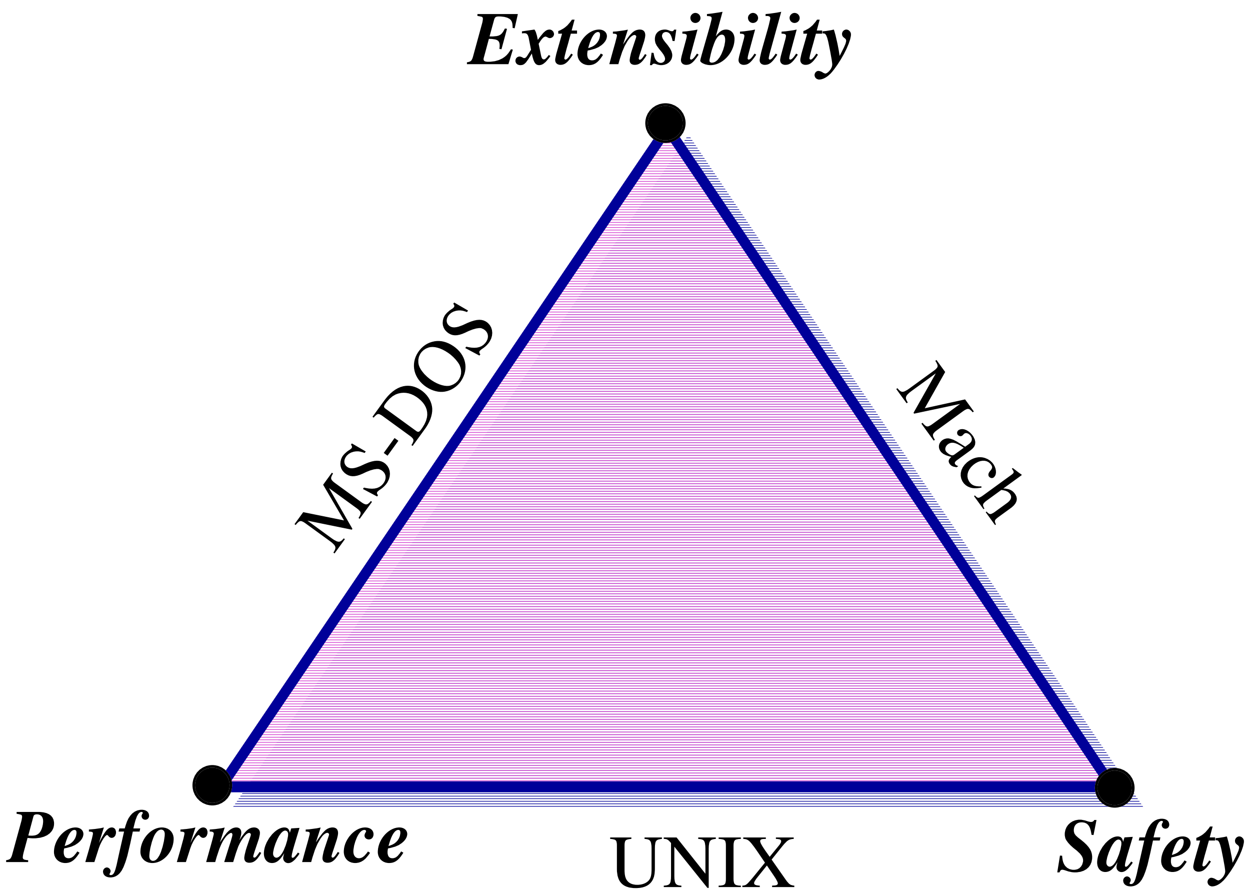
Language Support for Extensible Operating Systems

Emin Gun Sirer
The SPIN Project
University of Washington

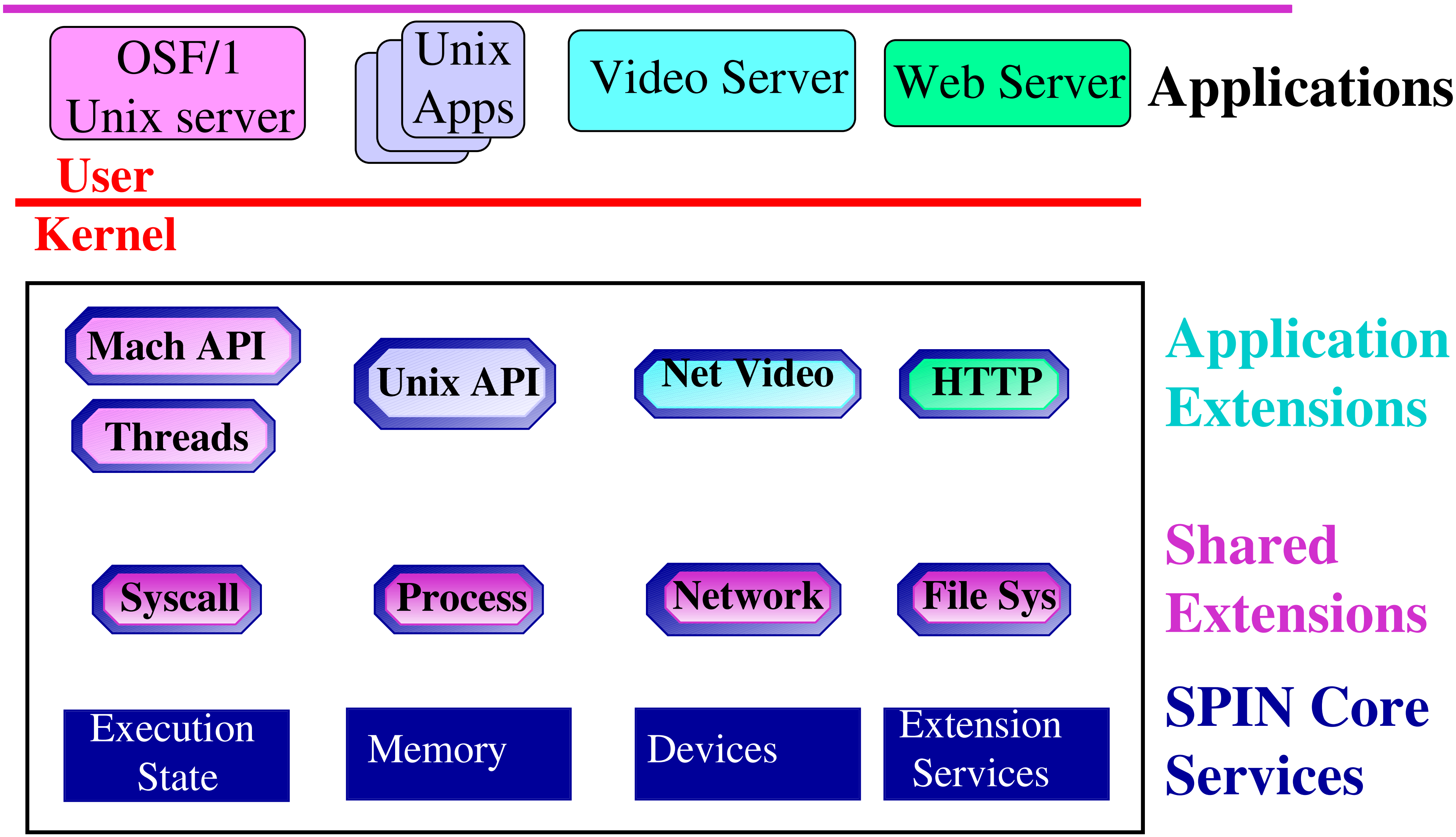
Motivation



Approach

- Extensibility
  - » Allow applications to extend any service
- Performance
  - » Dynamically inject application code into the kernel
- Safety
  - » Rely on language protection for memory-safety
  - » Rely on interface design for component-safety

Structure


OSF/1
Unix server
Unix
Apps
Video Server
Web Server
Applications
User
Kernel
Mach API
Threads
Unix API
Net Video
HTTP
Application
Extensions
Syscall
Process
Network
File Sys
Shared
Extensions
Execution
State
Memory
Devices
Extension
Services
SPIN Core
Services

Outline

- Motivation
- Structure of SPIN
- Protecting interfaces
- Language extensions
- Experience with Modula-3
- Conclusion

SPIN Protection Domains

- Kernel provided abstraction:
  - » *Logical Protection Domains*
- Handles for code management and linking
- Provide isolation within a single address space
- Named by capabilities

Operations on Domains

- Create
- Name
- Resolve
  - » Exercise access
- Export
  - » Share interfaces

```
MODULE TCP
...
   IP.Send(data);
```

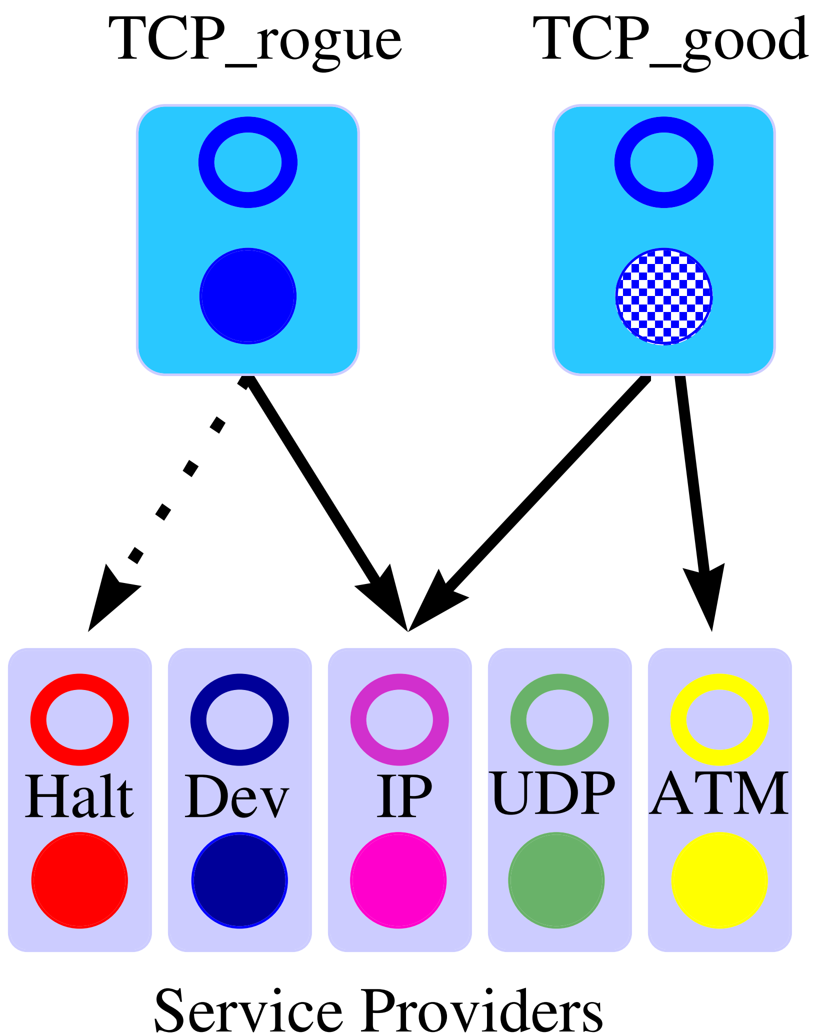
```
INTERFACE IP;

PROCEDURE Send();
```

```
DIp := Domain.Create(
          INTERFACE_UNIT(IP));
Nameserver.Register("ip",DIp);
```

```
DTcp := Domain.Create(
      ObjectFile);
Dip := Nameserver.Query("ip");
Domain.Resolve(DTcp, DIp);
Domain.Initialize(Dtcp);
```

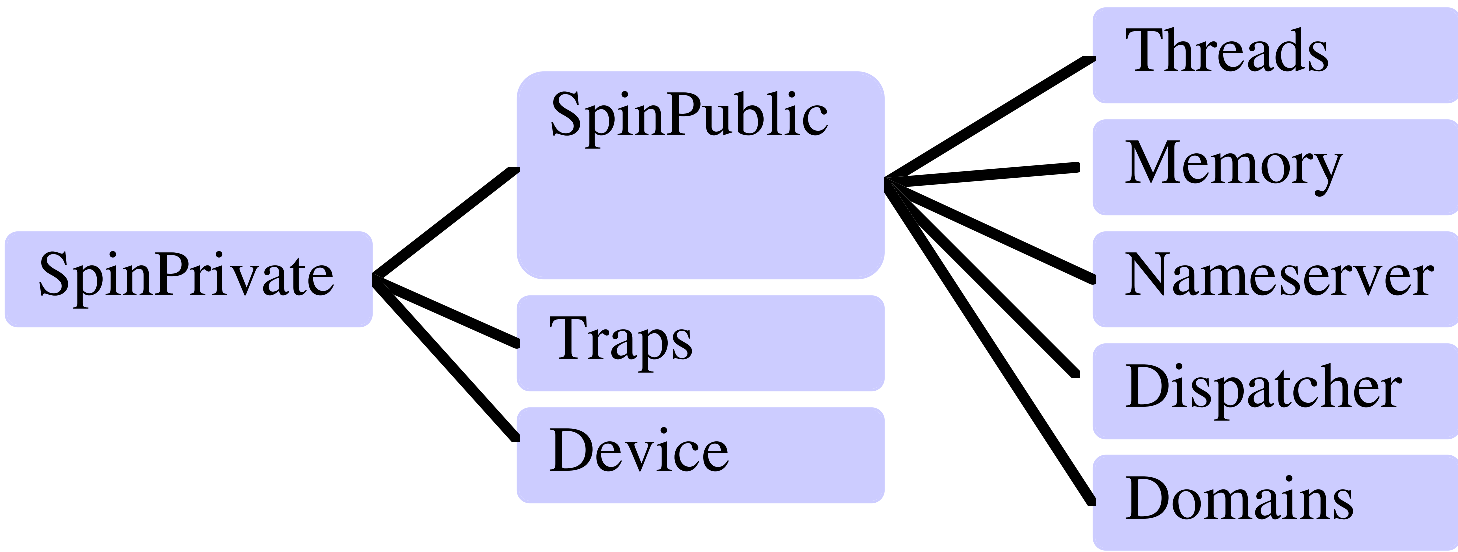
Using Domains



- *<u>Resolve</u>* symbol references to symbol definitions
- The types of the imported and exported symbol must match

Domains as Capabilities

- Domains nest to simplify capability management
- Binding code generated automatically
- Domain lookup through a nameserver



Domain Summary

- Logical protection domains within a single address space
- Complements type-safety to achieve system safety
- Sharing is cheap
  - » Share code by jumping directly
  - » Share data by passing pointers
- NO runtime overhead

Outline

- Motivation
- Structure of SPIN
- Protecting interfaces
- Language extensions
- Experience with Modula-3
- Conclusion

Language Extensions

- Run-time handles for interfaces and modules.
- Isolation of trust.
- Pointer-safe casting

Code Handles

- How do you name an interface ?
- Interfaces are not values
- Need a way to
  - » identify interfaces
  - » identify implementations
- INTERFACE_UNIT and MODULE_UNIT

Isolating Callers

- Execute untrusted code from interrupts
  - » Active messages
- Untrusted clients may not terminate
  - » Forceful termination may violate system state
- **`EPHEMERAL`** procedures can be terminated at any time
  - » Can only call other **`EPHEMERAL`** procedures.

```
EPHEMERAL PROCEDURE ActiveMsgHandler(m: Mbuf.T) =
  BEGIN
    time := time + VIEW(m.data, TimeDelta.T);
  END;
```

Safe Casts

- View raw data as typed data
  - » OSes require viewing bits as typed objects
  - » Copying is expensive and violates sharing
- `WITH NewView = VIEW(var, T) DO … END;`
  - » Cannot forge pointers or create illegal values

Outline

- Motivation
- Structure of SPIN
- Protecting interfaces
- Language extensions
- Experience with Modula-3
- Conclusion

Language Evolution

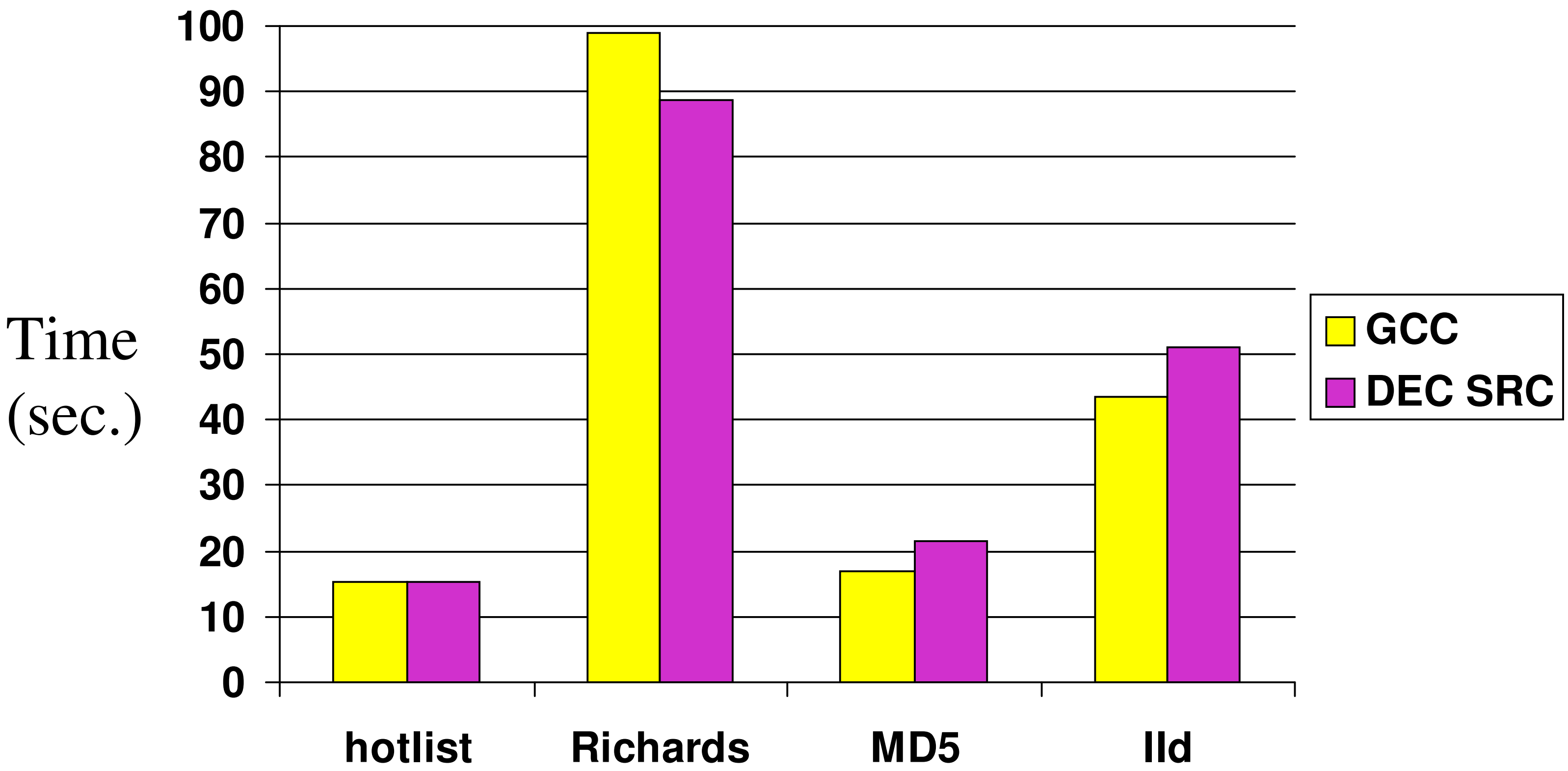
- Developed Cove
  - » Safe "super-subset" of C
  - » Pointer safety, tagged storage, exceptions, pre- and post-conditions, interfaces, modules
- Adopted Modula-3
  - » DEC SRC standard implementation
  - » Vortex-based optimizing compiler

Modula-3 Concerns

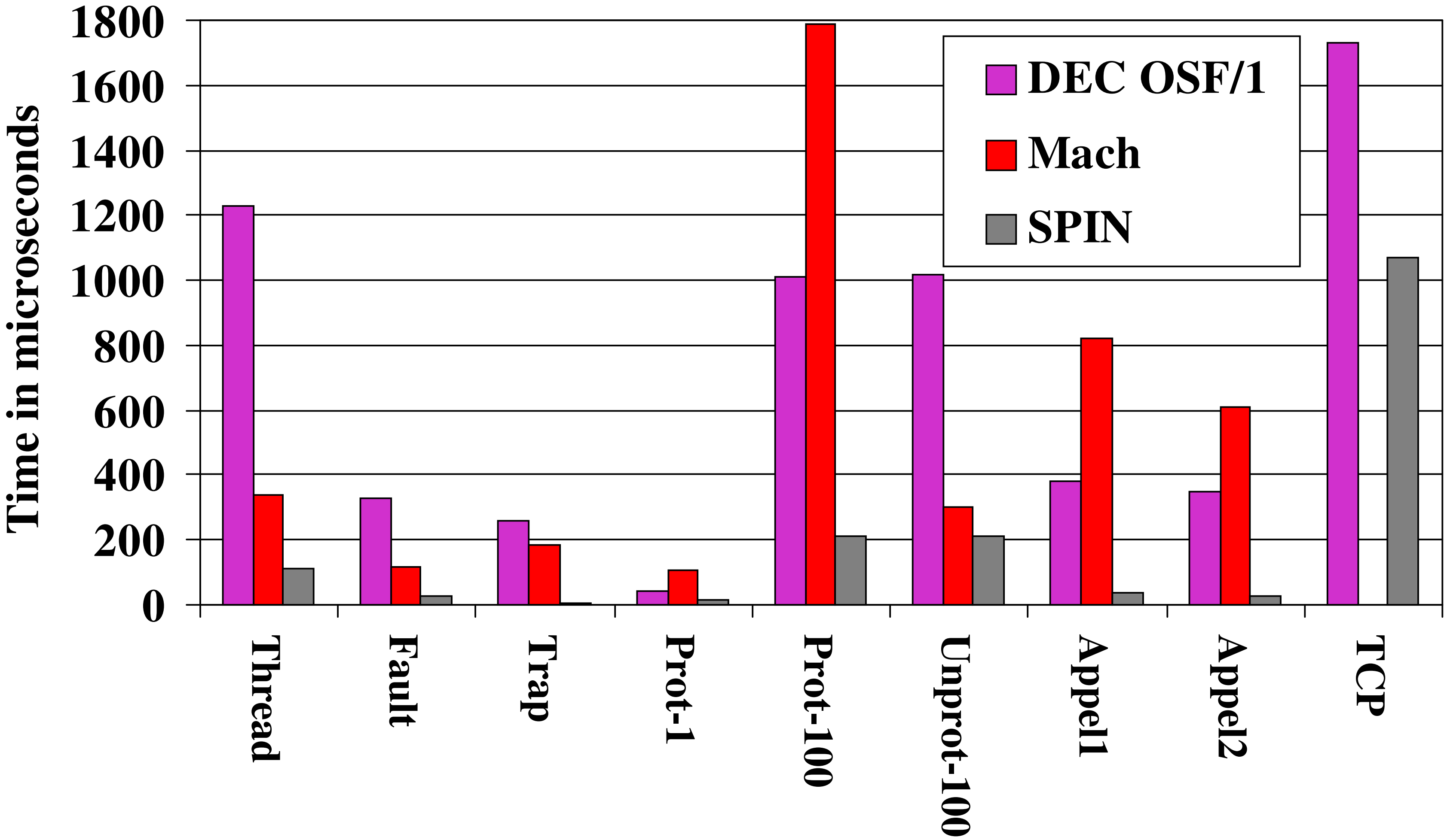
- Execution speed
- Threads, allocation, GC
- Memory usage
- Mixed-language environment

Execution speed

- Performance is comparable to that of C.



System Performance



Memory usage

- Code and data size is small
- Sharing reduces memory requirements
- Typical examples:
  - » Web server extension: 9K
  - » Cthreads Package: 4K
  - » TCP forwarder: 6K

Runtime Services

- Threads
  - » DEC SRC fork/join: 700 usecs.
  - » SPIN fork/join: 22 usecs.
- Allocator overhead
- Garbage collector overhead
  - » Enable incremental, generational collection

Mixing Languages

- Control transfer
  - » Automatic generation of C header files (C -> M3)
  - » Unsafe EXTERNAL pragma (M3 -> C)
- Data sharing
  - » Data layout is identical to that of C
  - » Immobilize heap data when sharing with C

Conclusions

- Language and compiler support enables faster and more flexible systems
- Retain full safety

`http://www-spin.cs.washington.edu`