# OS 2019

MIT ;)

https://pdos.csail.mit.edu/6.828/2019

# 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 izolaciu medzi nimi?**

# Izolacia adresneho priestoru

- Odpoved: strankovaci hardver

# Izolacia adresneho priestoru

- Odpoved: strankovaci hardver

- Xv6 vyuziva hardver procesora RISC-V

# Strankovanie

# Strankovanie

- Poskytuje nepriamu adresaciu

# Strankovanie

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

# Strankovanie

- Poskytuje nepriamu adresaciu
  - program pouziva ukazatele (adresy); tieto adresy vyuziva CPU na pristup k pamati
    - Nie su to vsak (vo vseobecnosti) adresy operandov vo fyzickej pamati!
    - Ide o tzv. Virtualne adresy!

# Strankovanie

- Poskytuje nepriamu adresaciu
  - program pouziva ukazatele (adresy); tieto adresy vyuziva CPU na pristup k pamati
    - Nie su to vsak (vo vseobecnosti) adresy operandov vo fyzickej pamati!
    - Ide o tzv. Virtualne adresy!
  - Hardver spravy pamate (MMU) “preklada” virtualne adresy na fyzicke adresy

# Strankovanie

- Poskytuje nepriamu adresaciu
  - program pouziva ukazatele (adresy); tieto adresy vyuziva CPU na pristup k pamati
    - Nie su to vsak (vo vseobecnosti) adresy operandov vo fyzickej pamati!
    - Ide o tzv. Virtualne adresy!
  - Hardver spravy pamate (MMU) “preklada” virtualne adresy na fyzicke adresy
    - V literature sa casto spomina aj termin “linearna adresa” (vid rozdiel “linearna” a “virtualna”)
    - Az fyzicka adresa je adresou operandu v RAM

# Strankovanie

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

# Strankovanie

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

- Jadro OS ma za ulohu riadit mapovanie kazdej VA na PA (nastavit udaje 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

# 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
- 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

# 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, takze 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
- Ked mame 64-bitove fyzicke adresy, kazda polozka tabulky musi mat aspon 64b, co je kolko B?

# RISC-V

- Mame 27 bitov na index, kolko indexov je to spolu? 128M
- Ked mame 64-bitove fyzicke adresy, kazda polozka tabulky musi mat aspon 64b, co je kolko B? 8B

# RISC-V

- Mame 27 bitov na index, kolko indexov je to spolu? 128M
- Ked mame 64-bitove fyzicke adresy, kazda polozka tabulky musi mat aspon 64b, co je kolko B? 8B
- Kolko teda bude zaberat cela PT v pamati?

# RISC-V

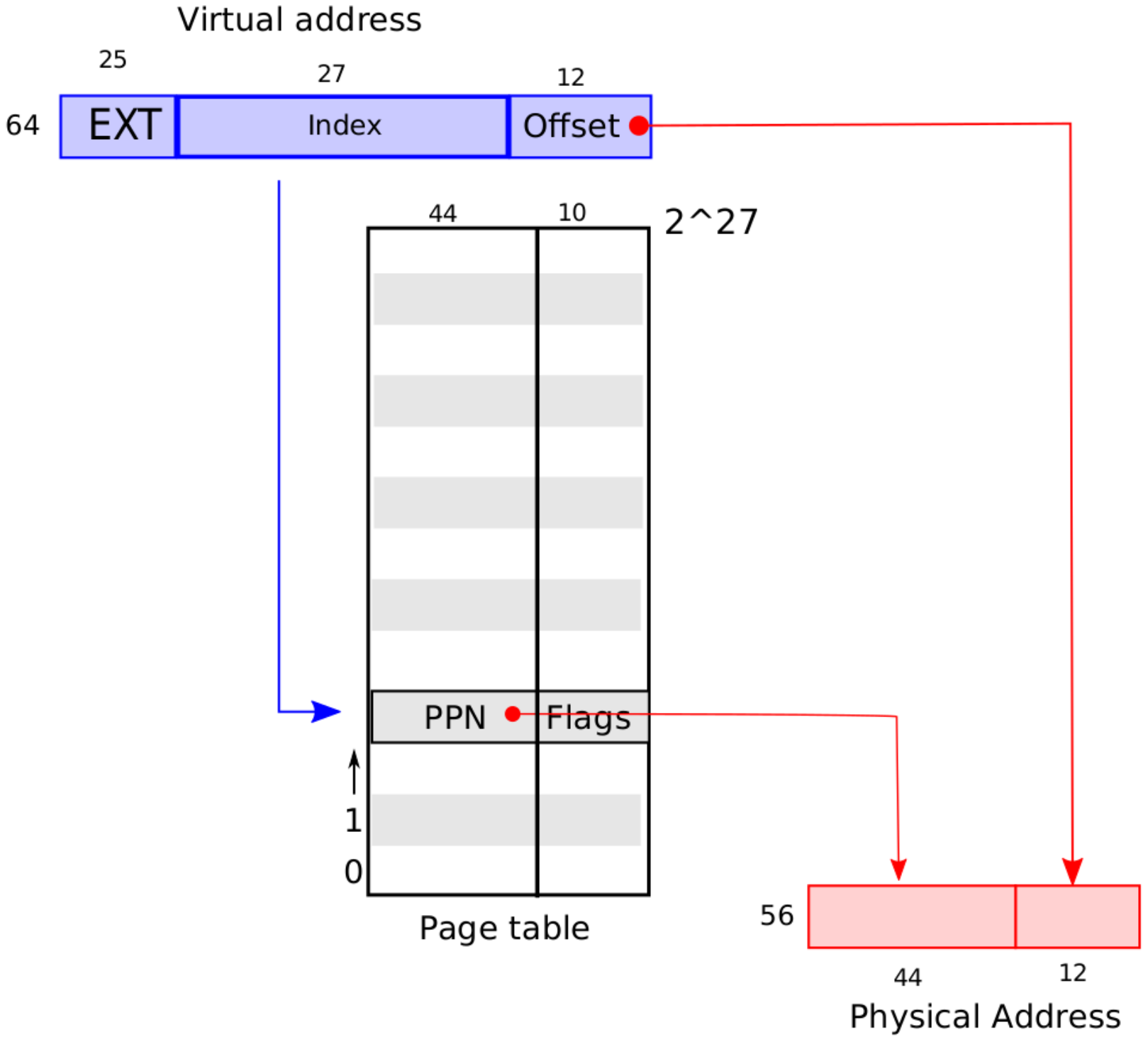
- Mame 27 bitov na index, kolko indexov je to spolu? 128M
- Ked mame 64-bitove fyzicke adresy, kazda polozka tabulky musi mat aspon 64b, co je kolko B? 8B
- Kolko teda bude zaberat cela PT v pamati? 128M*8B=1GB ;)

# RISC-V

- Mame 27 bitov na index, kolko indexov je to spolu? 128M
- Ked mame 64-bitove fyzicke adresy, kazda polozka tabulky musi mat aspon 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

# Preklad MMU


Virtual address
25
27
12
64
EXT
Index
Offset
44
10
2^27
PPN
Flags
1
0
Page table
56
44
12
Physical Address

# 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 na adresu

# Preklad MMU

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

# Preklad MMU

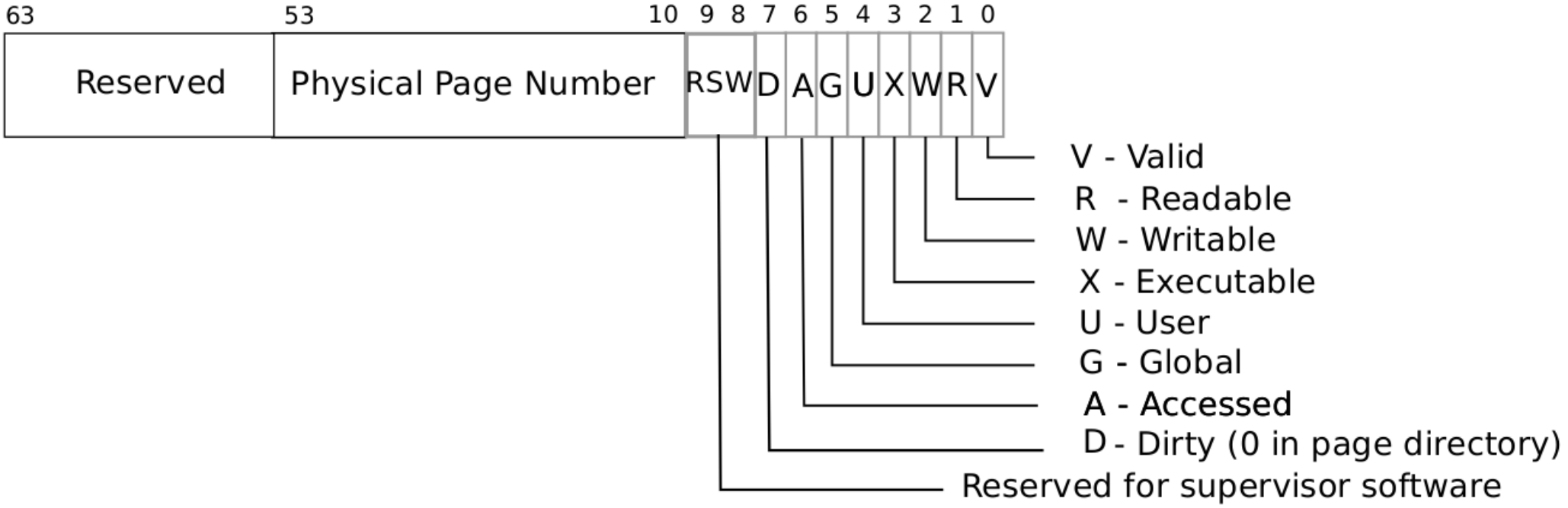
- Zaznam v tabulke PT sa nazyva PTE (Page Table Entry)
  - Ma 64b (8B), ale vyuziva sa “iba” 54 na adresu
  - Hornych 44 bitov PTE tvoria hornych 44 bitov fyzickej adresy (Physical Page Number = PPN)
  - Spodnych 10 su tzv. Priznaky
    - Present, Writeable, User, ...

# Preklad MMU

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


63
53
10
9
8
7
6
5
4
3
2
1
0
Reserved
Physical Page Number
RSW
D
A
G
U
X
W
R
V
V - Valid
R - Readable
W - Writable
X - Executable
U - User
G - Global
A - Accessed
D - Dirty (0 in page directory)
Reserved for supervisor software

PT

# PT

- Kde je PT ulozena?

# PT

- Kde je PT ulozena? V RAM

# PT

- Kde je PT ulozena? 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 aplikaciu (proces)

# PT

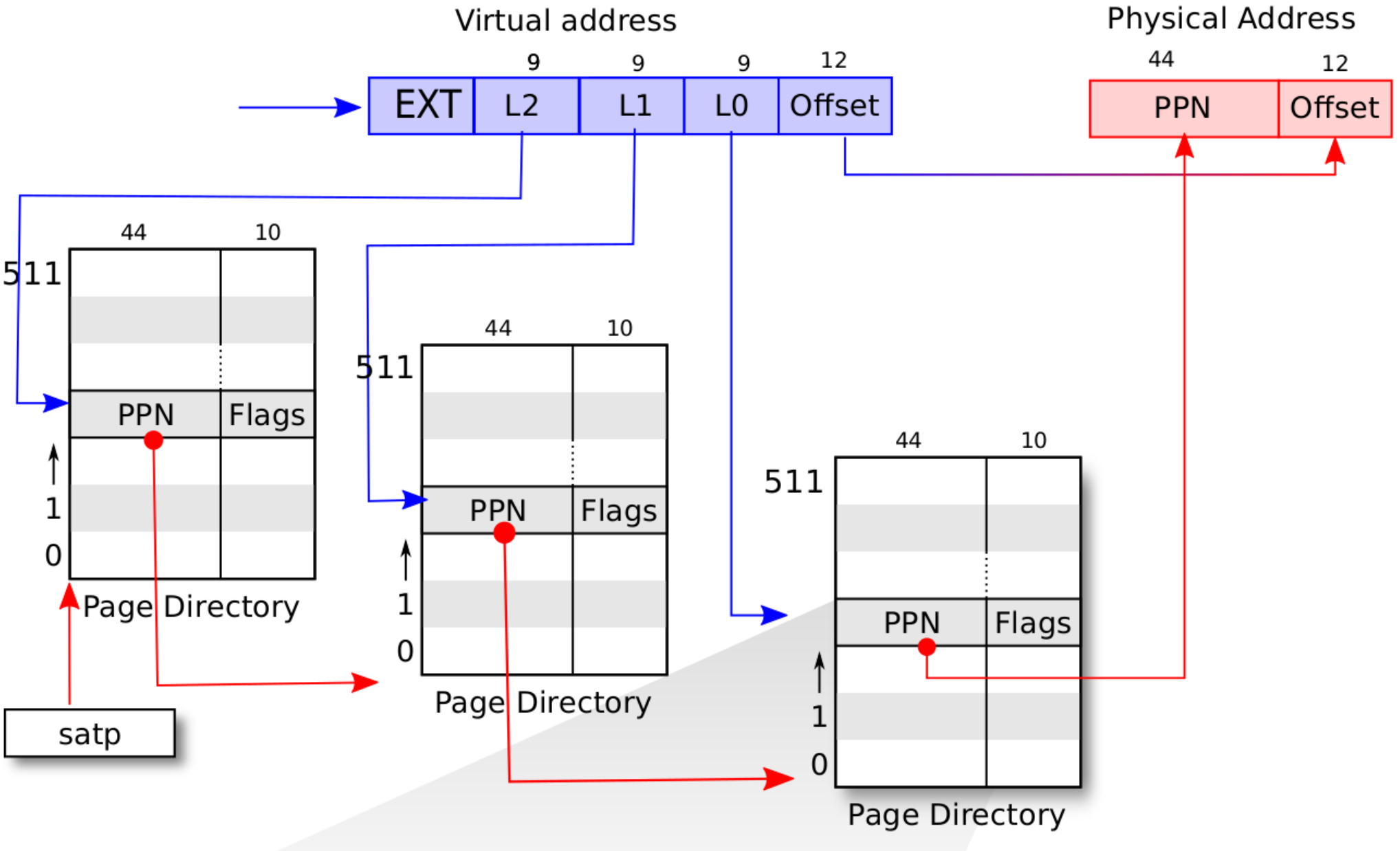
- Ako sme uz vypocitali, velkost PT je 1GB!
  - 1 PT na 1 adresny priestor
  - 1 adresny priestor pre 1 aplikaciu (proces)

- Avsak proces zvacsa potrebuje iba par k/MB pamate, takze 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


Virtual address
9
9
9
12
EXT
L2
L1
L0
Offset
Physical Address
44
12
PPN
Offset
44
10
511
PPN
Flags
1
0
Page Directory
44
10
511
PPN
Flags
1
0
Page Directory
44
10
511
PPN
Flags
1
0
Page Directory
satp

# PT

- V kazdej urovni (L0, L1, L2) mame k dispozicii 9 bitov na index → 512 poloziek v PD (Page Directory)

# PT

- V kazdej urovni (L0, L1, L2) mame k dispozicii 9 bitov na index → 512 poloziek v PD (Page Directory)
- 512*512*512 = 2^9 * 2^9 * 2^9 = 2^(3*9) = 2^27

# PT

- V kazdej urovni (L0, L1, L2) mame k dispozicii 9 bitov na index → 512 poloziek v PD (Page Directory)
- 512*512*512 = 2^9 * 2^9 * 2^9 = 2^(3*9) = 2^27
- PTE moze byt neplatne (bit Valid); taketo PTE “nejestvuju” (nejestvuje prepojenie t.j. mapovanie s RAM)
- Preto moze byt PT pre proces mala!

# PT

- V jednej urovni mame 512 poloziek PTE; jedna ma velkost 64b, takze celkovo je to 512*8B = 4kB

# PT

- V jednej urovni mame 512 poloziek PTE; jedna ma velkost 64b, takze celkovo je to 512*8B = 4kB (cirou “nahodou” to sedi na velkost stranky?)

# PT

- V jednej urovni mame 512 poloziek PTE; jedna ma velkost 64b, takze celkovo je to 512*8B = 4kB (cirou “nahodou” to sedi na velkost stranky?)
- 512 zaznamov moze ukazovat na 512 stranok v PA; 1 tabulkou vieme “pokryt” max. 512*4kB RAM, t.j. 2048kB, t.j. 2MB

# PT

- V jednej urovni mame 512 poloziek PTE; jedna ma velkost 64b, takze celkovo je to 512*8B = 4kB (cirou “nahodou” to sedi na velkost stranky?)
- 512 zaznamov moze ukazovat na 512 stranok v PA; 1 tabulkou vieme “pokryt” max. 512*4kB RAM, t.j. 2048kB, t.j. 2MB
- Na pokrytie mapovania niekolko desiatok MB nam stacia desiatky-stovky kB tabulky 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 v registri satp
- Prepisanim satp sa prepinaju adresne priestory!

# PT

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

- Na RISC-V je **FYZICKA** adresa hornej casti PT v registri satp
- Prepisanim satp sa prepinaju adresne priestory!

- Stranky PT mozu byt volne roztrusene v RAM-ke, nemusi ist o suvislu oblast 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 ukazuje do PD/L2; z L2[index] 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 ukazuje do PD/L2; z L2[index] ziskame PA pre obsah PD/L1
  - Dalsich 9 bitov indexu VA ukazuje do PD/L1; z L1[index] 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 ukazuje do PD/L2; z L2[index] ziskame PA pre obsah PD/L1
  - Dalsich 9 bitov indexu VA ukazuje do PD/L1; z L1[index] ziskame PA pre obsah PD/L0
  - Poslednych 9 bitov indexu VA ukazuje do PD/L0; z L0[index] ziskame PA pre PTE

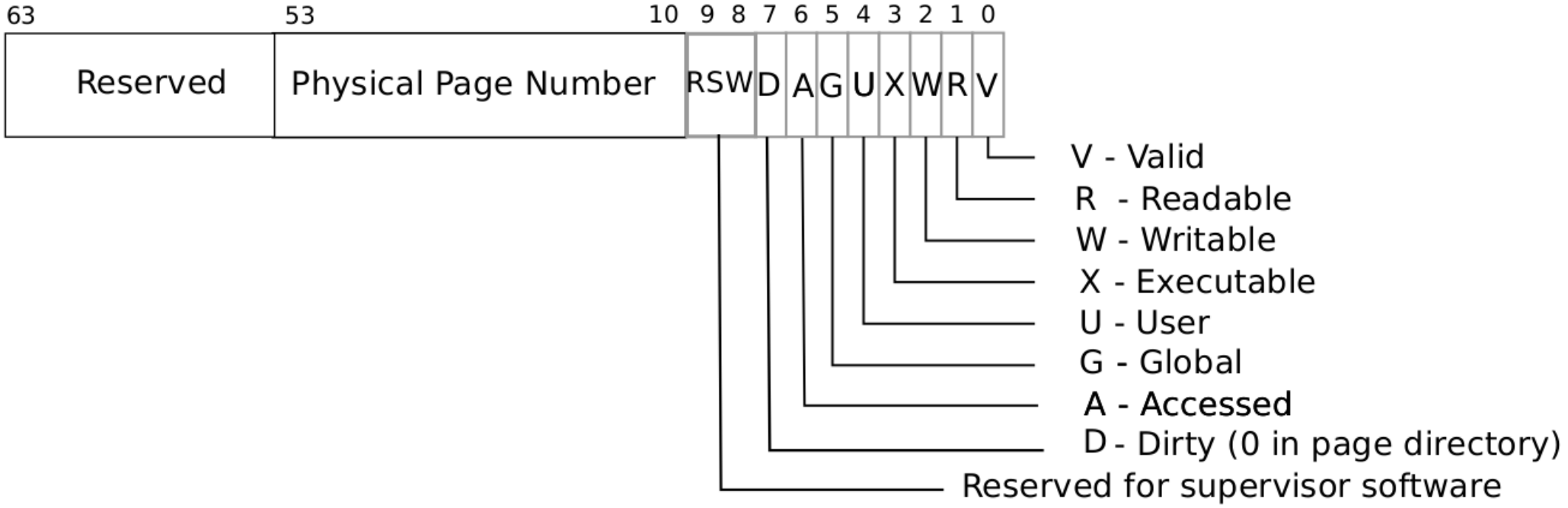
# Preklad va2pa

- 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 ukazuje do PD/L2; z L2[index] ziskame PA pre obsah PD/L1
  - Dalsich 9 bitov indexu VA ukazuje do PD/L1; z L1[index] ziskame PA pre obsah PD/L0
  - Poslednych 9 bitov indexu VA ukazuje do PD/L0; z L0[index] ziskame PA pre PTE
  - PA = PPN z PPN plus spodnych 12 bitov VA

# 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!
- Vacsina 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!
- Vacsina jadier OS vsak vyuziva VA; preco?
  - Je zlozite vypinat/zapinat strankovanie

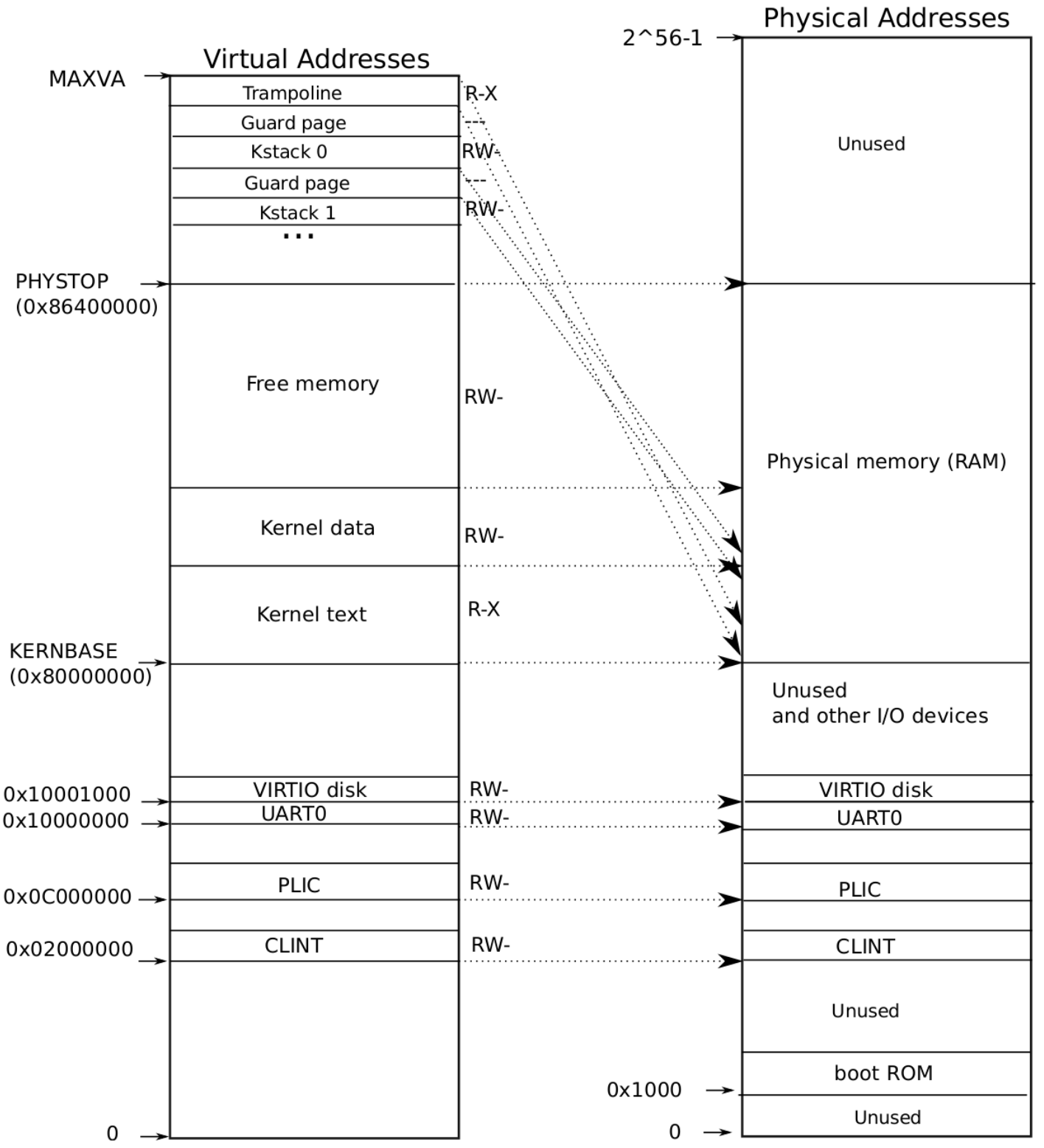
# Virtualna pamat v jadre

- Pre procesy je vyuzitie VM v poriadku, ale preco ju vyuzivat aj v jadre?
- Moze bezat jadro iba s fyzickou pamatou? ANO, moze!
- Vacsina 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!
- Vacsina 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 xv6


Virtual Addresses
Physical Addresses
MAXVA
2^56-1
Trampoline
Guard page
Kstack 0
Guard page
Kstack 1
R-X
---
RW-
---
RW-
Unused
PHYSTOP
(0x86400000)
Free memory
RW-
Physical memory (RAM)
Kernel data
RW-
Kernel text
R-X
KERNBASE
(0x80000000)
Unused
and other I/O devices
0x10001000
VIRTIO disk
RW-
VIRTIO disk
0x10000000
UART0
RW-
UART0
0x0C000000
PLIC
RW-
PLIC
0x02000000
CLINT
RW-
CLINT
Unused
boot ROM
0x1000
Unused
0
0

# Virtualna pamat xv6

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

# Virtualna pamat xv6

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

- Preco sa mapuju aj zariadenia?

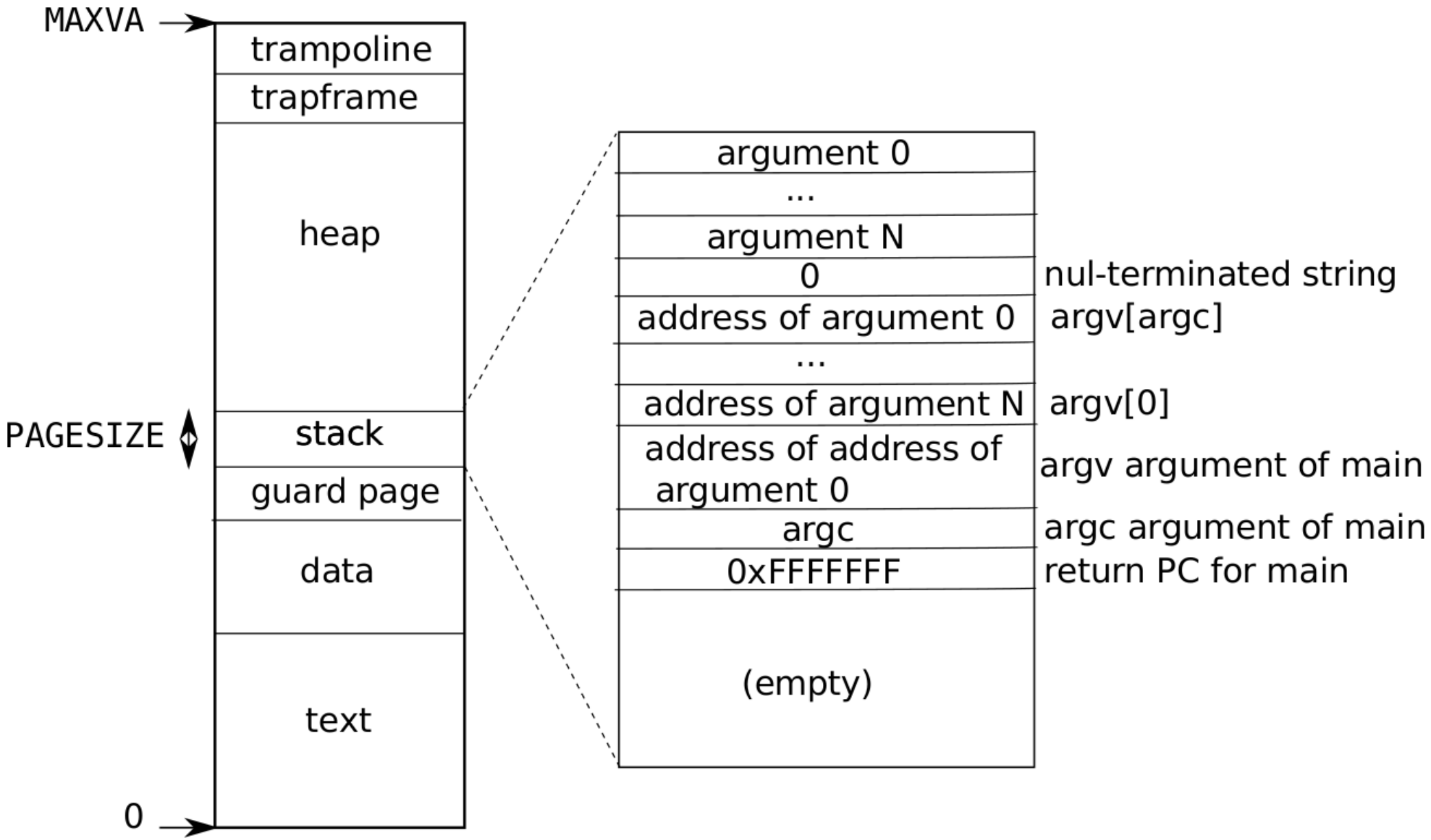
# Virtualna pamat xv6

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

# Virtualna pamat xv6

- Kazdy proces ma vlastny adresny priestor
- Vlastnu tabulku stranok
- Vid trampoline a trapframe – nie su zapisovatelne pre uzivatelsky proces!!!

# Virtualna pamat 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 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 ;)

# 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 ↔ 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 ↔ 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 ↔ 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 kvminit()
  - Kolko adresneho priestoru vie obsiahnut 1 L0 polozka? 4kB
  - Kolko 1 L1 polozka? 2MB
  - Kolko 1 L2 polozka? 1GB

# Kod VM xv6

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

# Kod VM xv6

- Kolko pamate (stranok) je pouzitych na reprezentaciu PT (t.j. mapovania adresneho priestoru) po prvom volani kvmmap()?
  - Kernel/vm.c:32 kvmmap(UART0, UART0, PGSIZE, PTE_R | PTE_W);

# Kod VM xv6

- Kolko pamate (stranok) je pouzitych na reprezentaciu PT (t.j. mapovania adresneho priestoru) po prvom volani kvmmap()?
  - Kernel/vm.c:32 kvmmap(UART0, UART0, PGSIZE, PTE_R | PTE_W);
- Kolko pamate (stranok) sa tymto volanim mapuje?

# Kod VM xv6

- Kolko poloziek v PT zabera CLINT?
  - kvmmap(CLINT, CLINT, 0x10000, …);
- Kolko poloziek v PT zabera PLIC?
  - kvmmap(PLIC, PLIC, 0x400000, …);

# Kod VM xv6

- Kolko poloziek v PT zabera CLINT?
  - kvmmap(CLINT, CLINT, 0x10000, …);
- Kolko poloziek v PT zabera 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

- 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 <va; va+size) na prislusny interval <pa; pa+size)
  - Nakolko va ani size nemusia byt zarovnane na velkost stranky, treba to osetrit

# Kod VM xv6

- Mappages()
  - Argumenty: top PD, va, size, pa, perm
  - Do PT zaznamenava mapovanie <va; va+size) na prislusny interval <pa; pa+size)
  - Nakolko va ani size nemusia byt zarovnane na velkost stranky, treba to osetrit
  - Algoritmus
    - Pre kazdu zarovnanu adresu va_addr:
      - Pomocou walkpgdir() najdi PTE pre va_addr
      - Cielovu PA vloz do PTE
      - Oznac 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 uz jestvuje
      - PTE2PA extrahuje PPN zo zaznamu v jestvujucej tabulke
    - If not set PTE_V in *PTE_addr
      - Alokuj tabulku dalsej urovne
      - Vypln *PTE_addr s PPN alokovanej tabulky (PA2PTE)
    - Vrat PTE z (vytvorenej/jestvujucej) tabulky

# 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”!!!

# 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()

# 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 hlavnu funkcionalitu
  - Pri prepnuti z jadra do user sa do satp ulozi adresa aktualizovanej PT

# Kod VM xv6

- Growproc() v kernel/proc.c
  - proc→sz je aktualna velkost procesu
  - Uvmalloc() obsahuje hlavnu funkcionalitu
  - Pri prepnuti z jadra do user 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)?

# Citanie na vecer/nad ranom

- Chapter 3: Page Tables