

Fundamentos de Sistemas de Operação

Unix Windows NT Netware Mac OS DOS/V/VS Vax/VMS
Linux Solaris HP/UX AIX Mach Chorus

*Subsistema de discos:
SSDs – tecnologia e desempenho*

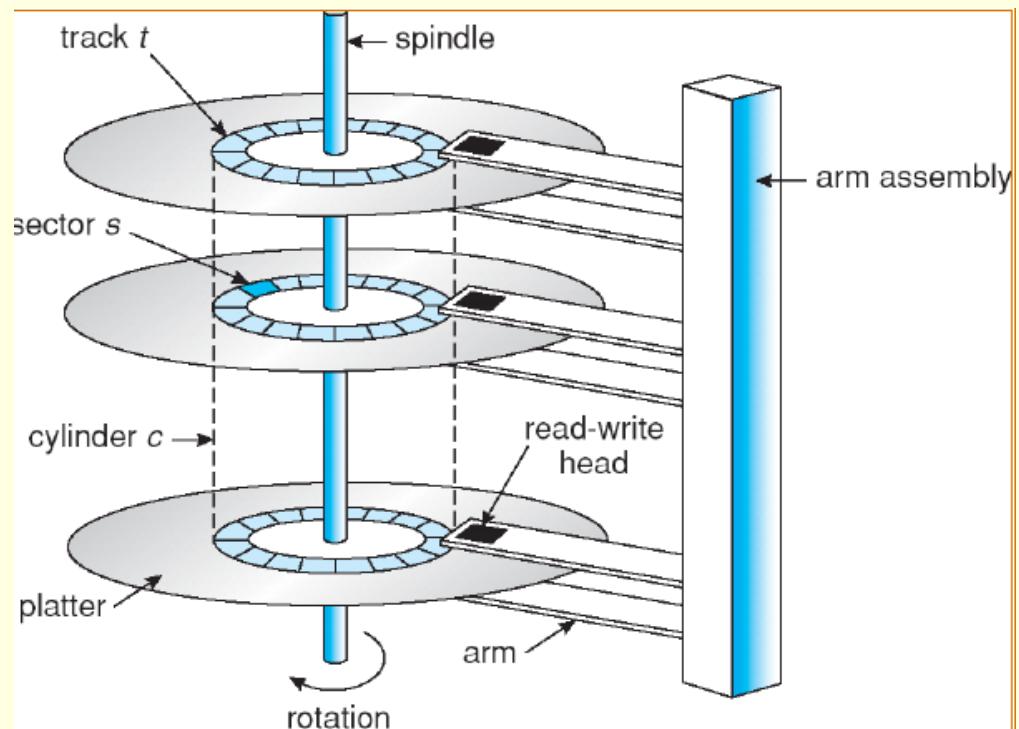
Desempenho de um HDD “top” (1)

□ Access time =

- 1) Seek time (move arm to target cylinder) +
- 2) Rotational latency (wait for target sector(s)) +
- 3) Transfer time disk-to-controller (sectors to disk controller) +
- 4) Transfer time controller-to-host (controller cache to host RAM)

Example: ST3300657FC

- 1) 0.2(R)/0.4(W) ms T2T¹
- 2) 2ms avg. @ 15K rpm
- 3) 1450-2370 Mb/s
- 4) 400 MB/s (FC)



¹ Seek plus Read or Write

Desempenho de um HDD “top” (2)

□ Largura de banda “muito baixa”

- Um HDD consegue \approx 200 **MB/s**
- A memória consegue \approx 20 **GB/s**
- 1 core consegue \approx 20 **GB/s**

100x mais

□ Latência “muito elevada”

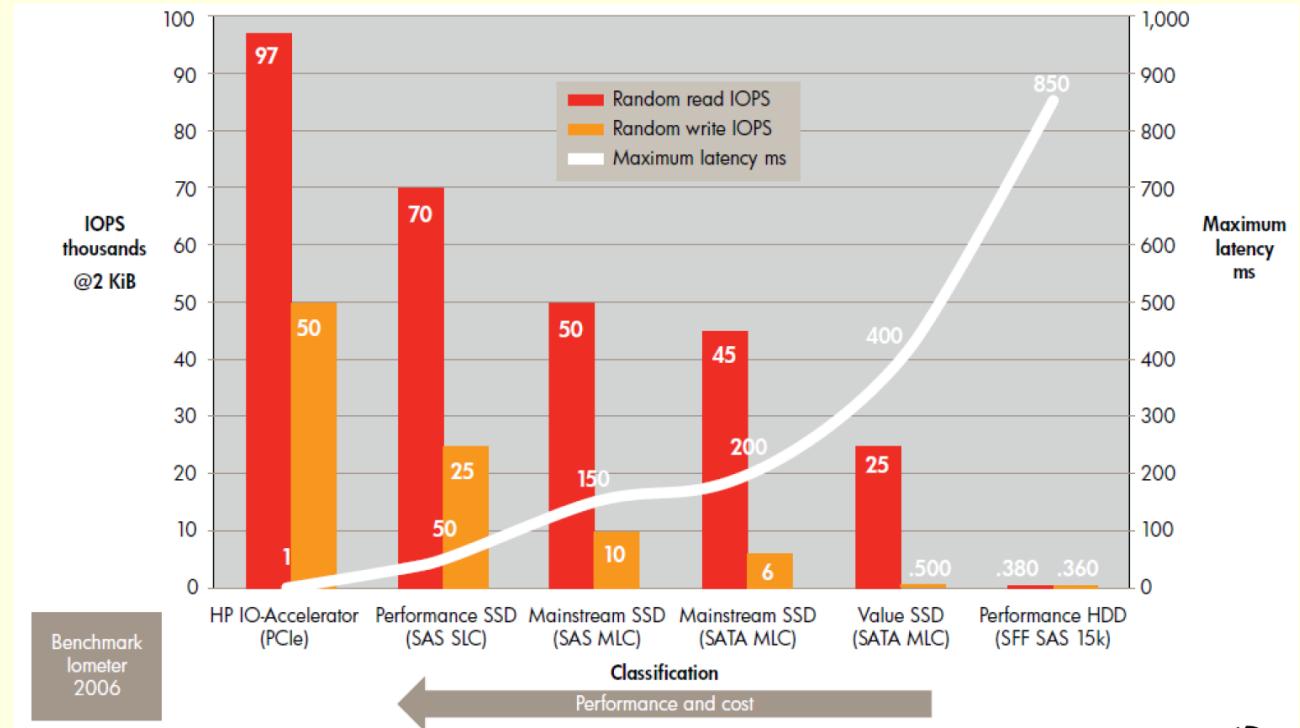
- A latência de um HDD é \approx 2 ms
- A latência da RAM é \approx 100 ns
- A latência dos registos é \approx 0.5 ns

20 000x menor que HDD
200x menor que a RAM

□ Preço de 1.8TB ~ 450€

Desempenho SSD vs. HDD (2010)

Unix Windows NT Netware Mac OS DOS/VMS Vax/VMS
Linux Solaris HP/UX AIX Mach Chorus



8 © Copyright 2011 Hewlett-Packard Development Company, L.P. The information contained herein is subject to change without notice.



Pedro Moreira, HP. Seminário GCD, 2012

Desempenho SSD (2019)

COMPARING PCIe AND SATA SSDs – SM951 PCIe VS. PM871 SATA

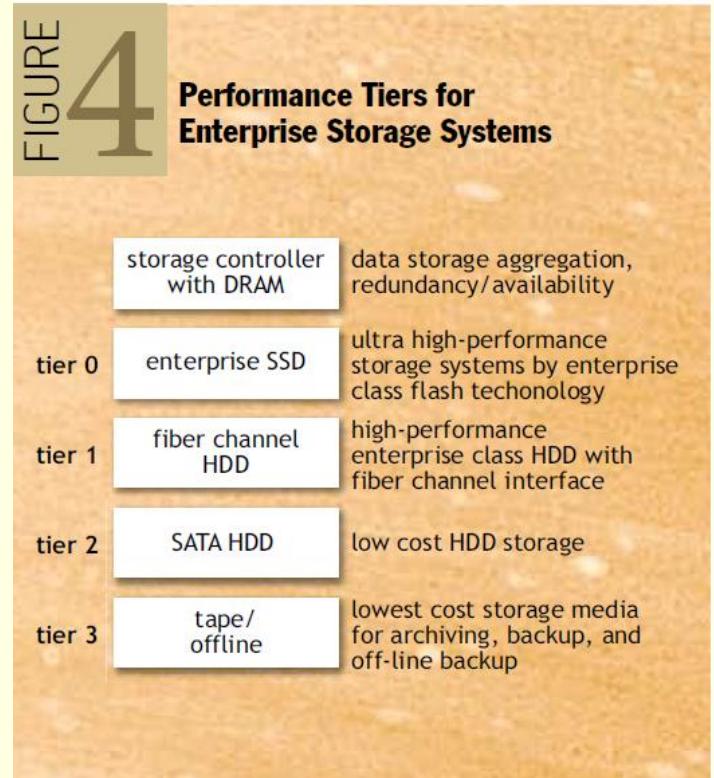
	SM951 PCIe	PM871 SATA
Capacity	128GB, 256GB, 512GB	
MTBF	1.5 Million Hours	
Host Interface	PCI-Express Gen 3.0 x 4 – 32 Gbps NVMe	SATA Gen 3.0 – 6 Gbps AHCI
Peak Read Sequential Performance	Up to 2150 MB/s	Up to 540 MB/s
Peak Write Sequential Performance	Up to 1550 MB/s	Up to 500 MB/s
Peak Read Random Performance	Up to 300K IOPs	Up to 97K IOPs
Peak Write Random Performance	Up to 100K IOPs	Up to 90K IOPs
Form Factor		M.2 2280 *
Dimensions		22 x 80 x 4 mm
Weight		10 grams

Fonte: Samsung

Preço SSD “empresarial” HP 1.8TB ~ 3000 a 4000 €

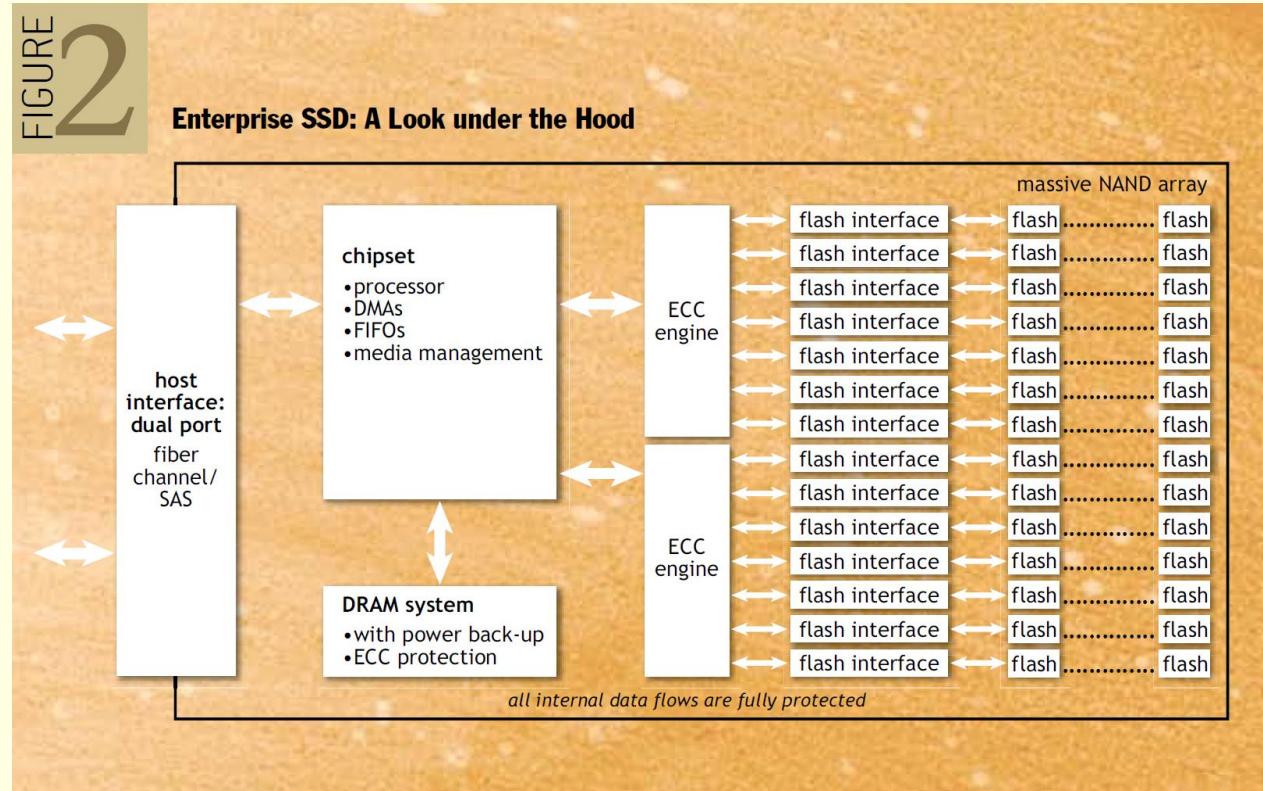
Uso de SSDs na organizações

Unix Windows NT Netware Mac OS DOS/V/S Vax/VMS
Linux Solaris HP/UX AIX Mach Chorus



Fonte: Enterprise SSDs: ACM Queue, July/August 2008

Arquitectura de um “SSD empresarial”



Fonte: Enterprise SSDs: ACM Queue, July/August 2008

Tecnologia Flash

□ Inventor: Fujio Masuoka, anos 80

- Baseada em células contendo transístores que armazenam cargas eléctricas (para representar 0s e 1s)
- Se uma célula armazena um só nível de carga (portanto, armazena 1 bit) a tecnologia designa-se SLC (Single Level Cell)
- Tecnologias mais “compactas” permitem armazenar 2 níveis por célula (MLC, Multi-Level Cell) ou 3 (TLC, Triple-Level Cell). Com estas constroem-se “discos” SSD com maior capacidade mas menor desempenho que os SLC. O “tempo de vida” também é menor...

Chip Flash: organização e acesso

- *Um chip de memória flash contém*
 - múltiplos bancos (também designados planos), compostos por milhares de células
 - cada banco pode ser acedido de duas formas: em blocos, tipicamente de 128KB ou 256KB, ou em páginas, de 4KB (não confundir com páginas de RAM...)
- *Operações ao nível do dispositivo flash*
 - **READ**: Ler (uma página); **ERASE**: Apagar um bloco; **PROGRAM**: Escrever uma página num bloco “vazio” (apagado)
 - **Consequência**: para escrever uma página (nem que seja só para mudar 1 bit), é preciso apagar o bloco que contém essa pagina.

Flash: desempenho e fiabilidade

□ Desempenho

Device	Read (μ s)	Program (μ s)	Erase (μ s)
SLC	25	200-300	1500-2000
MLC	50	600-900	~3000
TLC	~75	~900-1350	~4500

Figure 44.2: Raw Flash Performance Characteristics

□ Fiabilidade

- *Previsões dos fabricantes, para “escritas” no mesmo banco:*
MLC: 10000, SLC: 100000

Flash: ainda a fiabilidade (1)

- *Porquê a fiabilidade em termos de “apagar/programar”?*
 - *Na tecnologia flash o acto de apagar as células corresponde, de facto, a escrever 1s. Numa “escrita” (e é por isso que se chama programar e não escrever), apenas se escrevem os zeros nas posições (bits) necessárias.*
 - *Escrever 1s corresponde a “carregar” com cargas eléctricas as células e, com o tempo, os “carregamentos” (erase) sucessivos vão fazendo com que as células, a certa altura, já não consigam perder a carga ao serem programadas, e passar ao estado zero.*
- *E há mais problemas que podem influenciar a fiabilidade?*
 - *Sim; tal como nos HDDs, em que campos magnéticos de “bits” vizinhos de um dado bit podem alterar o valor desse bit, o mesmo pode acontecer nas flash: células vizinhas influenciam uma célula.*

Flash: ainda a fiabilidade (2)

□ Soluções para aumentar a fiabilidade?

- *Para aumentar a vida útil: distribuir os ciclos apagar/programar de forma equitativa pelos diferentes blocos.*
- *“Influências vizinhas” (cujo termo técnico é cross-talk): usar redundância e códigos de detecção e correcção de erros (CRC ou ECC) tal como nos HDDs e nas RAMs.*

Construir um SSD (1)

□ Modelo “direct-mapped”

Nº total de Páginas: NTP

Nº de Página Lógica: NPL

Nº de Página Física: NPF

Nº do Bloco: NB

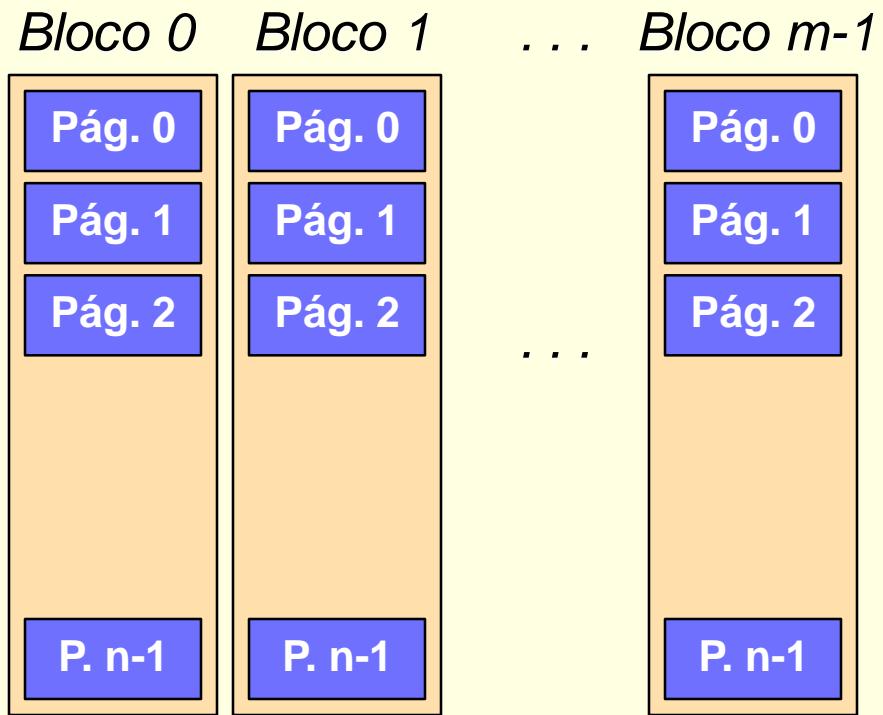
Nº de Páginas por Bloco: NPB

$$NTP = m \times n$$

$$NB = NPL / NPB,$$

$$NPF = NPL \% NPB$$

Consideremos que o SF quer ler a página lógica 25 num disco com 10 blocos e 10 páginas / bloco. Terá de ler a página física 5 do bloco 2 ☺



Construir um SSD (2)

□ *Modelo “direct-mapped”: o controlador do SSD*

- *Neste caso, o controlador: (i) recebe um pedido para ler o “bloco” (página) de disco 25, e decompõe esse endereço em ler (bloco 2, página 5). Depois, (ii) transfere o conteúdo dessa página para a interface.*
- *No caso do pedido ser para escrever no SSD um conteúdo de 4KB no “bloco” (página) de disco 25, o processo vai ser mais complexo:*
 - *Encontrar um bloco totalmente apagado - digamos que é o bloco k*
 - *Copiar para bloco k o conteúdo de todas as páginas do bloco 2 com exceção da página 5, cujo conteúdo não vem dessa página, mas dos dados que chegaram pela interface*
 - *Apagar todo o bloco 2,*
 - *Copiar de novo, agora de k para 2*
 - *Apagar o bloco k, deixando-o “livre” de novo*

Construir um SSD (3)

□ *Modelo “direct-mapped”: uma má escolha*

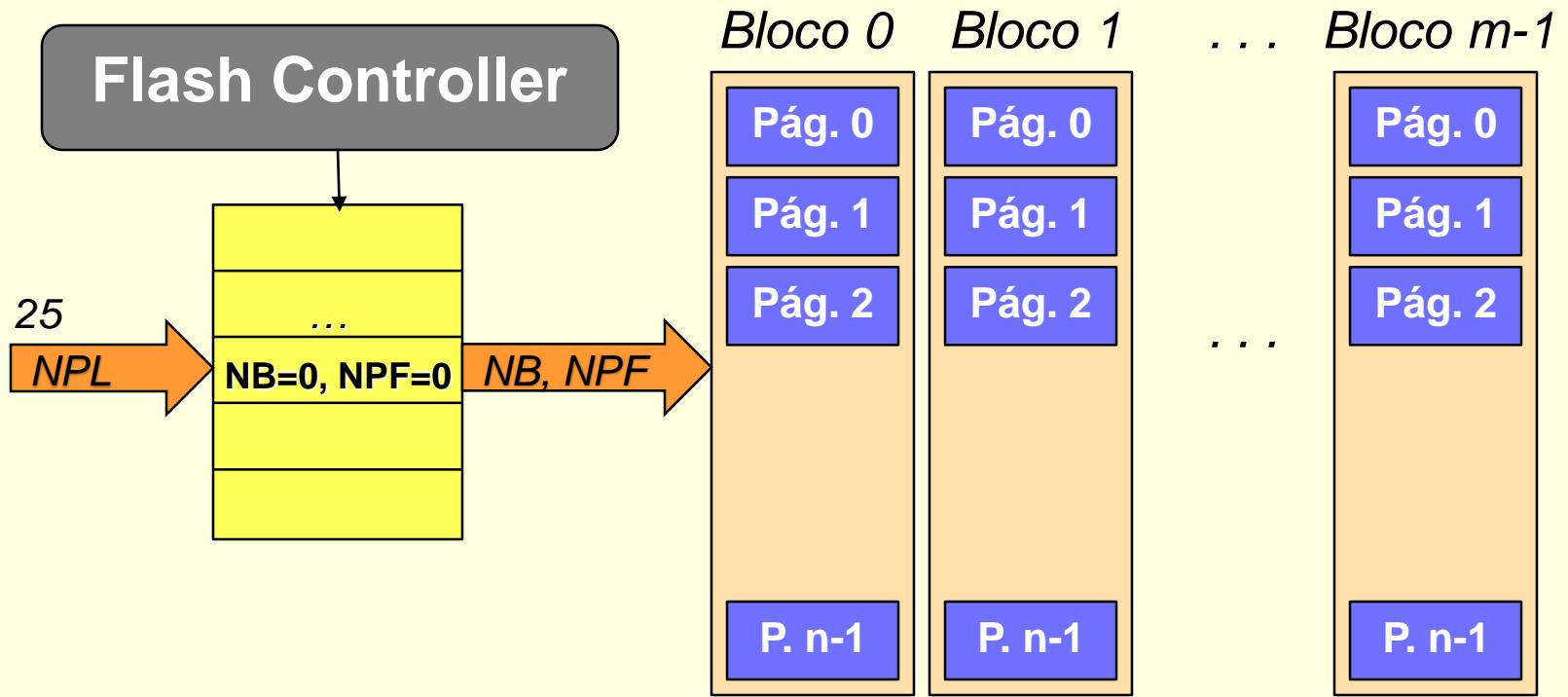
- *Desempenho “desastroso”: a nossa escrita de 4K no SSD deu origem a 2 operações READ, 2 ERASE e 2 PROGRAM – um total de $2x(25+200+1500) = 3450 \mu\text{s}$ gastos, idêntico a um HDD!!!*
- *O tempo de vida do SSD desceu com estes ERASES e PROGRAMS*

□ *Solução?*

- *Em vez de um algoritmo de endereçamento lógico/físico direct-mapped, construir um baseado numa “tabela de páginas”. Ou seja, em vez de, a partir do endereço lógico do “bloco” de disco, dado pelo SF, o controlador fazer os cálculos simples de transformação que indicamos anteriormente (slide 13), vamos usar um algoritmo e uma tabela de dados do tipo da usada na paginação.*

Construir um SSD (4)

- *Modelo “tabela de páginas”*



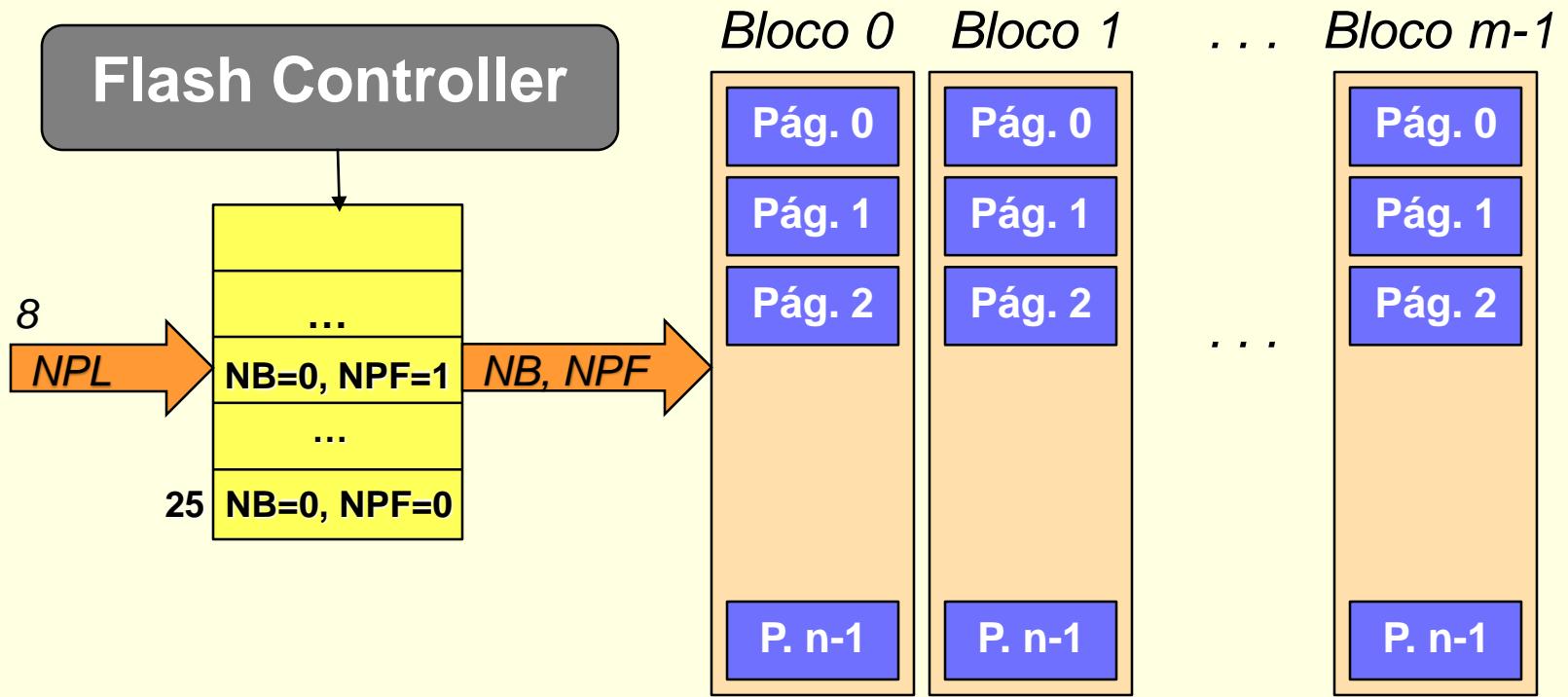
Construir um SSD (5)

□ *Modelo “tabela de páginas”*

- No exemplo anterior, o SF continua a querer escrever 4KB no “bloco” SSD 25 (endereço 25 ou página lógica 25). Mas agora, o bloco 0 estava livre (**ERASEd**)
 - O controlador do sistema flash escreve (**PROGRAMA**) os 4K no bloco 0, página 0 e actualiza a entrada 25 da tabela de páginas com essa informação. Tempo gasto: 200 μ s
- Agora o SF quer escrever 4KB no “bloco” SSD 8 (endereço 8 ou página lógica 8)
 - O controlador do sistema flash escreve (**PROGRAMA**) os 4K no bloco 0, página 1 e actualiza a entrada 8 da tabela de páginas com essa informação. Tempo gasto: 200 μ s

Construir um SSD (6)

- Modelo “tabela de páginas”



Construir um SSD (7)

□ Modelo “tabela de páginas”: conclusões

- Com este modelo, os dados vão sendo “acrescentados” aos blocos, evitando-se operações de **ERASE**. A tabela de páginas vai mantendo a ligação endereço lógico/físico.
- A re-escrita de uma página, com dados actualizados, não obriga a uma sequência **ERASE** + **PROGRAM**: pode pura e simplesmente escrever-se na próxima página livre do bloco, actualizar a tabela, e marcar a entrada correspondente à página antiga como “obsoleta” ou inválida

□ E mais?

- Pode ter-se um mecanismo que distribui os **ERASES** pelos diferentes blocos, mantendo-os mais ou menos “alinhados”

Para concluir... extracto do livro (1)

ASIDE: KEY SSD TERMS

- A **flash chip** consists of many banks, each of which is organized into **erase blocks** (sometimes just called **blocks**). Each block is further subdivided into some number of **pages**.
- Blocks are large (128KB–2MB) and contain many pages, which are relatively small (1KB–8KB).
- To read from flash, issue a read command with an address and length; this allows a client to read one or more pages.
- Writing flash is more complex. First, the client must **erase** the entire block (which deletes all information within the block). Then, the client can **program** each page exactly once, thus completing the write.
- A new **trim** operation is useful to tell the device when a particular block (or range of blocks) is no longer needed.
- Flash reliability is mostly determined by **wear out**; if a block is erased and programmed too often, it will become unusable.

Para concluir... extracto do livro (2)

- A flash-based **solid-state storage device (SSD)** behaves as if it were a normal block-based read/write disk; by using a **flash translation layer (FTL)**, it transforms reads and writes from a client into reads, erases, and programs to underlying flash chips.
- Most FTLs are **log-structured**, which reduces the cost of writing by minimizing erase/program cycles. An in-memory translation layer tracks where logical writes were located within the physical medium.
- One key problem with log-structured FTLs is the cost of **garbage collection**, which leads to ~~amazing~~ ~~amazing~~.
- Another problem is the size of the mapping table, which can become quite large. Using ~~prioritizing~~ ~~mapping~~ or just **caching** hot pieces of the FTL are possible remedies.
- One last problem is **wear leveling**; the FTL must occasionally migrate data from blocks that are mostly read in order to ensure said blocks also receive their share of the erase/program load.