Administració de memòria

Jerarquia de Memòria a un Computador

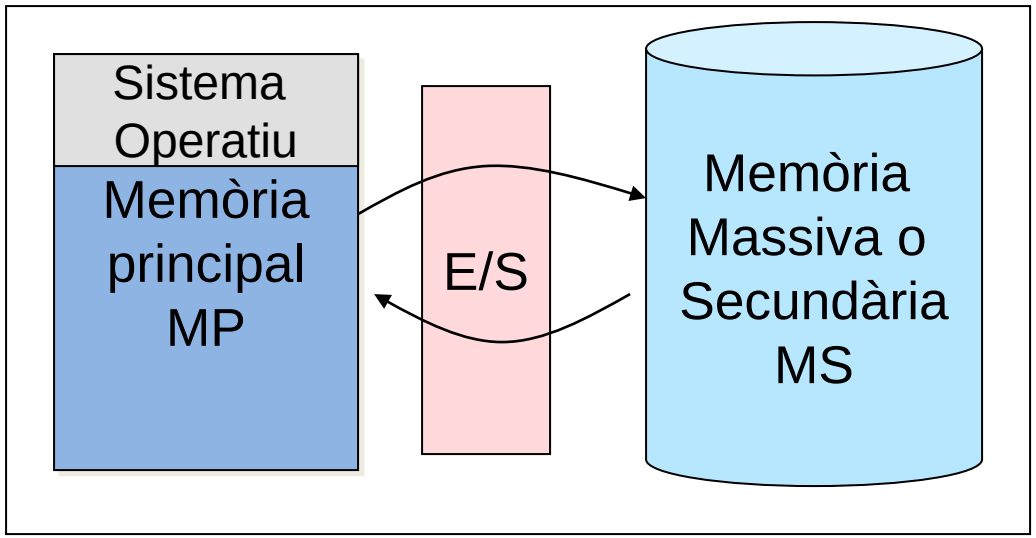
Com s'ha indicat al tema anterior, a un computador necessitem implementar diferents tipus de memòries perquè NO existeix actualment el dispositiu de memòria ideal.

Les característiques que hauria de tenir una memòria ideal són:

- Ràpida com el processador o més.
- Capacitat molt gran en poc espai (alta densitat d'integració).
- Poder retenir les dades quan no hi ha energia.
- Poder Escriure i Llegir a la mateixa velocitat.
- Econòmica.

Sistema Monotasca

L'estructura d'un sistema de propòsit general (computador)



- El SO residirà en la MP ja que ha de gestionar els recursos del sistema i l'execució dels programes.
- La Memòria Secundària (MS), en principi, conté programes i dades que no s'estan executant.

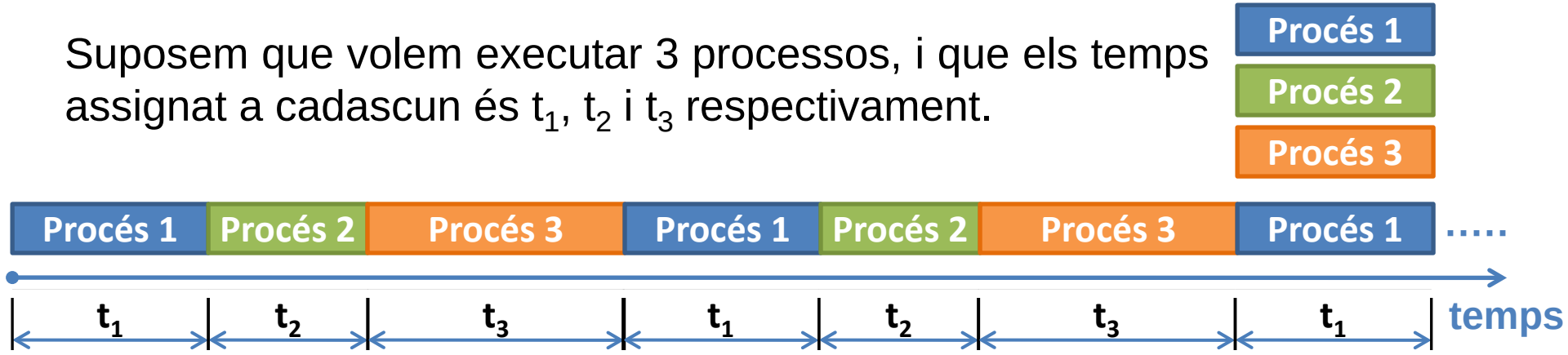
Sistemes Multitasca

Suposem que volem executar diferents programes simultàniament. Això és el denominarem sistema multitasca (o multiprocés).

En general, els sistemes multitasca basen el seu funcionament en multiplexació temporal:

- A cada tasca (procés) se li assigna un temps de CPU.
- Un cop ha passat aquest temps s'executarà un altre tasca.
- Quan s'ha fet això amb totes les tasques, es torna a la tasca inicial.
- Això es farà fins que vagin acabant d'executar-se les tasques.

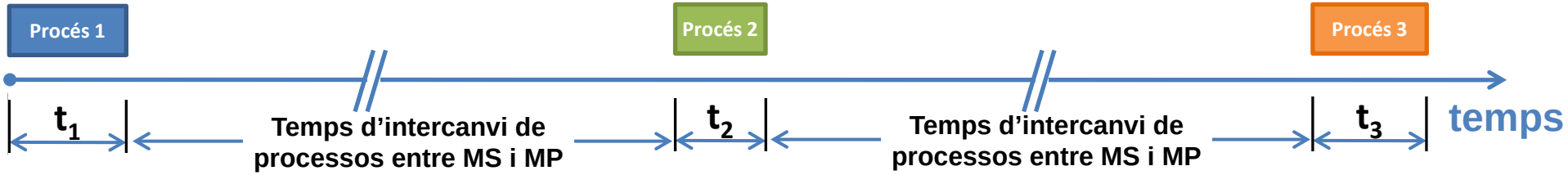
Suposem que volem executar 3 processos, i que els temps assignat a cadascun és t_1, t_2 i t_3 respectivament.



Primera solució: Intercanvi de tasques amb la Memòria Secundària

Pensant que només podem tenir un procés a MP en un instant determinat:

- Es carrega el procés 1 des de MS a MP i es comença a executar.
- Al cap del temps t_1 es retorna la tasca a MS i se'n carrega el següent procés.
- Es repeteix el procediment fins a acabar totes les tasques.

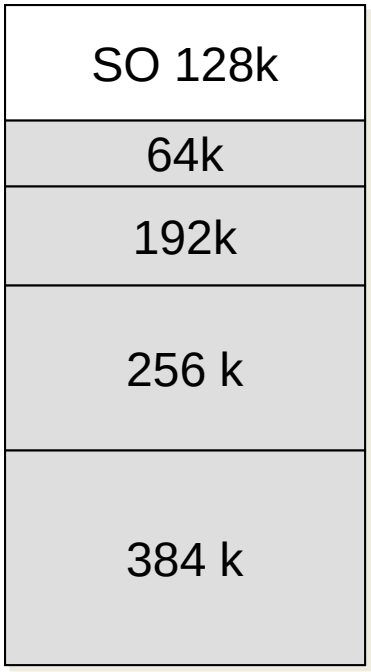


Problema fonamental: les transferències entre MP i MS són operacions d'E/S. Són processos lents que deixen la CPU "ociosa" durant massa temps.

Particions de mida fixa

De fet la limitació del sistema anterior resideix en no poder tenir més d'un procés a la vegada a MP.

Per evitar això dividírem la MP en "particions" independents:



Exemple d'una MP dividida en 5 particions. Pot albergar 4 processos + SO.

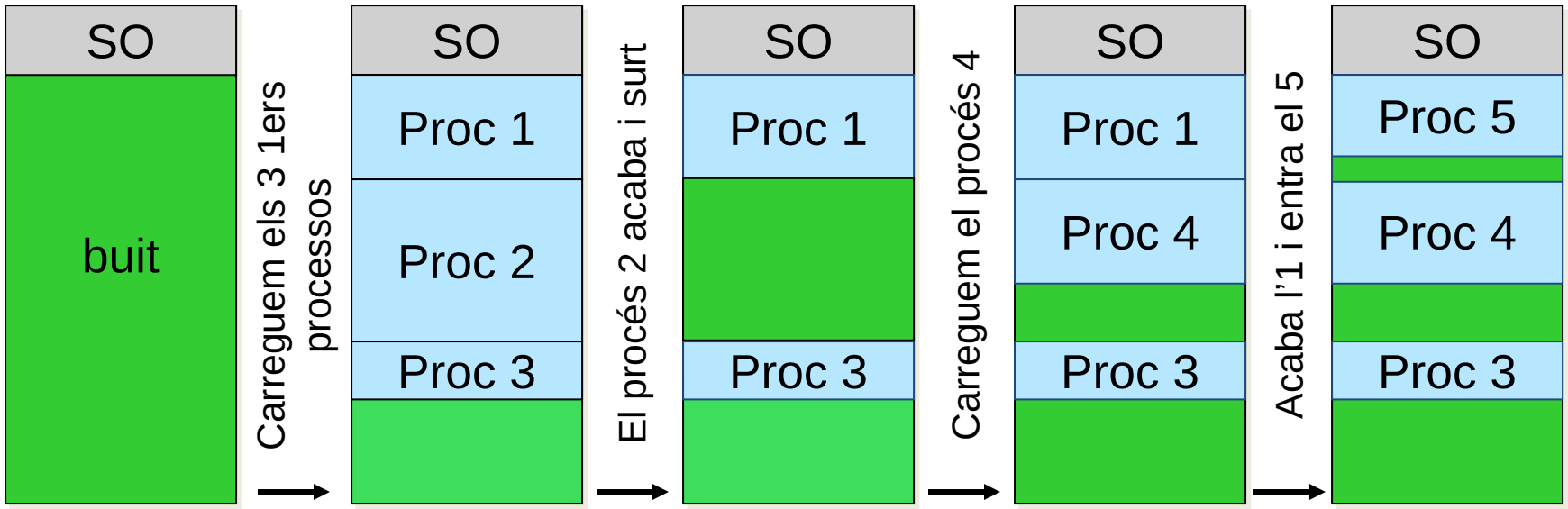
- Cada procés residirà a una partició.
- Al principi es carreguen els processos a les particions (només un cop).
- Després, per commutar d'un procés a l'altre (quan passin els temps, t_1, t_2, t_3....) només s'ha de canviar de partició.
- Particions de diferents mides, per no malbaratar tanta memòria.
- Quan un procés ha de ser executat es col·loca a la partició de memòria més petita disponible.

Problema fonamental: Hi haurà memòria malbaratada, ja que en la majoria dels casos un procés no requerirà tanta memòria com la prevista per la partició.

Particions de mida variable

- Per evitar el problema anterior farem les particions de mida variable.
- Quan carreguem un procés a MP, se li assigna la quantitat de memòria exacta que necessita
- Inicialment la MP està buida, excepte la part ocupada per el SO
- Posteriorment es van carregant processos a executar

Suposem que es volen executar 5 processos



Problema: Apareixen zones de memòria massa petites per encabir un procés. Zones de memòria malbaratades. Es diu que la memòria es va **fragmentant**.

Particions de mida variable

Compressió o Defragmentació

Per solucionar el problema de la fragmentació el es fa és que el Sistema Operatiu d'en tant en tant reubica els processos a la MP de forma consecutiva, per tal col·locar tota la memòria lliure en un bloc contigu.

Problema: procediment que consumeix temps de CPU. Mentre s'està fent el procés de compressió no s'executen les tasques que en realitat és el ens interessa a nosaltres.

Existeixen solucions que es veuran més endavant, per ara analitzarem algunes de les conseqüències que sorgeixen pel fet de treballar amb particions i/o re-ubicacions dels programes a la MP.

Adreces Lògiques i Adreces Físiques

Tornem al sistema de particions i considerem certs detalls

- Cada procés s'executa en una zona de memòria, la localització de la qual depèn de la disponibilitat de MP en el moment de ser carregat.
- Si es du a terme una Defragmentació ⇨ canviaran les adreces dels components (instruccions i dades) del procés.
- Això significa que, quan es fa algun salt en el programa, o quan s'accedeix a posicions de Memòria per a localitzar dades, les seves adreces a la MP varien segons el lloc en que s'hagin carregat.
- Per tant, no podem crear els programes amb adreces absolutes, sinó que s'ha de treballar amb adreces relatives.
- Direm que fem servir:

ADRECES LÒGIQUES

Adreces Lògiques i Adreces Físiques

Definim 2 tipus d'adreces:

1 **Adreça Lògica**: Qualsevol adreça utilitzada per un programador o generada per una tasca per especificar un element d'informació. (posició relativa al inici del programa)

Espai d'Adreces Lògiques: El conjunt de totes les adreces lògiques utilitzades per una tasca.

---

2 **Adreça Física**: Adreça utilitzada pels elements del sistema per accedir a la posició en la que es troba la informació emmagatzemada a la MP. (posició actual en MP)

Espai d'Adreces Físiques: Mida màxima de memòria adreçable pel processador. És igual a 2 elevat al número de bits del bus d'adreces.

Com es veurà, si els dos espais són independents, el codi (programa) i les dades són independents de la quantitat de MP del computador.

Adreces Lògiques i Adreces Físiques

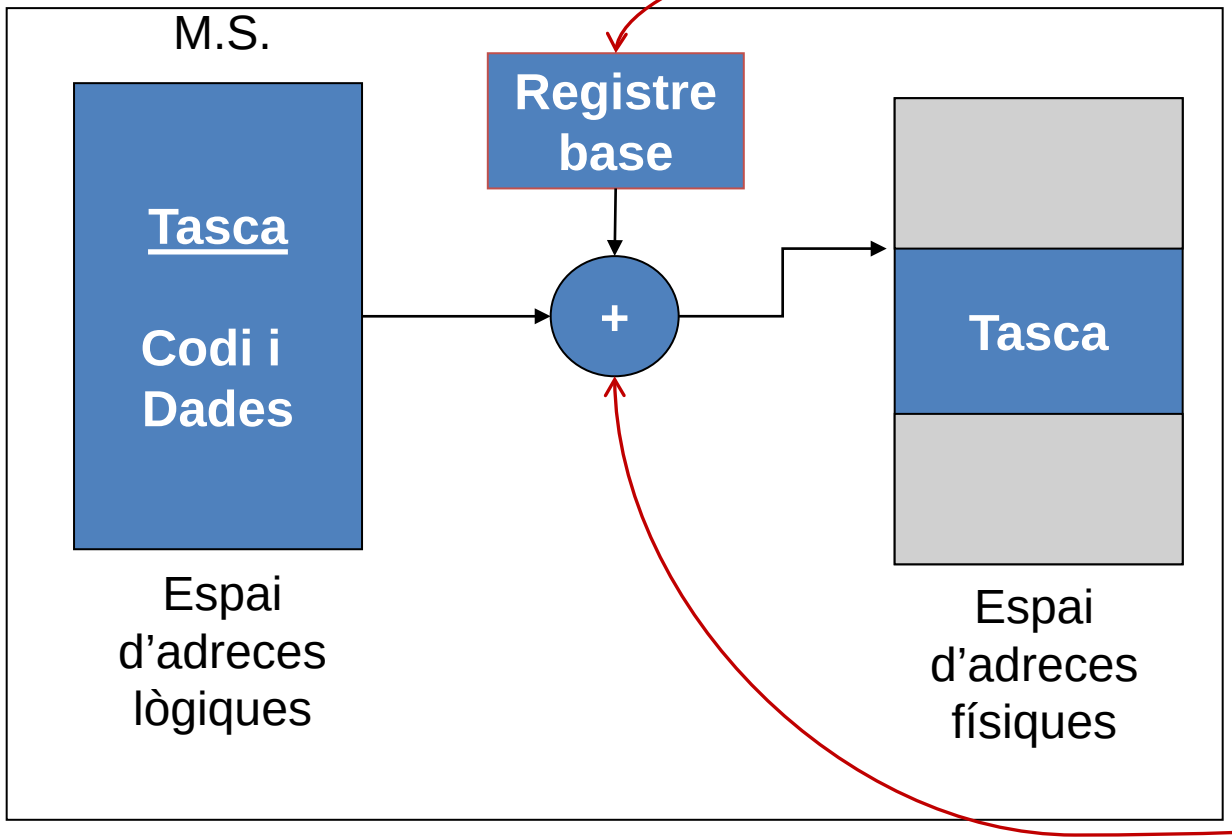
El codi objecte del programa **NO** conté adreces físiques, resideix en el seu propi espai de direccions lògiques i qualsevol referència a una element d'informació (instrucció o dada) es fa amb direccions lògiques

- En el moment en que es carrega un procés a MP, ja residirà en un espai d'adreces de memòria físic.
- Quan el procés s'està executant, s'han de convertir les adreces lògiques a direccions físiques en el mateix moment d'execució (en temps d'execucció).

La forma en que es realitza aquest procés varia segons l'esquema d'**ASSIGNACIÓ DE MEMÒRIA.**

Exemple de translació d'Adreces Lògiques i Adreces Físiques

En cas de particions de mida variable es fas servir un adreçament relatiu, fent servir un registre que denominarem: REGISTRE DE BASE



Quan es carrega una tasca a la MP se li assigna un Registre Base que conté l'adreça d'inici del seu espai físic.

Per traslladar una adreça lògica a una física consisteix simplement en sumar-li el contingut del Registre Base

Cada tasca està lligada a un Registre Base

PAGINACIÓ

- Consisteix en dividir la memòria principal en parts iguals de mida petita (**marcs, o marcs de pàgines**) i cada procés està dividit també en parts de la mateixa mida (**pàgines**).

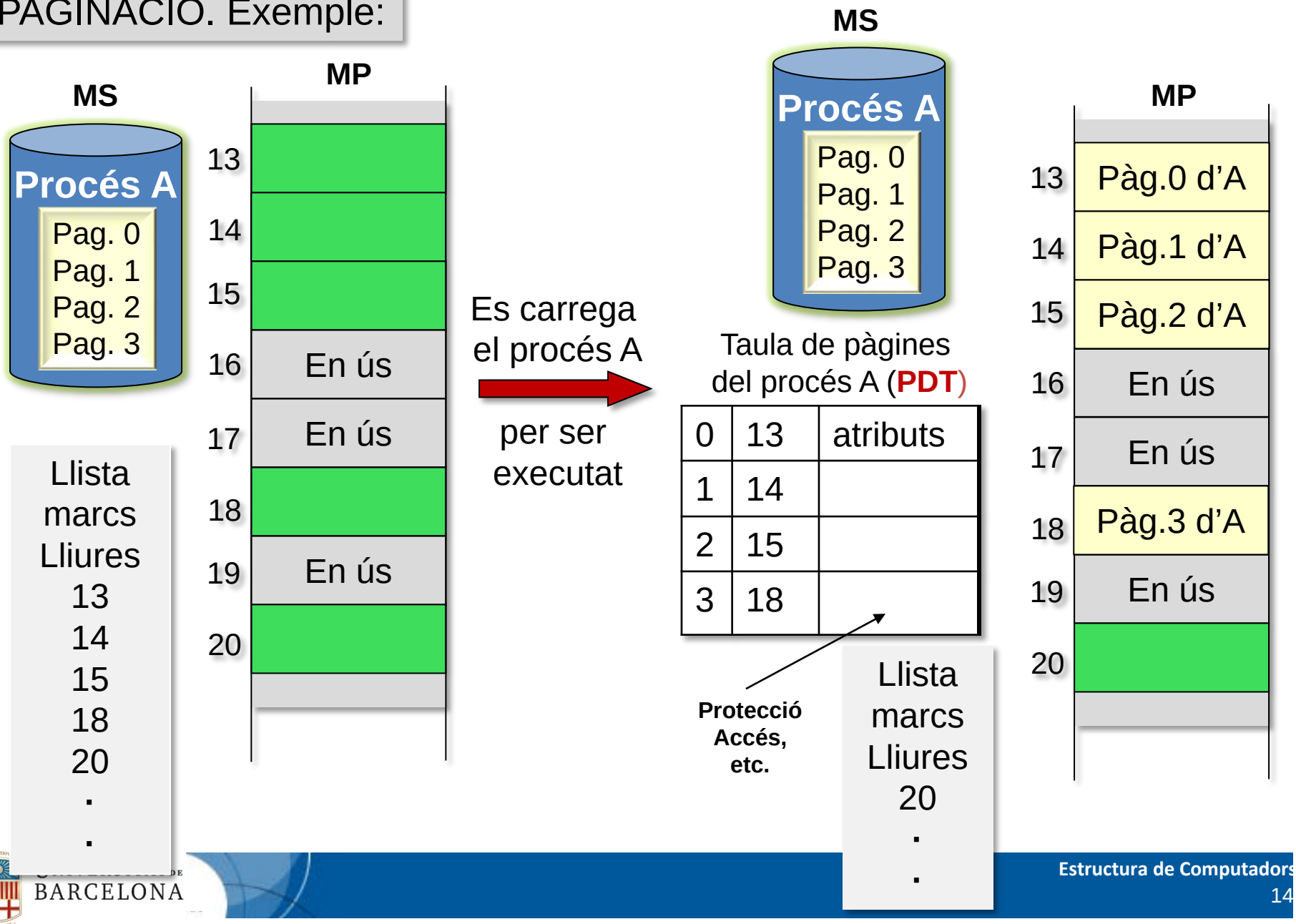
- Durant el procés de portar una tasca de MS a MP, a cada pàgina de la tasca (programa+dades) se li assigna un marc de pàgina de la MP.

- L'únic espai de MP que es desaprofita per tasca serà com a molt una part de la seva darrera pàgina.

- En general, a l'ordinador, tindrem una sèrie de marcs utilitzats per diferents tasques i una sèrie de marcs lliures.
  - La llista de marcs lliures està controlada pel SO

- La mida dels marcs (i per tant de les pàgines) ve donada pel processador (per exemple, en els Pentium = 4kB)

PAGINACIÓ. Exemple:

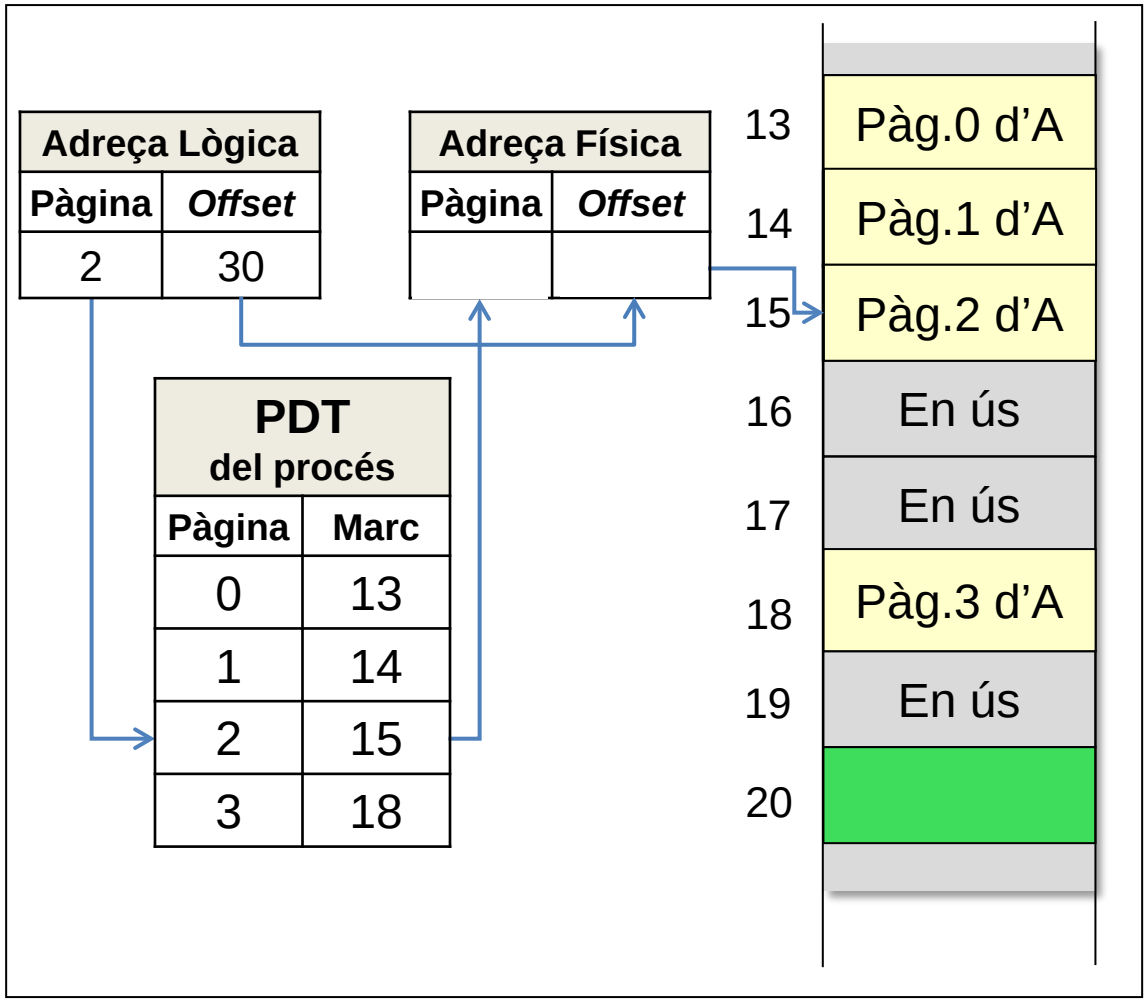


PDT: Page Descriptor Table (Taula de Descriptors de Pàgines)

Cal substituir el Registre Base per una taula que anomenarem **PDT (*Page Description Table*) Taula de Descriptors de Pàgines de procés**. Que és una taula de registres base amb atributs.

- La Taula indica el marc on s'ubica cada pàgina del procés.
- Cada entrada de la PDT, l'equivalent al registre base, es denomina **Descriptor de Pàgina** i té el número de pàgina, el de marc i uns bits de estatus o atributs (ja els veurem).
- Dins del programa cada adreça lògica està constituïda pel número de pàgina i una adreça relativa dins de la pàgina (*offset* o desplaçament).
- Cada procés té la seva pròpia PDT.
- No és necessari que tots el marcs assignats siguin contigus. La utilització d'adreces lògiques farà que el programador vegi la tasca com una unitat.
- El processador (MMU) s'encarregarà de la traducció d'adreces lògiques a adreces físiques a partir de la PDT.

Traducció d'adreces Lògiques a Físiques a la Paginació



Traducció d'adreça lògica a física mitjançant la PDT

En realitat l'adreça física és la traducció d'aquest dos números: marc (15) i *offset* (30) a un només.

Això depèn de la mida dels marcs. Per a un processador tipus Pentium (4kB de mida de marc/pàgina) a l'exemple, l'adreça física seria:

$$15 \cdot 4096 + 30 = 61470 = \mathrm{F01Eh}$$

PAGINACIÓ PER DEMANADA – MEMÒRIA VIRTUAL

L'estratègia de divisió de processos en pàgines va comportar el desenvolupament del concepte de Memòria Virtual.

- Cada pàgina d'un procés s'introdueix a memòria només quan es necessita, és a dir, quan es sol·licita.
- Considerem un procés gran (vàries pàgines) consistent en:
  - Programa de vàries pàgines.
  - Diversos blocs de dades (també en vàries pàgines).
- En un interval de temps curt l'execució pot ser confinada a una petita part del programa (subrutina o bucle) i només fa servir una sèrie de dades [Principi de localitat] no cal carregar totes les pàgines del procés. Només es carreguen les pàgines de la secció a executar en aquest moment.
- Si hi ha algun salt a una adreça lògica no carregada, es produeix una Falta de Pàgina i el SO s'encarrega de carregar la pàgina demandada.

PAGINACIÓ PER DEMANADA – MEMÒRIA VIRTUAL

Aquesta tècnica permet:

- Tenir més processos simultàniament a MP
- Executar programes més grans que la mida de la MP.

El SO ha de ser hàbil en l'intercanvi de pàgines. Existeixen algoritmes per gestionar els intercanvis per assolir resultats més eficients. SO intenta predir (basat en història recent) quines pàgines es faran servir.

A la MP se l'anomena "Memòria Real" ja que tot procés per executar-se ha d'estar en ella.

El programador percep una memòria més gran (distribuïda en el disc) anomenada **MEMÒRIA VIRTUAL**

Gestió de la Paginació

La paginació és una tasca que realitza el SO i el Hardware, és totalment transparent al programador, encara que es treballi amb memòria virtual

Cal tenir en compte que, sobre tot quan es treballa amb la memòria virtual, les entrades a les taules de pàgines poden ser realment complexes:

- La mida de la taula pot ser molt gran.
- Pot haver pàgines en MP i altres a la MS.
  - S'ha d'afegir un bit a cada element de la taula que notifiqui la situació de cada pàgina en cada instant..
- Per altre banda, la taula també està a MP, cada accés a un element d'informació pressuposa 2 accessos a MP*:
  1. Buscar l'entrada de la taula de pàgines (PDT).
  2. Accedir a la dada/instrucció desitjada.

Un esquema directe de paginació duplica el temps d'accés a memòria

Gestió de la Paginació. TLB

Per solucionar el problema dels dobles accessos a MP, la majoria de processadors tenen una caché específica per a una part de les taules de pàgines anomenada:

TLB (*Translation Look-Aside Buffer*). Al cas d'Intel.
ATC (*Addres Translation Cache*). Al cas de Motorola.

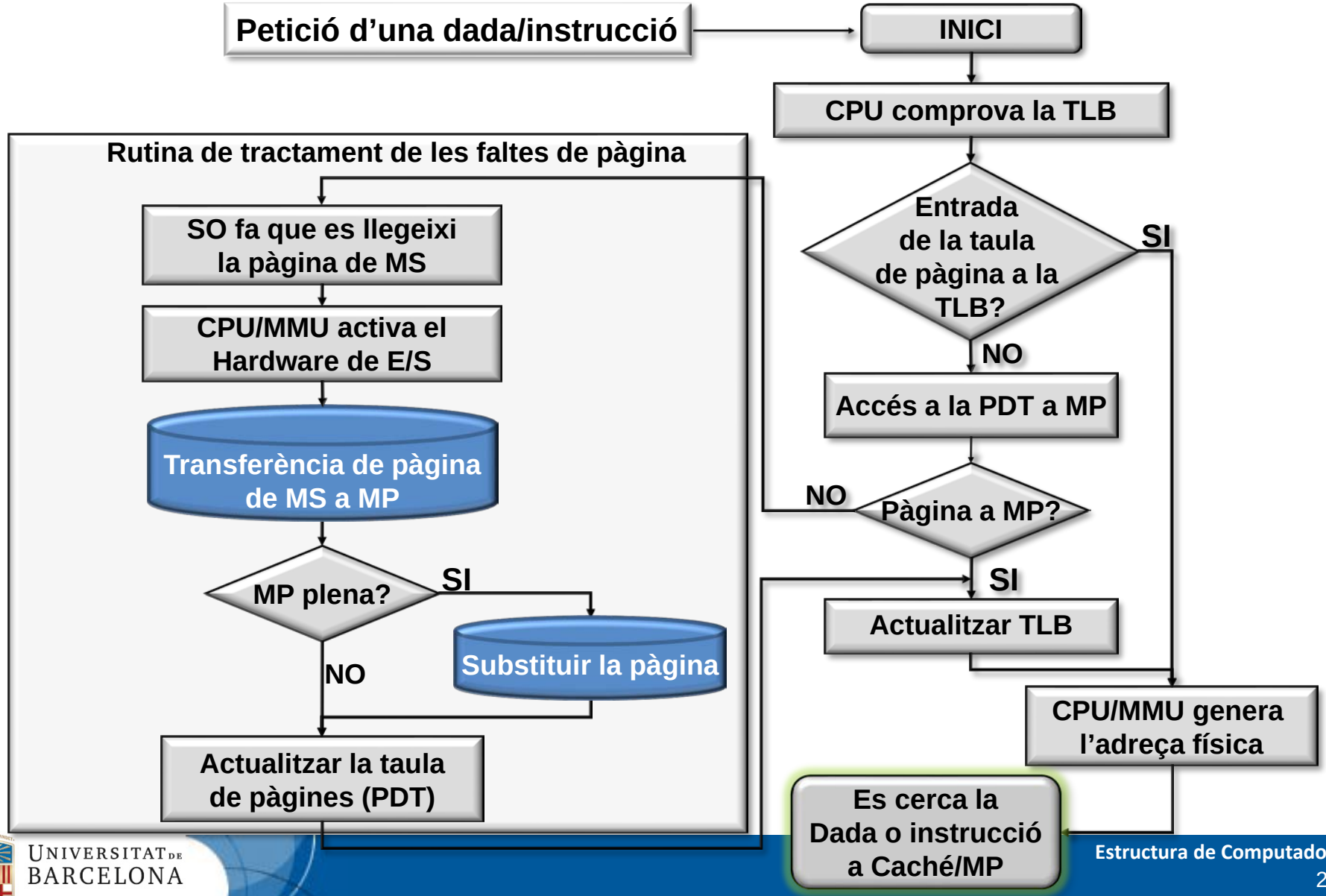
Són petites cachés amb poques entrades (de l'ordre de 32) que conté les pàgines actives (a MP) o les utilitzades recentment.

Està demostrat que millora considerablement les prestacions.

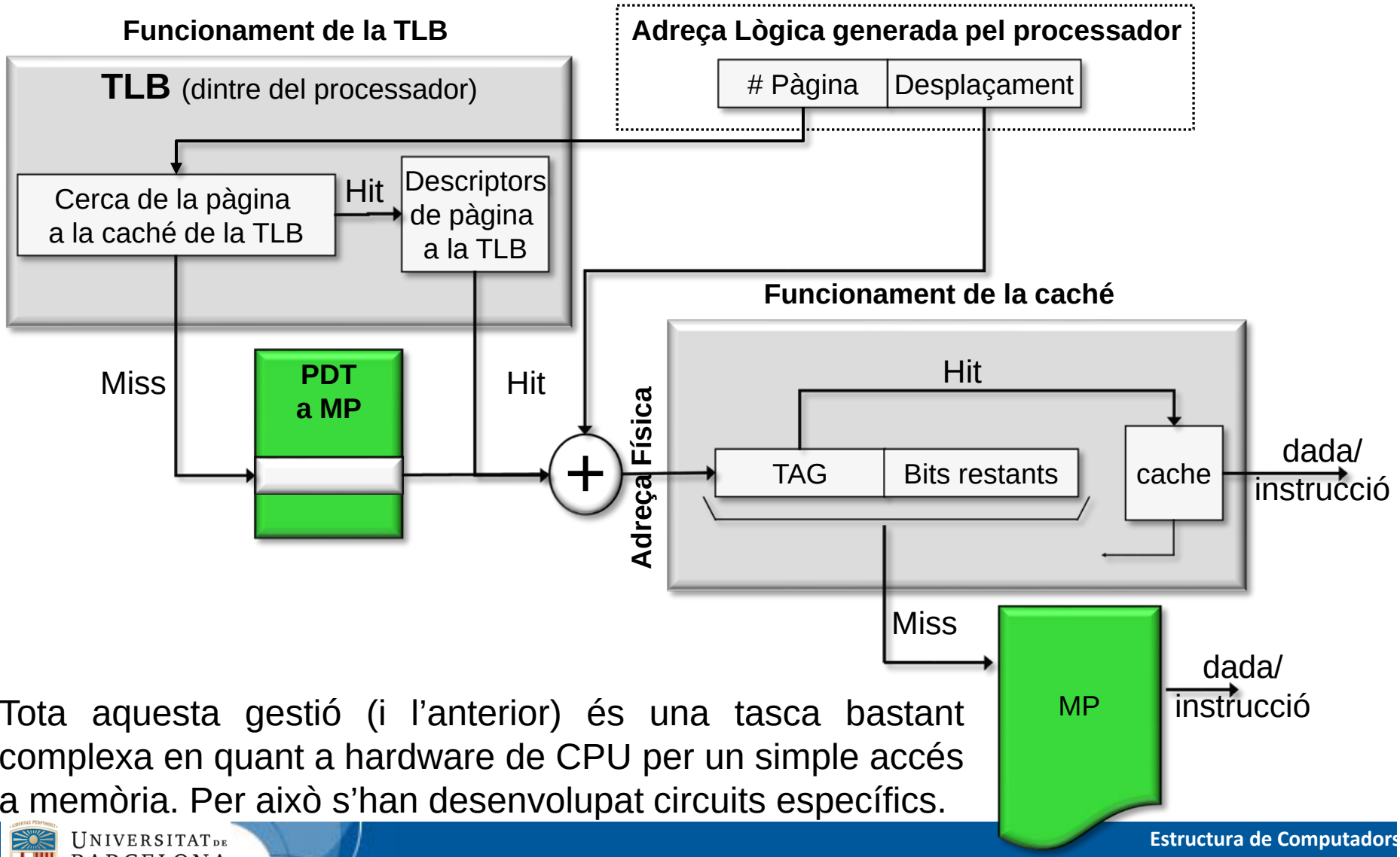
Exemple: en un sistema amb un pentium a 66MHz, un accés a una dada sense que la seva entrada es trobi TLB triga 1.8μs, mentre que si es troba l'entrada a la TLB l'accés a la dada és de 2ns (un factor 1000).

Els següent diagrames mostren el funcionament de la paginació i la TLB

Diagrama de Flux de la Gestió de la Paginació



Funcionament del conjunt TLB, PDT a MP, Caché (dades/instruccions) i MP.



Tota aquesta gestió (i l'anterior) és una tasca bastant complexa en quant a hardware de CPU per un simple accés a memòria. Per això s'han desenvolupat circuits específics.

MMU (*Memory Management Unit*)

La **MMU** (*Memory management Unit*) és un circuit especialitzat en l'administració de memòria, que proporciona les funcions descrites abans a més d'algunes afegides:

1. Assignació dinàmica de memòria (*Dynamic Memory Allocation*) per manipular eficientment l'espai de direccions físiques.
2. Implementació i gestió de la memòria virtual.
3. Genera adreces lògiques a partir de les físiques (mitjançant les TLB/PDT).
4. Proporciona protecció de memòria (o pàgines) i seguretat a les tasques, per aïllar-les entre sí (afegeix bits als *Page Descriptors* de les PDT).
5. Permet compartir parts de codi i dades en la MP a vàries tasques simultàniament.
6. Generen interrupcions perquè el SO carregui les pàgines necessàries durant l'execució d'una tasca.

Les primeres MMU eren circuits independents col·locats entre la CPU (que envia adreces lògiques) i la MP (o caché) que rep adreces físiques. Avui dia els processadors implementen la MMU internament.

SEGMENTACIÓ

Una altra forma d'administrar la memòria és la segmentació.

Proporciona al programador espais d'adreçament, **SEGMENTS,** més grans que les pàgines per organitzar programes i dades.

- Segments són de mida variable i de forma dinàmica.
- Cada segment pot tenir drets d'accés i ús.
- El programador/compilador assignarà segments a diferents parts del programa.
- Pot haver varis segments de programa (codi), així com varis segments de dades.

A diferència de la paginació (transparent al programador) la segmentació és visible al programador.

SEGMENTACIÓ

De forma similar a la paginació, per fer la traducció de adreces lògiques a adreces físiques, els processos tenen unes taules de **Descriptors de Segments** que es diuen **SDT** (*Segment Descriptor Table*).

Una adreça lògica consta del "segment usat" i del "desplaçament" (*offset*) dins del segment.

Els Descriptors de Segment a les SDT, a més de les dades que fèiem servir a les PDT, han d'indicar la mida del segment. Els camps que té cada descriptor són:

- Adreça base del segment.
- Mida del segment.
- Atributs (protecció d'accés...).

PAGINACIÓ i SEGMENTACIÓ

Tant la Paginació com la Segmentació són dos tècniques d'administració de memòria reals que es fan servir als computadors actuals.

Cap és millor que l'altre, això si algunes coses es gestionen millor amb la paginació i altres amb la segmentació:

1. La segmentació simplifica la gestió d'estructures creixents de dades, ja que els segments són de mida variable dinàmicament.
2. La segmentació permet que les unitats lògiques (programes, llibreries, estructures de dades) s'alterin i recompilin de forma independent, sense necessitat de que es carreguin i enllacin un conjunt sencer de programes.
3. Mitjançant la compartició de segments es facilita que diversos processos comparteixin funcions i dades.
4. La segmentació potencia la protecció de tasques.
5. La paginació és molt més eficaç optimitzant la MP en sistemes multitasca.

Per això, molts processadors de gama alta actuals implementen ambdós mecanismes simultàniament: paginació i segmentació.

PAGINACIÓ + SEGMENTACIÓ

Ara un Programa consta de:
- Varis Segments de Codi, Dades, Pila.....
- Cada Segment constaria de varies Pàgines