A User-Level Unix Server for the *SPIN* Operating System

David Dion

October 24, 1996

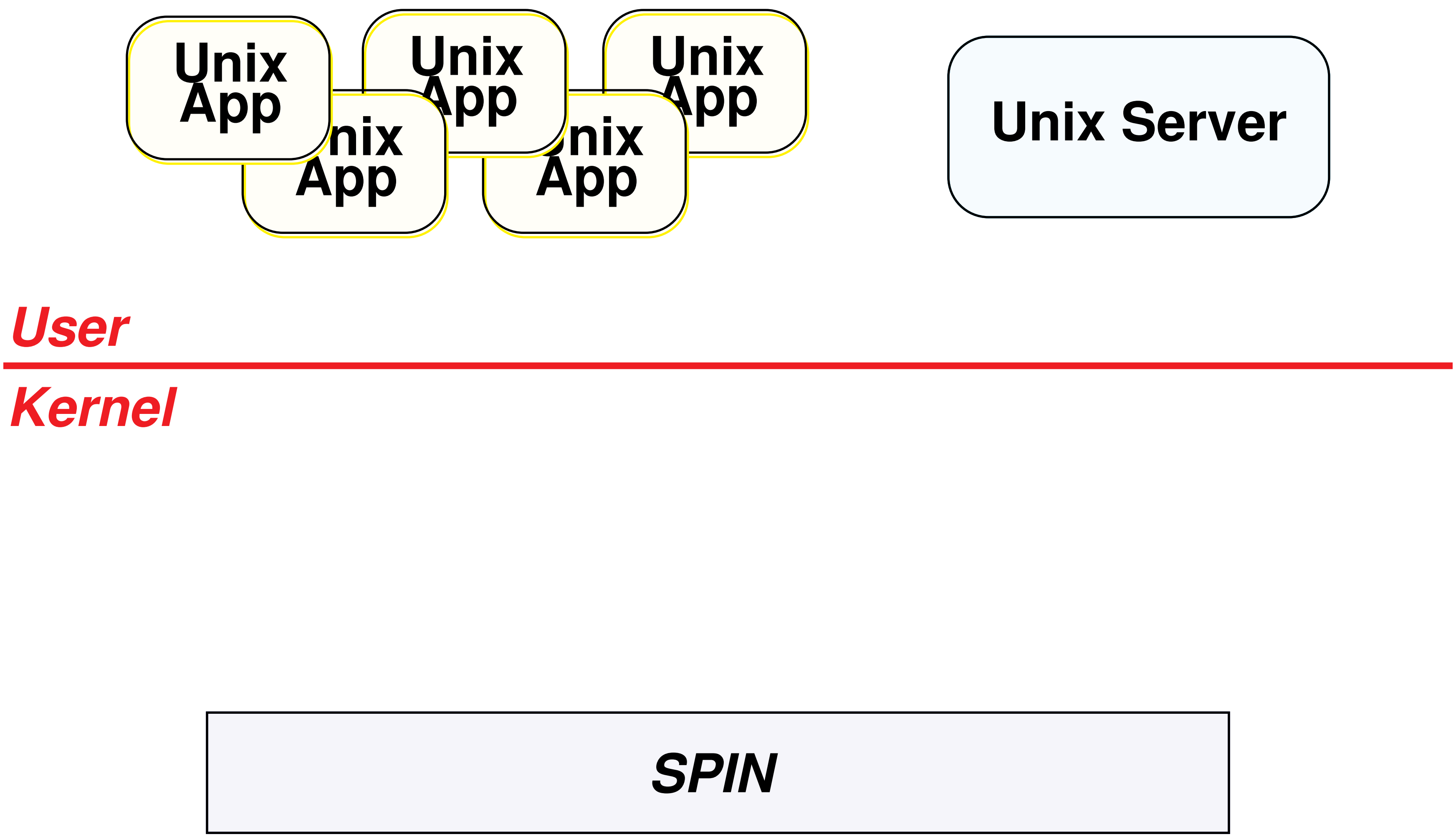
Objectives

Emulate Unix with binary compatibility

Reuse a large piece of existing code

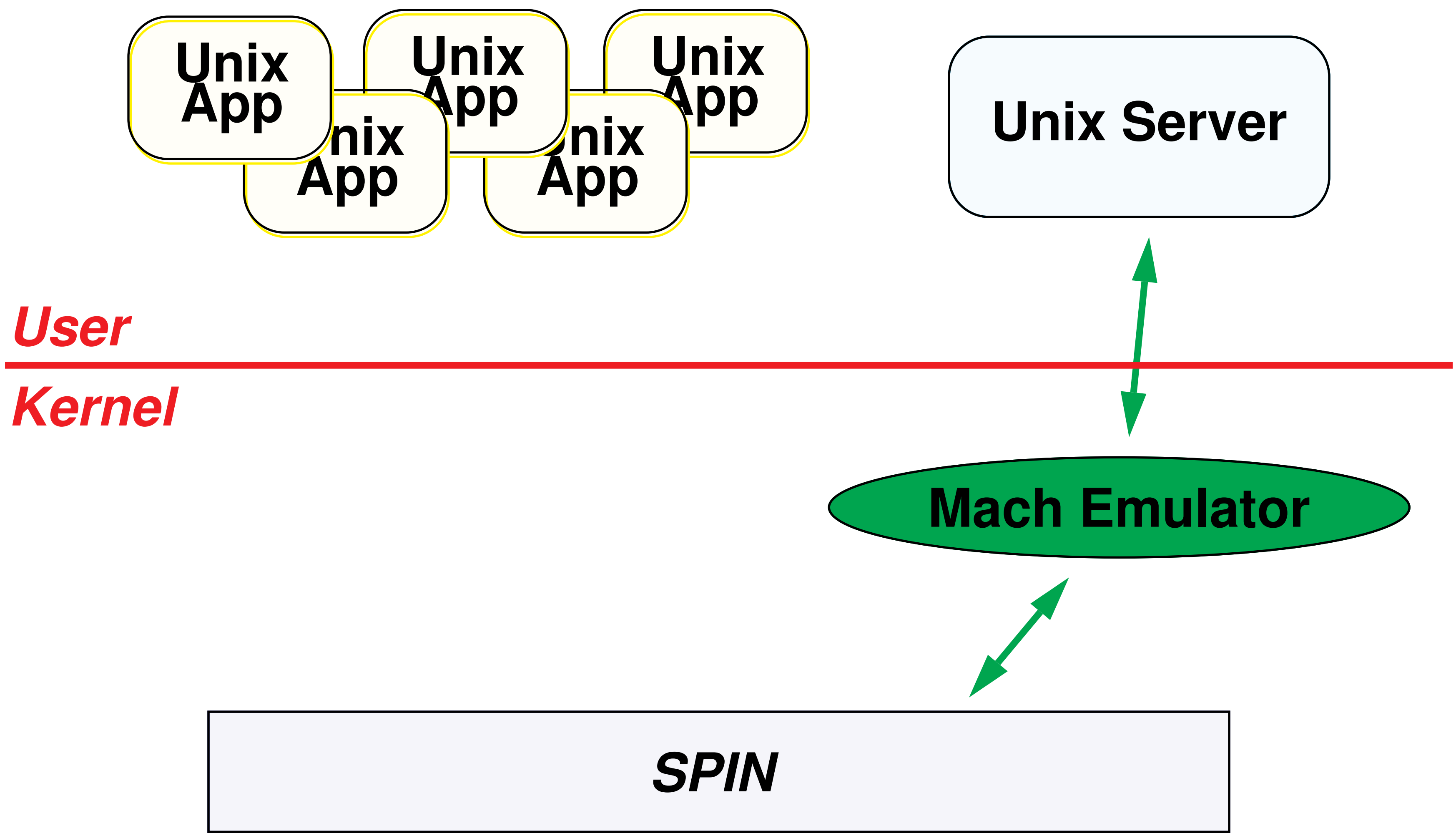
Run a large application on *SPIN*

Approach

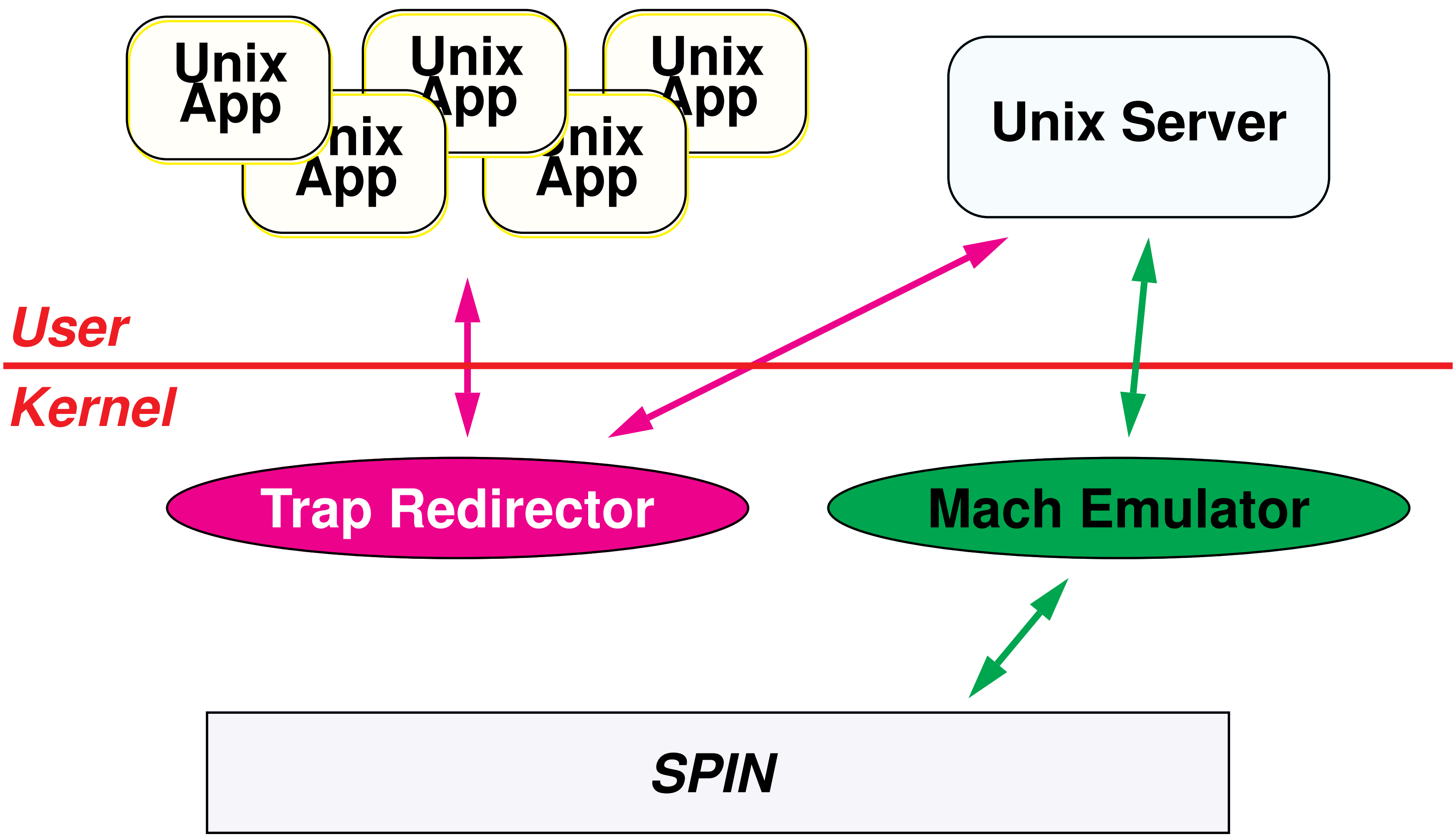


Unix App
Unix App
Unix App
Unix App
Unix App
Unix Server
User
Kernel
SPIN

Approach


Unix App
Unix App
Unix App
Unix App
Unix App
Unix Server
User
Kernel
Mach Emulator
SPIN

Approach


Unix App
Unix App
Unix App
Unix App
Unix App
Unix Server
User
Kernel
Trap Redirector
Mach Emulator
SPIN

Unix Applications

Lots of popular programs

Well-known system interface

SPIN

Safety, extensibility, performance

Untrusted applications dynamically extend the kernel

SPIN Extensions

Bind *handlers* to *events*

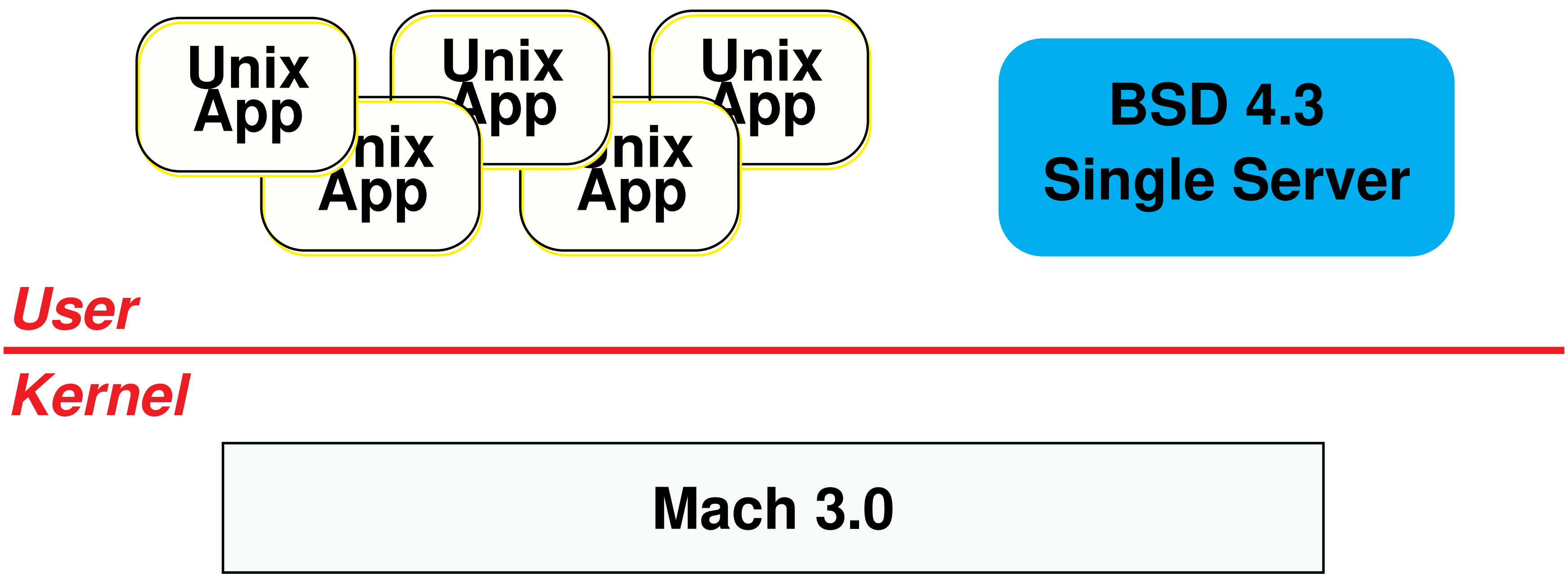
Event: change in system state

Handler: procedure that responds to an event

Example: system call is an event

Unix Server

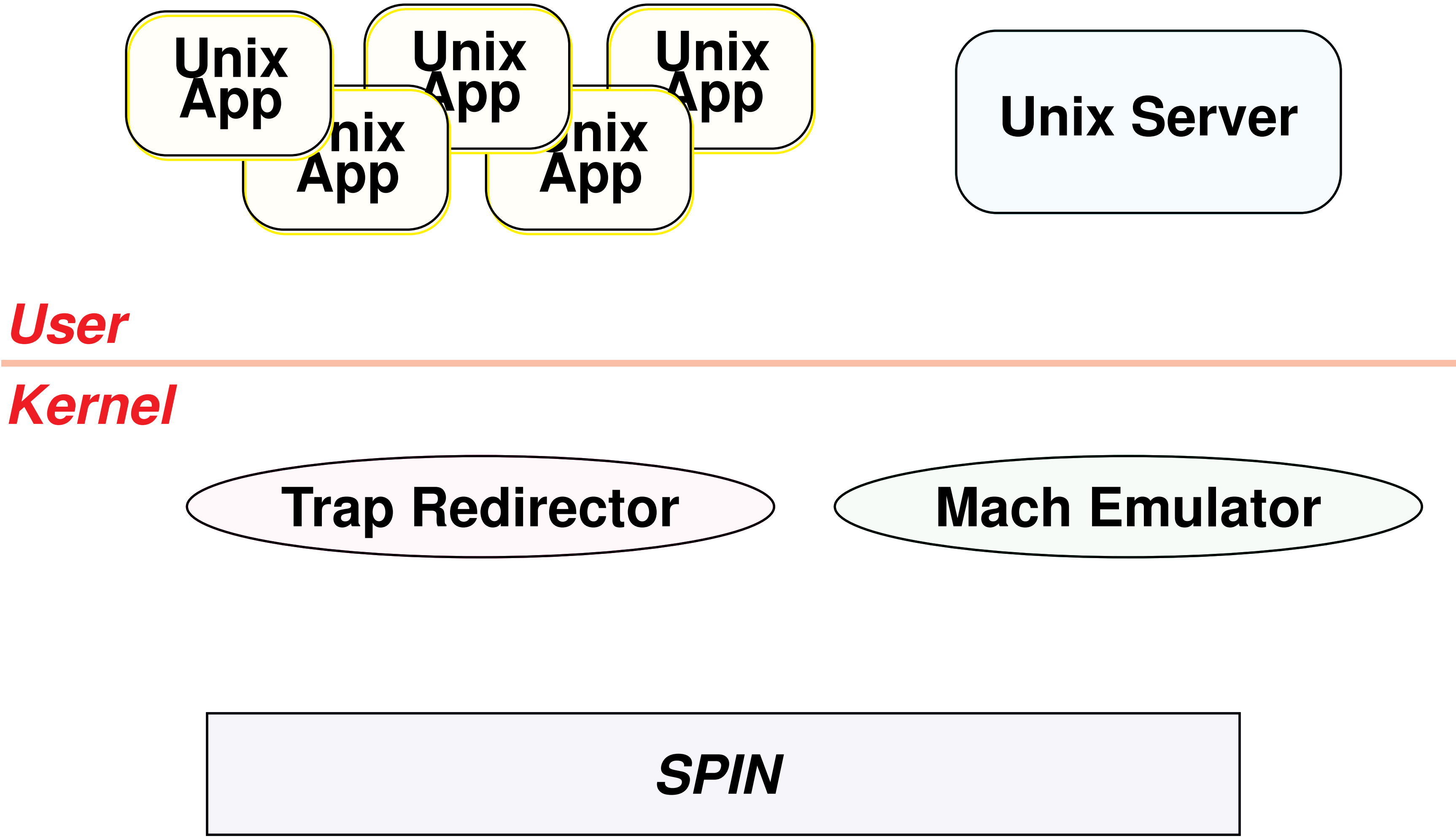
Once upon a time ...



Unix Server Evolution

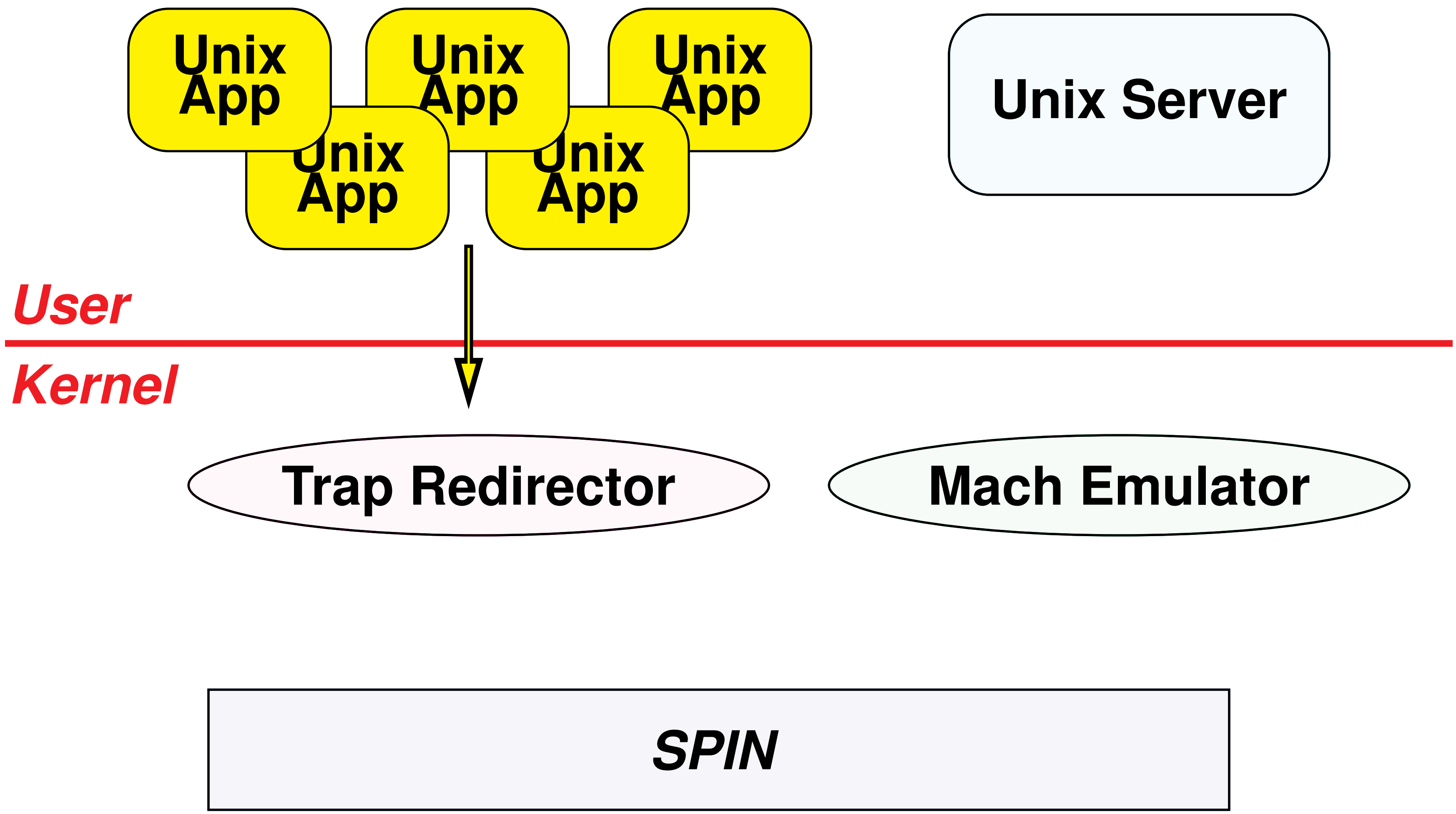
Life of a System Call

How does it fit together?



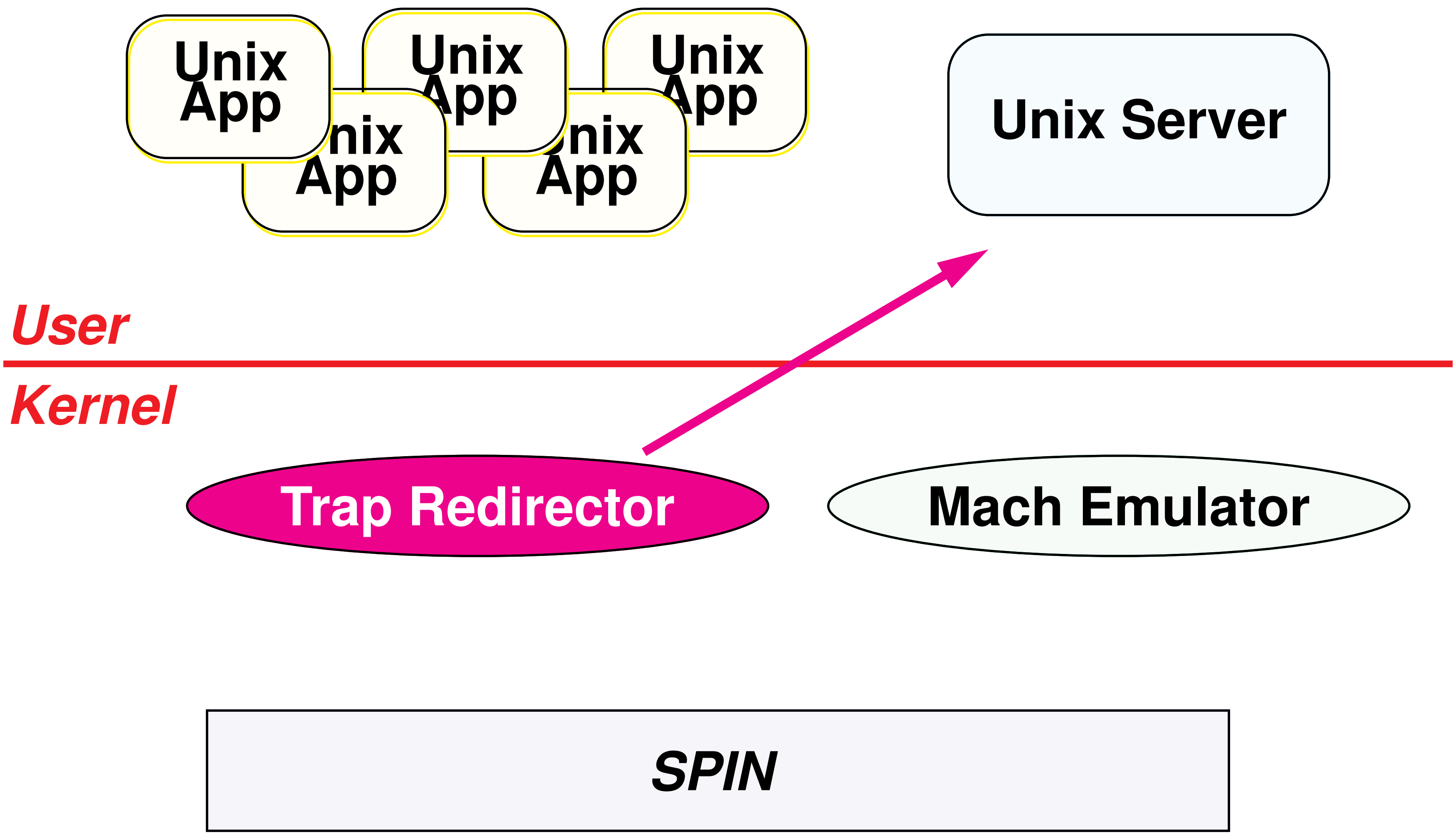
Life of a System Call

Invocation: read()



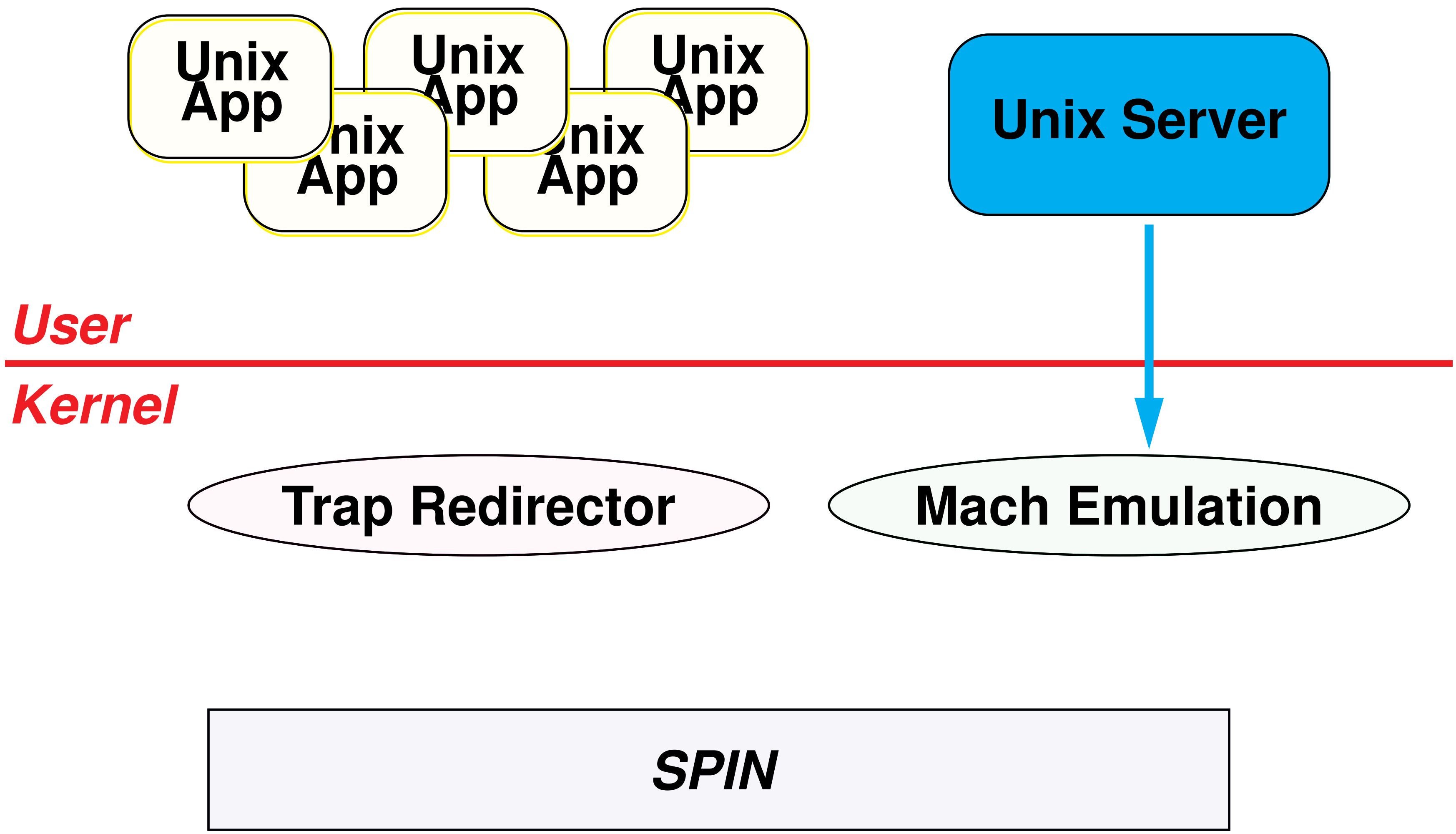
Life of a System Call

Intercept and hand-off to server



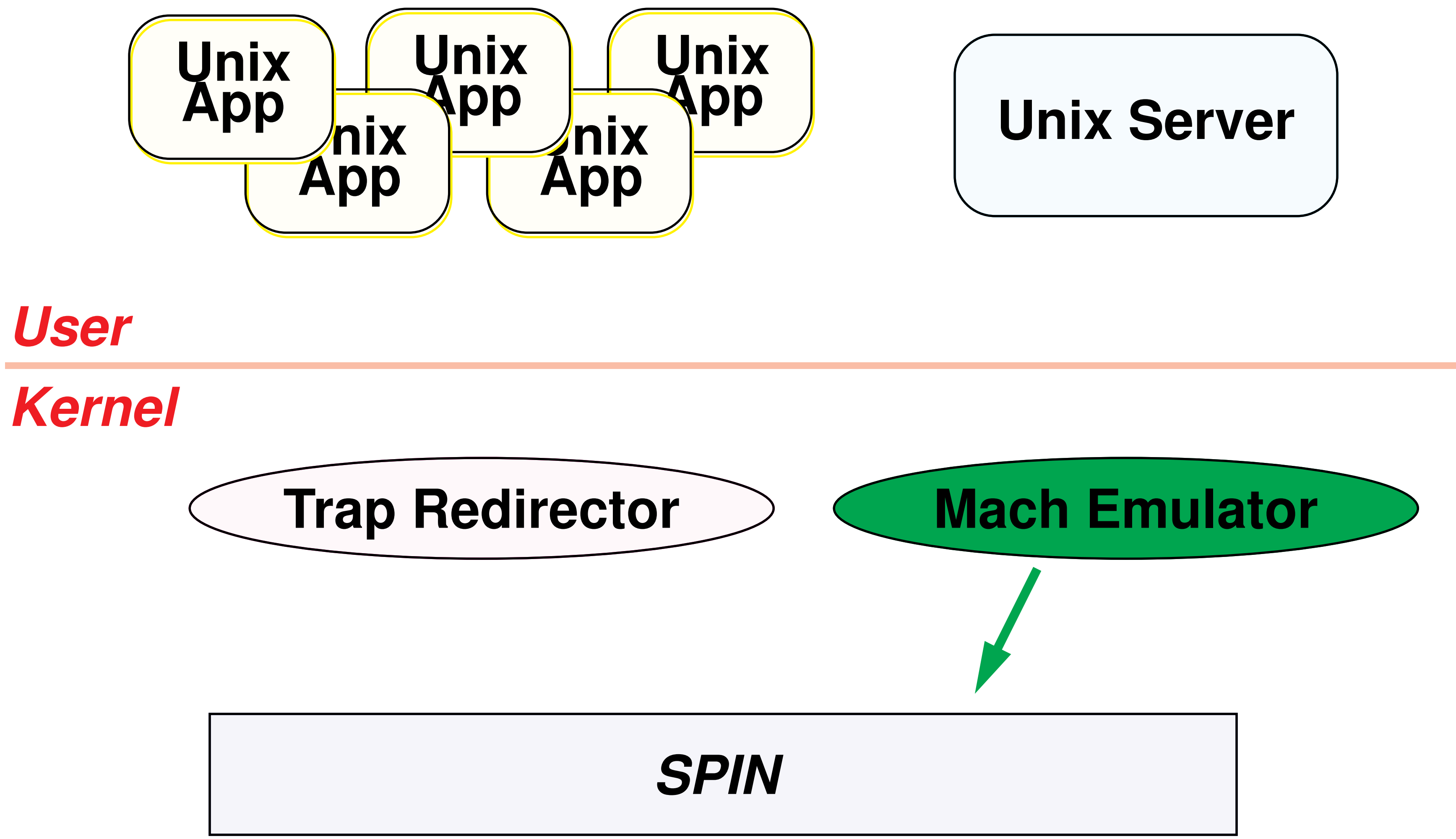
Life of a System Call

Invoke kernel services: device_read()



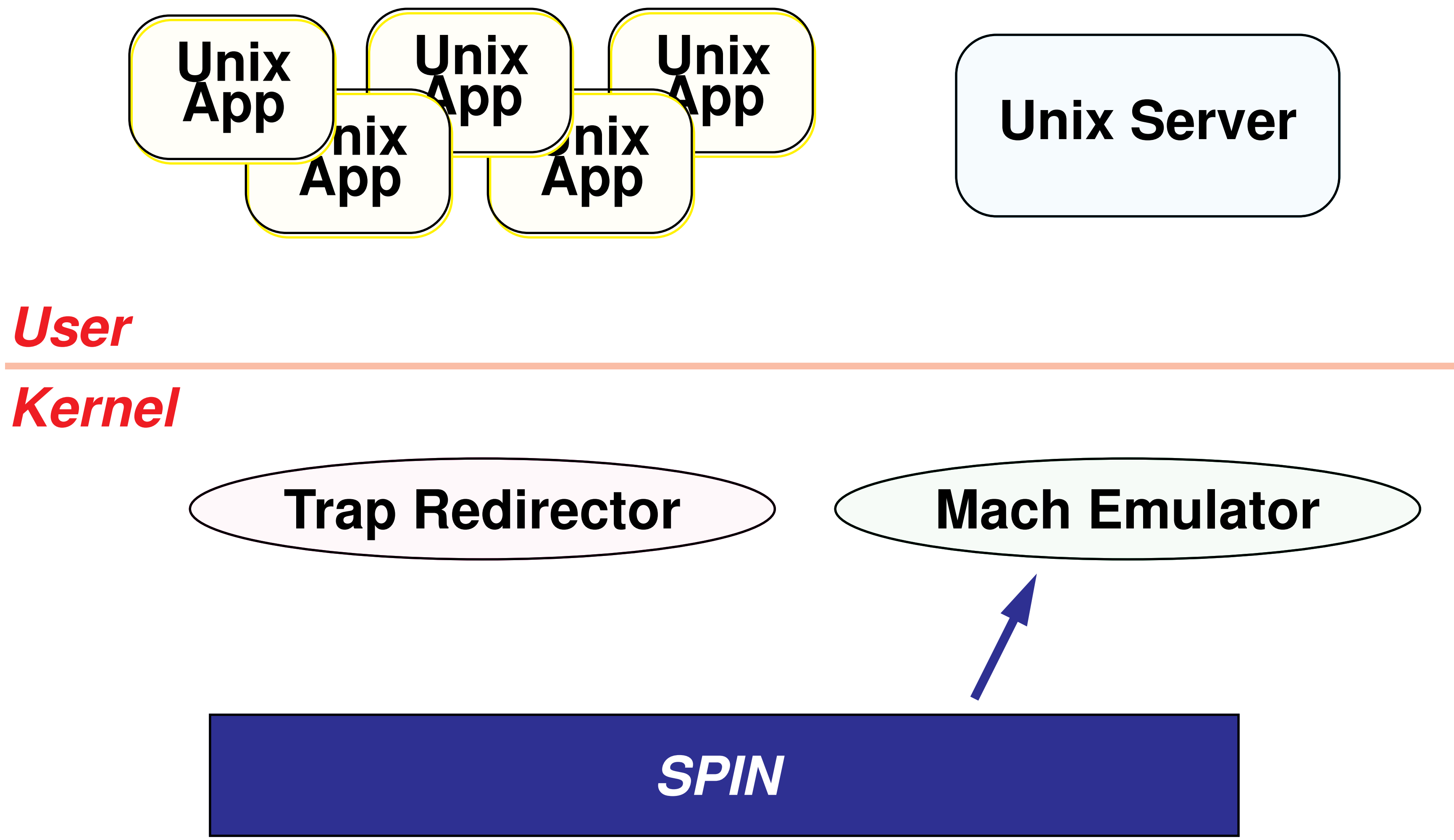
Life of a System Call

Invoke SPIN services: DiskDevice.Read()



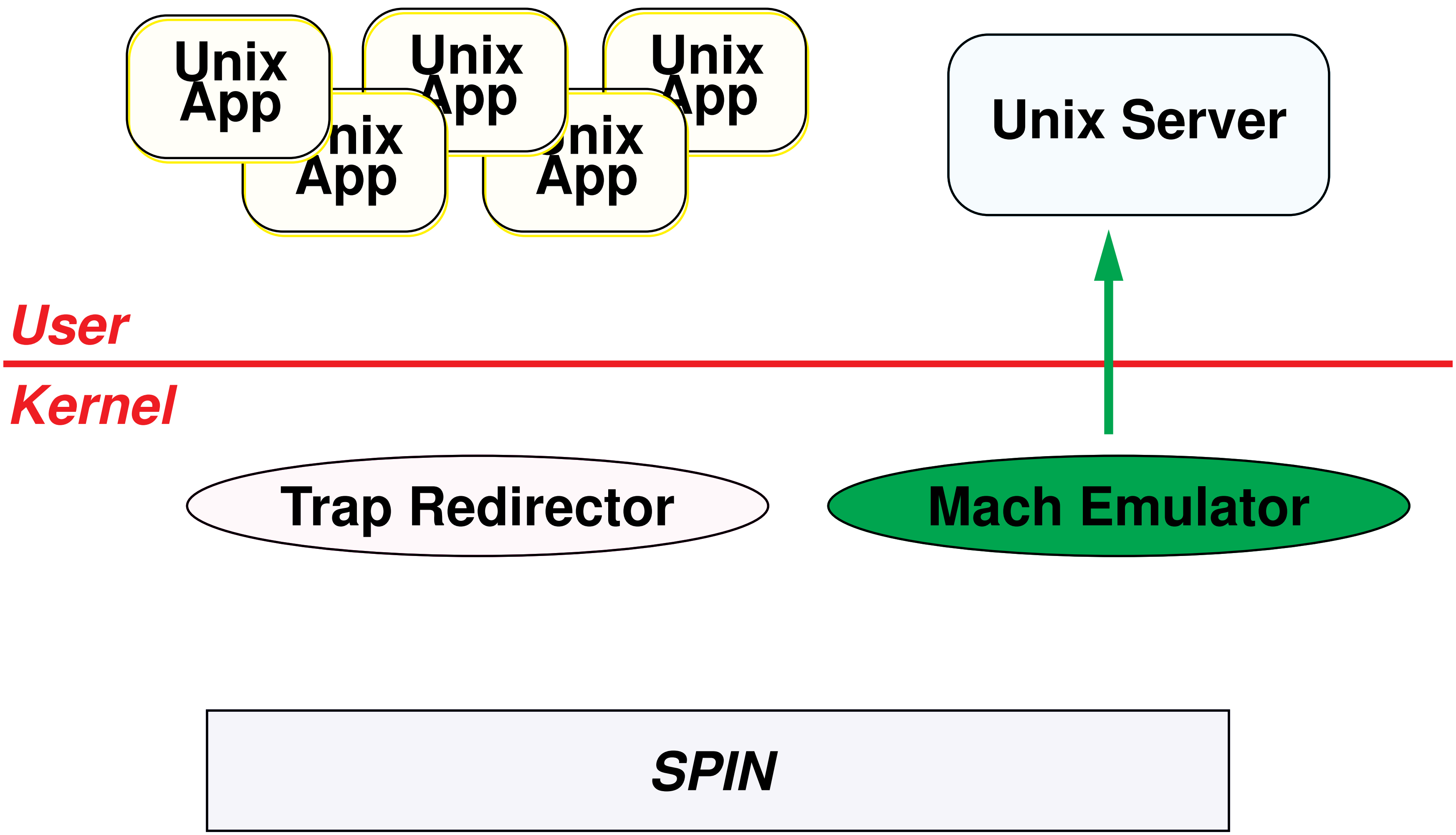
Life of a System Call

Get data from disk and give it to emulator



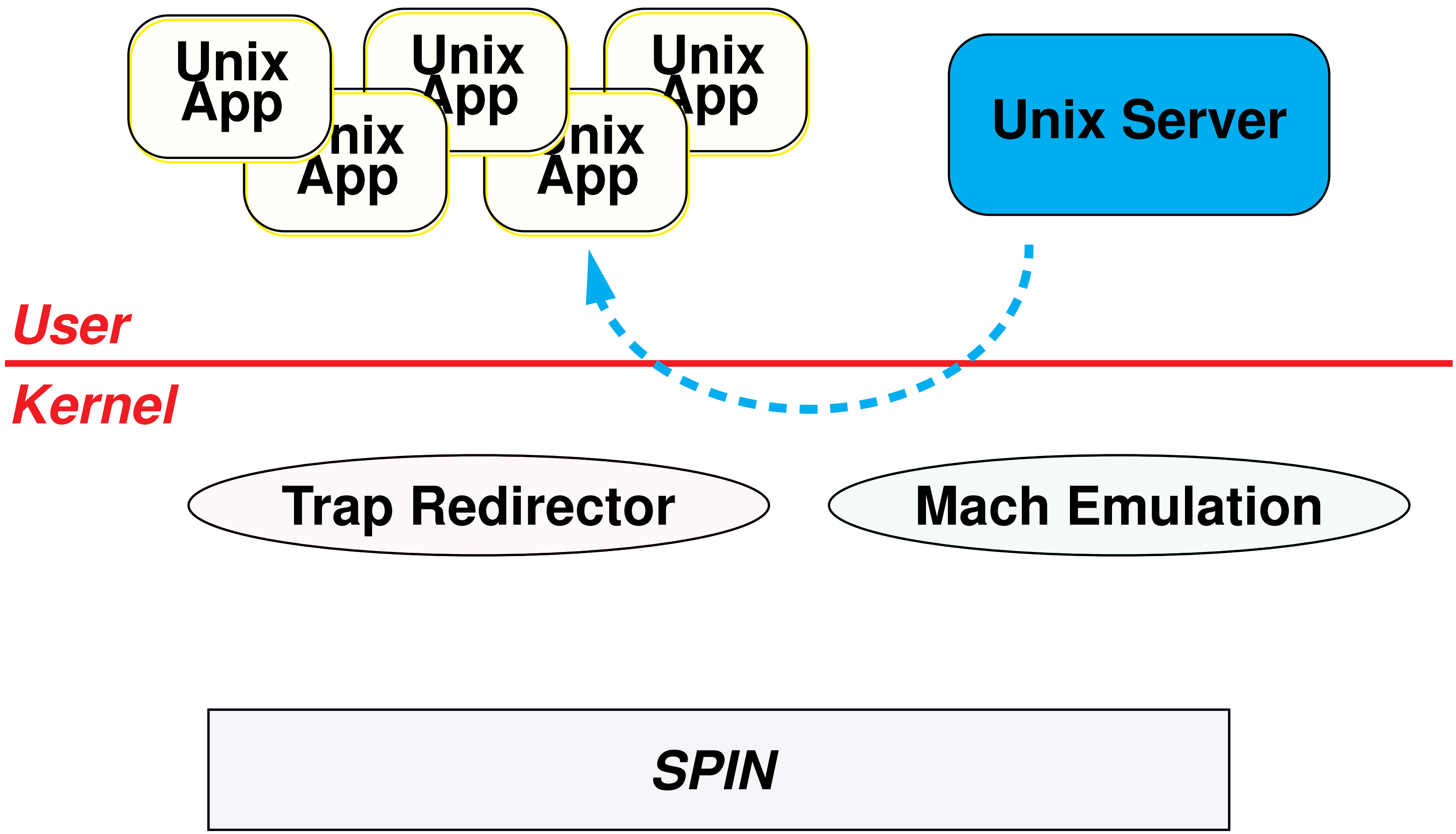
Life of a System Call

Copy data to server



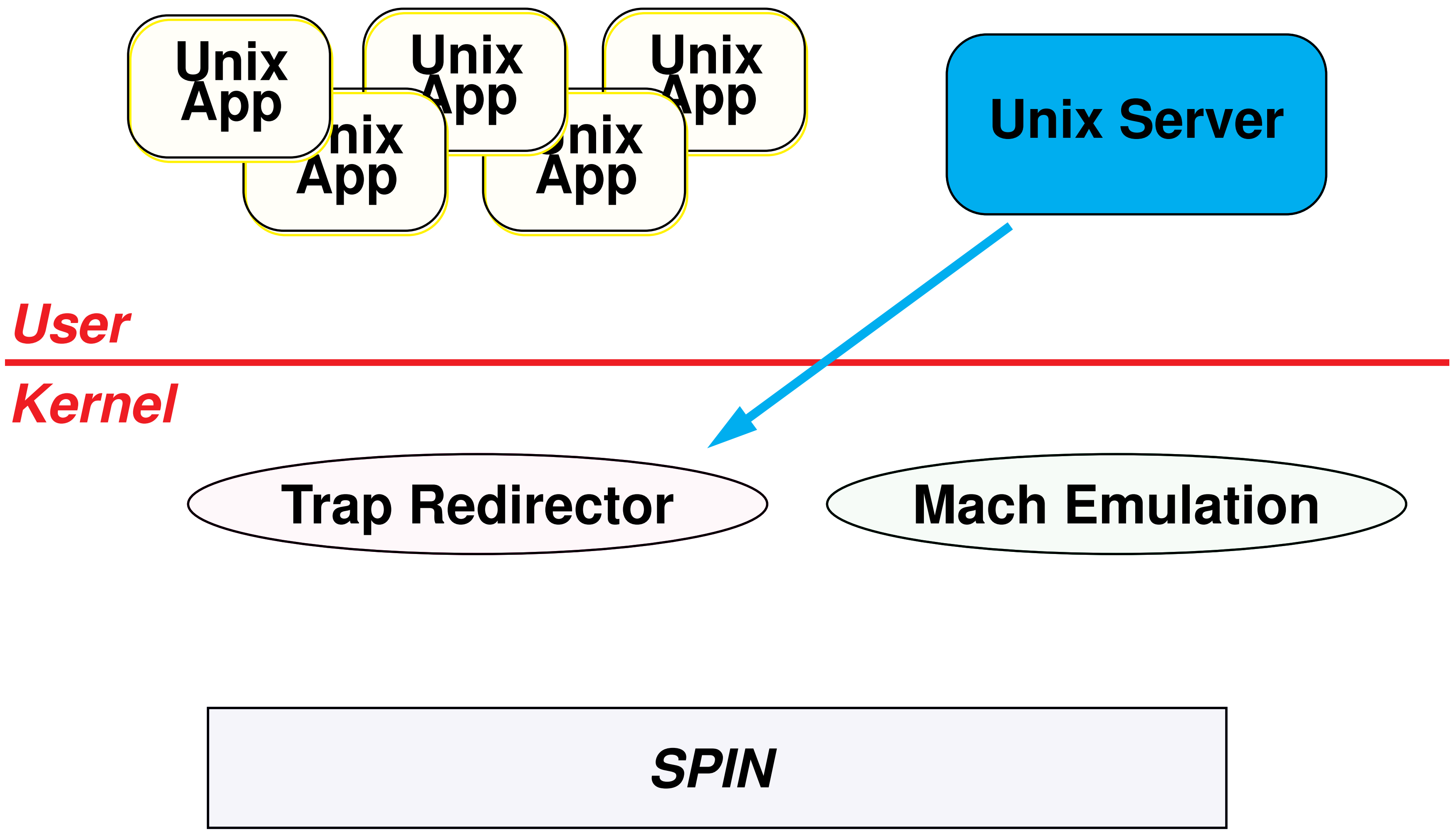
Life of a System Call

Copy data to application



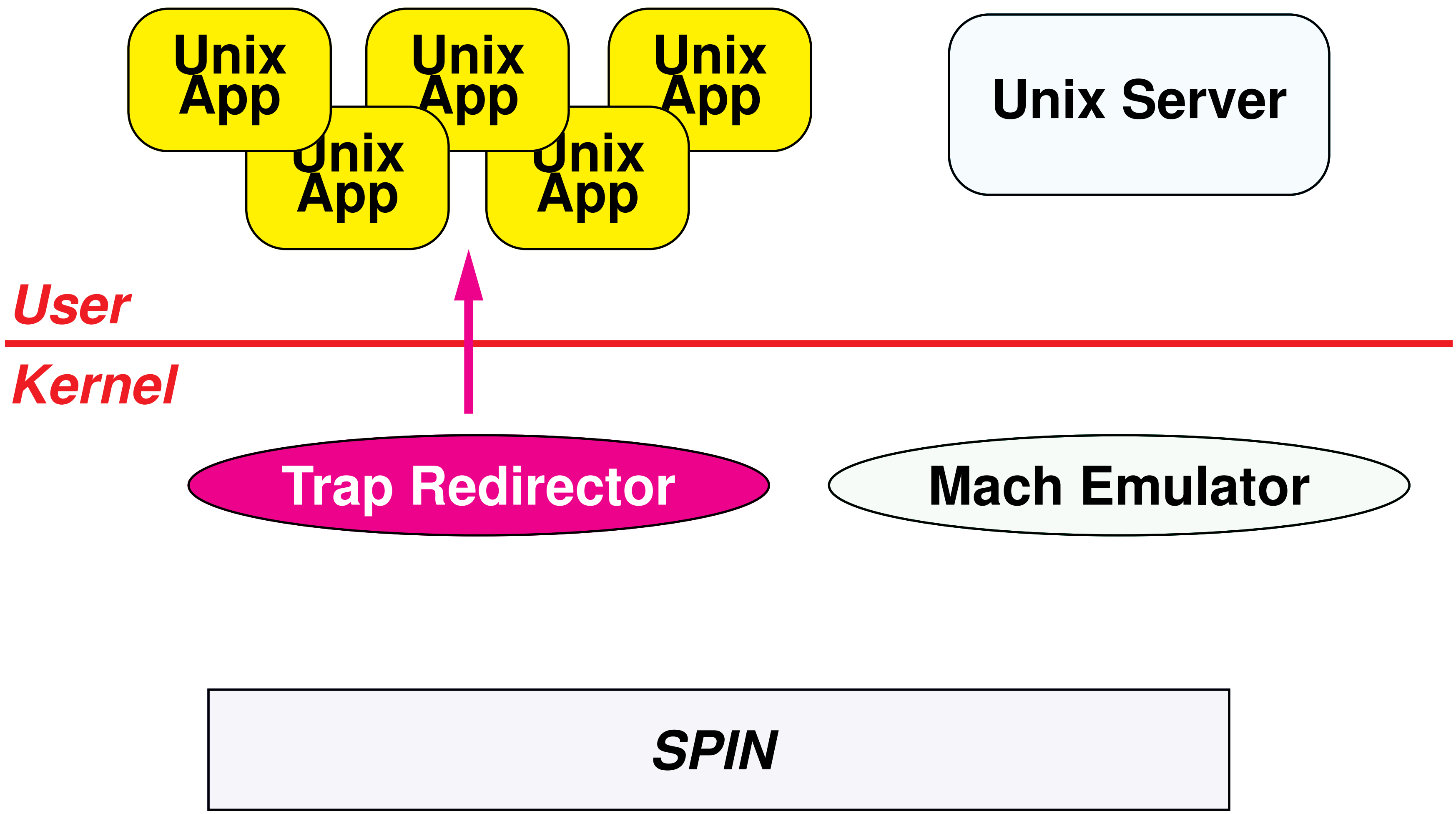
Life of a System Call

Return to Trap Redirector

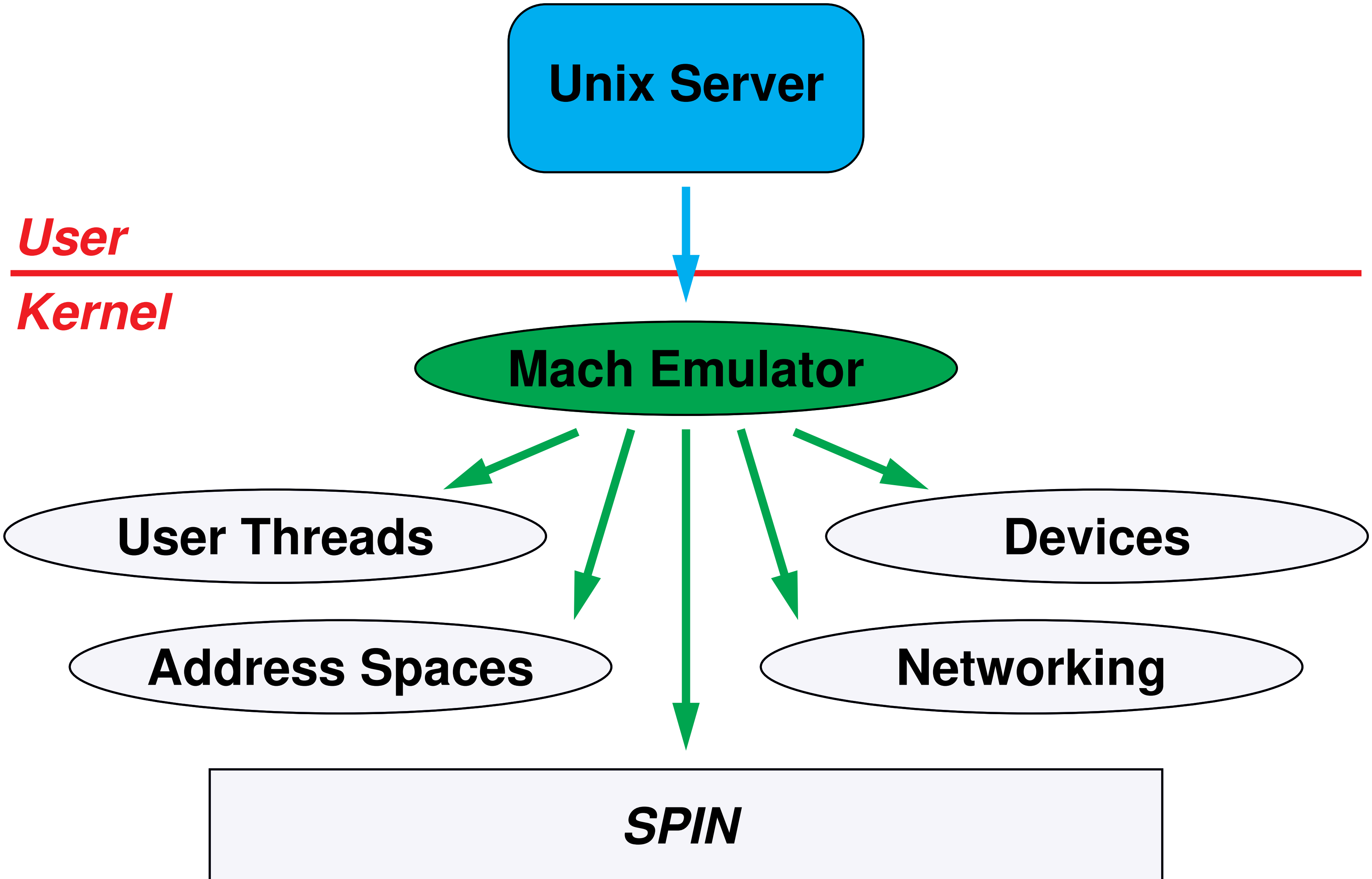


Life of a System Call

Return to application



Emulating Mach



Mach VM and Tasks

Mach VM/Task interface richer than *SPIN*

Address Space interface

Keep track of lots of data

Construct new services

Aggressive copying for correctness

Result: it works, but it’s not pretty

Performance

The user-level server model is inherently slow.

System call latency

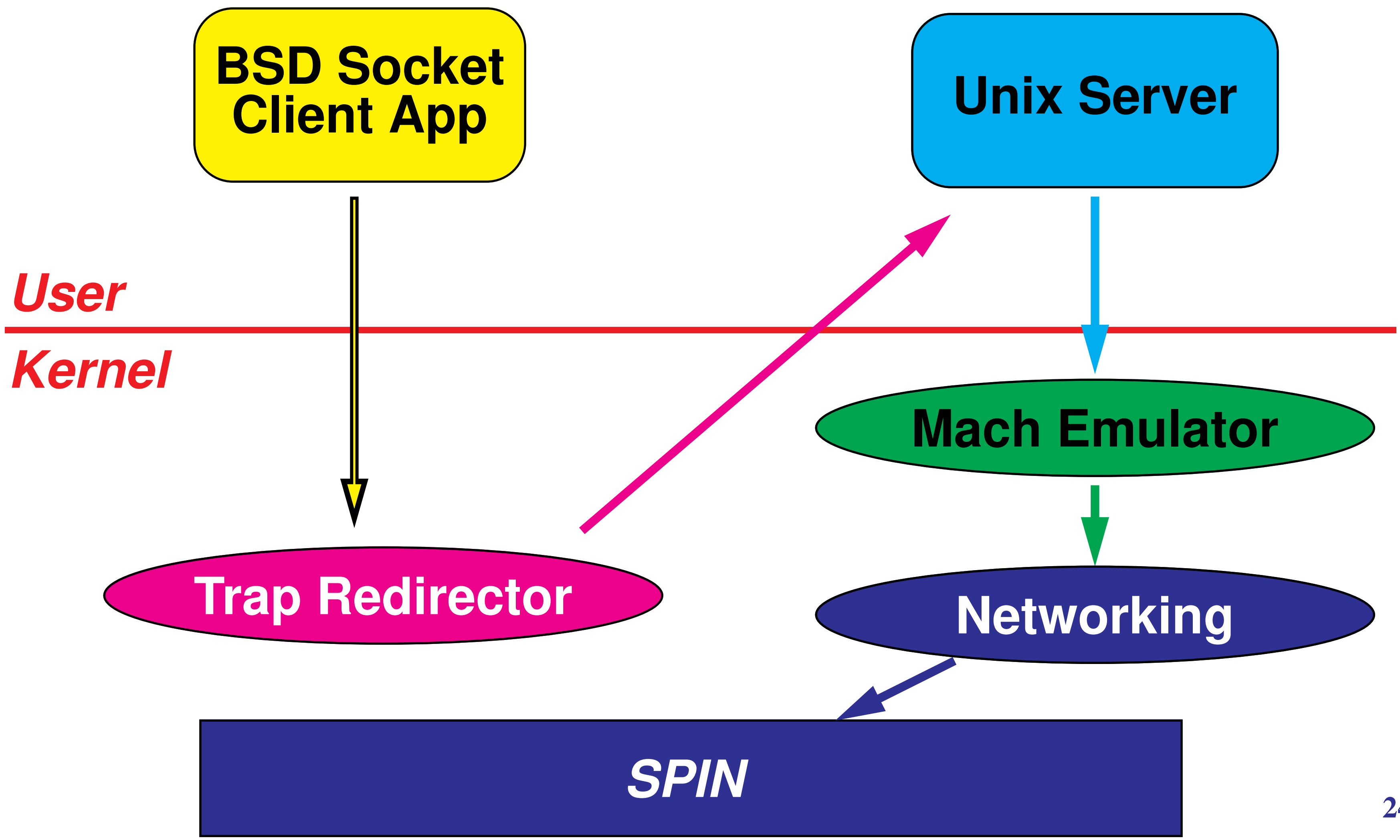
Operation	Digital Unix	*SPIN* + Unix Server
getpid()	2 μsec	597 μsec
read() 1KB	24 μsec	643 μsec

Application performance

Operation	Digital Unix	*SPIN* + Unix Server
ls -l (cold)	246 msec	2043 msec
ls -l (warm)	159 msec	530 msec

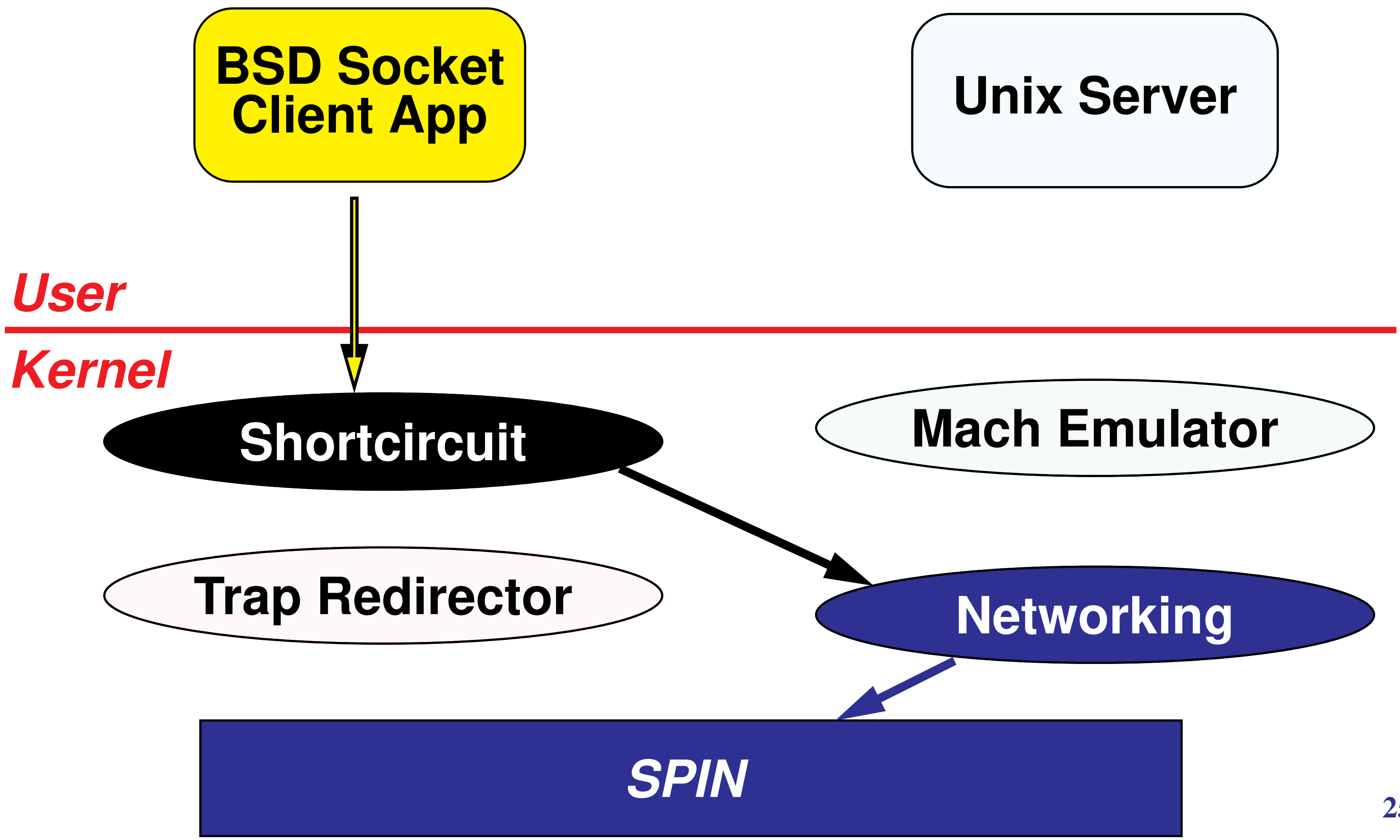
Improving Performance

Server path for UDP packet send



Shortcircuit

Shortcircuit path for UDP packet send



Shortcircuit Performance

Comparable to Digital Unix

Operation	**DEC Unix**	***SPIN* + UX**	***SPIN* + UX w/short**
UDP round trip	620 μsec	3.8 msec	660 μsec

Current Status

System boots, runs simple programs

Mach Emulator:

- **~50 system calls emulated**
- **~2500 lines of Modula-3 code**

Trap Redirector:

- **Completely implemented**
- **~500 lines of Modula-3 code**

Conclusion

Lessons learned

Emulation easy with *SPIN* extensions

Strategy for emulation: identify critical services first

Future Work

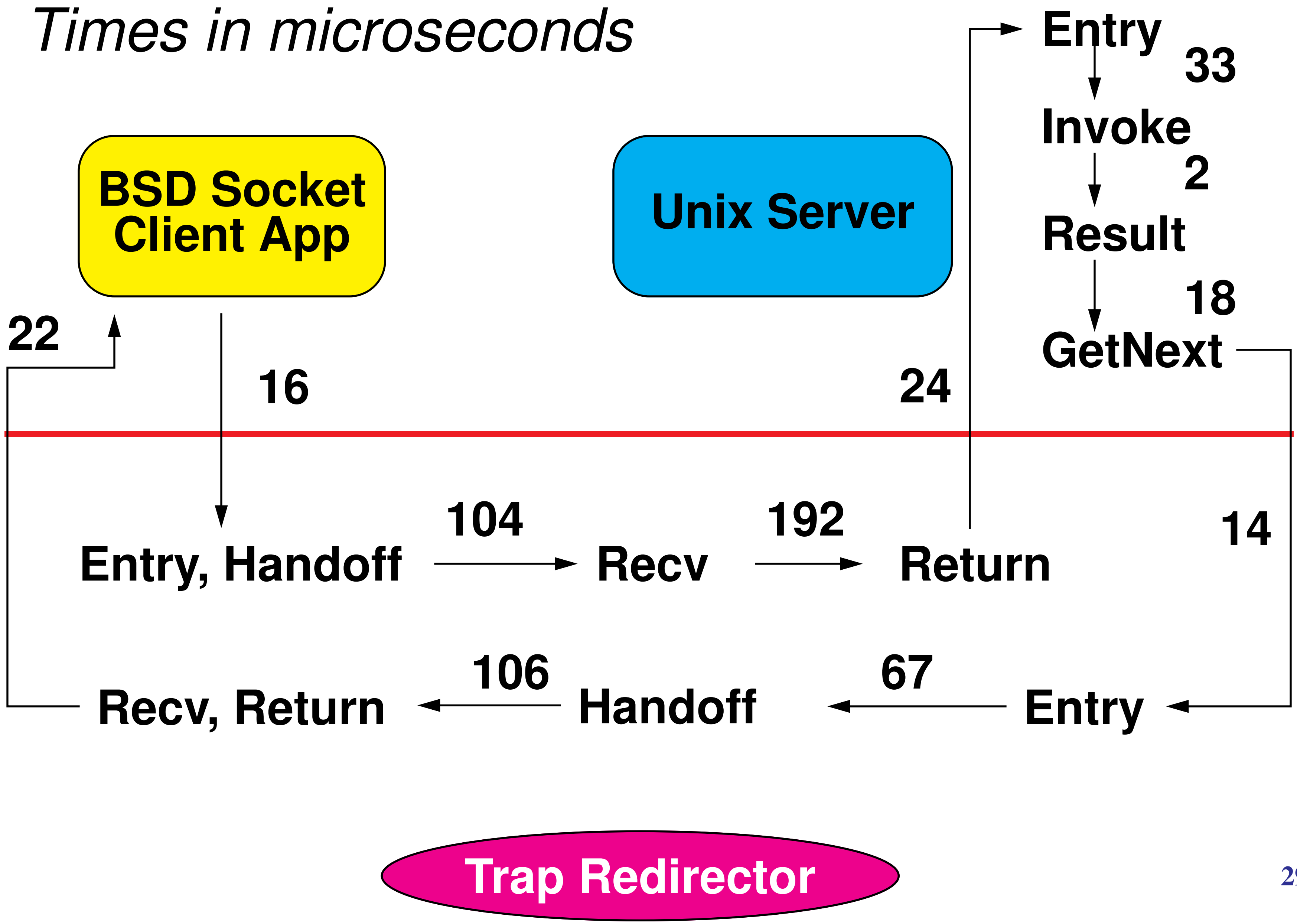
Integrate new Mach VM/Task support

Tune emulation to improve performance

Invoke shortcircuit for other applications

Trap Redirection

Times in microseconds



Mach + Unix Server

Operation	DEC Unix	Mach + UX	*SPIN* + UX
getpid()	5 µsec	1383 msec	1108 µsec
read() 1KB	44 µsec	1533 µsec	1207 µsec
ls -l (cold)	246 msec	8360 msec	2043 msec
ls -l (warm)	159 msec	7672 msec	530 msec