

OS MMXX

MIT ;)

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

Virtualna pamat

Struktura prednasky

- Adresne priestory
- Strankovaci hardver
- Programovy kod xv6 pre spravu VM

Tema

Tema

- Majme program, ktory z casu na cas zapise nieco na nahodnu adresu v pamati

Tema

- Majme program, ktory z casu na cas zapise nieco na nahodnu adresu v pamati
- Ako udrzat takyto program “na uzde”?

Tema

- Majme program, ktory z casu na cas zapise nieco na nahodnu adresu v pamati
- Ako udrzat takyto program “na uzde”?
- Izolujme adresne priestory procesov (a jadra)

Izolacia adresneho priestoru

Izolacia adresneho priestoru

- Kazdy proces ma vlastny adresny priestor

Izolacia adresneho priestoru

- Kazdy proces ma vlastny adresny priestor
- Moze citat/zapisovat iba v ramci svojho priestoru (nemoze citat ani zapisovat v inom)

Izolacia adresneho priestoru

- Kazdy proces ma vlastny adresny priestor
- Moze citat/zapisovat iba v ramci svojho priestoru (nemoze citat ani zapisovat v inom)
- **Ako SUCASNE implementovat viac adresnych priestorov v ramci jedinej fyzickej pamate (RAM) a zabezpecit izolaciju medzi nimi?**

Izolacia adresneho priestoru

- Odpoved: strankovaci hardver

Izolacia adresneho priestoru

- Odpoved: strankovaci hardver
- Xv6 vyuziva hardver procesora RISC-V

Strankovanie

Strankovanie

- Poskytuje nepriamu adresáciu

Strankovanie

- Poskytuje nepriamu adresáciu
 - program používa ukazatele (adresy); tieto adresy vyuziva CPU na pristup k pamati

Strankovanie

- Poskytuje nepriamu adresáciu
 - program používa ukazatele (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!

Strankovanie

- Poskytuje nepriamu adresáciu
 - program používa ukazatele (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 spravy pamäte (MMU) “preklada” virtuálne adresy na fyzické adresy

Strankovanie

- Poskytuje nepriamu adresáciu
 - program používa ukazatele (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 spravy pamäte (MMU) “preklada” virtuálne adresy na fyzické adresy
 - V literatúre sa často spomína aj termín “lineárna adresa” (viď rozdiel “lineárna” a “virtuálna”)
 - Az fyzická adresa je adresou operandu na adresnej zbernici (nemusi to byť iba RAM!)

Strankovanie

- Program dokaze pracovat iba s virtualnymi adresami, nie fyzickymi (co sa tyka menenia/citania obsahu RAM)

Strankovanie

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

Strankovanie

- MMU pouziva na “preklad” tabulku, v ktorej

Strankovanie

- MMU pouziva na “preklad” tabulku, v ktorej
 - VA je indexom
 - PA je hodnotou na indexe

Strankovanie

- MMU pouziva na “preklad” tabulku, v ktorej
 - VA je indexom
 - PA je hodnotou na indexe
- Tabulka ma nazov “Tabulka stranok” (Page Table)

Strankovanie

- MMU pouziva na “preklad” tabulku, v ktorej
 - VA je indexom
 - PA je hodnotou na indexe
- Tabulka ma nazov “Tabulka stranok” (Page Table)
- Jedna tabulka popisuje jeden adresny priestor (mapovanie VA na PA)

Strankovanie

- MMU pouziva na “preklad” tabulku, v ktorej
 - VA je indexom
 - PA je hodnotou na indexe
- Tabulka ma nazov “Tabulka stranok” (Page Table)
- Jedna tabulka popisuje jeden adresny priestor (mapovanie VA na PA)
- MMU dokaze obmedzit, ktore VA su pristupne v user mode CPU

Strankovanie

- Odkaz na aktualnu tabulku stranok (tabulku, ktora sa pri preklade pouziva), sa nachadza v specialnom registri CPU

Strankovanie

- Odkaz na aktualnu tabulku stranok (tabulku, ktora sa pri preklade pouziva), sa nachadza v specialnom registri CPU
- Menit hodnotu tohto CPU je mozne iba v privilegovanom rezime (kernel mod)
 - Pokus o zmenu v user mode vedie ku vynimke

Strankovanie

- CPU startuje v rezime, kedy je strankovanie vypnute
 - Preto? Ak by bolo zapnute, tak máme problém sliedky a vajca...

Strankovanie

- CPU startuje v rezime, kedy je strankovanie vypnute
 - Preto? Ak by bolo zapnute, tak máme problém sliepky a vajca...
- Inicializacny kod jadra
 - Vyplni tabulku
 - Nastavi register, ktorý ukazuje na tabulku stránok
 - Zapne strankovanie CPU

Strankovanie

- Strankovanie umožňuje

Strankovanie

- Strankovanie umožňuje
 - Rozlisenie opravenia typu pristupu (Read / Write)

Strankovanie

- Strankovanie umožňuje
 - Rozlisenie opravenia typu pristupu (Read / Write)
 - Urcit, ci mapovanie jestvuje (Present)

Strankovanie

- Strankovanie umožňuje
 - Rozlisenie opravenia typu pristupu (Read / Write)
 - Urcit, ci mapovanie jestvuje (Present)
 - Rozlisenie opravenia pristupu (Kernel / User)

Strankovanie

- Strankovanie umožňuje
 - Rozlisenie opravenia typu pristupu (Read / Write)
 - Urcit, ci mapovanie jestvuje (Present)
 - Rozlisenie opravenia pristupu (Kernel / User)
 - Zistit, ci pristup k urcitej adrese vyvolal zmenu hodnot v pamati (Dirty)

Strankovanie

- Strankovanie umožňuje
 - Rozlisenie opravenia typu pristupu (Read / Write)
 - Urcit, ci mapovanie jestvuje (Present)
 - Rozlisenie opravenia pristupu (Kernel / User)
 - Zistit, ci pristup k urcitej adrese vyvolal zmenu hodnot v pamati (Dirty)
 - Zistit, ci bolo vobec niekedy pristupene k nejakej adrese (A – access bit)

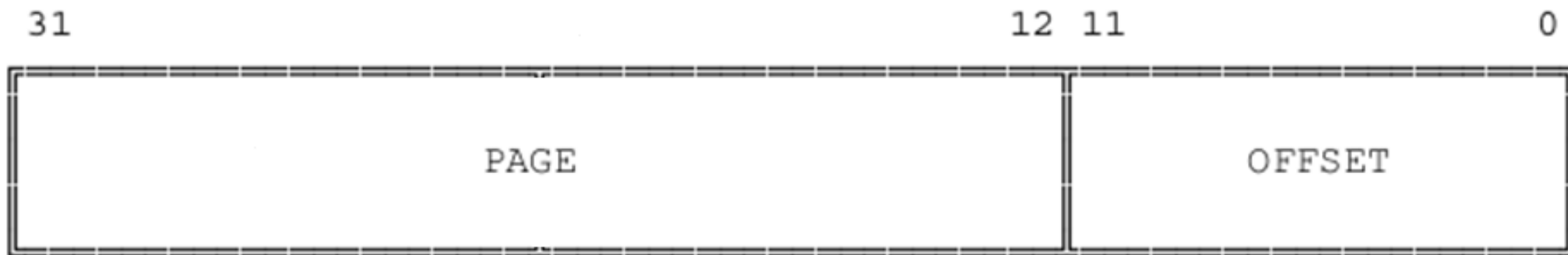
Strankovanie

- Strankovanie umožňuje
 - Rozlisenie opravenia typu pristupu (Read / Write)
 - Urcit, ci mapovanie jestvuje (Present)
 - Rozlisenie opravenia pristupu (Kernel / User)
 - Zistit, ci pristup k urcitej adrese vyvolal zmenu hodnot v pamati (Dirty)
 - Zistit, ci bolo vobec niekedy pristupene k nejakej adrese (A – access bit)
- Triky s virtualnou pamatou

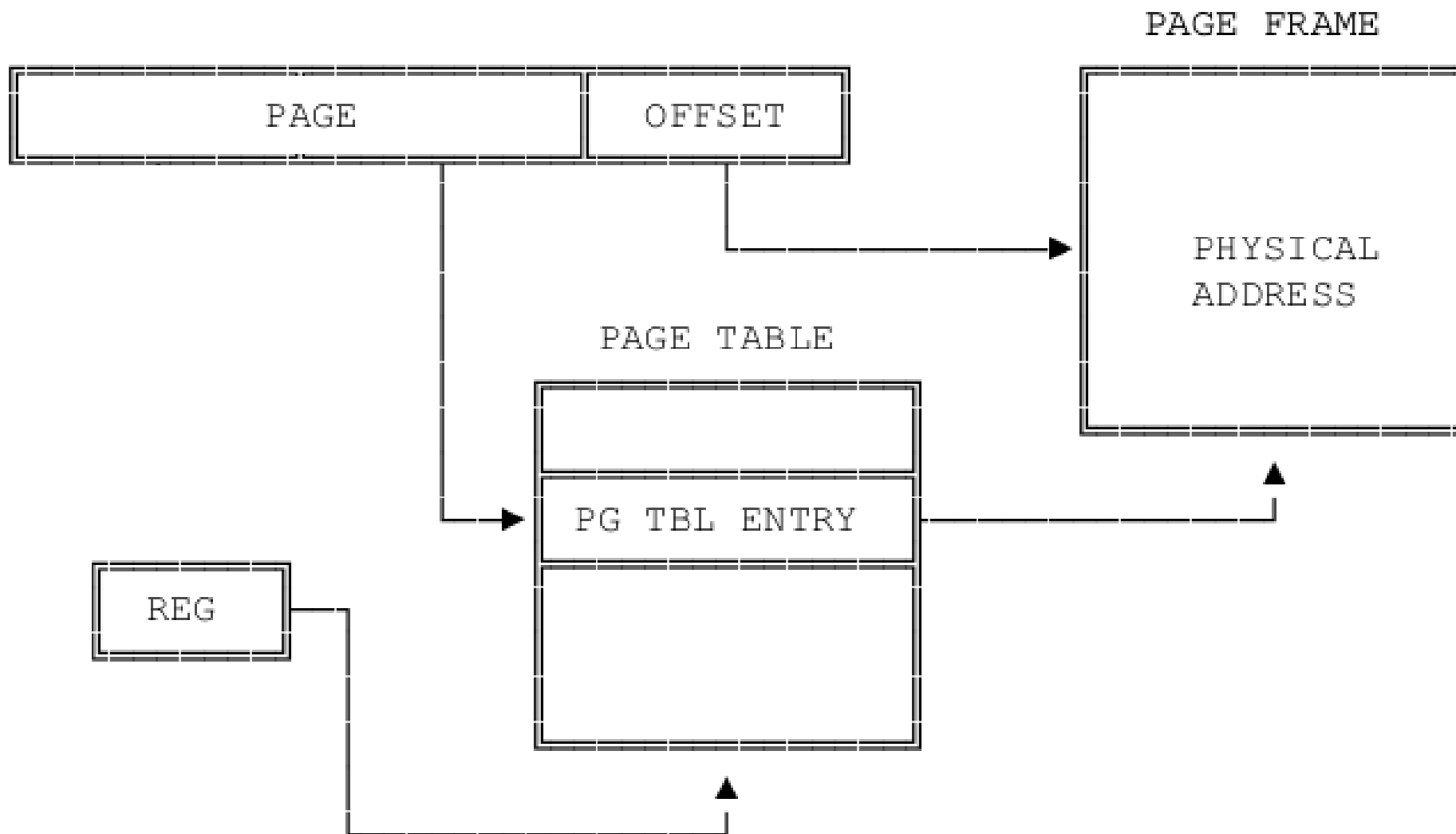
Strankovanie – co nam k tomu treba

- Ramec (page frame) versus Stranka (page)
- Linearna adresa
- Tabulka stranok
- Polozka tabulky stranok
- Ako sa robi preklad

Format linearnej adresy



Preklad Linearnej Adresy na Fyzicku



RISC-V

RISC-V

- Mapuje stranky o velkosti 4kB

RISC-V

- Mapuje stranky o velkosti 4kB
- Zarovnane na hranicu 4kB (0, 4k, 8k, 12k, ...)

RISC-V

- Mapuje stranky o velkosti 4kB
- Zarovnane na hranicu 4kB (0, 4k, 8k, 12k, ...)
- Kolko bitov potrebujeme na indexovanie (adresovanie) v ramci jednej stranky 4kB?

RISC-V

- Mapuje stranky o velkosti 4kB
- Zarovnane na hranicu 4kB (0, 4k, 8k, 12k, ...)
- Kolko bitov potrebujeme na indexovanie (adresovanie) v ramci jednej stranky 4kB? 12

RISC-V

- Mapuje stranky o velkosti 4kB
- Zarovnanane na hranicu 4kB (0, 4k, 8k, 12k, ...)
- Kolko bitov potrebujeme na indexovanie (adresovanie) v ramci jednej stranky 4kB? 12
- Xv6 vyuziva RISC-V v 64-bitovom adresnom rezime

RISC-V

- Mapuje stranky o velkosti 4kB
- Zarovnane na hranicu 4kB (0, 4k, 8k, 12k, ...)
- Kolko bitov potrebujeme na indexovanie (adresovanie) v ramci jednej stranky 4kB? 12
- Xv6 vyuziva RISC-V v 64-bitovom adresnom rezime
 - Na index do PT sa vyuziva $64-12 = 52$ bitov z VA

RISC-V

- Mapuje stranky o velkosti 4kB
- Zarovnane na hranicu 4kB (0, 4k, 8k, 12k, ...)
- Kolko bitov potrebujeme na indexovanie (adresovanie) v ramci jednej stranky 4kB? 12
- Xv6 vyuziva RISC-V v 64-bitovom adresnom rezime
 - Na index do PT sa vyuziva $64-12 = 52$ bitov z VA
 - Nie tak celkom, hornych 25 bitov z tychto 52 sa nevyuziva, takže index ma 27 bitov

RISC-V

- Mame 27 bitov na index, kolko indexov je to spolu?

RISC-V

- Mame 27 bitov na index, kolko indexov je to spolu? 128M

RISC-V

- Mame 27 bitov na index, kolko indexov je to spolu? 128M
- Kazda polozka tabulky ma 64b, co je kolko B?

RISC-V

- Mame 27 bitov na index, kolko indexov je to spolu? 128M
- Kazda polozka tabulky ma 64b, co je kolko B? 8B

RISC-V

- Mame 27 bitov na index, kolko indexov je to spolu? 128M
- Kazda polozka tabulky ma 64b, co je kolko B? 8B
- Kolko teda bude zaberat cela PT v pamati?

RISC-V

- Mame 27 bitov na index, kolko indexov je to spolu? 128M
- Kazda polozka tabulky ma 64b, co je kolko B? 8B
- Kolko teda bude zaberat cela PT v pamati?
 $128M * 8B = 1GB$;)

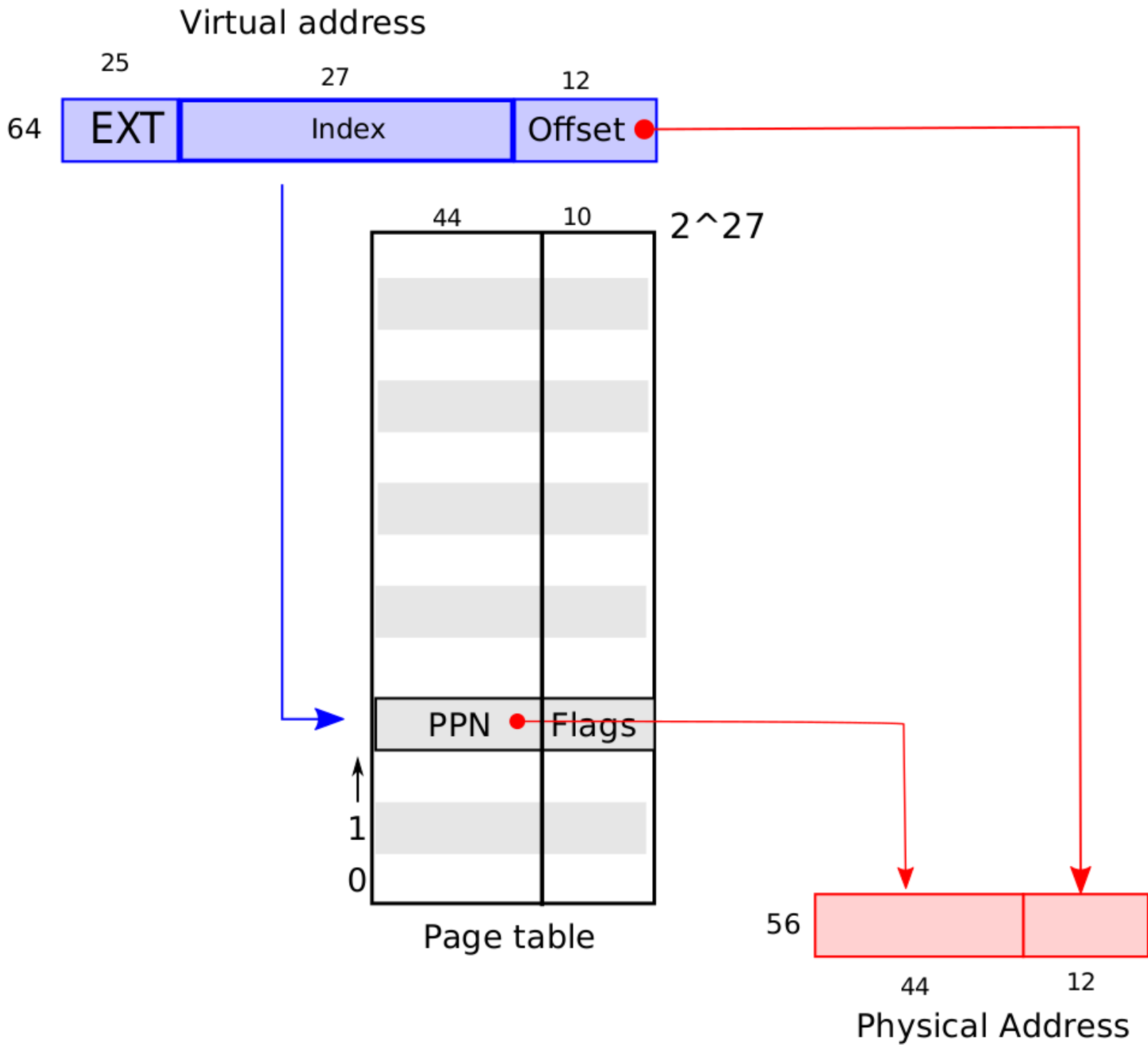
RISC-V

- Mame 27 bitov na index, kolko indexov je to spolu? 128M
- Kazda polozka tabulky ma 64b, co je kolko B? 8B
- Kolko teda bude zaberat cela PT v pamati? $128M * 8B = 1GB$;)
- Ak PT pre 1 proces zabera 1GB v RAM, kolko procesov asi tak moze bezat naraz? :D :D :D

RISC-V

- Mame 27 bitov na index, kolko indexov je to spolu? 128M
- Kazda polozka tabulky ma 64b, co je kolko B? 8B
- Kolko teda bude zaberat cela PT v pamati? $128M * 8B = 1GB$;)
- Ak PT pre 1 proces zabera 1GB v RAM, kolko procesov asi tak moze bezat naraz? :D :D :D
- Xv6 ma 128MB RAM... Tak tu nieco nesedi...

Preklad MMU



Preklad MMU

- Zaznam v tabulke PT sa nazyva PTE (Page Table Entry)

Preklad MMU

- Zaznam v tabulke PT sa nazyva PTE (Page Table Entry)
 - Ma 64b (8B), ale vyuziva sa “iba” 54

Preklad MMU

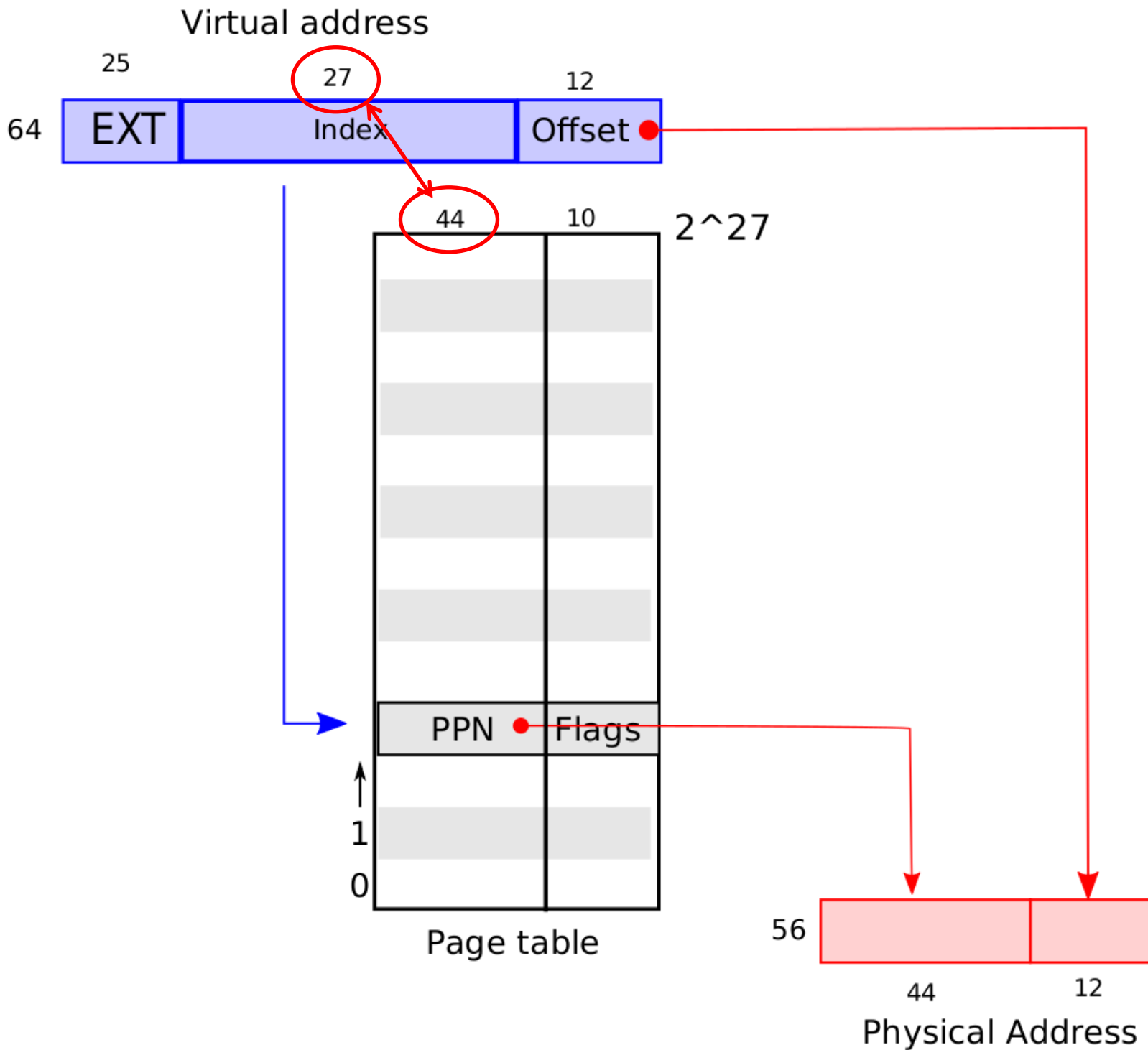
- Zaznam v tabulke PT sa nazyva PTE (Page Table Entry)
 - Ma 64b (8B), ale vyuziva sa “iba” 54
 - Hornych 44 bitov PTE tvoria hornych 44 bitov fyzickej adresy (Physical Page Number = PPN)

Preklad MMU

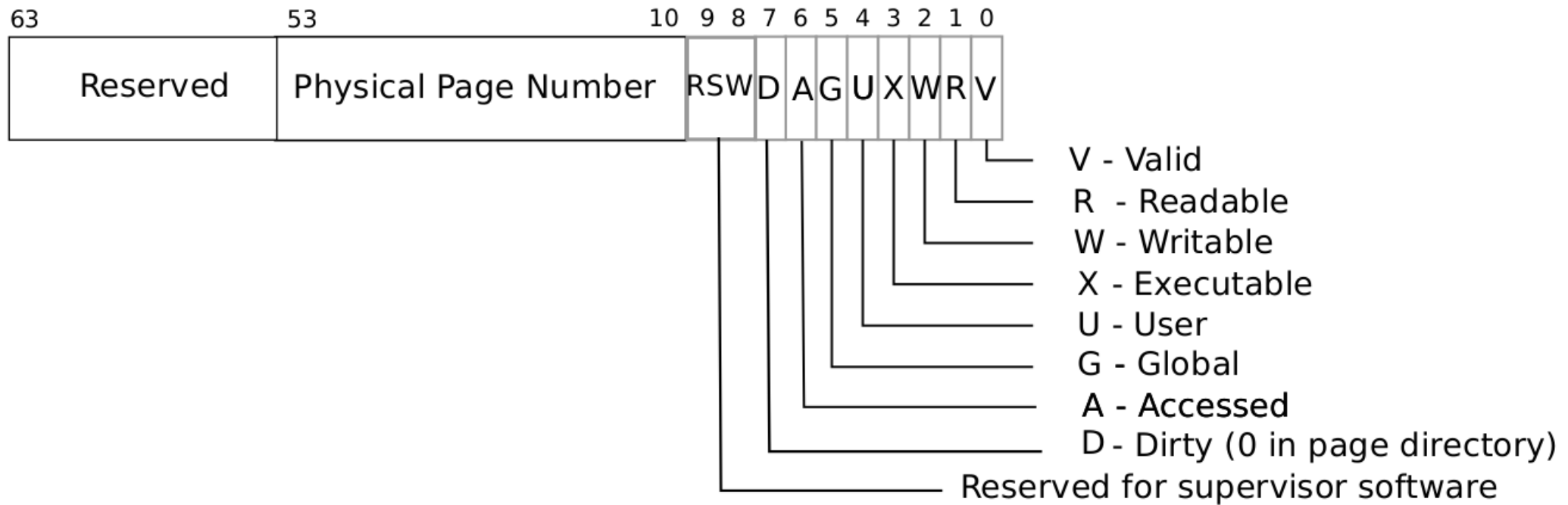
- Zaznam v tabulke PT sa nazyva PTE (Page Table Entry)
 - Ma 64b (8B), ale vyuziva sa “iba” 54
 - Hornych 44 bitov PTE tvoria hornych 44 bitov fyzickej adresy (Physical Page Number = PPN)
 - Spodnych 10 su tzv. priznaky
 - Present, Writeable, User, ...

Preklad MMU

- Zaznam v tabulke PT sa nazyva PTE (Page Table Entry)
 - Ma 64b (8B), ale vyuziva sa “iba” 54
 - Hornych 44 bitov PTE tvoria hornych 44 bitov fyzickej adresy (Physical Page Number = PPN)
 - Spodnych 10 su tzv. priznaky
 - Present, Writeable, User, ...
- **!POZOR! velkost VA != velkost PA !ROZOP!**



PTE



PT

PT

- Kde je PT uložena?

PT

- Kde je PT uložena? V RAM

PT

- Kde je PT uložena? V RAM
- MMU dokaze manipulovat so zaznamami PTE
- Podobne to dokaze aj OS

PT

- Ako sme uz vypocitali, velkost PT je 1GB!
 - 1 PT na 1 adresny priestor
 - 1 adresny priestor pre 1 aplikaciju (proces)

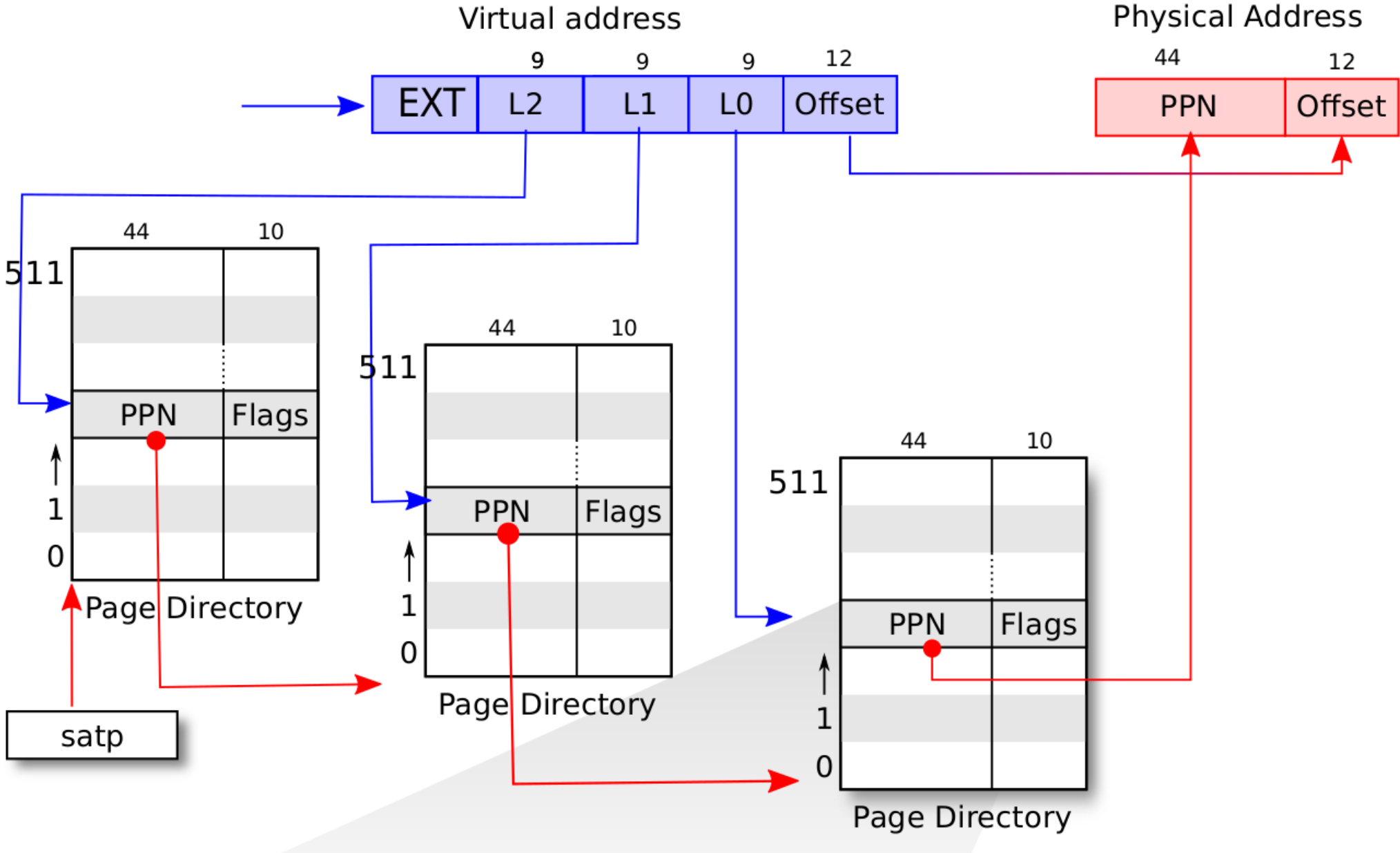
PT

- Ako sme uz vypocitali, velkost PT je 1GB!
 - 1 PT na 1 adresny priestor
 - 1 adresny priestor pre 1 aplikaciu (proces)
- Avsak proces zvacsa potrebuje iba par k/MB pamate, takže iba par k/M zaznamov PTE bude vyuzitych, ostatne budu prazdne!

PT

- RISC-V 64 vyuziva na usetrenie miesta tzv. “troj urovnovu” tabulku stranok

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 \rightarrow 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); taketo PTE “nejestvujú” (nejestvuje prepojenie t.j. mapovanie s RAM)
- Preto môže byť PT pre proces mála!

PT

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

PT

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

PT

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

PT

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

PT

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

PT

- Ako MMU “vie”, kde v RAM sa PT nachadza?
- Na RISC-V je **FYZICKA** adresa hornej casti PT (pre index L2) v registri *satp*
- Prepisaním *satp* sa prepínajú adresne priestory!

PT

- Ako MMU “vie”, kde v RAM sa PT nachadza?
- Na RISC-V je **FYZICKA** adresa hornej casti PT (pre index L2) v registri *satp*
- Prepisaním *satp* sa prepínajú adresne priestory!
- Stranky PT môžu byť voľne roztrúsené v RAM-ke, nemusí ísť o súvislú oblasť RAM!!!

Preklad va2pa

Preklad va2pa

- MMU musi najst spravny zaznam PTE zodpovedajuci danej VA; ako?

Preklad va2pa

- MMU musi najst spravny zaznam PTE zodpovedajuci danej VA; ako?
 - Z registra satp vieme PA pre obsah tabulky PD_{L2}

Preklad va2pa

- MMU musi najst spravny zaznam PTE zodpovedajuci danej VA; ako?
 - Z registra satp vieme PA pre obsah tabulky PD_{L2}
 - Hornych 9 bitov indexu VA (L2) ukazuje do PD_{L2} ; z $PD_{L2}[L2]$ ziskame PA pre obsah PD_{L1}

Preklad va2pa

- MMU musi najst spravny zaznam PTE zodpovedajuci danej VA; ako?
 - Z registra satp vieme PA pre obsah tabulky PD_{L2}
 - Hornych 9 bitov indexu VA (L2) ukazuje do PD_{L2} ; z $PD_{L2}[L2]$ ziskame PA pre obsah PD_{L1}
 - Dalsich 9 bitov indexu VA (L1) ukazuje do PD_{L1} ; z $PD_{L1}[L1]$ ziskame PA pre obsah PD_{L0}

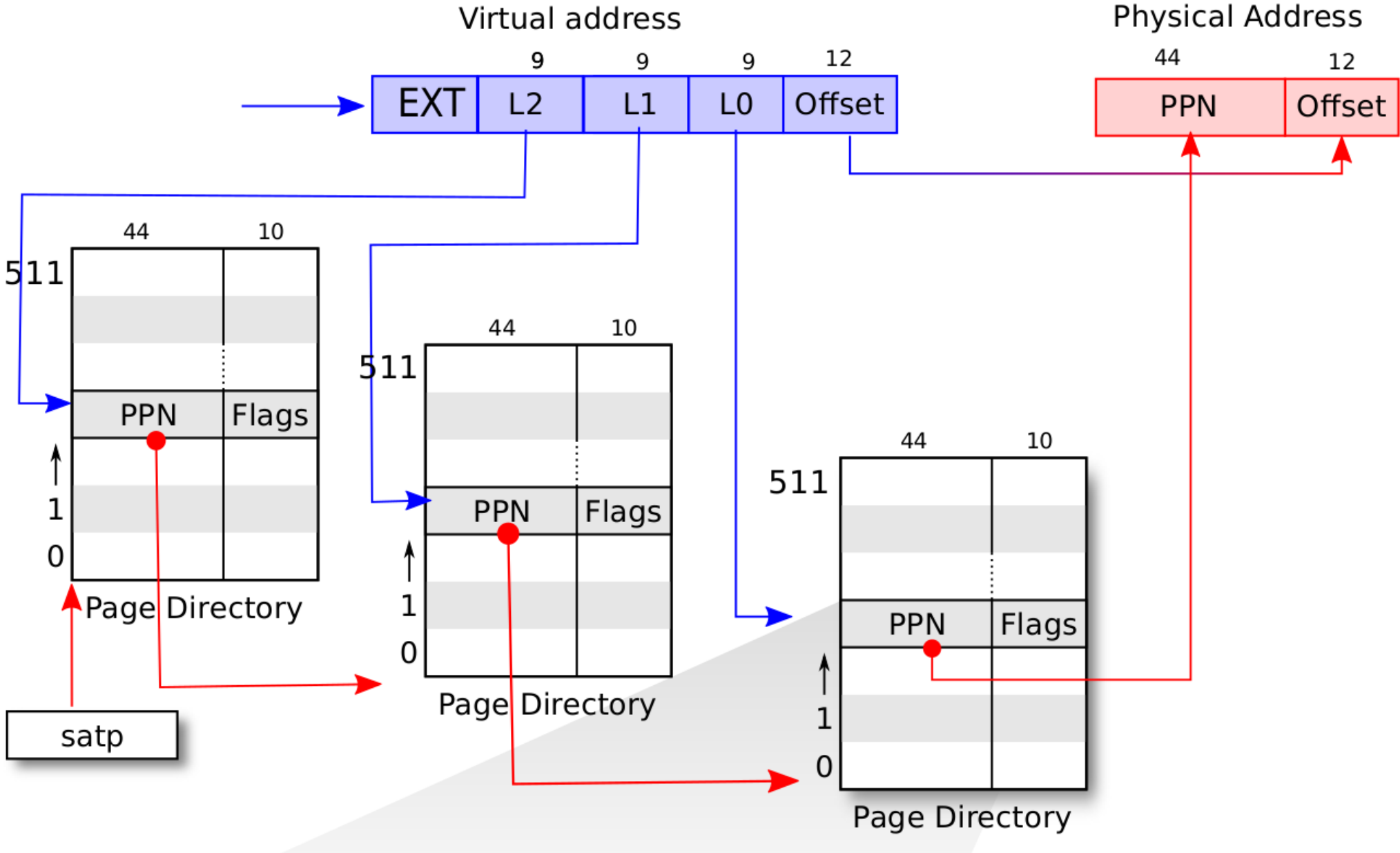
Preklad va2pa

- MMU musi najst spravny zaznam PTE zodpovedajuci danej VA; ako?
 - Z registra satp vieme PA pre obsah tabulky PD_{L2}
 - Hornych 9 bitov indexu VA (L2) ukazuje do PD_{L2} ; z $PD_{L2}[L2]$ ziskame PA pre obsah PD_{L1}
 - Dalsich 9 bitov indexu VA (L1) ukazuje do PD_{L1} ; z $PD_{L1}[L1]$ ziskame PA pre obsah PD_{L0}
 - Poslednych 9 bitov indexu VA (L0) ukazuje do PD_{L0} ; z $PD_{L0}[L0]$ ziskame PA pre PTE

Preklad va2pa

- MMU musi najst spravny zaznam PTE zodpovedajuci danej VA; ako?
 - Z registra satp vieme PA pre obsah tabulky PD_{L2}
 - Hornych 9 bitov indexu VA (L2) ukazuje do PD_{L2} ; z $PD_{L2}[L2]$ ziskame PA pre obsah PD_{L1}
 - Dalsich 9 bitov indexu VA (L1) ukazuje do PD_{L1} ; z $PD_{L1}[L1]$ ziskame PA pre obsah PD_{L0}
 - Poslednych 9 bitov indexu VA (L0) ukazuje do PD_{L0} ; z $PD_{L0}[L0]$ ziskame PA pre PTE
 - $PA = PPN$ z PTE plus spodnych 12 bitov VA

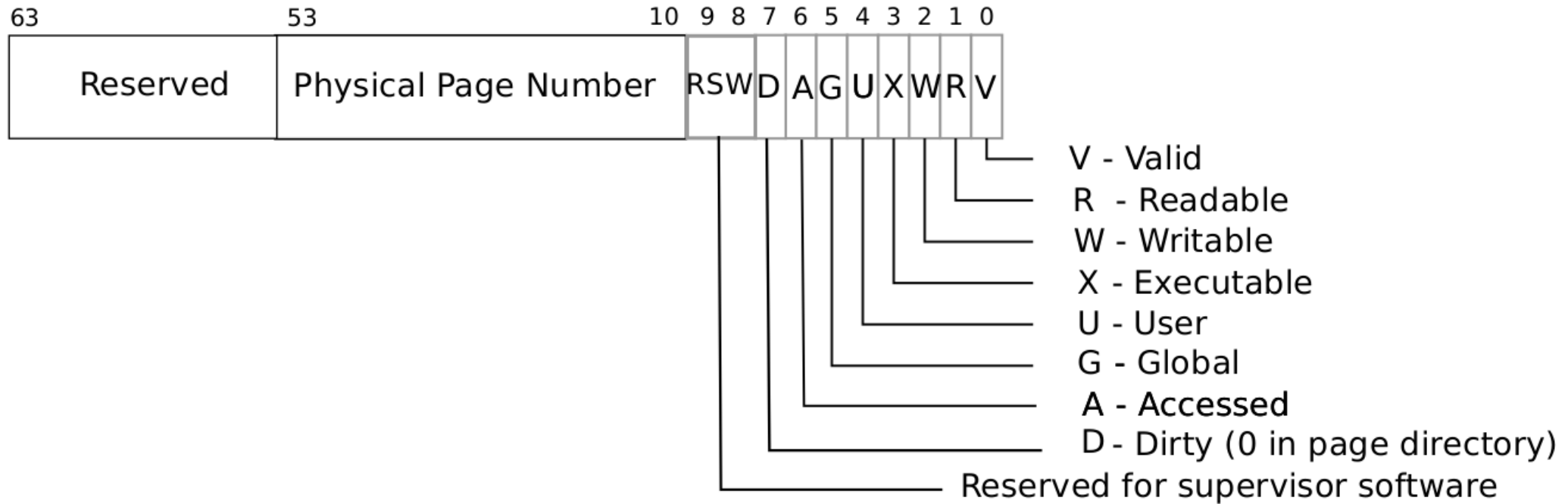
PT RISC-V



Priznaky PTE

- Xv6 vyuziva V, R, W, X, U

Priznaky PTE



Priznaky PTE

Priznaky PTE

- Co ak bit V nie je nastaveny? Alebo zapisujeme a nie je nastaveny W?

Priznaky PTE

- Co ak bit V nie je nastaveny? Alebo zapisujeme a nie je nastaveny W?
- Vypadok stranky

Priznaky PTE

- Co ak bit V nie je nastaveny? Alebo zapisujeme a nie je nastaveny W?
- Vypadok stranky
 - Vynuteny presun do jadra (trap.c)

Priznaky PTE

- Co ak bit V nie je nastaveny? Alebo zapisujeme a nie je nastaveny W?
- Vypadok stranky
 - Vynuteny presun do jadra (trap.c)
 - Jadro bud vypise chybovu spravu a ukonci proces, ktory chybu sposobil (“usertrap(): unexpected scause...”)

Priznaky PTE

- Co ak bit V nie je nastaveny? Alebo zapisujeme a nie je nastaveny W?
- Vypadok stranky
 - Vynuteny presun do jadra (trap.c)
 - Jadro bud vypise chybovu spravu a ukonci proces, ktory chybu sposobil (“usertrap(): unexpected scause...”)
 - Alebo moze nainstalovat chybajuci PTE a obnovit beh procesu (napr. ak sa pouziva swapovanie pamate RAM na disk)

Vyhody strankovania

Vyhody strankovania

- Spojity virtualny adresny priestor nevyzaduje spojity fyzicky adresny priestor! (vobec neprichadza ku externej fragmentacii!!!)

Vyhody strankovania

- Spojity virtualny adresny priestor nevyzaduje spojity fyzicky adresny priestor! (vobec neprichadza ku externej fragmentacii!!!)
- “lazy allocation”; alokacia pamate az pri jej prvom pouziti (nastane vypadok, jadro alokuje PTE a proces restartuje instrukciu)

Vyhody strankovania

- Spojity virtualny adresny priestor nevyzaduje spojity fyzicky adresny priestor! (vobec neprichadza ku externej fragmentacii!!!)
- “lazy allocation”; alokacia pamate az pri jej prvom pouziti (nastane vypadok, jadro alokuje PTE a proces restartuje instrukciu)
- “copy-on-write fork”; kopia stranky az pri prvom pokuse o zapis

Virtualna pamat v jadre

Virtualna pamat v jadre

- Pre procesy je vyuzitie VM v poriadku, ale preco ju vyuzivat aj v jadre?

Virtualna pamat v jadre

- Pre procesy je vyuzitie VM v poriadku, ale preco ju vyuzivat aj v jadre?
- Moze bezat jadro iba s fyzickou pamatou?

Virtualna pamat v jadre

- Pre procesy je vyuzitie VM v poriadku, ale preco ju vyuzivat aj v jadre?
- Moze bezat jadro iba s fyzickou pamatou?
ANO, moze!

Virtualna pamat v jadre

- Pre procesy je vyuzitie VM v poriadku, ale preco ju vyuzivat aj v jadre?
- Moze bezat jadro iba s fyzickou pamatou?
ANO, moze!
- Vaccsina jadier OS vsak vyuziva VA; preco?

Virtualna pamat v jadre

- Pre procesy je vyuzitie VM v poriadku, ale preco ju vyuzivat aj v jadre?
- Moze bezat jadro iba s fyzickou pamatou?
ANO, moze!
- Vaccsina jadier OS vsak vyuziva VA; preco?
 - Je zlozite vypinat/zapinat strankovanie

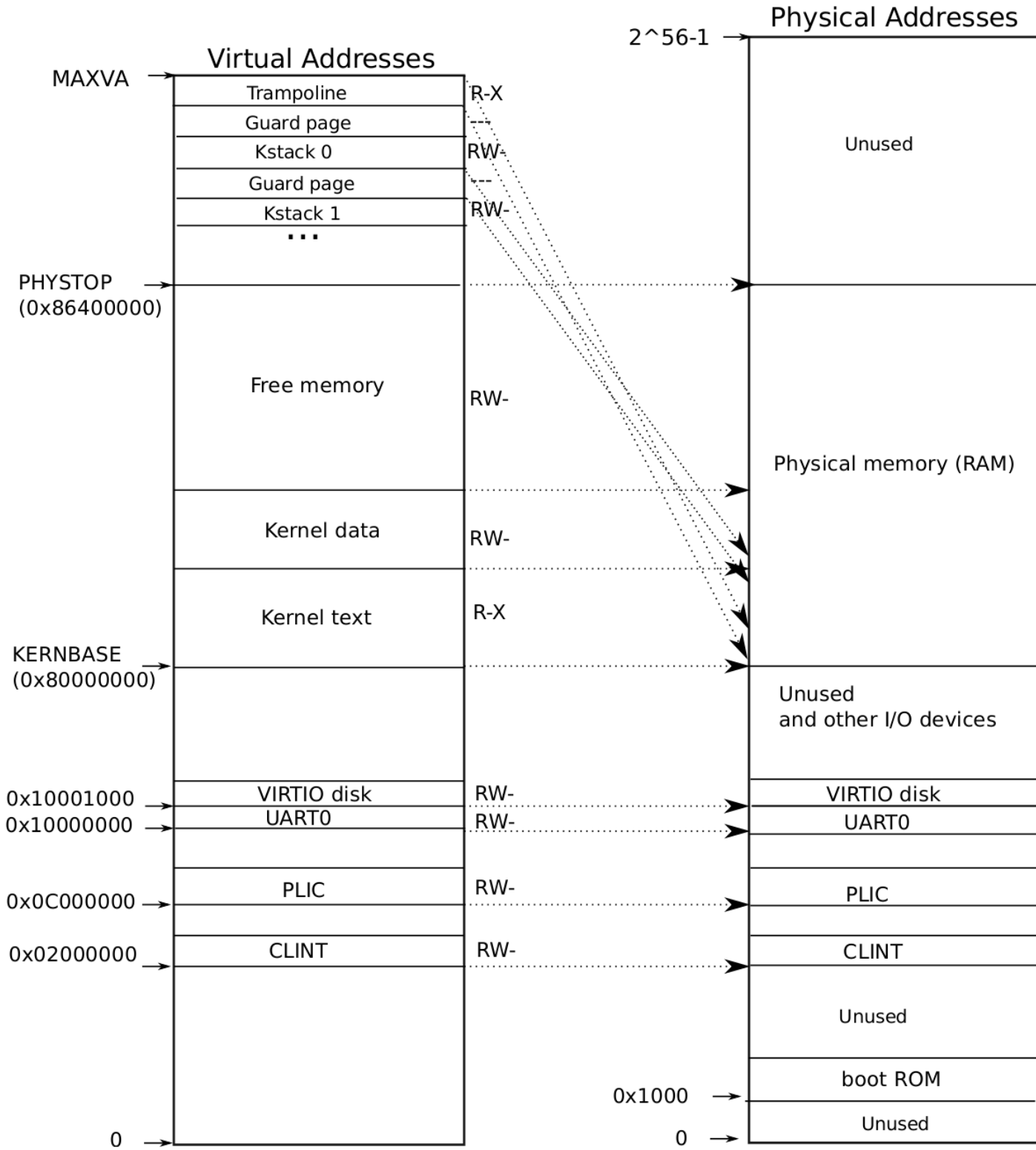
Virtualna pamat v jadre

- Pre procesy je vyuzitie VM v poriadku, ale preco ju vyuzivat aj v jadre?
- Moze bezat jadro iba s fyzickou pamatou?
ANO, moze!
- Vaccina jadier OS vsak vyuziva VA; preco?
 - Je zlozite vypinat/zapinat strankovanie
 - Ulahcuje to hladanie chyb
 - Text jadra oznacime X, udaje nie
 - Ponechanie “diery” pod zasobnikom

Virtualna pamat v jadre

- Pre procesy je vyuzitie VM v poriadku, ale preco ju vyuzivat aj v jadre?
- Moze bezat jadro iba s fyzickou pamatou?
ANO, moze!
- Vaccsina jadier OS vsak vyuziva VA; preco?
 - Je zlozite vypinat/zapinat strankovanie
 - Ulahcuje to hladanie chyb
 - Text jadra oznacime X, udaje nie
 - Ponechanie “diery” pod zasobnikom
 - Ulahcuje prechod medzi user/kernel (tym istym mapovanim tej istej stranky – vid trampoline)

Virtualna pamat JADRA xv6



Virtualna pamat JADRA xv6

- Jednoduché mapovanie virtualnej pamate na fyzicku jedna-k-jednej

Virtualna pamat JADRA xv6

- Jednoduche mapovanie virtualnej pamate na fyzicku jedna-k-jednej
- Preto sa mapuju aj zariadenia?

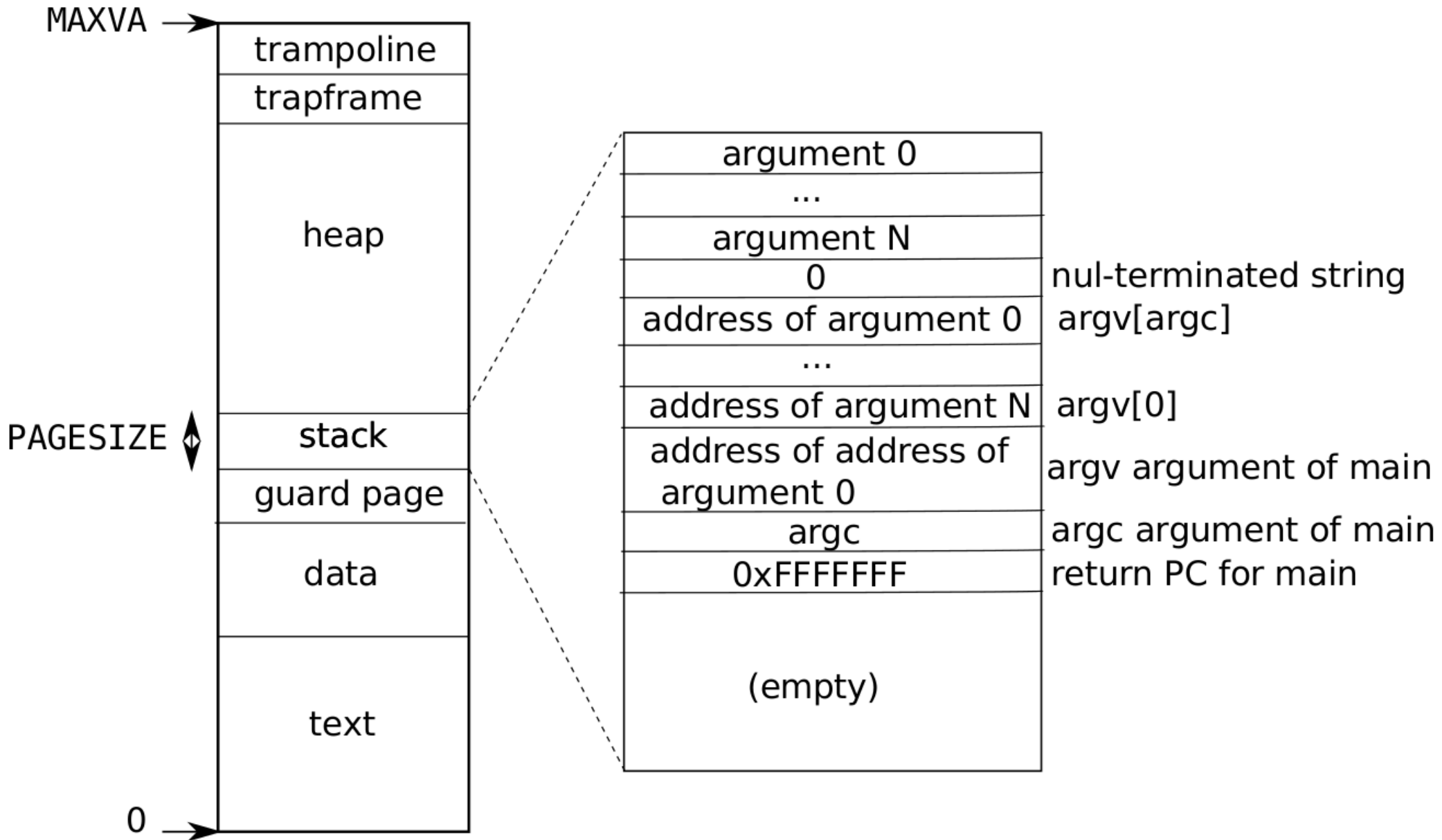
Virtualna pamat JADRA xv6

- Jednoduche mapovanie virtualnej pamate na fyzicku jedna-k-jednej
- Preto sa mapuju aj zariadenia?
- Vid opravenia roznych oblasti...

Virtualna pamat UZIVATELA xv6

- Kazdy proces ma vlastny adresny priestor
- Vlastnu tabulku stranok
- Vid trampoline a trapframe – nie su zapisovatelne pre uzivatelsky proces!!!
- Jadro OS nastavuje register satp pri prepinaní procesov (usertrapret() kernel/trap.c:122)

Virtualna pamat UZIVATELA xv6



Virtualna pamat xv6

- Vlastnosti takejto organizacie adresneho priestoru

Virtualna pamat xv6

- Vlastnosti takejto organizacie adresneho priestoru
 - VA uzivatela zacina od 0 (ovsem v kazdom procese je VA uzivatela od 0 mapovana na inu RAM – vo vseobecnosti)

Virtualna pamat xv6

- Vlastnosti takejto organizacije adresneho priestoru
 - VA uzivatela zacina od 0 (ovsem v kazdom procese je VA uzivatela od 0 mapovana na inu RAM – vo vseobecnosti)
 - 16 777 216 GB halda uzivatela ;)

Virtualna pamat xv6

- Vlastnosti takejto organizacie adresneho priestoru
 - VA uzivatela zacina od 0 (ovsem v kazdom procese je VA uzivatela od 0 mapovana na inu RAM – vo vseobecnosti)
 - 16 777 216 GB halda uzivatela ;)
 - Jednoduchy prechod user \leftrightarrow kernel mapovanim trampoliny a TF

Virtualna pamat xv6

- Vlastnosti takejto organizacie adresneho priestoru
 - VA uzivatela zacina od 0 (ovsem v kazdom procese je VA uzivatela od 0 mapovana na inu RAM – vo vseobecnosti)
 - 16 777 216 GB halda uzivatela ;)
 - Jednoduchy prechod user \leftrightarrow kernel mapovanim trampoliny a TF
 - Nelahky pristup jadra k pamati uzivatela!!!

Virtualna pamat xv6

- Vlastnosti takejto organizacie adresneho priestoru
 - VA uzivatela zacina od 0 (ovsem v kazdom procese je VA uzivatela od 0 mapovana na inu RAM – vo vseobecnosti)
 - 16 777 216 GB halda uzivatela ;)
 - Jednoduchy prechod user \leftrightarrow kernel mapovanim trampoliny a TF
 - Nelahky pristup jadra k pamati uzivatela!!!
 - Lahky pristup jadra k fyzickej pamati: $pa(x)$ mapovana na $va(x)$

Kod VM xv6

Kod VM xv6

- Inicializacia adresneho priestoru jadra

Kod VM xv6

- Inicializacia adresneho priestoru jadra
- Kernel/memlayout.h

Kod VM xv6

- Inicializacia adresneho priestoru jadra
- Kernel/memlayout.h a kernel/vm.c kvminit()

Kod VM xv6

- Inicializacia adresneho priestoru jadra
- Kernel/memlayout.h a kernel/vm.c kvminit()
 - Kolko adresneho priestoru vie obsiahnut 1 L0 polozka?
 - Kolko 1 L1 polozka?
 - Kolko 1 L2 polozka?

Kod VM xv6

- Inicializacia adresneho priestoru jadra
- Kernel/memlayout.h a kernel/vm.c kvm_init()
 - Kolko adresneho priestoru vie obsiahnut 1 L0 položka? 4kB
 - Kolko 1 L1 položka? 2MB
 - Kolko 1 L2 položka? 1GB

Kod VM xv6

- Inicializacia adresneho priestoru jadra
- Kernel/memlayout.h a kernel/vm.c kvminit()
 - Kolko adresneho priestoru vie obsiahnut 1 L0 položka? 4kB
 - Kolko 1 L1 položka? 2MB
 - Kolko 1 L2 položka? 1GB
 - Aky je velky cely adresny priestor?

Kod VM xv6

- Inicializacia adresneho priestoru jadra
- Kernel/memlayout.h a kernel/vm.c kvminit()
 - Kolko adresneho priestoru vie obsiahnut 1 L0 položka? 4kB
 - Kolko 1 L1 položka? 2MB
 - Kolko 1 L2 položka? 1GB
 - Aky je velky cely adresny priestor? 512GB

Kod VM xv6

- Kolko pamate (stranok) je pouzitych na reprezentáciu PT (t.j. mapovania adresneho priestoru) po prvom volani `kvmmap()`?
 - `Kernel/vm.c:30 kvmmap(UART0, UART0, PGSIZE, PTE_R | PTE_W);`

Kod VM xv6

- Kolko pamate (stranok) je pouzitych na reprezentáciu PT (t.j. mapovania adresneho priestoru) po prvom volani `kvmmap()`?
 - `Kernel/vm.c:30 kvmmap(UART0, UART0, PGSIZE, PTE_R | PTE_W);`
- Kolko pamate (stranok) sa tymto volanim mapuje?

Kod VM xv6

- Kolko pamate (stranok) je pouzitych na reprezentáciu PT (t.j. mapovania adresneho priestoru) po prvom volani `kvmmap()`?
 - `Kernel/vm.c:30 kvmmap(UART0, UART0, PGSIZE, PTE_R | PTE_W);`
- Kolko pamate (stranok) sa tymto volanim mapuje?
- Ukazka `vmprint()` za tymto prvym volanim

Kod VM xv6

- Kolko položiek v PT zaberá CLINT?
 - `kvmmap(CLINT, CLINT, 0x10000, ...);`
- Kolko položiek v PT zaberá PLIC?
 - `kvmmap(PLIC, PLIC, 0x400000, ...);`

Kod VM xv6

- Kolko položiek v PT zbera CLINT?
 - `kvmmap(CLINT, CLINT, 0x10000, ...);`
- Kolko položiek v PT zbera PLIC?
 - `kvmmap(PLIC, PLIC, 0x400000, ...);`
- Je trampolina mapovana 2x?

Kod VM xv6

- `Kvminithart()`
 - `w_satp()`
 - `sfence_vma()`
- Preco po nastaveni PT musi nasledovat instrukcia `sfence_vma()`? (`kernel/riscv.h`)

Kod VM xv6

- `Kvminithart()`
 - `w_satp()`
 - `sfence_vma()`
- Preto po nastaveni PT musi nasledovat instrukcia `sfence_vma()`? (`kernel/riscv.h`)
 - TLB (Translation Lookaside Buffer) – vyrovnávací paměť pro překlad $VA \rightarrow PA$

Kod VM xv6

- mappages()

Kod VM xv6

- mappages()
 - Argumenty: top PD, va, size, pa, perm

Kod VM xv6

- `mappages()`
 - Argumenty: `top PD, va, size, pa, perm`
 - Do PT zaznamenava mapovanie `<va; va+size)` na prislusny interval `<pa; pa+size)`

Kod VM xv6

- `mappages()`
 - Argumenty: top PD, va, size, pa, perm
 - Do PT zaznamenava mapovanie $\langle va; va+size \rangle$ na prislusny interval $\langle pa; pa+size \rangle$
 - Nakolko va ani size nemusia byt zarovnane na velkost stranky, musia sa zaokruhliť (PGSIZE!!!)

Kod VM xv6

- mappages()
 - Argumenty: top PD, va, size, pa, perm
 - Do PT zaznamenava mapovanie $\langle va; va+size \rangle$ na prislusny interval $\langle pa; pa+size \rangle$
 - Nakolko va ani size nemusia byt zarovnane na velkost stranky, musia sa zaokruhliť (PGSIZE!!!)
 - Algoritmus
 - Pre kazdu zarovnanu adresu va_addr:
 - Pomocou walk() najdi &PTE (nie iba obsah!) pre va_addr
 - Cielovu PA vlož do PTE
 - Označ PTE za validne (priznak PTE_V)

Kod VM xv6

- walk()

Kod VM xv6

- walk()
 - Napodobnuje cinnost MMU; pre danu VA a PT najde prislusny PTE zaznam v PT
 - Makro PX(level, va) extrahuje 9 bitov indexu na urovni level

Kod VM xv6

- walk()
 - Makro PX(level, va) extrahuje 9 bitov indexu na urovni level
 - Algoritmus
 - PTE_addr = &pagetable[PX(level, va)]
 - If is set PTE_V in *PTE_addr
 - Prislusna tabulka v PT jestvuje
 - PTE2PA extrahuje PPN zo zaznamu PTE
 - If not set PTE_V in *PTE_addr
 - Alokuj tabulku dalsej urovne
 - Vypln *PTE_addr s PPN alokovanej tabulky (PA2PTE)
 - Vrat adresu PTE z (vytvorenej/jestvujucej) tabulky L0

Kod VM xv6

- `procinit()` v `kernel/proc.c`

Kod VM xv6

- `procinit()` v `kernel/proc.c`
- Staticke pole procesov
- Kazdemu procesu alokuj v ramke stranku na zasobnik jadra (velkost `PGSIZE`) a namapuj do VA kernelu
- Kazdy proces ma vlastny zasobnik v jadre

Kod VM xv6

- `procinit()` v `kernel/proc.c`
- Staticke pole procesov
- Kazdemu procesu alokuj v ramke stranku na zasobnik jadra (velkost `PGSIZE`) a namapuj do VA kernelu
- Kazdy proces ma vlastny zasobnik v jadre
- Kazdy zasobnik ma “guard page”!!! (obrazok)

Kod VM xv6

- Inicializacia uzivatelskeho adresneho priestoru

Kod VM xv6

- Inicializacia uzivatelskeho adresneho priestoru
 - Allocproc() v kernel/proc.c
 - Fork() v kernel/proc.c
 - Exec() v kernel/exec.c

Kod VM xv6

- Inicializacia uzivatelskeho adresneho priestoru
 - Allocproc() alokuje prazdnu PT najvyssieho stupna
 - Fork() uvmcopy()
 - Exec() prepise PT procesu novou
 - Uvmalloc()
 - Loadseg()
- Ukazka vmprint() pre procesy init a sh

Kod VM xv6

- Ak chce proces (uzivatelsky) viac pamate (alokacia z haldy), vyvola systemove volanie `sbrk(n)`; o `n` sa zvacsi pamat procesu
 - Vid `user/umalloc.c` volanie `sbrk()`

Kod VM xv6

- Ak chce proces (uzivatelsky) viac pamate (alokacia z haldy), vyvola systemove volanie `sbrk(n)`; o `n` sa zvacsi pamat procesu
 - Vid `user/umalloc.c` volanie `sbrk()`
- Kazdy proces ma svoju velkost; volanie `sbrk()` pridava procesu na konci pamat, zvacsuje velkost procesu

Kod VM xv6

- Ak chce proces (uzivatelsky) viac pamate (alokacia z haldy), vyvola systemove volanie `sbrk(n)`; o `n` sa zvacsi pamat procesu
 - Vid `user/umalloc.c` volanie `sbrk()`
- Kazdy proces ma svoju velkost; volanie `sbrk()` pridava procesu na konci pamat, zvacsuje velkost procesu (`kernel/sysproc.c`)
 - Alokuje fyzicku pamat (RAM)
 - Mapuje ju do PT procesu
 - Vracia pociatocnu adresu tejto novej pamate

Kod VM xv6

- `Growproc()` v `kernel/proc.c`
 - `proc→sz` je aktualna velkost procesu
 - `uvmalloc()` obsahuje glavnu funkcionalitu
 - Pri prepnuti z jadra do user space sa do `satp` ulozi adresa aktualizovanej PT

Kod VM xv6

- `Growproc()` v `kernel/proc.c`
 - `proc→sz` je aktualna velkost procesu
 - `uvmalloc()` obsahuje glavnu funkcionalitu
 - Pri prepnuti z jadra do user space sa do `satp` ulozi adresa aktualizovanej PT

- `Uvmalloc()` v `kernel/vm.c`
 - Preco je tam `PGROUNDUP`?
 - Preco `mappages(..., PTE_W|PTE_X|PTE_R|PTE_U)`?

