

# open

## Kernel OpenSolaris: Introdução a Memória Virtual e ao Escalonador

Rafael Vanoni

[rafael.vanoni@sun.com](mailto:rafael.vanoni@sun.com)

Solaris Kernel Performance Group

Sun Microsystems Inc.

開  
放  
的  
열린  
مفتوح  
libre  
मुक्त  
ಮುಕ್ತ  
livre  
libero  
ముక్త  
开放的  
açık  
open  
nyílt  
•••••  
πικρ  
オープン  
livre  
ανοικτό  
offen  
otevřený  
öppen  
открытый  
வெளிப்படை

# Agenda

---

- Arquitetura do kernel
- Processos, threads e LWPs
- Escalonador e *dispatcher*
- Memória Virtual
- Compilando o kernel
- Participando da comunidade

## Arquitetura do kernel

- Interface de chamadas de sistema: permite que processos acessem utilidades do kernel
- Execução de processos e escalonamento: criação, execução, gerencia e termino de processos
- Gerencia de memoria: o sistema de memoria virtual mapeia memoria física para processos de usuário e para o kernel
- Gerencia de recursos: permite a alocação de recursos específicos de hardware para certas aplicações
- Sistema de arquivos: implementação de um sistema de arquivos virtual sob o qual diferentes sistemas de arquivos podem ser configurados (de disco, de rede, pseudo, ..)

## Arquitetura do kernel

- Gerencia de I/O e dispositivos
- Facilidades do kernel: interrupções de clock, timers, primitivas de sincronização e suporte a módulos
- Networking: suporte a IPv4 e Ipv6, interfaces de socket e STREAMS

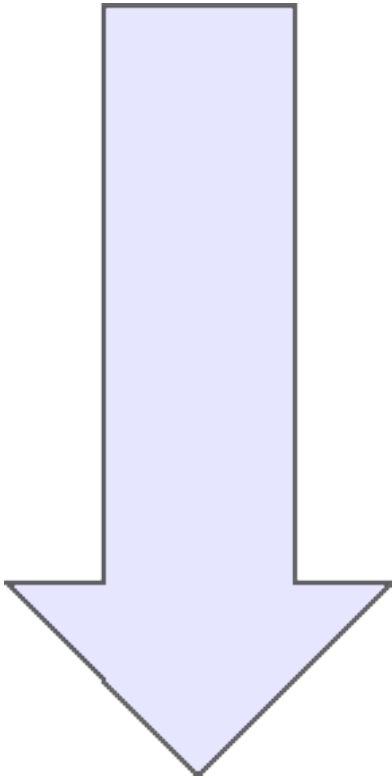
## Organização do código

- /src/uts                      Unix Time Sharing (kernel)
- /src/uts/common            código independente de arquitetura
- /src/uts/i86pc              código para arquitetura x86
- /src/uts/sparc              código para arquitetura SPARC

## Neste escopo

- /src/uts/[common,i86pc]/sys            Headers do kernel
- /src/uts/[common,i86pc]/os            Implementações
- /src/uts/[common,i86pc]/vm            Memoria Virtual
- /src/uts/common/disp            Escalonador e dispatcher

## Modelo e Implementação



**Processo:** forma executável de um programa

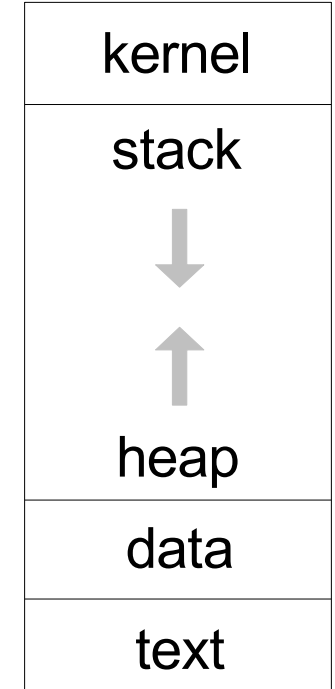
**User Thread:** estado de uma thread mantido em um processo de usuário

**Lightweight Process (LWP):** a visão do kernel do ponto de vista de uma thread de usuário

**Kernel Thread:** o objeto de escalonamento e execução em um processador

## Conceitos

- um processo é a combinação de um programa e o estado atual de sua execução (uma instância do programa)
- possui um espaço de endereçamento com segmentos de texto, dados, heap, pilha e área do kernel
- contem um ou mais fluxos de execução (threads), que compartilham o espaço de endereçamento (exceto a pilha)
- definido no kernel em `uts/common/sys/proc.h`
- tipo `proc_t`, contem inúmeros campos/estruturas
- funções em `uts/common/os/proc.c`

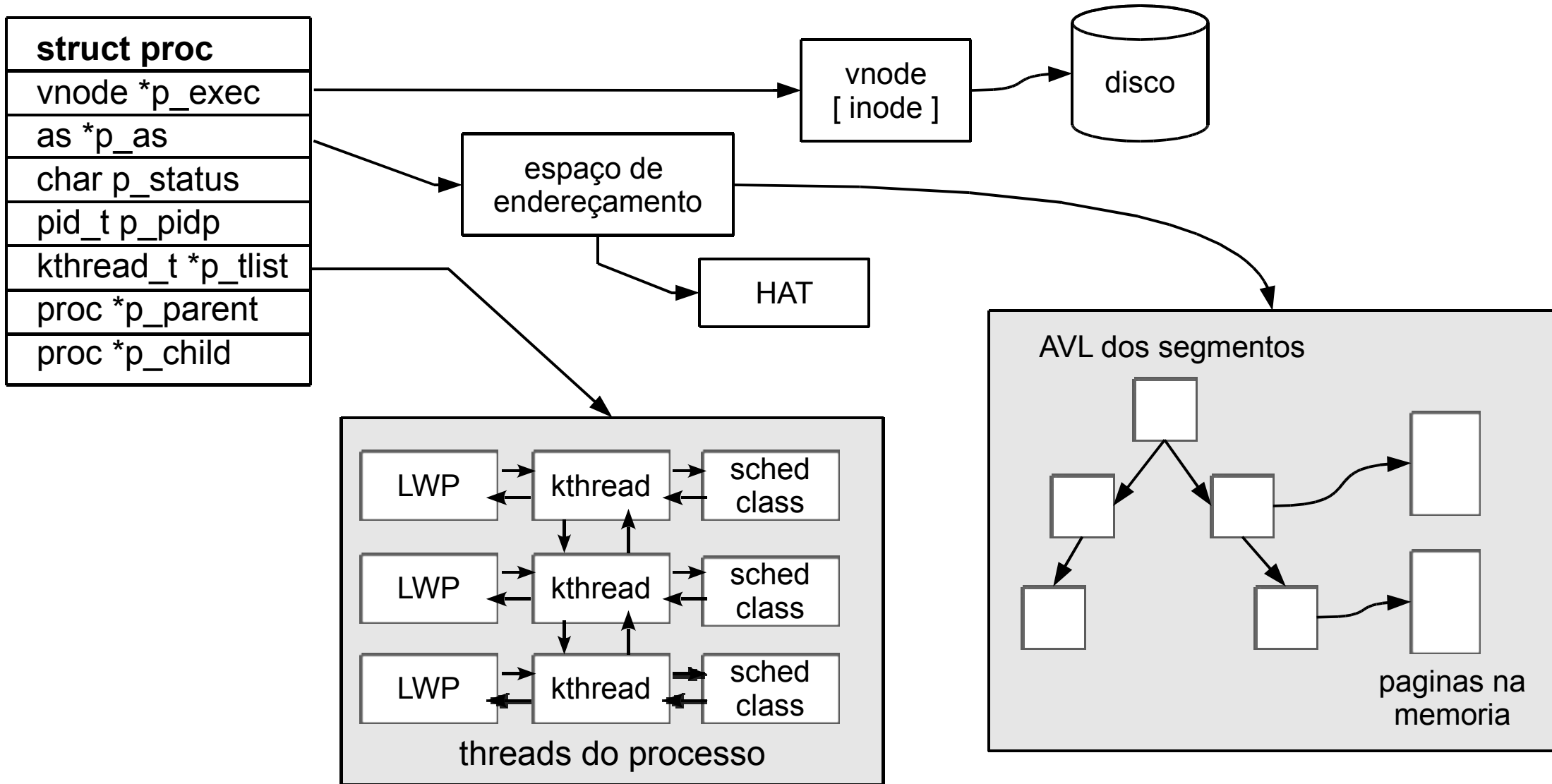


## uts/common/sys/proc.h

- struct proc com mais de 100 campos
- locks para gerenciar acesso concorrente
- ponteiro para executável
- ponteiro para espaço de endereçamento
- diversos ponteiros para listas de processos (pai, filhos, próximo, anterior, ..)
- identificador do processo (pid)
- estruturas para contabilidade do sistema
- estrutura sys/user.h (uso de memória, diretório atual, ..)

# Processos

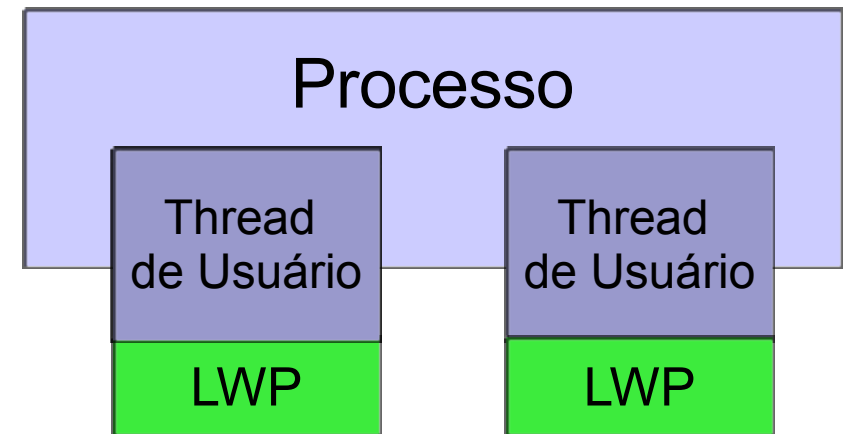
uts/common/sys/proc.h



## Light Weight Process (LWP)

- cada thread em um processo possui uma estrutura LWP
- esta estrutura mantem o estado de uma thread de usuário
- definida em uts/common/sys/klwp.h
- tipo `klwp_t`

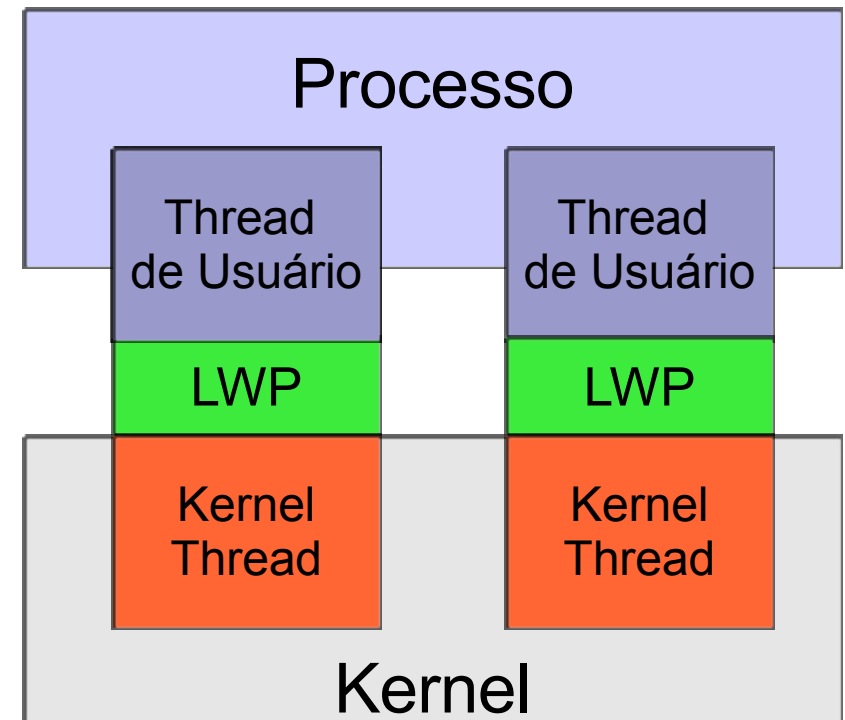
<b>struct _klwp</b>
pcb lwp_pcb
void *lwp_regs
char lwp_state
kthread *lwp_thread
proc *lwp_proc



## Kernel Threads

- cada LWP esta ligada a uma Kernel Thread
- é a unidade básica de escalonamento do sistema
- definida em uts/common/sys/thread.h
- tipo `kthread_t`

<b>struct _kthread</b>
<code>_kthread *t_link</code>
<code>caddr_t t_stk</code>
<code>void (*t_startpc)(void)</code>
<code>uint_t t_state</code>
<code>pri_t t_pri</code>
<code>caddr_t t_swap</code>
<code>cpu *t_cpu</code>
<code>klwp_t *t_lwp</code>



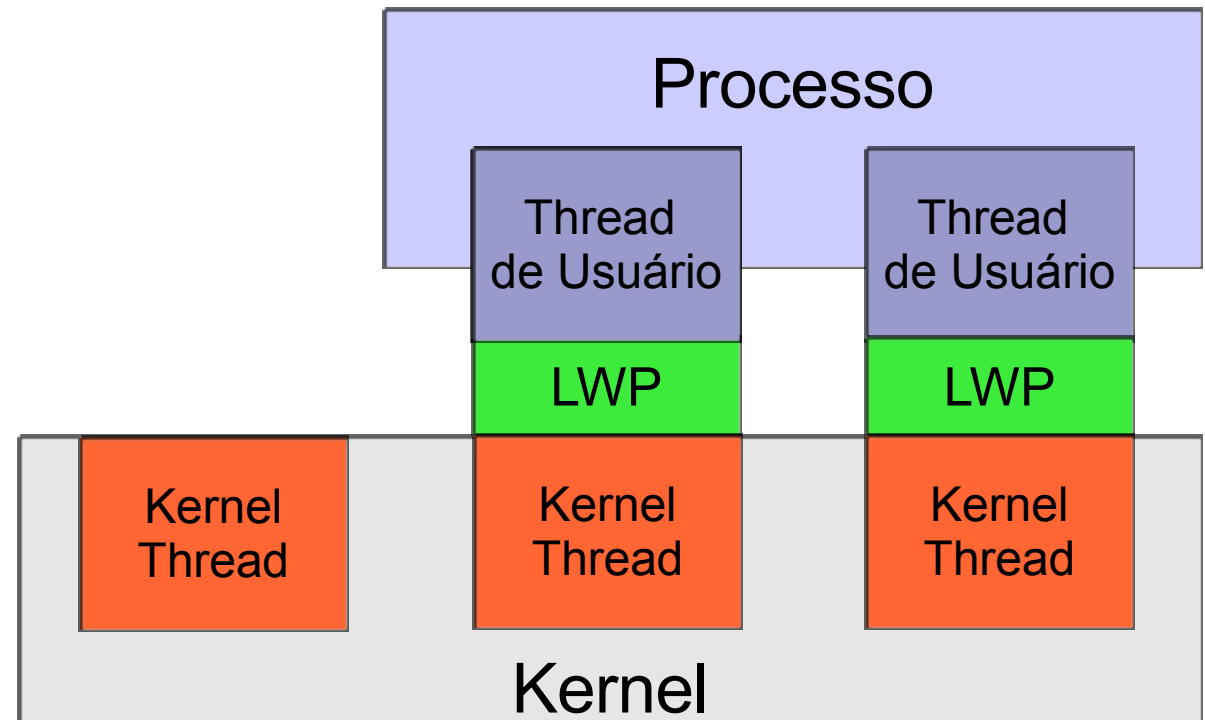
## Kernel Threads

- nem toda kthread esta ligada a uma thread de usuário
- por exemplo: thread ociosa (idle), gerencia de filas, ..

## Implementação

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

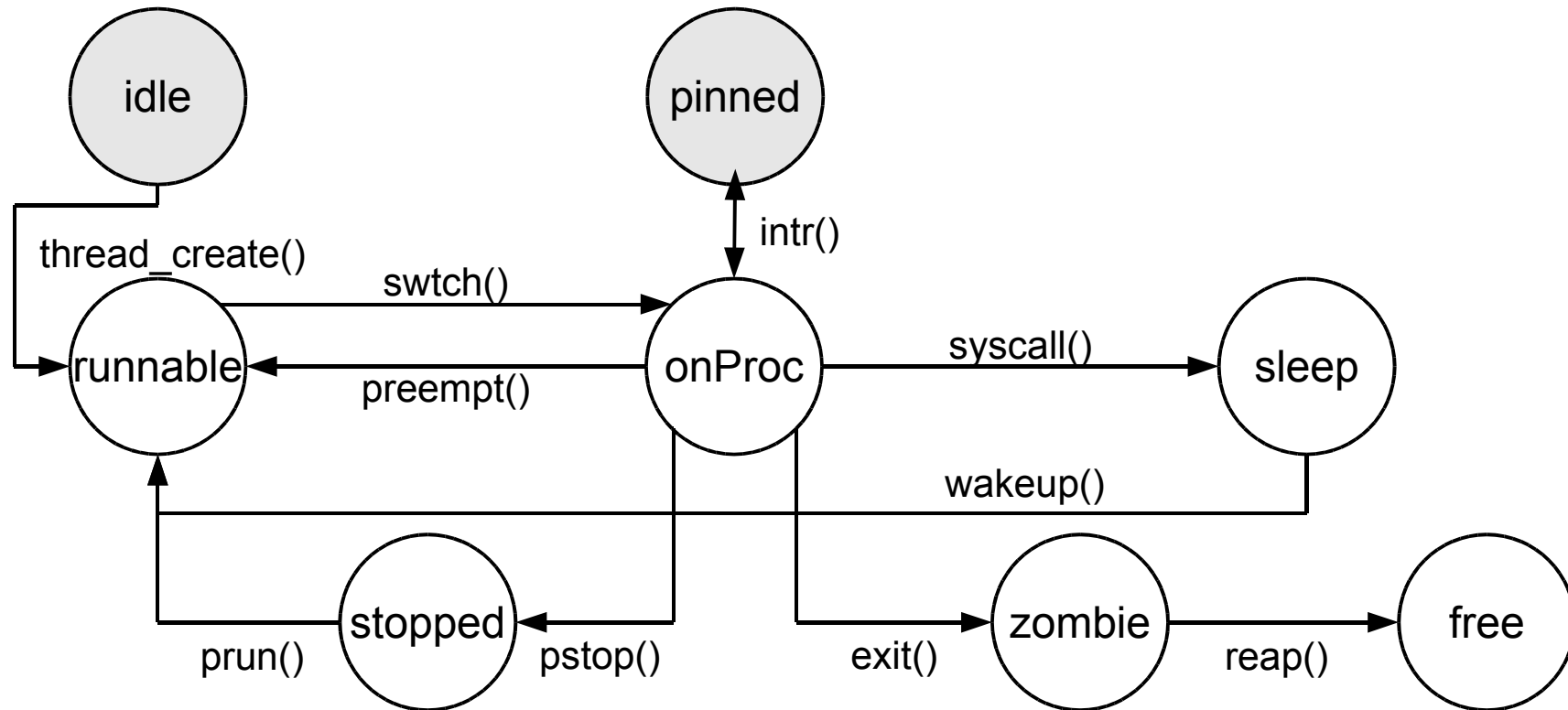
```
...  
void          thread_init()  
kthread*     thread_create()  
void         thread_exit()  
void         thread_join()  
void         thread_free()  
...
```



## Estados

- *Runnable*: pronta para ser escalonada
- *On Proc*: executando em uma CPU
- *Sleeping*: bloqueada aguardando um evento (por exemplo, I/O)
- *Stopped*: suspensa
- *Zombie*: terminada mas ainda ocupando espaço em memória
- *Free*: terminada e com espaço liberado

## Diagrama de estados



## Conceitos

- o escalonador é o subsistema do kernel que administra a execução de threads no sistema através da gerencia de filas
- o *dispatcher* é o componente que aplica as decisões do escalonador, ele decide **onde** uma thread sera executada e é encarregado da **troca de contexto**
- preemptivo, inclusive para threads do kernel
- utiliza um modelo de prioridades globais para selecionar qual thread deve ser executada (0-170)
- possui diversas **classes de escalonamento** que dividem esta faixa, alem de reservar o topo para atendimento de interrupções

# Escalonador e *dispatcher*

## Responsabilidades

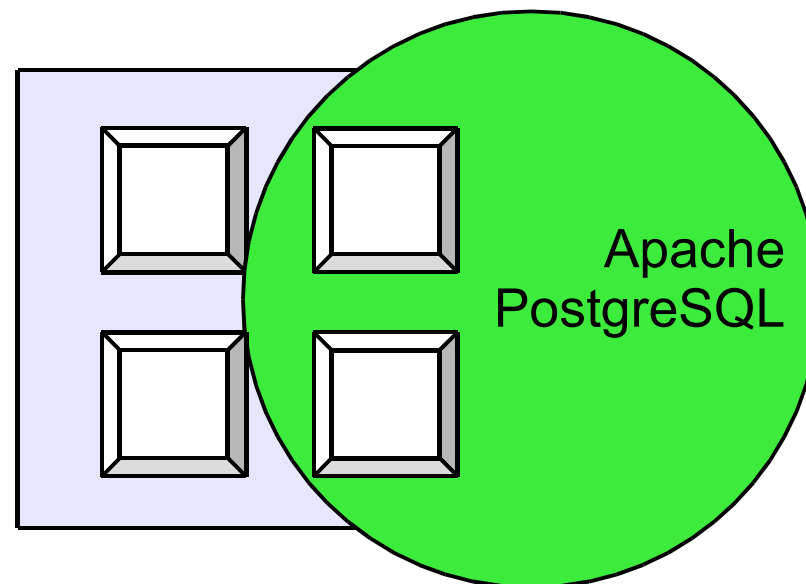
- gerencia de filas
- escolha de thread
- seleção de processador
- troca de contexto

## Decisões de escalonamento

- esquema de prioridades
- parâmetros da gerencia de recursos
- arquitetura do sistema

## Parâmetros de gerencia de recursos

- tecnologias do OpenSolaris para gerenciar recursos de hardware:
  - *Processor Binding*: amarra processos a processadores
  - *Processor Sets*: permite a criação de conjuntos de processadores (formando subconjuntos do total de processadores) e a amarração de processos a estes conjuntos



## Parâmetros de gerencia de recursos

- *Resource pools*: basicamente *Processor Sets stateful*
- *Zones*: provem um ambiente de execução virtualizado, podendo amarrar um *Resource Pool* a uma *Zone*

## Arquitetura do sistema

- modificações feitas ao *dispatcher* em função de características do sistema
  - arquiteturas NUMA (*NonUniform Memory Access*)
  - processadores CMT (*Chip Multithreading*)
- por exemplo, explorar *cache warmth*

## Classes de escalonamento

- Time Share (TS) src/uts/common/disp/ts.c
  - faixa de prioridades 0-59
  - prioridades ajustadas de acordo com quanto tempo threads utilizam recursos do processador
  - classe padrão para processos e kernel threads
- Interactive (IA) src/uts/common/disp/ia.c
  - faixa de prioridades 0-59
  - igual a TS, mas com *boost* para aplicações interativas

## Classes de escalonamento

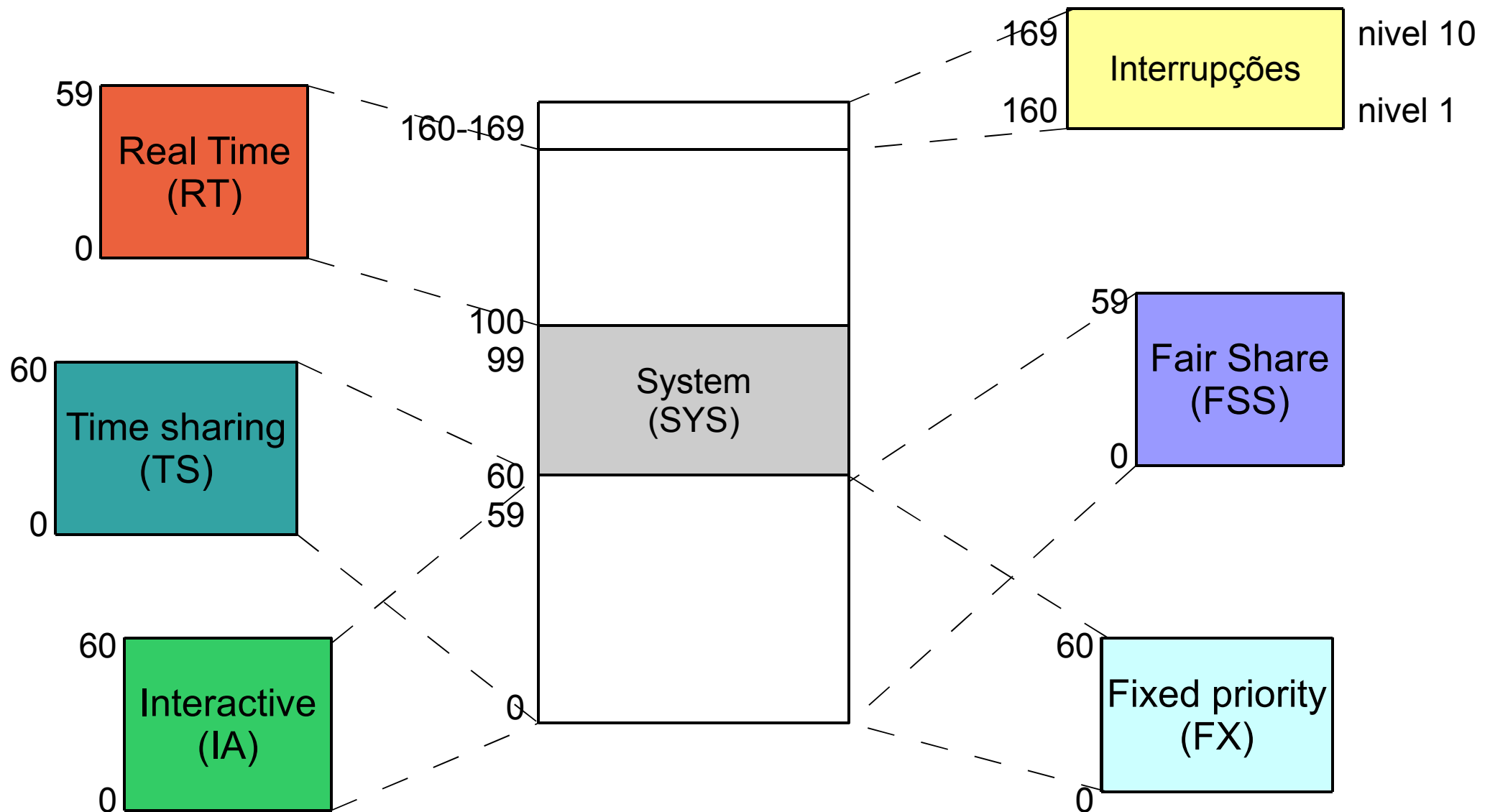
- Fair Share (FSS) `src/uts/common/disp/fss.c`
  - faixa de prioridades 0-59
  - recursos de processamento divididos em cotas designadas a processos pelo gerenciamento de recursos do OpenSolaris
  - prioridades ajustadas de acordo com cotas e utilização dos recursos
- Fixed Priority (FX) `src/uts/common/disp/fx.c`
  - faixa de prioridades 0-60
  - prioridades estáticas

## Classes de escalonamento

- System (SYS) src/uts/common/disp/sysclass.c
  - faixa de prioridades 60-99
  - usada por threads de sistema
- Real Time (RT) src/uts/common/disp/rt.c
  - faixa de prioridades 100-159
  - podem *preemptar* o kernel

# Escalonador e *dispatcher*

## Classes de escalonamento



## Funcionamento

- threads entram no dispatcher quando mudam de estado ou quando causam outra thread a mudar de estado

$[sleep, onProc] > [runnable]$

thread entra no dispatcher que escolhe uma CPU e a coloca na fila desta CPU

$[runnable] > [onProc]$

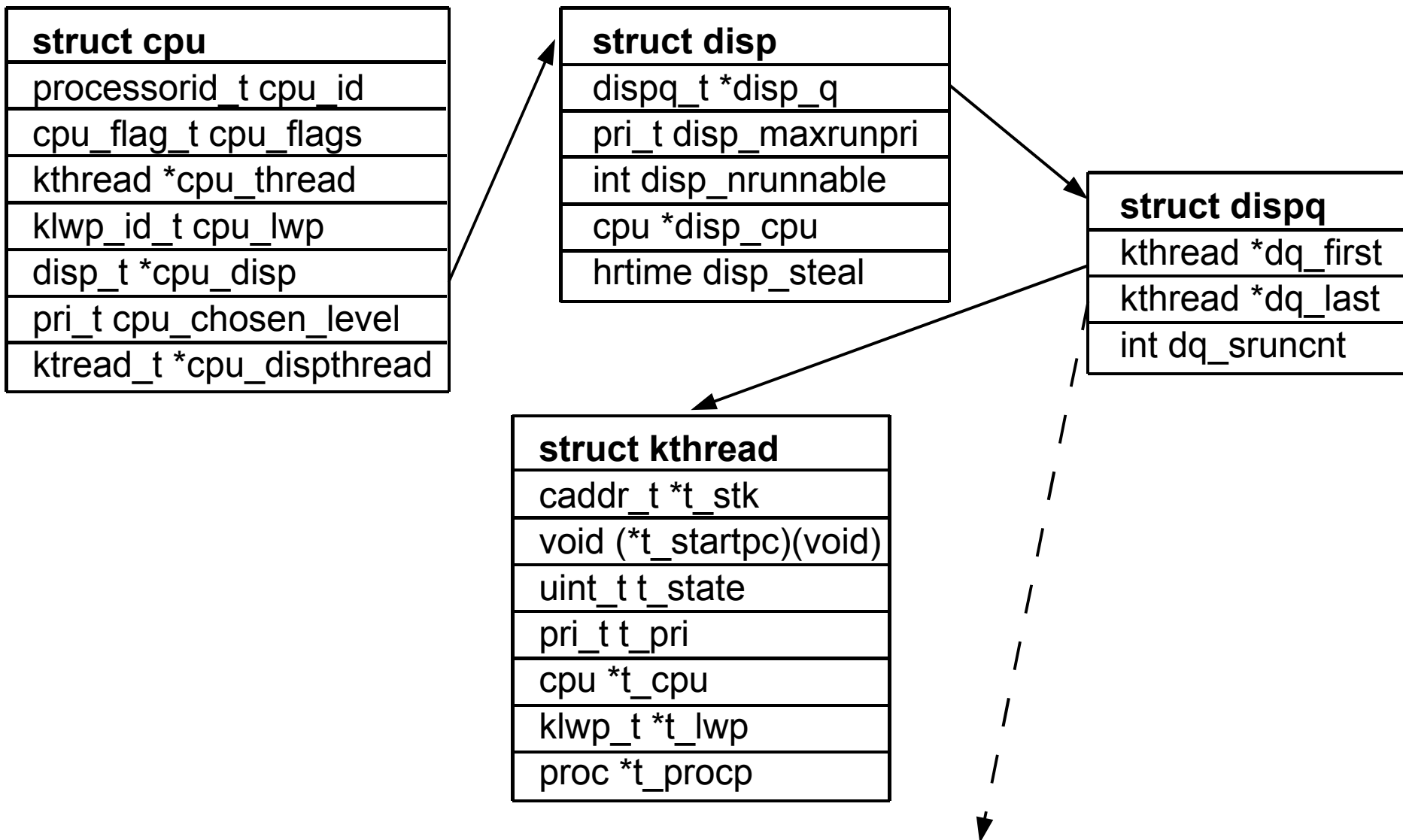
thread entra no dispatcher e a thread com maior prioridade e retirada da fila e o contexto e trocado

$[onProc] > [sleep]$

thread fica bloqueada na fila do lock do objeto em questão, tire a thread com maior prioridade da fila da CPU e troca para esta

# Escalonador e dispatcher

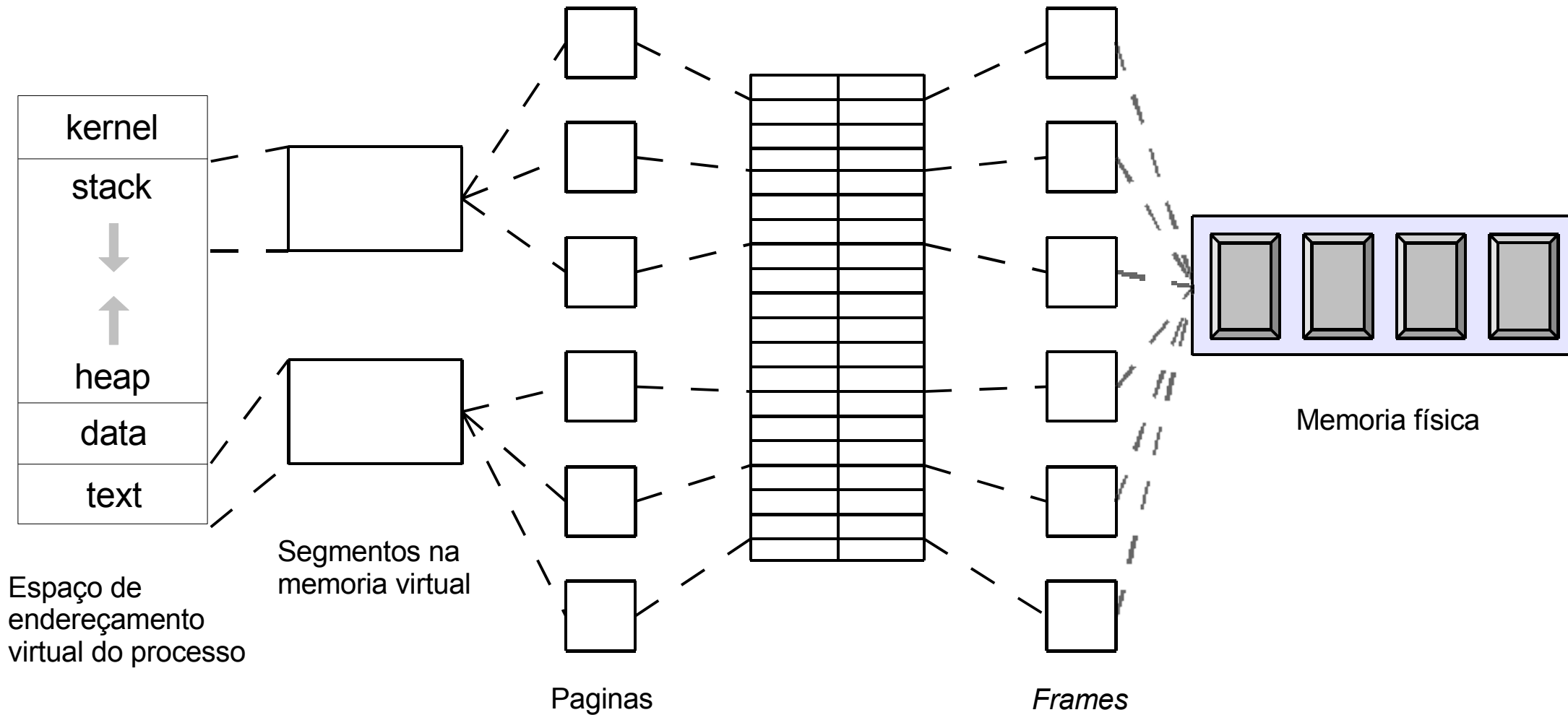
Estruturas (src/uts/common/sys/disp.h)



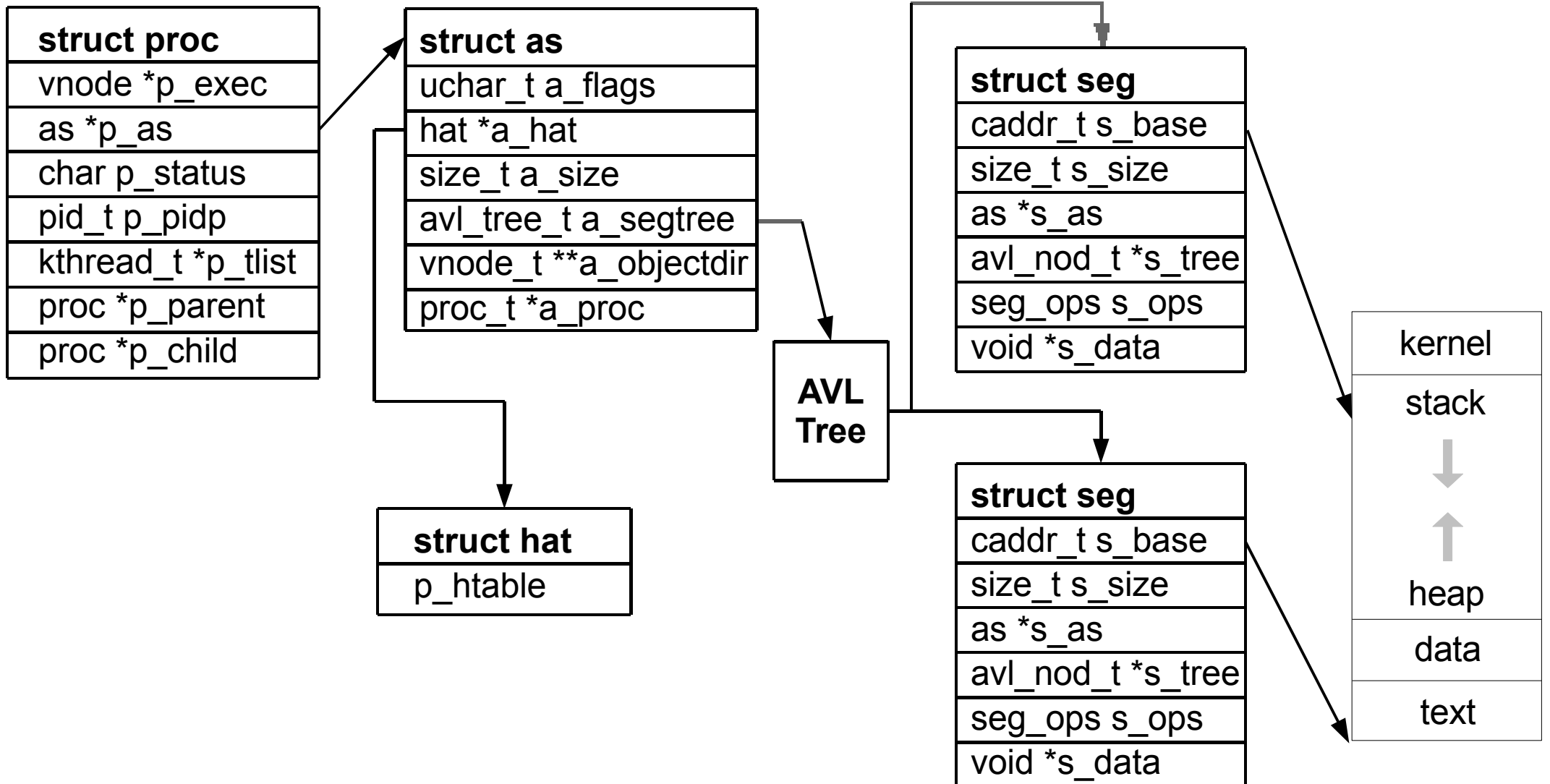
## Conceitos

- objetivos principais:
  - livrar restrições de tamanho de processos a memória física
  - permitir que maior nível de multiprogramação, buscando manter apenas as partes necessárias do processo em memória
- paginação:
  - divide a memória e o processo em pedaços de tamanho fixo, chamados de *frames* e paginas
  - processo gera endereços lógicos que são traduzidos pela MMU em endereços físicos através de tabelas de paginas
- implementação em camadas: independente de hardware e dependente de hardware (HAT)

## Espaços de endereçamento, segmentos e paginas



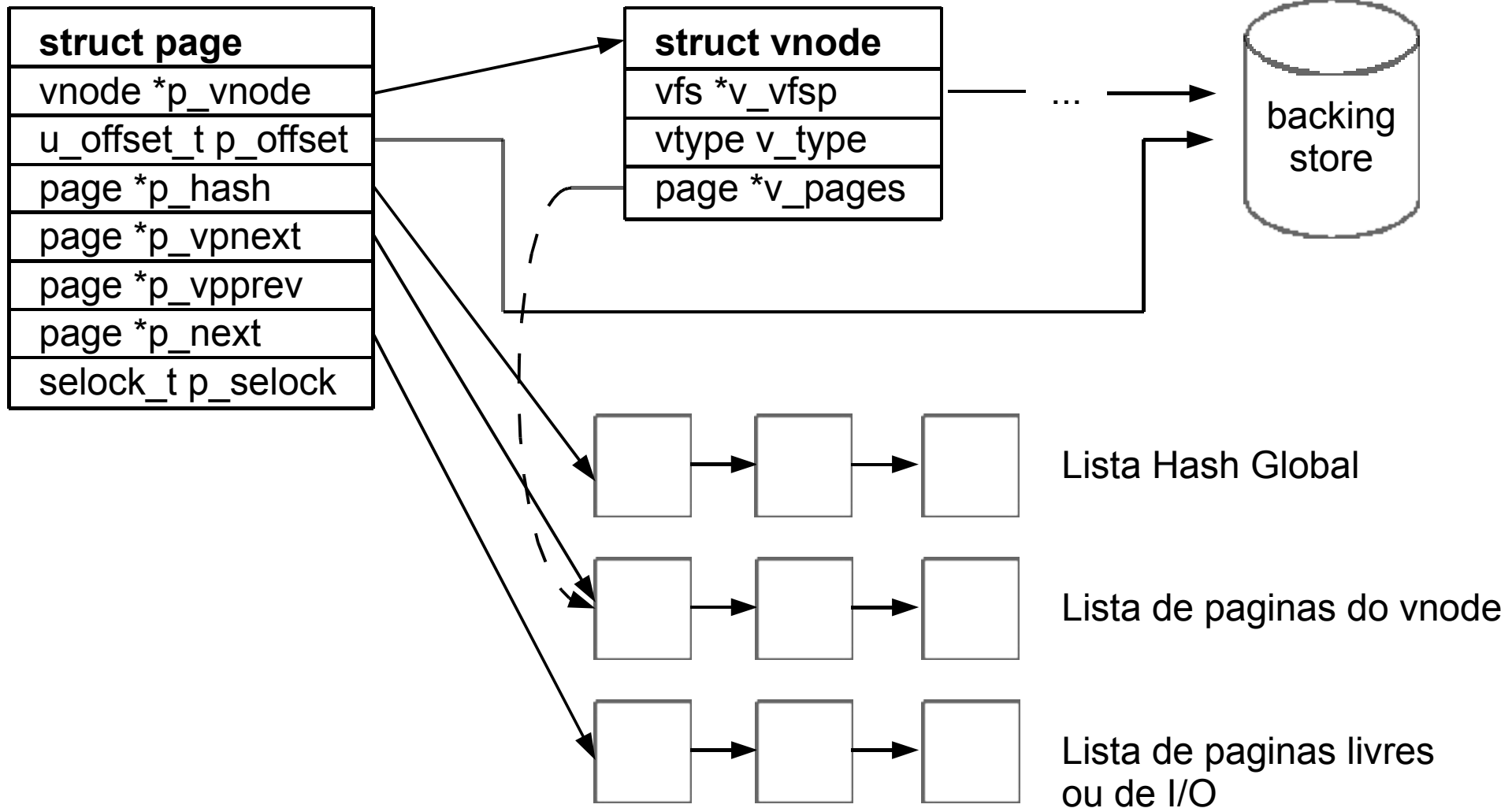
## Estruturas



## Paginas

- unidade fundamental de memoria
- toda pagina ativa (não livre) é um mapeamento entre um arquivo (vnode) e a memoria
- paginas são identificadas partir do vnode e de um offset (par vnode/offset)
- este par e o *backing store* para a pagina
- reusabilidade: o par também é usado para mapear o arquivo em swap ou para cache de arquivo
- o mapeamento entre pagina física e seu espaço virtual e feito pelo HAT
- uma lista *hash* global de paginas contem ponteiros para listas de paginas e é indexada por uma função `hash(vnode/offset)`

## Estruturas

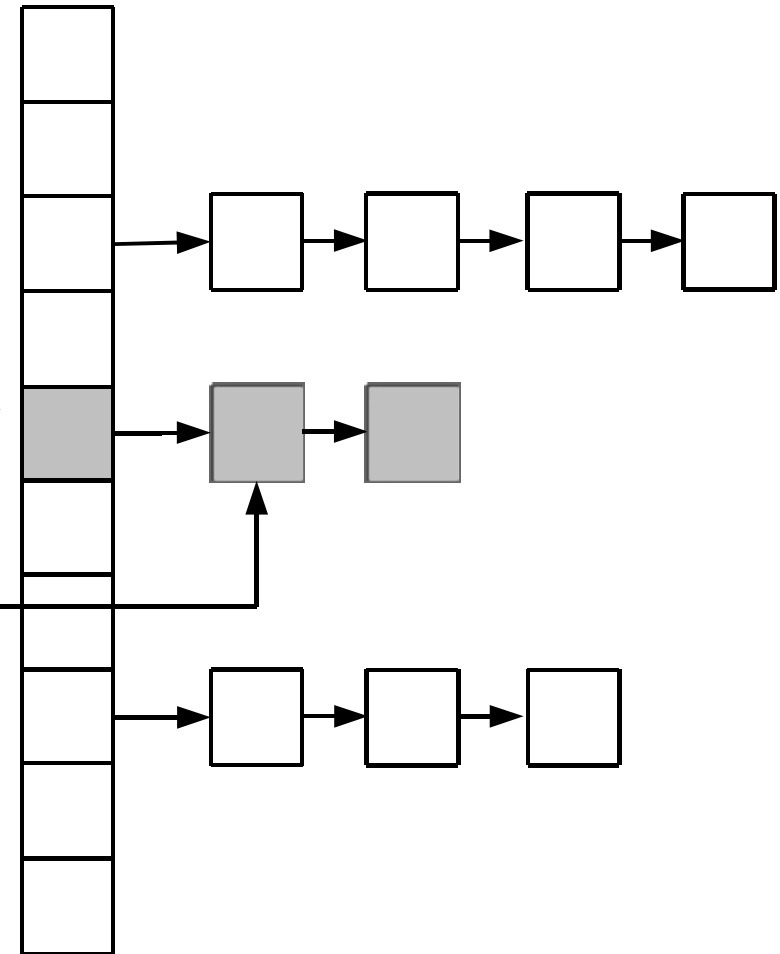


## Achando uma pagina

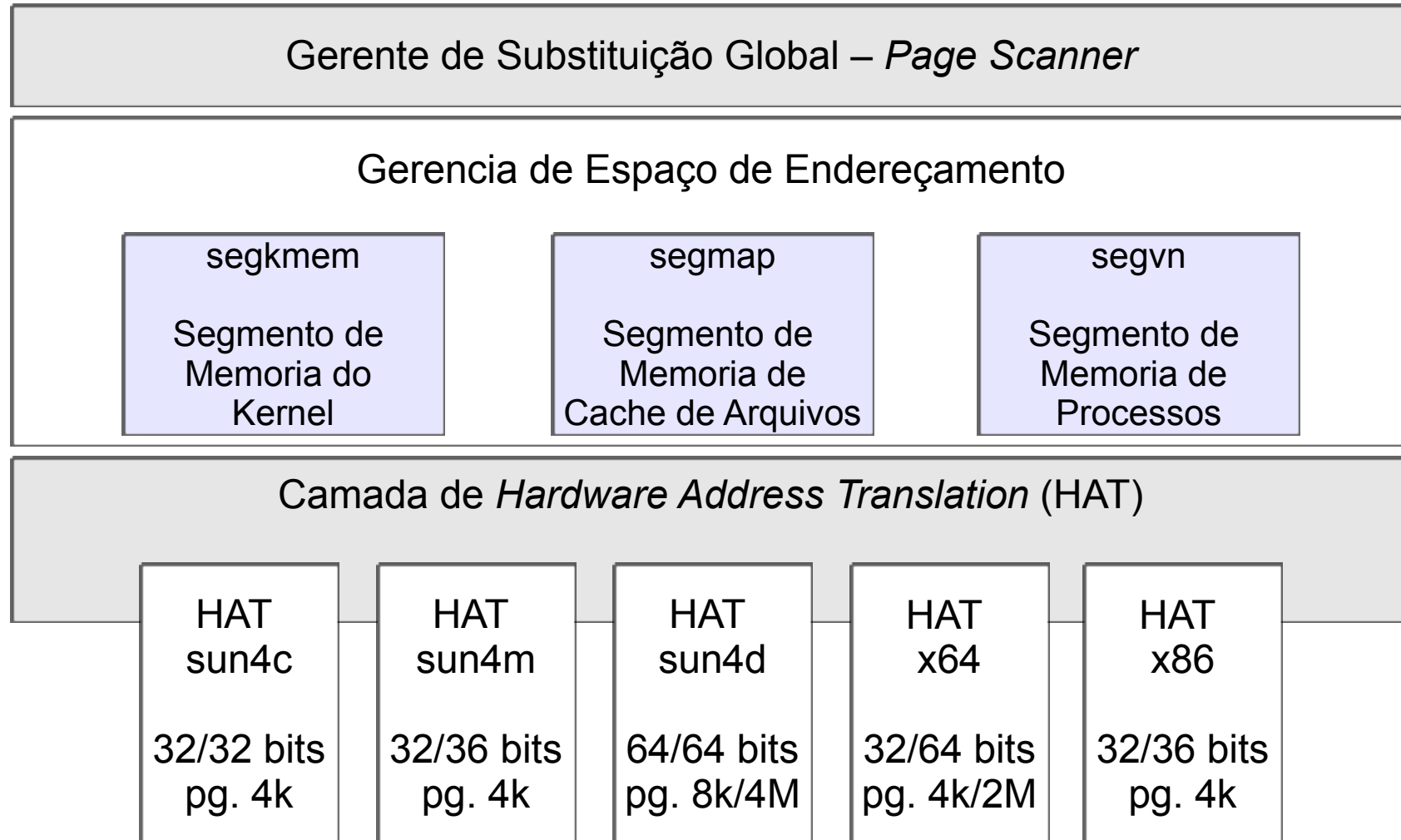
```
page_find(vnode, offset)
{
    index = PAGE_HASH_FUNC(vnode,
                           offset);

    page = PAGE_HASH_SEARCH(index,
                           vnode,
                           offset);
}
```

Lista Hash Global



## Camadas do sistema



## Baixando o código fonte

- Mercurial

```
$hg clone ssh://anon@hg.opensolaris.org/hg/onnv/onnv-gate minha-copia
```

- Outros componentes

- on-closed-bins.i386.tar.bz2

- SUNWonbld.i386.tar.bz2

- <http://dlc.sun.com/osol/on/downloads/current/>

- instale o Sun Studio 11 ou o gcc

- adicione o pacote SUNWonbld

- leia o README.OpenSolaris

## Descompactando

- `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 .`

## Editando o opensolaris.sh

- altere GATE para o diretório b73
- altere CODEMGR\_WS para o *path* do b73 (`/export/opensolaris`)
- altere STAFFER para o seu login
- altere VERSION para identificar a versão

## Compilando

- toda a arvore:
  - nightly ./opensolaris.sh &
- apenas um componente (no caso, o kernel):
  - bldenv -d ./opensolaris.sh
  - cd usr/src/uts
  - dmake all

## Bootando o novo kernel

- `cd usr/src/uts`
- `Install -G meu.kernel -k i86pc`
- `tar xf /tmp/Install.meu.kerne /`
- adicione `/platform/i86pc/meu.kernel` ao arquivo `/boot/solaris/filelist.ramdisk`
- adicione a nova entrada ao grub

## Outras maneiras..

- BFU se você modificou alguma biblioteca/*userland*
- garante que todos bits sendo bootados estarão consistentes

## Comunidade OpenSolaris

- **<http://www.opensolaris.org>**
- diversas listas de discussões, fóruns e grupos de usuários
- documentação extensa
- vários projetos de desenvolvimento
- discussões gerais: **[opensolaris-discuss@opensolaris.org](mailto:opensolaris-discuss@opensolaris.org)**
- discussões sobre código: **[opensolaris-code@opensolaris.org](mailto:opensolaris-code@opensolaris.org)**
- **<http://br.opensolaris.org>**
- **<http://opensolaris.org/os/project/poaosug/>**

## Desenvolvendo

- participe das comunidades e de seus projetos
- *bite sized bugs*

[http://opensolaris.org/os/bug\\_reports/oss\\_bite\\_size/](http://opensolaris.org/os/bug_reports/oss_bite_size/)

- Solaris Internals

Richard McDougall e Jim Mauro. Prentice Hall, 2006

- Eric Saxe's Dump o' Core

Blog do Eric Saxe <http://blogs.sun.com/esaxe/>

Slides em [http://blogs.sun.com/esaxe/resource/cpu\\_sched\\_svosug.pdf](http://blogs.sun.com/esaxe/resource/cpu_sched_svosug.pdf)

- whacked.net

Blog do Stephan Lau <http://whacked.net/>

Slides em <http://whacked.net/2006/07/06/svlug/>

# Perguntas ?

---

opensolaris™

Rafael Vanoni

[rafael.vanoni@sun.com](mailto:rafael.vanoni@sun.com)

Solaris Kernel Performance Group

Sun Microsystems Inc.