

Systemy komputerowe: architektura i programowanie (SYKOM)

Wykład: Mechanizmy wspierania systemu operacyjnego

Aleksander Pruszkowski

Instytut Telekomunikacji Politechniki Warszawskiej

PLAN WYKŁADY

- Separacja procesów
- Translacja adresów
- Wirtualizacja adresów
- Mechanizmy ochrony zasobów

**Tworzenie jednowątkowego kodu
dla zasobów peryferyjnych**

Tworzenie jednowątkowego kodu dla zasobów peryferyjnych

■ Realizacja zadań czasowych krytycznie

■ Czyli operacje wykonywane co zadany interwał czasu

- przykład: zmiana stanu „pinu” GPIO co określony czas – podejście trywialne (przykład zaczerpnięty z Arduino)

```
void setup() {  
    pinMode(LED_BUILTIN, OUTPUT);  
}  
void loop() {  
    digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on (HIGH is the voltage level)  
    delay(1000);                        // wait for a second  
    digitalWrite(LED_BUILTIN, LOW);     // turn the LED off by making the voltage LOW  
    delay(1000);                        // wait for a second  
}
```

- gdybyśmy chcieli zmieniać stan dwóch diod – druga miałaby migać np.: co 0,732sek
 - w systemie wielowątkowym moglibyśmy napisać (mamy dwa równoległe wykonujące się wątki loop_thread_1 i loop_thread_2

```
void setup() {  
    pinMode(LED1, OUTPUT);  
    pinMode(LED2, OUTPUT);  
    ... //tu powinno być wywołanie usługi powołania do życia  
        //dwóch wątków (loop_thread1 i loop_thread2)  
}  
void loop_thread_1() {  
    digitalWrite(LED1, HIGH);  
    delay(1000);  
    digitalWrite(LED1, LOW);  
    delay(1000);  
}  
void loop_thread_2() {  
    digitalWrite(LED2, HIGH);  
    delay(700);  
    digitalWrite(LED2, LOW);  
    delay(700);  
}
```

Tworzenie jednowątkowego kodu dla zasobów peryferyjnych

- Realizacja zadań czasowych krytycznie, cd
 - A gdy możemy tworzyć wyłącznie jedno wątkowy kod!?
 - generalna zasada – unikamy blokowania kodu jeżeli to możliwe – ale jak?
 - Na początku spróbujmy przepisać kod do postaci bez funkcji delay()
 - rozwiązanie inspirowane artykułem „multi-tasking-the-arduino” z <https://learn.adafruit.com>

```
...
#define DELAY_TIME 1000
unsigned long prev_time=0;
char state=0;
void setup() {
    pinMode(LED1, OUTPUT);
}
void loop() {
    unsigned long current_time=millis();
    if(current_time - prev_time > DELAY_TIME) {
        prev_time=current_time;
        if(state!=0) {
            digitalWrite(LED1, HIGH);
        }else{
            digitalWrite(LED1, LOW);
        }
        state=!state;
    }
}
```

- Nieco dłużej ale pozbyliśmy się niechcianej funkcji delay()
 - koszt to dodatkowe zmienne: `current_time`, `prev_time`, `state`

Tworzenie jednowątkowego kodu dla zasobów peryferyjnych

■ Realizacja zadań czasowych krytycznie, cd

- Dodajmy sterowanie dwóch diod – druga miała by migać np.: co 0,732sek

```
...
#define DELAY_TIME_1  1000
#define DELAY_TIME_2  732

unsigned long prev_time_1=0;
unsigned long prev_time_2=0;

char state_1=0;
char state_2=0;

void setup() {
    pinMode(LED1, OUTPUT);
    pinMode(LED2, OUTPUT);
}

void loop() {
    //pobieramy czas systemowy
    unsigned long current_time=millis();
    if(current_time - prev_time_1 >= DELAY_TIME_1){
        prev_time_1=current_time;
        if(state_1!=0){
            digitalWrite(LED1, HIGH);
        }else{
            digitalWrite(LED1, LOW);
        }
        state_1=!state_1;
    }
    current_time=millis(); //ponownie pobieramy czas
    if(current_time - prev_time_2 >= DELAY_TIME_2){
        prev_time_2=current_time;
        if(state_2!=0){
            digitalWrite(LED2, HIGH);
        }else{
            digitalWrite(LED2, LOW);
        }
        state_2=!state_2;
    }
}
```

A dlaczego nie „==”?

- kod stał się nieco dłuższy ale cel osiągnięto – brak miejsc blokowania się kodu

Tworzenie jednowątkowego kodu dla zasobów peryferyjnych

- Inne metody realizacji wielowątkowości bez wsparcia systemu operacyjnego - mechanizm „Protothreads” (stosowany w systemie Contiki)
 - Zapewnia tworzenie kodu wielowątkowego bez stosowania skomplikowanych niskopoziomowych mechanizmów wywłaszczania
 - mechanizm wywłaszczania - przełącza procesor między zadaniami
 - *mechanizm ma budowę skomplikowaną i mocno nie przenośną między różnymi procesorami*
 - typowa implementacja mechanizmu wywłaszczania wymaga dużych zasobów pamięciowych
 - „protothreads” ze swoim przełącznikiem zadowala się tylko 2...4B tej pamięci dla każdego wątku
 - „Protothreads” wykorzystuje mechanizm pre-procesingu języka C - dla zmiany kodu wielowątkowego w kod jedno-wątkowy
 - Ważniejsze wady podejścia
 - zmienne automatyczne nie są zachowywane przy przełączaniu zadań(!)
 - aby nie stracić ich zawartości - trzeba je świadomie zachować przed przełączeniem się do innego wątku
 - lub zmienić ich charakter - np.: poprzez deklaracje takich zmiennych jako statyczne - unika się umieszczania takich zmiennych na stosie procesora

Tworzenie jednowątkowego kodu dla zasobów peryferyjnych

- Inne metody realizacji wielowątkowości bez wsparcia systemu operacyjnego - mechanizm „Protothreads” (stosowany w systemie Contiki), cd.
 - „Protothreads” API
 - struct pt – struktura opisująca wątek
 - PT_THREAD(name_args) - deklaracja implementacji wątku
 - PT_INIT(pt) - inicjalizacja wątku
 - PT_BEGIN()/PT_END() - deklaracja początku i końca właściwej sekcji wątku
 - PT_WAIT_UNTIL(pt, cond)/PT_WAIT_WHILE(pt, cond) - warunkowe blokowanie wątku przydatne np.: gdy wątek czeka na dane z sensora
 - Dodatkowe funkcje
 - PT_WAIT_THREAD(pt, thread) - zaczekaj aż potomek „tego” wątku zakończy swoje działanie
 - PT_SPAWN()/PT_RESTART()/PT_EXIT()/PT_SCHEDULE()/PT_YIELD()/PT_YIELD_UNTIL - zaawansowane funkcje/makra sterowania pracą wątków
 - Ale jak to działa?

Tworzenie jednowątkowego kodu dla zasobów peryferyjnych

- Inne metody realizacji wielowątkowości bez wsparcia systemu operacyjnego - mechanizm „Protothreads” (stosowany w systemie Contiki), cd.
 - Przykład (za źródłami w: esb\apps\beeper.c)

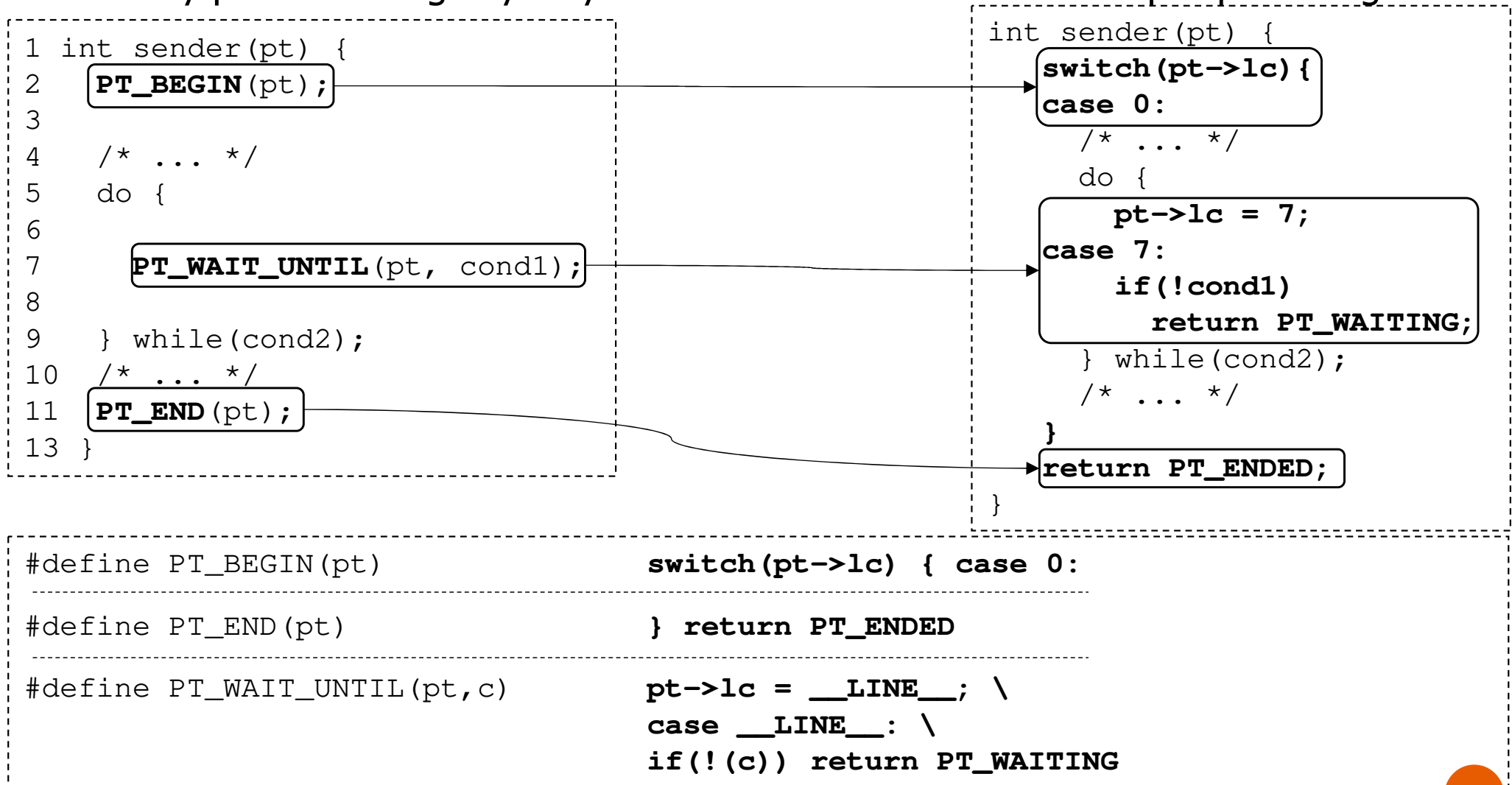
```
...
static struct pt beeper_pt;
static PT_THREAD(beeper_thread(struct pt *pt)) {
    PT_BEGIN(pt);
    while(1) {
        PT_WAIT_UNTIL(pt, etimer_expired(&etimer));
        etimer_reset(&etimer);
        leds_invert(LED_RED);
        ...
    }
    PT_END(pt);
}

void main(void) {
    PT_INIT(&beeper_pt);
    ...
    while(1) {
        beeper_thread(&beeper_pt);
        ...
    }
}
```

- Ale jak ten kod „widzi” procesor?

Tworzenie jednowątkowego kodu dla zasobów peryferyjnych

- Inne metody realizacji wielowątkowości bez wsparcia systemu operacyjnego - mechanizm „Protothreads” (stosowany w systemie Contiki), cd.
 - Aby procesor mógł wykonywać kod - do działania wchodzi pre-procesing



Wsparcie dla wielozadaniowości


Wsparcie dla wielozadaniowości

■ Jakie chciałoby się tworzyć programy

```
void task1(void){
    for(;;){
        digitalWrite(RED_LED, HIGH);
        delay(1000);
        digitalWrite(RED_LED, LOW);
        delay(1000);
    }
}

void task2(void){
    for(;;){
        digitalWrite(BLUE_LED, HIGH);
        delay(700);
        digitalWrite(BLUE_LED, LOW);
        delay(700);
    }
}

void main(void){
    for(;;){
        task1();
        task2();
    }
}
```



Taki kod wywoła task1 ale „zagłodzi” kod task2

Wersja taka:

```
for(;;){
    task2();
    task1();
}
```

Zagłodzi ale tym razem task1.

■ Więc rozwiązanie to „ślepa” uliczka

Wsparcie dla wielozadaniowości

■ Jakie chciałoby się tworzyć programy, cd.

```
void task1(void){
    for(;;){
        digitalWrite(RED_LED, HIGH);
        delay(1000);
        digitalWrite(RED_LED, LOW);
        delay(1000);
    }
}

void task2(void){
    for(;;){
        digitalWrite(BLUE_LED, HIGH);
        delay(700);
        digitalWrite(BLUE_LED, LOW);
        delay(700);
    }
}

void main(void){
    turn_on_timers_interrupts();

    create_task(task1, &task1state);
    create_task(task2, &task2state);

    switch_context(&task1state, NULL);

    for(;;){ //to już nie powinno się
    }        //wykonywać
}
```

Użyjmy kod nadzorujący:

```
void turn_on_timers_interrupts(void){
    ... //ustaw mechanizm przerwań
    ... //cyklicznie wywołujący timer_int()
    ... //fragment zależny od sprzętu
}

int state=0;
void timer_isr(void) ... {
    if(state==0){
        switch_context(&task1state, &task2state);
        state=1;
    }else{
        switch_context(&task2state, &task1state);
        state=0;
    }
}
```

Wsparcie dla wielozadaniowości

- Jakiego chciałoby się tworzyć programy, cd.
- Ale czym są: **create_task()**, **switch_context()**, **task1_state** i **task2_state**?
 - Ich implementacja jest specyficzna dla danej architektury (!!!)
 - `create_task(task, &task_state)` – tworzy dla funkcji opisu zadania 'task' (implementacji danego zadania) odpowiedni zapis w 'task_state'
 - `switch_context(&task_state_new, &task_state_old)` – dotychczasowy stan zapamiętywany jest w `task_state_old`, natomiast aktualny stan systemu odtwarzany jest z zapisanego w `task_state_new`
 - `task1state` i `task2state` to struktury gdzie przechowywany jest opis stanu danego zadania, są to:
 - aktualna zawartość PC
 - zawartość rejestrów
 - stos danego zadania
 - ...

Wsparcie dla wielozadaniowości

- Jakiego chciałoby się tworzyć programy, cd.
- Ale czym są: **create_task()**, **switch_context()**, **task1_state** i **task2_state?**, cd.
 - Funkcja `switch_context()` przełącza stan procesora z jednego opisu stanu na opis stanu drugiego zadania
 - Oba zadania otrzymują czas CPU dla swojego działania
 - Jak często to przełączanie ma występować?
 - Zależnie od przyjętej polityki tzw. scheduler'a
 - Gdy dane zadanie zatrzyma się na dostępie do We/Wy
 - gdy proces czeka (np.: na dane z dysku, z sieci, na akcję użytkownika, ...) to przekazuje sterowanie do innych procesów które są gotowe do dalszej pracy
 - A jak zabezpieczyć oba zadania przed złym zachowaniem tego drugiego?
 - konieczne jest wprowadzenie mechanizmów ochrony zasobów, w tym pamięci, np.: poprzez separację zadań/procesów
 - Proces, zadanie, wątek - różnią się tylko detalami (np.: inne API używane do ich tworzenia)

Separacja zadań

Separacja procesów

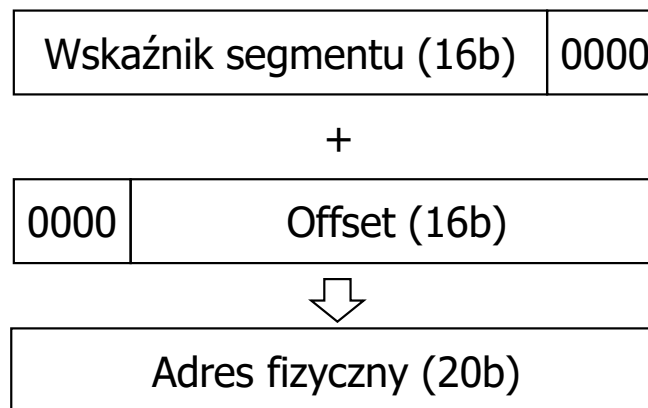
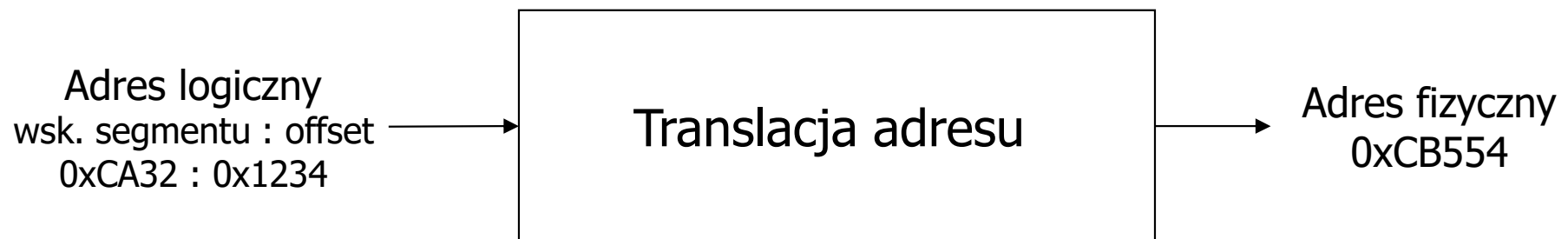
■ Separacja procesów

- To jedna z metod ochrony przed nadużyciami złośliwego oprogramowania
 - Aplikacje „mają wrażenie” bycia jedynymi w systemie
- Przestrzenie adresowe mogą stosować
 - Adresy logiczne
 - Inaczej adresy tworzące przestrzeń wirtualną, pomagającą odzwierciedlić budowę aplikacji (kod, dane, stos)
 - Adresy fizyczne
 - Faktyczny adres używany do odwołania się do określonego miejsca w zewnętrznych modułach pamięci
 - W systemach bez tzw. translacji adresów – czyli bez MMU – adresy fizyczne są równoważne adresom wirtualnym
 - Mikroprocesor wyposażony w MMU także nie wspiera powyższych mechanizmów dopóki nie zostanie odpowiednio skonfigurowany jego MMU
- Typowo systemy komputerowe wspierają translacje adresów opartą o
 - Segmentacje
 - Gdzie adres tworzą: wskaźnik segmentu i offset w jego obrębie
 - Stronicowanie
 - Gdzie przestrzeń jest podzielona na jednakowej długości strony
 - Są systemy które wykorzystują oba mechanizmy(!)

Translacja adresów

■ Translacja adresów

- Translacja adresu logicznego na fizyczny - przypadek z architektury X86 pracującej w tzw. **trybie rzeczywistym**



Translacja adresów

■ Translacja adresów

- Przypadek z architektury X86 pracującej w tzw. **trybie rzeczywistym**, cd.

- Zalety

- Prostota

- Wady

- Brak separacji

- Segmenty mogą nachodzić na siebie(!)

- Ograniczona przestrzeń adresowa

- Adres może składać się z tylko 20bitów – rozwiązanie tego problemu wymagało sztuczek sprzętowych i programistycznych (np.: Extended MEM, ...)

- Wskaźników segmentów – przechowywały taki wskaźnik rejestry segmentowe

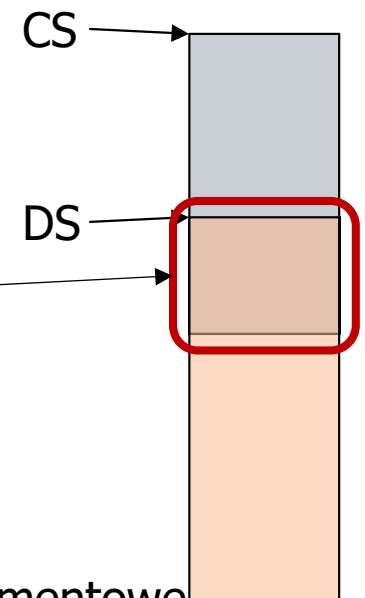
- CS – segment kodu

- DS – segment danych

- SS – segment stosu

- ES, FS, GS – segmenty dodatkowe

- Segmenty miały stałą długość 65536 B



Wirtualizacja adresów

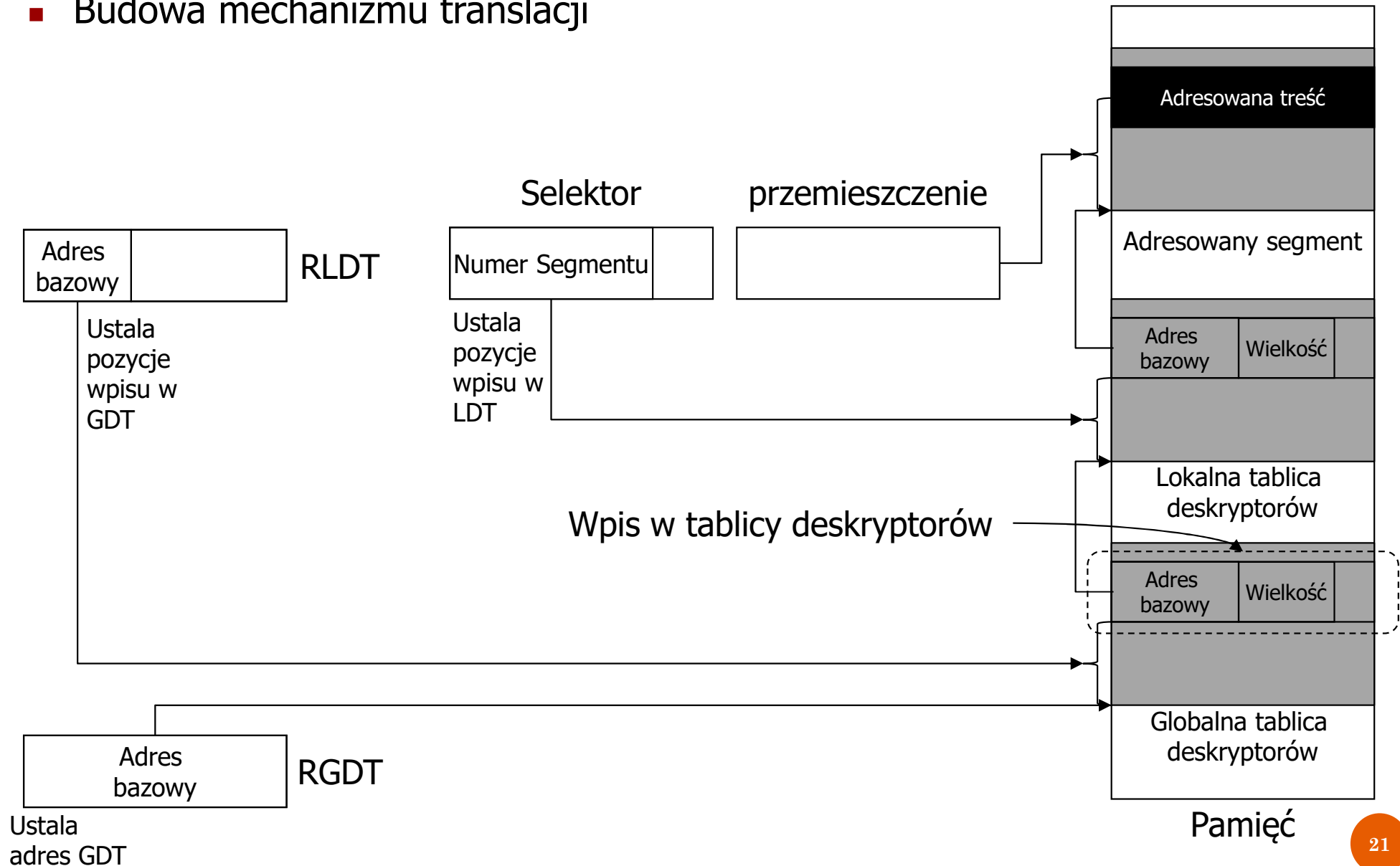
- Wirtualizacja adresów z wykorzystaniem **segmentacji** w architekturze **X86 (tryb chroniony)**
 - Adres składa się z elementów:

Selektor : przemieszczenie

- Selektor to
 - 13 bitów wskazujących na numer segmentu w tablicy deskryptorów
 - 1 bit (TI) określający typ segmentu (0-globalny, 1-lokalny)
 - 2 bity (RPL - Requested Privilege Level) poziom żądania przywileju przy dostępie do segmentu
 - Poziomy uprzywilejowania:
 - 0 – największe uprzywilejowanie, np.: jądro systemu
 - 1 – sterowniki systemu
 - 2 – bazy danych, usługi
 - 3 – aplikacje użytkowe

Wirtualizacja adresów

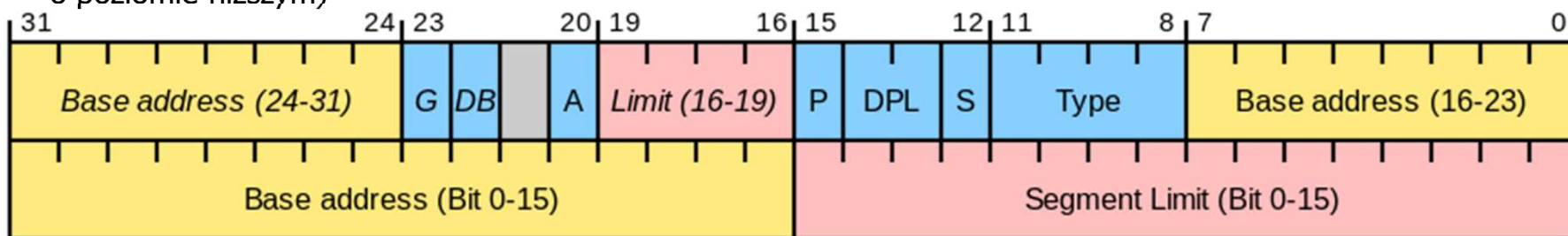
- Wirtualizacja adresów z wykorzystaniem **segmentacji** w architekturze **X86 (tryb chroniony)**, cd.
 - Budowa mechanizmu translacji



Wirtualizacja adresów

■ Wirtualizacja adresów z wykorzystaniem **segmentacji** w architekturze **X86 (tryb chroniony)**, cd.

- Pola w tablicy deskryptorów (zarówno globalnej jak i lokalnej)
 - Adres bazowy segmentu (32 bity), wielkość segmentu (20 bitów)
 - Atrybuty segmentu (12 bitów) a wśród nich najważniejsze to
 - P – (Presence) obecność segmentu w pamięci (1 bit), DPL – poziom ochrony segmentu (2 bity)
 - G – (Graniness) ziarnistości pola wielkość segmentu (1 bit), jeżeli 0 to pole wielkości opisuje jednostka 1B, jeżeli 1 to jednostką jest 4096B
 - wtedy dla wyznaczenia wielkości segmentu pole wielkości segmentu przemnaża się przez 4096
 - A – (Absence) użycie segmentu (1 bit), jeżeli 1 to segment był użyty
 - TYPE – co przechowuje segment (4 bity) i może to być segment:
 - Z danymi tylko do odczytu lub z danymi do odczytu i zapisu
 - Rozszerzalny w dół tylko do odczytu lub rozszerzalny w dół do odczytu i zapisu
 - Z kodem i być tylko do wykonania lub do wykonania i odczytu
 - Tzw. zgodny z kodem tylko do wykonania lub zgodny do wykonania i odczytu, ...
(segment zgodny umożliwia dostęp do niego nie tylko z segmentu o takim samym poziomie ochrony a także o poziomie niższym)



Mechanizmy ochrony zasobów

- Wirtualizacja adresów z wykorzystaniem **segmentacji** w architekturze **X86 (tryb chroniony)**, cd.
 - Separacji zdań
 - Zadania widzą pamięć tak jakby w systemie były same
 - Kontrola przemieszczeń
 - System operacyjny przydziela zasoby pamięciowe zadaniom a mechanizmy sprzętowe kontrolują przestrzeganie tego przydziału
 - Segment jest opisany jego wielkością – łatwo wykryć odwołanie poza jego obszar
 - Kontrola typów segmentów podczas odwołań
 - Mechanizm blokuje możliwość użycia segmentu w niewłaściwy sposób
 - Segment zawiera pole określające typ - pozwala ono uniemożliwić np.: wykonywanie kodu z pamięci danych, modyfikacje segmentu kodu, ...

Mechanizmy ochrony zasobów

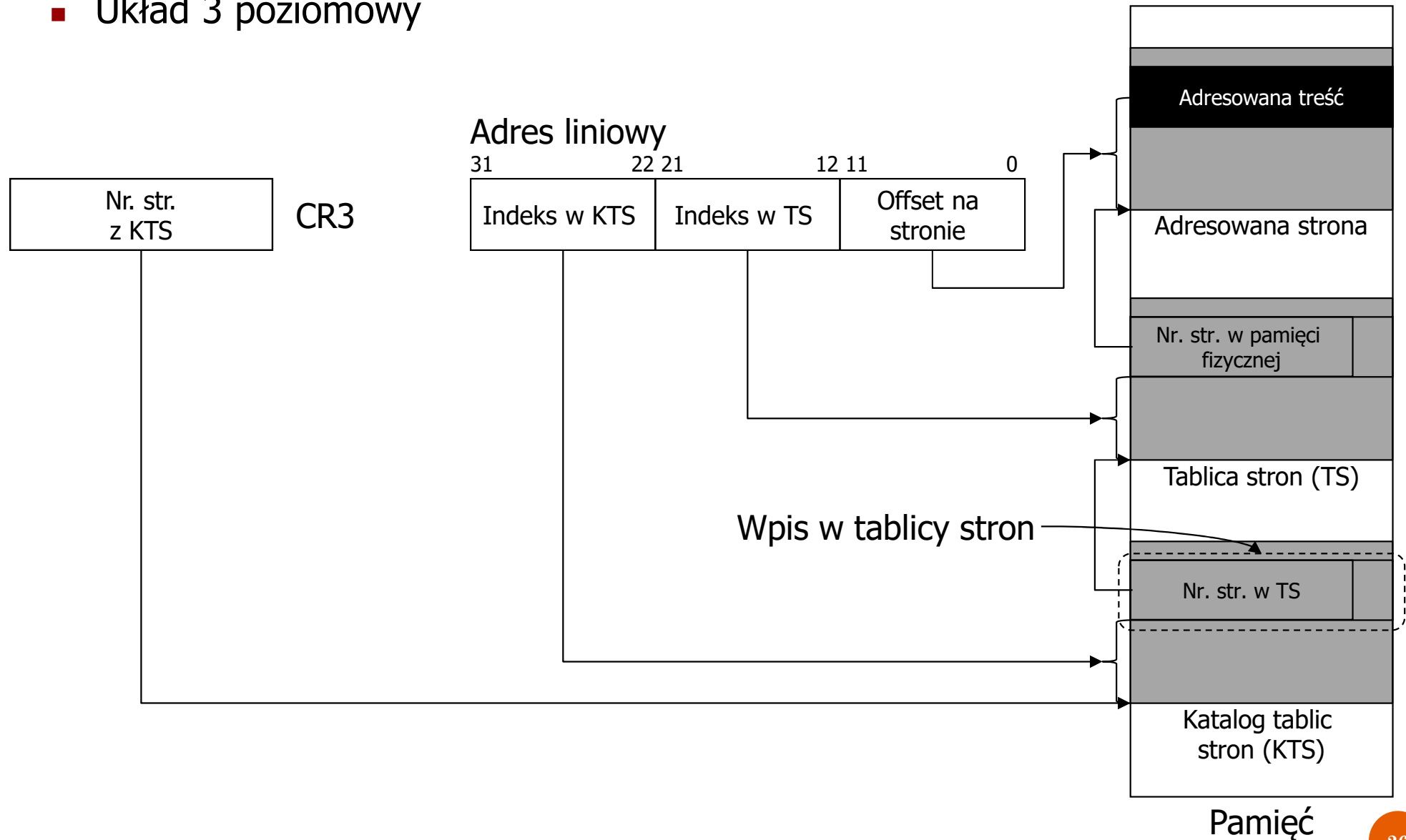
- Wirtualizacja adresów z wykorzystaniem **segmentacji** w architekturze **X86 (tryb chroniony)**, cd.
 - Kontrola poziomów ochrony
 - Mechanizm określa kto z danego zasobu pamięciowego może korzystać
 - Deskryptor segmentu zawiera pole DPL (Descriptor Privilege Level)
 - Podczas działania danego zadania wykonuje się ono z określonym przez system operacyjny poziomem CPL (Current Privilege Level)
 - Zadanie ma dostęp tylko do segmentów __danych__ których DPL jest większy lub równy od CPL (co do wartości)
 - Zadanie nie może odwoływać się do danych będących własnością systemu operacyjnego
 - Zadanie może operować na danych równie ważnych co to zadanie lub mniej ważnych
 - Zadanie ma dostęp tylko do segmentów __kodu__ których DPL jest mniejszy lub równy względem CPL
 - Zadanie może swobodnie wywoływać usługi systemu operacyjnego
 - Zadanie nie może wywoływać mniej zaufanego kodu
 - Istnieje także tzw. RPL (Requestor's Privilege Level)
 - Rozwiązuje problem gdy funkcja systemowa ma odczytać dane użytkownika by np.: zapisać je na dysku (systemowa funkcja write()) lub wywołać funkcję main() aplikacji użytkowej
 - Dzięki sprawdzeniu warunku $\max(\text{CPL}, \text{RPL}) \leq \text{DPL}$, prowadzi do sytuacji gdy RPL staje się CPL koduwołającego usługę systemowa wtedy mechanizm może dopuścić do wykonania danej operacji

Wirtualizacja adresów

- Wirtualizacja adresów z wykorzystaniem **stronicowania** w architekturze **X86 (tryb chroniony)**
 - Stronicowanie – mechanizm oparty o jednostki zwane stronami o stałej długości
 - Budowa wpisu w tablicy stron (TS) lub w katalogu tablic stron (KTS) – pola
 - Numer ramki strony w pamięci fizycznej
 - Atrybuty strony (12 bitów), najważniejsze z nich
 - D (Dirty) – czy treść strony została zmieniona (1 bit)
 - A (Accessed) – czy strona była używana (1 bit)
 - P (Present) – czy strona jest obecna w pamięci fizycznej (1 bit)
 - U/S (User/Supervisor) – czy właścicielem strony jest użytkownik czy system (1 bit)

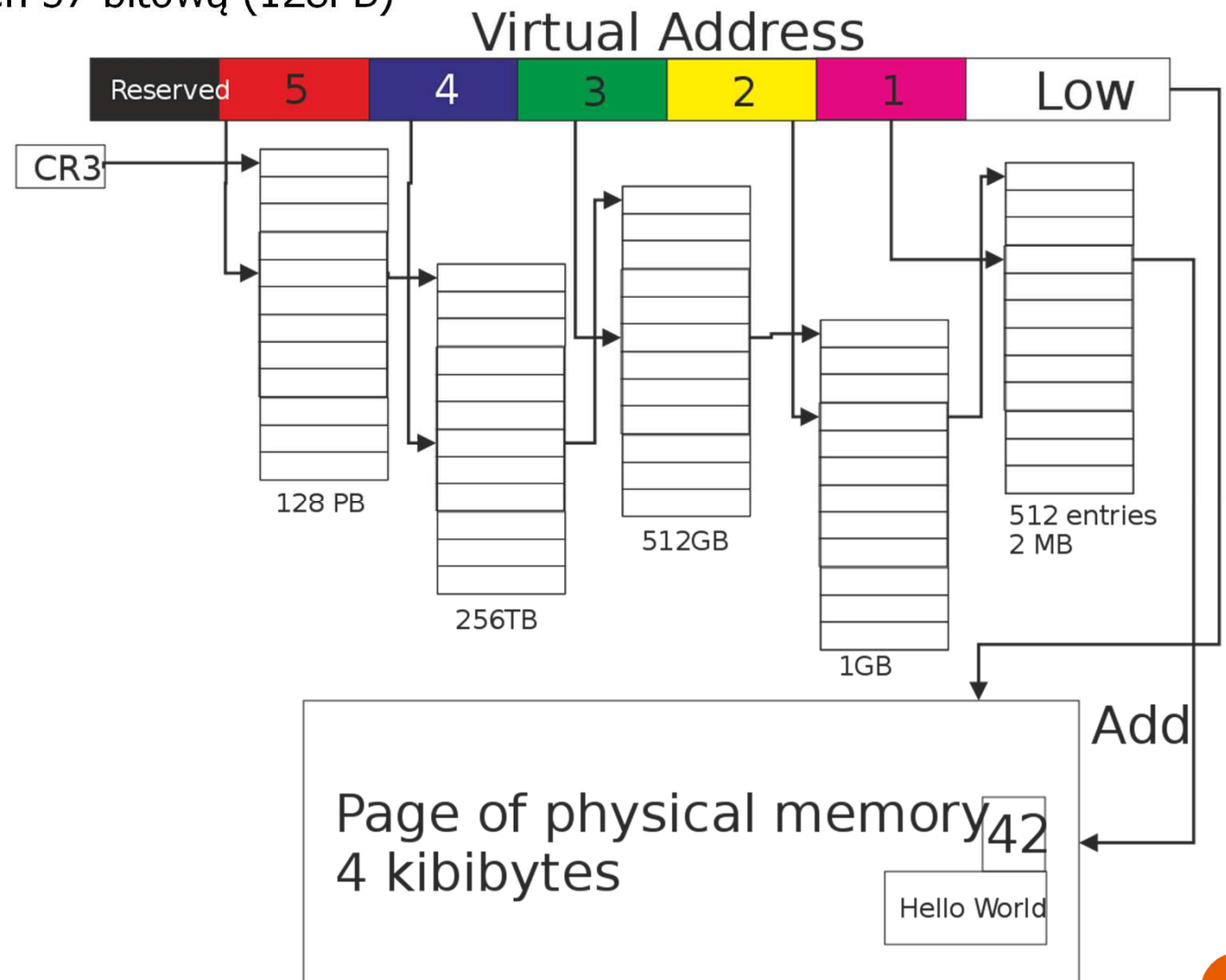
Wirtualizacja adresów

- Wirtualizacja adresów z wykorzystaniem **stronicowania** w architekturze **X86 (tryb chroniony)**, cd.
 - Układ 3 poziomowy



Wirtualizacja adresów

- Wirtualizacja adresów z wykorzystaniem **stronicowania** w architekturze **X86-64 (tryb chroniony)**
 - Układ 5 poziomowy (Intel 5-level paging)
 - Obsługuje przestrzeń 57 bitową (128PB)



Mechanizmy ochrony zasobów

■ Zalety i wady obu mechanizmów ochrony pamięci w **x86**

■ Zalety obu

- Możliwość przechowywania treści segmentów w pamięci dyskowej (plik/partycja wymiany) gdy w danym momencie brak pamięci fizycznej
- Manipulując rejestrami RGDT/RLDT lub CR3 można przełączać się między zadaniami
 - Każde zadanie ma własny zestaw segmentów/stron – więc dla nowego zadania przełącza się tylko(!) organizację pamięci

■ Zalety segmentacji

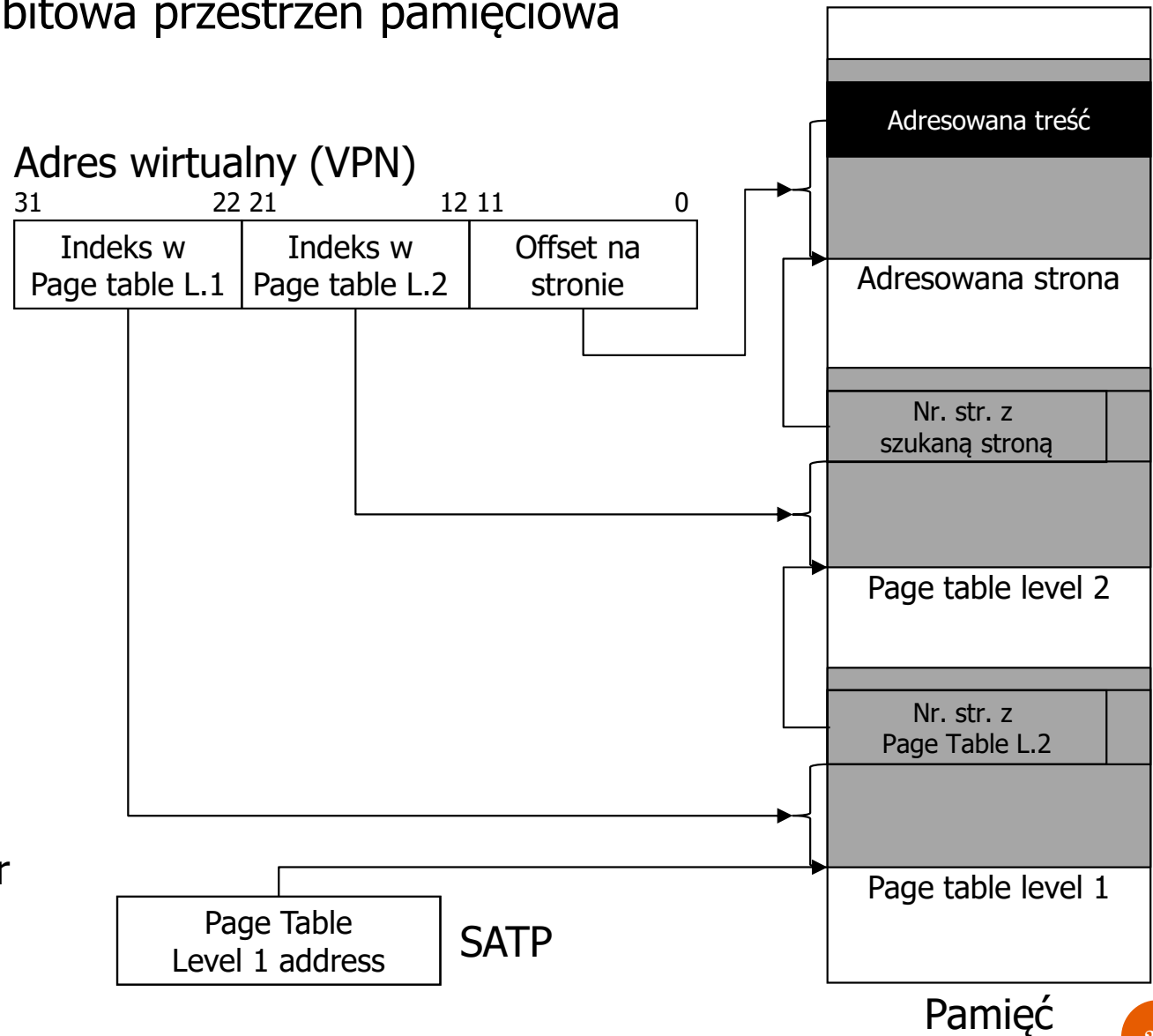
- Możliwość uporządkowania przestrzeni pod względem spełnianych funkcji
 - Przestrzeń pamięci kodu odseparowana od pamięci danych, stos rozszerzający się, możliwa jest np.: separacja jednej dużej tablicy z danymi od innych struktur

■ Zalety stronicowania

- Podział pamięci na stałej długości elementy dzięki czemu łatwo ich treść przenosić z pamięci fizycznej do pamięci wymiany
 - Tu widać pewną ułomność segmentacji – segmenty o różnych długościach trudniej przenosić z/do pamięci wymiany, pojawia się wtedy problem dopasowania i pozostających dziur w takiej pamięci wymiany

Wirtualizacja adresów

- Wirtualizacja adresów z wykorzystaniem **stronicowania** w architekturze **Risc-V**
 - Mechanizm **Sv32** – 32bitowa przestrzeń pamięciowa



- Strony 4KB
- VPN – Virtual Page Number

Wirtualizacja adresów

■ Wirtualizacja adresów z wykorzystaniem **stronicowania** w architekturze **Risc-V**, cd.

■ Format wpisu w tablicy stron (page-table entries, PTE)

■ Physical Page Number (PPN)

- PPN[1] – bity 31...20
- PPN[2] – bity 19...10

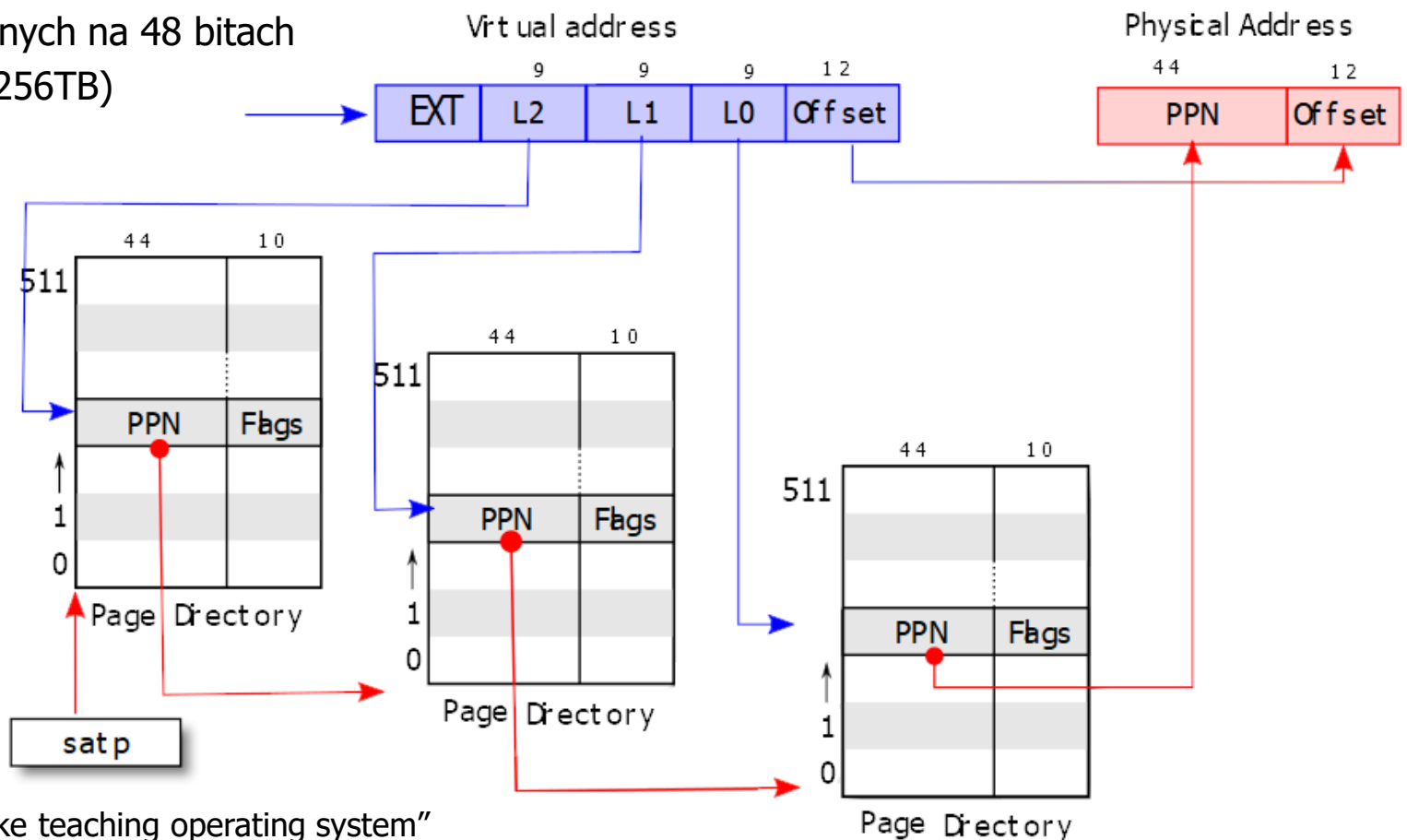
■ Atrybuty strony (10 bitów), najważniejsze z nich

- X/W/R – atrybut praw dostępu wykonanie/pisanie/czytanie (3 bity)
 - Gdy te bity mają wartość 000 – wykonywane jest odwołanie od „Page Table” następnego poziomu
- U – strona użytkowana przez system czy aplikację użytkową (1bit)
- D (Dirty) – czy treść strony została zmieniona (1 bit)
- A (Accessed) – czy strona była używana (1 bit)
- V (Valid) – czy wpis jest aktywny (1 bit)

X	W	R	Meaning
0	0	0	Pointer to next level of page table.
0	0	1	Read-only page.
0	1	0	<i>Reserved for future use.</i>
0	1	1	Read-write page.
1	0	0	Execute-only page.
1	0	1	Read-execute page.
1	1	0	<i>Reserved for future use.</i>
1	1	1	Read-write-execute page.

Wirtualizacja adresów

- Wirtualizacja adresów z wykorzystaniem **stronicowania** w architekturze **Risc-V**, cd.
 - Mechanizm translacji dostępne w architekturach 64bitowych (RV64)
 - **Sv39**
 - Implementacja wirtualnej przestrzeni o adresach zapisanych na 39 bitach (czyli 512GB)
 - **Sv48**
 - Implementacja wirtualnej przestrzeni o adresach zapisanych na 48 bitach (czyli 256TB)



Dziękujemy za uwagę!