Operační systémy

Správa paměti – úvod.

Jan Trdlička

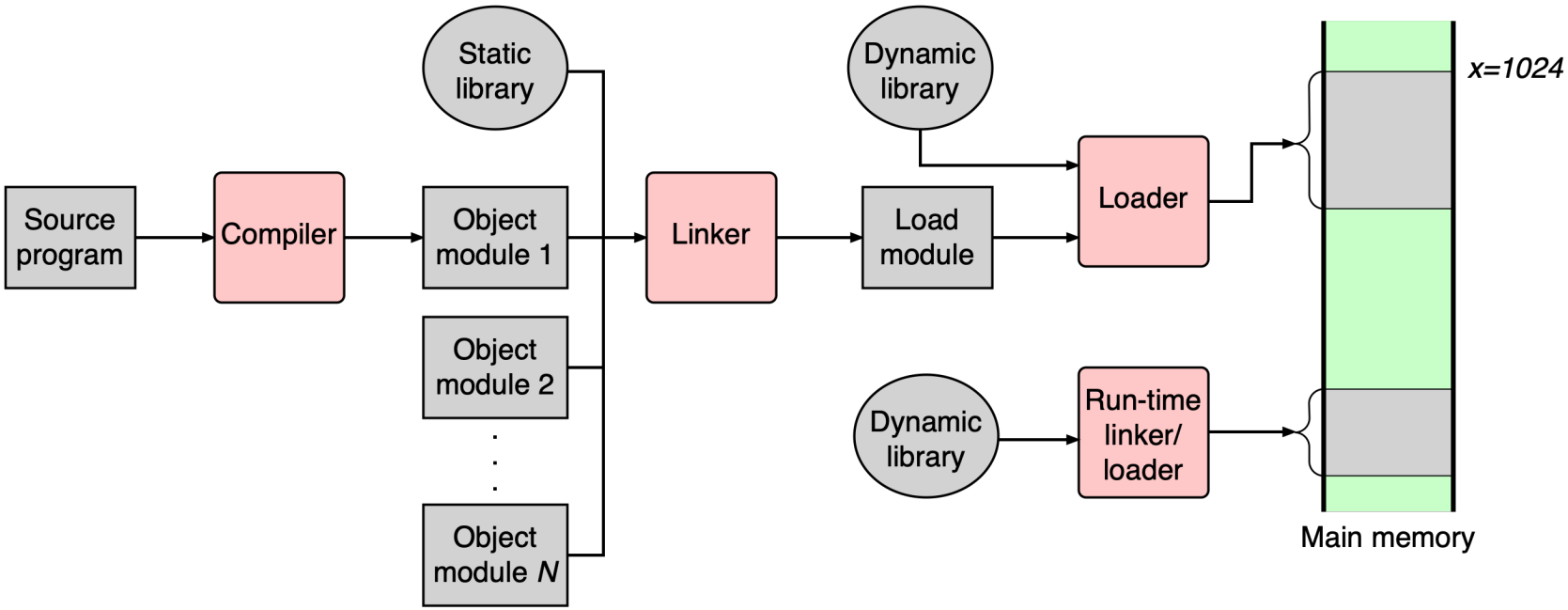
České vysoké učení technické v Praze, Fakulta informačních technologií
Katedra počítačových systémů

https://courses.fit.cvut.cz/BI-OSY

Obsah přednášky

Základní pojmy

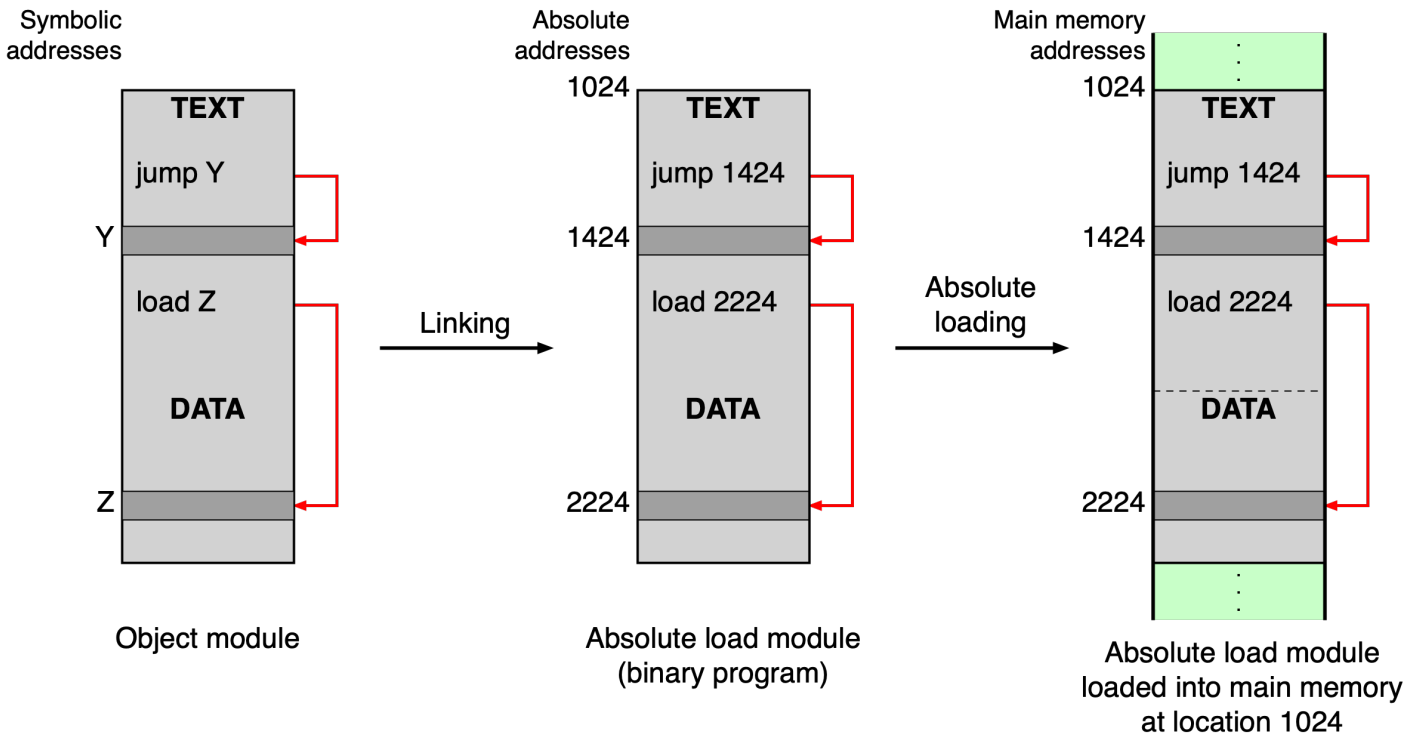
- **Kompilace, sestavení a zavedení programu do paměti.**
  - Object module: množina funkcí, proměnných a metadat zkompilovaná do přemístitelného kódu v binárním formátu, který není přímo spustitelný.
  - Load module: spustitelný binární program.
  - Formát obou typů modulů je zavislý na OS (např. ELF, PE/PE32+).
  - Static library: archiv objektových modulů $\Rightarrow$ efektivnější sestavení.
  - Dynamic library: "speciální" spustitelný binární program.



Zavedení programu do paměti (loading)

1 Absolutní zavedení (absolute loading)

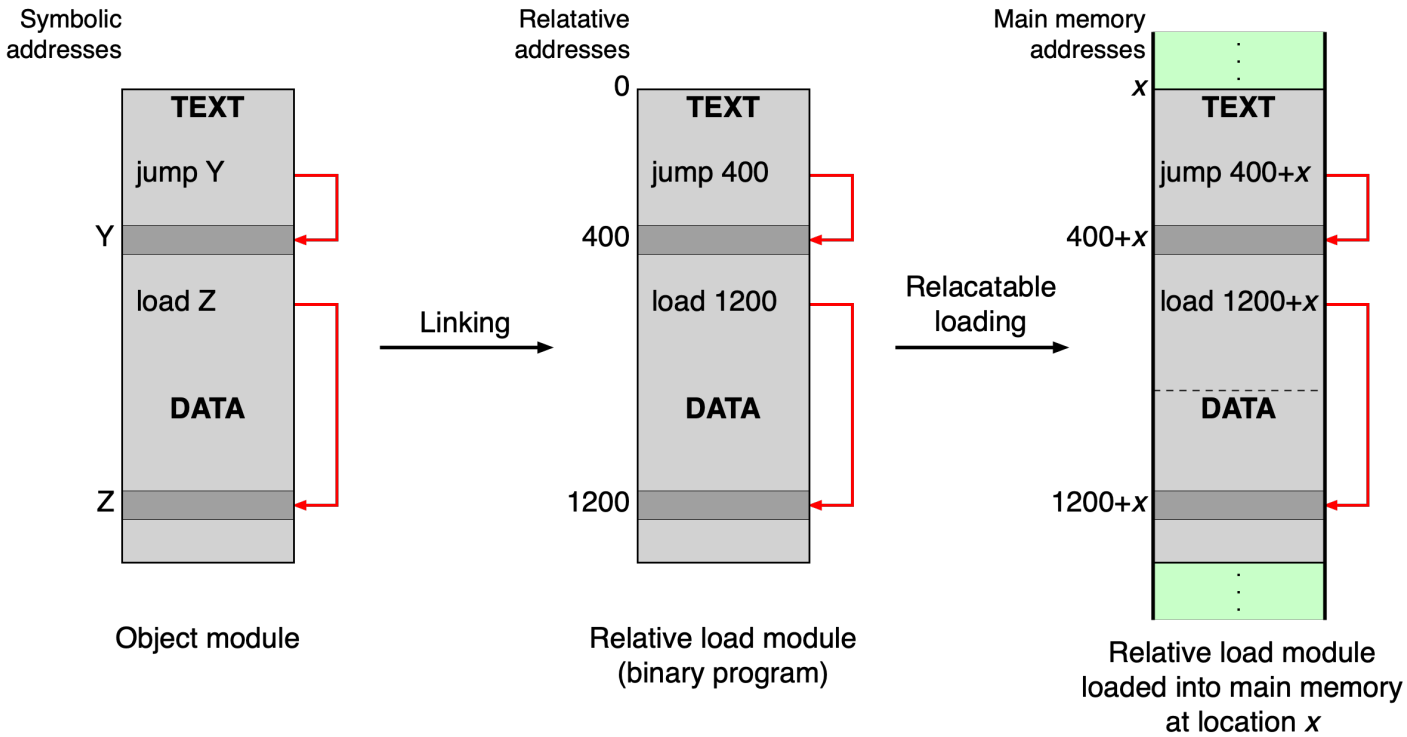
- Každý odkaz do paměti v bin. programu obsahuje absolutní adresu
  ⇒ program musí být zaveden vždy od dané fyzické adresy,
  ⇒ při sestavování musíme určit, kam bude zaveden a "Linker" vypočítá absolutní fyzické adresy.



Zavedení programu do paměti (loading)

2 Přemístitelné zavedení (relocatable loading)

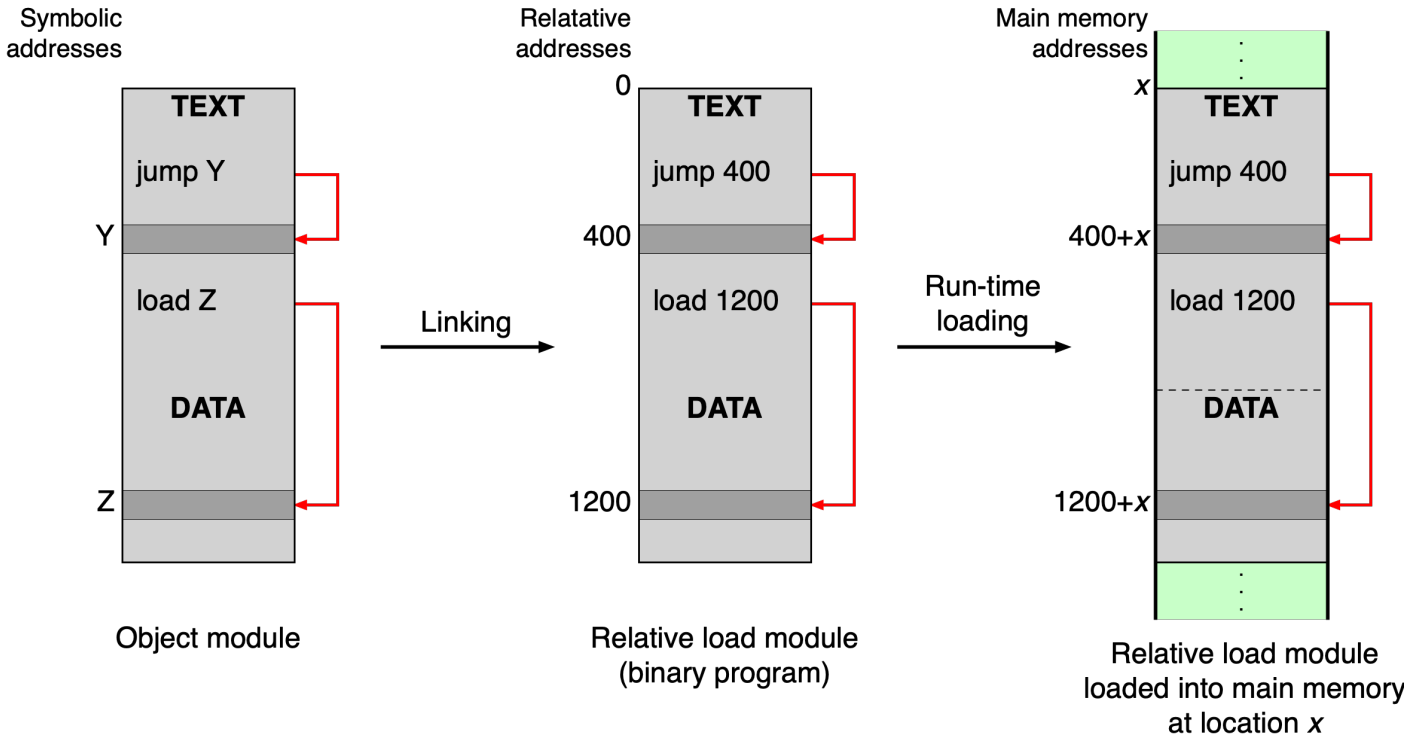
- Každý odkaz do paměti v bin. programu obsahuje relativní adresu (adresa vztažená k určitému bodu).
- "Loader" přepočítá relativní adresy na fyzické během zavedení programu do paměti.
- Informace o paměťových odkazech je uložena v "relocation dictionary" $\Rightarrow$ rychlejší přepočítání.



Zavedení programu do paměti (loading)

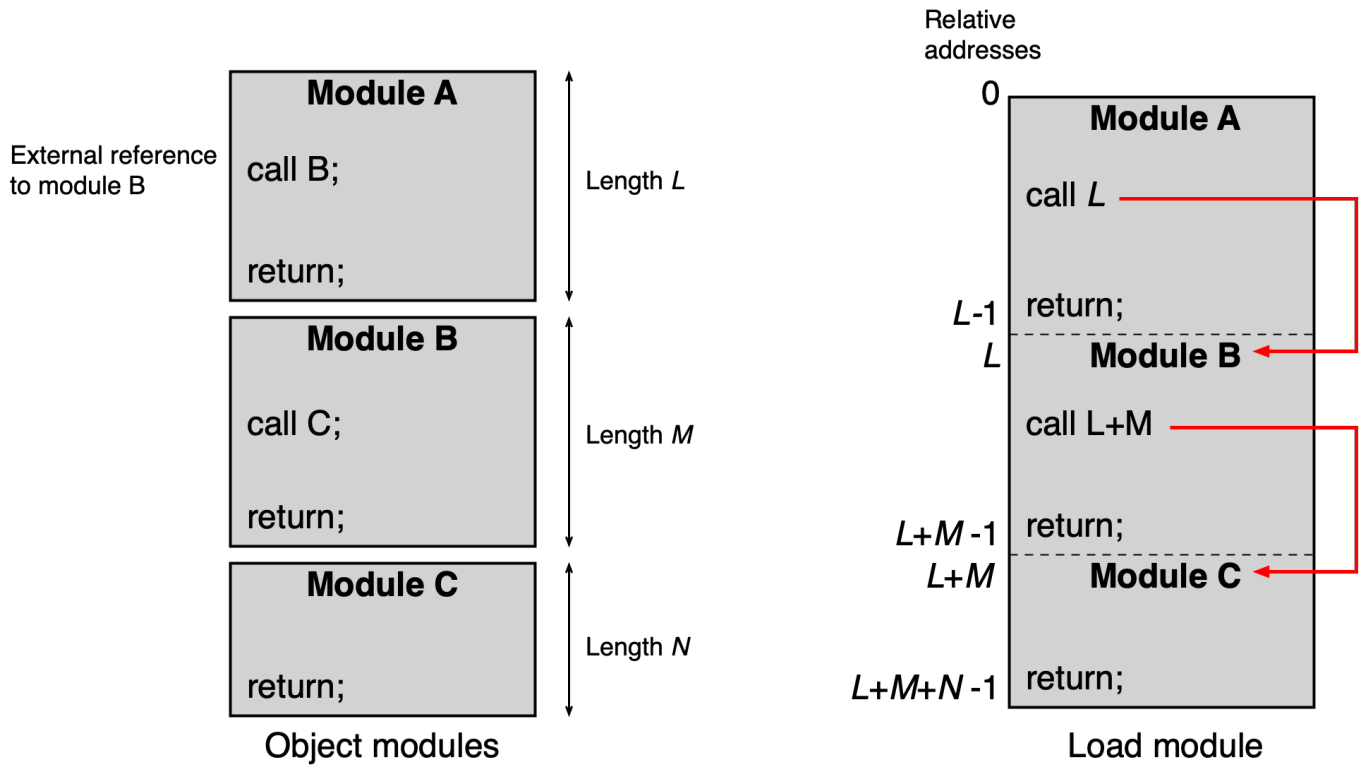
3 Dynamic run-time loading

- Každý odkaz do paměti v bin. programu obsahuje relativní adresu.
- Program je zaveden do paměti s relativními adresami.
- Relativní adresa se přepočte na fyzickou teprve v okamžiku, kdy se přistupuje k instrukci/datům na této adrese.
- Přepočtení provádí hardware (MMU = Memory management unit) ve spolupráci s OS.



Sestavení programu (linking)

- **Sestavení**
  - Sestavovací program (linker) vytvoří z jednoho/několika objektových modulů (object modules) jeden spustitelný modul (load module).
  - Symbolické adresy objektových modulů musí být nahrazeny adresami definujícími umístění uvnitř spustitelného modulu vztaženými k jeho začátku.



Sestavení programu (linking)

1. **Statické sestavení (static linking)**
   - "Linker" vytvoří z jednoho/více objektových modulů jeden samostatný spustitelný modul (load module) s absolutními adresami (absolute loading) nebo relativními adresami vztaženými k začátku modulu (relocatable loading/dynamic run-time loading).
   - Při spuštění tohoto modulu (programu) stačí do paměti nahrát pouze tento modul, který obsahuje vše potřebné.
2. **Dynamické sestavení (dynamic linking)**
   - "Linker" vytvoří spustitelný modul obsahující odkazy na další moduly (dynamické knihovny).
   - Procesy sdílejí dynamické knihovny $\Rightarrow$ v paměti je jedna instance každé knihovny.

   a. Load-time dynamic linking
      - ★ Odkazy na další moduly se nahradí v okamžiku zavedení do paměti.
      - ★ "Loader" musí zajistit, aby byly v paměti požadované knihovny.

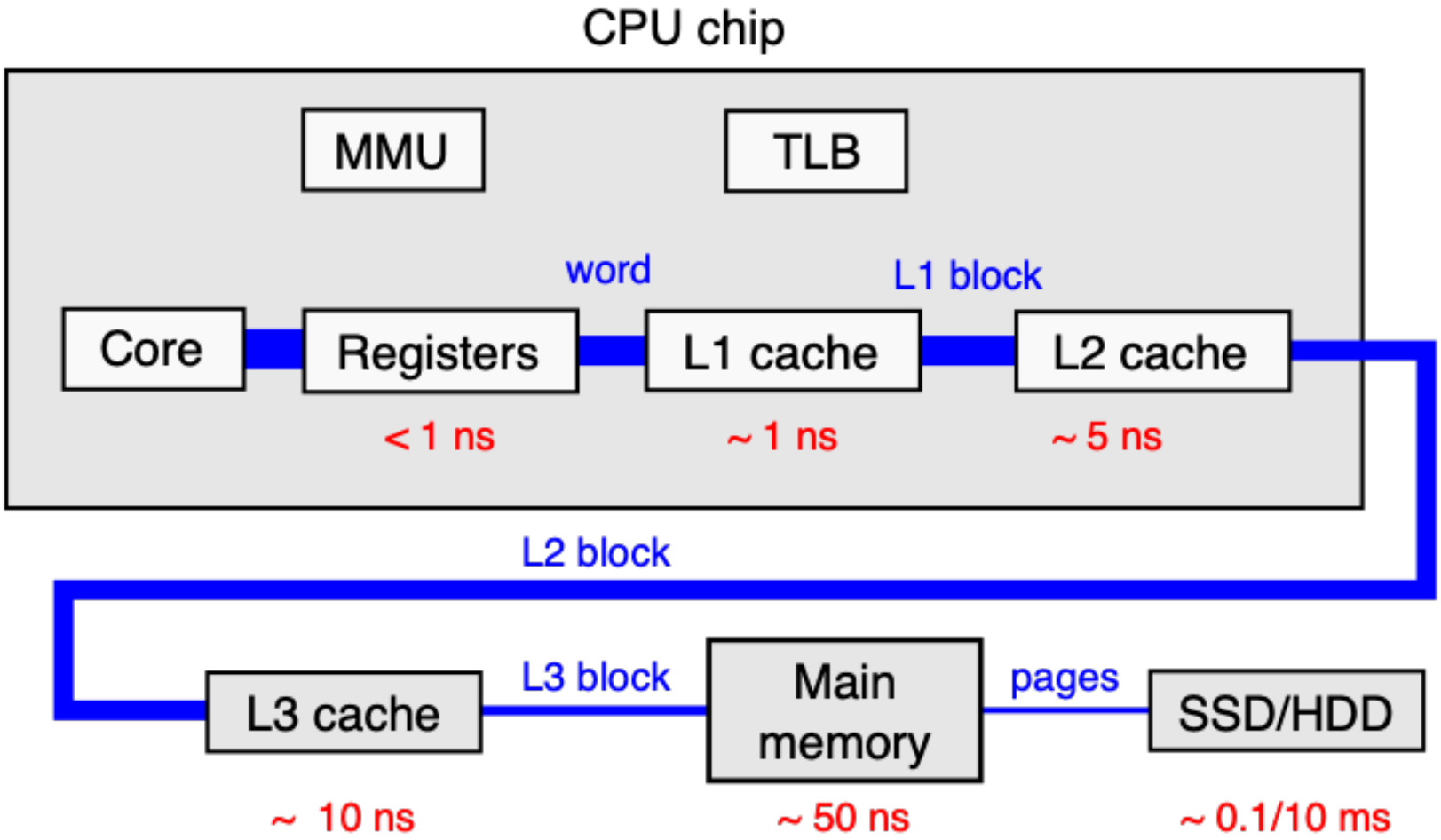
   b. Run-time dynamic linking
      - ★ Odkazy na další moduly se nahradí v okamžiku volání (viz. funkce `dlopen()` v Unixu nebo `LoadLibraryA()` ve Windows).
      - ★ "Run-time linker/loader zajistí nahrání požadovaných knihoven.

Fyzická organizace paměti

- Z důvodu rostoucího výkonu CPU na jedné straně a "pomalých" pamětí na druhé straně má fyzická paměť hierarchickou organizaci.
- Pomocí několika úrovní skrytých pamětí (cache) se minimalizuje průměrný čas přístupu k instrukcím/datům.
- Pro adresaci instrukcí/dat v hlavní paměti se používají fyzické adresy vztažené k začátku fyzické paměti.

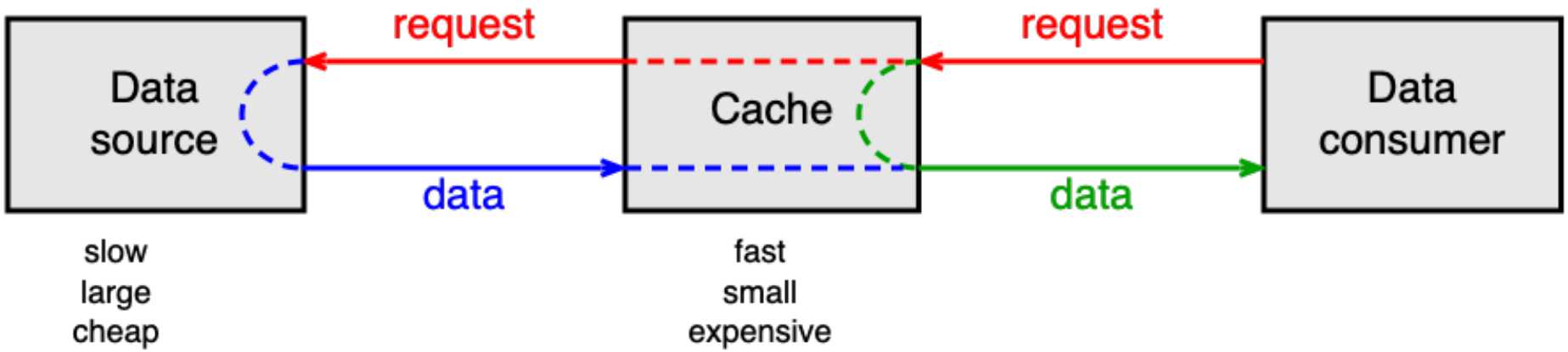
Typ paměti	Velikost	Čas přístupu	Kdo spravuje
Registry CPU	1KB	<1ns	Překladač/programátor
L1 cache	64 KB	~1 ns	HW
L2 cache	256 KB	~5 ns	HW
L3 cache	4 MB	~10 ns	HW
Hlavní paměť	1 GB – 16 TB	~50 ns	OS
SSD	250 GB – 4TB	~0.1 ms	OS/procesy
HDD	500 GB – 16 TB	~10 ms	OS/procesy
Flash paměť	4 GB – 4 TB	~100 ms	OS/procesy
Páska	500 GB – 16 TB	>1s	OS/procesy

Fyzická organizace paměti



MMU = memory management unit
TLB = translation lookaside buffer

Skrytá paměť (cache)



- Skrytá paměť umožňuje **minimalizovat průměrnou dobu přístupu** k datům a je založena na následujících principech.
  - Časová lokalita: k datů, ke kterým se přistupovalo nedávno, se bude přistupovat i v blízké budoucnosti (skrytá paměť je "malá" $\Rightarrow$ obsahuje pouze nedávno používaná data).
  - Prostorová lokalita: k datům, která jsou v okolí aktuálně používaných dat, se bude přistupovat v blízké budoucnosti (do skryté paměti se mohou načítat požadovaná data včetně okolních).
- **Příklady použití skryté paměti**
  - Všude, kde platí aspoň jeden z předchozích principů.
  - CPU (L1, L2 a L3 cache), disky, RAID, systémy souborů.

Skrytá paměť (cache)

- **Výkonnostní parametry**
  - Cache hit count(n_h): počet případů, kdy data byla ve skryté paměti,
  - Hit time (t_h): čas přístupu k datům ve skryté paměti,
  - Cache miss count(n_m): počet případů, kdy data nebyla ve skryté paměti,
  - Miss penalty (t_m): čas přístupu k datům ve zdroji dat (data source),
  - Cache reference: celkový počet přístupů k datům $n_r = n_h + n_m$,
  - **Cache Hit Ratio:** $r_h = \frac{n_h}{n_r} = \frac{n_h}{n_h+n_m}$,
  - Average Access Time: $t_{avg} = t_h + (1 - r_h) \times t_m$.
- **Příklad**
  - Předpokládejme následující parametry
    - L1 cache: $t_h = 1$ ns,
    - hlavní paměť: $t_m = 50$ ns.
  - Jaký bude t_{avg} pro různě velké r_h?
    a. $r_h = 1.00$: $t_{avg} = 1 + (1 - 1.00) \times 50 = 1$ ns,
    b. $r_h = 0.90$: $t_{avg} = 1 + (1 - 0.90) \times 50 = 6$ ns,
    c. $r_h = 0.80$: $t_{avg} = 1 + (1 - 0.80) \times 50 = 11$ ns,
    d. $r_h = 0.00$: $t_{avg} = 1 + (1 - 0.00) \times 50 = 51$ ns.

Skrytá paměť (cache)

- **CPU**
  - Některé procesory umožňují monitorování svého výkonu pomocí programovatelných hardwarových čítačů, které se inkrementují při výskytu dané události (např. cache reference, cache miss, ...).
  - Tyto informace lze zjistit v OS pomocí přílušných příkazů/aplikací (např. `cpustat`, `cputrack`, DTrace, SystemTap,...)
    $\Rightarrow$ můžeme zjistit, jak se systém/aplikace chová,
    $\Rightarrow$ můžeme opravit chyby v aplikaci (např. změnit přístup k datům).
- **Příklad**
  - Pomocí následujícího příkazu můžeme zjistit "cache reference" (EC_ref) a "cache miss" (EC_misses) skrytých pamětí L3 na dvou instalovaných CPU v následujících 3 sekundách.

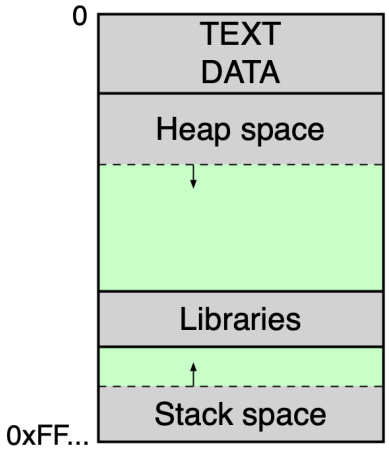
```
root@solaris:~ > cpustat -c EC_ref,EC_misses 1 3

    time cpu event      pic0      pic1
   1.008   0  tick     69284      1647
   1.008   1  tick     43284      1175
   2.008   0  tick    179576      1834
   2.008   1  tick    202022     12046
   3.008   0  tick     93262       384
   3.008   1  tick     63649      1118
   3.008   2 total    651077     18204
```

Logická organizace paměti

Virtuální adresový prostor procesu (VAS)

- Každý proces má svůj VAS, který se mapuje do fyzické paměti.
- Jakým způsobem bude VAS procesu mapován do fyzické paměti závisí na konkrétní architektuře CPU a použitém OS.
- **VAS typicky obsahuje oblasti pro následující struktury**



- program (TEXT),
- globální proměnné (DATA),
- halda (Heap space),
- knihovny (Libraries),
- zásobník/zásobníky (Stack space)

- Při rozmístění struktur ve VAS a při alokaci fyzické paměti je nutné pamatovat na to, že některé struktury (halda, zásobník) mění svojí velikost během existence procesu.
- Některé struktury ve VAS mohou být sdílené mezi procesy (TEXT, knihovny,...).
- **Address space layout randomization (ASLR)**: z důvodu bezpečnosti (ochrana např. proti útoku typu buffer-overflow, ...) mohou být struktury ve VAS umisťovány na náhodné adresy.

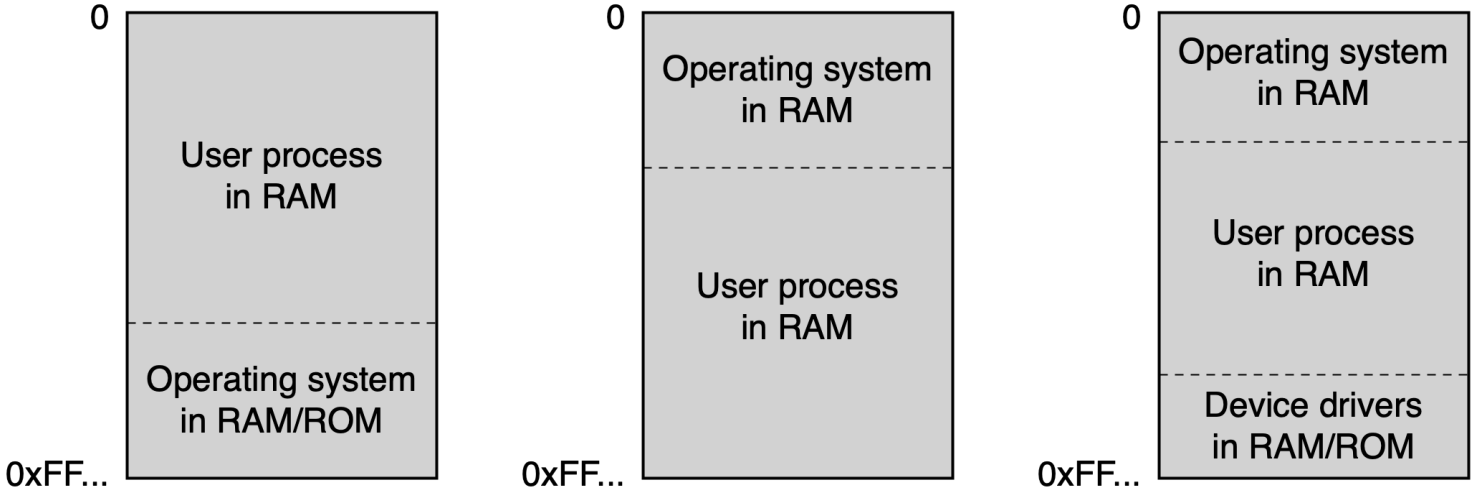
Implementace VAS

- VAS může být implementován různými způsoby.
- Podle počtu segmentů VAS (jeden/více), podle počtu procesů v fyzické paměti, a podle způsobu alokace fyzické paměti jednotlivým procesům (souvislé/nesouvislé oblasti) můžeme rozlišit následující implementace.

1. **VAS jako jeden jednorozměrný logický prostor (segment)**
   - Ve fyzické paměti bude pouze jeden proces.
     a. VAS identický s fyzickou pamětí: VAS je jednorozměrný prostor alokovaný jako souvislá oblast ve fyzické paměti.
   - Ve fyzické paměti může být současně více procesů.
     b. Dynamické oblasti: VAS alokován jako souvislá oblast ve fyzické paměti.
     c. Stránkování: VAS alokován jako množina stejně velkých oblastí (stránek) fyzické paměti.
2. **VAS jako několik jednorozměrných segmentů $\Rightarrow$ segmentace**
   - Ve fyzické paměti může být současně více procesů a VAS je složen z několika segmentů (jednorozměrný logický prostor).
     a. Dynamické oblasti: VAS alokován jako několik segmentů, kde každý segment je souvislá oblast ve fyzické paměti.
     b. Stránkování: VAS alokován jako několik segmentů, kde každý segment je množina stejně velkých oblastí (stránek) fyzické paměti.

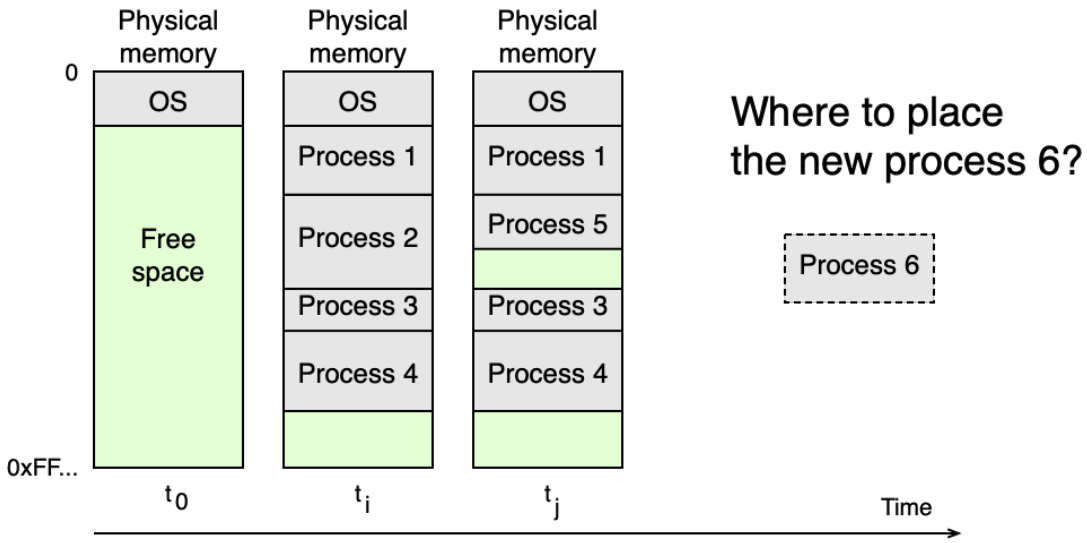
Implementace VAS – VAS identický s fyzickou pamětí

- Nejjednodušší implementace, která byla/je typická pro "main-frame" počítače (před 1960), mini počítače (před 1970), osobní počítače (před 1980) a pro některé současné jednoduché vestavěné (embedded) systémy.
- Ve fyzické paměti se nachází pouze OS a jeden uživatelský proces, který je nahrán vždy od stejné adresy (absolutní zavedení).
- Umístění OS a uživatelského procesu se může lišit pro různé architektury.



Implementace VAS – Dynamické oblasti

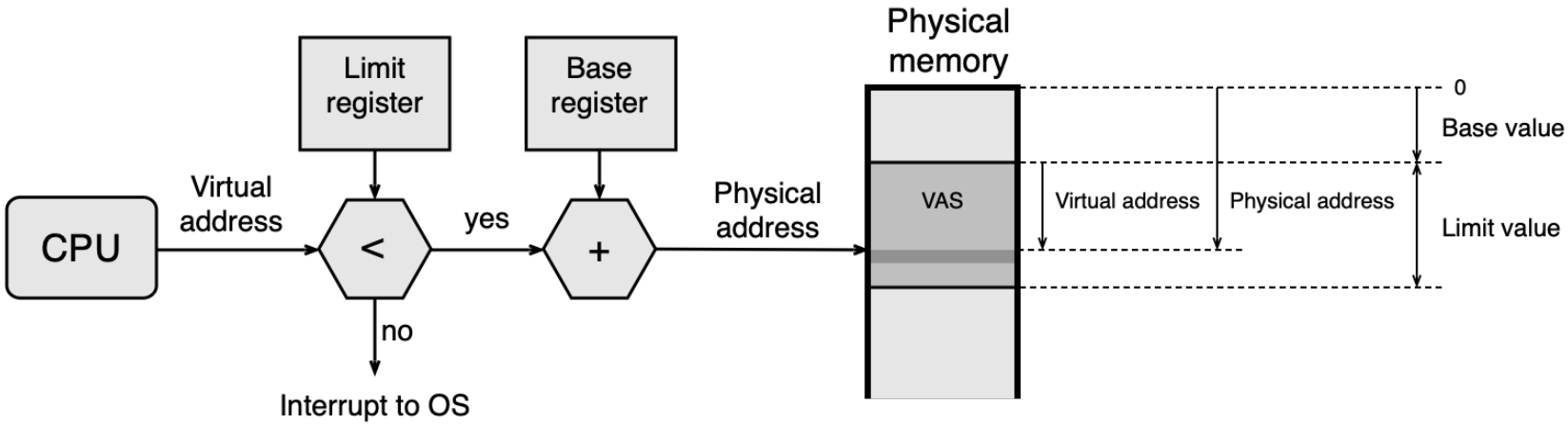
- Jeden ze způsobů, jak mapovat VAS do fyzické paměti, je alokovat jednu souvislou oblast fyzické paměti pro celý VAS.
- Tento způsob je označovaný jako "alokování pomocí dynamických oblastí" (Dynamic partitioning) a byl používaný v "main-frame" počítačích.
- Ve fyzické paměti je OS a typicky několik uživatelských procesů.



- **Problém s fragmentací fyzické paměti**
  - Externí fragmentace: po určitém čase je volná paměť reprezentována příliš malými oblastmi, do kterých se již nevejdou nově vznikající procesy.
  - Interní fragmentace: VAS obsahuje volnou paměť, do které se může rozpínat halda a zásobníky vláken.
- Při této alokaci paměti vznikají problémy, jejichž řešení vedlo ke vzniku virtuální paměti, stránkování a segmentace.

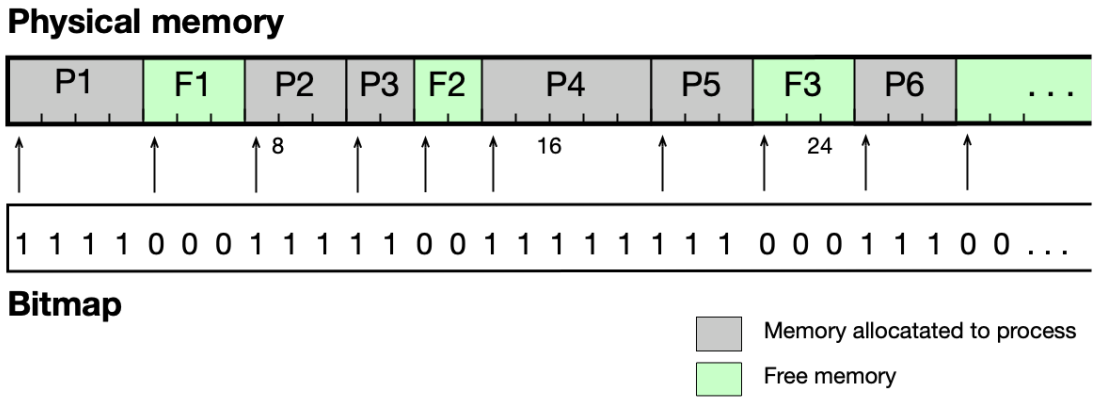
Implementace VAS – Dynamické oblasti

- **Odkládání procesů (swapping)**
  - Technika, kdy při nedostatku paměti, se "vhodné" procesy (např. ve stavu "Blocked") dočasně přesunou na disk a uvolněné místo se přidělí dalším procesů $\Rightarrow$ řeší dočasný nedostatek fyzické paměti.
- **Přepočítávání logických adres na fyzické**
  - Relocatable loading.
  - Dynamic run-time loading.
    - ★ Řešení pomocí dvou hardwarových registrů: Base registr obsahuje fyzickou adresu začátku oblasti (relocation) a Limit register obsahuje velikost oblasti (oba registry současně zajišťují ochranu VAS).



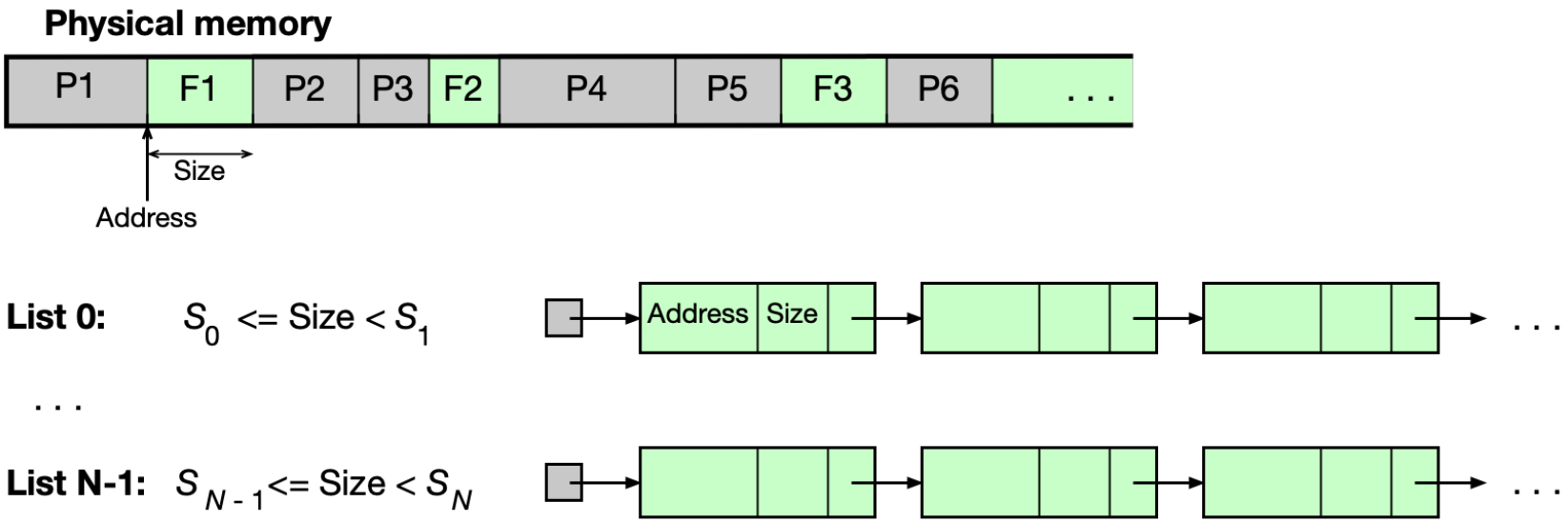
Implementace VAS – Dynamické oblasti

- **Správa volné paměti**
  - Jedním z úkolů OS při správě paměti je udržování informace o volných oblastech fyzické paměti a jejich velikostech. K tomuto účelu lze použít bitovou mapu nebo zřetezené seznamy.
- **Bitová mapa**
  - Můžeme si představit, že fyzická paměť je složená z malých alokačních jednotek stejné velikosti. Potom v bitové mapě bude jeden bit (0=volná/1=alokovaná) reprezentovat jednu alokační jednotku.
  - Alokace paměti: nalezení řetězce nul požadované délky $\Rightarrow$ pomalé.
  - Uvolnění paměti: změna příslušného řetězce jedniček na nuly.



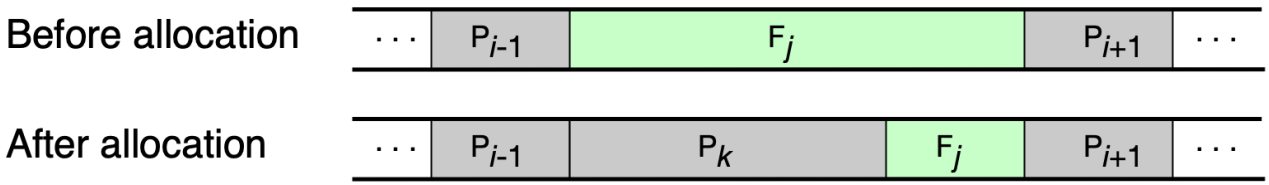
Implementace VAS – Dynamické oblasti

- **Zřetězené seznamy**
  - Informaci o volných oblastech budeme uchovávat v N zřetězených seznamech L_i, $1 \leq i \leq N$.
  - Každý seznam L_i bude obsahovat informace pouze o oblastech určitých velikostí. Nechť jsou definovány hodnoty $S_0 < \ldots < S_N$. V seznamu L_i budou informace pouze o volných oblastech, pro jejichž velikost S platí $S_i \leq S < S_{i+1}$.

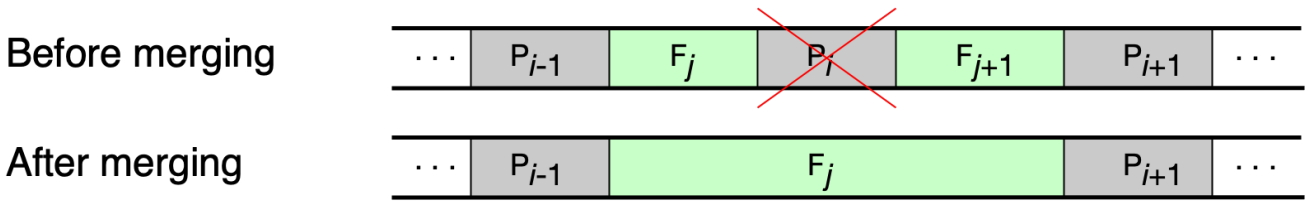


Implementace VAS – Dynamické oblasti

- **Zřetězené seznamy**
  - Alokace paměti: Pokud chceme alokovat paměť velikosti S, pak použijeme první volnou oblast z příslušného seznamu L_i (neprázdný seznam s nejnižším i, kde $S_i \geq S$), ze které jednu část alokujeme procesu a informaci o druhé volné části (pokud bude existovat) přidáme do příslušného seznamu. $\Rightarrow$ rychlé.



  - Uvolnění paměti: Při uvolnění paměti musíme zjistit, zda před nebo za touto pamětí není volná oblast, a pokud ano, pak tyto oblasti musíme sloučit do jedné větší volné oblasti a umístit do příslušného seznamu $\Rightarrow$ pomalé.

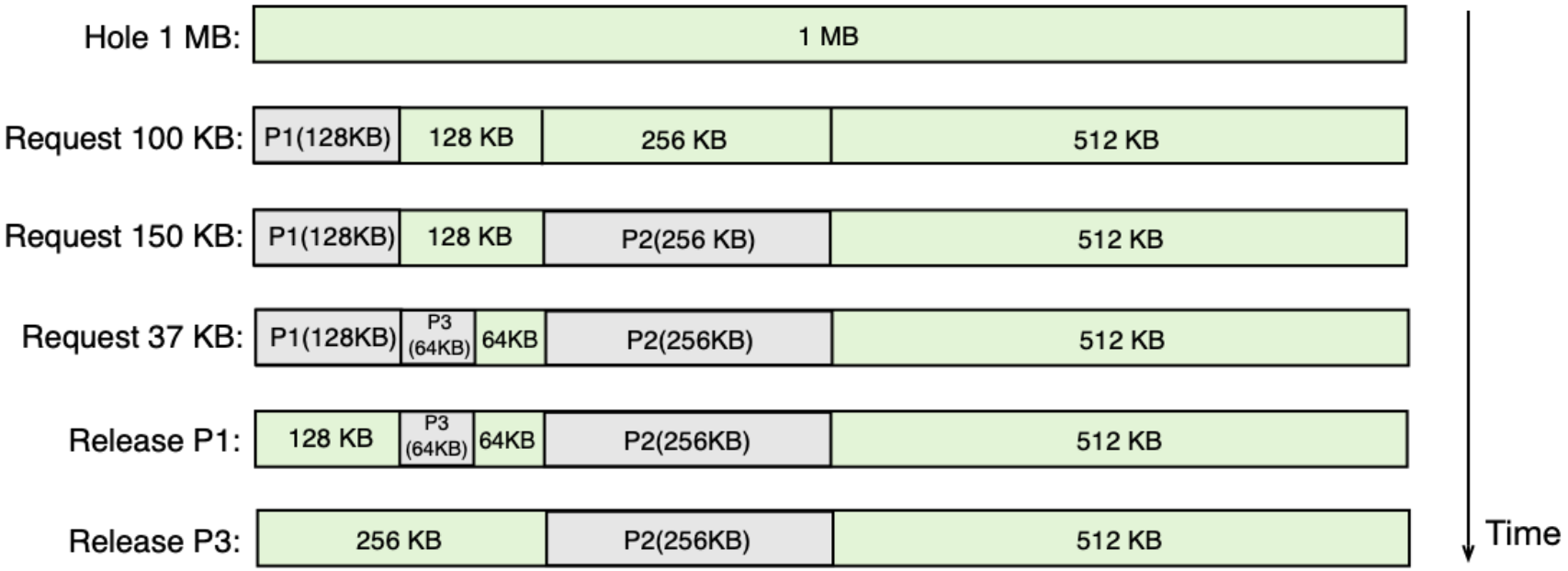


Implementace VAS – Dynamické oblasti

- **Buddy systém**
  - Varianta zřetězených seznamů, ve kterých volné oblasti mají pouze velikosti, které jsou mocninou dvou. To umožní rychleji slučovat uvolněnou paměť.
  - Alokace paměti
    - ★ Předpokládejme, že na počátku je celá paměť volná a má velikost 2^M.
    - ★ Pokud nový proces požaduje paměť o velikost S, pro kterou platí $S = 2^M$, potom mu přidělíme celou oblast 2^M.
    - ★ Jinak oblast 2^M budeme rekurzivně dělit na poloviny (vždy jednu z polovin), dokud nezískáme oblast 2^i, pro kterou $2^{i-1} < S \leq 2^i$ a tu přidělíme procesu.
    - ★ Nově vzniklé volné oblasti přidáme do příslušných seznamů $\Rightarrow$ rychlé.
  - Uvolnění paměti
    - ★ Opačný postup než při alokaci.
    - ★ Rekurzivně slučujeme odpovídající poloviny dokud to jde (dokud existují) $\Rightarrow$ rychlé.

Implementace VAS – Dynamické oblasti

- **Příklad: Buddy systém**
  - Na počátku je paměť o velikosti 1M je prázdná.
  - Postupně jsou vytvořeny procesy P1, P2, P3, které vyžadují paměť o velikostech minimálně 100KB, 150KB a 37KB.
  - Na závěr se ukončí proces P1 a P3.



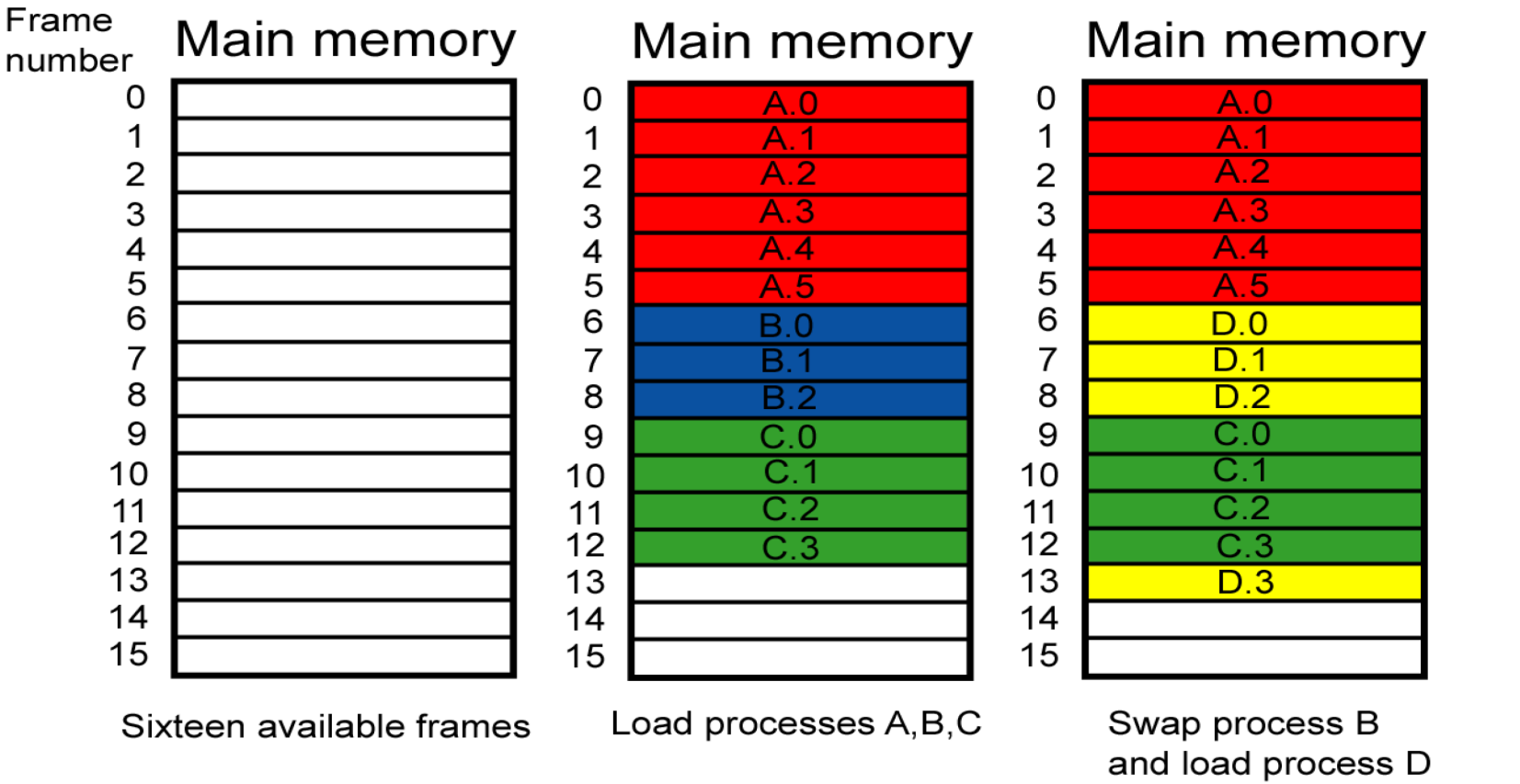
Implementace VAS – Dynamické oblasti

- **Problémy**
  - Fyzická paměť je po určitém čase fragmentovaná (obsahuje mnoho malých volných oblastí)
    $\Rightarrow$ problém najít dostatečně velkou souvislou oblast pro nový proces.
- **Řešení**
  - VAS budeme alokovat jako množinu malých oblastí
    $\Rightarrow$ odpadne problém s nalezením volné velké souvislé oblasti,
    $\Rightarrow$ stránkování.

Implementace VAS – Stránkování

- **Fyzická paměť (hlavní paměť)**
  - Představme si, že je rozdělená na úseky stejné velikosti (např. 4KB) nazývané rámce (frames).
- **VAS**
  - Představme si, že je rozdělený na úseky stejné velikosti nazývané stránky (pages).
- Velikost rámce a stránky je stejná.
- Jednotlivé stránky VAS se nahrávají do volných rámců fyzické paměti.
- **OS si musí pamatovat**
  - rámce přidělené jednotlivým procesům (např. pomocí tabulky stránek,...),
  - volné rámce.

Implementace VAS – Stránkování



Použité zdroje

1. A. S. Tanenbaum, H. Bos: *Modern Operating Systems (4th edition)*, Pearson, 2014.
2. W. Stallings: *Operating Systems: Internals and Design Principles (9th edition)*, Pearson, 2017.
3. A. Silberschatz, P. B. Galvin, G. Gagne: *Operating System Concepts (9th edition)*, Wiley, 2012.