grupo de usuários opensolaris Porto Alegre

Kernel OpenSolaris

- Introduction
- Processes, threads and LWPs
- Scheduler and dispatcher
- Virtual Memory
- Building the kernel
- Participating

Rafael Vanoni
rafael.vanoni@sun.com
blogs.sun.com/rv

Introduction

What is OpenSolaris ?

- open source operating system based on Solaris (UNIX developed by Sun Microsystems)
- launched on June 2005
- licensed under the Common Development and Distribution License (CDDL)
- open development
- distributions: Solaris Express Developer Edition, NexentaOS, BeleniX, SchiliX, MartuX, OpenSolaris Indiana, ..
- uses Mercurial as its versioning system
- *Source Browser* allows anyone to easily navigate the code

Introduction (2/5)

What is OpenSolaris made of ?

- ~41k files, ~11m lines of code
- windowing systems (Xorg): GNOME, CDE, KDE*
- file systems: UFS, ZFS, .. (packages for accesing extNfs, FATs, ..)
- Development (SXDE): Sun Studio compilers, gcc, JDK, Python, Perl, PostgreSQL, Java DB, ..

Differentials

- DTrace
- ZFS
- MDB
- Zones - Containers

Introduction (3/5)

opensolaris

Kernel Architecture

- System call interface: allows processes to access kernel facilities
- Process execution and scheduling: creation, execution, management and termination of processes
- Memory management: the virtual memory system manages mapping physical memory to user processes and the kernel
- Resource management: allows the allocation of specific hardware resources to application
- File systems: OpenSolaris implements a virtual file system (VFS) framework under which different files systems can be configured at the same time (disk-based, network, pseudo)

Introduction (4/5)

opensolaris

Kernel Architecture

- I/O and device management
- Kernel facilities: clock interruptions, timers, synchronization primitives, and loadable module support
- Networking: IPv4 and Ipv6 support, socket-based interfaces and STREAMS

Introduction (5/5)

opensolaris

Code layout

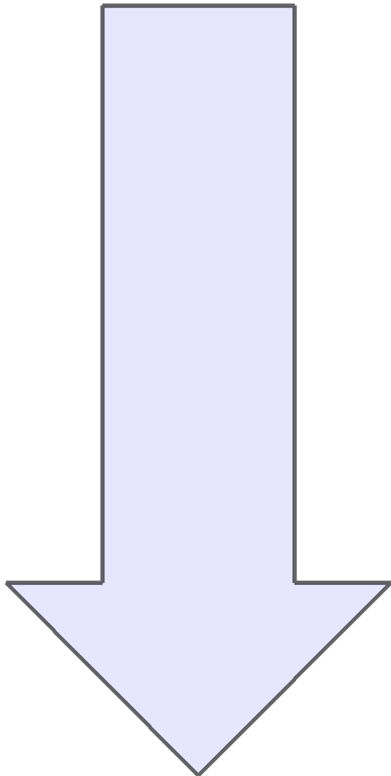
/src/uts	Unix Time Sharing (kernel)
/src/uts/common	architecture independent code
/src/uts/i86pc	x86 architecture
/src/uts/sparc	SPARC architecture

For this presentation

/src/uts/[common,i86pc]/sys	kernel headers
/src/uts/[common,i86pc]/os	implementations
/src/uts/[common,i86pc]/vm	virtual memory
/src/uts/common/disp	dispatcher and scheduler

Processes and Threads

Model and Implementation



Process: the executable form of a program

User Thread: a user level thread state within a process

Lightweight Process (LWP): the kernel visible execution context for a user thread

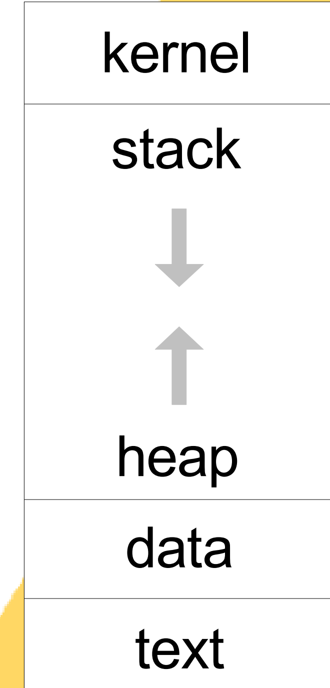
Kernel Thread: the object that gets scheduled and executed

opensolaris

Processes

Concepts

- combination of a process and its current state of execution (an instance of a program)
- has an address space with text, data, heap and stack segments, and a shared kernel segment
- has one or more execution flow (thread) that share the process' address space (except the stack)
- declared in uts/common/sys/proc.h
- type proct_t, contains many fields/structures
- implementations in uts/common/os/proc.c



Processes (2/3)

opensolaris

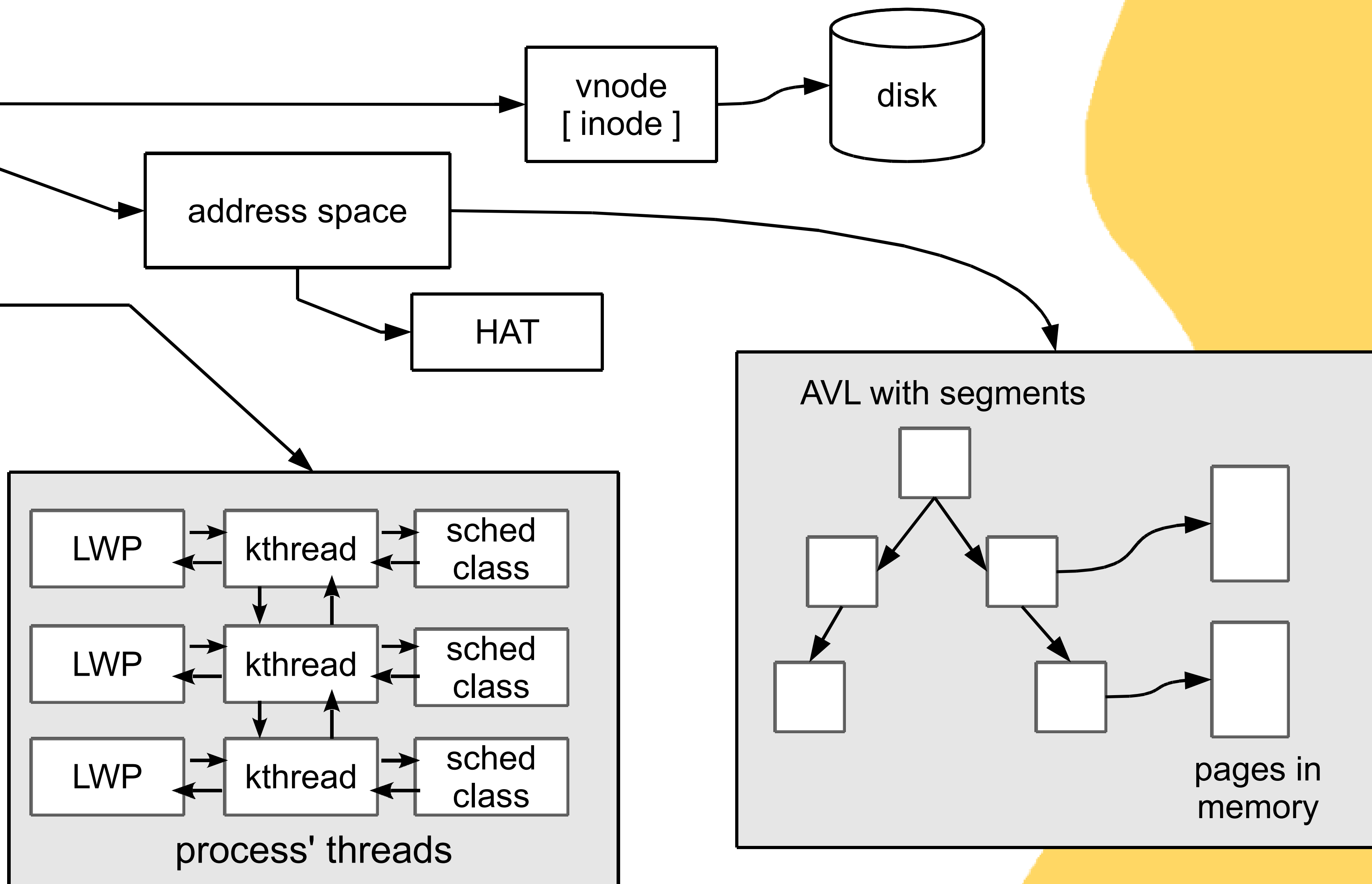
uts/common/sys/proc.h

- struct proc has over 100 fields
- locks to manage concurrent accesses
- pointer to executable
- pointer to address space
- points to different process and lists (parent, children, next, previous, ..)
- process identifier (pid)
- structures for system accounting purposes
- struct sys/user.h (memory usage, current dir, ..)

Processes(3/3)

opensolaris

uts/common/sys/proc.h



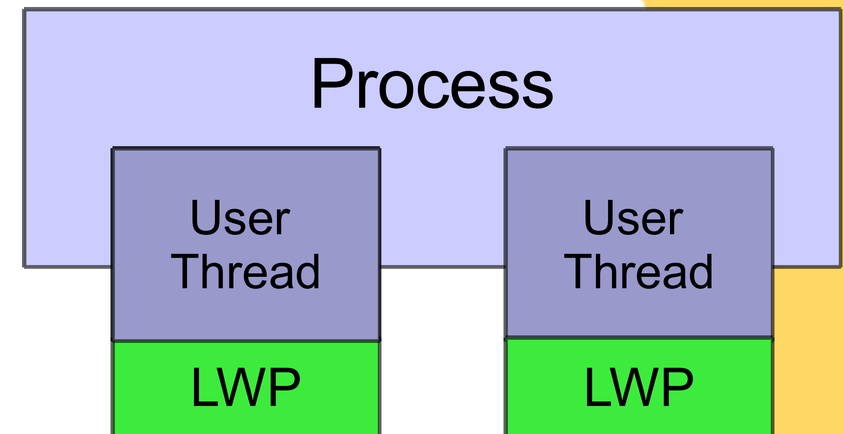
Threads

opensolaris

Light Weight Process (LWP)

- each user thread has a LWP structure
- the LWP stores the user thread' state
- declared in uts/common/sys/klwp.h
- type klwp_t

struct _klwp
pcb lwp_pcb
void *lwp_regs
char lwp_state
kthread *lwp_thread
proc *lwp_proc



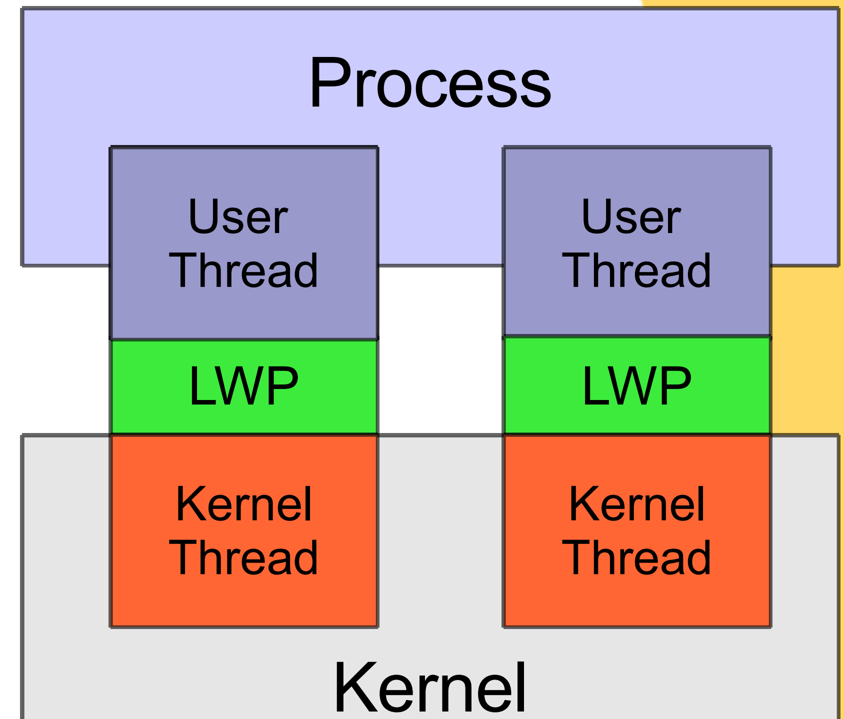
Threads (2/5)

opensolaris

Kernel Threads

- each LWP is linked to a kernel thread
- basic scheduling and execution unit in the system
- declared in uts/common/sys/thread.h
- type kthread_t

struct _kthread
_kthread *t_link
caddr_t t_stk
void (*t_startpc)(void)
uint_t t_state
pri_t t_pri
caddr_t t_swap
cpu *t_cpu
klwp_t *t_lwp



Threads (3/5)

opensolaris

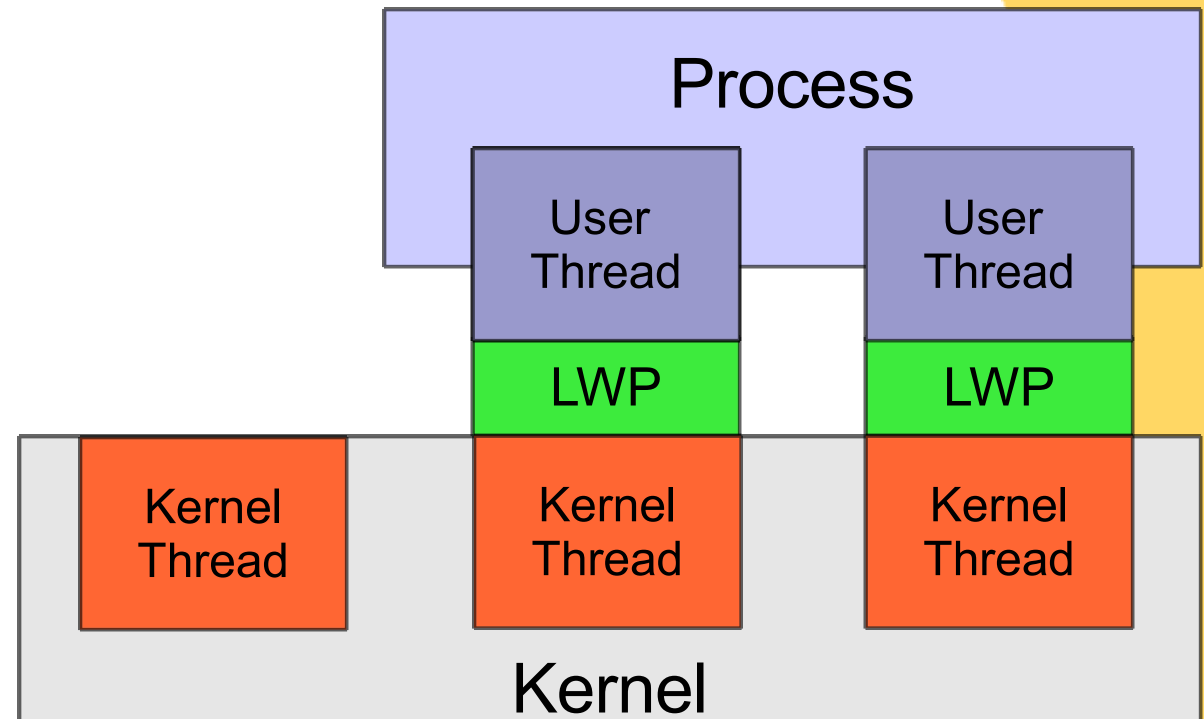
Kernel Threads

- not every kernel thread is linked to a user thread
- a classic example is the idle() thread

Implementation

- src/uts/common/disp/thread.c

```
...
void        thread_init()
kthread*    thread_create()
void        thread_exit()
void        thread_join()
void        thread_free()
...
```



Threads (4/5)

opensolaris

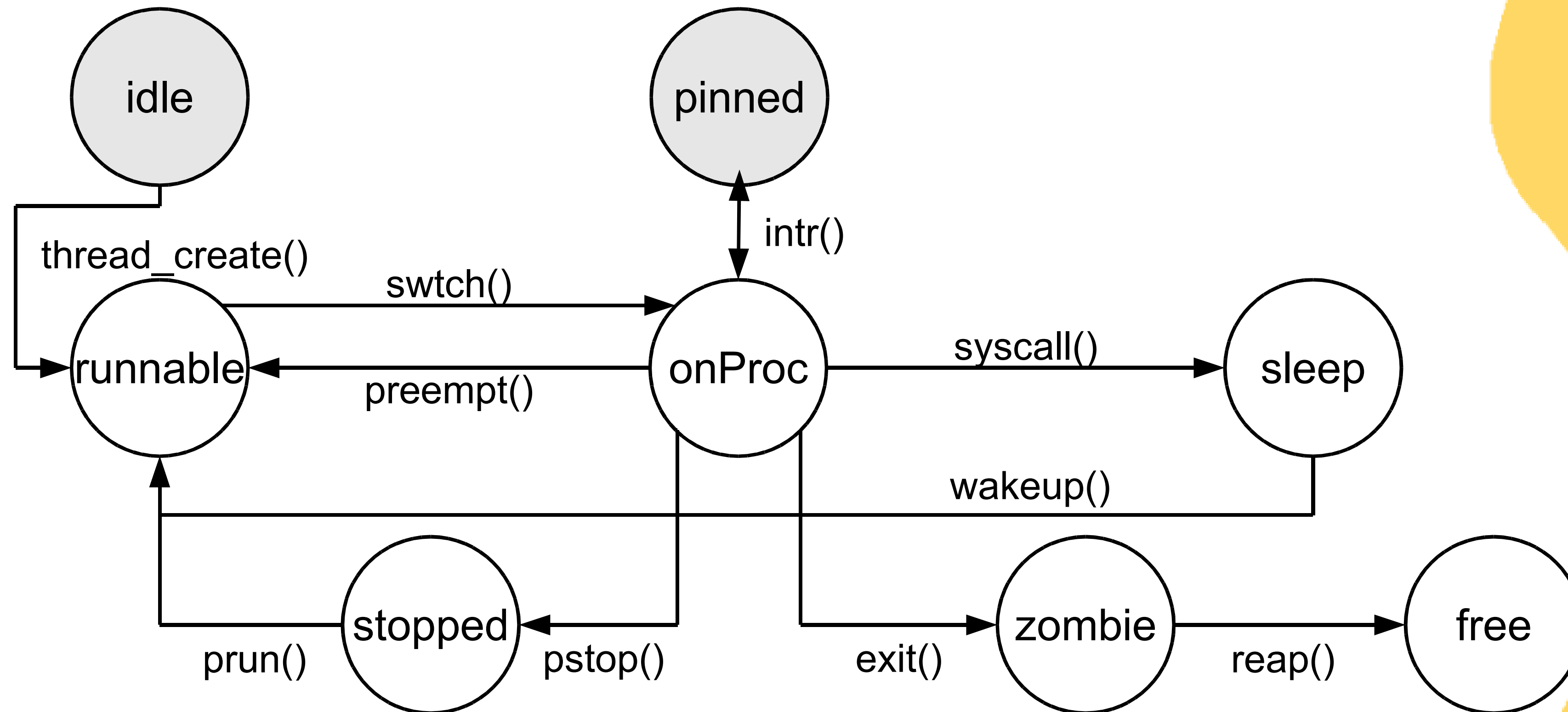
Thread states

- *Runnable*: thread is ready, waiting to be executed
- *On Proc*: executing on a processor
- *Sleeping*: blocked waiting for an event (for instance, I/O)
- *Stopped*: suspended
- *Zombie*: terminated but still occupying memory space
- *Free*: terminated and had it's memory space reaped

Threads (5/5)

opensolaris

State diagram



Scheduler and dispatcher

opensolaris

Concepts

- the scheduler is the kernel subsystem that manages thread queues
- the dispatcher is the subsystem that applies scheduling decisions, he decides **where** a thread will be executed and is in charge of the **context switch**
- preemptive, for both user and kernel processes
- uses a global priority model to choose which thread should be executed next (0-170)
- has different **scheduling classes** that divide the global range
- keeps the topmost values for interruptions

Scheduler and dispatcher (2/10)

opensolaris

Responsibilities

- queue management
- thread selection
- processor selection
- context switch

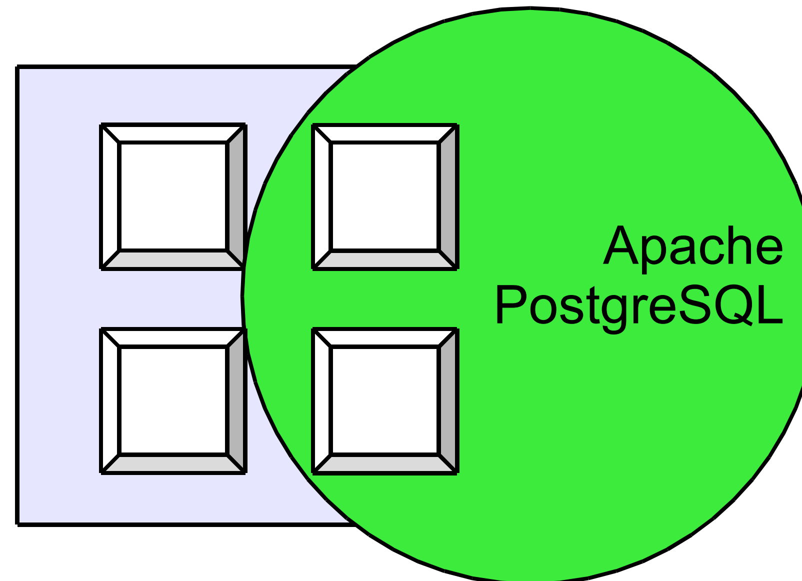
Scheduling decisions

- priority scheme
- resource management parameters
- system architecture

Scheduler and dispatcher (3/10)

Resource management parameters

- different OpenSolaris technologies to manage hardware resources
  - Processor Binding: binding *processes* to *processors*
  - Processor Sets: creation of processor sets (subsets of the total amount of processors) and binding of processes to these sets



Scheduler and dispatcher (4/10)

Resource management parameters

- Resource pools: stateful *Processor Sets*, basically
- Zones: a virtualized execution environment. Binding a Resource Pool to a Zone is a generic and more flexible way of binding processes to processor sets (Containers)

System architecture

- modifications to the dispatcher because of system characteristics
  - NUMA architectures (*Non Uniform Memory Access*)
  - CMT processors (*Chip Multi threading*)
- for example, exploring *cache warmth*

Scheduler and dispatcher(5/10)

opensolaris

Scheduling Classes

- Time Share (TS) src/uts/common/disp/ts.c
  - priority range 0-59
  - priorities adjustment based on how long threads utilize processor resources
  - default class
- Interactive (IA) src/uts/common/disp/ia.c
  - priority range 0-59
  - same as TS, but *boosts* interactive apps (Xorg)

Scheduler and dispatcher (6/10)

opensolaris

Scheduling Classes

- Fair Share (FSS) src/uts/common/disp/fss.c
  - priority range 0-59
  - processor resources divided in shares, which are assigned to processes by OpenSolaris' resource management framework
- Fixed Priority (FX) src/uts/common/disp/fx.c
  - priority range 0-60
  - static priorities

Scheduler and dispatcher (7/10)

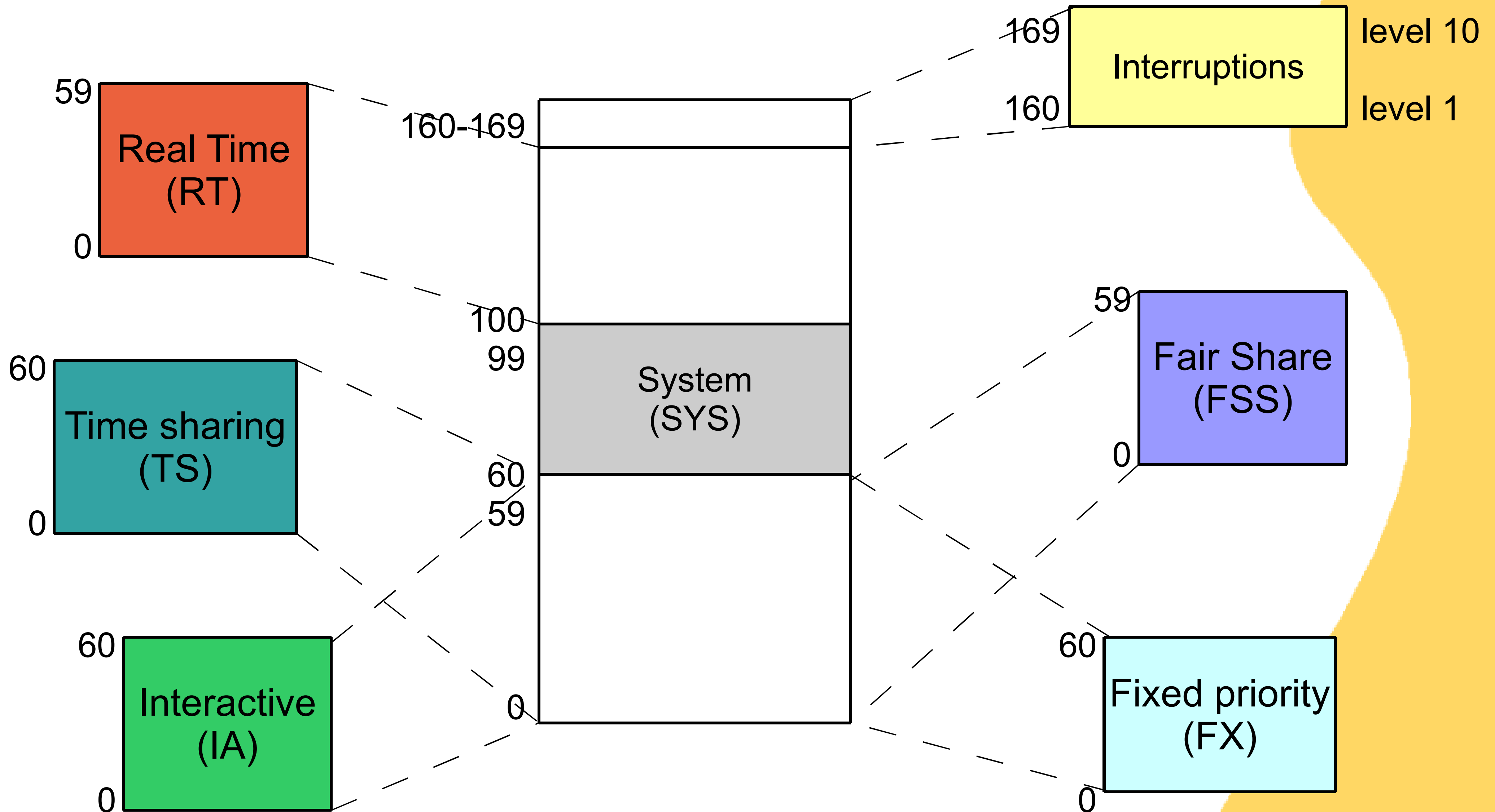
opensolaris

Scheduling Classes

- System (SYS) src/uts/common/disp/sysclass.c
  - priority range 60-99
  - used by system threads
- Real Time (RT) src/uts/common/disp/rt.c
  - priority range 100-159
  - can preempt the kernel

Scheduler and dispatcher(8/10)

Scheduling Classes



Scheduler and dispatcher (9/10)

opensolaris

Changing states

- threads enter the dispatcher when changing states or when are causing threads to change state

[sleep, onProc] > [runnable]

thread enters the dispatcher which chooses a CPU and places it on that CPU's queue

[runnable] > [onProc]

thread enters the dispatcher and the highest priority thread is dequeued and context switched to

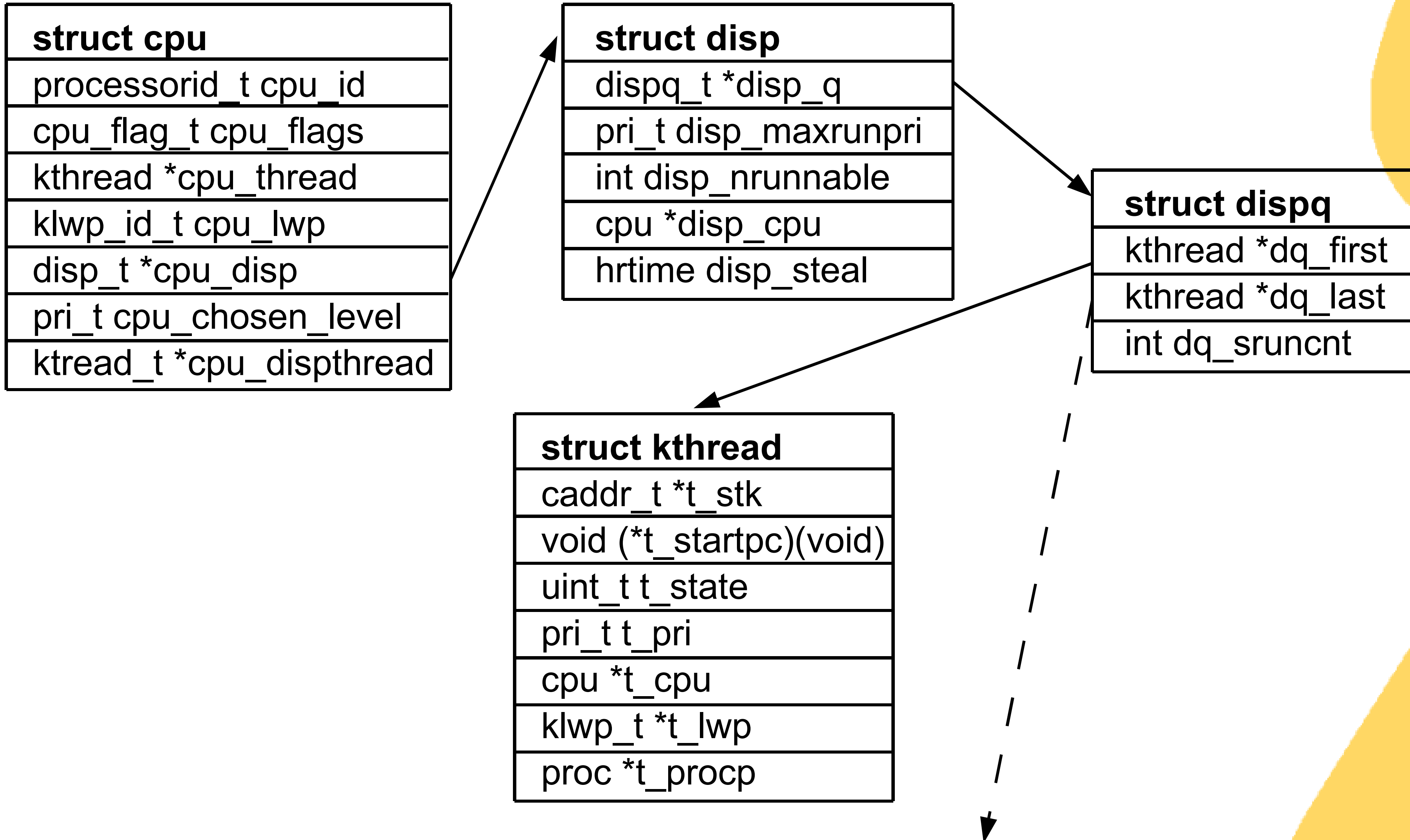
[onProc] > [sleep]

thread blocks itself on the object's lock queue and puts the highest priority thread on the CPU's queue to execution

Scheduler and dispatcher(10/10)

opensolaris

Structures (src/uts/common/sys/disp.h)



Virtual Memory

opensolaris

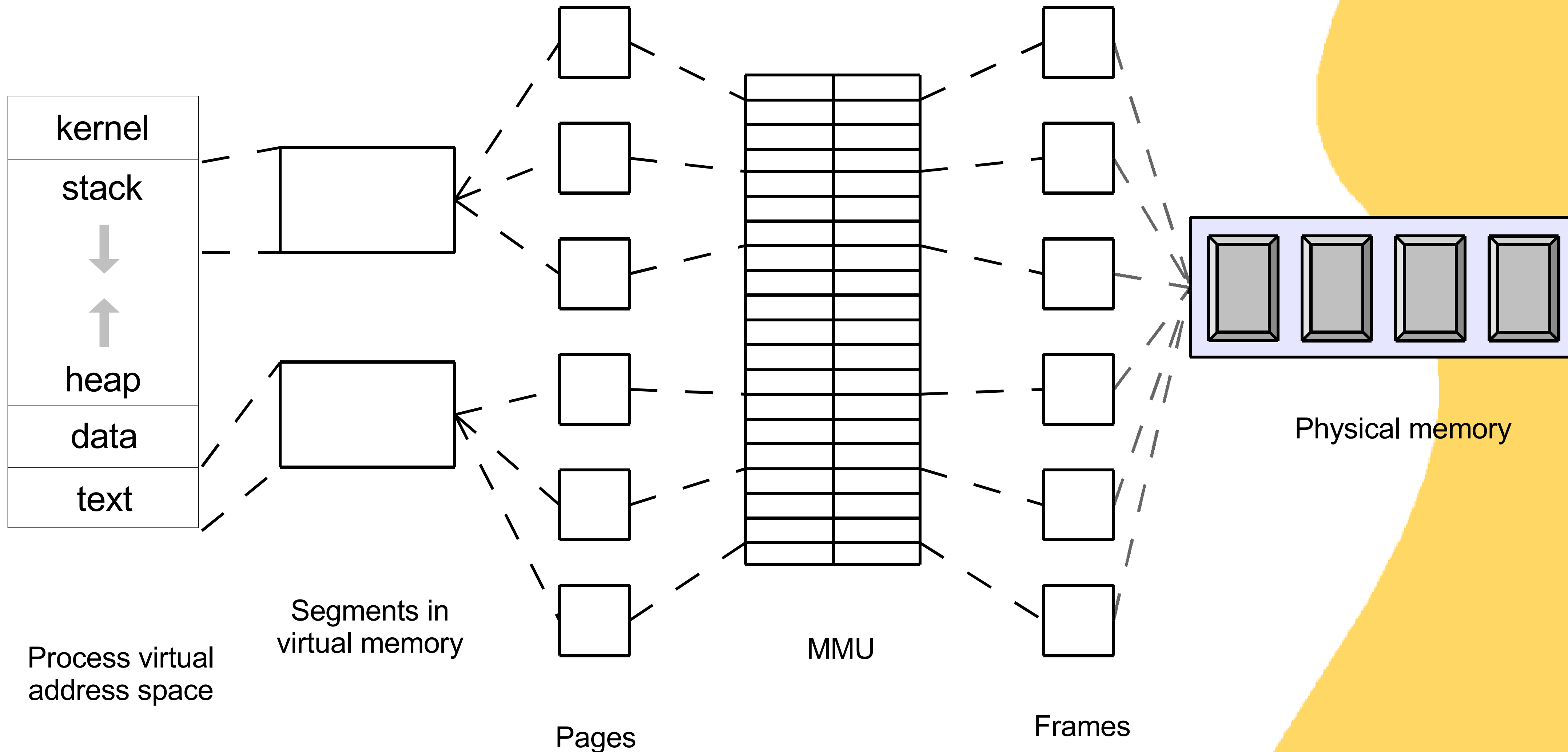
Concepts

- main goals:
  - free processes of memory size restrictions
  - allow a higher level of multiprogramming by only keeping a process' *resident set* in memory
- paging:
  - divides physical memory process space in fixed size chunk, called *frames* and *pages*
  - logical addresses are translated to physical addresses by the MMU through *page tables*
- implemented in layers: hardware independent and hardware dependent (HAT)

Virtual Memory(2/7)

opensolaris

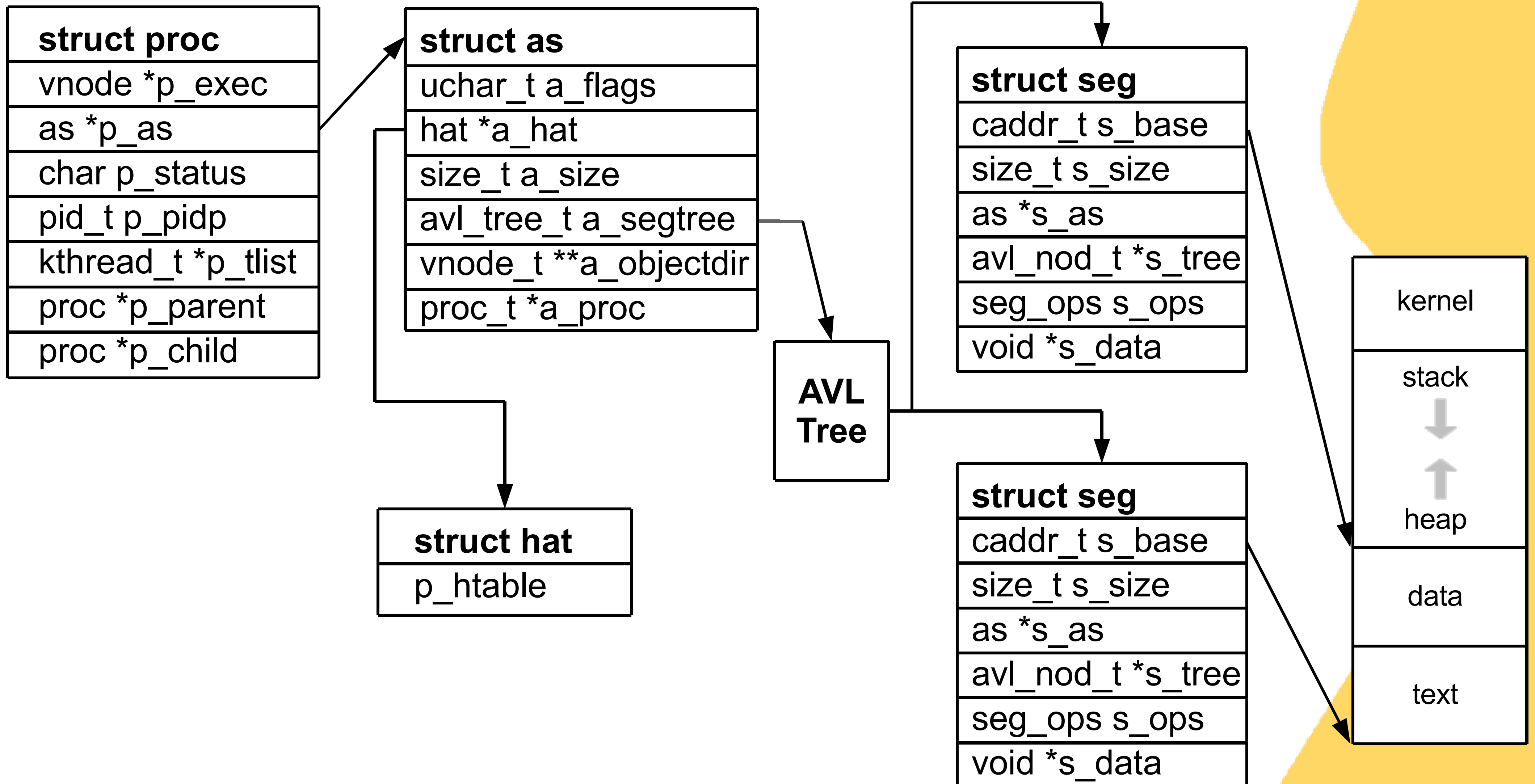
Address spaces, segments and pages



Virtual Memory (3/7)

opensolaris

Structures



Virtual Memory (4/7)

opensolaris

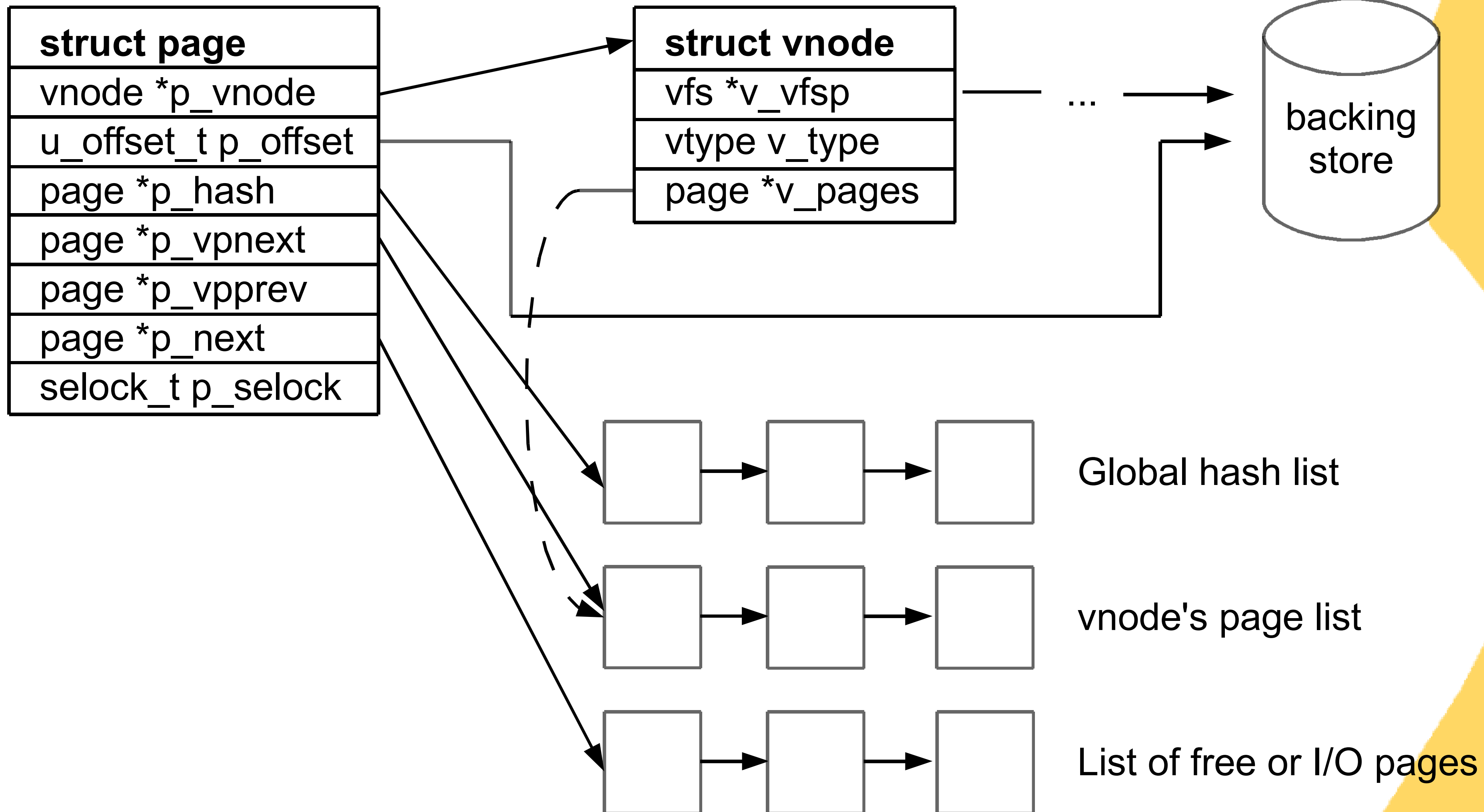
Pages

- basic memory unit
- Every active page (not free) is a mapping between a file (vnode) and memory
- pages are identified by its vnode and an offset (vnode/offset pair)
- such pair is the page's *backing store*
- reusability: the pair is also used to map a swap file or a file cache
- the mapping between a physical page and its virtual space is done by the HAT layer
- a global hash list of pages contains pointers to page lists, and is indexed by a hash function based on the vnode/offset pair

Virtual Memory (5/7)

opensolaris

Structures

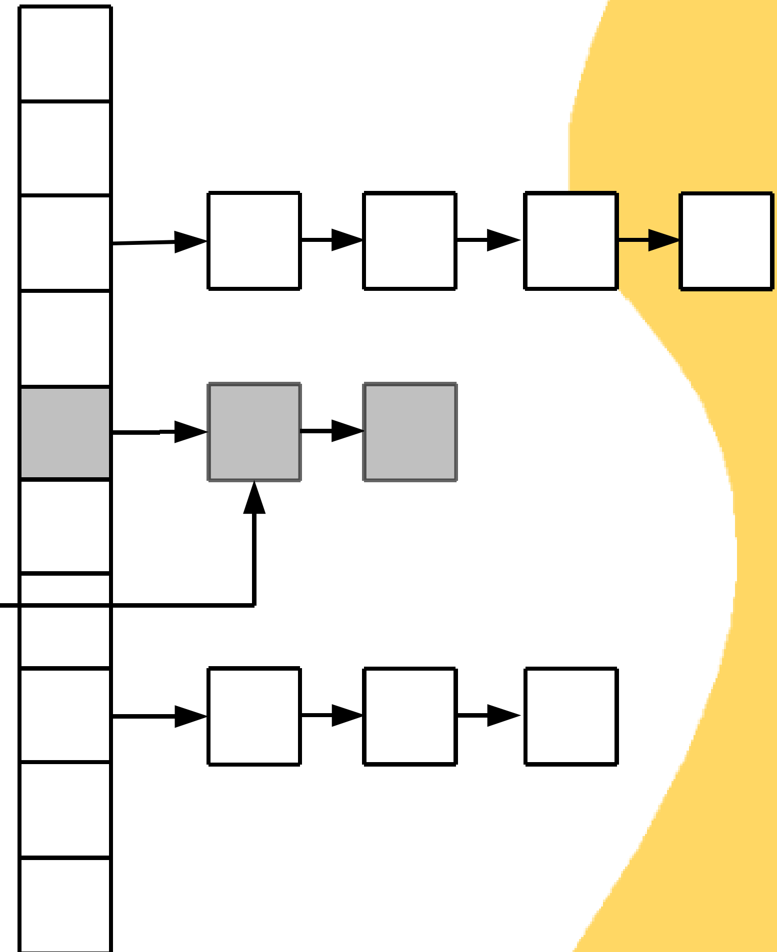


Virtual Memory(6/7)

opensolaris

Finding a page

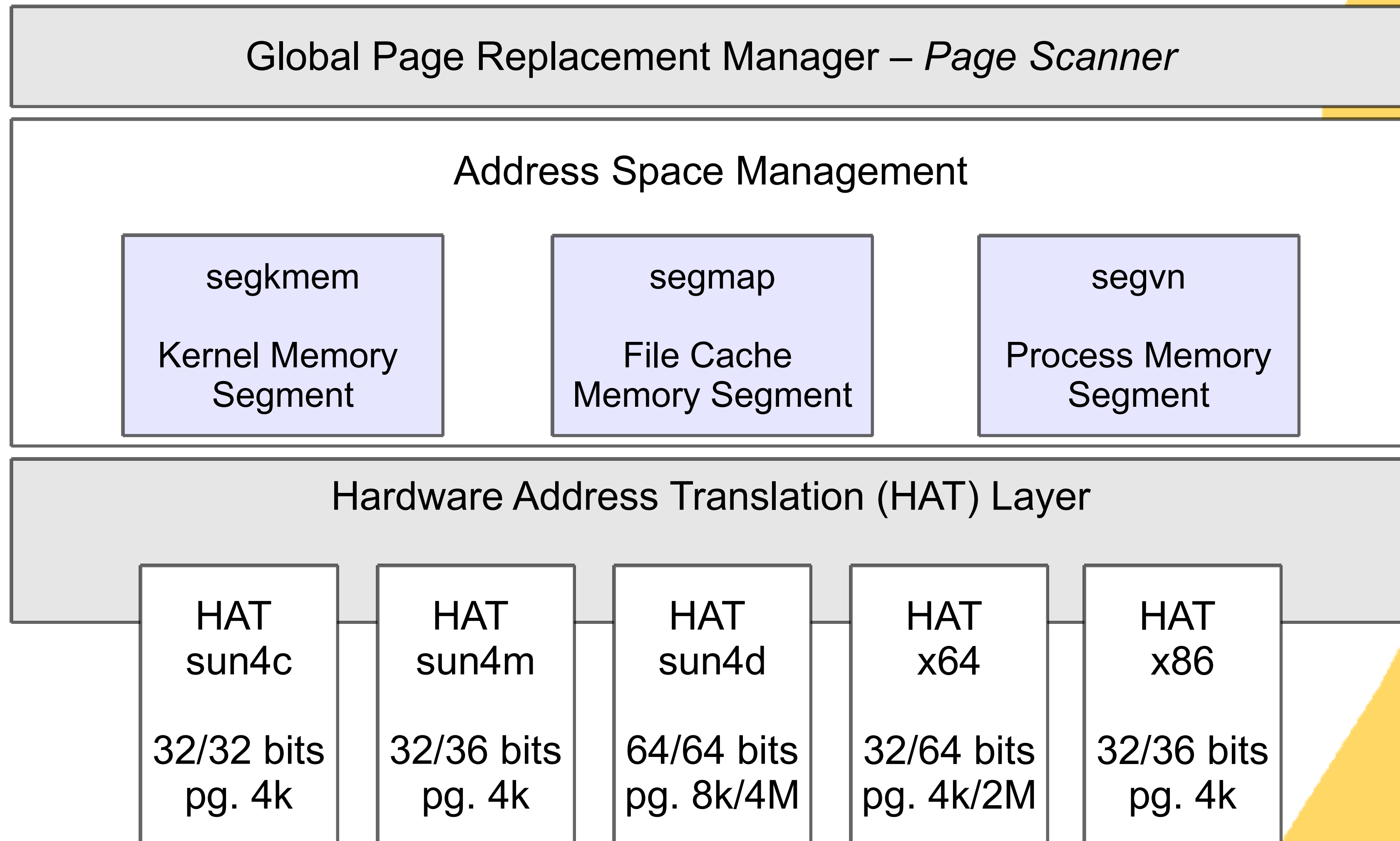
```
page_find(vnode, offset)
{
    index = PAGE_HASH_FUNC(vnode,
                           offset);

    page = PAGE_HASH_SEARCH(index,
                            vnode,
                            offset);

}
```



Virtual Memory (7/7)

opensolaris

System layers



Building the kernel

Getting the source code

- Mercurial

  $hg clone ssh://anon@hg.opensolaris.org/hg/onnv/onnv-gate my-copy

- Other components
  - on-closed-bins.i386.tar.bz2
  - SUNWonbld.i386.tar.bz2
  - http://dlc.sun.com/osol/on/downloads/current/
- install Sun Studio 11 or gcc
- install the SUNWonbld package (pkgadd -d SUNWonbld)
- read the README.OpenSolaris file (serious)

Building the kernel (2/4)

opensolaris

Workspace

- mkdir /export/opensolaris/b73
- cd /export/opensolaris/b73
- bunzip2 -c on-closed-bins.i386.tar.bz2 | tar xf -
- bunzip2 -c SUNWonbld.i386.tar.bz2 | tar xf -
- cp usr/src/tools/env/opensolaris.sh .

Editing opensolaris.sh

- GATE to source folder
- CODEMGR_WS to source folder's path
- STAFFER to your login
- VERSION to whatever you want

Building the kernel (3/4)

opensolaris

Compiling

- to build the entire ON tree
  - nightly ./opensolaris.sh &
- just the kernel:
  - bldenv -d ./opensolaris.sh
  - cd usr/src/uts
  - dmake all

Building the kernel (4/4)

Booting the new kernel

- cd usr/src/uts
- Install -G my.kernel -k i86pc
- tar xf /tmp/Install.meu.kernek /
- gedit /boot/solaris/filelist.ramdisk and add an /platform/i86pc/my.kernel entry to the end
- add the new kernel to GRUB

Other possibilities

- BFU if you changed something in userland
- guarantees that all bits are consistent

Participating

OpenSolaris community

- **http://www.opensolaris.org**
- different mailing lists, forums and user groups (find the one closest to you :)
- extensive documentation
- many ongoing development projects
- general discussions: **opensolaris-discuss@opensolaris.org**
- code talk: **opensolaris-code@opensolaris.org**
- **http://br.opensolaris.org**
- **http://www.inf.ufrgs.br/opensolaris**

opensolaris

Participating

Developing

- join the communities and projects you're interested in
- *bite sized bugs*

  http://opensolaris.org/os/bug_reports/oss_bite_size/

Bibliography

- Solaris Internals

  Richard McDougall and Jim Mauro. 2nd Edition. Prentice Hall, 2006

- Eric Saxe's Dump o' Core http://blogs.sun.com/esaxe/

  Slides at http://blogs.sun.com/esaxe/resource/cpu_sched_svosug.pdf

- whacked.net http://whacked.net/

  Slides at http://whacked.net/2006/07/06/svlug/

Questions ?

opensolaris

Rafael Vanoni

rafael.vanoni@sun.com

blogs.sun.com/rv