opensolaris

Chapter 2 Process, thread, and scheduling

——— *Solaris Multithreaded Process*

Zhao Xia
zhaoxia@os.pku.edu.cn

Outline

- **Introduction to Solaris Processes**
- Multithreaded Process Model
- Proc tools

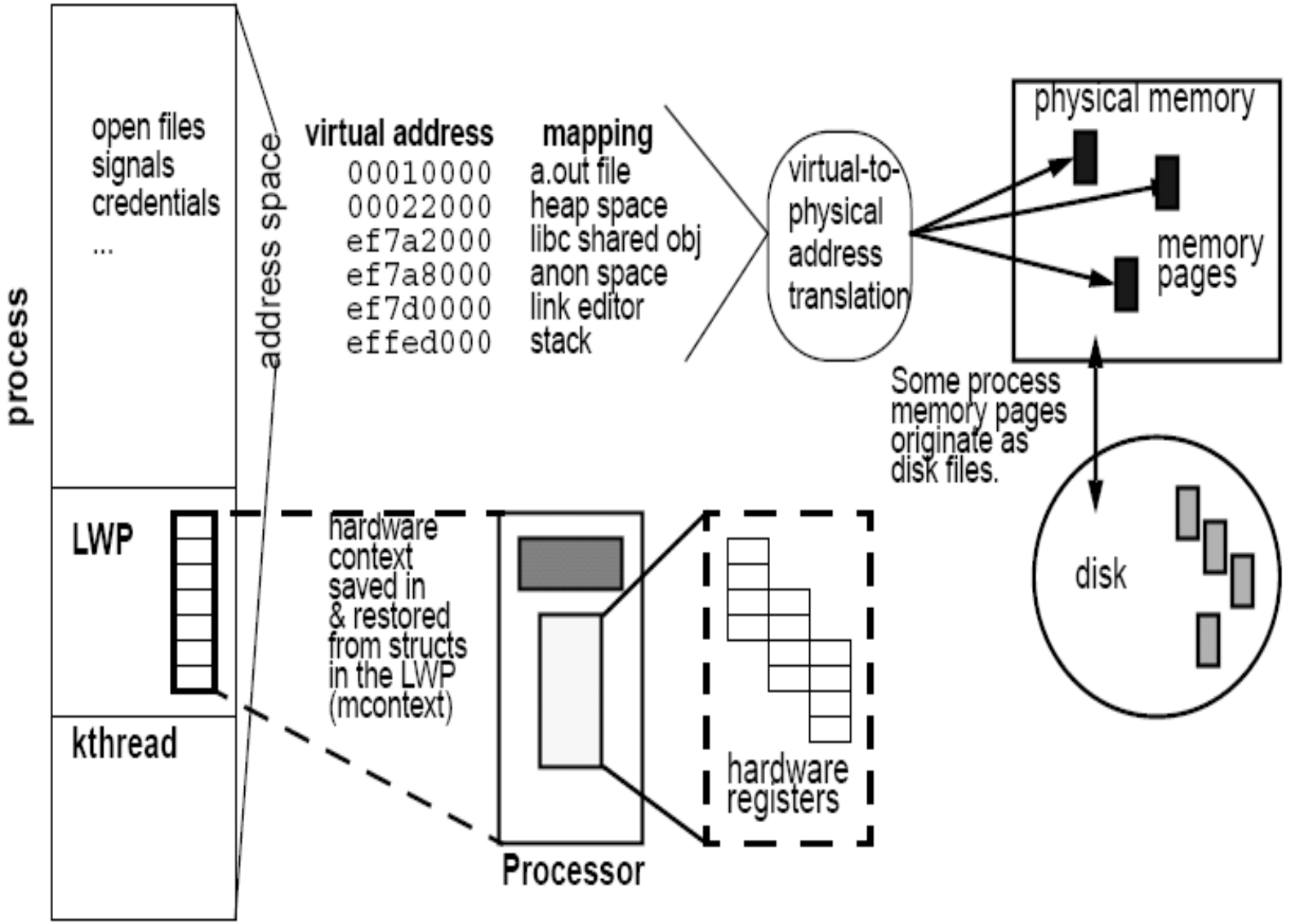
The Process Model

- **Solaris Kernel is Multi-threaded**
  - Kernel level threads (kthreads) are the unit of concurrency within the kernel
  - Scheduling, synchronization are kernel-level (kthread) concepts
- **Processes are a combination of state and one or more user threads**
  - Process threads are abstracted upon kernel threads
  - Single threaded processes have just one thread

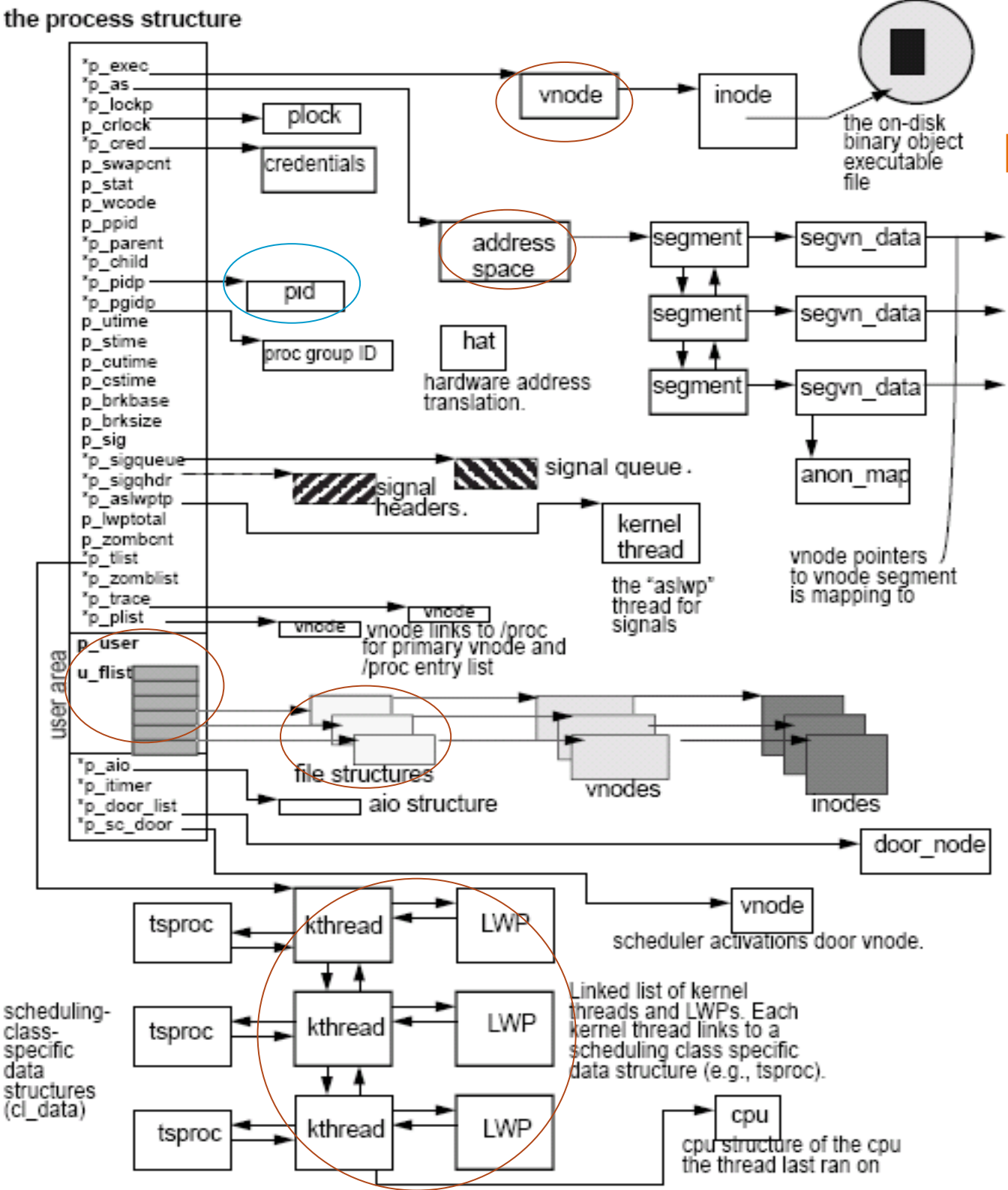
The Process Model

- **Processes**
  - All processes begin life as a program , a disk file (ELF object)
  - All processes have “state” or context that defines their execution environment - hardware & software context
- **Hardware context**
  - The processor state, which is CPU architecture dependent.
  - In general, the state of the hardware registers (general registers, privileged registers)
  - Maintained in the LWP
- **Software context**
  - Address space, credentials, open files, resource limits, etc – stuff shared by all the threads in a process
  - can be further divided into “hardware” context and “software” context

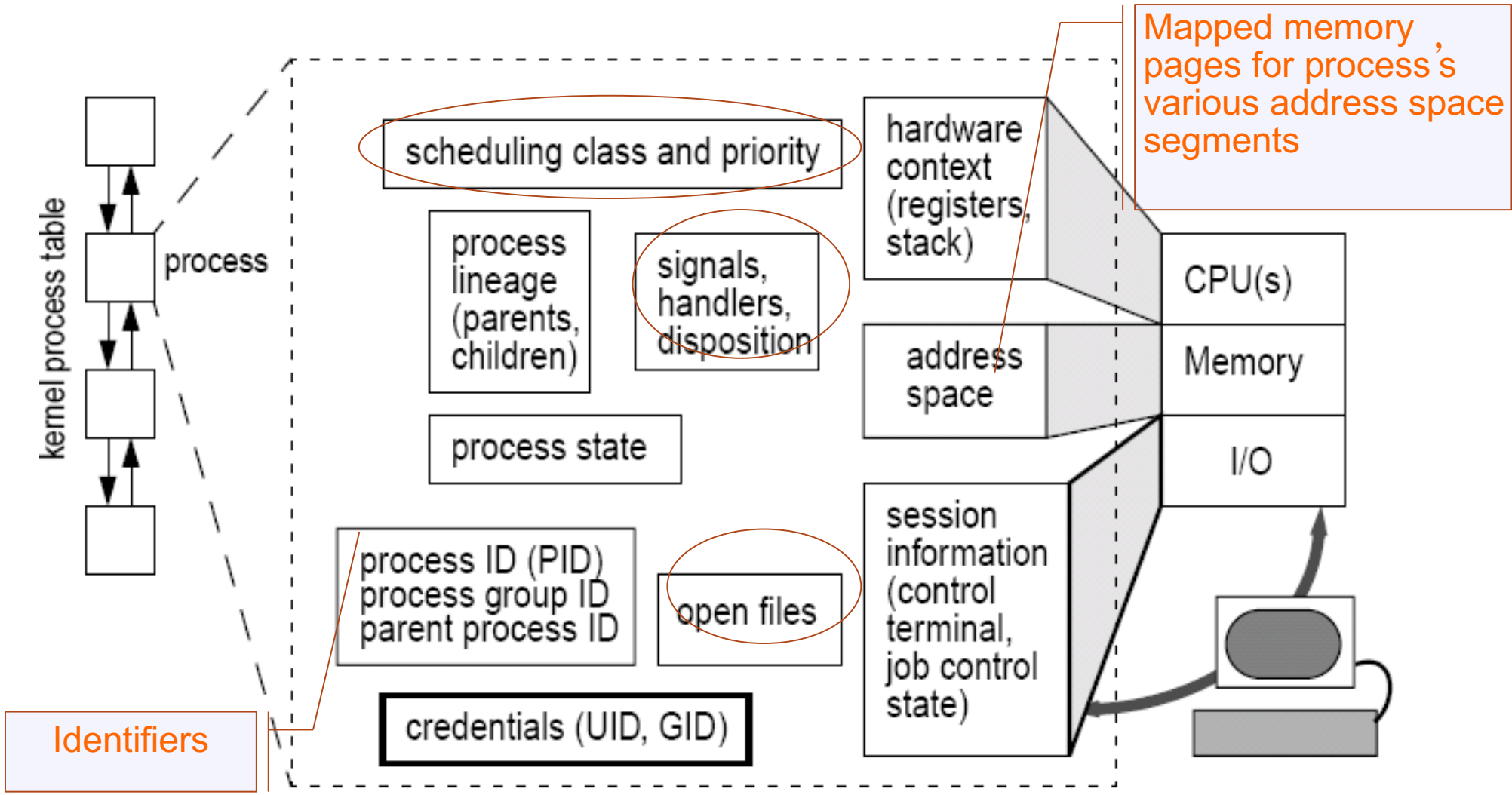
Conceptual View of a Process



Proc structure

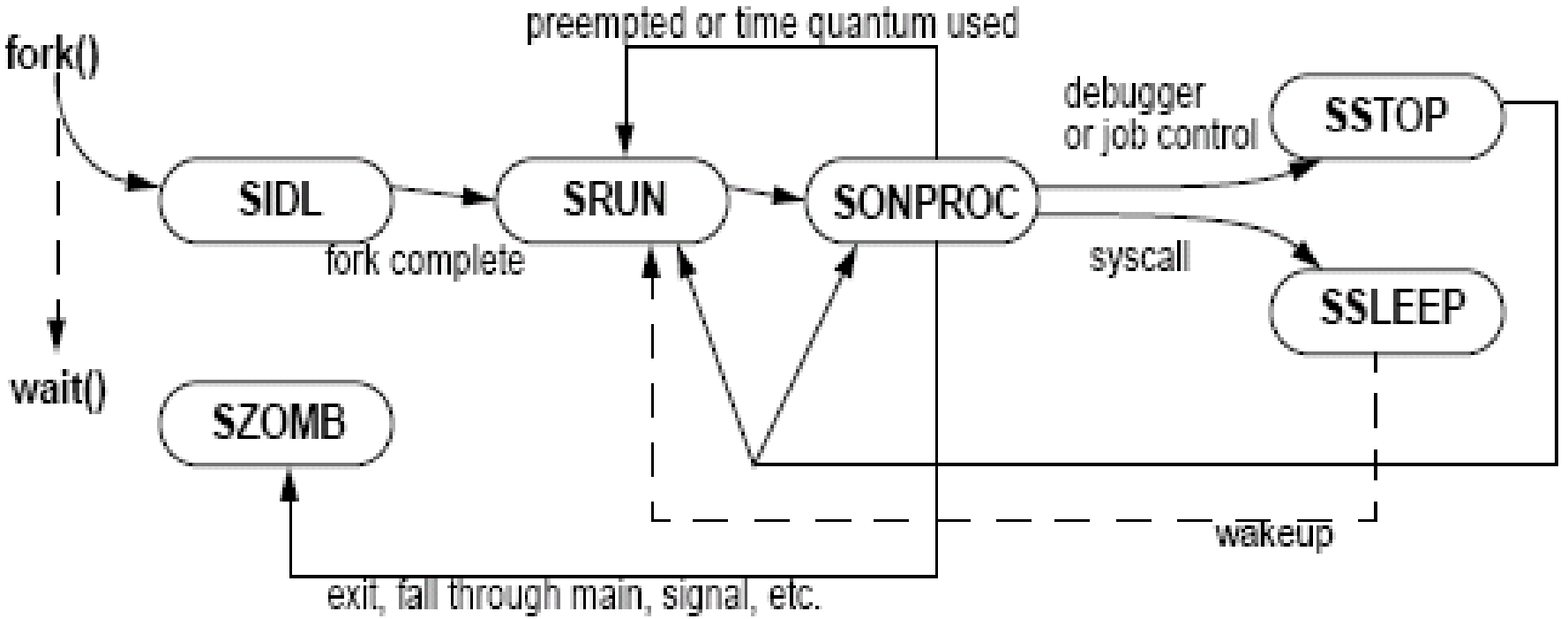


Process Execution Environment



Process State Diagram

- For the most part, for each process state, there is a corresponding kthread state
- Somewhat misleading - kthreads change state, not processes



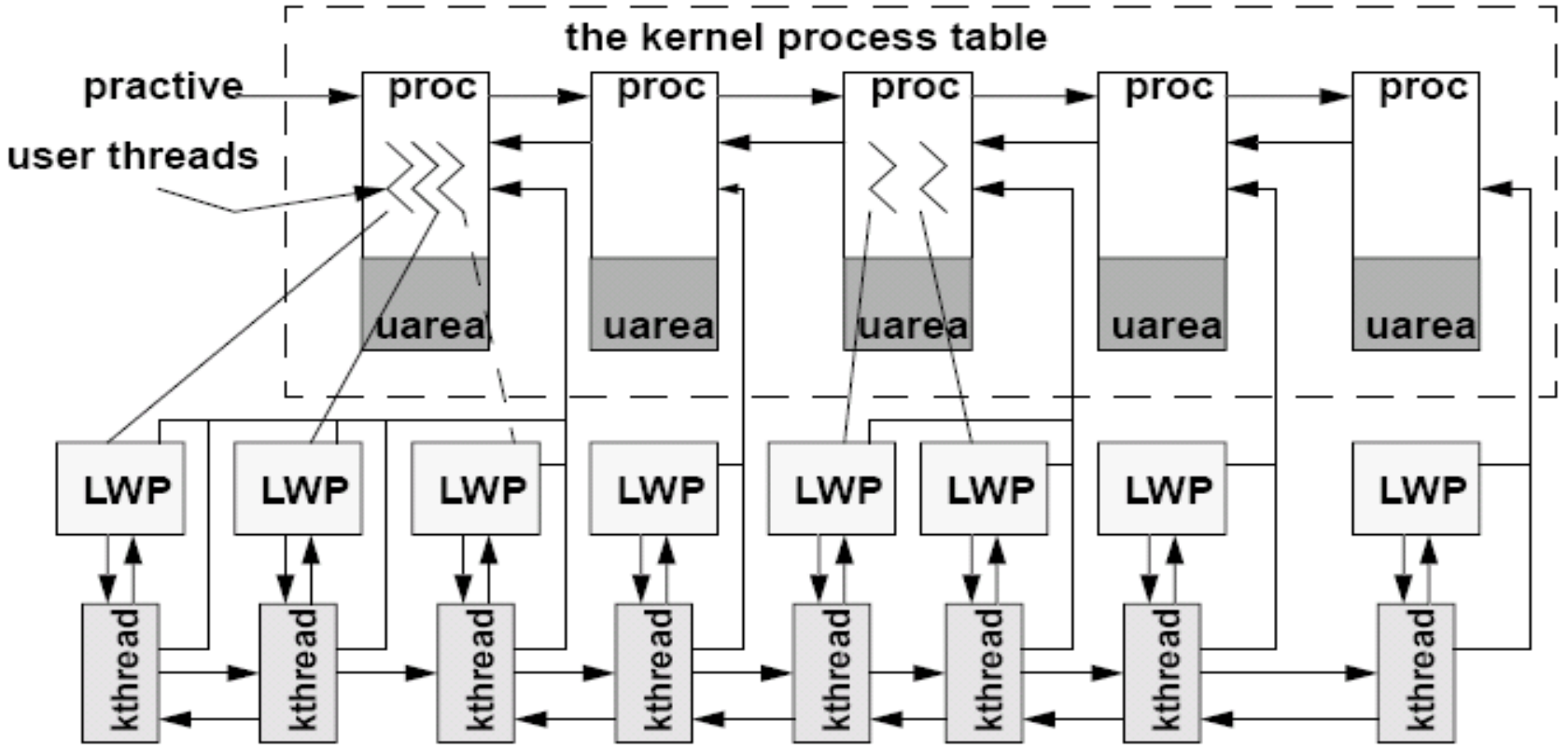
Process and Kernel Thread States

Process	Kernel Thread	Description
SIDL		State during fork(2) (creation).
SRUN	TS_RUN	Runnable.
SONPROC	TS_ONPROC	Running on a processor.
SSLEEP	TS_SLEEP	Sleeping (blocked).
SSTOP	TS_STOPPED	Stopped.
SZOMB	TS_ZOMB	Kthread/process has terminated.
	TS_FREE	Thread is waiting to be reaped.

- Kthread creation is not flagged as a distinct state - they go right to TS_RUN
- Kthread structures are flagged as TS_FREE when the proc or kthread/LWP is terminated
  - This allows the kernel to maintain a cache of free kthread/LWP structures

Process, LWP, and Kthread Linkage

- Kernel maintains system-wide linked lists of processes, LWPs and kthreads
- Relationship links maintained at every level



Solaris Thread Concepts

- **Kernel Threads**
  - Kernel's unit of concurrency
- **LWP**
  - Implemented to allow concurrent system calls from a single process
  - Without LWPs, user threads would contend at system call
- **User Threads**
  - The thread abstraction of the userland programming model

The Lightweight Process (LWP)

- the attribute of a LWP
  - Resource utilization counters and microstate accounting information
  - The sum total of all LWPs resource usage is stored in the process
  - Most of the LWP structure members exist to support system calls and to maintain hardware context information
  - An LWP blocked on a system call does not cause the entire process to block

The kernel thread (KThread)

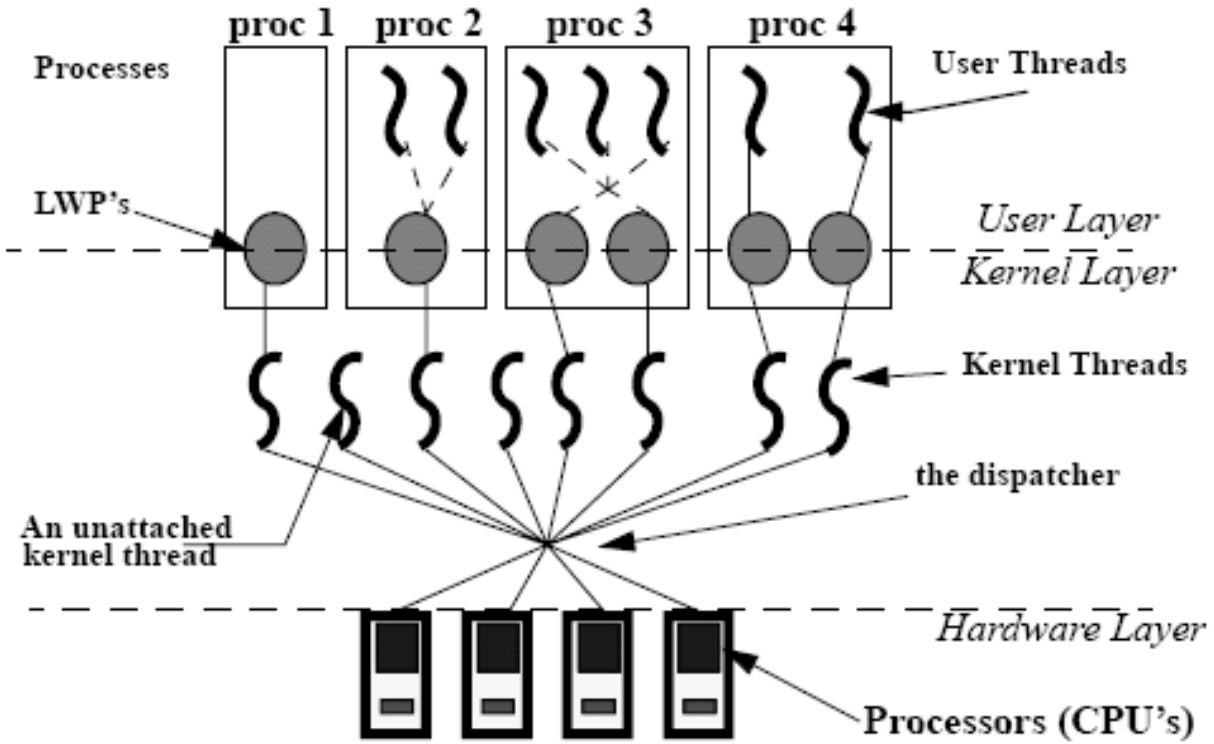
- Features
  - the entity that actually gets put on a dispatch queue and scheduled
  - scheduling class and priority is assigned to a kthread , not the process
  - kthread associated with the LWP, has a priority and scheduling class

Outline

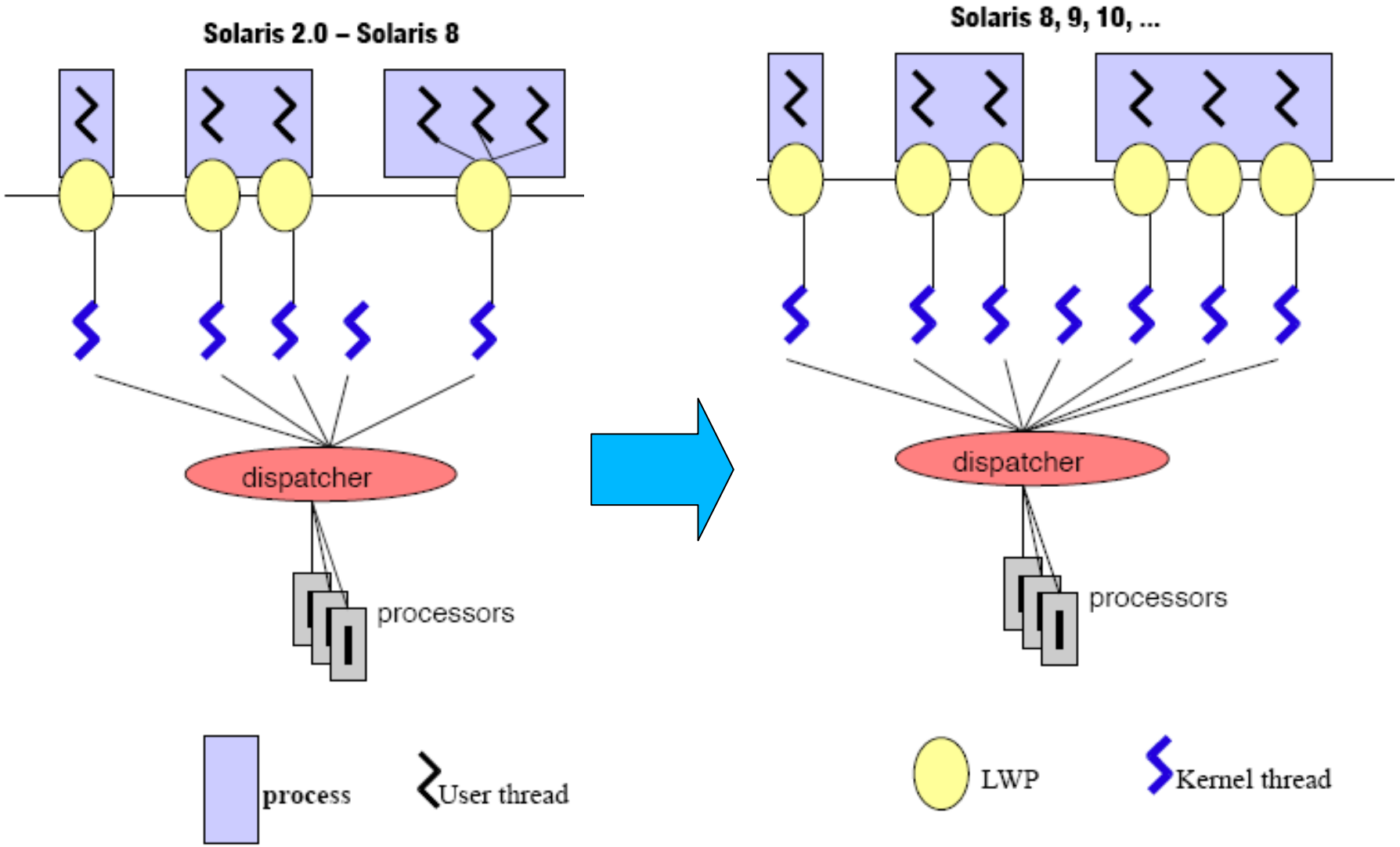
- Introduction to Solaris Processes
- **Multithreaded Process Model**
- Proc tool

Multithreaded Process Model

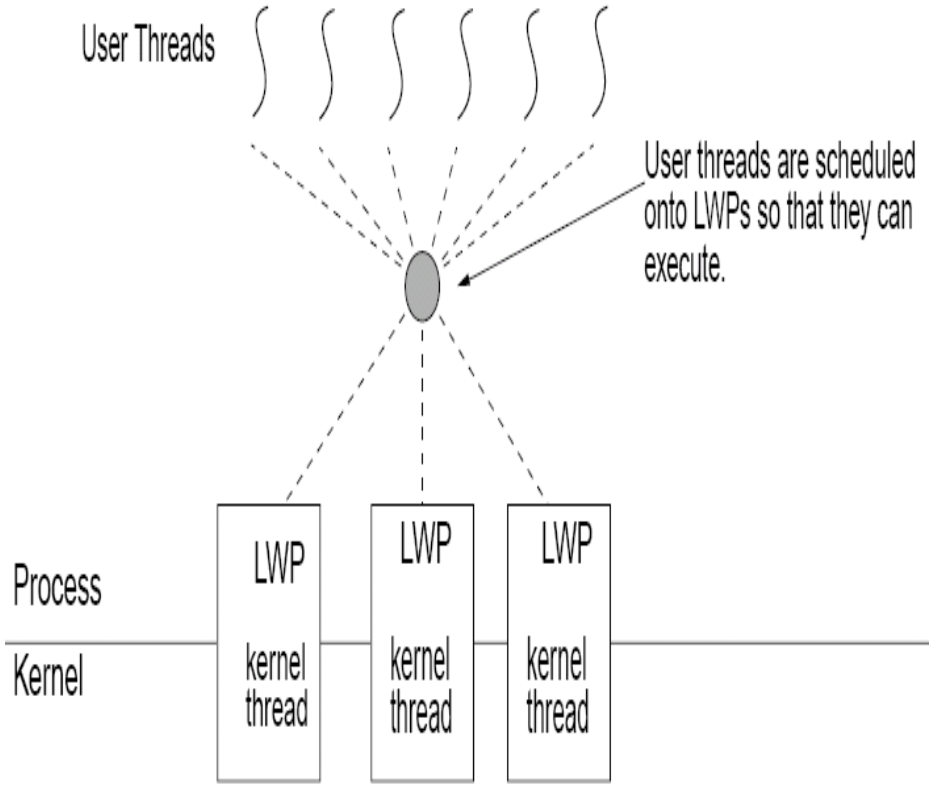
- Processes can have varying numbers of user
- threads, LWPs and kernel threads



The Multithreading Revolution

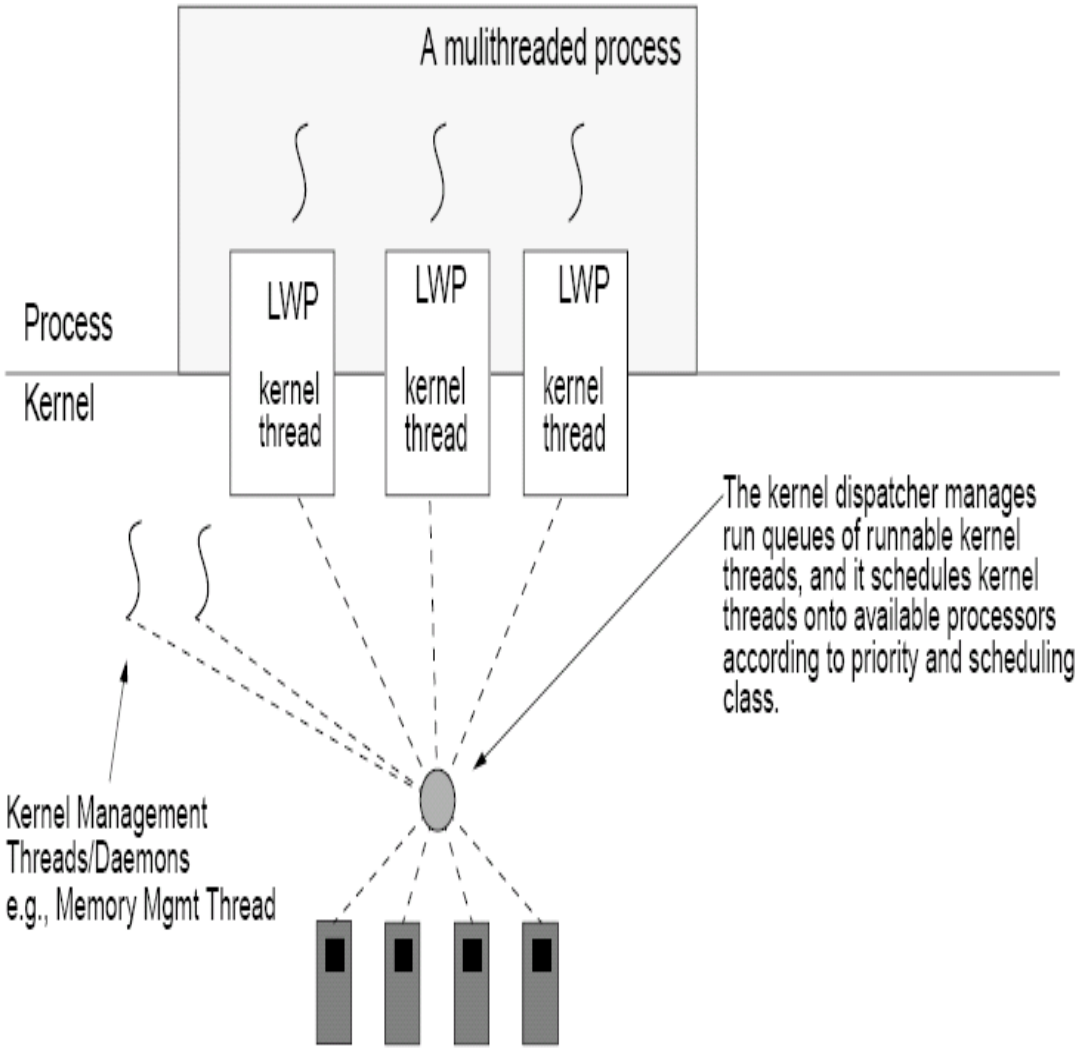


Multi-level Thread Model (M:N thread model)



- **Pros:**
  - Fast user thread create and destroy
  - No system call required for synchronization
  - Fast context-switching
- **Cons:**
  - Complex, and tricky programming model
  - Signal delivery

Single-level Thread Model (1:1 Thread Model)



- Every user level thread has an lwp, and a kthread
- Kernel level scheduling
- More expensive thread create/destroy, synchronization
- More responsive scheduling, synchronization

Outline

- Introduction to Solaris Processes
- Multithreaded Process Model
- **Proc tool**

proc(1) Debugging Utilities

- Solaris provides a powerful and unrivaled set of debugging and observation utilities – fully documented in the proc(1) man page.
- Solaris 10 provides substantial improvements for two of these tools, and a new directory in /proc
  - pmap(1)
  - pfiles(1)
  - /proc/*pid*/path

pmap(1)

- Shows information about the address space of a process.

```
example$ pmap 121969
121969: ./stacks
00010000       8K r-x-- /tmp/stacks
00020000       8K rwx-- /tmp/stacks
FEFFA000       8K rwx-R    [ stack tid=4 ]
FF0FA000       8K rwx-R    [ stack tid=3 ]
FF1FA000       8K rwx-R    [ stack tid=2 ]
FF220000      64K rw---    [ altstack tid=4 ]
FF240000     112K rw---    [ anon ]
...
FFBFA000      24K rwx--    [ stack ]
 total      1400K
```

Location and size of every thread stack, identified by tid

Alternate signal stacks shown for each thread that has one (see `sigaltstack(2)`)

pmap(1)

- May also be used on core files
- Segments not present in the core files are marked with a '*'

```
example$ pmap core
core 'core' of 404654: ksh
00010000     200K r-x--* /usr/bin/ksh
00052000       8K rwx-- /usr/bin/ksh
00054000    1160K rwx--    [ heap ]
FF200000     856K r-x--* /lib/libc.so.1
FF2E6000      32K rwx-- /lib/libc.so.1
FF2EE000       8K rwx-- /lib/libc.so.1
..
```

Reference

- Jim Mauro, Richard McDougall, Solaris Internals-Core Kernel Components, Sun Microsystems Press, 2000
- Sun, Multithreading in the Solaris Operating Environment, A Technical White Paper,2002
- Max Bruning, Threading Model In Solaris, Training lectures,2005
- Solaris internals and performance management, Richard McDougall, 2002

End

- Last.first@Sun.COM