

OS MMXXI

MIT ;)

<https://pdos.csail.mit.edu/6.828>

Virtuálna pamäť

Štruktúra prednášky

- Adresné priestory
- Stránkovací hardvér
- Programový kód xv6 pre správu VM

Téma

- Majme program, ktorý z času na čas zapíše niečo na náhodnú adresu v pamäti
- Ako udržať takýto program “na uzde”?

Téma

- Majme program, ktorý z času na čas zapíše niečo na náhodnú adresu v pamäti
- Ako udržať takýto program “na uzde”?
- Izolujme adresné priestory procesov (a jadra)

Izolácia adresného priestoru

Izolácia adresného priestoru

- Každý proces má vlastný adresný priestor
- Môže čítať/zapisovať iba v rámci svojho priestoru (nemôže čítať ani zapisovať v inom)

Izolácia adresného priestoru

- Každý proces má vlastný adresný priestor
- Môže čítať/zapisovať iba v rámci svojho priestoru (nemôže čítať ani zapisovať v inom)
- **Ako SÚČASNE implementovať viac adresných priestorov v rámci jedinej fyzickej pamäte (RAM) a zabezpečiť izoláciu medzi nimi?**

Izolácia adresného priestoru

- Odpoveď: stránkovací hardvér
- Xv6 využíva hardvér procesora RISC-V

Stránkovanie

Stránkovanie

- Poskytuje nepriamu adresáciu

Stránkovanie

- Poskytuje nepriamu adresáciu
 - program používa ukazovatele (adresy); tieto adresy využíva CPU na prístup k pamäti

Stránkovanie

- Poskytuje nepriamu adresáciu
 - program používa ukazovatele (adresy); tieto adresy využíva CPU na prístup k pamäti
 - Nie sú to však (vo všeobecnosti) adresy operandov vo fyzickej pamäti!
 - Ide o tzv. virtuálne adresy!

Stránkovanie

- Poskytuje nepriamu adresáciu
 - program používa ukazovatele (adresy); tieto adresy využíva CPU na prístup k pamäti
 - Nie sú to však (vo všeobecnosti) adresy operandov vo fyzickej pamäti!
 - Ide o tzv. virtuálne adresy!
 - Hardvér správy pamäte (MMU) “prekladá” virtuálne adresy na fyzické adresy

Stránkovanie

- Poskytuje nepriamu adresáciu
 - program používa ukazovatele (adresy); tieto adresy využíva CPU na prístup k pamäti
 - Nie sú to však (vo všeobecnosti) adresy operandov vo fyzickej pamäti!
 - Ide o tzv. virtuálne adresy!
 - Hardvér správy pamäte (MMU) “prekladá” virtuálne adresy na fyzické adresy
 - V literatúre sa často spomína aj termín “lineárna adresa” (v rámci predmetu nerozlišujeme)
 - Až fyzická adresa je adresou operandu na adresnej zbernici (nemusí to byť iba RAM!!!)

Stránkovanie

- Program dokáže pracovať iba s virtuálnymi adresami, nie fyzickými (čo sa týka menenia/čítania obsahu RAM)
- Jadro OS má za úlohu riadiť mapovanie každej VA na PA (nastaviť údaje pre toto mapovanie)

Stránkovanie

- MMU používa na “preklad” tabuľku, v ktorej
 - VA je indexom (zjednodušené)
 - PA je hodnotou na indexe (zjednodušené)

Stránkovanie

- MMU používa na “preklad” tabuľku, v ktorej
 - VA je indexom (zjednodušené)
 - PA je hodnotou na indexe (zjednodušené)
- Tabuľka má názov “Tabuľka stránok” (angl. *Page Table*)

Stránkovanie

- MMU používa na “preklad” tabuľku, v ktorej
 - VA je indexom (zjednodušené)
 - PA je hodnotou na indexe (zjednodušené)
- Tabuľka má názov “Tabuľka stránok” (angl. *Page Table*)
- Jedna tabuľka popisuje jeden adresný priestor (mapovanie VA na PA)
- MMU dokáže obmedziť, ktoré VA sú prístupné v *user* móde CPU

Stránkovanie

- Odkaz na aktuálnu tabuľku stránok (tabuľku, ktorá sa pri preklade používa), sa nachádza v špeciálnom registri CPU

Stránkovanie

- Odkaz na aktuálnu tabuľku stránok (tabuľku, ktorá sa pri preklade používa), sa nachádza v špeciálnom registri CPU
- Meniť hodnotu tohto registra je možné iba v privilegovanom režime CPU (*kernel* mód)
 - Pokus o zmenu v *user* móde vedie ku výnimke

Stránkovanie

- CPU štartuje v režime, kedy je stránkovanie vypnuté
 - Prečo?

Stránkovanie

- CPU štartuje v režime, kedy je stránkovanie vypnuté
 - Prečo?
- Inicializačný kód jadra
 - Vyplní tabuľku
 - Nastaví register, ktorý ukazuje na tabuľku stránok
 - Zapne stránkovanie CPU

Stránkovanie

- Stránkovanie umožňuje

Stránkovanie

- Stránkovanie umožňuje
 - Rozlíšenie oprávnenia typu prístupu (*Read / Write*)
 - Určiť, či mapovanie jestvuje (*Present*)
 - Rozlíšenie oprávnenia prístupu (*Kernel / User*)

Stránkovanie

- Stránkovanie umožňuje
 - Rozlíšenie oprávnenia typu prístupu (*Read / Write*)
 - Určiť, či mapovanie jestvuje (*Present*)
 - Rozlíšenie oprávnenia prístupu (*Kernel / User*)
 - Zistiť, či prístup k určitej adrese vyvolal zmenu hodnôt v pamäti (*D – dirty bit*)
 - Zistiť, či bolo vôbec niekedy prístupené k nejakej adrese v stránke (*A – access bit*)

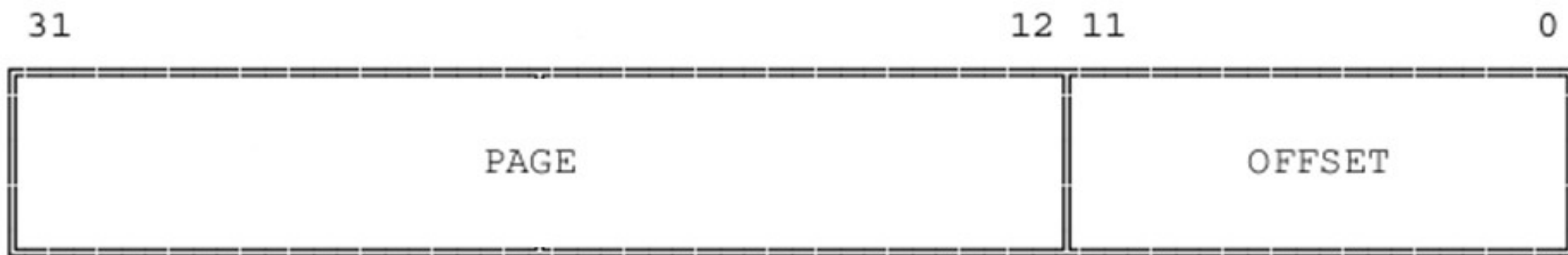
Stránkovanie

- Stránkovanie umožňuje
 - Rozlíšenie oprávnenia typu prístupu (*Read / Write*)
 - Určiť, či mapovanie jestvuje (*Present*)
 - Rozlíšenie oprávnenia prístupu (*Kernel / User*)
 - Zistiť, či prístup k určitej adrese vyvolal zmenu hodnôt v pamäti (*D – dirty bit*)
 - Zistiť, či bolo vôbec niekedy prístupené k nejakej adrese v stránke (*A – access bit*)
- Triky s virtuálnou pamäťou

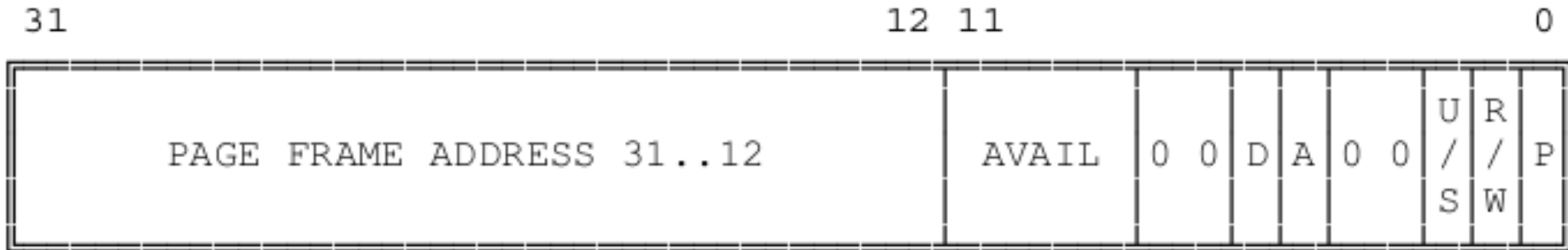
Stránkovanie – čo nám k tomu treba

- Rámec (*page frame*) versus stránka (*page*)
- Lineárna adresa
- Tabuľka stránok
- Položka tabuľky stránok
- Ako sa robí preklad

Formát lineárnej adresy



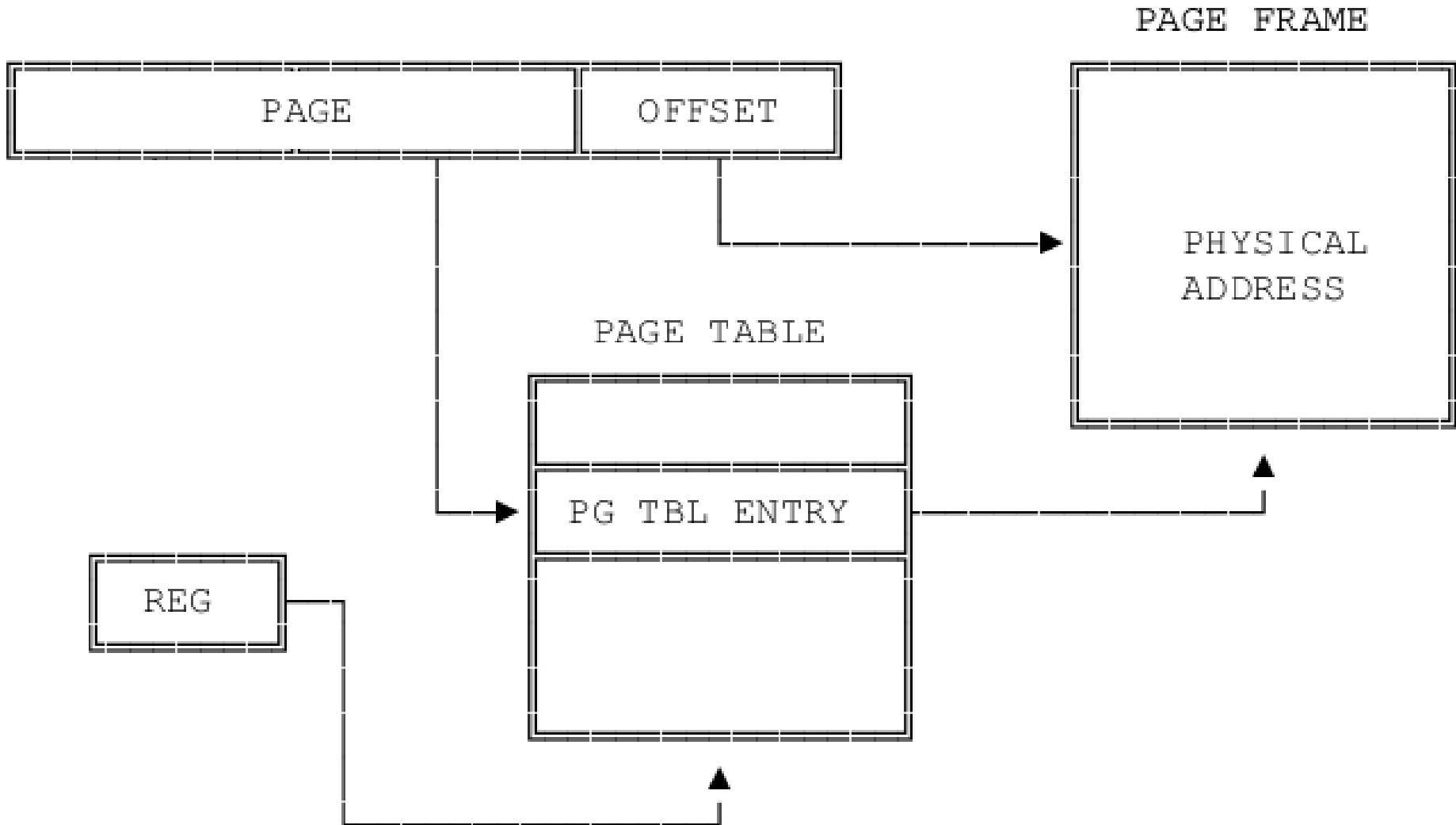
Položka tabuľky stránok (Intel)



- P - PRESENT
- R/W - READ/WRITE
- U/S - USER/SUPERVISOR
- D - DIRTY
- AVAIL - AVAILABLE FOR SYSTEMS PROGRAMMER USE

NOTE: 0 INDICATES INTEL RESERVED. DO NOT DEFINE.

Preklad lineárnej adresy na fyzickú



RISC-V

RISC-V

- Mapuje stránky o velikosti 4 kB
- Zarovnané na hranicu 4 kB (0, 4 k, 8 k, 12 k, ...)

RISC-V

- Mapuje stránky o veľkosti 4 kB
- Zarovnané na hranicu 4 kB (0, 4 k, 8 k, 12 k, ...)
- Koľko bitov potrebujeme na indexovanie (adresovanie) v rámci jednej stránky 4 kB?

RISC-V

- Mapuje stránky o veľkosti 4 kB
- Zarovnané na hranicu 4 kB (0, 4 k, 8 k, 12 k, ...)
- Koľko bitov potrebujeme na indexovanie (adresovanie) v rámci jednej stránky 4 kB? 12

RISC-V

- Mapuje stránky o veľkosti 4 kB
- Zarovnané na hranicu 4 kB (0, 4 k, 8 k, 12 k, ...)
- Koľko bitov potrebujeme na indexovanie (adresovanie) v rámci jednej stránky 4 kB? 12
- Xv6 využíva RISC-V v 64-bitovom adresnom režime

RISC-V

- Mapuje stránky o veľkosti 4 kB
- Zarovnané na hranicu 4 kB (0, 4 k, 8 k, 12 k, ...)
- Koľko bitov potrebujeme na indexovanie (adresovanie) v rámci jednej stránky 4 kB? 12
- Xv6 využíva RISC-V v 64-bitovom adresnom režime
 - Na index do PT sa využíva $64 - 12 = 52$ bitov z VA

RISC-V

- Mapuje stránky o veľkosti 4 kB
- Zarovnané na hranicu 4 kB (0, 4 k, 8 k, 12 k, ...)
- Koľko bitov potrebujeme na indexovanie (adresovanie) v rámci jednej stránky 4 kB? 12
- Xv6 využíva RISC-V v 64-bitovom adresnom režime
 - Na index do PT sa využíva $64 - 12 = 52$ bitov z VA
 - Nie tak celkom, horných 25 bitov z týchto 52 sa nevyužíva, takže index má reálne 27 bitov

RISC-V

- Máme 27 bitov na index, koľko indexov je to spolu?

RISC-V

- Máme 27 bitov na index, koľko indexov je to spolu? $2^{27} = 2^7 * 2^{20} = 128 * 1 \text{ M} = 128 \text{ M}$

RISC-V

- Máme 27 bitov na index, koľko indexov je to spolu? $2^{27} = 2^7 * 2^{20} = 128 * 1 \text{ M} = 128 \text{ M}$
- Každá položka tabuľky má 64 b, čo je ... B

RISC-V

- Máme 27 bitov na index, koľko indexov je to spolu? $2^{27} = 2^7 * 2^{20} = 128 * 1 \text{ M} = 128 \text{ M}$
- Každá položka tabuľky má 64 b, čo je 8 B

RISC-V

- Máme 27 bitov na index, koľko indexov je to spolu? $2^{27} = 2^7 * 2^{20} = 128 * 1 \text{ M} = 128 \text{ M}$
- Každá položka tabuľky má 64 b, čo je 8 B
- Koľko teda bude zaberat' celá PT v pamäti?

RISC-V

- Máme 27 bitov na index, koľko indexov je to spolu? $2^{27} = 2^7 * 2^{20} = 128 * 1 \text{ M} = 128 \text{ M}$
- Každá položka tabuľky má 64 b, čo je 8 B
- Koľko teda bude zaberat' celá PT v pamäti?
 $128 \text{ M} * 8 \text{ B} = \mathbf{1 \text{ GB}}$;)

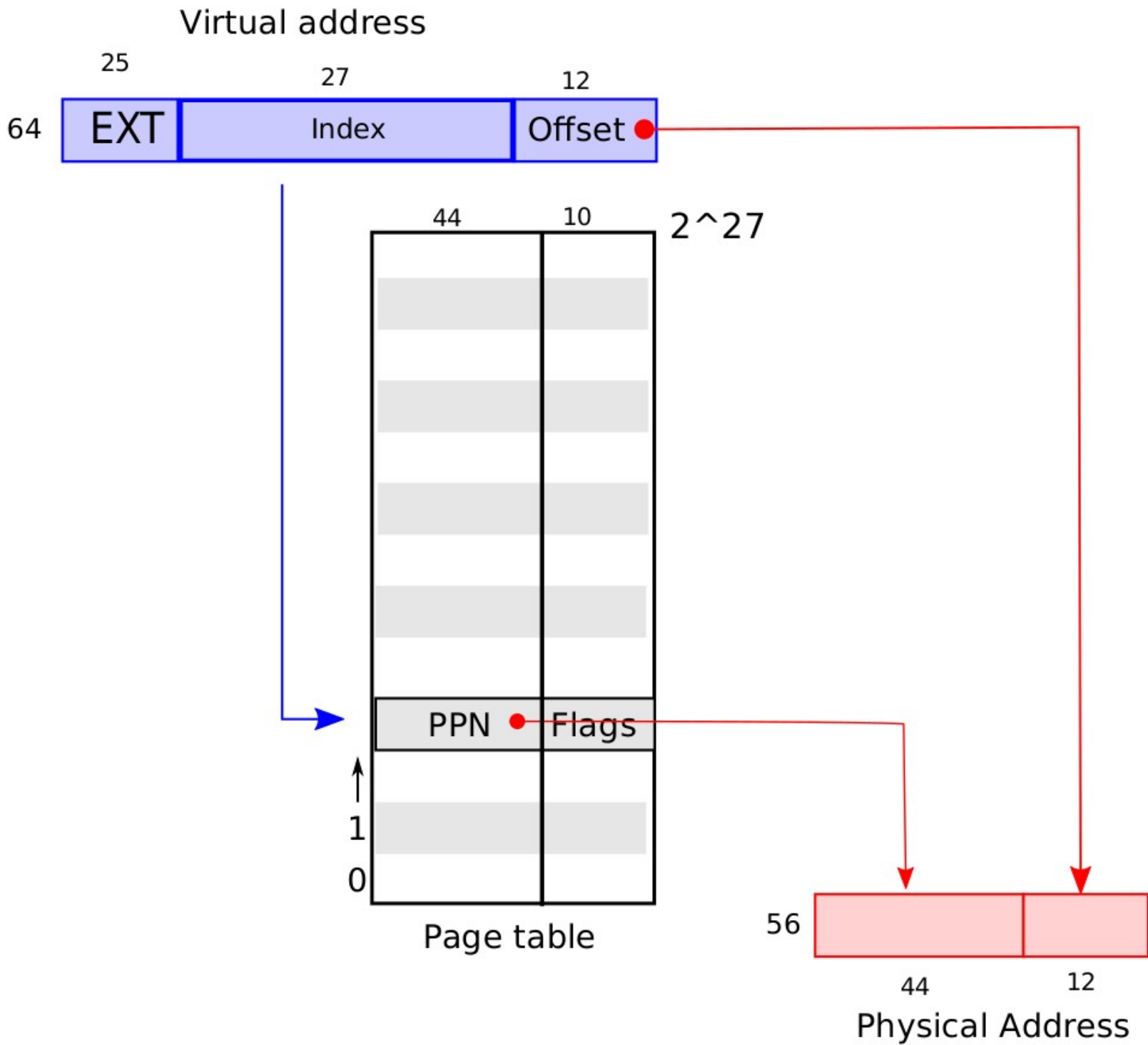
RISC-V

- Máme 27 bitov na index, koľko indexov je to spolu? $2^{27} = 2^7 * 2^{20} = 128 * 1 \text{ M} = 128 \text{ M}$
- Každá položka tabuľky má 64 b, čo je 8 B
- Koľko teda bude zaberat' celá PT v pamäti?
 $128 \text{ M} * 8 \text{ B} = \mathbf{1 \text{ GB}}$;)
- Ak PT pre 1 proces zaberá 1 GB v RAM, koľko procesov asi tak môže bežať v OS? :D :D :D

RISC-V

- Máme 27 bitov na index, koľko indexov je to spolu? $2^{27} = 2^7 * 2^{20} = 128 * 1 \text{ M} = 128 \text{ M}$
- Každá položka tabuľky má 64 b, čo je 8 B
- Koľko teda bude zaberat' celá PT v pamäti?
 $128 \text{ M} * 8 \text{ B} = \mathbf{1 \text{ GB}}$;)
- Ak PT pre 1 proces zaberá 1 GB v RAM, koľko procesov asi tak môže bežať v OS? :D :D :D
- Xv6 má 128 MB RAM... Tak tu **niečo nesedí...**

Preklad MMU



Preklad MMU

- Záznam v tabuľke PT sa nazýva PTE (*Page Table Entry*)
 - Má 64 b (8 B), ale využíva sa “iba” 54

Preklad MMU

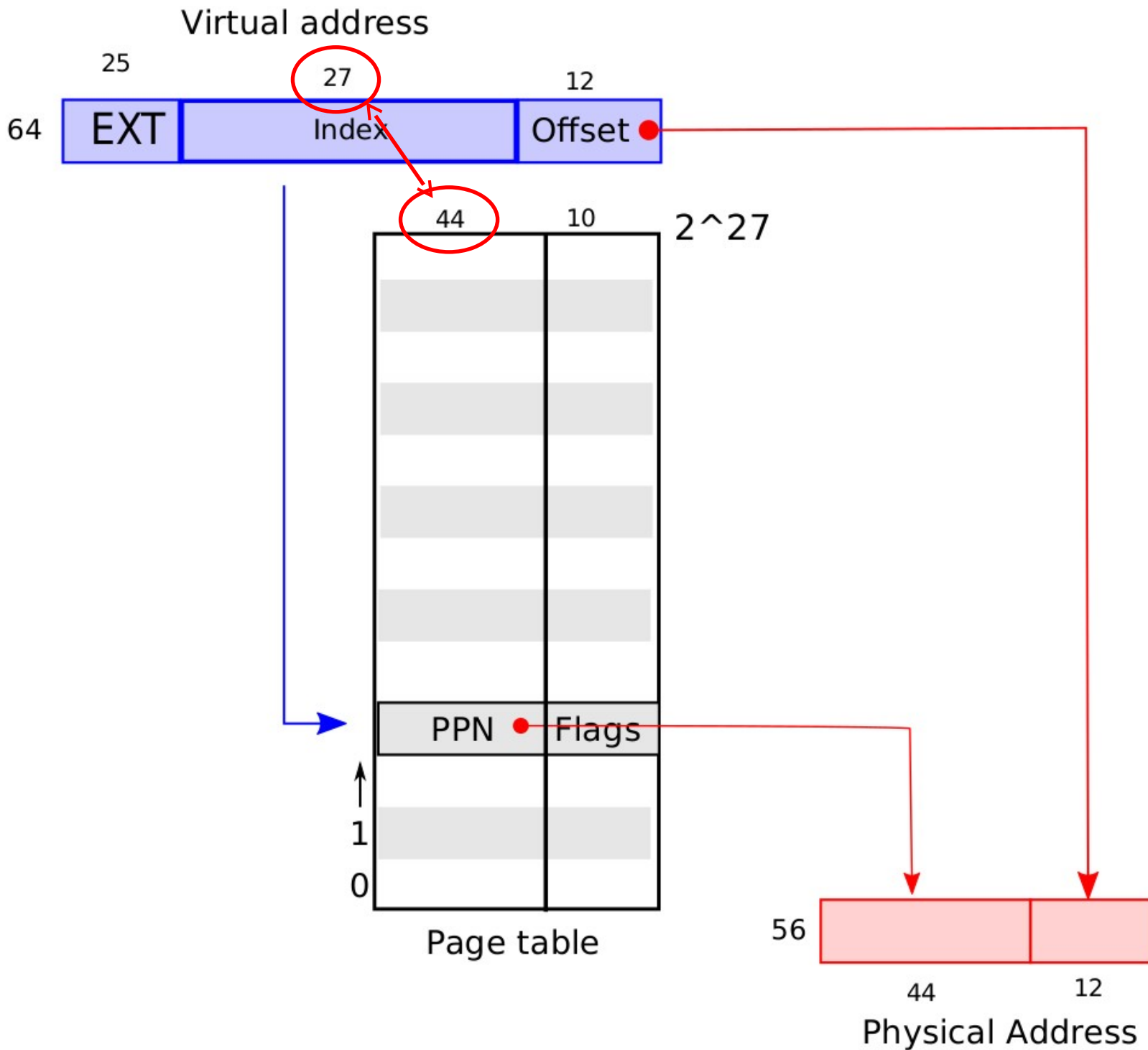
- Záznam v tabuľke PT sa nazýva PTE (*Page Table Entry*)
 - Má 64 b (8 B), ale využíva sa “iba” 54
 - Horných 44 bitov PTE tvoria horných 44 bitov fyzickej adresy (*Physical Page Number = PPN*)

Preklad MMU

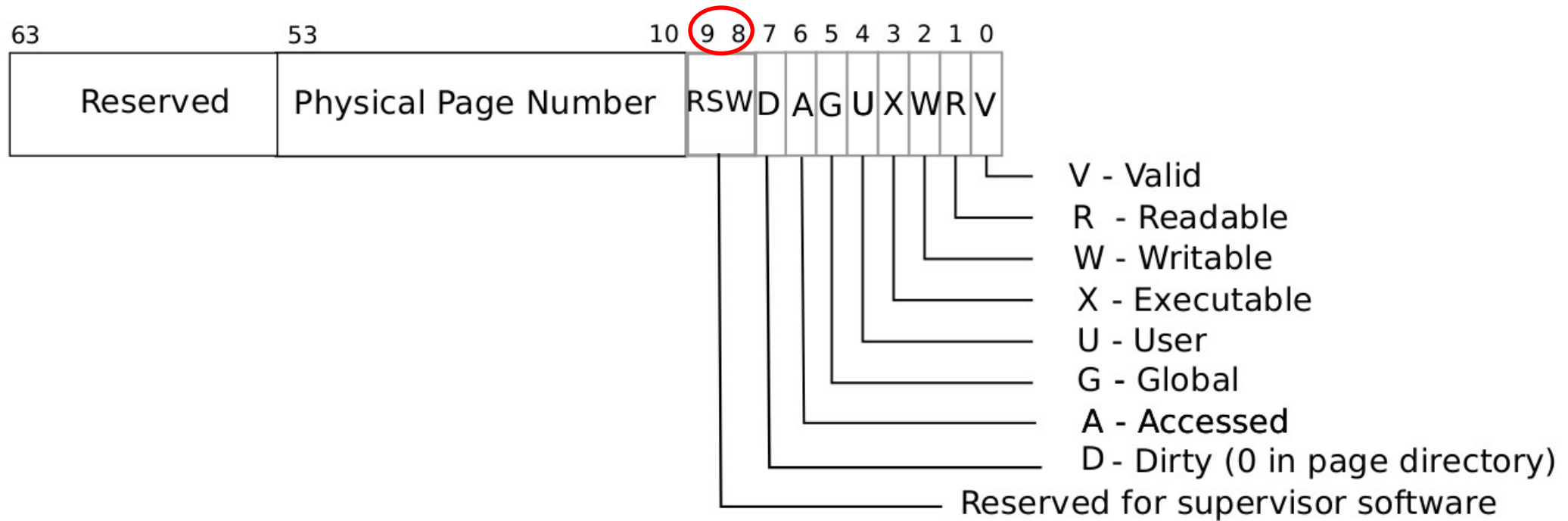
- Záznam v tabuľke PT sa nazýva PTE (*Page Table Entry*)
 - Má 64 b (8 B), ale využíva sa “iba” 54
 - Horných 44 bitov PTE tvoria horných 44 bitov fyzickej adresy (*Physical Page Number = PPN*)
 - Spodných 10 sú tzv. príznaky
 - *Present, Writeable, User, Accessed, Dirty...*

Preklad MMU

- Záznam v tabuľke PT sa nazýva PTE (*Page Table Entry*)
 - Má 64 b (8 B), ale využíva sa “iba” 54
 - Horných 44 bitov PTE tvoria horných 44 bitov fyzickej adresy (*Physical Page Number* = PPN)
 - Spodných 10 sú tzv. príznaky
 - *Present, Writeable, User, Accessed, Dirty...*
- **!POZOR! veľkosť VA != veľkosť PA**



PTE



PT

- Kde je PT uložená? V RAM
- MMU dokáže manipulovat' so záznamami PTE
- Podobne to dokáže aj OS

PT

- Ako sme už vypočítali, veľkosť PT je 1 GB!
 - 1 PT na 1 adresný priestor
 - 1 adresný priestor zväčša pre 1 aplikáciu (proces)

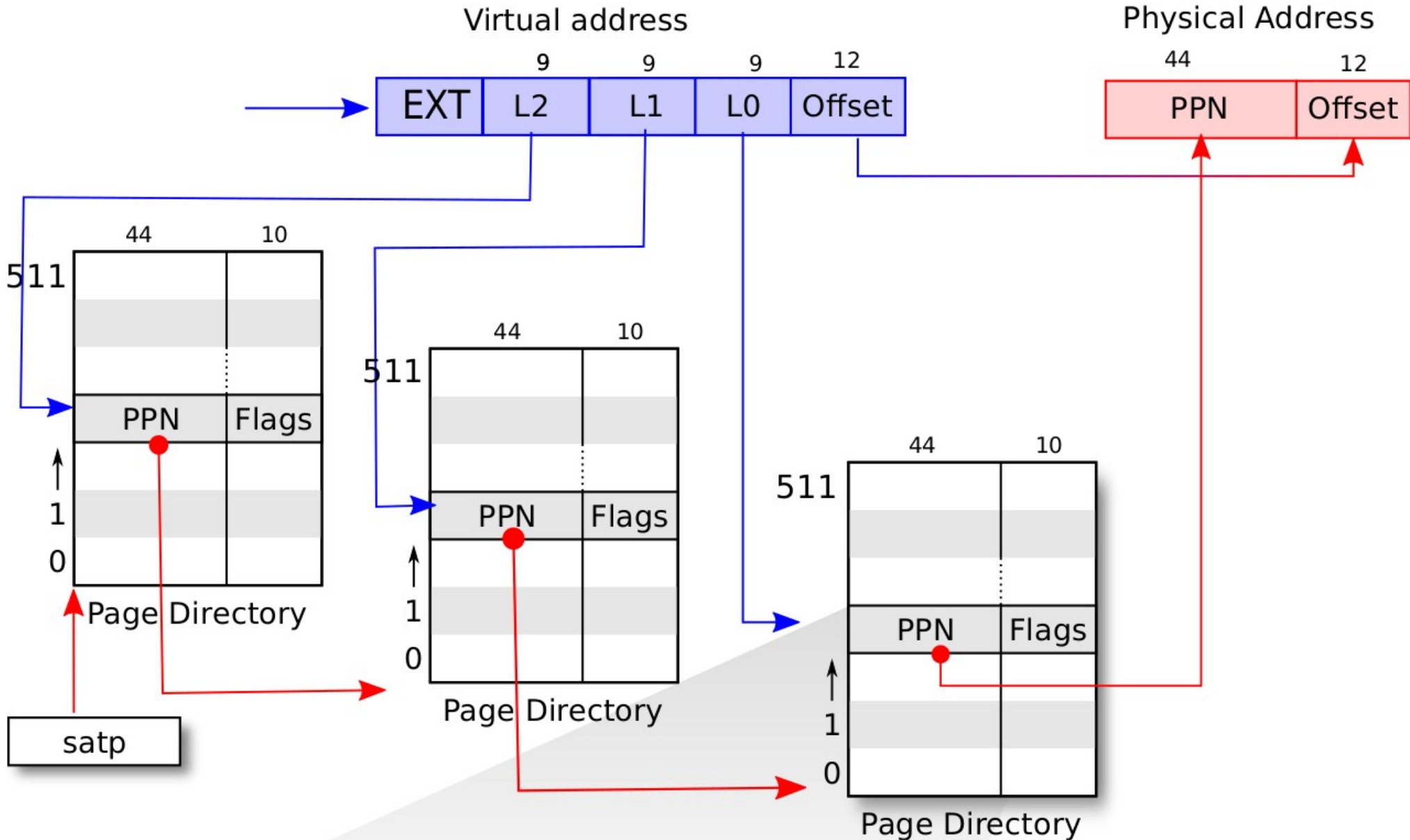
PT

- Ako sme už vypočítali, veľkosť PT je 1 GB!
 - 1 PT na 1 adresný priestor
 - 1 adresný priestor zväčša pre 1 aplikáciu (proces)
- Avšak proces často potrebuje iba pár k/MB pamäte, takže iba pár k/M záznamov PTE bude využitých, ostatné budú prázdne!

PT

- RISC-V 64 využíva na ušetrenie miesta tzv. **trojúrovňovú** tabuľku stránok

PT RISC-V



PT

- V každej úrovni (L0, L1, L2) máme k dispozícii 9 bitov na index → 512 položiek v PD (*Page Directory*)

PT

- V každej úrovni (L0, L1, L2) máme k dispozícii 9 bitov na index → 512 položiek v PD (*Page Directory*)
- $512 * 512 * 512 = 2^9 * 2^9 * 2^9 = 2^{(3*9)} = 2^{27}$

PT

- V každej úrovni (L0, L1, L2) máme k dispozícii 9 bitov na index → 512 položiek v PD (*Page Directory*)
- $512 * 512 * 512 = 2^9 * 2^9 * 2^9 = 2^{(3*9)} = 2^{27}$
- PTE môže byť neplatné (bit *Valid*); nejestvuje prepojenie t.j. mapovanie s RAM)

PT

- V každej úrovni (L0, L1, L2) máme k dispozícii 9 bitov na index → 512 položiek v PD (*Page Directory*)
- $512 * 512 * 512 = 2^9 * 2^9 * 2^9 = 2^{(3*9)} = 2^{27}$
- PTE môže byť neplatné (bit *Valid*); nejestvuje prepojenie t.j. mapovanie s RAM)
- Preto môže byť PT pre proces malá!

PT

- V jednej úrovni máme 512 položiek PTE; jedna má veľkosť 64 b, takže celkovo je to $512 * 8 \text{ B} = 4 \text{ kB}$

PT

- V jednej úrovni máme 512 položiek PTE; jedna má veľkosť 64 b, takže celkovo je to $512 * 8 \text{ B} = 4 \text{ kB}$ (čírou “náhodou” to sedí na veľkosť stránky?)

PT

- V jednej úrovni máme 512 položiek PTE; jedna má veľkosť 64 b, takže celkovo je to $512 * 8 \text{ B} = 4 \text{ kB}$ (čírou “náhodou” to sedí na veľkosť stránky?)
- 512 záznamov môže ukazovať na 512 stránok v RAM; 1 tabuľkou vieme “pokryť” max. $512 * 4 \text{ kB RAM}$, t.j. 2048 kB, t.j. 2 MB

PT

- V jednej úrovni máme 512 položiek PTE; jedna má veľkosť 64 b, takže celkovo je to $512 * 8 \text{ B} = 4 \text{ kB}$ (čírou “náhodou” to sedí na veľkosť stránky?)
- 512 záznamov môže ukazovať na 512 stránok v RAM; 1 tabuľkou vieme “pokryť” max. $512 * 4 \text{ kB RAM}$, t.j. 2048 kB, t.j. 2 MB
- Na pokrytie mapovania niekoľko desiatok MB nám stačia desiatky-stovky kB tabuľky PT namiesto 1 GB

PT

- Ako MMU “vie”, kde v RAM sa PT nachádza?

PT

- Ako MMU “vie”, kde v RAM sa PT nachádza?
- Na RISC-V je **FYZICKÁ** adresa hornej časti PT (pre index L2) v registri `satp`
- Prepísaním `satp` sa prepínajú adresné priestory!

PT

- Ako MMU “vie”, kde v RAM sa PT nachádza?
- Na RISC-V je **FYZICKÁ** adresa hornej časti PT (pre index L2) v registri `satp`
- Prepísaním `satp` sa prepínajú adresné priestory!
- Stránky PT môžu byť voľne roztrúsené v RAM, nemusí ísť o súvislú oblasť RAM!!!

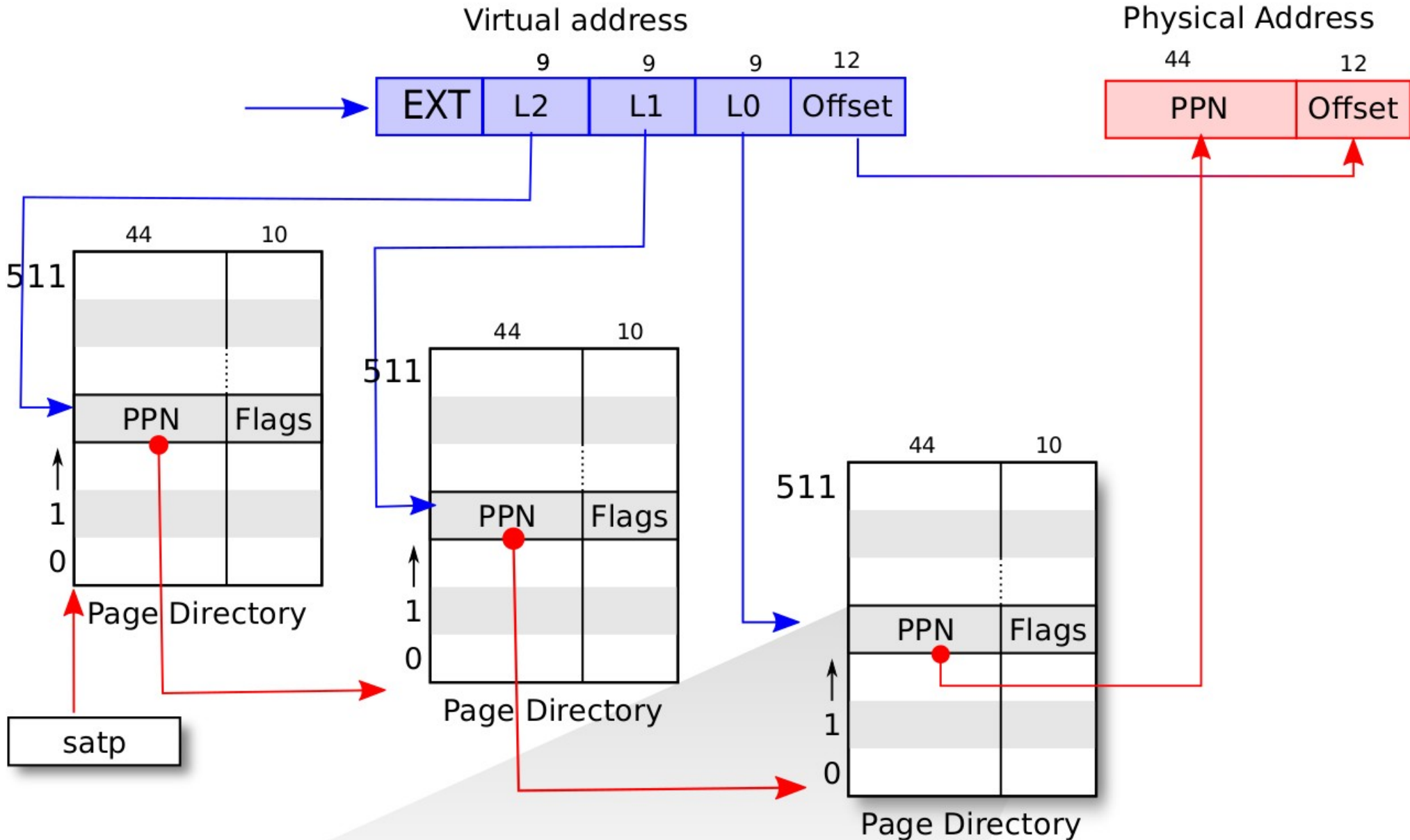
Preklad va2pa

- MMU musí nájsť správny záznam PTE zodpovedajúci danej VA; ako?

Preklad va2pa

- MMU musí nájsť správny záznam PTE zodpovedajúci danej VA; ako?
 - Z registra `satp` vieme PA pre obsah tabuľky PD_{L2}
 - Horných 9 bitov indexu VA (L2) ukazuje do PD_{L2} ; z $PD_{L2}[L2]$ získame PA pre obsah PD_{L1}
 - Ďalších 9 bitov indexu VA (L1) ukazuje do PD_{L1} ; z $PD_{L1}[L1]$ získame PA pre obsah PD_{L0}
 - Posledných 9 bitov indexu VA (L0) ukazuje do PD_{L0} ; z $PD_{L0}[L0]$ získame PA pre PTE
 - $PA = PPN$ z PTE plus spodných 12 bitov VA

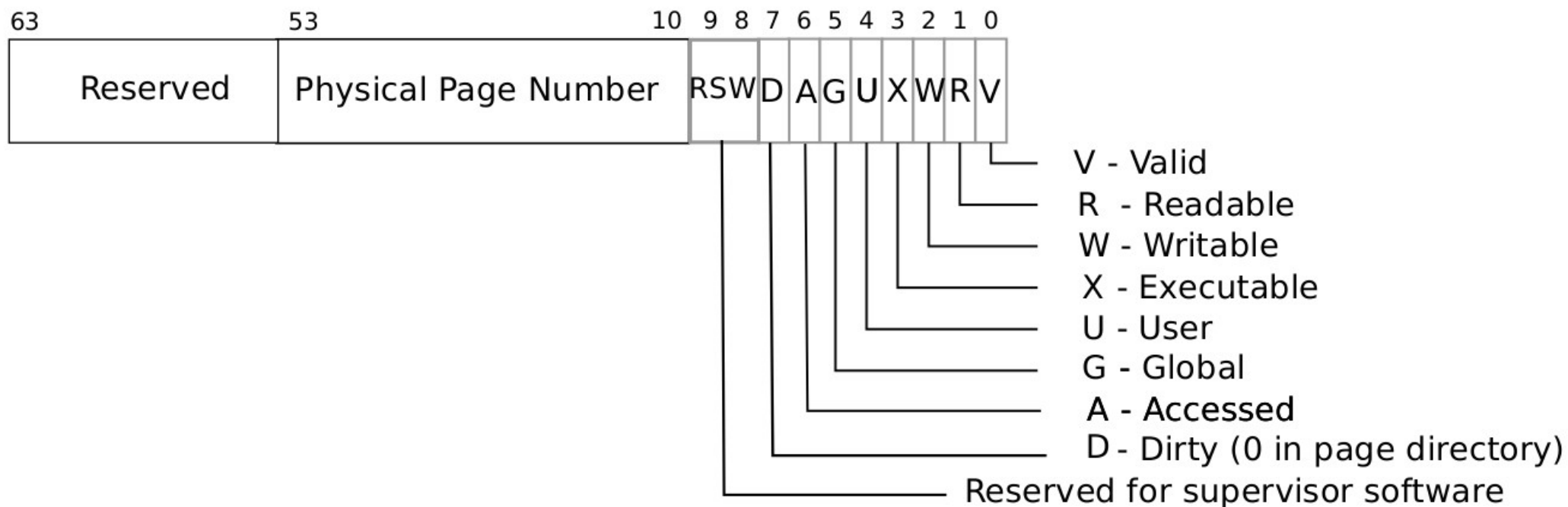
PT RISC-V



Príznamy PTE

- Xv6 využíva V, R, W, X, U
- V labe 3 musíte využiť príznak A
- Neskôr využijeme aj niektorý z tých, ktoré sú voľne dostupné pre OS

Príznaky PTE



Príznaky PTE

- Čo ak bit V nie je nastavený? Alebo zapisujeme a nie je nastavený W?

Príznaky PTE

- Čo ak bit V nie je nastavený? Alebo zapisujeme a nie je nastavený W?
- Výpadok stránky (ang. *Page Fault*)

Príznaky PTE

- Čo ak bit V nie je nastavený? Alebo zapisujeme a nie je nastavený W?
- Výpadok stránky (ang. *Page Fault*)
 - Vynútený presun do jadra (trap.c)

Príznaky PTE

- Čo ak bit V nie je nastavený? Alebo zapisujeme a nie je nastavený W?
- Výpadok stránky (ang. *Page Fault*)
 - Vynútený presun do jadra (trap.c)
 - Jadro buď vypíše chybovú správu a ukončí proces, ktorý chybu spôsobil (“usertrap(): unexpected scause...”) (viď ukážka cat.c)

Príznaky PTE

- Čo ak bit V nie je nastavený? Alebo zapisujeme a nie je nastavený W?
- Výpadok stránky (ang. *Page Fault*)
 - Vynútený presun do jadra (trap.c)
 - Jadro buď vypíše chybovú správu a ukončí proces, ktorý chybu spôsobil (“usertrap(): unexpected scause...”) (vid’ ukážka cat.c)
 - Alebo môže nainštalovať chýbajúci PTE a obnoviť beh procesu (napr. ak sa používa swapovanie pamäte RAM na disk)

Výhody stránkovania

- Spojitý virtuálny adresný priestor nevyžaduje spojitý fyzický adresný priestor! (vôbec neprichádza ku externej fragmentácii!!!)

Výhody stránkovania

- Spojitý virtuálny adresný priestor nevyžaduje spojitý fyzický adresný priestor! (vôbec neprichádza ku externej fragmentácii!!!)
- “*lazy allocation*”; alokácia pamäte až pri jej prvom použití (nastane výpadok, jadro alokuje PTE a proces reštartuje inštrukciu)

Výhody stránkovania

- Spojitý virtuálny adresný priestor nevyžaduje spojitý fyzický adresný priestor! (vôbec neprichádza ku externej fragmentácii!!!)
- “*lazy allocation*”; alokácia pamäte až pri jej prvom použití (nastane výpadok, jadro alokuje PTE a proces reštartuje inštrukciu)
- “*copy-on-write fork*”; kópia stránky až pri prvom pokuse o zápis

Virtuálna pamäť v jadre

- Pre procesy je využitie VM v poriadku, ale prečo ju využívať aj v jadre?

Virtuálna pamäť v jadre

- Pre procesy je využitie VM v poriadku, ale prečo ju využívať aj v jadre?
- Môže bežať jadro iba s fyzickou pamäťou?

Virtuálna pamäť v jadre

- Pre procesy je využitie VM v poriadku, ale prečo ju využívať aj v jadre?
- Môže bežať jadro iba s fyzickou pamäťou?
ÁNO, môže!

Virtuálna pamäť v jadre

- Pre procesy je využitie VM v poriadku, ale prečo ju využívať aj v jadre?
- Môže bežať jadro iba s fyzickou pamäťou?
ÁNO, môže!
- Väčšina jadier OS však využíva VA; prečo?

Virtuálna pamäť v jadre

- Pre procesy je využitie VM v poriadku, ale prečo ju využívať aj v jadre?
- Môže bežať jadro iba s fyzickou pamäťou?
ÁNO, môže!
- Väčšina jadier OS však využíva VA; prečo?
 - Je zložité vypínať/zapínať stránkovanie

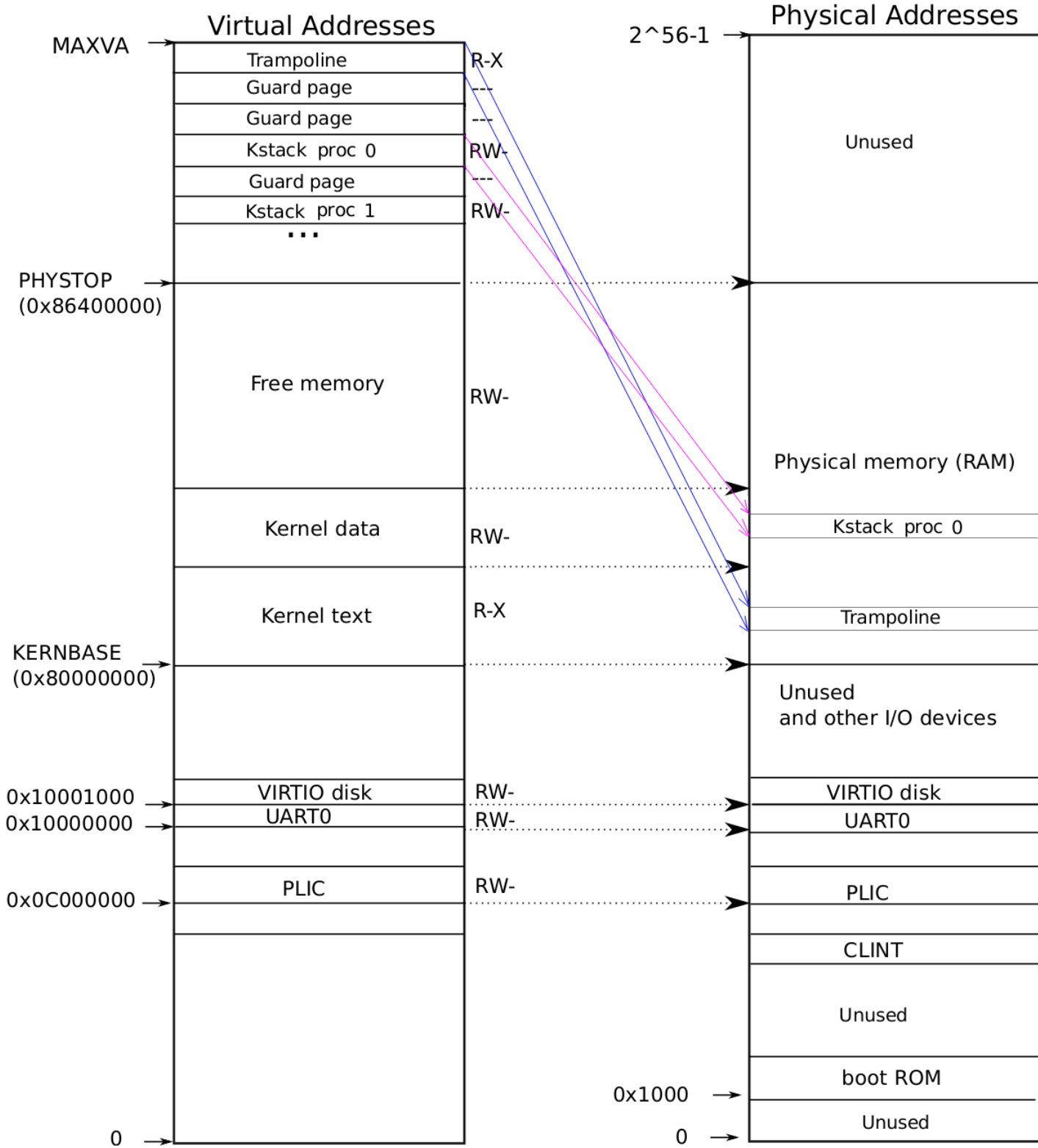
Virtuálna pamäť v jadre

- Pre procesy je využitie VM v poriadku, ale prečo ju využívať aj v jadre?
- Môže bežať jadro iba s fyzickou pamäťou?
ÁNO, môže!
- Väčšina jadier OS však využíva VA; prečo?
 - Je zložité vypínať/zapínať stránkovanie
 - Uľahčuje to hľadanie chýb
 - Text (kód) jadra označíme X, údaje nie
 - Ponechanie pamätevej “diery” pod zásobníkom

Virtuálna pamäť v jadre

- Pre procesy je využitie VM v poriadku, ale prečo ju využívať aj v jadre?
- Môže bežať jadro iba s fyzickou pamäťou?
ÁNO, môže!
- Väčšina jadier OS však využíva VA; prečo?
 - Je zložité vypínať/zapínať stránkovanie
 - Uľahčuje to hľadanie chýb
 - Text jadra označíme X, údaje nie
 - Ponechanie pamäťovej “diery” pod zásobníkom
 - Uľahčuje prechod medzi user/kernel (tým istým mapovaním tej istej stránky – vid’ *trampoline*)

Virtuálna pamäť JADRA xv6



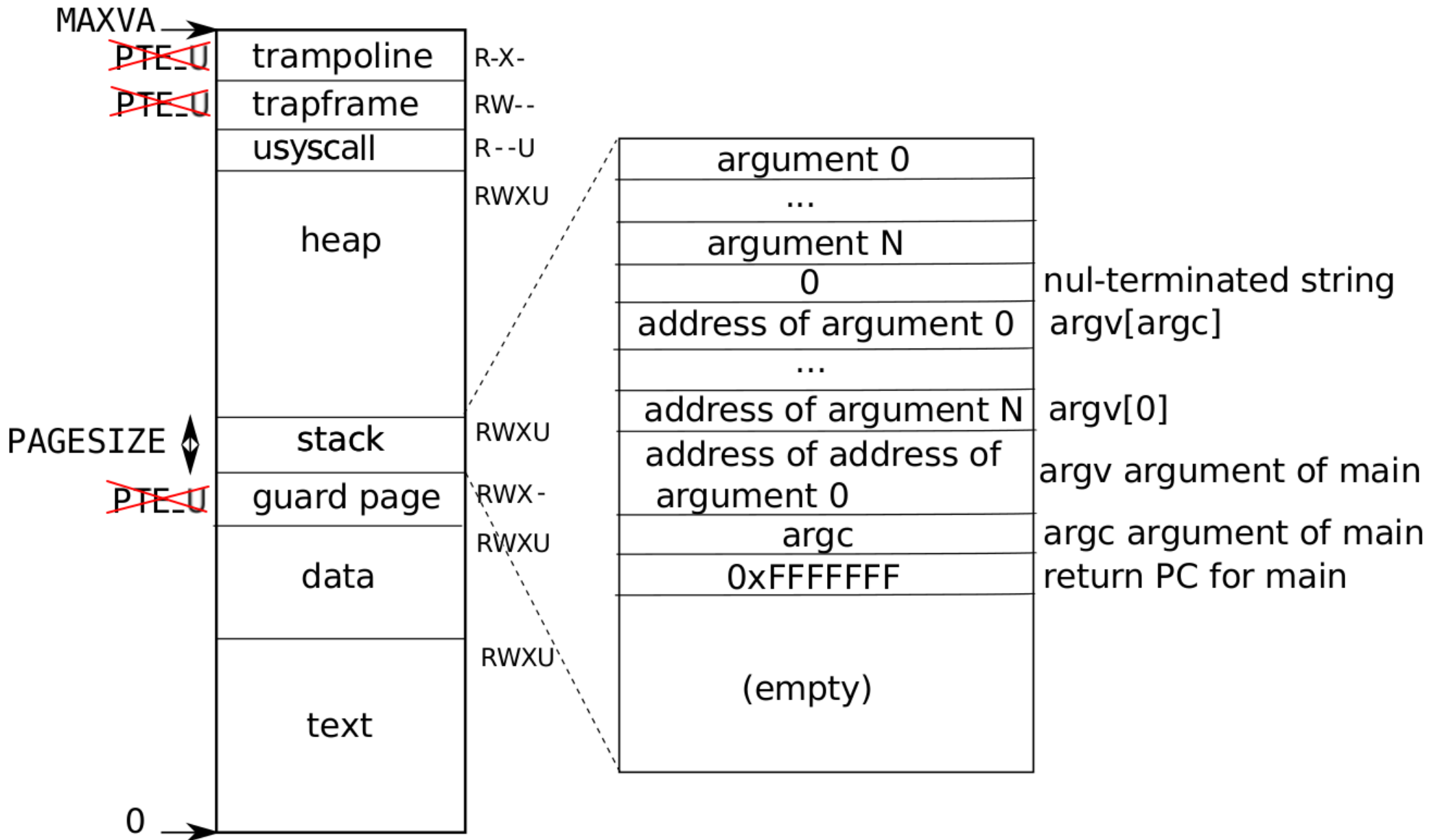
Virtuálna pamäť JADRA xv6

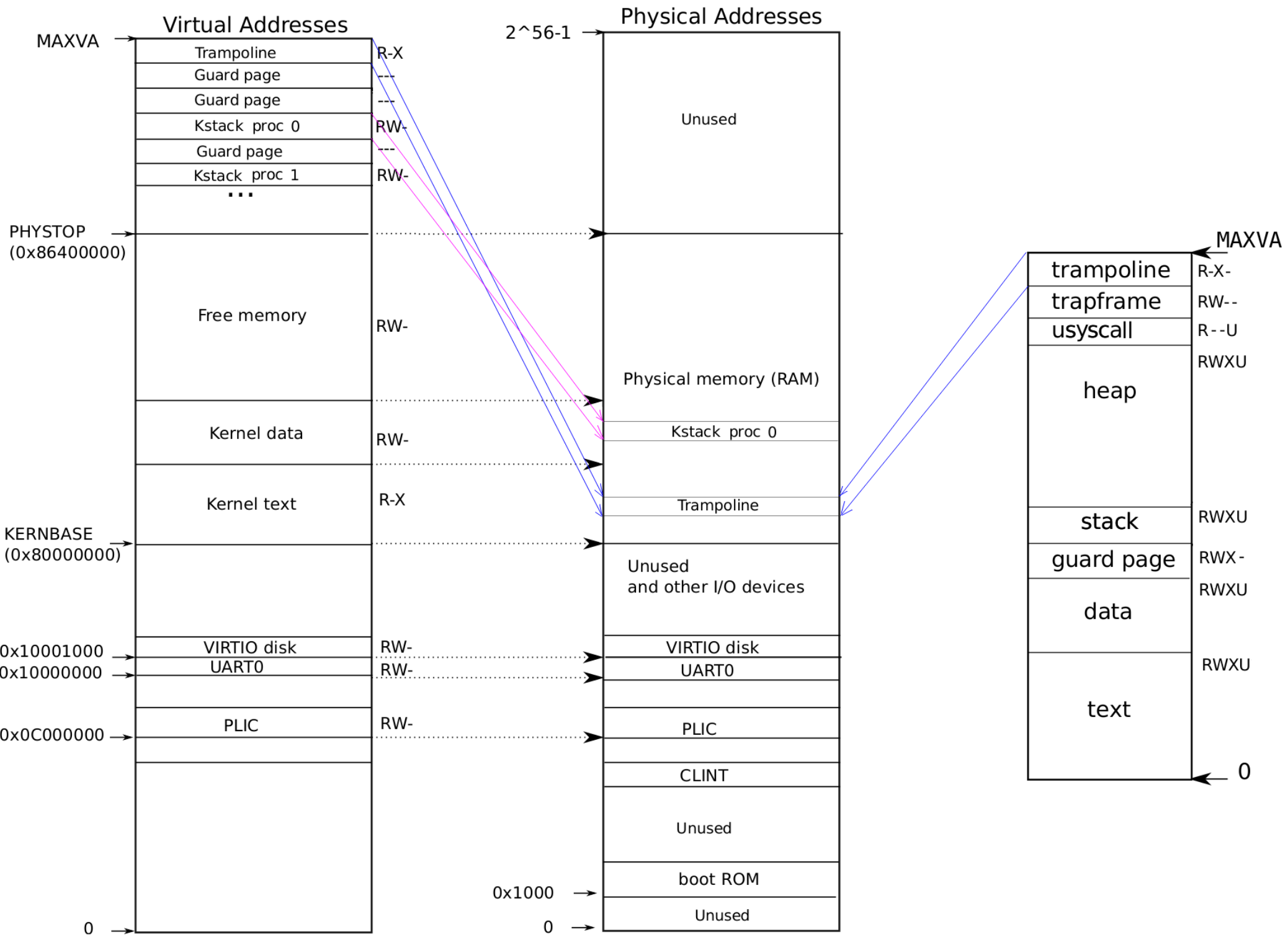
- Jednoduché mapovanie virtuálnej pamäte na fyzickú jedna-k-jednej
- Prečo sa mapujú aj zariadenia?
- Vid' oprávnenia rôznych oblastí...

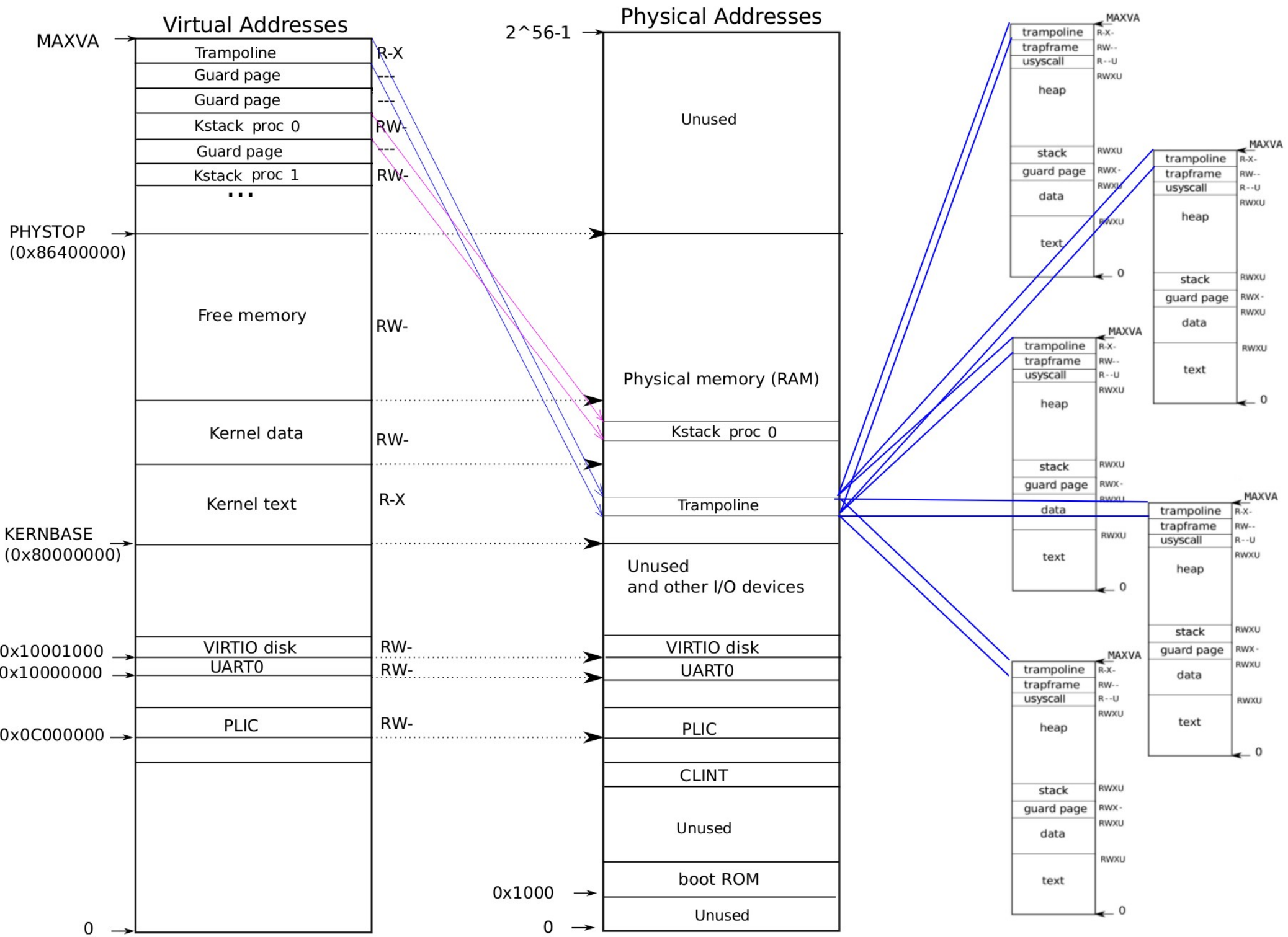
Virtuálna pamäť UŽÍVATEĽA xv6

- Každý proces má vlastný adresný priestor
- Vlastnú tabuľku stránok
- Vid' `trampoline` a `trapframe` – nie sú zapisovateľné pre užívateľský proces!!!
- Jadro OS nastavuje register `satp` pri prepínaní procesov (`usertrapret()` `kernel/trap.c:122`)

Virtuálna pamäť UŽÍVATEĽA xv6







Virtuálna pamäť xv6

- Vlastnosti takejto organizácie adresného priestoru

Virtuálna pamäť xv6

- Vlastnosti takejto organizácie adresného priestoru
 - VA užívateľa začína od 0 (avšak v každom procese je VA užívateľa od 0 mapovaná na inú RAM – vo všeobecnosti)

Virtuálna pamäť xv6

- Vlastnosti takejto organizácie adresného priestoru
 - VA užívateľa začína od 0 (avšak v každom procese je VA užívateľa od 0 mapovaná na inú RAM – vo všeobecnosti)
 - 256 GB halda užívateľa ;) (viď MAXVA)

Virtuálna pamäť xv6

- Vlastnosti takejto organizácie adresného priestoru
 - VA užívateľa začína od 0 (avšak v každom procese je VA užívateľa od 0 mapovaná na inú RAM – vo všeobecnosti)
 - 256 GB halda užívateľa ;) (viď MAXVA)
 - Jednoduchý prechod user ↔ kernel mapovaním trampolíny a *trapframe* (o tom nabudúce)

Virtuálna pamäť xv6

- Vlastnosti takejto organizácie adresného priestoru
 - VA užívateľa začína od 0 (avšak v každom procese je VA užívateľa od 0 mapovaná na inú RAM – vo všeobecnosti)
 - 256 GB halda užívateľa ;) (viď MAXVA)
 - Jednoduchý prechod user \leftrightarrow kernel mapovaním trampolíny a *trapframe* (o tom nabudúce)
 - Neľahký prístup jadra k pamäti užívateľa!!!
 - Ľahký prístup jadra k fyzickej pamäti: $pa(x)$ mapovaná na $va(x)$

Kód VM xv6

- Inicializácia adresného priestoru jadra
- Kernel/memlayout.h a kernel/vm.c kvminit()

Kód VM xv6

- Inicializácia adresného priestoru jadra
- Kernel/memlayout.h a kernel/vm.c kvminit()
 - Koľko adresného priestoru vie obsiahnuť 1 L0 položka?
 - Koľko 1 L1 položka?
 - Koľko 1 L2 položka?
 - Ako veľký je celý adresný priestor?

Kód VM xv6

- Inicializácia adresného priestoru jadra
- Kernel/memlayout.h a kernel/vm.c kvminit()
 - Koľko adresného priestoru vie obsiahnuť 1 L0 položka? 4 kB
 - Koľko 1 L1 položka? 2 MB
 - Koľko 1 L2 položka? 1 GB
 - Ako veľký je celý adresný priestor? 512 GB

Kód VM xv6

- Koľko pamäte (stránok) je použitých na reprezentáciu PT (t.j. mapovania adresného priestoru) po prvom volaní `kvmmap()`?
 - `kernel/vm.c:28 kvmmap(kpgtbl, UART0, UART0, PGSIZE, PTE_R | PTE_W);`
- Koľko pamäte (stránok) sa týmto volaním mapuje?
- Ukážka `vmprint()` za týmto prvým volaním

Kód VM xv6

- Koľko položiek v PT jadra zaberá VIRTIO
 - `kvmmap(kpgtbl, VIRTIO0, ...);`
- Koľko položiek v PT jadra zaberá PLIC
 - `kvmmap(kpgtbl, PLIC, ..., 0x4000000, ...);`
- Je trampolína mapovaná v `kpgtbl` iba raz?
- A čo zásobníky procesov?

