Operačné systémy

Manažment Pamäte 3.

Ing. Martin Vojtko, PhD.

2024/2025

1. Návrh stránkovania
   - Alokačná politika
   - Zdieľanie stránok
2. Implementácia stránkovania
   - Spracovanie Page Fault-u
   - Úložisko vyhodených stránok
3. Algoritmy výmeny stránok
   - Jednoduchšie Algoritmy
   - Least Recently Used (LRU)
   - Working Set (WS)
4. Zhrnutie

Návrh stránkovania

Demand Paging

Demand Paging

je technika kedy sa stránky procesu načítajú do hlavnej pamäte keď ich je treba.

- Pri štarte procesu sa postupne načítajú stránky (program, dáta a zásobník).
- V prípade ak máme proces, ktorý načítame zo swap-u situácia je podobná.
- Výsledkom je množstvo Page Faults pri spúšťaní procesu.

Pre-paging

Pre-paging

je technika kedy sa vybrané stránky procesu načítajú do hlavnej pamäte ešte pred tým, ako sa proces začne vykonávať.

Working Set (WS)

je množina vybraných stránok, ktoré sú procesom používané najčastejšie v danom okamihu. Množina sa s časom môže meniť podľa toho v akej časti životného cyklu proces momentálne je.

- OS pre jednotlivé procesy určuje working set na základe používanosti ich stránok.
- Veľkosť setu je určená alokačnou politikou, množstvom procesov, množstvom pamäte a prioritou procesu.

Veľkosť stránky

- Od veľkosti stránky závisí:
  - Interná fragmentácia poslednej stránky segmentu.
  - Množstvo neaktívnej pamäte v hlavnej pamäti.
  - Počet záznamov v tabuľkách stránok.
  - Rýchlosť vyhľadania a čítania stránky z disku.
  - Efektivita využitia TLB.

Interná fragmentácia

ak n je počet segmentov procesu a p je veľkosť stránky, tak priemernú internú fragmentáciu vieme vyjadriť: $f = np/2$

Réžia stránkovania

ak s je priemerná veľkosť procesu v B, p je veľkosť stránky v B, e je veľkosť záznamu v tabuľke stránok tak, priemerný počet B v tabuľke stránok je se/p a interná fragmentácia pre jeden segment je $p/2$
celková réžia stránkovania je: $r = se/p + p/2$

Alokačná politika

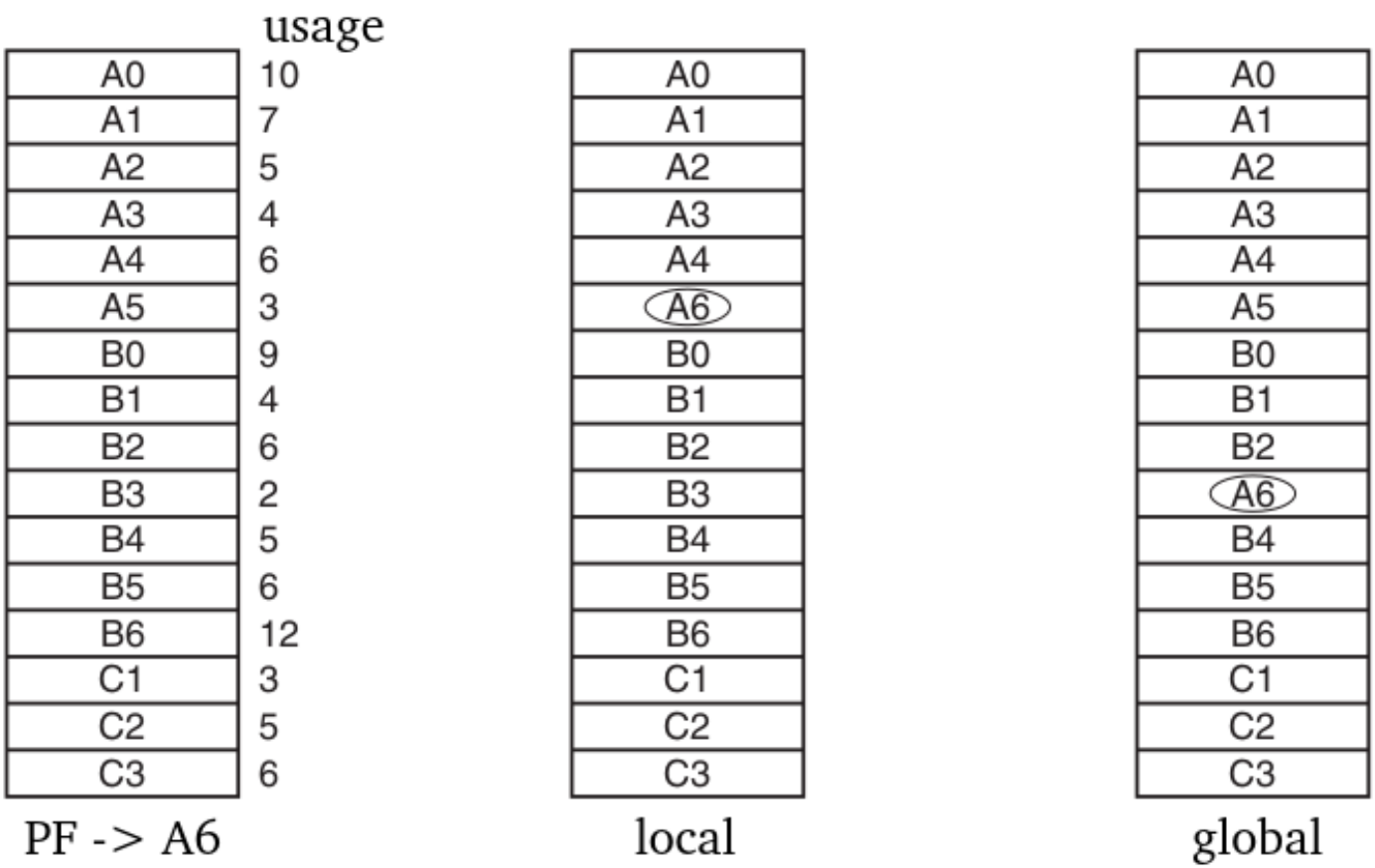
Lokálna Alokačná politika

Každý process má svoj vlastný vyhradený priestor pre stránky. Algoritmus výberu obete vyberá len rámce patriace aktuálnemu procesu.

Globálna Alokačná politika

OS udržiava informácie o rámcoch bez rozlišovania jednotlivých procesov. V prípade Page Fault sa vyberá obeť, ktorá je všeobecne najmenej používaná. Procesy môžu mať rôzne veľké WS.

Alokačná politika



Alokačná politika

- Local
  - Fixné maximum pridelených rámcov.
  - ak WS $>$ max : dochádza k tzv. thrashing.
  - ak WS $<$ max : pamäť je nevyužitá.
- Global
  - Variabilné pridelenie rámcov v závislosti od veľkosti WS.
  - Pri vyššej záťaži umožňuje odložiť všetky stránky vybraného procesu. (SWAP)

Process Thrashing

je výrazné spomalenie vykonávania procesu z dôvodu častého výpadku stránok. Môže byť spôsobený zlým výberom obetí pri riešení Page Fault-u alebo nedostatkom voľných stránkových rámov procesu. Bežne môže dochádzať k thrashing ak WS je väčší ako maximálny pridelený počet rámov procesu.

Politika čistenia pamäte

- Ak nastane Page Fault potrebujeme hľadať voľný stránkový rám.
- Ak nenájdeme voľný rám tak musíme hľadať obeť.
- Ak nájdeme modifikovanú obeť musíme ju zapísať do pamäte.
- Ak by sme zabezpečili, že vždy bude v pamäti voľný priestor. Obmedzíme zdržanie procesu.

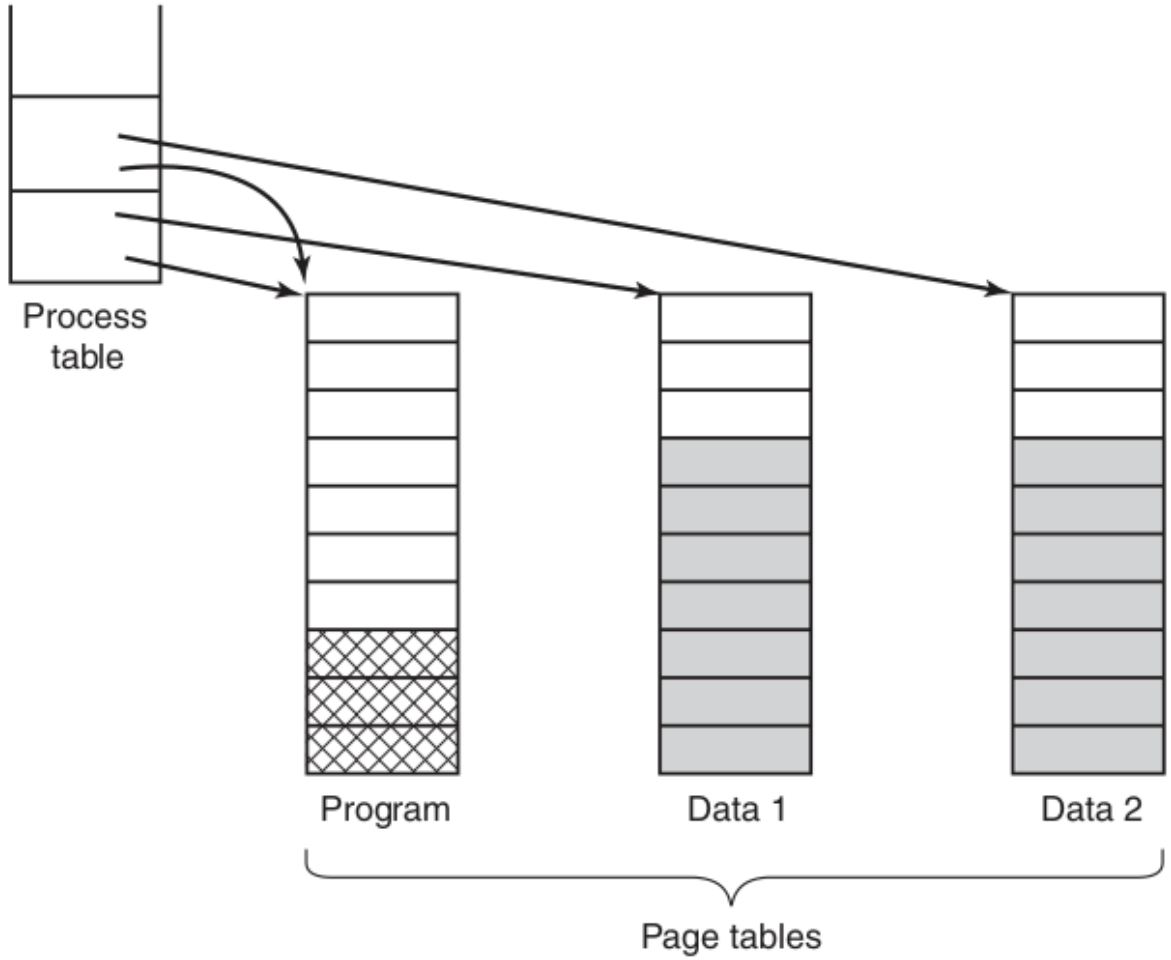
Paging Daemon

je proces OS, ktorý väčšinu času spí a je periodicky prebúdzaný. Ak je pamäte málo proces vyberá stránky, ktoré sú odstránené. Ak je stránka modifikovaná proces zabezpečí jej uloženie na disk.

Zdieľanie stránok programu = úspora pamäte

- Program a konštantné dáta sú uložené v stránkach, do ktorých sa nedá zapisovať (RO).
- Ak dva procesy sú inštanciami toho istého programu je možné RO stránky v pamäti zdieľať.
- Problémy:
  - Potrebujeme segmenty alebo nejakú formu označenia zdieľania stránok.
  - Čo ak jeden proces skončí.
  - Čo ak jeden proces uložíme do pamäte (swap).
  - Čo ak OS vylúči procesu stránku, ktorú druhý používa.
- RW stránky sa dajú zdieľať kým ich obsah je totožný. (copy-on-write)
  - fork() nastaví všetky RW stránky ako RO.
  - ak proces chce zapísať do stránky vyvolá sa TRAP OS.
  - OS urobí kópiu stránky a obe nastaví ako RW.

Zdieľanie stránok programu



Zdieľanie knižníc

- Bežný program používa zdroje z už existujúcich knižníc.
- Knižnice redukujú duplikáciu kódu pri vývoji programu.
- Knižnice môžeme linkovať staticky, kedy sa kód knižnice stane súčasťou kódu procesu. Takéto riešenie je neefektívne nakoľko sa duplikuje kód.
- Knižnice môžeme linkovať dynamicky, kedy do programu odkladáme referencie jednotlivých funkcií z knižníc.
- Knižnice sú linkované do tzv. dynamic link libs resp. shared objects. (.dll, .so)

Zdieľanie knižníc - výhody

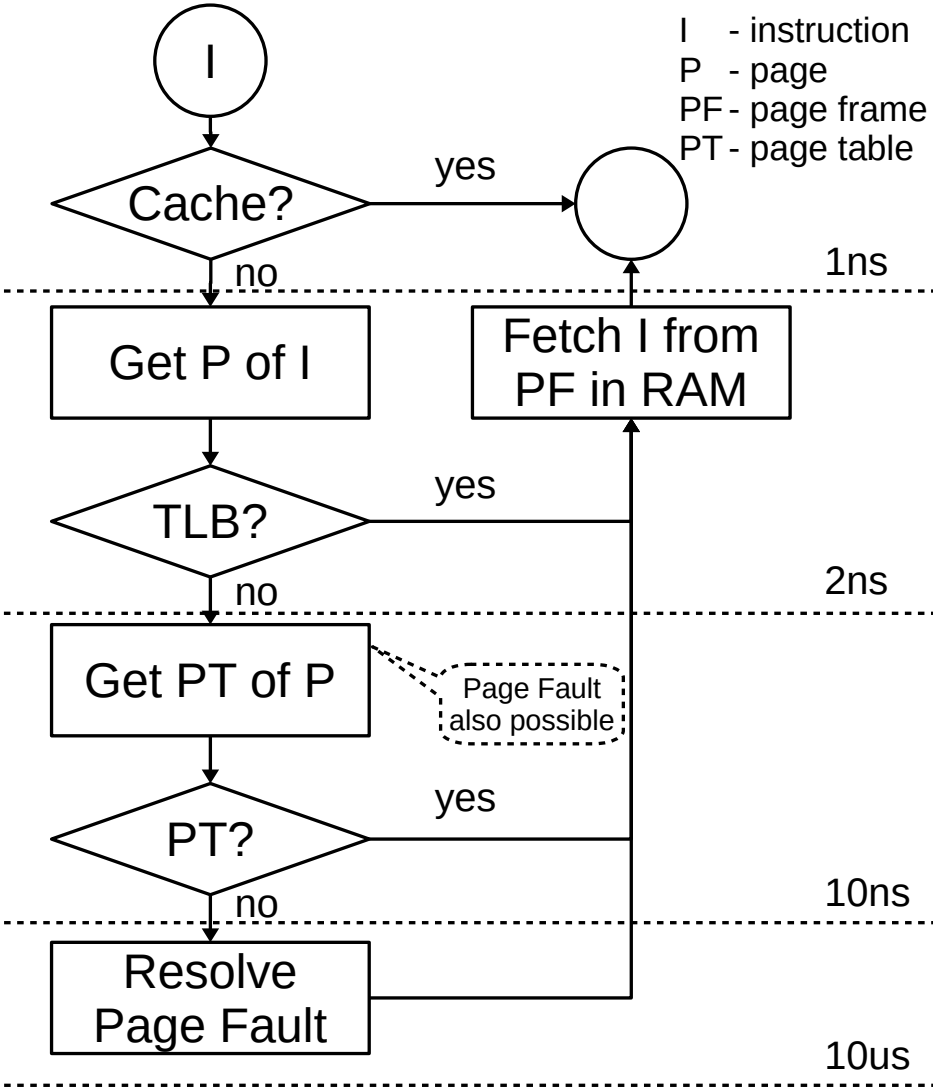
- Bežná knižnica reprezentujúca GUI dosahuje veľkosť 20-50 MB. Každá aplikácia používajúca túto knižnicu je menšia o 20-50 MB vďaka dynamickému linkovaniu. Šetrí sa miesto na disku a aj RAM.
- Akákoľvek zmena do knižnice. Napr. bug fix vyžaduje prekompilovanie kódu knižnice. Statické knižnice vyžadujú prekompilovanie všetkých programov. Pri dynamickom linkovaní toto nie je potrebné.
- Oprava chýb v knižniciach sa následne redukuje na výmenu súboru knižnice za nový pri aktualizácií OS.

Implementácia stránkovania

OS a stránkovanie

- OS je zainteresovaný do stránkovania ak:
  - proces vzniká, vykonáva sa alebo zaniká.
  - prebieha Page Fault.
  - prebieha pravidelné čistenie pamäte.
- Pri vzniku procesu OS:
  - zistí jeho veľkosť a vytvorí príslušnú Page Table,
  - vytvorí príslušný priestor v swap pamäti,
  - uloží ukazovatele na Page Table a Swap do PCB.
- Pri prepnutí kontextu OS:
  - vyprázdni/nahradí TLB,
  - nastaví registre segmentov a tabuliek stránok,
  - načíta vybrané stránky procesu. (stránky tabuľky stránok, WS)
- Po skončení procesu sa musí všetko vyčistiť.

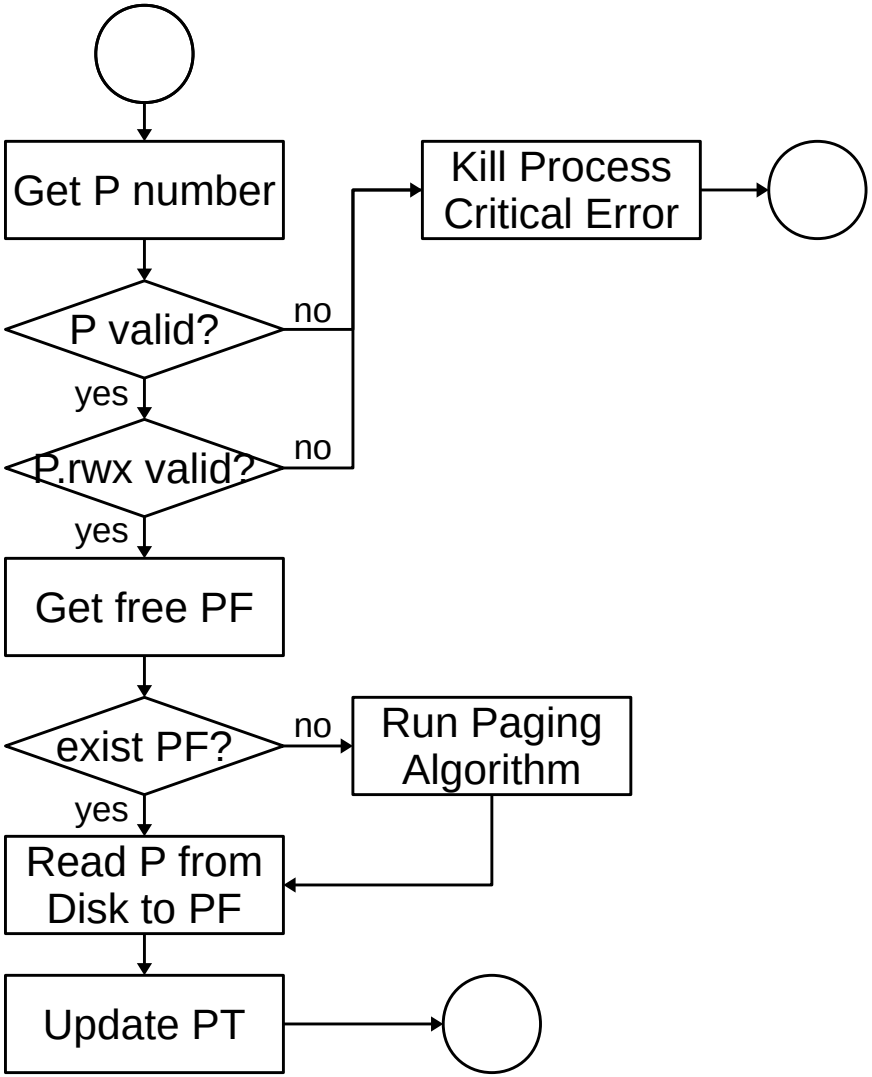
OS vykonanie inštrukcie



Page fault - 1. vznik

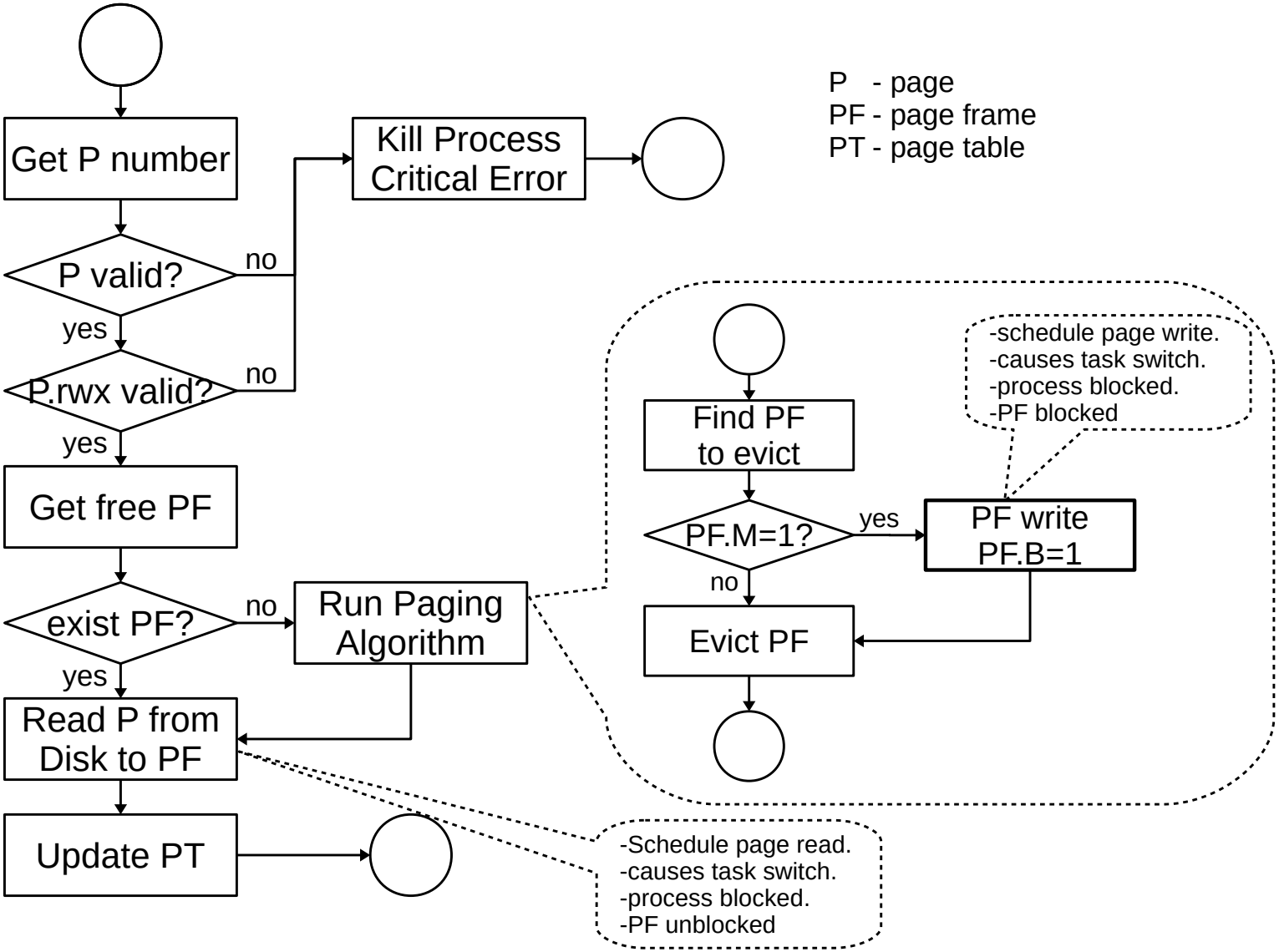
- HW odloží PC do stack-u.
- HW odloží stav vykonania inštrukcie.
- HW vyvolá TRAP = prerušenie.
- spustí sa pod-program OS spracujúci prerušenie
  - odloží kontext procesu (registre, stav).
  - identifikuje typ prerušenia (Page Fault).
  - vykoná Page Fault pod-program.

Page fault - 2. spracovanie



P - page
PF - page frame
PT - page table

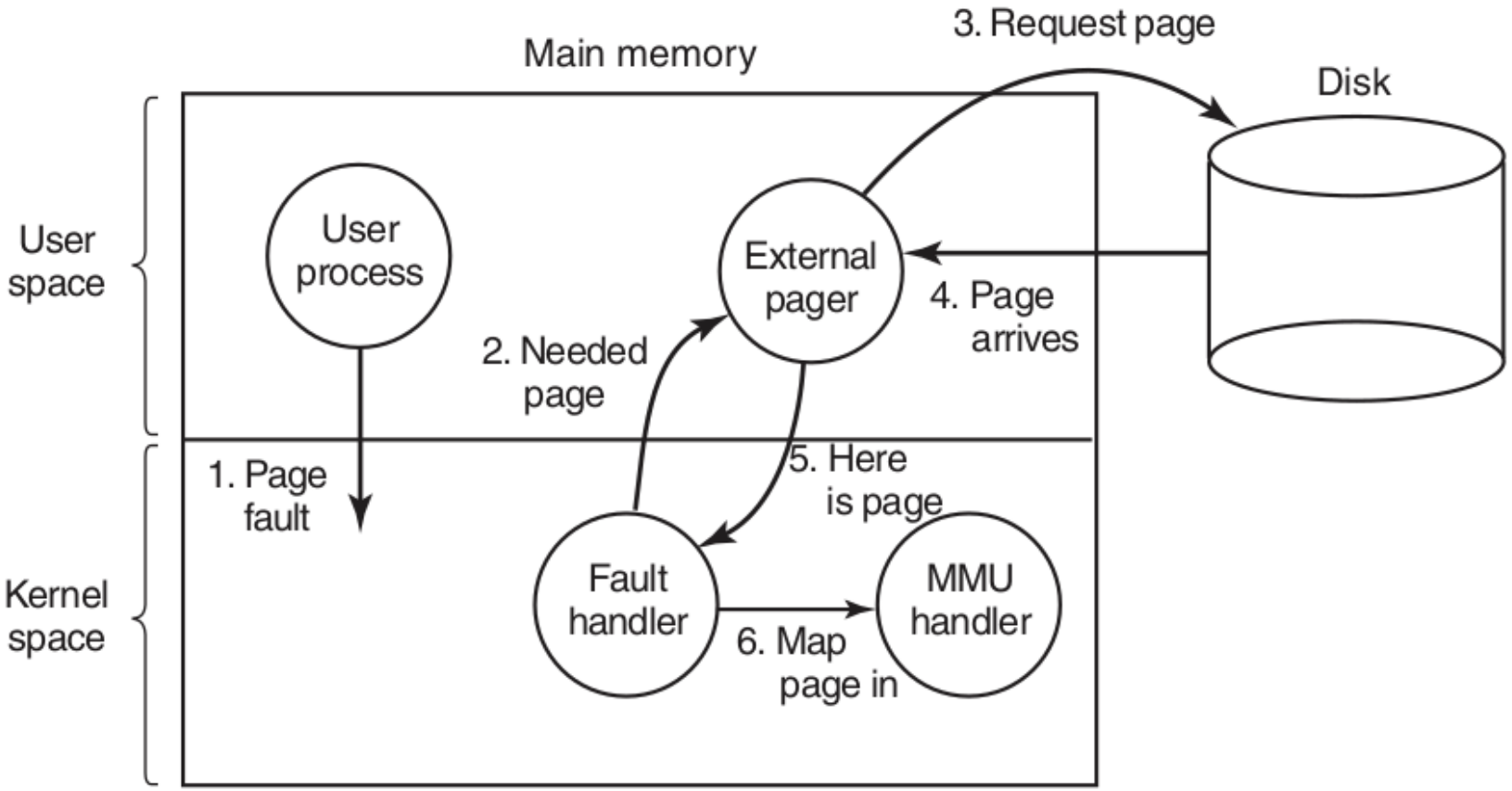
Page fault - 2. spracovanie



Page fault - 3. Ukončenie prerušenia

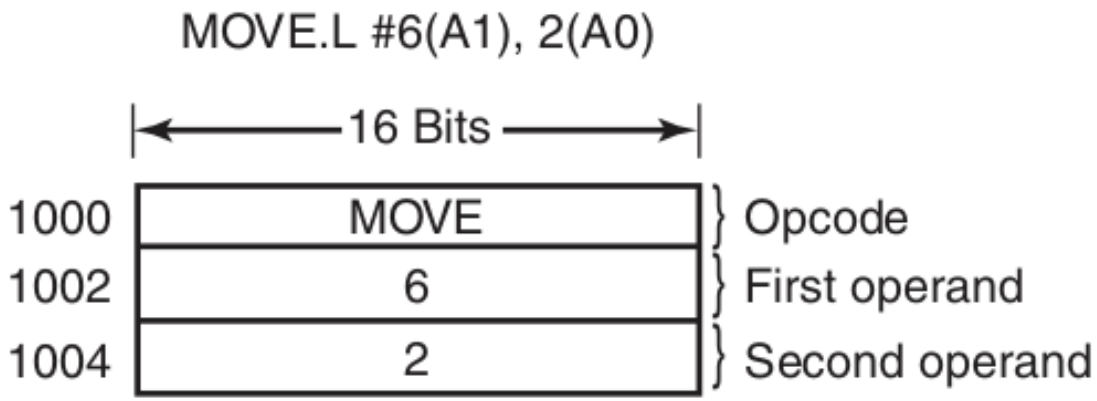
- pod-program načíta pôvodný kontext procesu.
- prečíta sa PC zo zásobníku.
- vystúpi sa z prerušenia a pokračuje sa vo vykonaní pôvodnej inštrukcie.

Page fault - Z hľadiska vrstiev OS



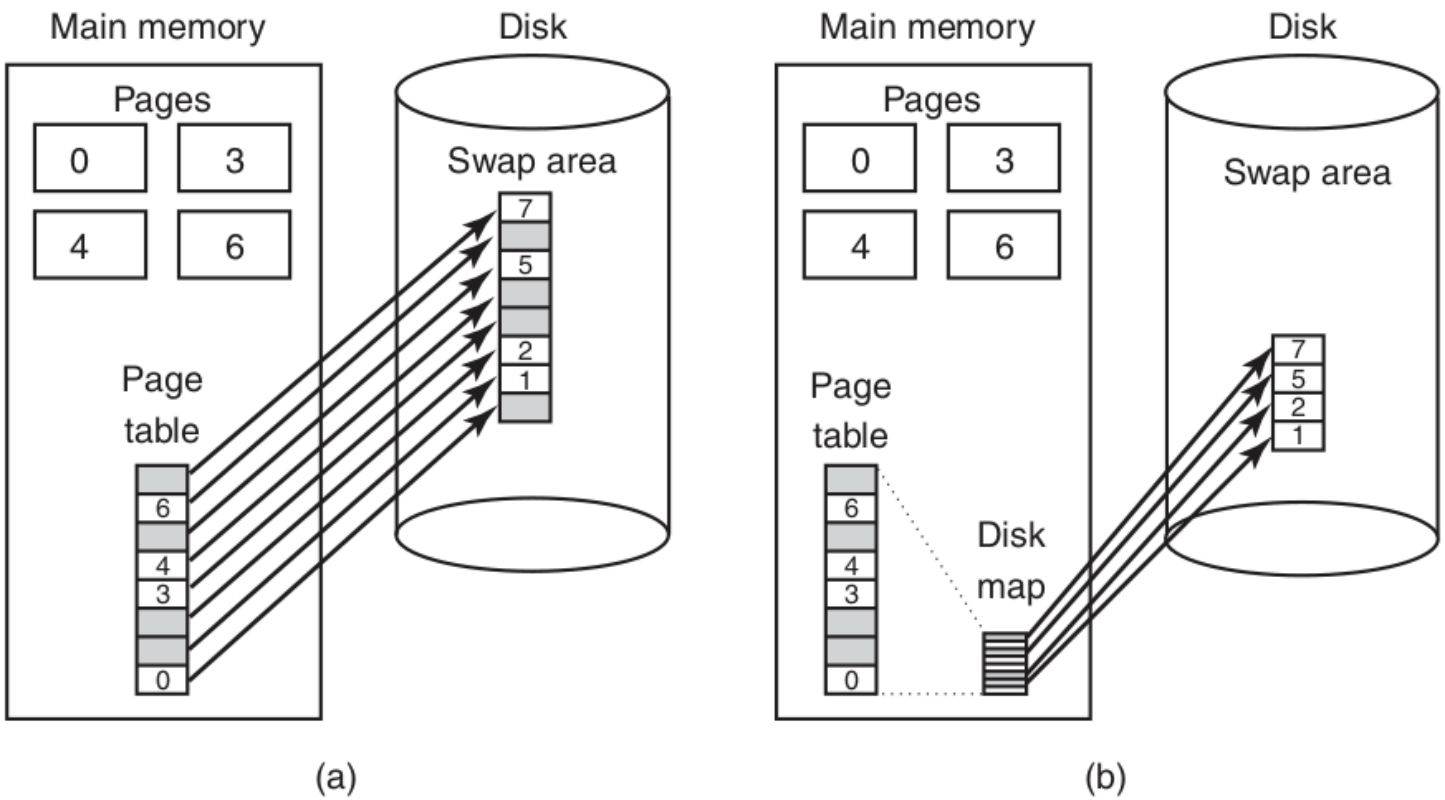
Záloha inštrukcie (CISC)

- Page Fault môže vzniknúť:
  - pri čítaní inštrukcie z pamäte.
  - pri čítaní operandov inštrukcie z pamäte.
- Ak nastane Page Fault pri čítaní operandov prerušenie nastane v strede vykonania inštrukcie.
- OS nemá šancu vyhodnotiť, že došlo k takejto situácií ak HW nedodá potrebné dáta.



Úložisko vyhodených stránok

- Page Table má obraz na disku (a)
- Dynamické úložisko na disku (b)



Algoritmy výmeny stránok

Optimálny algoritmus

- Algoritmus založený na predvídaní budúcnosti.[1]
- O každej stránke si udržiava informáciu kedy bude potrebná.
  - Napríklad počet inštrukcií kým bude referencovaná.
- V prípade page fault sa za obeť vyberie tá stránka, ktorá bude potrebná najneskôr.
- Optimálny algoritmus sa nedá implementovať so súčasným poznaním.
- Algoritmus vieme použiť ako referenčný algoritmus pri simulácií.

[1] Príklad uvedieme na cvičeniach.

Not Recently Used (NRU)

- Algoritmus využíva M-bit a R-bit pri výbere obete.
- Stránky sú roztriedené do tried:
  - Trieda 0: M = 0, R = 0
  - Trieda 1: M = 1, R = 0
  - Trieda 2: M = 0, R = 1
  - Trieda 3: M = 1, R = 1
- Obeť je náhodne vybraná z najnižšie očíslovanej, neprázdnej triedy.
- Je efektívnejšie vymeniť modifikovanú nereferencovanú ako nemodifikovanú referencovanú stránku.

First-in First-Out (FIFO)

- Algoritmus udržuje poradie v akom boli stránky načítané do pamäte.[2]
- V prípade Page Fault-u je obeťou tá najstaršia stránka.
- Problém: najstaršia $\neq$ nepoužívaná.

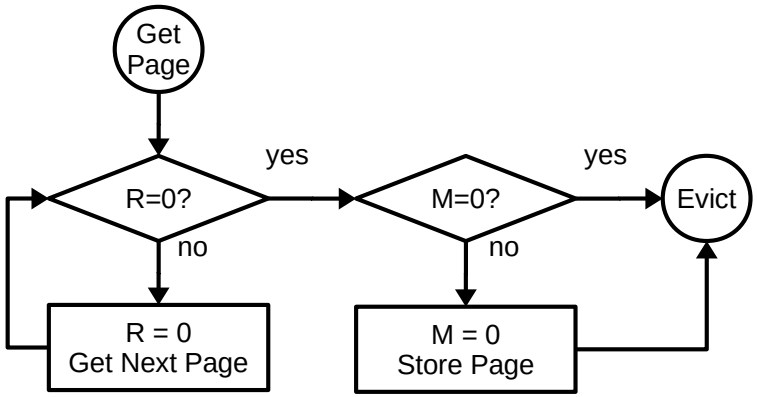
[2] Príklad uvedieme na cvičeniach.

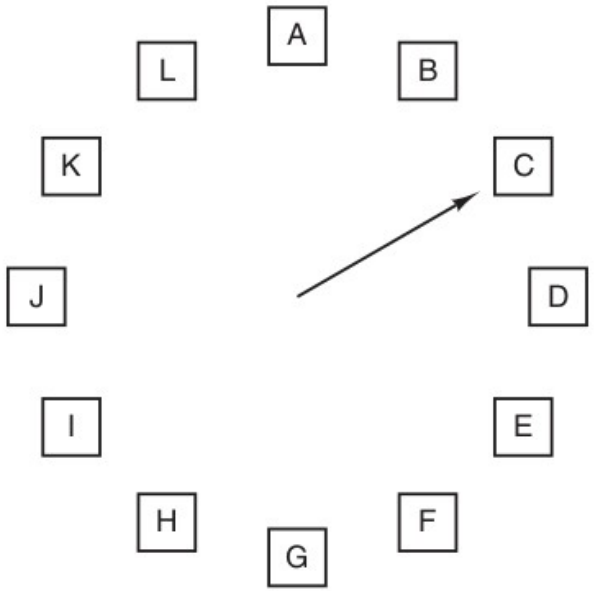
Second-Chance

- Upravený algoritmus FIFO, ktorý zohľadňuje R-bit pri výbere obete.
- Ak R = 0 stránka je stará a nepotrebná - Môže byť vylúčená.
- Ak R = 1 stránka sa posunie na koniec radu. R-bit sa resetuje.
- Algoritmus pokračuje druhou najstaršou stránkou.
- Ak všetky stránky majú R = 1 algoritmus degeneruje do FIFO.
- Presúvanie stránok zo začiatku radu na jeho koniec pridáva na komplexnosti algoritmu.

Clock

- Algoritmus udržuje stránky v cyklickom zozname.
- Algoritmus ukazuje na najstaršiu stránku v zozname.
- Pri výbere obete sa kontroluje R-bit.
- Ak R = 0 stránka je obeťou načíta sa nová. Algoritmus ukazuje na druhú najstaršiu.
- Ak R = 1 stránka sa používa, R-bit sa resetuje. Pokračujeme druhou najstaršou stránkou.





Least Recently Used (LRU)

- Blízka aproximácia optimálneho algoritmu.
- Často používané stránky budú pravdepodobne často používané aj naďalej.
- Najmenej používané stránky sú vyberané ako obete.
- LRU v najčistejšej forme udržuje zoznam stránok, ktorý sa neustále usporadúva podľa používanosti stránok.
- Zoznam sa musí aktualizovať pri akomkoľvek prístupe do pamäte.
- Usporadúvanie je vždy neefektívne.

Least Recently Used (LRU) - HW optimalizácia

- Usporiadanie pomocou SW je nákladné. HW to dokáže urýchliť.
- Počítadlo prístupov:
  - Majme počítadlo C, ktoré rastie s každou vykonanou inštrukciou.
  - Počítadlo C je špeciálnym registrom v MMU.
  - Akonáhle je stránka vymenená. Jej príslušný záznam v PT sa aktualizuje hodnotou C.
  - Obetou je stránka s najmenším uloženým C.
- Matica prístupov:
  - HW udržuje maticu nxn, kde n je počet rámcov hlavnej pamäte.
  - Každý prístup do stránky s prideleným rámom k vykoná:
    - set 1 do každej položky riadku k matice.
    - set 0 do každej položky stĺpca k matice.
  - Riadok matice s najmenším počtom 1 je LRU stránka.[3]

[3] Príklad uvedieme na cvičeniach.

Not Frequently Used (NFU)

- LRU HW sa nepoužíva v bežnom CPU. Je to drahé a neflexibilné.
- NFU je aproximácia LRU bez usporiadania zoznamu stránok.
- Každá stránka má počítadlo C, ktoré vyjadruje počet prístupov.
- Pri každom cykle, OS inkrementuje C tých stránok, ktorých R = 1.
- Stránka s najmenším C je obeť.

```
1  void onClockCycle()
2  {
3      foreach page in presentPages do {
4          if (page.R == 1) {
5              page.C++;
6              page.R = 0;
7          }
8          minC = min(minC, page.C);
9      }
10 }
```

Not Frequently Used (NFU)

- Problém s dlhodobou pamäťou. V minulosti používané stránky, ktoré sa už nepoužívajú trčia v systéme.

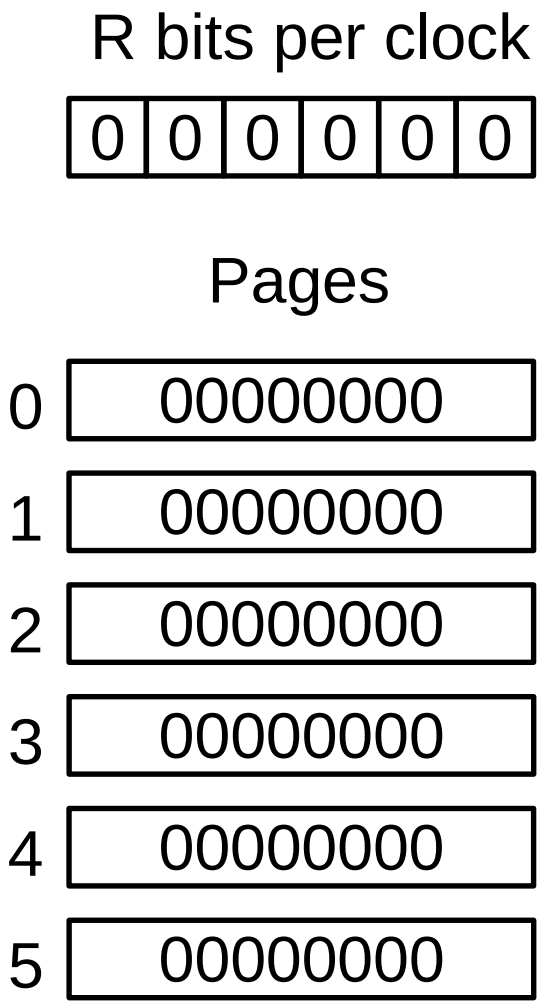
```
1 //Number of Frames = 3;
2 void main() { //Page 1
3     for (int i = 0; i < 10; i++)
4         doWork1(i); //Page 2
5
6     //C1=d10, C2=d10
7     for (int i = 0; i < 10; i++) {
8         doWork2(i); //Page 3; i=0: C1=d11, C2=d10, C3=d1
9         doWork3(i); //Page 4; i=0: C1=d12, C2=d10, C4=d1, C3 evict
10    }
11 }
```

Not Frequently Used with Aging (NFUA)

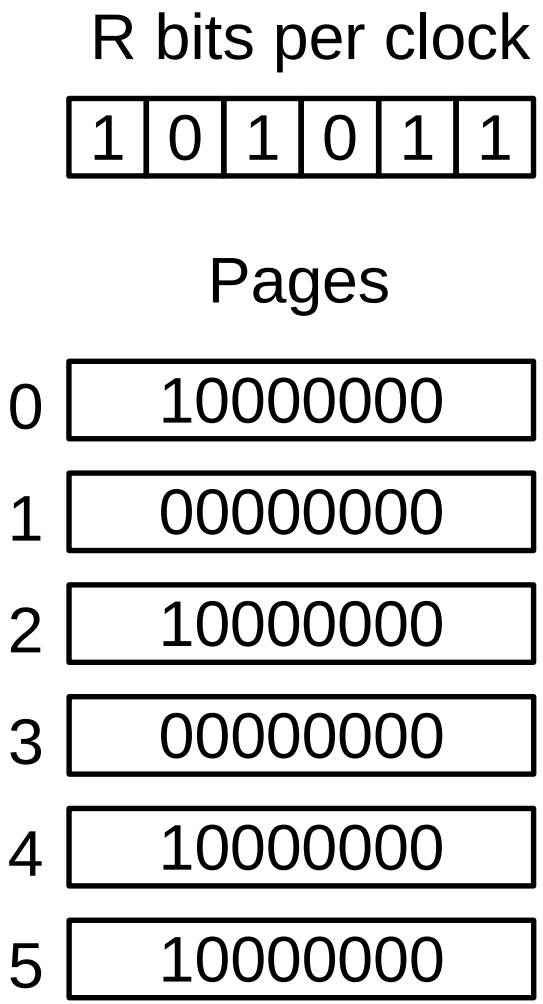
- Starnutie - každý prístup ku stránke znamená:
  - bitový posun C doprava o 1 bit (C » 1)
  - pripočítanie 1 na najvýznamnejší bit ($C|2^{32}$).
- Stránky, ktoré sa nepoužívajú s každým cyklom stratia vplyv.
- Stránky použité naposledy majú najväčší vplyv.

```
1  #define MSB 0x80000000 //if C has 32 bits
2  void onClockCycle()
3  {
4      foreach page in presentPages do {
5          page.C = page.C >> 1;
6          if (page.R == 1) {
7              page.C |= MSB
8              page.R = 0;
9          }
10         minC = min(minC, page.C);
11     }
12 }
```

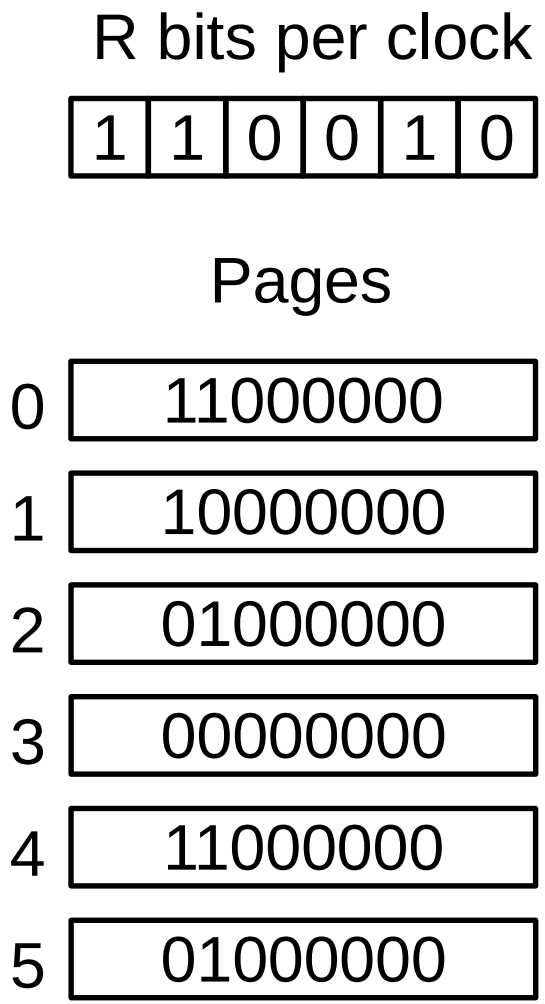
Not Frequently Used with Aging (NFUA)



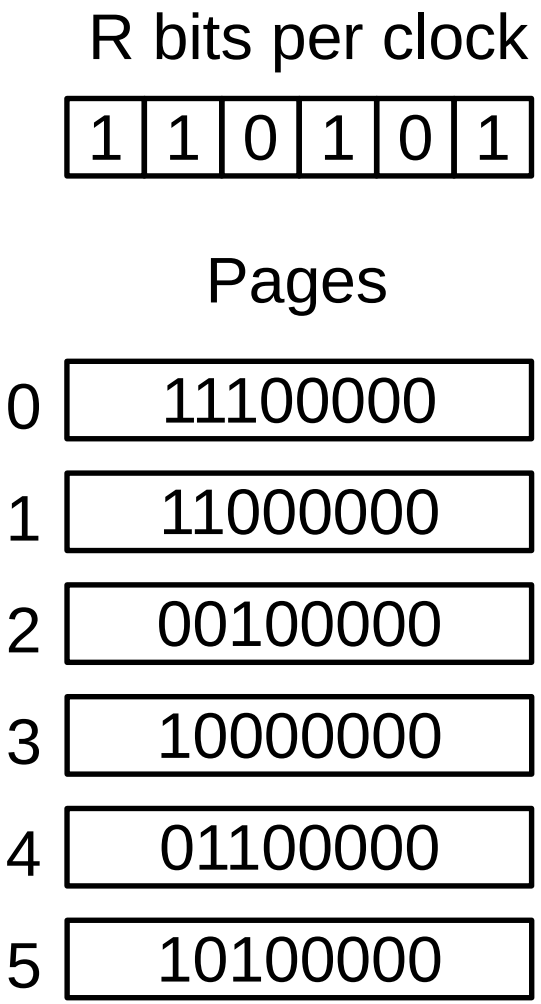
Not Frequently Used with Aging (NFUA)



Not Frequently Used with Aging (NFUA)



Not Frequently Used with Aging (NFUA)



Not Frequently Used with Aging (NFUA)

R bits per clock

1	0	0	0	1	0

Pages

0	11110000
1	01100000
2	00010000
3	01000000
4	10110000
5	01010000

Not Frequently Used with Aging (NFUA)

R bits per clock

0	1	1	0	0	0

Pages

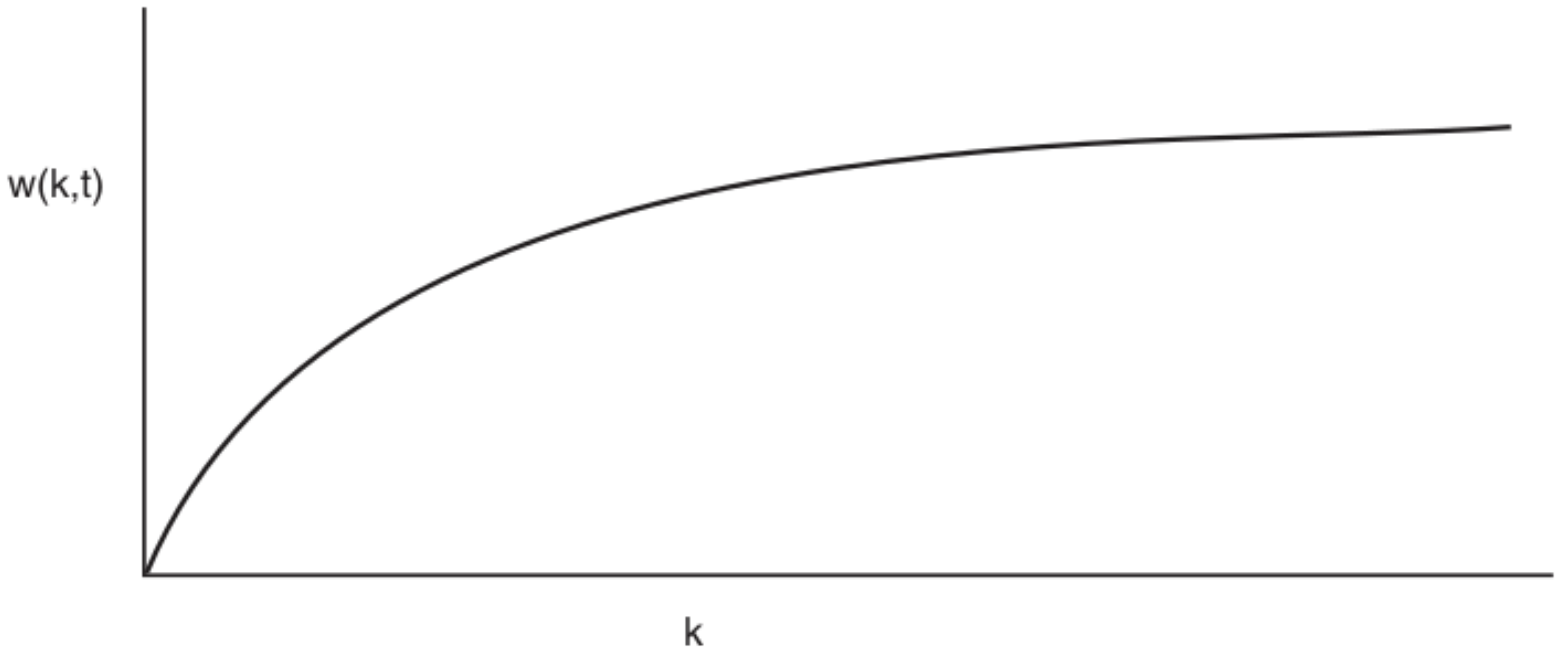
0	01111000
1	10110000
2	10001000
3	00100000
4	01011000
5	00101000

Not Frequently Used with Aging (NFUA)

```
1  //Number of Frames = 3;
2  void main() { //Page 1
3      for (int i = 0; i < 10; i++)
4          doWork1(i); //Page 2 C1=b01, C2=b10
5
6      //C1=b10, C2=b01
7      for (int i = 0; i < 10; i++) {
8          doWork2(i); //Page 3; i=0: C1=b01, C2=b00, C3=b10
9          doWork3(i); //Page 4; i=0: C1=b01, C3=b01, C4=b10, C2 evict
10     }
11 }
```

Working Set (WS)

- Working Set = Aktuálne najpoužívanejšie stránky
- WS môžeme vyjadriť ako funkciu $ws(k,t)$, kde k je počet nedávno použitých stránok v čase t.
- k je počet sledovaných stránok od času t do minulosti.
- $ws(k,t)$ je monotonická neklesajúca funkcia k.



Working Set (WS)

```
1 void main() { //Page 1
2     doInit(); //Page 2
3     while(true) {
4         doWork1(); //Page 3
5         doWork2(); //Page 4
6     }
7 }
8 // clock cycles: 1  2  3  4  5  6  7  8  9  10 11...
9 //Page requests: 1  2  1  3  1  4  1  3  1  4  1 ...
```

- $ws(k,t)_{k=1}$: **1;** 2; x[1; 3; 1; 4; 1; 3; 1; 4; 1; ...]
- $ws(k,t)_{k=2}$: **1;** 1,2; x[1,3; 1,4; 1,3; 1,4; ...]
- $ws(k,t)_{k=3}$: **1;** 1,2; 1,2,3; x[1,3; 1,3,4; 1,4; 1,3,4; ...]
- $ws(k,t)_{k=4}$: **1;** 1,2; 1,2,3; x[1,3,4; ...]
- $ws(k,t)_{k=5}$: **1;** 1,2; 1,2,3; 1,2,3,4; x[1,3,4; ...]

Working Set (WS)

```
1 void main() { //Page 1
2     doInit(); //Page 2
3     while(true) {
4         doWork1(); //Page 3
5         doWork2(); //Page 4
6     }
7 }
8 // clock cycles: 1  2  3  4  5  6  7  8  9  10 11...
9 //Page requests: 1  2  1  3  1  4  1  3  1  4  1 ...
```

- $ws(k,t)_{k=1}$: 1; **2;** x[1; 3; 1; 4; 1; 3; 1; 4; 1; ...]
- $ws(k,t)_{k=2}$: 1; **1,2;** x[1,3; 1,4; 1,3; 1,4; ...]
- $ws(k,t)_{k=3}$: 1; **1,2;** 1,2,3; x[1,3; 1,3,4; 1,4; 1,3,4; ...]
- $ws(k,t)_{k=4}$: 1; **1,2;** 1,2,3; x[1,3,4; ...]
- $ws(k,t)_{k=5}$: 1; **1,2;** 1,2,3; 1,2,3,4; x[1,3,4; ...]

Working Set (WS)

```
1 void main() { //Page 1
2     doInit(); //Page 2
3     while(true) {
4         doWork1(); //Page 3
5         doWork2(); //Page 4
6     }
7 }
8 // clock cycles: 1  2  3  4  5  6  7  8  9  10 11...
9 //Page requests: 1  2  1  3  1  4  1  3  1  4  1 ...
```

- $ws(k,t)_{k=1}$: 1; 2; x[**1;** 3; 1; 4; 1; 3; 1; 4; 1; ...]
- $ws(k,t)_{k=2}$: 1; 1,2; x[**1,3;** 1,4; 1,3; 1,4; ...]
- $ws(k,t)_{k=3}$: 1; 1,2; **1,2,3;** x[1,3; 1,3,4; 1,4; 1,3,4; ...]
- $ws(k,t)_{k=4}$: 1; 1,2; **1,2,3;** x[1,3,4; ...]
- $ws(k,t)_{k=5}$: 1; 1,2; **1,2,3;** 1,2,3,4; x[1,3,4; ...]

Working Set (WS)

```
1 void main() { //Page 1
2     doInit(); //Page 2
3     while(true) {
4         doWork1(); //Page 3
5         doWork2(); //Page 4
6     }
7 }
8 // clock cycles: 1  2  3  4  5  6  7  8  9  10 11...
9 //Page requests: 1  2  1  3  1  4  1  3  1  4  1 ...
```

- $ws(k,t)_{k=1}$: 1; 2; x[1; **3;** 1; 4; 1; 3; 1; 4; 1; ...]
- $ws(k,t)_{k=2}$: 1; 1,2; x[1,3; **1,4;** 1,3; 1,4; ...]
- $ws(k,t)_{k=3}$: 1; 1,2; 1,2,3; x[**1,3;** 1,3,4; 1,4; 1,3,4; ...]
- $ws(k,t)_{k=4}$: 1; 1,2; 1,2,3; x[**1,3,4;** ...]
- $ws(k,t)_{k=5}$: 1; 1,2; 1,2,3; **1,2,3,4;** x[1,3,4; ...]

Working Set (WS)

```
1 void main() { //Page 1
2     doInit(); //Page 2
3     while(true) {
4         doWork1(); //Page 3
5         doWork2(); //Page 4
6     }
7 }
8 // clock cycles: 1  2  3  4  5  6  7  8  9  10 11...
9 //Page requests: 1  2  1  3  1  4  1  3  1  4  1 ...
```

- $ws(k,t)_{k=1}$: 1; 2; x[1; 3; **1;** 4; 1; 3; 1; 4; 1; ...]
- $ws(k,t)_{k=2}$: 1; 1,2; x[1,3; 1,4; **1,3;** 1,4; ...]
- $ws(k,t)_{k=3}$: 1; 1,2; 1,2,3; x[1,3; **1,3,4;** 1,4; 1,3,4; ...]
- $ws(k,t)_{k=4}$: 1; 1,2; 1,2,3; x[**1,3,4;** ...]
- $ws(k,t)_{k=5}$: 1; 1,2; 1,2,3; 1,2,3,4; x[**1,3,4;** ...]

Working Set (WS)

```
1 void main() { //Page 1
2     doInit(); //Page 2
3     while(true) {
4         doWork1(); //Page 3
5         doWork2(); //Page 4
6     }
7 }
8 // clock cycles: 1  2  3  4  5  6  7  8  9  10 11...
9 //Page requests: 1  2  1  3  1  4  1  3  1  4  1 ...
```

- $ws(k,t)_{k=1}$: 1; 2; x[1; 3; 1; **4;** 1; 3; 1; 4; 1; ...]
- $ws(k,t)_{k=2}$: 1; 1,2; x[1,3; 1,4; 1,3; **1,4;** ...]
- $ws(k,t)_{k=3}$: 1; 1,2; 1,2,3; x[1,3; 1,3,4; **1,4;** 1,3,4; ...]
- $ws(k,t)_{k=4}$: 1; 1,2; 1,2,3; x[**1,3,4;** ...]
- $ws(k,t)_{k=5}$: 1; 1,2; 1,2,3; 1,2,3,4; x[**1,3,4;** ...]

Working Set (WS)

```
1 void main() { //Page 1
2     doInit(); //Page 2
3     while(true) {
4         doWork1(); //Page 3
5         doWork2(); //Page 4
6     }
7 }
8 // clock cycles: 1  2  3  4  5  6  7  8  9  10 11...
9 //Page requests: 1  2  1  3  1  4  1  3  1  4  1 ...
```

- $ws(k,t)_{k=1}$: 1; 2; x[1; 3; 1; 4; **1;** 3; 1; 4; 1; ...]
- $ws(k,t)_{k=2}$: 1; 1,2; x[**1,3;** 1,4; 1,3; 1,4; ...]
- $ws(k,t)_{k=3}$: 1; 1,2; 1,2,3; x[1,3; 1,3,4; 1,4; **1,3,4;** ...]
- $ws(k,t)_{k=4}$: 1; 1,2; 1,2,3; x[**1,3,4;** ...]
- $ws(k,t)_{k=5}$: 1; 1,2; 1,2,3; 1,2,3,4; x[**1,3,4;** ...]

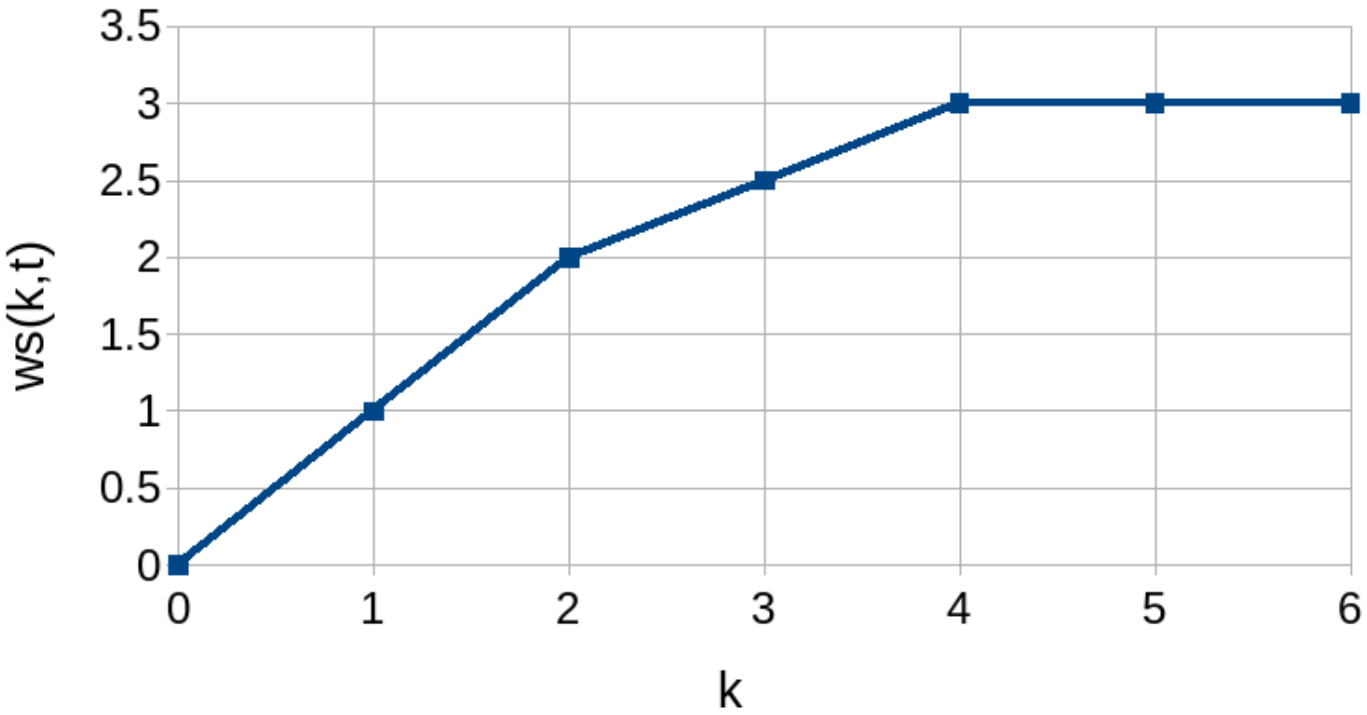
Working Set (WS)

```
1 void main() { //Page 1
2     doInit(); //Page 2
3     while(true) {
4         doWork1(); //Page 3
5         doWork2(); //Page 4
6     }
7 }
8 // clock cycles: 1  2  3  4  5  6  7  8  9  10 11...
9 //Page requests: 1  2  1  3  1  4  1  3  1  4  1 ...
```

- $ws(k,t)_{k=1}$: 1; 2; x[1; 3; 1; 4; 1; **3;** 1; 4; 1; ...]
- $ws(k,t)_{k=2}$: 1; 1,2; x[1,3; **1,4;** 1,3; 1,4; ...]
- $ws(k,t)_{k=3}$: 1; 1,2; 1,2,3; x[**1,3;** 1,3,4; 1,4; 1,3,4; ...]
- $ws(k,t)_{k=4}$: 1; 1,2; 1,2,3; x[**1,3,4;** ...]
- $ws(k,t)_{k=5}$: 1; 1,2; 1,2,3; 1,2,3,4; x[**1,3,4;** ...]

Working Set (WS)

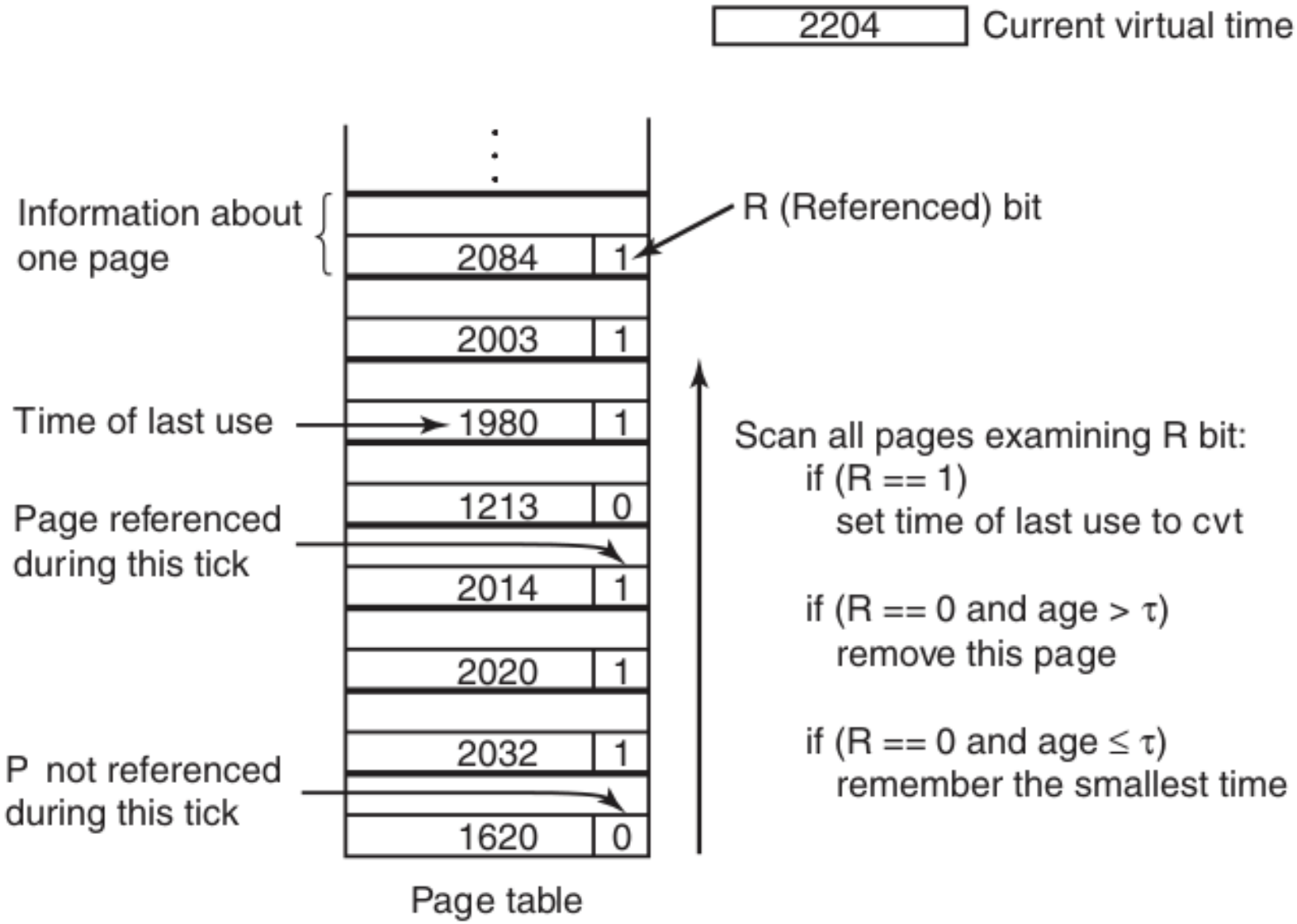
- $ws(k,t)_{k=1}$: 1; 2; x[1; 3; 1; 4; 1; 3; 1; 4; 1; ...]
- $ws(k,t)_{k=2}$: 1; 1,2; x[1,3; 1,4; 1,3; 1,4; ...]
- $ws(k,t)_{k=3}$: 1; 1,2; 1,2,3; x[1,3; 1,3,4; 1,4; 1,3,4; ...]
- $ws(k,t)_{k=4}$: 1; 1,2; 1,2,3; x[1,3,4; ...]
- $ws(k,t)_{k=5}$: 1; 1,2; 1,2,3; 1,2,3,4; x[1,3,4; ...]



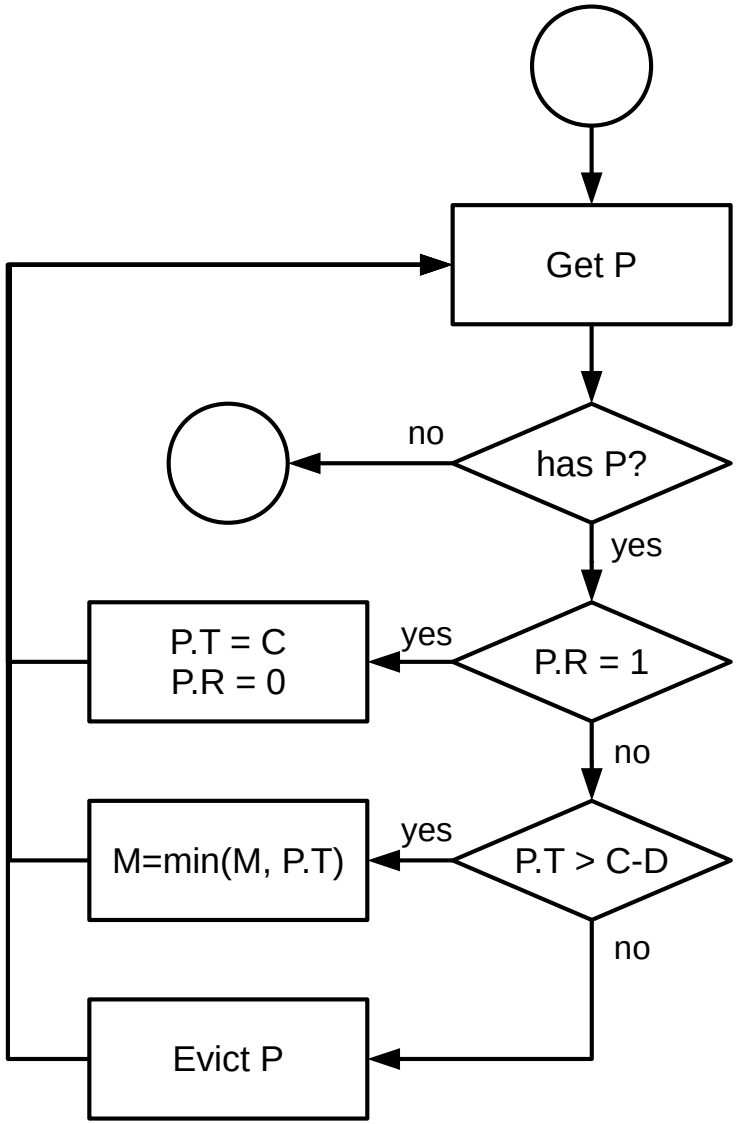
Working Set (WS) - realizácia

- Uložiť do radu k referencovaných stránok.
  - Na začiatok radu pridávam nové stránky. Na konci radu ich odoberám.
  - Po zanedbaní duplicít dostaneme aktuálny WS procesu.
  - Pridelené stránky procesu, ktoré nie sú v rade môžu byť vylúčené.
  - Prechádzanie takéhoto radu a zanedbávanie duplicít je nepraktické.
- Použiť čas vykonávania procesu.
  - Stránka je súčasťou WS procesu ak bola referencovaná v určenom časovom intervale. (napr. 100ms)
  - HW nastavuje R-bit. OS po každom prerušení z hodín R-bit čistí.
  - Algoritmus prechádza všetky stránkové rámy procesu po každom Page Fault.

Working Set (WS) - realizácia



Working Set (WS) - realizácia

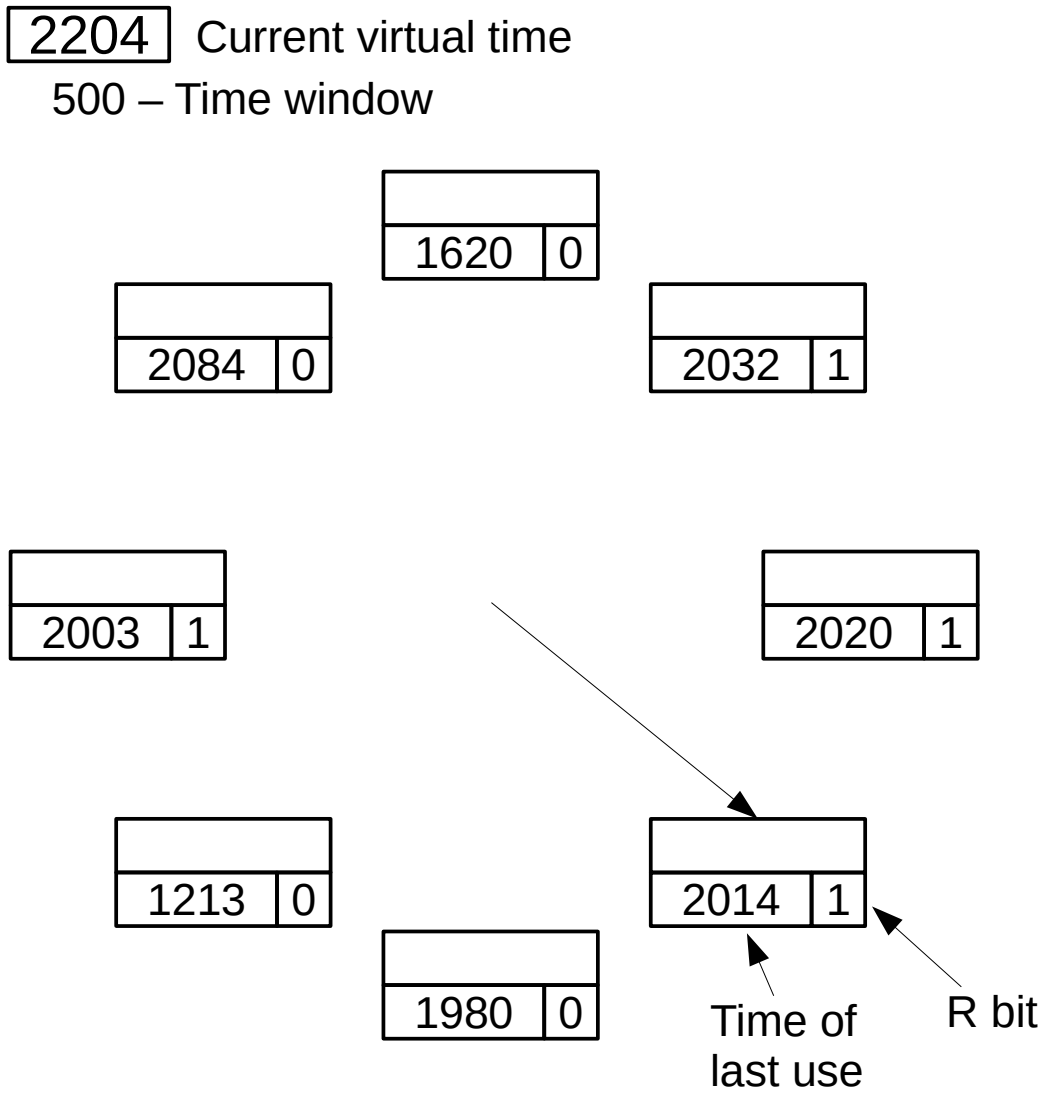


P - page
C - current time
D - delta time
M - page with minimum time

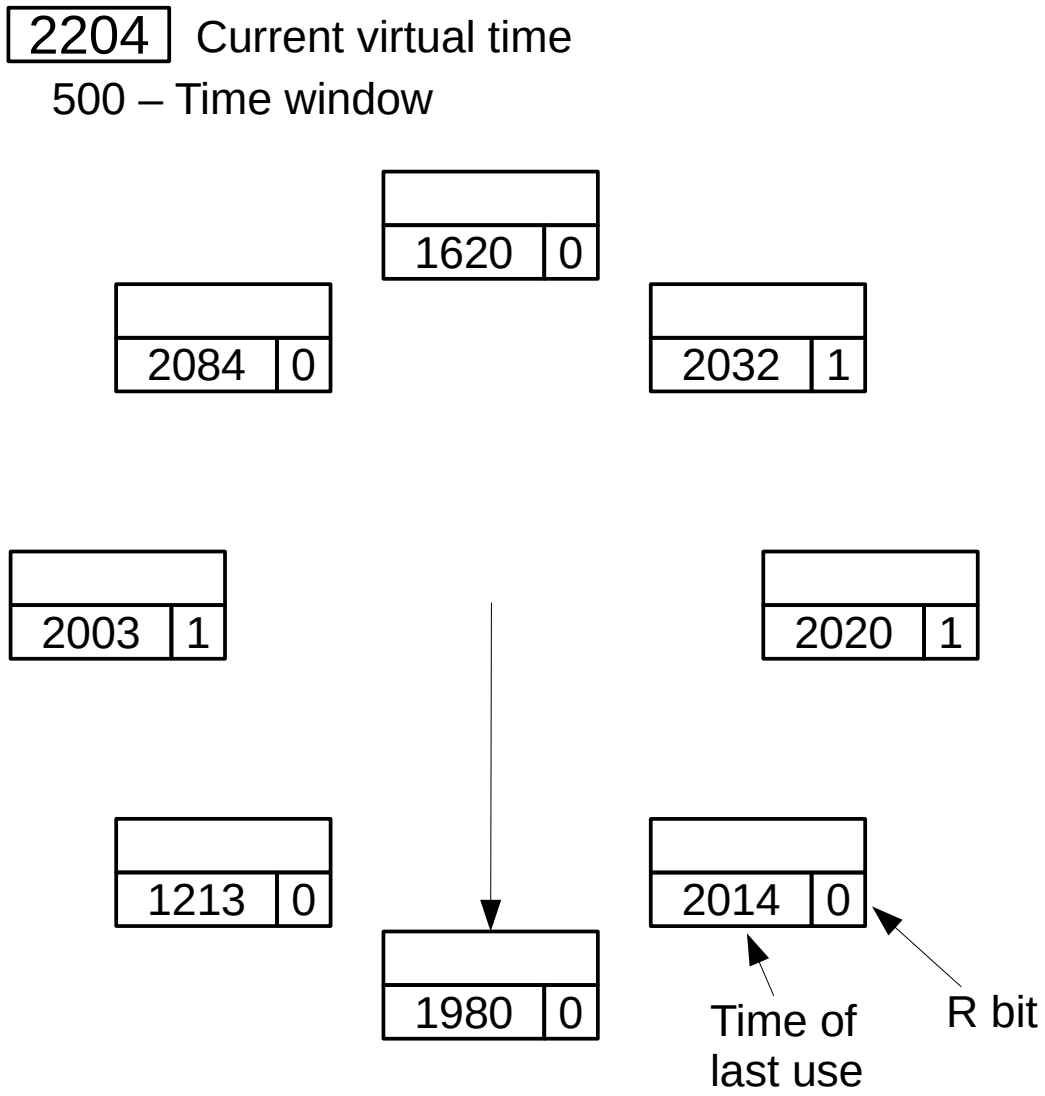
Working Set Clock (WSC)

- Prechádzať všetky stránkové rami je neefektívne skúsme zapojiť Clock algoritmus.
- Podobne ako pri Clock algoritme udržiavame stránky patriace do WS v cyklickom zozname.
- V prípade Page Fault vyberáme obeť, ktorá nebola referencovaná a je staršia ako stanovený interval.
- Algoritmus neaktualizuje všetky referencované záznamy.

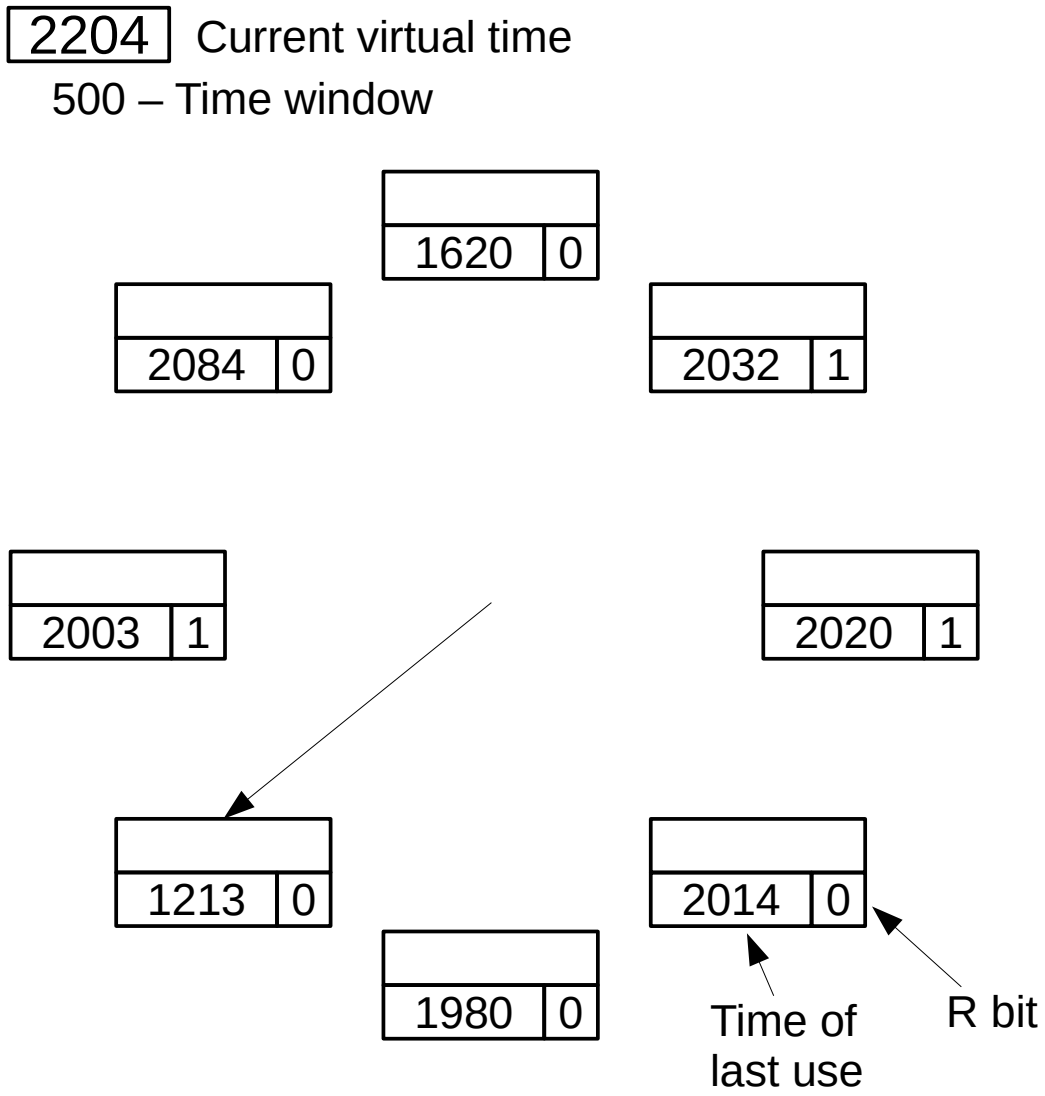
Working Set Clock (WSC)



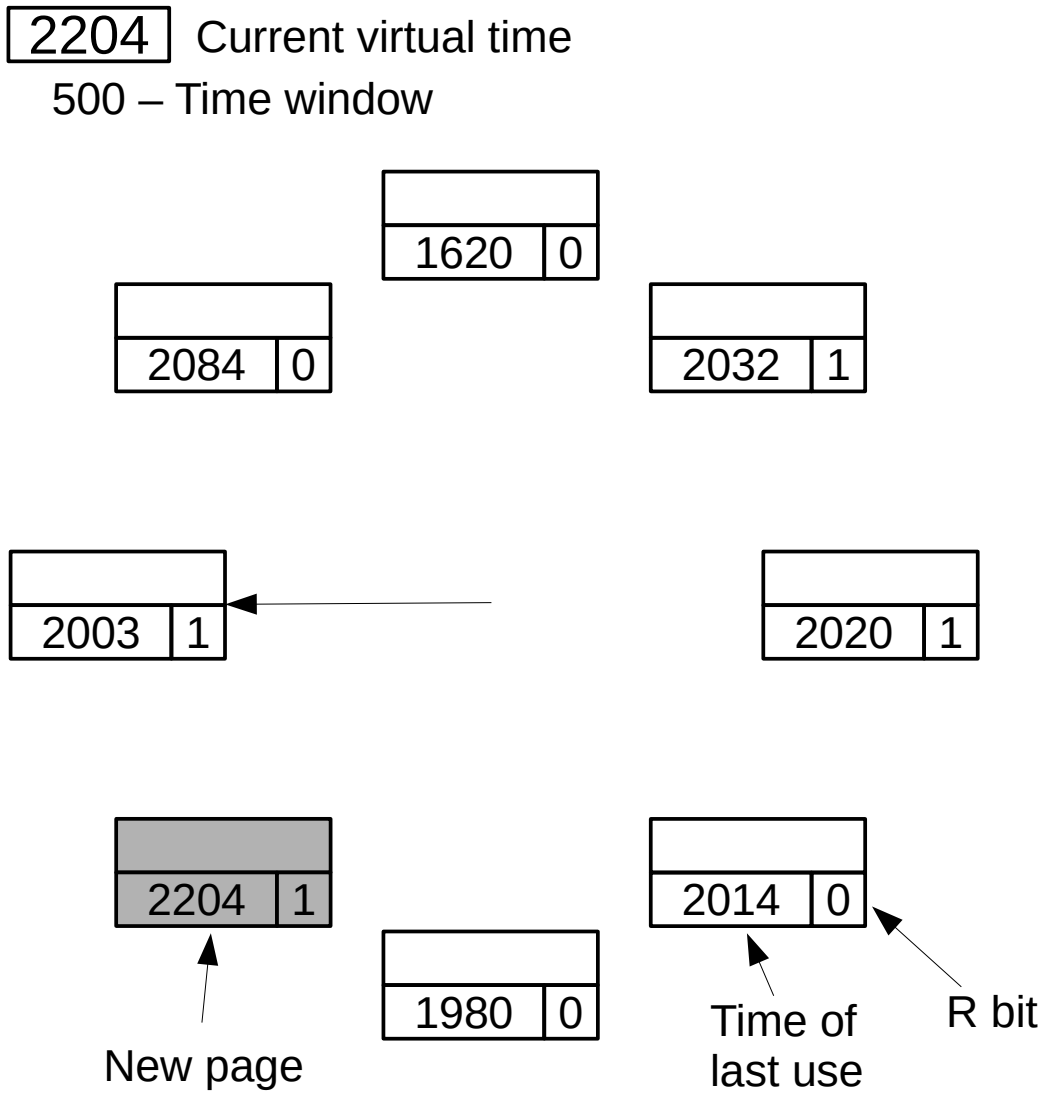
Working Set Clock (WSC)



Working Set Clock (WSC)



Working Set Clock (WSC)



Algotritmy výmeny stránok

- Optimálny Algoritmus - Nedá sa implementovať, dobrý benchmark
- Not Recently Used (NRU) - Zhruba aproximuje LRU
- First-in First-Out (FIFO) - Môže vyhodiť dôležité stránky
- Second-Chance - Veľké vylepšenie FIFO
- **Clock** - Použiteľný a jednoduchý algoritmus
- Least Recently Used (LRU) - Drahý na implementáciu
  - Not Frequently Used (NFU) - zhruba aproximuje LRU
  - **NFU with Aging (NFUA)** - dobre aproximuje LRU
- Working Set (WS) - Drahý na implementáciu.
- **Working Set Clock (WSC)** - Efektívny algoritmus.

Zhrnutie

Zhrnutie

- Návrh stránkovania musí zohľadniť špecifiká výpočtového systému na základe ktorých je nutné zvážiť:
  - Výber alokačnej politiky stránok a politiky čistenia.
  - Spôsob čítania stránok do pamäte
  - Výber veľkosti stránky.
  - Zohľadniť zdieľanie stránok.
  - Výber algoritmu výmeny stránok.
- Algoritmus výberu stránok je veľa ale najpoužívanejšie sú Clock, NFU with aging alebo WS Clock. Prípadne ich ekvivalenty.
- Paging Daemon - čistenie pamäte aby sme nemuseli hľadať obete.
- Page Fault - najhorší scenár pri vykonávaní procesu.
- SWAP - odkladací priestor pre stránky, ktoré sa stali obeťami.

Čo robiť do ďalšej prednášky

- Prečítať kapitolu 4. z Tanenbauma.