



UNIFOR
ENSINANDO E APRENDENDO

Disciplina de Sistemas Operacionais (Aula 5)



Gerência de Memória

Apresentação



Francisco Nauber Bernardo Gois

Analista aprendizado de máquina
no Serviço Federal de Processamento
de Dados

Doutorando em Informática Aplicada
Mestre em Informática Aplicada
Especialista em desenvolvimento WEB

Dúvidas: naubergois@gmail.com

**Jovem Padawan
procure na aula**

**ao telefone
não falar**

**Sem a presença
Você não passará**

**Para melhor
desempenho na
aula**

**Procure
não
conversar
durante a
aula**

**Buscar
aprendizado
ao invés de
pontos**



**Não
teremos
pontuação fora dos
trabalhos e provas
da disciplina**



**Cuidado
com o
Horário**



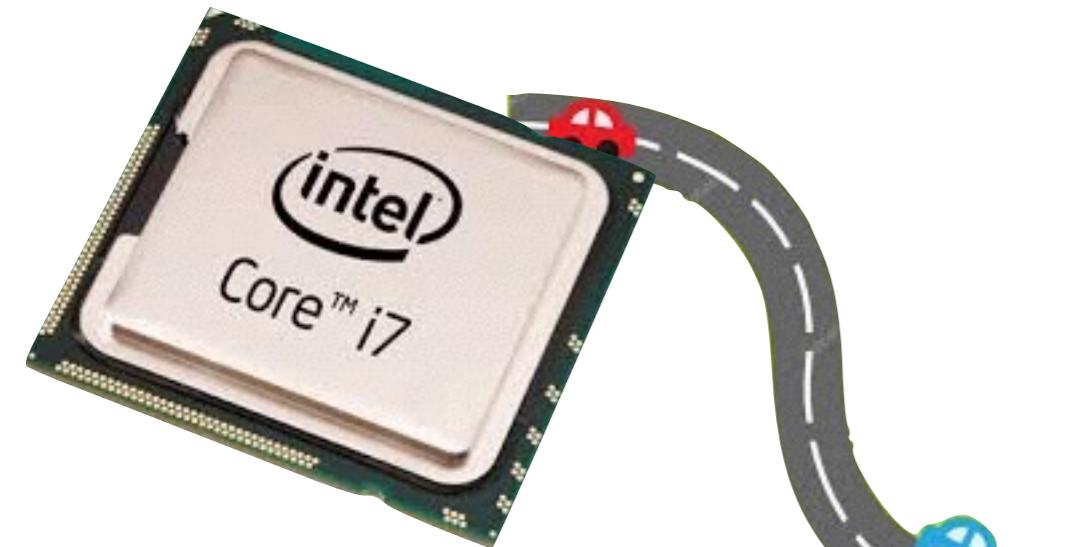
OS trabalhos
deveram ser
entregues uma
semana antes
da prova

Um cadeira longa
e prospera

Não teremos
pontos após a prova
não adianta pedir



Cronograma da Disciplina



**Gerência
de Processador**

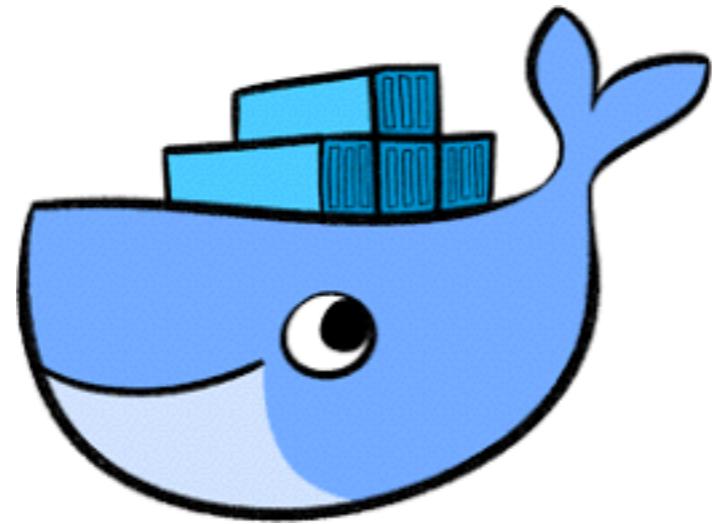


**Gerência de
Memória**



Docker com Apache

```
docker run -p 80:80 -dit --name my-app -v "$PWD":/usr/local/apache2/htdocs/ httpd:2.4
```



- `/dev`—Contains device drivers
- `/bin` and `/usr/bin`—Contains standard Linux commands
- `/lib` and `/usr/lib`—Contains standard Linux libraries
- `/var`—Contains configuration and log files
- `/etc`—Contains default configuration files
- `/usr/local/bin`—Contains commands not a part of the distribution, added by your administrator
- `/opt`—Contains commercial software
- `/tmp`—Stores temporary files
- `/sbin` and `/usr/sbin`—Contains system administration commands (`/sbin` stands for “safe” bin)

Gerenciamento de Memória

Memória

- A **memória** pode ser vista como um *array* (vetor) de células de armazenamento (palavras ou bytes), cada célula com seu endereço

0	2
1	31
2	4
3	35
4	26
5	124
6	42
7	12
8	42

•
•
•

Gerenciamento de Memória

Memórias física, lógica e virtua

- Memória física
 - É a memória implementada pelo hardware
- Memória lógica de um processo
 - É a memória endereçada pelas instruções de máquina do processo
- Memória Virtual
 - É uma memória implementada pelo SO, com o auxílio da memória secundária (disco). Comumente, é implementada através de paginação ou segmentação.
 - Normalmente, é maior que a memória física do computador

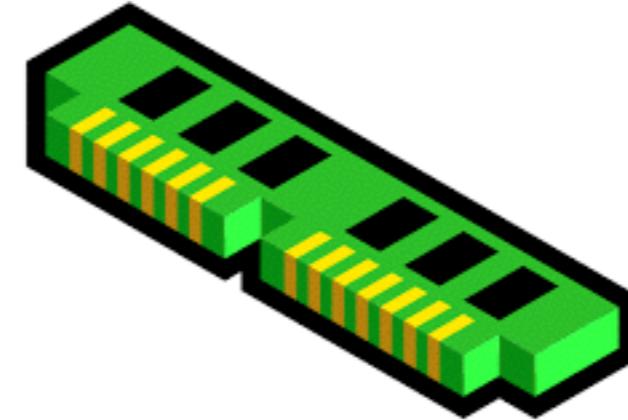


•
•
•

Gerenciamento de Memória

Mecanismos para Gerência de Memória

- máquina pura
- monitor residente
- swapping
- partções múltiplas
- paginação
- segmentação
- sistemas combinados



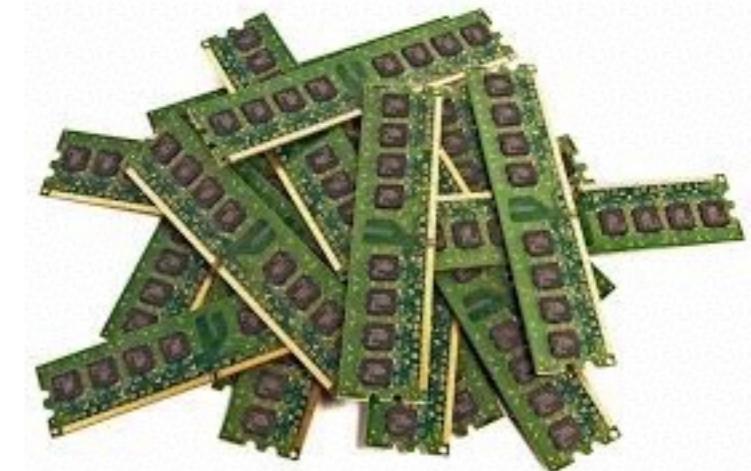
⋮

Gerenciamento de Memória

Máquina Pura

- É o esquema mais simples, pois não existe gerência de memória
- O usuário lida diretamente com o hw e possui total controle sobre toda a memória
- Fornece maior flexibilidade para o usuário, máxima simplicidade e custo mínimo, pois não exige sw ou hw especiais
- O software para essas máquinas é desenvolvido através de compiladores que executam em outras máquinas (compiladores cruzados)

•
•
•

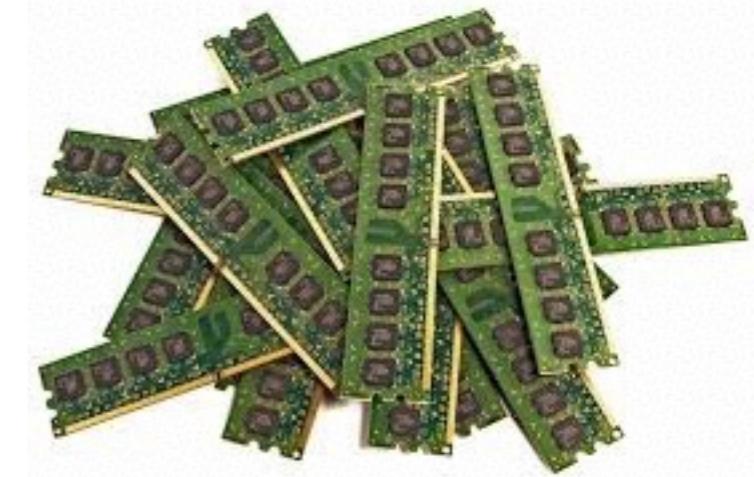


Gerenciamento de Memória

Máquina Pura

- Problemas:
 - não existe a infra-estrutura do SO (rotinas de E/S, por exemplo)
 - não há monitor residente para controlar chamadas de sistema ou erros
- Viável apenas em sistemas dedicados, onde o computador controla um equipamento específico.

•
•
•



Gerenciamento de Memória

Sistemas monoprogramados

- Com monoprogramação a gerência de memória fica simples
- O espaço é dividido entre o SO e o processo do usuário que está sendo executado

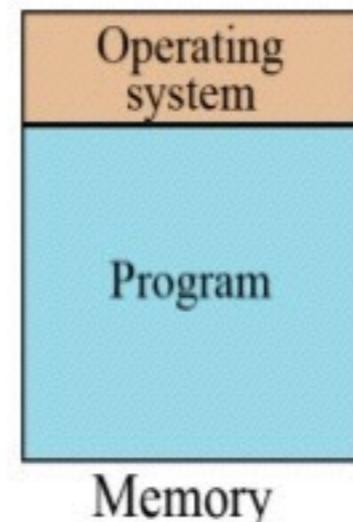


FIG 5(a): mono programming

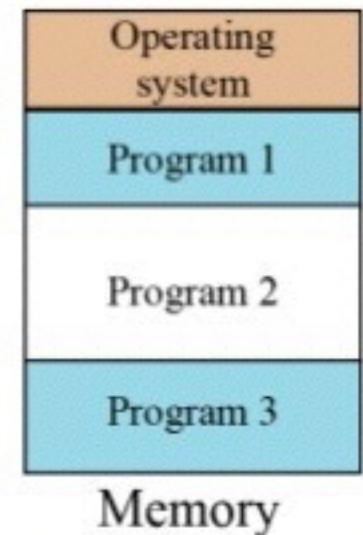
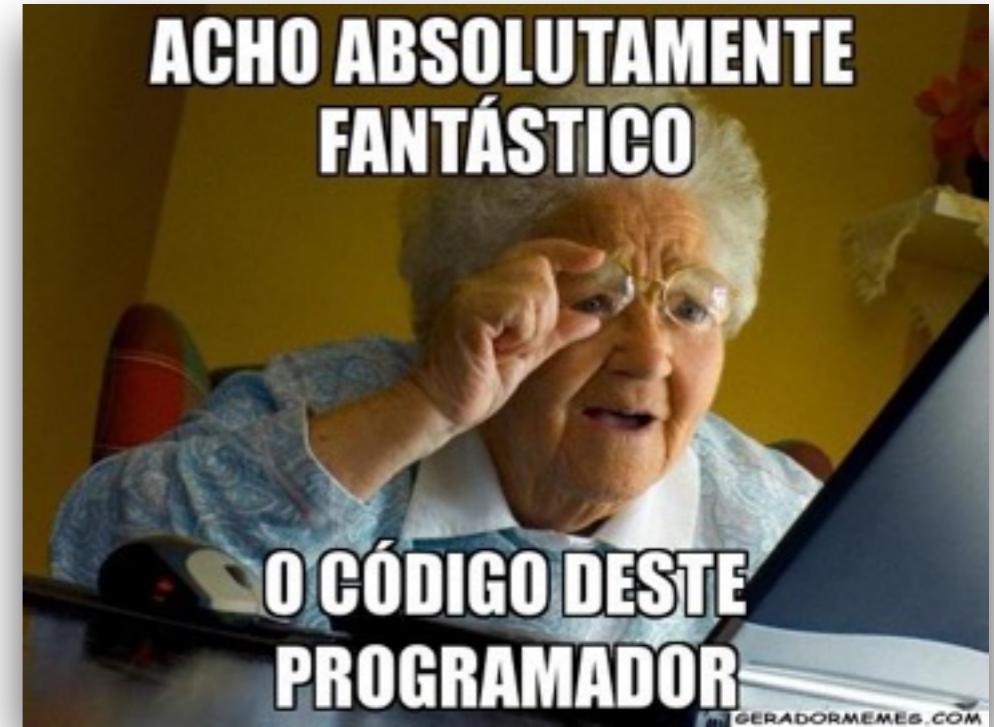


FIG 5(b): multiprogramming

Coding Dojo

```
#!/bin/sh
echo -n "Can you write device drivers? "
read answer
answer=`echo $answer | tr [a-z] [A-Z]`
if [ $answer = Y ]
then
    echo Wow, you must be very skilled
    echo this is answer: $answer
else
    echo Neither can I, Im just an example shell script
    echo this is answer: $answer
fi
.
.
```



Coding Dojo

```
username=""  
echo "${username:=$LOGNAME}"
```



•
•
•

Coding Dojo

Addition: `answer=$((c+d))`
Subtraction: `answer=$((c-d))`
Multiplication: `answer=$((c*d))`
Division: `answer=$((c/d))`
Remainder: `answer=$((c%d))`
Exponentiation: The first argument raised to the power of the second;
`answer=$((c**d))`



•
•
•

Multiple Commands

Multiple commands can be combined on a single line. How they are executed depends on what symbols separate them.

If each command is separated by a semicolon, the commands are executed consecutively, one after another.

```
$ printf "%s\n" "This is executed" ; printf "%s\n" "And so is this"  
This is executed  
And so is this
```

If each command is separated by a double ampersand (&&), the commands are executed until one of them fails or until all the commands are executed.

```
$ date && printf "%s\n" "The date command was successful"  
Wed Aug 15 14:36:32 EDT 2001  
The date command was successful
```

If each command is separated by a double vertical bar (||), the commands are executed as long as each one fails until all the commands are executed.

```
$ date 'duck!' || printf "%s\n" "The date command failed"  
date: bad conversion  
The date command failed
```

Coding Dojo

```
$ date  
Wed Apr  4 11:55:58 EDT 2001  
$ !d  
Wed Apr  4 11:55:58 EDT 2001
```

If there is no matching command, Bash replies with an event not found error message.

```
$ !x  
bash: !x: event not found
```

A double ! repeats the last command.

```
$ date  
Thu Jul  5 14:03:25 EDT 2001  
$ !!  
date  
Thu Jul  5 14:03:28 EDT 2001
```



Coding Dojo

```
$ date  
Thu Jul  5 14:04:54 EDT 2001  
$ printf "%s\n" $PWD  
/home/kburtch/  
$ !-2  
date  
Thu Jul  5 14:05:15 EDT 2001
```

The `!#` repeats the content of the current command line. (Don't confuse this with `#!` in shell scripts.) Use this to run a set of commands twice.

```
$ date ; sleep 5 ; !#  
date ; sleep 5 ; date ; sleep 5 ;  
Fri Jan 18 15:26:54 EST 2002  
Fri Jan 18 15:26:59 EST 2002
```

•
•
•



Coding Dojo

```
$ history 10
1026 set -o emacs
1027 stty
1028 man stty
1029 stty -a
1030 date edhhh
1031 date edhhh
1032 date
1033 date
1034 !
1035 history 10
```

```
$ history -p !d
history -p date
date
```

⋮



Coding Dojo

```
$ date  
Thu Jul 5 14:12:33 EDT 2001  
$ !?ate  
date  
Thu Jul 5 14:12:38 EDT 2001  
$ !?da? '+%Y'  
date '+%Y'  
2001
```

⋮
⋮
⋮



Coding Dojo

```
$ pwd  
/home/kburtsch  
$ cd .  
$ pwd  
/home/kburtsch  
$ cd ..  
$ pwd  
/home  
$ cd kburtsch  
$ pwd  
/home/kburtsch
```

:

:



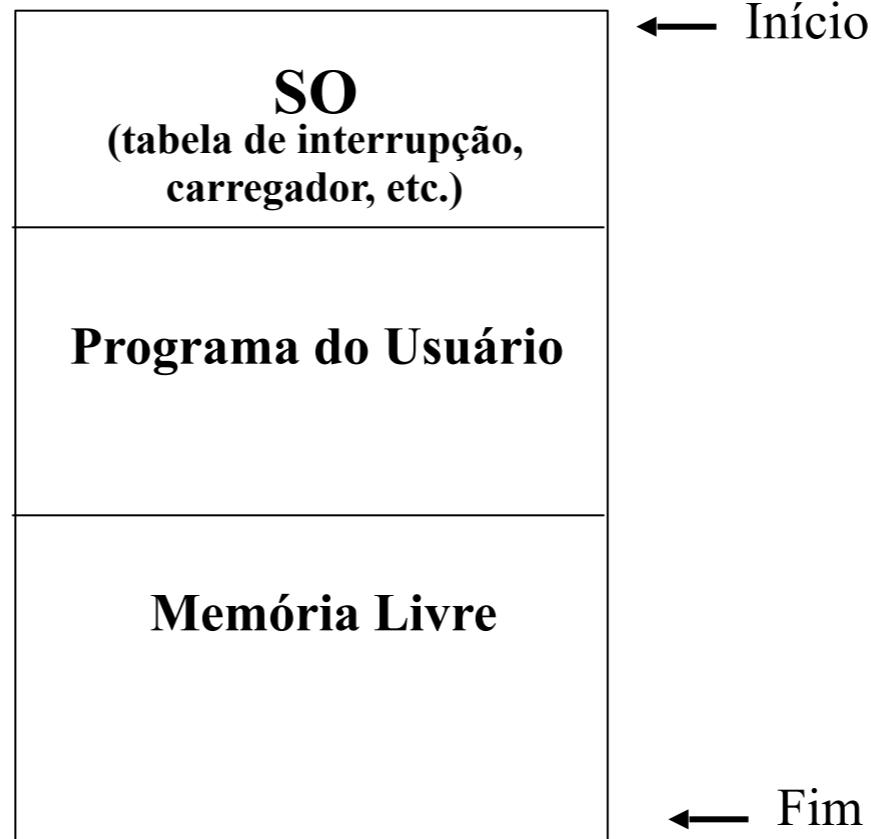
Coding Dojo

```
$ pwd  
/home/kburtch  
$ cd ..  
$ pwd  
/home  
$ cd -  
$ pwd  
/home/kburtch  
$ cd -  
$ pwd  
/home
```



Gerenciamento de Memória

Monoprogramação



Gerenciamento de Memória

Monoprogramação

- Vantagens:
 - simplicidade
 - custo baixo de implementação e uso
 - não ocorrência de *overheads* decorrentes do gerenciamento de memória
 - flexibilidade



Gerenciamento de Memória

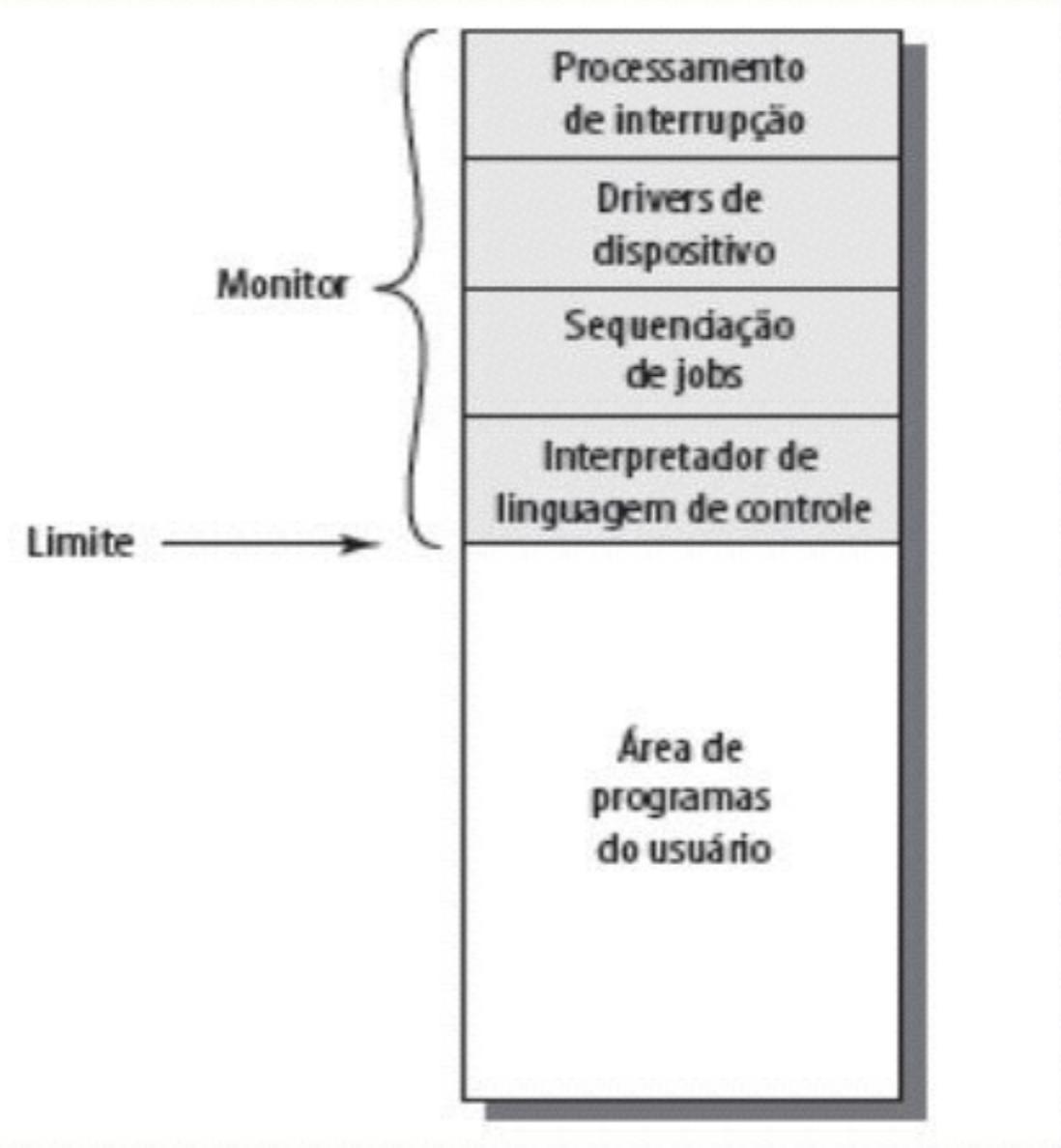
Monitor Residente

- Normalmente, este esquema é usado em sistemas monoprogramados
- Memória dividida em duas partes:
 - área do SO
 - área do usuário
- Registrador limite: contém o primeiro endereço do programa usuário



Gerenciamento de Memória

Monitor Residente



Gerenciamento de Memória

O problema da Relocação

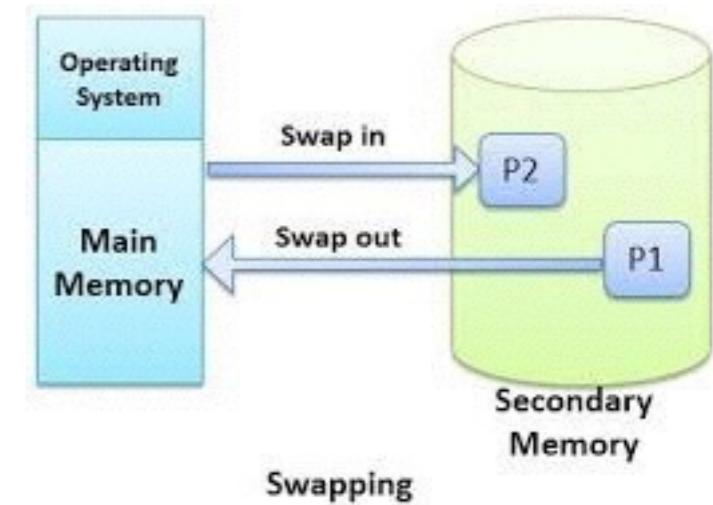
- **Relocação** é a transformação dos endereços relativos do programa em endereços absolutos
- Se o conteúdo do registrador limite é previamente conhecido, os endereços absolutos podem ser gerados na compilação
- Se o endereço inicial do programa só vai ser conhecido no momento da carga, deve haver relocação:
 - **Relocação estática** : realizada pelo carregador
 - **Relocação dinâmica** : os endereços não são modificados, pois usa-se um **registrador base**



Gerenciamento de Memória

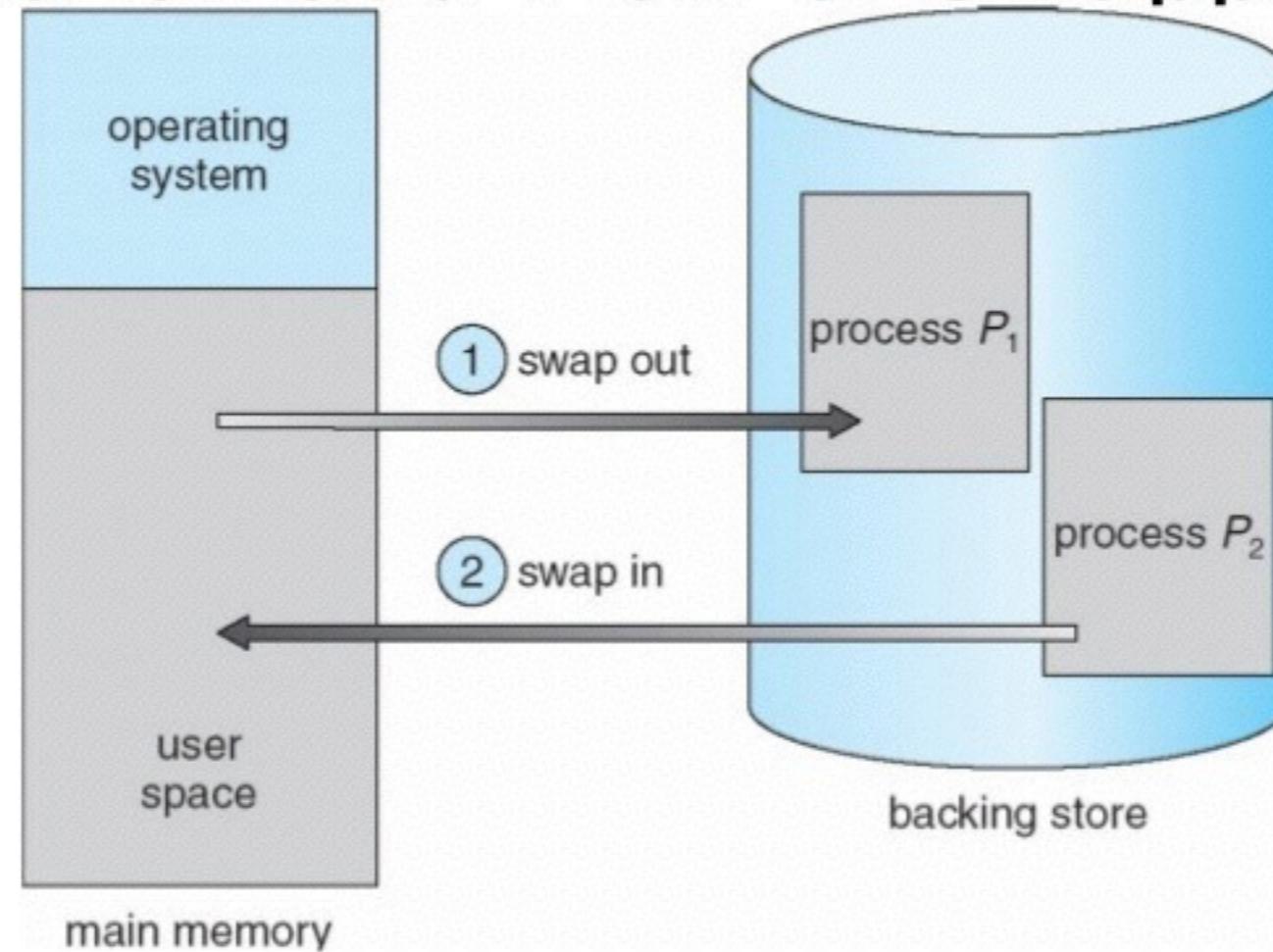
Multiprogramação através de Swapping

- É implementada por um SO do tipo monitor residente
- O esquema de gerenciamento de memória é estendido para implementar **swapping**
- O programa que perde a CPU é copiado p/ disco, enquanto o programa que ganha a CPU é transferido do disco p/ a memória principal



Gerenciamento de Memória

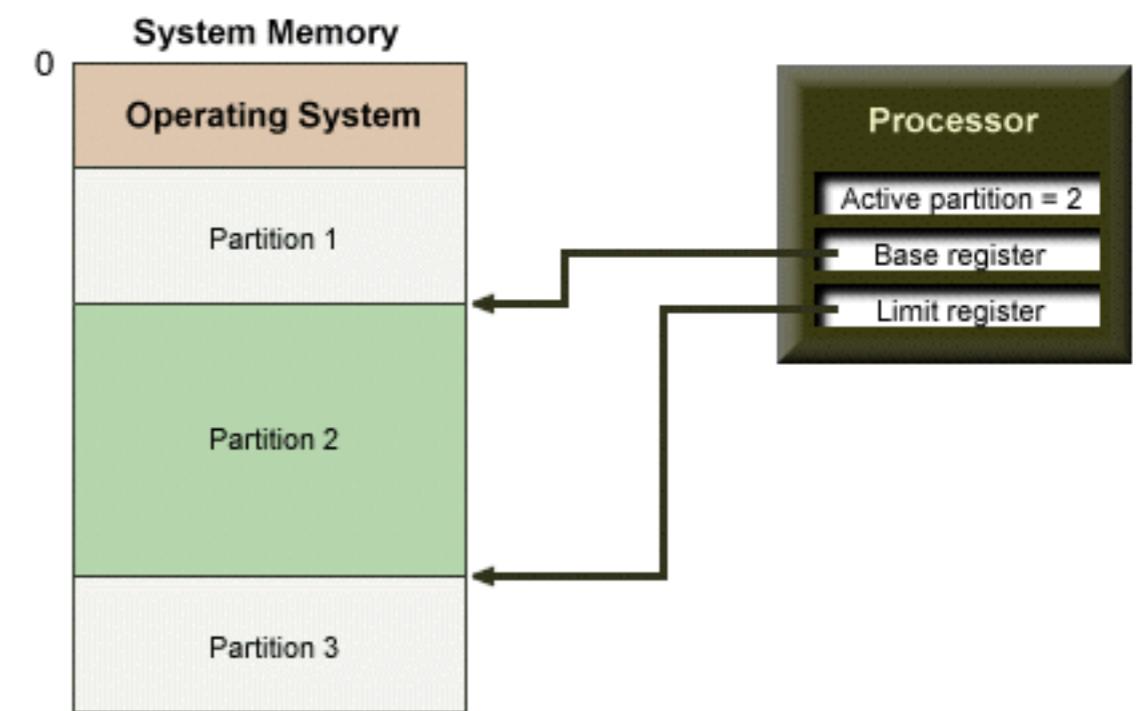
Schematic View of Swapping



Gerenciamento de Memória

Partições Múltiplas

- Com multiprogramação, é conveniente ter vários programas na memória ao mesmo tempo para que a CPU seja rapidamente alternada entre eles
- Solução: dividir a memória em partições (cada partição irá conter um programa)
 - **partições fixas** (normalmente o hw usa registradores **limite inferior** e **limite superior**)
 - **partições variáveis** (normalmente o hw usa registradores **base** e **limite**)





Dúvidas: naubergois@gmail.com

Gerenciamento de Memória



Gerenciamento de Memória

Partições Fixas

- Divide-se a memória em um número fixo de blocos (do mesmo tamanho ou não)
- Quando um processo é criado, ele é colocado em uma fila (em disco) à espera que uma partição de tamanho suficiente se torne disponível

Gerenciamento de Memória

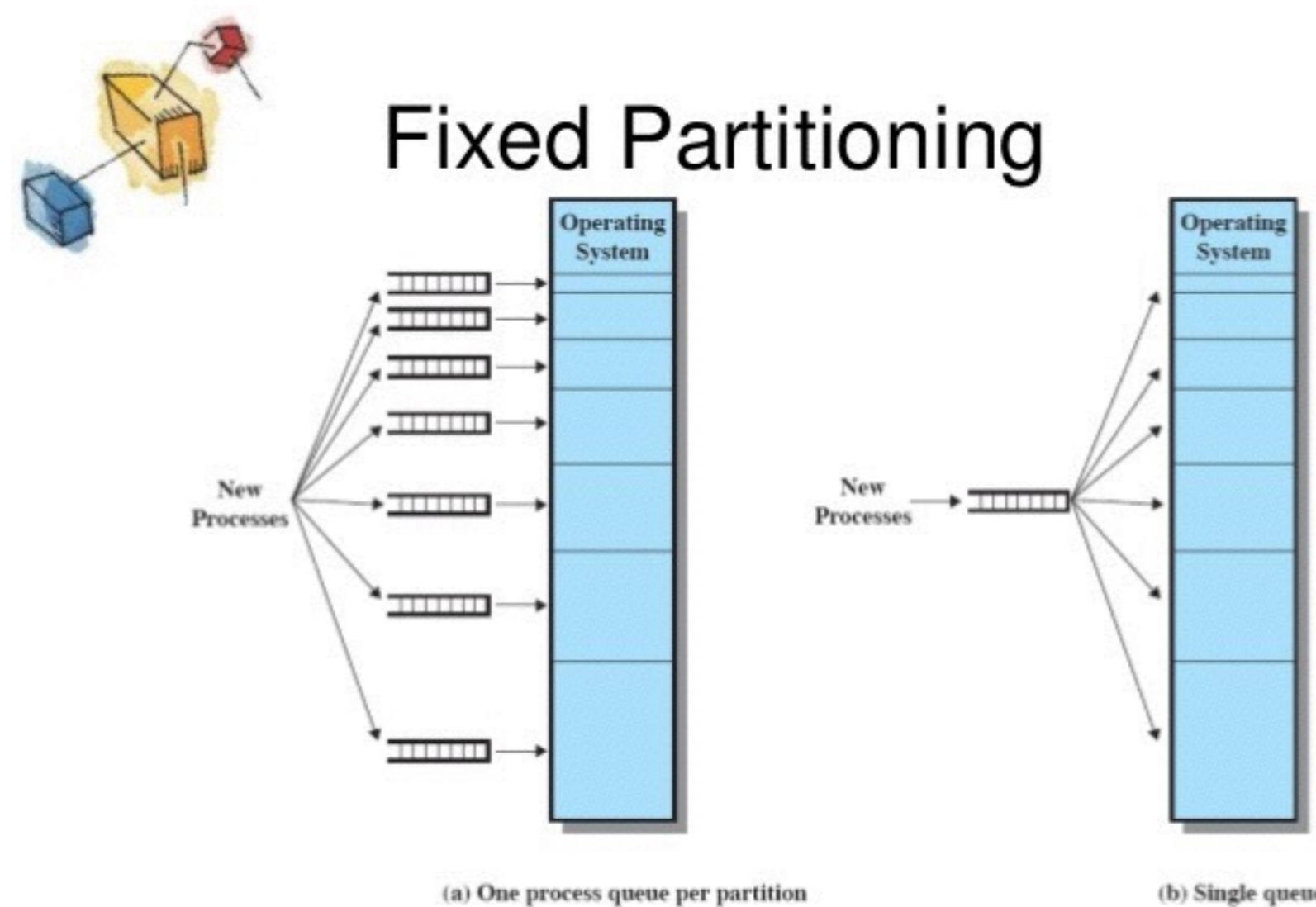


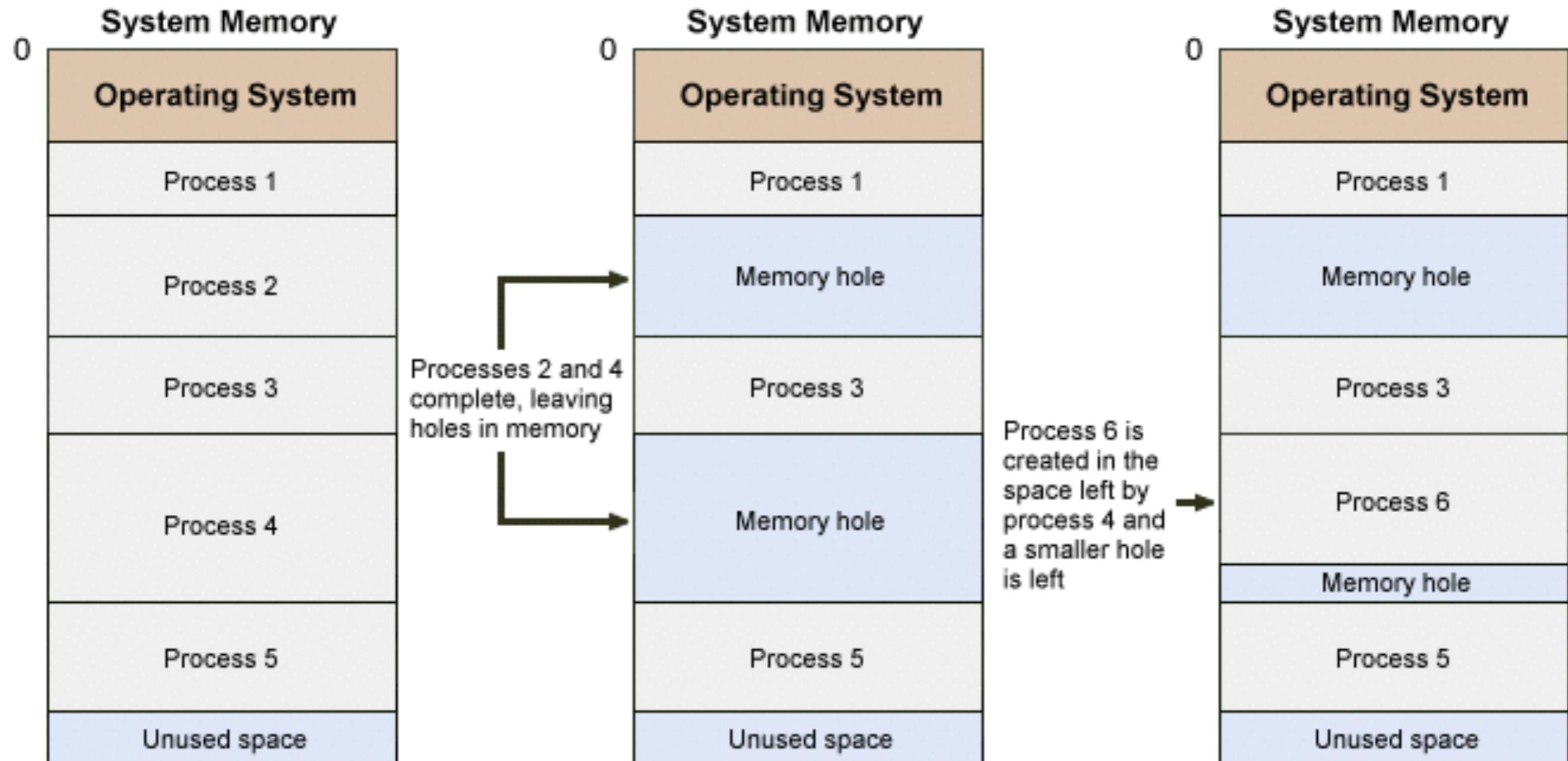
Figure 7.3 Memory Assignment for Fixed Partitioning

Gerenciamento de Memória

Partições Fixas

- Para definir a partição onde o programa vai ser colocado, existem duas opções:
 - Montar uma fila individual para cada partição
 - Montar uma fila única para todas as partições

Gerenciamento de Memória



Gerenciamento de Memória

Multiprogramming with Fixed Partitions (2)

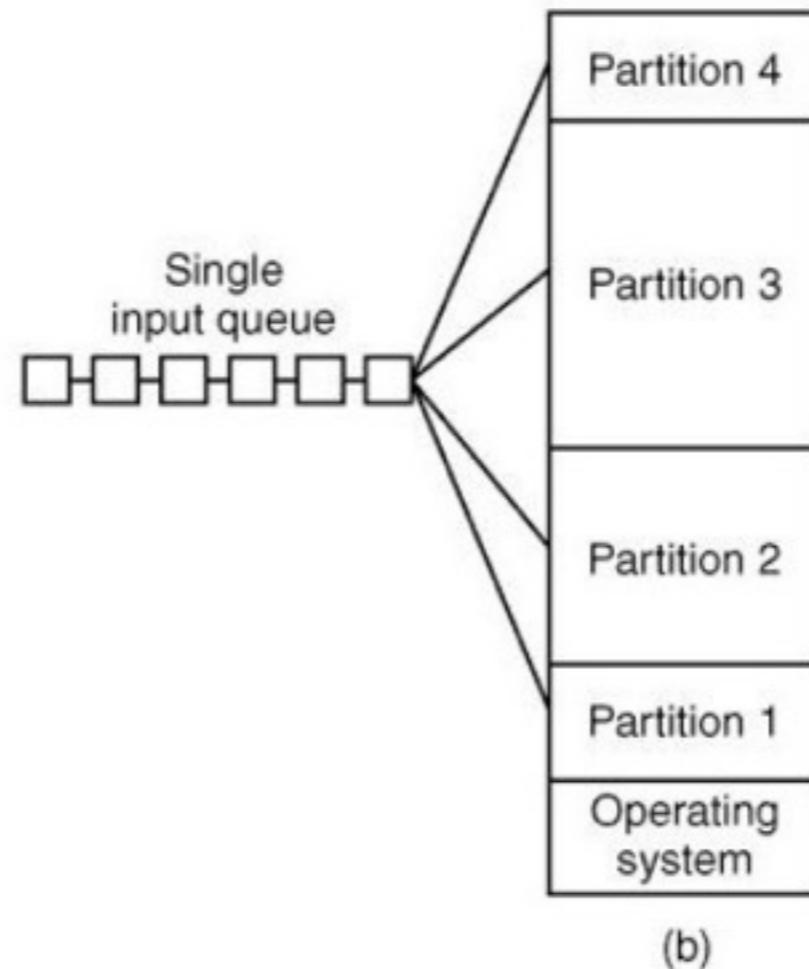


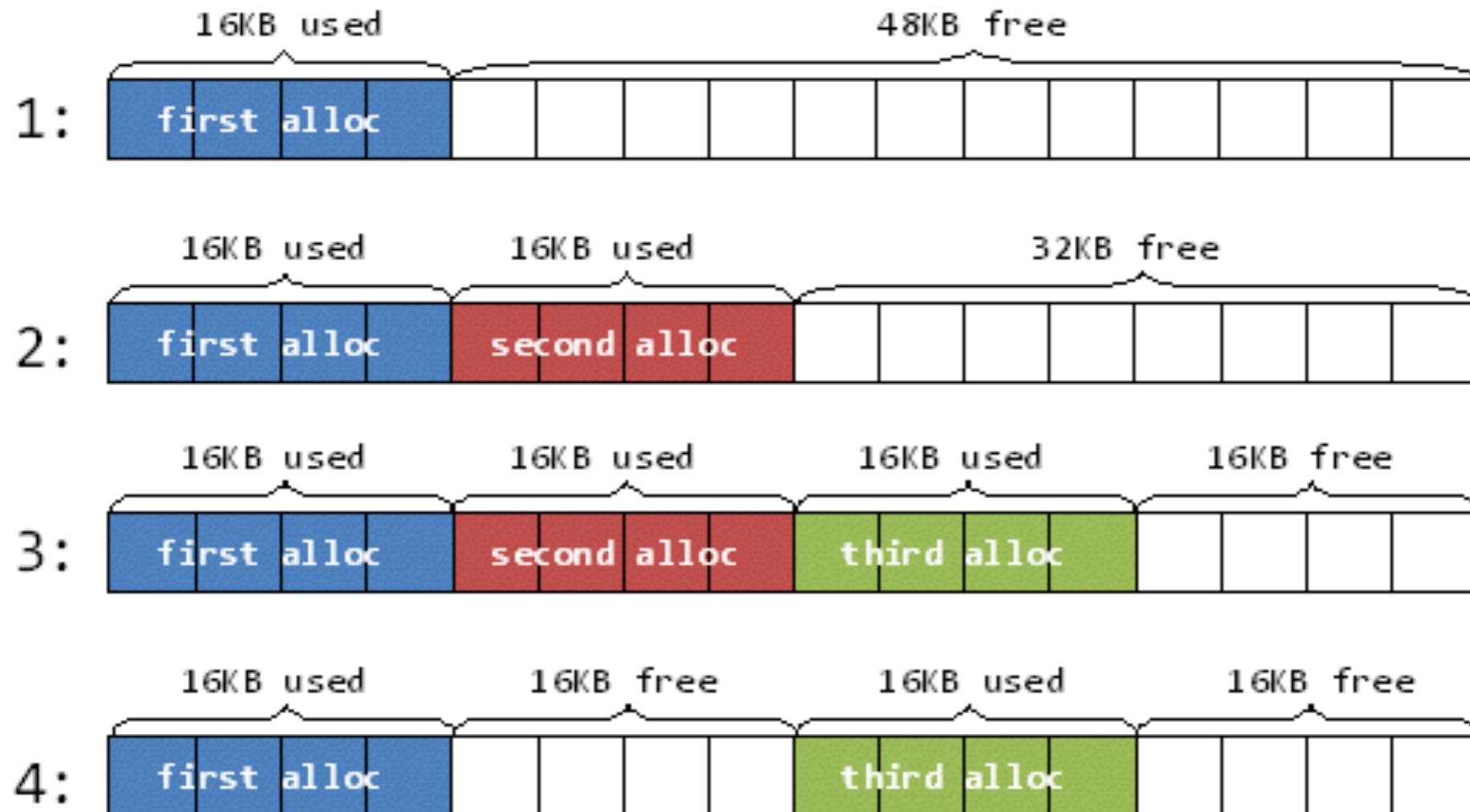
Figure 4-2. (b) Fixed memory partitions with a single input queue.

Gerenciamento de Memória

Fragmentação

- São perdas (desperdício) de memória:
 - **fragmentação interna:** memória é perdida dentro da partição alocada (é um desperdício de espaço dentro da partição usada pelo processo)
 - **fragmentação externa:** ocorre quando existe espaço disponível mas este é pequeno demais para os processos que estão à espera (perda de espaço fora das partições alocadas)

Gerenciamento de Memória



Gerenciamento de Memória

Fragmentation

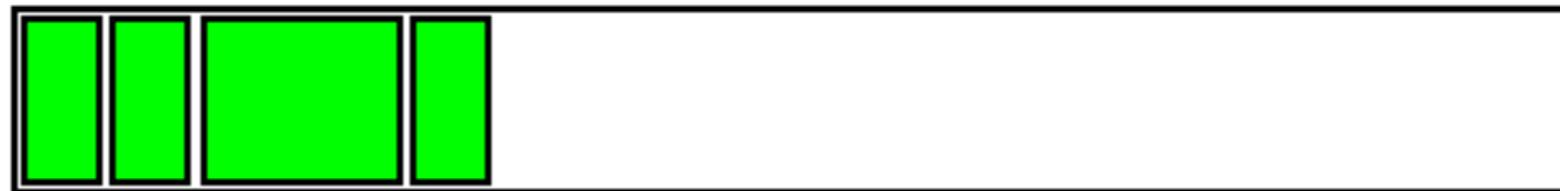


Figure 1.



Figure 2.



Figure 3.



Figure 4.

Coding Dojo

```
$ dirs  
~
```

The built-in `pushd` (*push directory*) command adds (or pushes) directories onto the list and changes the current directory to the new directory.

```
$ pushd /home/kburtsch/invoices  
~/invoices ~  
$ pushd /home/kburtsch/work  
~/work ~/invoices ~  
$ pwd  
/home/kburtsch/work
```



Coding Dojo

```
$ dirs -1  
/home/kburtch/work /home/kburtch/invoices /home/kburtch
```

The `-v` switch displays the list as a single column, and `-p` shows the same information without the list position. The `-c` switch clears the list. The `-N` (*view Nth*) shows the nth directory from the left (or, with `+N`, from the right).

```
$ dirs +1  
~
```

The built-in `popd` (*pop directory*) command is the opposite of the `pushd`. `popd` discards the first directory and moves to the next directory in the list.

```
$ popd  
~/invoices ~  
$ pwd  
/home/kburtch/invoices
```

Coding Dojo

```
$ :  
$
```

The colon command can have parameters and file redirections. This can have strange effects, such as running the date command using backquotes, giving the results to the null command that quietly discards them.

```
$ : `date`  
$
```

This has the same effect as redirecting the output of date to the /dev/null file.

```
$ date > /dev/null
```

Backquotes and redirection are discussed in an upcoming chapter.

•
•
•

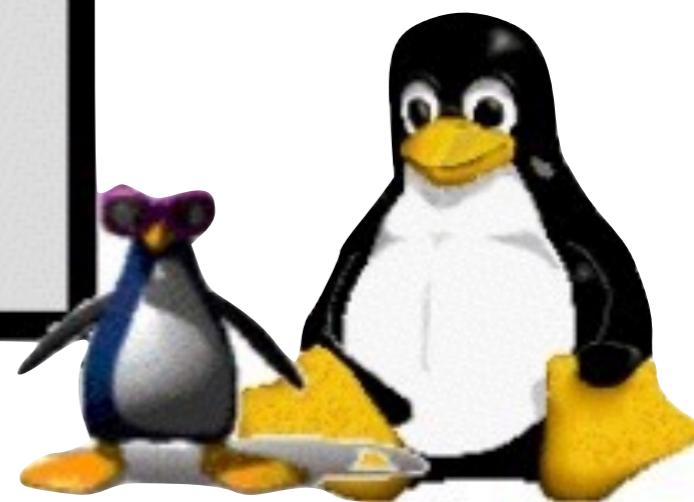
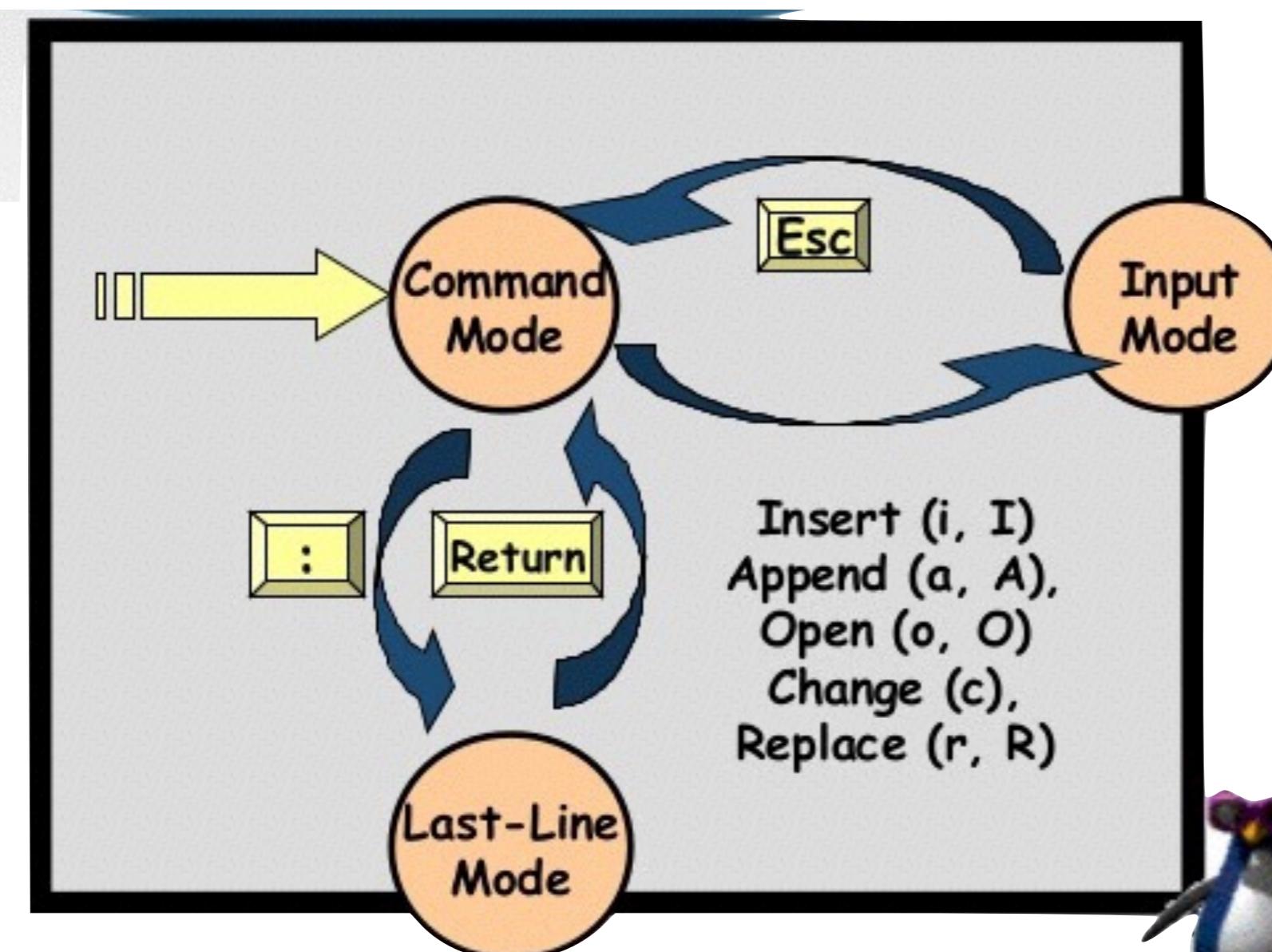
Coding Dojo

```
$ declare -x PS1="Bash $ "
Bash $ pwd
/home/kburtch/archive
Bash $
```

```
$ declare -x PS1="
\$PWD (\$OLDPWD)
\$LOGNAME@'uname -n'\['tput bold'\] \$ \['tput rmso'\]"
/home/kburtch/archive (/home/kburtch/work)
kburtch@linux_box $
```

Utilitários

VI



- **Moving Cursor Position**
- You can move around only when you are in the command mode
- Arrow keys usually works(but may not)
- The standard keys for moving cursor are:
 - **h** - for left
 - **l** - for right
 - **j** - for down
 - **k** - for up
- **w** - to move one word forward
- **b** - to move one word backward
- **\$** - takes you to the end of line
- <**enter**> takes the cursor the beginning of next line

▫ Page 9





Francisco Nauber Bernardo Gois
Email: naubergois@gmail.com