inix: criado por ken thompson em 1968 no bell labs (at&t) . inplementado em assembly e reescrito em c em 1978 por dennis ritchie

linux: criado por linus torvalds em 1991 na universidade de helninky - finlandia

2.gerencia de memoria secundária

2.1 m.s.: memoria não volatil de grande capacidade e baixa velocidade

2.2 midias k7,disquete,hd,cd,dvd,pendrive,ssd, etc

2.3 controladores

ide/ata ⇒ obsoleto

serial ata(sata) 1⇒ obsoleto

serial ata(sata) 2⇒ obsoleto

serial ata(sata) 3⇒ atual

scsi:para servidores ⇒ obsoleto

serial scsi(sas) ⇒ atual

pci express

cache ⇒ back side bus ⇒ ponte norte

cpu ⇒ front side bus ⇒ ponte norte

PCI-e ⇒ ponte norte

video ⇒ ponte norte

ponte norte ⇒ ponte sul

ide ⇒ ponte sul

sata ⇒ ponte sul

usb ⇒ ponte sul

2.4 os dados em m.s. são armazenados em arquivos

definição : conjunto de dados estruturado

caracteristicas :

* nome
* tamanho
* estrutura
* extensão (tipo)
* path
* metadados
* icone

operações:

* ler
* executar
* alterar conteudo
* excluir
* mover
* copiar
* alterar proprietario
* renomear

2.5 diretórios

* forma como os arquivos são organizados
* geralmente arvore

arvore windows:multi arvores orientadas a volumes: so de redes ou seja é explicito se há pastas compartilhadas de rede

arvore linux: unica arvore independente da origem dos diretórios (so distribuido)

2.6 linhas de comando

* caracteres curingas
  + \* substitui qualquer cadeia de caracteres
  + ? substitui 1 ou nenhum caractere

arquivos:

| comando | windows | UNIX |
| --- | --- | --- |
| copiar | move origem destino | cp origem destino |
| mover | move origem destino | mv origem destino |
| deletar | del origem destino | rm origem destino |
| renomear | rename origem destino | mv origem destino (nome é alterado no destino) |

pastas:

| comando | win | unix |
| --- | --- | --- |
| criar | mkdir | mkdir |
| atual | exipe automaticamente | pwd(print world directory) |
| listar | dir | ls |
| navegar | cd | cd |
| apagar | rmdir | rm -r |
| copiar | copy | cp |
| mover | move | mv |
| renomear | rename | mv |

pastas especiais:

./ ⇒ atual

.. ⇒ pasta a cima

~ ⇒ pasta do usuario atual

c: ⇒ raiz do sistema windows

/ ⇒ raiz sistema linux

observações: root= super usuário virtual

passwd + %user% = altera a senha do usuário

sudo = grupo de super usuário

add group %user% sudo

quatro grupos de usuário :

criador ⇒ owner = (u) criador do recurso

grupo ⇒ groups (g) = grupo que o criador faz parte

outros ⇒ others (o)= outros usuários

todos ⇒ all (a)=todos os usuários

cada recurso está associado a uma trinca de operações e grupos de usuários

u=RWX

g=RWX

o=RWX R=read W=write X=execução

listando as permissões ls -l

dados.txt [rw\_] [r\_ \_] [\_ \_ \_]

{ RW p/ u} {R p/ g} {nada p/ o}

alteração de permissão

comando : chmod permissão recurso

2° formato : trinca de octal

| permissão | bin | octal |
| --- | --- | --- |
| \_ \_ \_ | 0 0 0 | 0 |
| \_ \_ x | 0 0 1 | 1 |
| \_ w \_ | 0 1 0 | 2 |
| \_ w x | 0 1 1 | 3 |
| r \_ \_ | 1 0 0 | 4 |
| r \_ x | 1 0 1 | 5 |
| r w \_ | 1 1 0 | 6 |
| r w x | 1 1 1 | 7 |

outros comandos:

sort -> ordenação de string

sort -g -> ordenação numérica

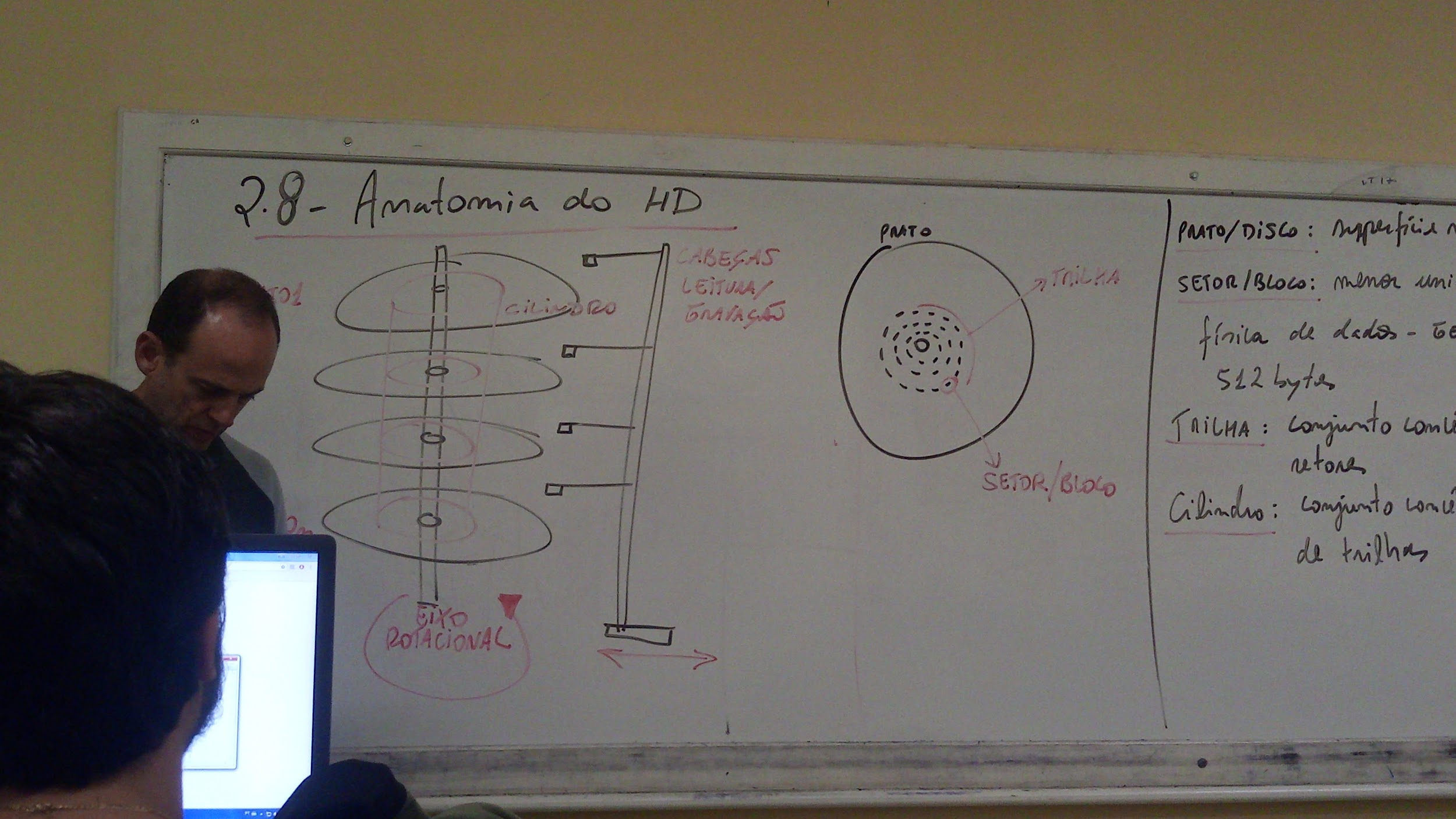
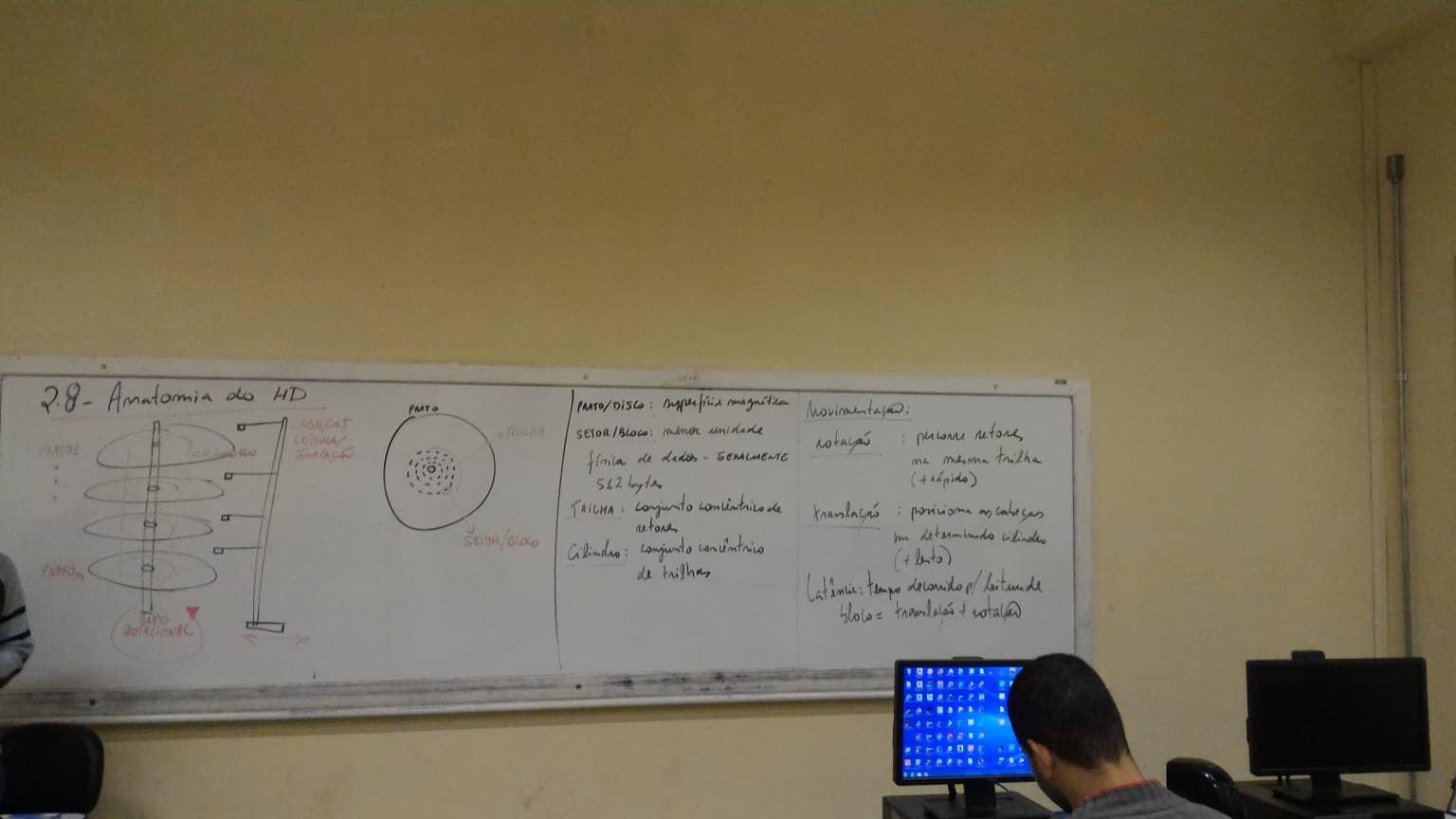
sort -g -r -> ordenação inversa

cal -> calendario

date -> data/hora

time -> tempo p/ execução do comando

2.8 anatomia hd



prato/disco: superficie magnetica

setor/bloco: menor unidade

física de dados: geralmente 512 bytes

trilha : conjunto concentrico de setores

cilindro : conjunto concentrico de trilhas

cluster : conjunto de 1 ou mais blocos menor unidade lógica de armazenamento

movimentação:

rotação: percorre vetores na mesma trilha(mais rapido)

translação : posiciona as cabeças em determinado cilindro(mais lento)

latencia : tempo decorrido p/leitura de bloco= translação+rotação

obs: o tamanho dos blocos e a quantidade de trilhas é definida na formatação física , realizada pelo fabricante .

a definição do tamanho dos clusters é definida na formatação lógica feita pelo so

ex: salvar a .doc de 2 kb

clusters: mínimo : 512 bytes

| fat | | hd | | | |
| --- | --- | --- | --- | --- | --- |
| file | cluster |
| a | 0 | A | A | A | A |
| a | 1 |  |  |  |  |
| a | 2 |  |  |  |  |
| a | 3 |  |  |  |  |

perda 0

cluster de 4kb:

| fat | | hd |
| --- | --- | --- |
| file | cluster |
| A | 0 | A A A A |
|  |  | livre |
|  |  | ... |

perda de 2kb

| cluster | vantagem | desvantagem |
| --- | --- | --- |
| pequeno | menos desperdicio | facilita a fragmentação externa o que faz ser mais lento |
| grande | velocidade nas operações pois a fat é menor o que diminui o tempo de busca pelos clusteres de arquivos | desperdicio de espaço = fragmentação interna |

obs:quando formatamos uma partipção podemos escolher o tamanho do cluster , ou deixar que o so faça de forma proporcional ao tamanho do hd

obs2: fragmentação interna :desperdiciode espaço no ultimo cluster de um arquivo

obs3:fat=file allocation table,tabela dos clusters dos arquivos

setor de book:

mbr ou bpt(SOs modernos)

1° setor dos 1° utilizadores do hd

contém : bootrap:carregador do so (grub por exemplo)

tabela de partições

tabelas fat

obs sequencia de boot:

1. cpu executa o post da bios(rom) e verifica a integridade do hardware
2. cpu executa o setup para ler e alterar configurações do hardware
3. cpu le o mbr/gpt e executa o bootstrap
4. o bootstrap carrega o so

obs : fragmentação de arquivos é na verdade a dispersão dos clusters pelo hd

ex:gravar

A1.doc 3 cluster A2 1 cluster A3 2 cluster

| fat | | | hd | | | |
| --- | --- | --- | --- | --- | --- | --- |
| A | C | R |
| A1 | 0 | 0 | A1 | A1 | A1 | A2 |
| A1 | 1 | 0 | A3 | A3 |  |  |
| A1 | 2 | 0 |  |  |  |  |
| A2 | 3 | 0 |  |  |  |  |
| A3 | 4 | 0 |  |  |  |  |
| A3 | 5 | 0 |  |  |  |  |

A= arquivo c= cluster r = removido

remover A2.doc

| fat | | | hd | | | |
| --- | --- | --- | --- | --- | --- | --- |
| A | C | R |
| A1 | 0 | 0 | A1 | A1 | A1 | A2 |
| A1 | 1 | 0 | A3 | A3 |  |  |
| A1 | 2 | 0 |  |  |  |  |
| A2 | 3 | 1 |  |  |  |  |
| A3 | 4 | 0 |  |  |  |  |
| A3 | 5 | 0 |  |  |  |  |

gravar A4.doc 4 cluster

| fat | | | hd | | | |
| --- | --- | --- | --- | --- | --- | --- |
| A | C | R |
| A1 | 0 | 0 | A1 | A1 | A1 | A4 |
| A1 | 1 | 0 | A3 | A3 | A4 | A4 |
| A1 | 2 | 0 | A4 |  |  |  |
| A4 | 3 | 0 |  |  |  |  |
| A3 | 4 | 0 |  |  |  |  |
| A3 | 5 | 0 |  |  |  |  |
| A4 | 6 | 0 |  |  |  |  |
| A4 | 7 | 0 |  |  |  |  |
| A4 | 8 | 0 |  |  |  |  |

ex: uma partição de 1 tb, em uma arquitetura (hardware e so) de 64 bit , em uma partição possui 1000 arquivos.

calcule: o desperdício de espaço no caso médio (o ultimo cluster usa metade do espaço)e no pior caso(ultimo cluster usa apenas o minimo de espaço,que no caso seria de 64bit) em :

1. formatação com cluster de 64k

1000\*32\*1024=32768000[médio caso]

1000\*[(64\*1024)-(64/8)]=65528000[pior caso]

1. formatação com cluster de 512 bytes(mínimo)

1000\*256=256000[médio caso]

1000\*[512-(64/8)]=504000[pior caso]

desperdício = =65528\*1000=62.5mB

obs:=fragmentação interna máxima

=número de arquivos

caso médio:=

2.9 escalonamento de disco

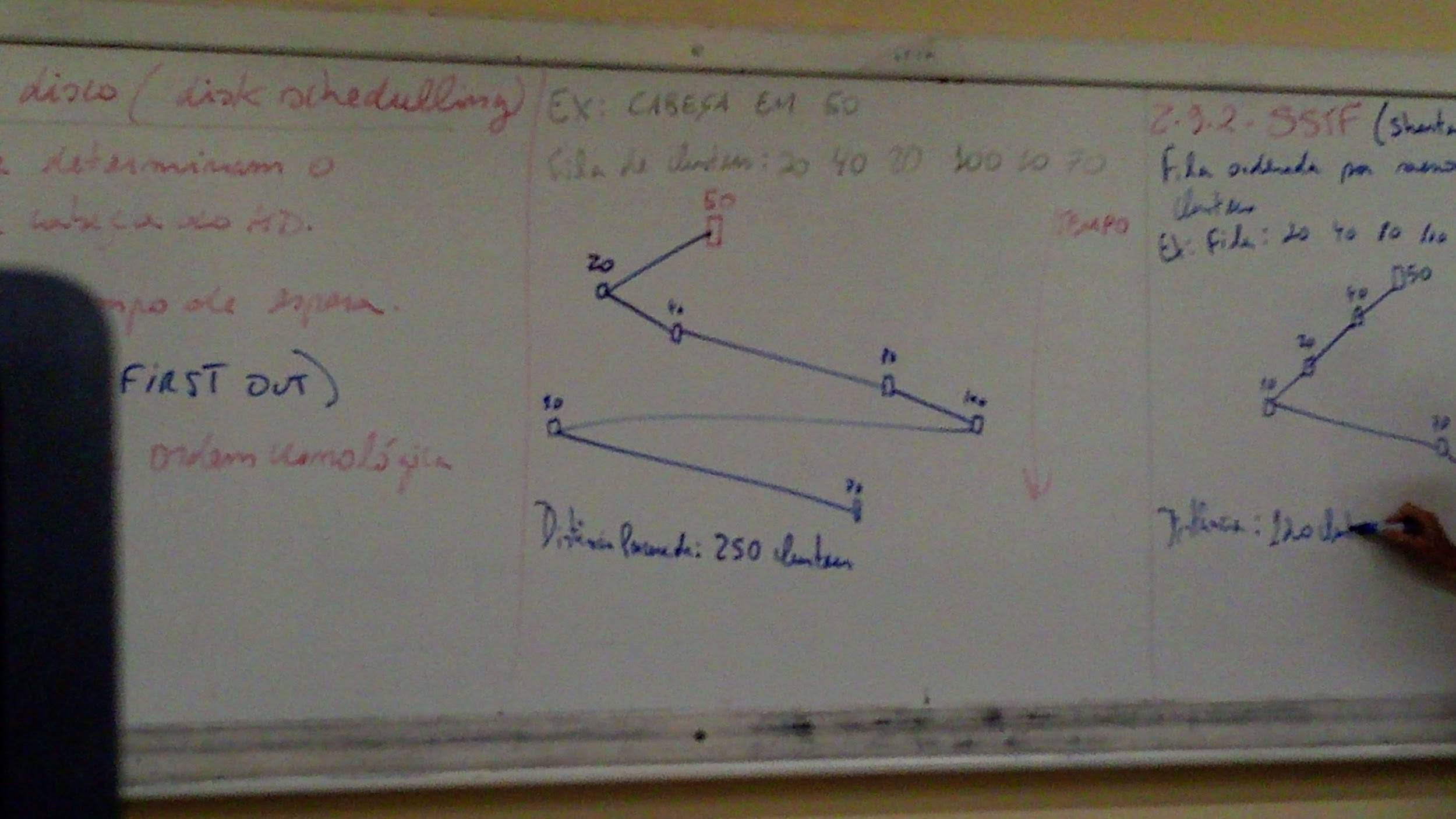
são algoritimos que determinam o comportamento da cabeça do hd

objetivo: minimiza tempo de espera

2.9.1- fifo(first in first out)

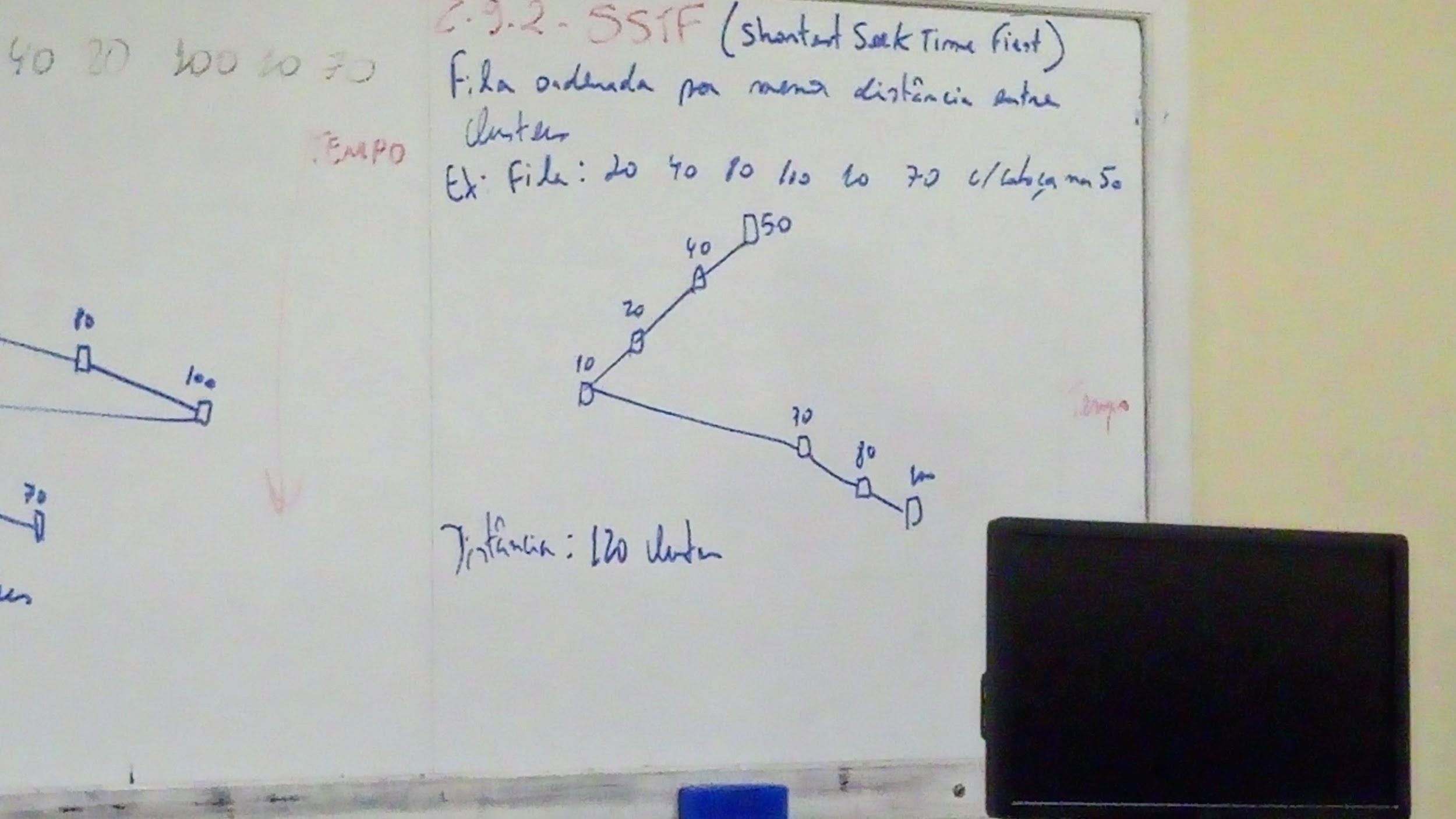
cluster visitados na oredem cronológica de requisição ex: cabeça em 50

fila de cluster:20 40 80 100 10 70



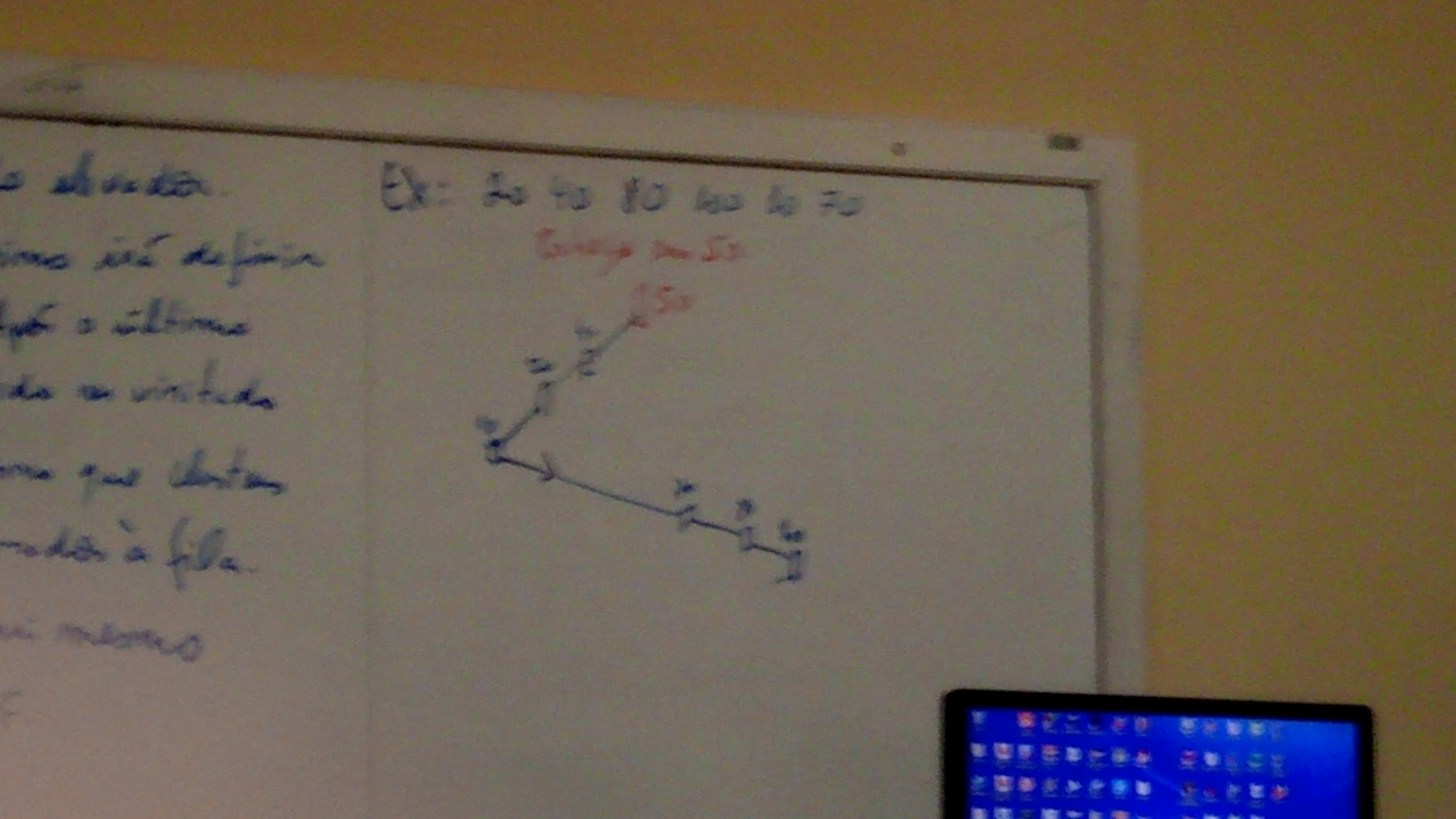
2.9.2-sstf(shortest seek time first)

fila ordenada por menor distância entre cluster



problema de starvation,pode achar um otimo local e nunca sair dele fazendo com que nunca acesse os arquivos de uma região do hd

scan:algoritimo do elevador 0 1° cluster mais próximo irá definir o sentido da leitura.após o ultimo cluster ao fim do sentido ser visitado ele altera o sentido,mesmo que clusters próximos sejam adicionados à fila. dependendo do caso ,possui mesmo comportamento do sstf



c-scan - scan unidirecional:varre sempre

em um unico sentido

no linux :

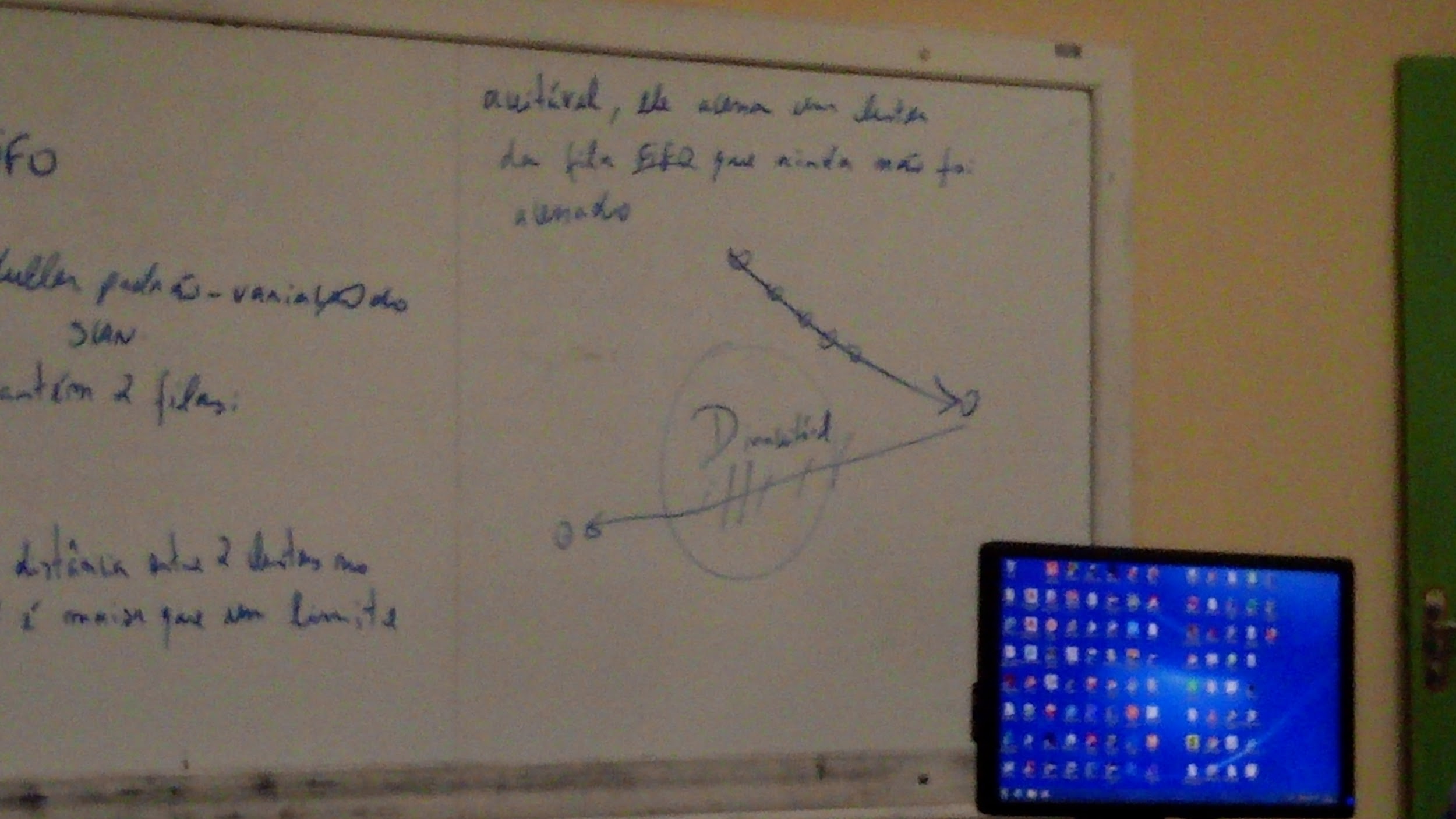
-moop:fifo

-cfg - scheduller padrão-variação do scan

deadline: mantém 2 filas:

* fifo
* scan

quando a distancia entre 2 clusters no scan é maior que um limite aceitável , ele acessa uma lista de fila FIFO que ainda não foi acessado



para visualizar o scheduller em ação:

cat /sys/block/[device]/queue/scheduler

[device] = partição

para alterar :

sudo echo [scheduller]>/sys/block/device/queue/scheduller

[scheduller] = moop/cfg/deadline

ssd(solid state drive)

nova tecnologia de armazenamento em massa

evoluiu das antigas memórias rom

atualmente , tecnicamente , é uma memória FLASH-ROM rom (read only memory) é uma memória projetada para apenas leitura

essa caracteristica histórica , aponta para a deficiencia dessa tecnologia:as operações de leitura são mais eficientes que a escrita.

a escrita de dados em um ssd diminui seu tempo de vida

não se deve realizar formatação completa em um ssd por esse motivo

obs: pendrive ,memoria sd , microsd,e outras memórias flash possuem caracteristicas semelhantes

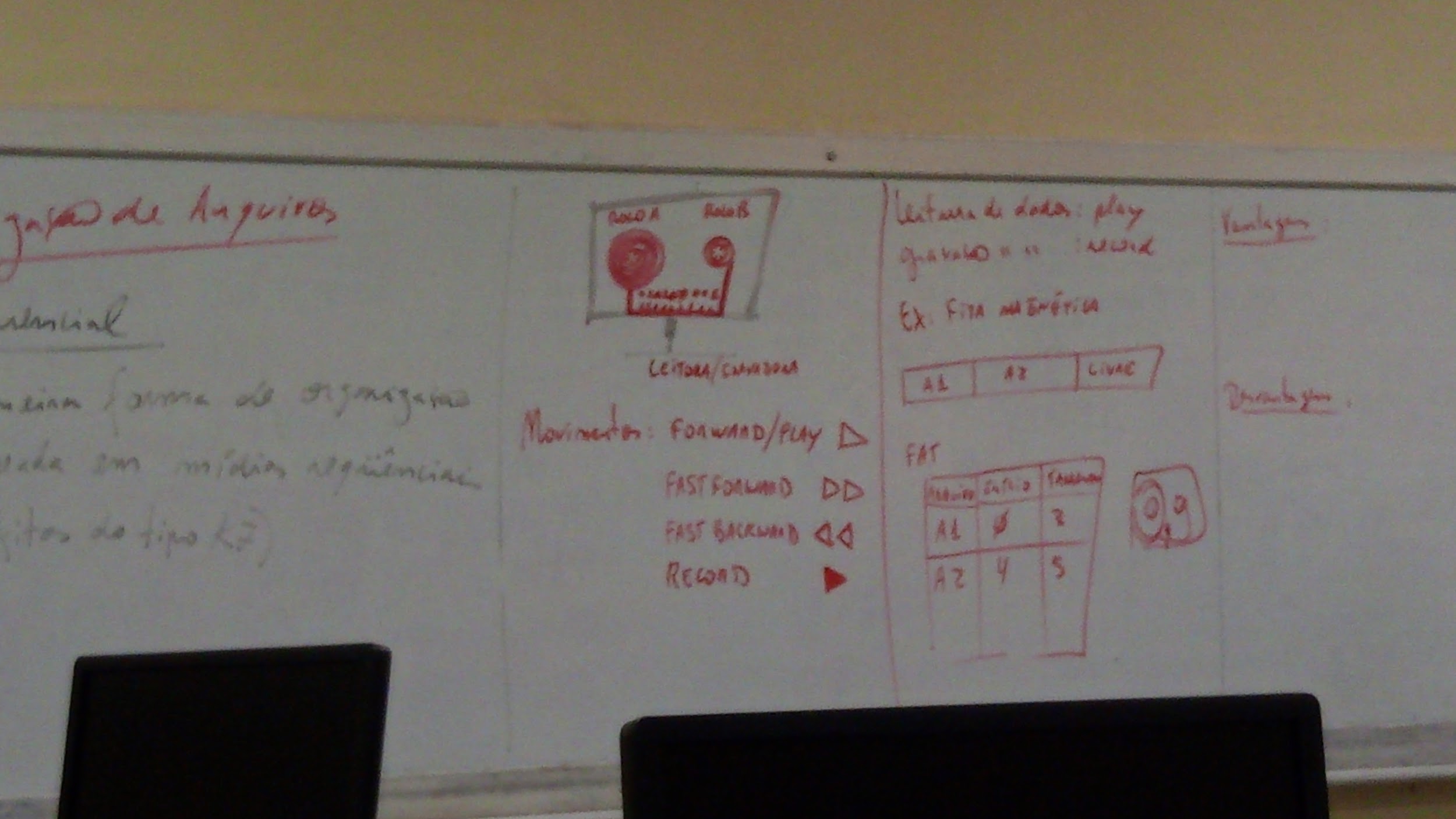
obs2:o acesso aos dados no ssd é direto sem diferenças significativas de tempo de acesso

filesystem:

organização de arquivos:

1. sequencial

primeira forma de organização baseada em mídias sequenciais(fitas do tipo k7)



* pros
  + simplicidade
  + não existe fragmentação interna
* contras
  + dificuldade para realocação de espaço para arquivos
  + tempo extenso de acesso devido a busca sequencial
  + fragmentação externa:área livre na midia que não foi aproveitada
  + arquivos não realocaveis,a menos por uma operação extremamente custosa

1. ligada(linked)

nessa tecnologia ,o arquivo é separado em clusters , onde um cluster aponta para os outros

ex:fat

* + pros
    - não existe fragmentação externa
    - fat pequena,ou seja o inicio do arquivo é rapidamente encontrado pois tem apenas um registro por arquivo na fat
  + contras
    - não pode fazer busca binária
    - a lista encadeada força a utilização de busca sequencial ou seja para se acessar um segmento é necessário percorrer todos os anteriores

1. indexada

também separa o arquivo em segmento (clusters) e mantém tabela de indices de cada segmento

3 gerncia de memória principal

componentes da m.p.

ram- random acess memory: componente principal da mp, ao ocntrário da fita magnética sequencial , os dados podem ser alocados em qualquer posição (random) - (volatil)

-registradores da CPU: pequena memória volátil onde os dados são processados (1kb na mips) {não podem ser mto grandes pois aumenta a quantidade de registradores necessários para serem salvos os endereços de cada dado dos registradores}

caches l1,l2,l3: memórias de alto desempenho que exploram a localidade temporal dos dados ,ou seja,as caches mantém os dados utilizados recentimente para nãp ter que ir até a ram novamente

as caches podem atualizar a ram de varias formas , dentre elas :

write through:assim que o dado é alterado na CPU(sw) , todos os niveis de cache e a ram são alterados

vantagens: consistencia dos dados ,principalmente para caches dedicadas à nucleos em sistemas multicores

desvantagens: gera muito tráfego no frontside bus e processamento extra na ponte norte

writeback atualiza somente as caches ,a ram é atualizada somente quando o dado é substituido nos caches ou quando é requisitado pela cpu

pantagens : maior velocidade pois gera menos tráfego de dados

desvamtagens : menor segurança quanto a integridade dos dados

rom: responsável pela inicialização da máquina(bios)

cmos:pequena memória volátil alimentada pela bateria da placa mãe armazena os dados do setup bios

3.1: organização da ram

3.1.1partição unica: somente 1 processo em execução . sistema mais antigo.

vantagem : simplicidade

desvantagem: desperdício de memória

fragmentação externa

3.1.2 partições fixas:ram dividida arbitráriamente em pedaços

3.1.2.1fila unica:

vantagens: permite a multiprogramação e elimina a fragmentação externa

desvantagens: ocorre fragmentação interna e starvation de processos esperando por uma partição que caiba

3.1.2.2-filas múltiplas: o so separa os processos por tamanho adequados para cada partição:

vantagem: ameniza o starvation

3.1.3 partições variáveis

o so particiona dinamicamente a ram , de acordo com a fila de processos

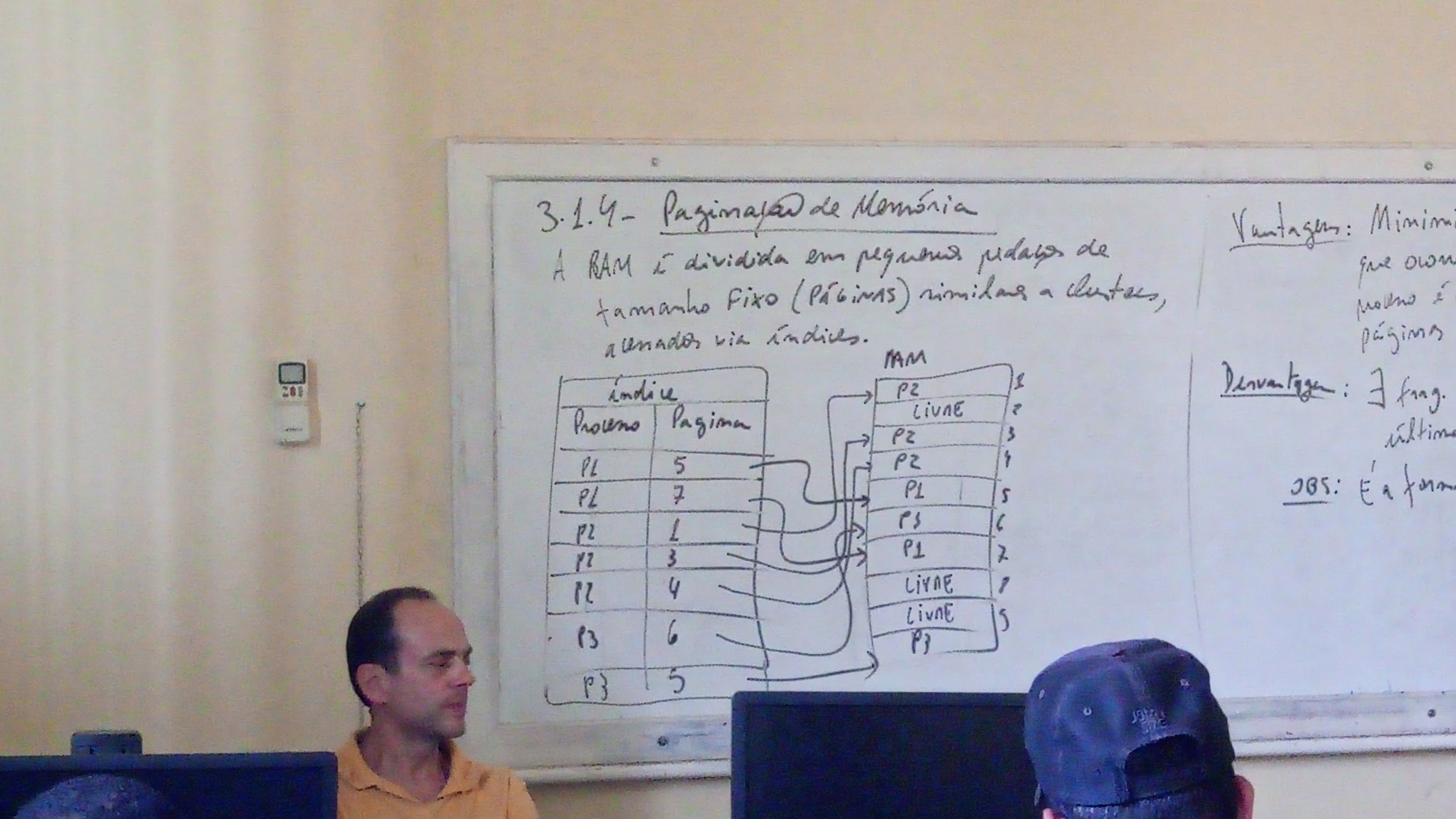
vantagens: sem fragmentação interna

desvantagem: fragmentação externa,porém menor que com partição única

pode haver starvation,ou seja um processo ter que esperar pela liberação de 1 ou mais processos

3.1.4 paginação de memória

a ram é divididaem pequenos pedaços de tamanho fixo (páginas)similares a clusters ,acessados via indices



vantagens : minimiza fragmentação externa,que ocorre somente quando um processo é maior que a soma das paginas livres

desvantagens:não existe fragmentação pequena somente na pagina

obs: é a forma atual de gerencia de ram

3.1.5 segmentação

a nam é subdividida de acordo com os modulos de um programa nam

forma mais natural de divisão da memória

características:

não existe fragmentação interna

não existe fragmentação externa , caso bloco livre de memória não suporte o tamanho de um segmento

inviável em programas sem módulo , pois gera segmentos persistentes muito grandes o que pode causar fragmentação externa

3.1.6 segmentação com paginação

resolve o problema de fragmentação externa

memoria virtual(swap)

estratégia utilizada quando não há memória ram suficiente para um novo processo

quando um processo requisita memória com a num estando cheia uma página da ran é salva em disco e é liberada para novo vso (interação swapout )

quando um processo solicita acesso a uma página arquivada no hd é gerada uma interrupção page fault(swap in)

nessas interrupções o controle é dado ao sistema operaciinal para realizar a gravação em disco (swap out) ou a leitura de disco para a nam(swap in) alem da atualização da tabela de m.v.

quando o soprecisa fazer swap out , ele deve escolher uma página para “arquivo morto” há varias estratégias (page replacement algorithm)(algoritimo de substitituição de páginas)

a)ótima : escolhe a página que não mais será acessada no futuro .não é possivel implementa-lo

pois não é viável antecipar o comportamento futuro de um programa

é usado apenas como benchmark para comparação de performance com outros algorítimos.

b)aleatório:substitui uma página aleatória qualquer . possui performance ruim

c)FIFO: substitui a página mais antiga

apresenta performance ruim caso o programa utilize muitas variáveis globais que são utilizadas ao longo de todo o programa

ex:

int x=7; → página antiga

longo tempo…

x=9; → page fault

longo tempo …

x-10; → page fault

d)LFU(least frequent used)

menos usado frequentemente

um contador por página é incorporado cada vez que a página é incrementado cada vez que a página é acessada .

A página com menor contador é escolhida.

não é usado na prática , pois uma variável “nova”,com contador baixo ,pode ser arquivada mesmo se ela tiver uso intenso a frente

ex: int x=7;

x++;

x--;

//adiante x não será mais usado

for(i=0;i<1000;i++)

caso na 1° iteração do for , a variavel i será arquivada , mesmo sendo crucial para o desempenho do programa

e) nru (not recently used)

de tempos em tempos , toda página em uso é setada para 1 , as outras são geradas .

qualquer página gerada pode ser arquivada

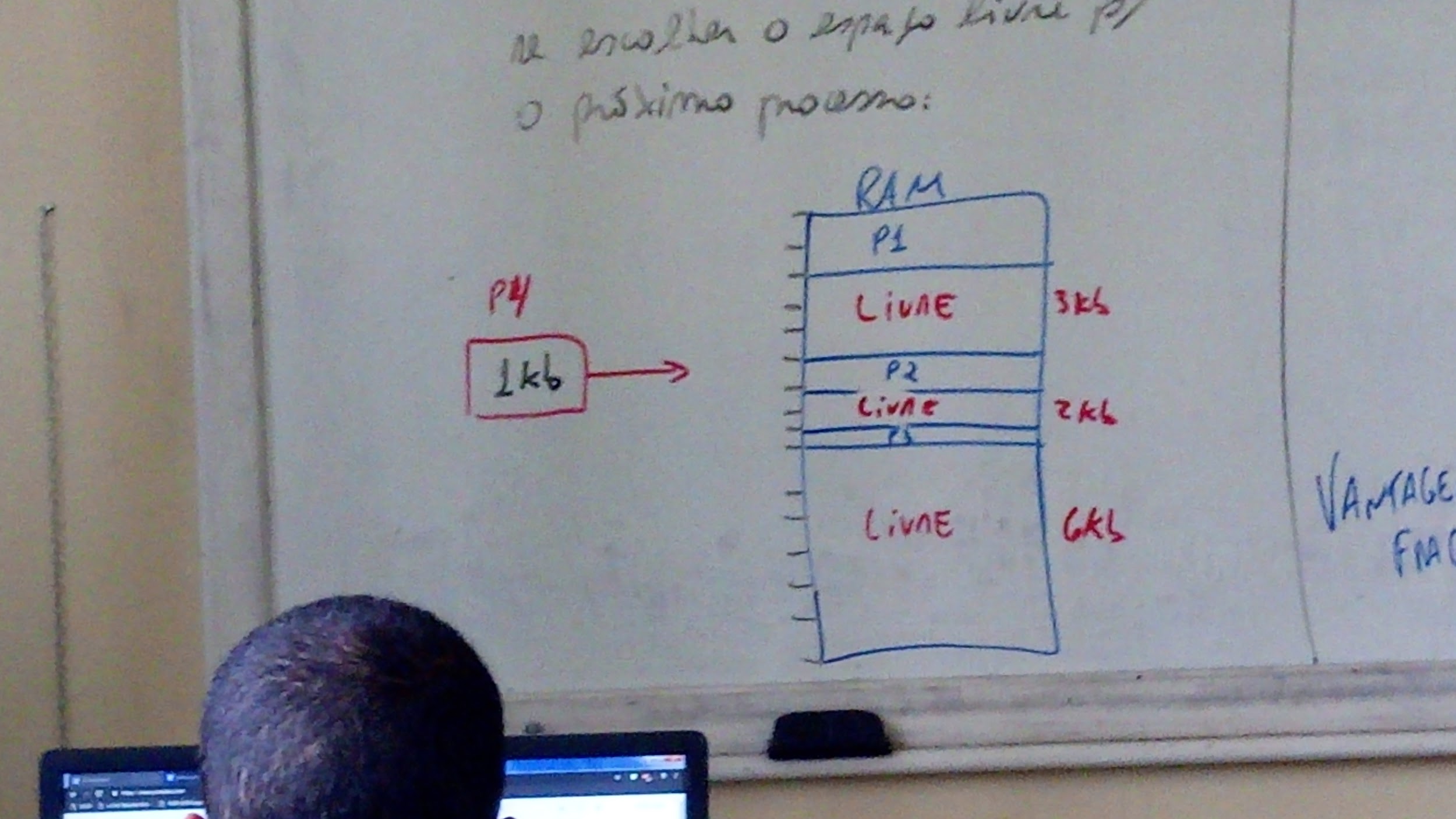
Desse modo , explora-se a localidade temporal ,ou seja, variáveis locais localizadas na mesma página não preservadas na nam

outras variáveis mais sofisticadas desse algoritimo podem usar outros valores de prioridade como penultima ou antipenultima págiba acessada,o que diminuiria page-faults,mas sobrecarrega o processamento.

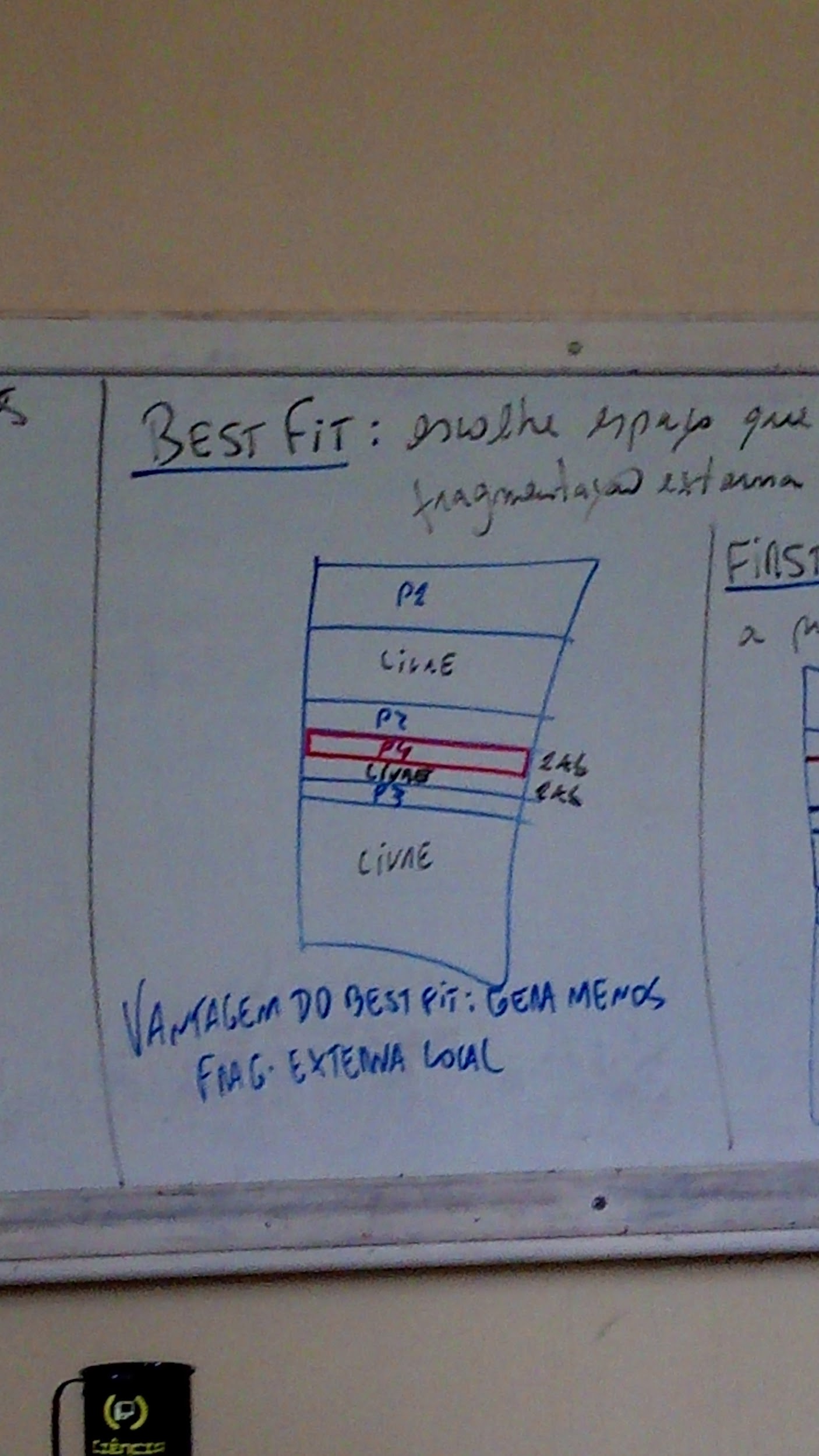
trashing:excesso de page-faults,gerando lentidão

causas:

na organização em partições variaveis ,há 3 formas de se escolher o espaço livre para o próximo processo:



best fit:escolhe espaço que gera menos fragmentação externa total



vantagem do best fit:gera menos fragmentação externa total

first fit : escolhe a primeira área livre :

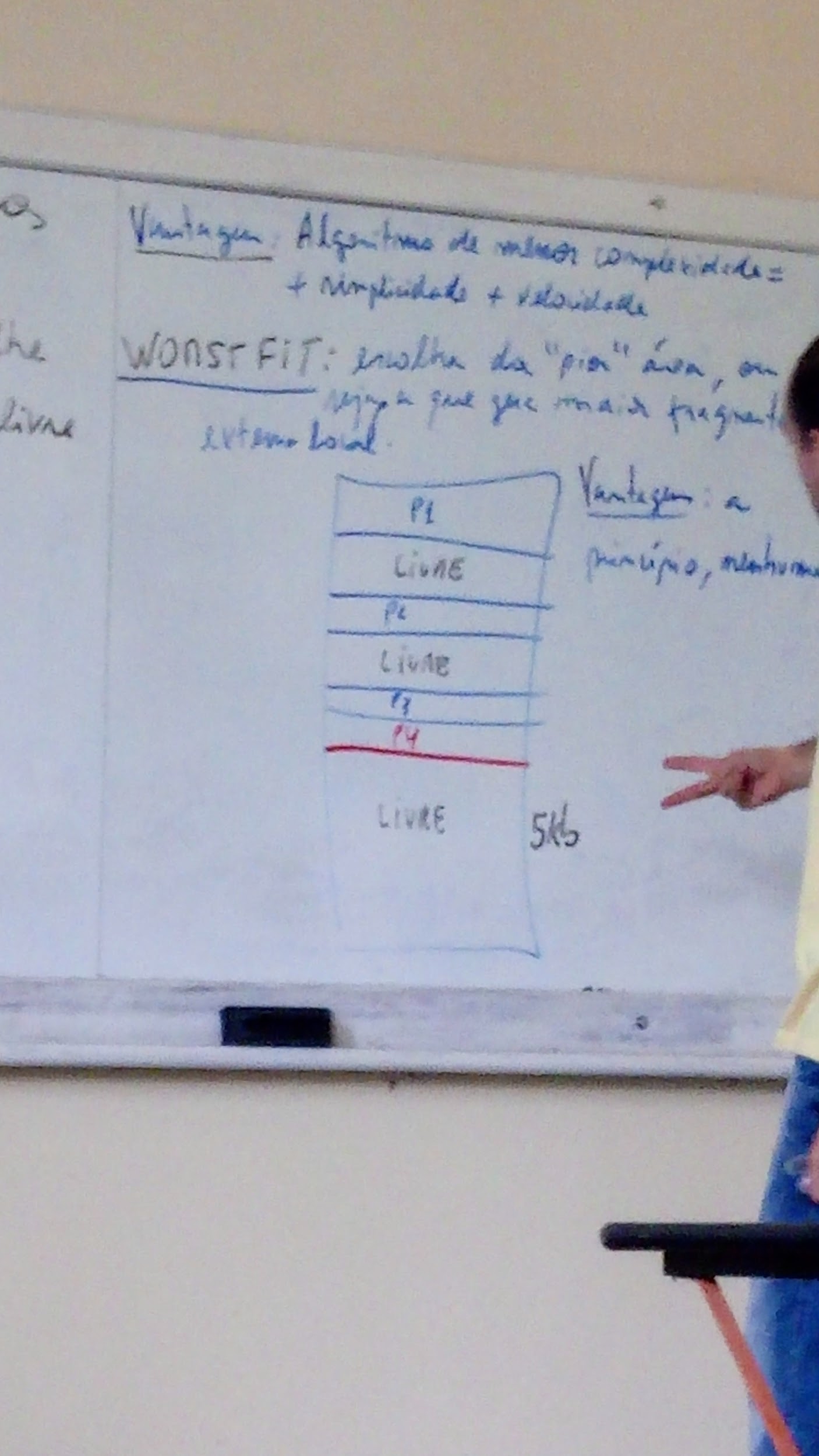


vantagem: algoritimo de menor complexidade = +simplificado + velocidade

worst fit: escolhe da “pior” área , ou seja a que gera maior fragmentação externa local.

na verdade ,o worst fit é o melhor algoritimo ,pois , ao priorizar áreas livres maiores,permite um maior grau de utilização dessas áreas

ao contrário o Best fit,gera menores espaços livres que serão de dificil utilização por processos maiores



vantagens:a principio , nenhum

4 gestão de CPU

o S.O. gerencia a forma como processos ocupam a CPU .

obs : programa= conjunto de instruções em linguagem de máquina ,armazenados em memória secundária

processo = programa em execução = instruções e dados na memória virtual(ram ou disco),nos registradores e no circuito da CPU

objetivo : maximiza uso de CPU

-minimizar ociosidade da CPU

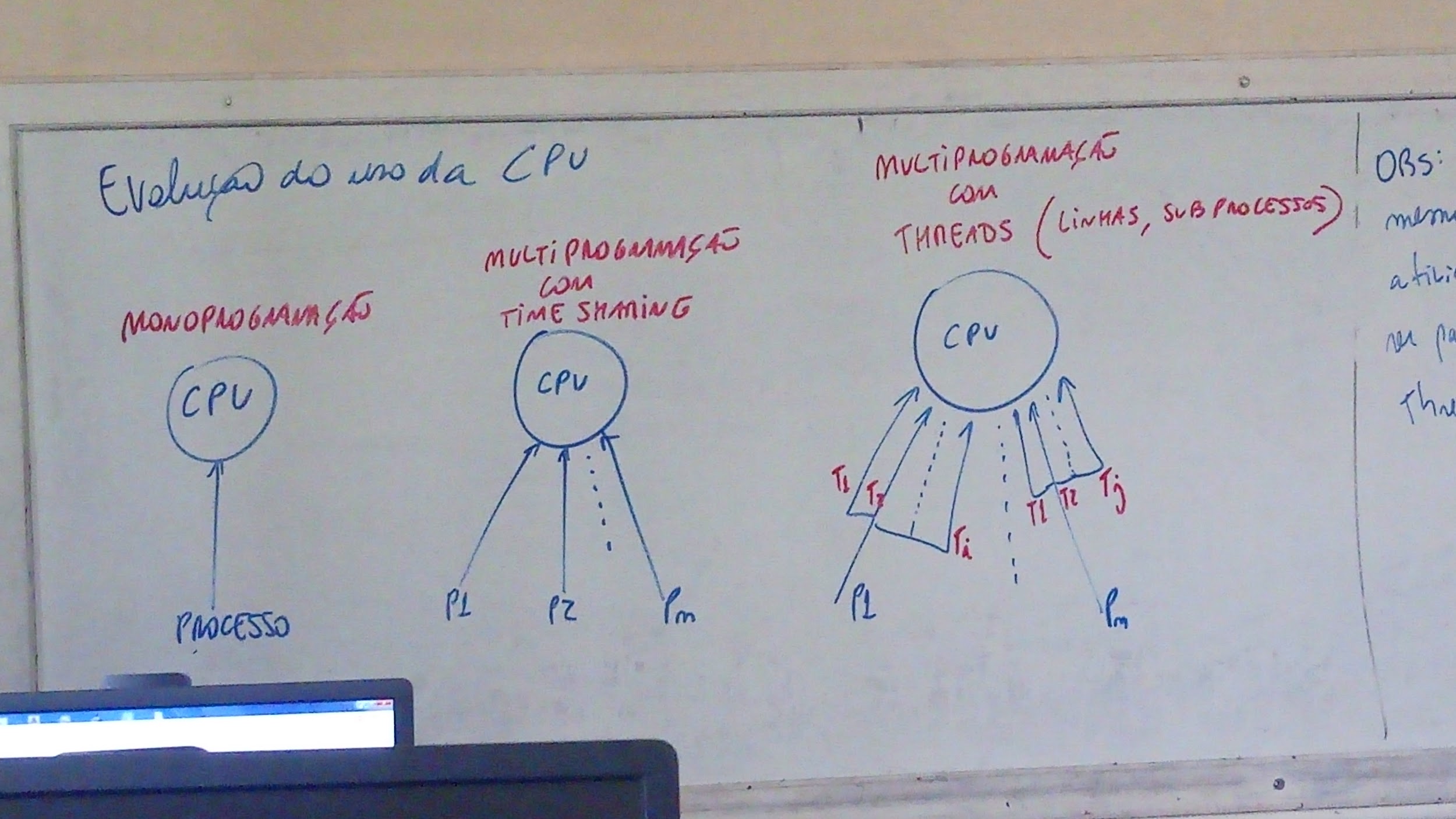
ou maximizar a satisfação do usuário

ex: processamento paralelo(ambientes desktop)

maximizar uso da CPU

ex: aplicativo em tela (smartphone)

maximiza satisfação do usuário ⇒ cpu idle = ociosa



obs:em um mesmo processo , há atividades que podem ser paralelizadas em threads.

monoprogramação:simplicidade ,ociosidade de CPU e espera dos outros processos(starvation)

time sharing: sensação de paralelismo ,onde a CPU executa alguns intervalos de cada diferente processo por determinado periodo de tempo

threads: aumento das linhas de competição pela CPU por processo

escalonamento de CPU: escolha do processo ou thread que irá executar

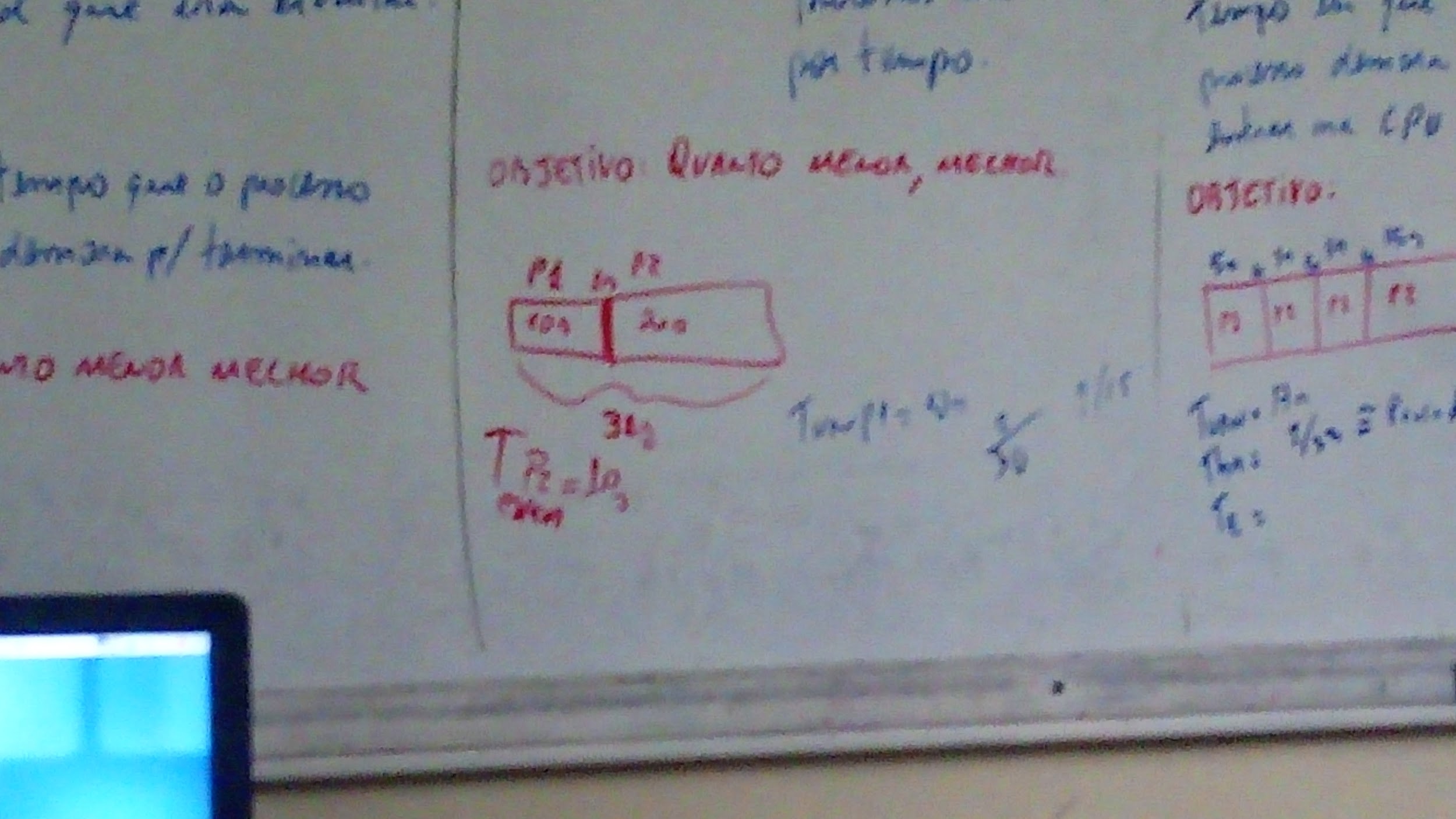
critérios:

turnaround: tempo que o processo demora para terminar

objetivo: quanto menor melhor

trhougput: quantidade de processos resolvidos por tempo.

objetivo quanto menor melhor



tempo de espera: tempo em que o processo demora para entrar no CPU

objetivo :qnt menor melhor

tempo de resposta: tempo que o processo demora para responder ao usuário

objetivo : quanto menor melhor

4.1 escalonamento de cpu: otimiza uso de cpu

algoritimos que decidem qual processo irá executar

4.1.1-não preenditivos

uma vez que o processo começa a executar ele usa a cpu até o fim

turnaround:otimo

throughput:de indiferente (sem i/o) a pessimo (muito i/o)

tempo de espera:muito alto

tempo de resposta:otimo

obs: no caso de i/o a cpu fica ociosa aguardando , que envolver processamento e tráfego mais lentos (ponte norte ,ponte sul e barramentos)

desvantagens: trhoughput (em caso de I/O) e tempos de espera altos demais para aplicações interativas

4.1.1.1-fcfs(first in first served)

primeiro processo requisitado entra na cpu

4.1.1.2-sjf(shortest job first)

o menor processo da fila de espera é escolhido . gera starvation em processos grandes

4.1.1.3-prioridade

processo com maior prioridade são executados primeiramente . gera starvation

4.1.2- preenditivos

processos podem ser retirados da cpu antes de ser terminado

o processo executa por um periodo de tempo e é retirado caso o processo precise realizar i/o ele é retirado de execução .quando um programa tem a sua execução requisitada(processo),ele entra em fila de espera até que o so o escolha para execução . ele executará por um periodo de tempo e irá retornar para a fila de espera , onde pode até ir para a memória secundária(memória virtual)

conceitos :

quantum: tempo de execução fixa na cpu

troca de contexto: retirada do processo em execução na cpu pelo so

consiste em copiar os registradores da cpu para a ram do processi que são e vice-versa para o processo que entra .

é uma operação que consome certo tempo (baixo)

fila de espera: fila de processos aguardando execução (ram ou disco)

novo(new): programa requisitado para execução

pronto(ready): processo na fila de espera

executando: processo na cpu

processo na cpu

aguardando (waiting/blocked): processo aguardado por i/i

terminado (finished):processo terminado

obs: time slice pequeno demais gera muita troca de contexto,o que pode deteriorar a performance geral time value muito grande prejudica a sensação de paralelismo

a montante de tempo para a execução de um time slice para n processos não deve ser maior que o tempo de percepção visual humana

obs tipos de processos: i/o bound-interativos: costumam não gastar todo o seu time slice .geralmente param antes do i/o

cpu-bound: não interativos , geralmente gastam todo o seu time slice-slice .

batch: processos do s.o. ,executam em background ,não visiveis ao usuário . também chamado de DAEMONS

tempo real: necessitam de alta prioridade ,geralmente não podem ser substituidos por outros processos

round robin com prioridades

cada processo (thread) é atribuido a uma prioridade . cada prioridade define um tempo de execução na cpu .

processos com prioridades menor podem sofrer muito , no caso de vários processos compartilhados pela cpu.

para se atenuar esse problema , podem-se utilizar a tecnica de aging (envelhecimento) , ou seja à cada execução na cpu , processos com alta prioridade vão perdendo essa prioridade aos poucos (não usada com sistemas em tempo real)

(grafico 1)

4.3 threads

subprocessos concorrentes derivados de um processo principal cada thread compete pelo uso da cpu . toda atividade em um problema que pode ser paralelizada , deve ser implementada via threads

(foto 1)

ex:

(foto 2)

cada requisição sql de cliente deve ser “paralelizada” para não gerar tempo de resposta muito alto para as ultimas conexões

(grafico 2)

ex2:

obs: nem tudo pode ser paralelizado,a paralelização avança mais rapidamente no processamento de imagens(GPU)

implementação de threads

veremos o javaa,que oferece ALTO NIVEL para o tratamento de threads

as threads deverão executar tarefas paralelizaveis para criar uma thread,devemos fazer uma subclasse da classe thread,

sobrescrevendo(override) o método run

o método run , contém a atividade que comcorrerá pela cpu.

demais métodos:

start(): inicia a execução do método run de forma concorrente

setPriority(thread.prioridade){prioridade =[min\_priority/norm\_priority,max\_priority]}

altera a prioridade da thread

isAlive():verifica se a thread já terminou sua execução(método run)

obs:jvm trabalha com prioridades de 1 a 10 independente do SO subjacente

sincronização e comunicação entre processos(threads)

processos que compartilham recursos precisam de sincrinização para evitar inconsistencias ou comportamentos indevidos

a inconsistencia em recursos compartilhados é um problema grave que pode acontecer no processamento paralelo ou concorrente

nesses casos as seções críticas devem ser sincronizadas

seções críticas: funções que manipulam recursos compartilhas

nessas seções devem haver exclusão mútua ,ou seja,apenas uma thread terá acesso por vez

implementação em java

syncronized: marca um método como atômico , ou seja , uma acçao excludente,somente uma ação deste tipo pode operar sobre recurso compartilhado , enquanto durar seu código = atomicidade de operações

wait: indica que essa thread deve aguardar ,ou seja,sair da CPU e ir para a fila de espera.

notfy: notifica as threads que estão em wait para acordarem , ou seja,retornar à execução

ex: empilha desempilha

se não cheia

topo++

troca contexto →

se não vazia

x= corpo [topo]

topo --

return x ← x=lixo

corpo[topo]=y

topo errado

exercicio 2:

implemente uma conta corrente sincronizada :

| cc |
| --- |
| saldo |
| saque(x)  deposito(x) |

saque : se saldo positivo:

saldo-=x e exibir saldo

senão avisar saldo insuficiente e dermir thread

depósito: saldo += x

exibe saldo

imitar 2 threads :

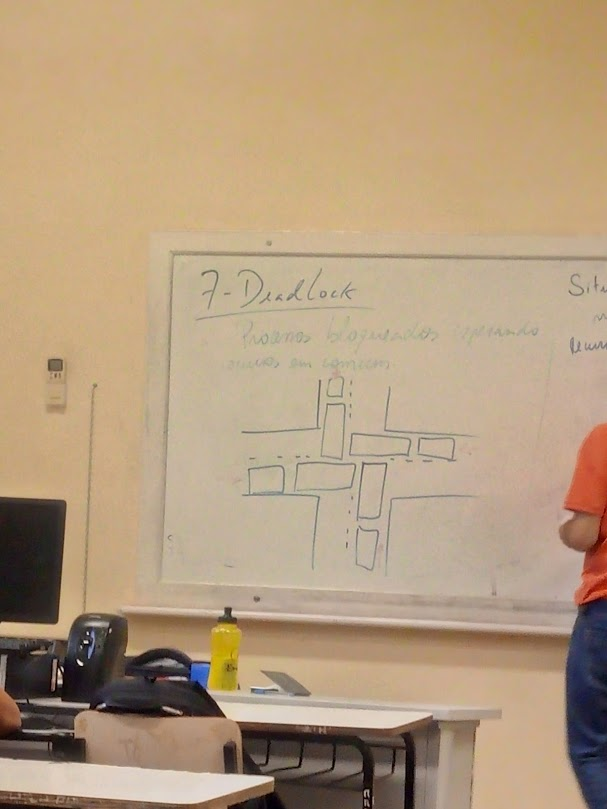
depósito thread: deposita

100 valores aleatórios entre 0 e 999

saque thread: saque 100 valores aleatórios entre 0 e 999

7 dead lock

processos bloqueados esperando recursos em comum



situação onde um processo aguarda recurso que nunca será liberado

recursos: impressora,arquivo,onde há exclusão mútua , ou seja, apenas um processo pode usar por vez



são 4 as condições para um deadlock:

1. recursos que exigem exclusão mútua
2. espera por um recurso para continuar o procedimento
3. não preempção de um recurso, ou seja, desalocar recurso já alocado
4. espera circular

prevenção de deadlock : garantir que pelo menos uma condição não seja satisfeita

na prática,nenhuma das 4 condições é atacada, pois podem gerar inconsistencia e comportamentos indesejados

na prática , os processos congelados são abortados pelo proprio usuário(kill)

9 shellscript

procedimentos :

1. criar arquivo de texto com terminação .sh
2. dar permissão de execução :chmod 700 arquivo.sh
3. executar: ./arquivo.sh

9.1 e/s e variáveis

* saida: echo”texto”
* entrada: read variavel
* não precisa declarar variável
* para acessar conteudo use $

parametros da linha de comandos

$\* - lista de parametros

$# - quantidade de parametros

$1 - 1° parametro

shell if

sintaxe:

if: if [ COMANDO ];then

…..

elif [ COMANDO ] ; then

…

then

…

else

…

fi

comandos suportados:

| -eq | equal | == |
| --- | --- | --- |
| -ne | not equal | != |
| -lt | less then | < |
| -le | less equal | <= |
| -gt | greater than | > |
| -ge | greater equal | >= |
| ! | not | ! |
| -d nome | se é diretório |  |
| -f nome | se é pasta |  |
| -x nome | se é executável |  |
| -w nome | se pode ser escrio |  |
| -r nome | se pode se lido |  |

for: sintaxes:

for var in cs1 cs2 cs3….

for var in {cs1,cs2,cs3...}

for var in $(COMANDO)

for var in "${arrayName[@]}"

for var in ((ini;cond;inc/dec))//igual do c

bloco definido como :

for var in $(cmd)

do

…

done

obs:

for x in $(ls)

x recebe todos os arquivos e pastas listados