TDIU11 – Föreläsning 3

Minneshantering (Memory management)

Filip Strömbäck

LINKÖPING UNIVERSITY

Planering

Vecka	Föreläsning	Seminarie
3	Processer och trådar	—
4	Minneshantering	Challenge 1: Schemaläggning
5	Virtuellt minne	Artikel 1: Schemaläggning
6	Filsystem och lagring	Challenge 2: Virtuellt minne
7	Säkerhet	Challenge 3: Filsystem
8	Repetition/utblickar	Artikel 2: Filsystem
9	—	Challenge 4: Säkerhet
10	Tentaförberedelse	Challenge 5: Repetition
11	(omtenta-p)	Artikel 3: Säkerhet (reserv)

LINKÖPING UNIVERSITY

Mål

Process 1

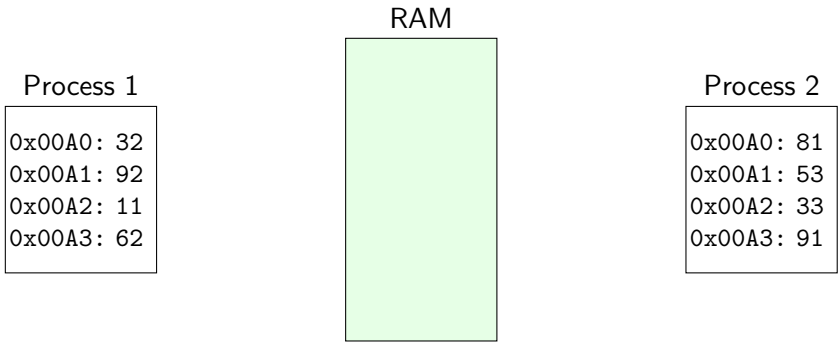
```
0x00A0: 32
0x00A1: 92
0x00A2: 11
0x00A3: 62
```

Hur går detta ihop?

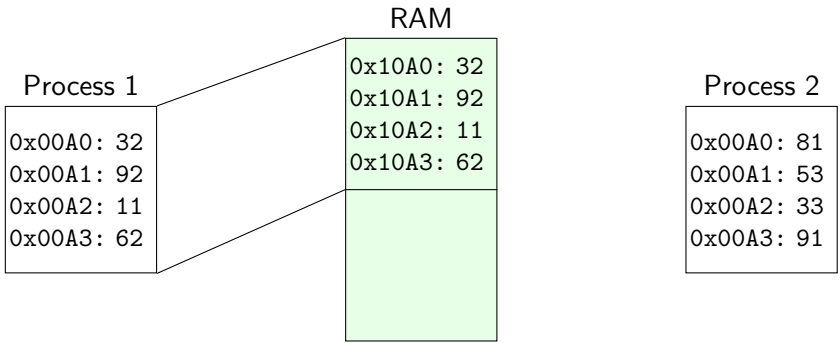
Process 2

```
0x00A0: 81
0x00A1: 53
0x00A2: 33
0x00A3: 91
```

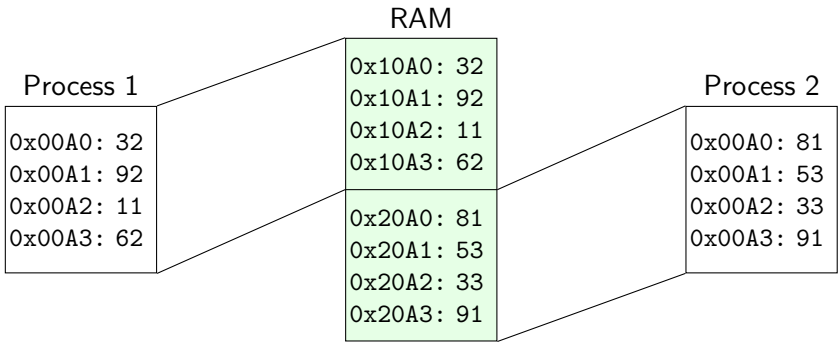
Mål


RAM
Process 1
0x00A0: 32
0x00A1: 92
0x00A2: 11
0x00A3: 62
Process 2
0x00A0: 81
0x00A1: 53
0x00A2: 33
0x00A3: 91

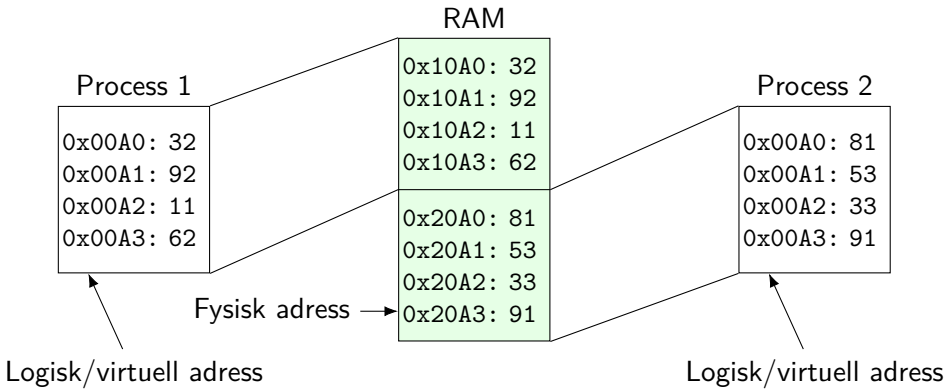
Mål


RAM
Process 1
0x00A0: 32
0x00A1: 92
0x00A2: 11
0x00A3: 62
0x10A0: 32
0x10A1: 92
0x10A2: 11
0x10A3: 62
Process 2
0x00A0: 81
0x00A1: 53
0x00A2: 33
0x00A3: 91

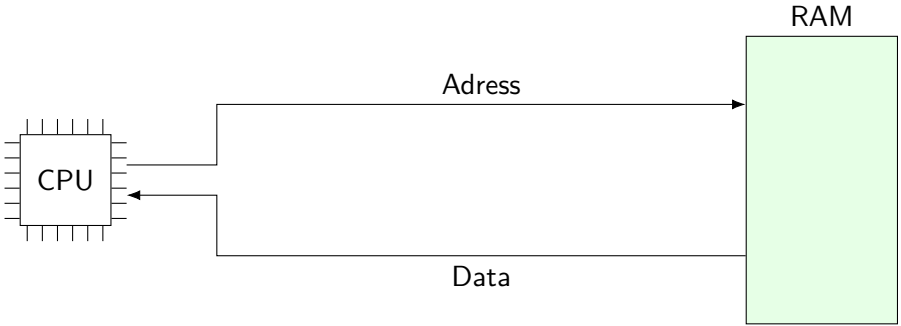
Mål


RAM
Process 1
0x00A0: 32
0x00A1: 92
0x00A2: 11
0x00A3: 62
0x10A0: 32
0x10A1: 92
0x10A2: 11
0x10A3: 62
0x20A0: 81
0x20A1: 53
0x20A2: 33
0x20A3: 91
Process 2
0x00A0: 81
0x00A1: 53
0x00A2: 33
0x00A3: 91

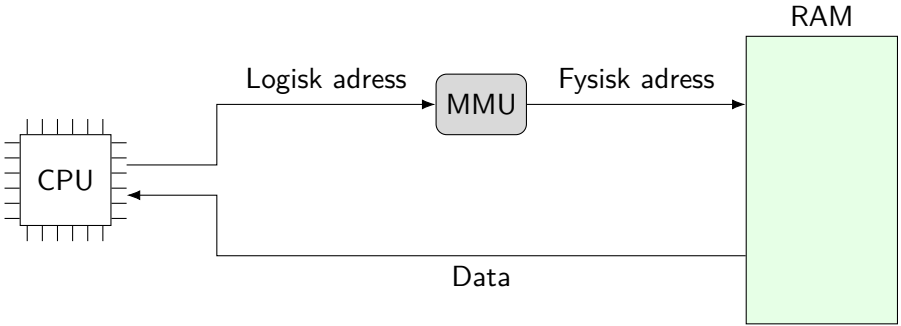
Mål


RAM
Process 1
0x00A0: 32
0x00A1: 92
0x00A2: 11
0x00A3: 62
0x10A0: 32
0x10A1: 92
0x10A2: 11
0x10A3: 62
0x20A0: 81
0x20A1: 53
0x20A2: 33
0x20A3: 91
Process 2
0x00A0: 81
0x00A1: 53
0x00A2: 33
0x00A3: 91
Fysisk adress
Logisk/virtuell adress
Logisk/virtuell adress

Hur löser vi detta?



Hur löser vi detta?



Notera: logisk och fysisk adress kan vara olika stora!

LINKÖPING UNIVERSITY

Hur stor är en megabyte?

- Storlek på minne är ofta 2^n för något heltal $n \geq 0$
- Smidigt med 2^{10} bytes i en kilobyte
- Hur många bytes är 1 MB?

Prefix	SI	Tvåpotens
kilo	1 kB = 10^3 B	1 KiB = 2^{10} B = 1024 B
mega	1 MB = 10^6 B	1 MiB = 2^{20} B = 1024 KiB = 1048576 B
giga	1 GB = 10^9 B	1 GiB = 2^{30} B = 1024 MiB = 1073741824 B

Räkna med tvåpotenser

Ofta smidigt att behålla tal i bas 2:

Exempel: Hur stor är en logisk minnesrymd på 32 och 48 bitar?

$$2^{32}\ \text{bytes} = 2^2 \cdot 2^{30}\ \text{bytes} = 2^2\ \text{GiB} = \quad 4\ \text{GiB}$$

$$2^{48}\ \text{bytes} = 2^8 \cdot 2^{40}\ \text{bytes} = 2^8\ \text{TiB} = \quad 256\ \text{TiB}$$

(Titta på tiotalet i exponenten, det ger rätt enhet)

$2^0 =$	1		
$2^1 =$	2		
$2^2 =$	4	$2^{10} =$	1 KiB
$2^3 =$	8	$2^{20} =$	1 MiB
$2^4 =$	16	$2^{30} =$	1 GiB
$2^5 =$	32	$2^{40} =$	1 TiB
$2^6 =$	64	$2^{50} =$	1 PiB
$2^7 =$	128	$2^{60} =$	1 EiB
$2^8 =$	256		
$2^9 =$	512		

Hexadecimala tal

Ofta smidigt att jobba i bas 16 (hexadecimalt):

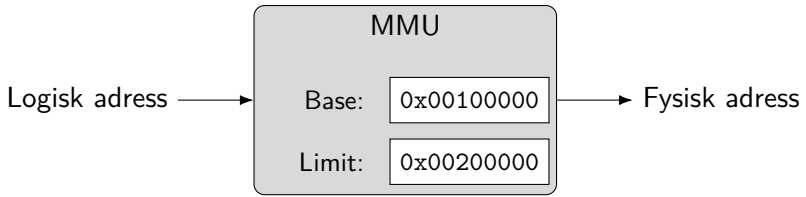
- 0-9, A-F $\Rightarrow$ Varje position är 4 bitar
- Konvention i kursen: Prefix `0x` som i C och C++

Exempel: 32-bitars tal

Hex:	`0x`	`1`	`2`	`4`	`8`	`1`	`3`	`7`	`F`
Bin:		`0001`	`0010`	`0100`	`1000`	`0001`	`0011`	`0111`	`1111`

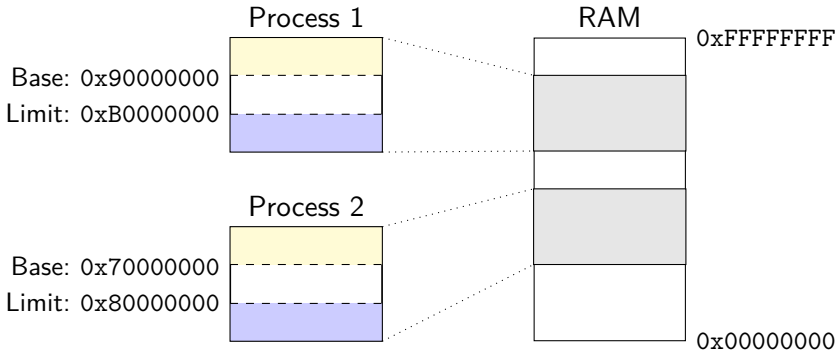
LINKÖPING UNIVERSITY

Kontinuerlig (Continuous allocation)

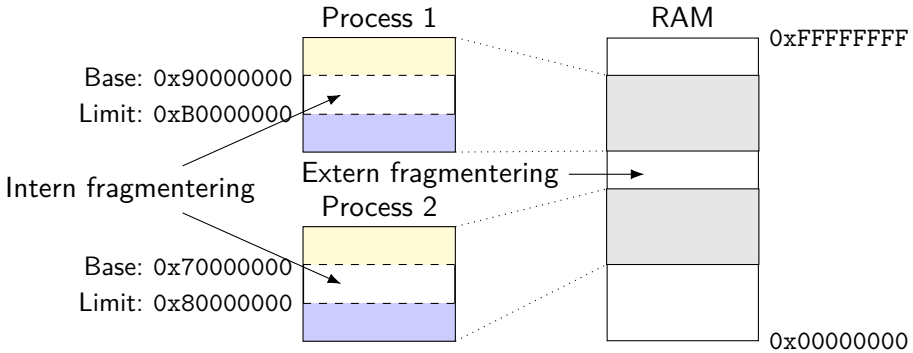


Notera: Varje process har olika MMU-inställningar!

Kontinuerlig (Continuous allocation)



Kontinuerlig (Continuous allocation)

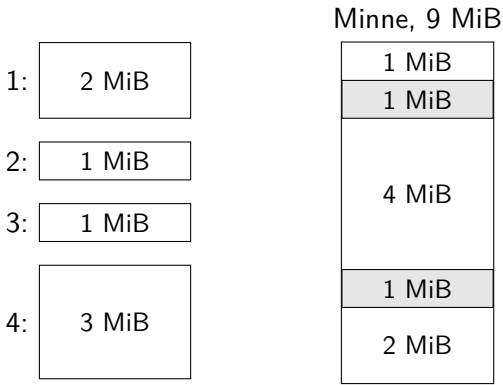


Allokering av block

Finns olika strategier:

- *First-fit*
- *Best-fit*
- *Worst-fit*

Notera: Kan också användas för att implementera `new` och `malloc`.



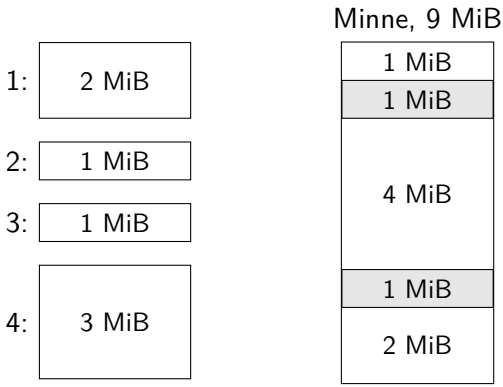
Allokering av block

Finns olika strategier:

⇒ *First-fit*

- *Best-fit*
- *Worst-fit*

Notera: Kan också användas för att implementera `new` och `malloc`.



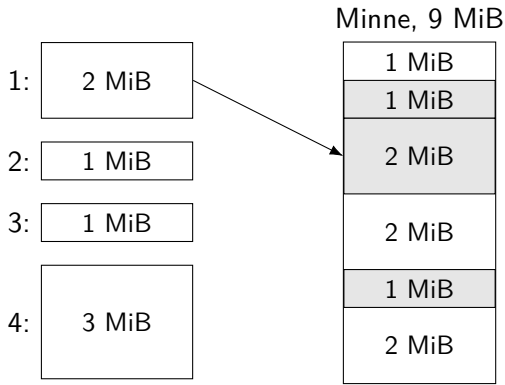
Allokering av block

Finns olika strategier:

⇒ *First-fit*

- *Best-fit*
- *Worst-fit*

Notera: Kan också användas för att implementera `new` och `malloc`.

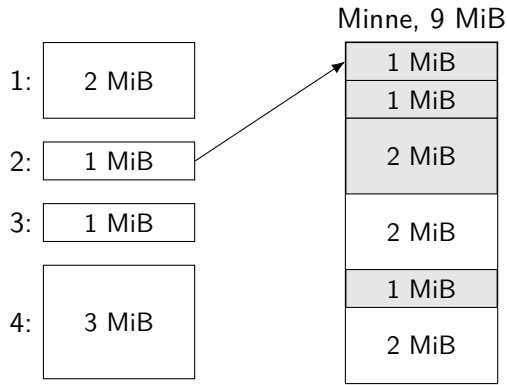


Allokering av block

Finns olika strategier:

- ⇒ *First-fit*
- *Best-fit*
- *Worst-fit*

Notera: Kan också användas för att implementera `new` och `malloc`.

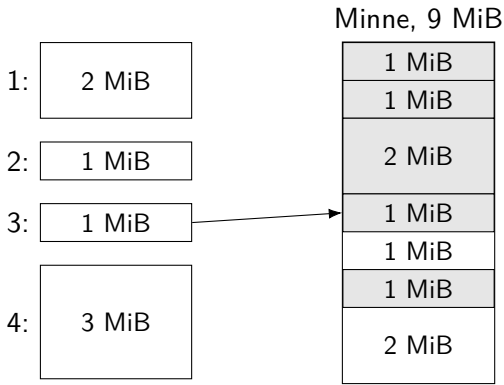


Allokering av block

Finns olika strategier:

- ⇒ *First-fit*
- *Best-fit*
- *Worst-fit*

Notera: Kan också användas för att implementera `new` och `malloc`.



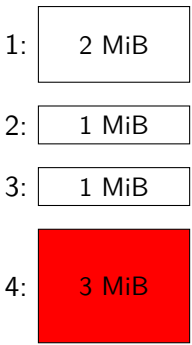
Allokering av block

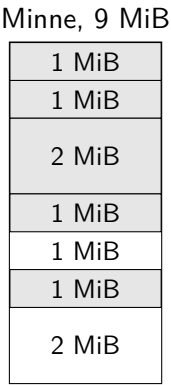
Finns olika strategier:

⇒ *First-fit*

- *Best-fit*
- *Worst-fit*

Notera: Kan också användas för att implementera `new` och `malloc`.





Allokering av block

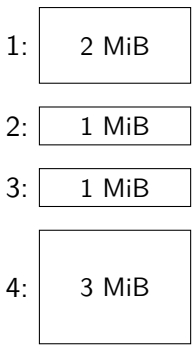
Finns olika strategier:

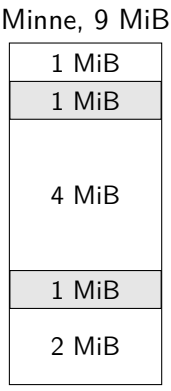
- *First-fit*

⇒ *Best-fit*

- *Worst-fit*

Notera: Kan också användas för att implementera `new` och `malloc`.





Allokering av block

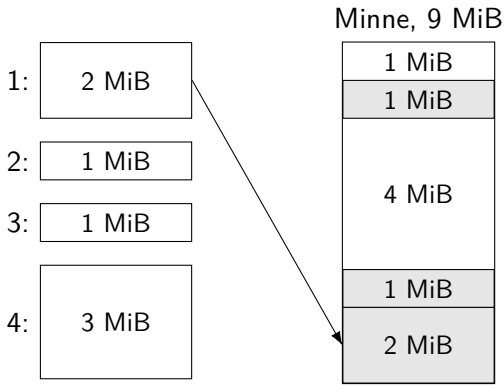
Finns olika strategier:

- *First-fit*

⇒ *Best-fit*

- *Worst-fit*

Notera: Kan också användas för att implementera `new` och `malloc`.



Allokering av block

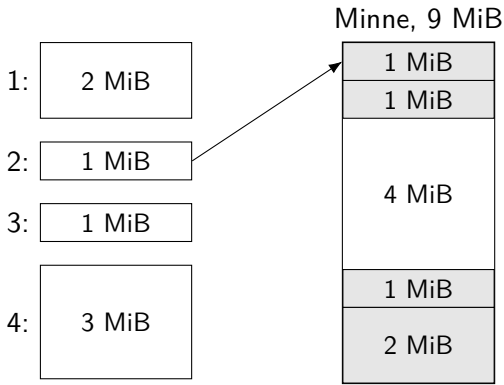
Finns olika strategier:

- *First-fit*

⇒ *Best-fit*

- *Worst-fit*

Notera: Kan också användas för att implementera `new` och `malloc`.



Allokering av block

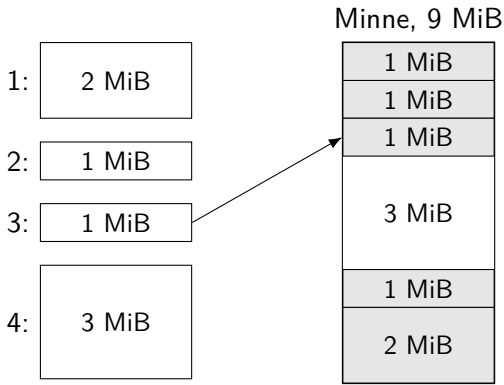
Finns olika strategier:

- *First-fit*

⇒ *Best-fit*

- *Worst-fit*

Notera: Kan också användas för att implementera `new` och `malloc`.



Allokering av block

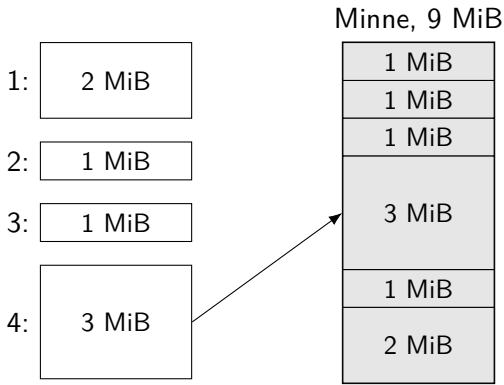
Finns olika strategier:

- *First-fit*

⇒ *Best-fit*

- *Worst-fit*

Notera: Kan också användas för att implementera `new` och `malloc`.



Allokering av block

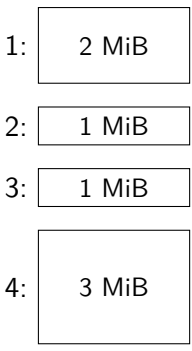
Finns olika strategier:

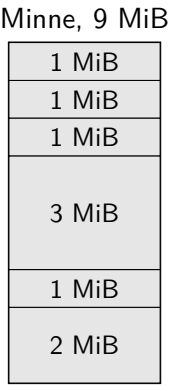
- *First-fit*

⇒ *Best-fit*

- *Worst-fit*

Notera: Kan också användas för att implementera `new` och `malloc`.





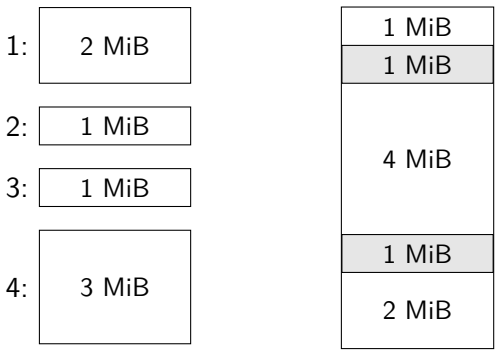
Allokering av block

Finns olika strategier:

- *First-fit*
- *Best-fit*

$\Rightarrow$ *Worst-fit*

Notera: Kan också användas för att implementera `new` och `malloc`.



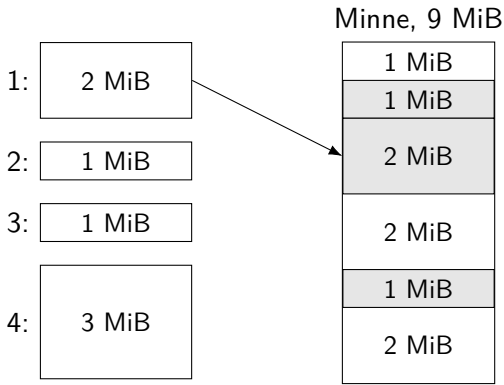
Allokering av block

Finns olika strategier:

- *First-fit*
- *Best-fit*

$\Rightarrow$ *Worst-fit*

Notera: Kan också användas för att implementera `new` och `malloc`.



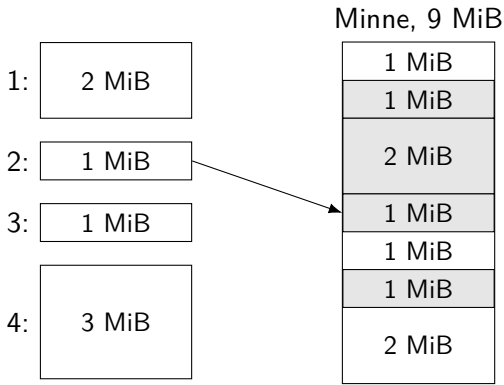
Allokering av block

Finns olika strategier:

- *First-fit*
- *Best-fit*

$\Rightarrow$ *Worst-fit*

Notera: Kan också användas för att implementera `new` och `malloc`.



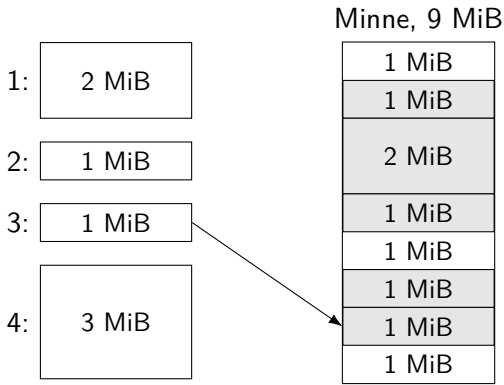
Allokering av block

Finns olika strategier:

- *First-fit*
- *Best-fit*

$\Rightarrow$ *Worst-fit*

Notera: Kan också användas för att implementera `new` och `malloc`.



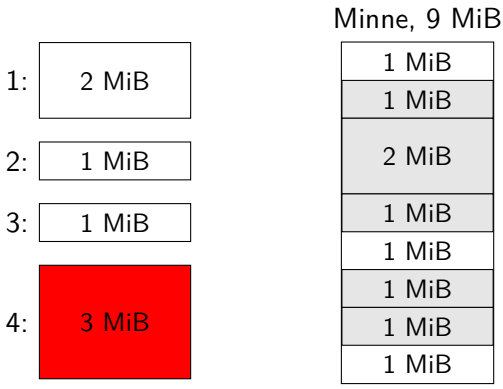
Allokering av block

Finns olika strategier:

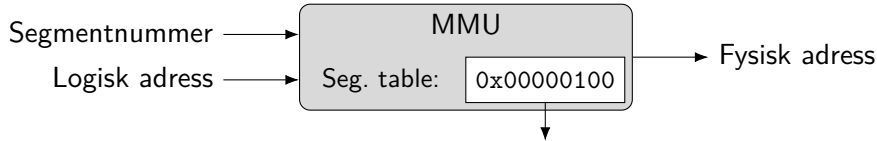
- *First-fit*
- *Best-fit*

⇒ *Worst-fit*

Notera: Kan också användas för att implementera `new` och `malloc`.



Segmentering

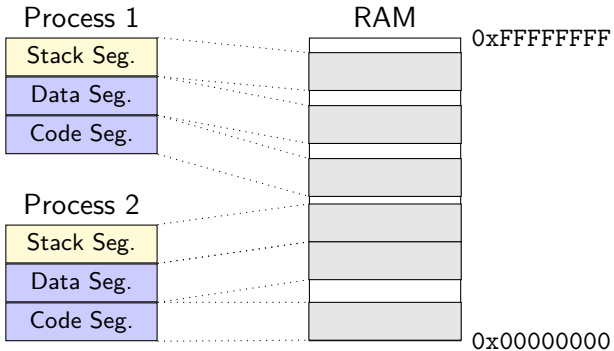


Seg	Base	Limit
0	0x00100000	0x00180000
1	0x00200000	0x00280000
2	0x00180000	0x001A0000

Mindre block ⇒ Mindre risk för fragmentering

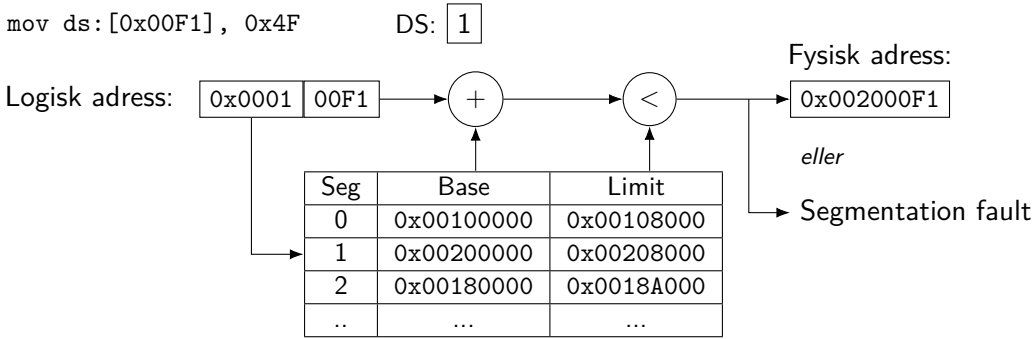
Notera: Varje process har olika segmenttabeller!

Segmentering



Segmentering: Hur ser logiska adresser ut?

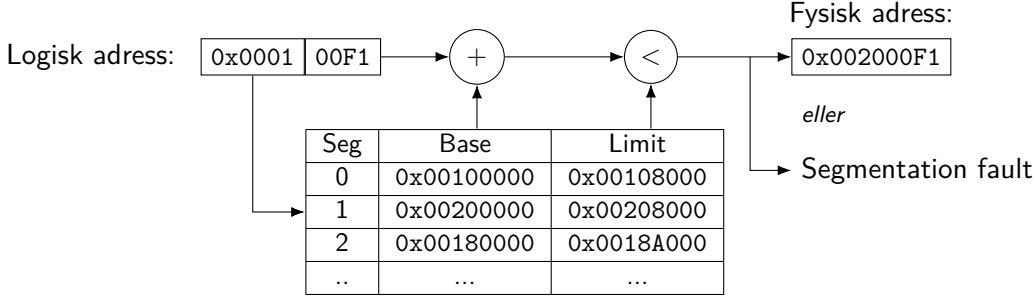
Alternativ 1: Segmentregister (t.ex. Protected Mode i Intel x86)



Segmentering: Hur ser logiska adresser ut?

Alternativ 2: Mest signifikanta bitar i logisk adress

```
mov [0x000100F1], 0x4F
```



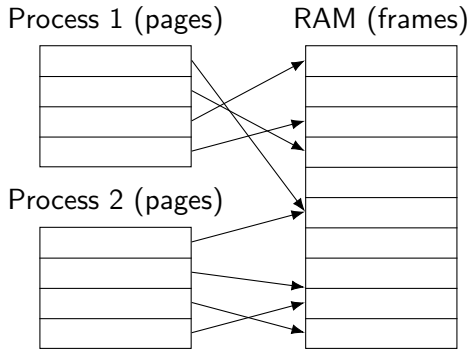
Paging

- Vi vill undvika *extrern fragmentering*
- Idé: Vi delar upp den virtuella adressrymden i *pages*
- Alla *pages* har *samma storlek*

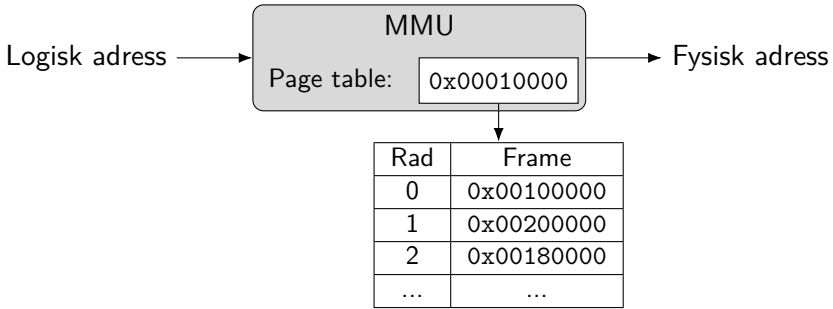
⇒ Vi kan lagra en *page* var som helst i RAM (i vilken *frame* som helst)

⇒ Bara *intern fragmentering* kvar!

- Hur stor ska en *page* vara?



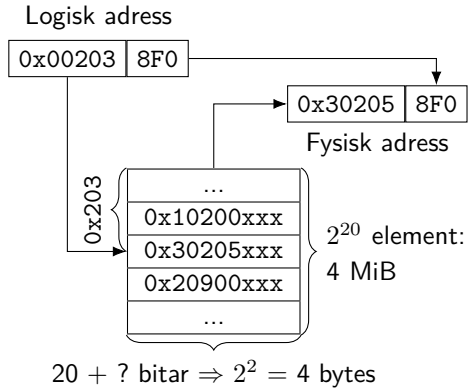
Paging



Olika page tables för olika processer!

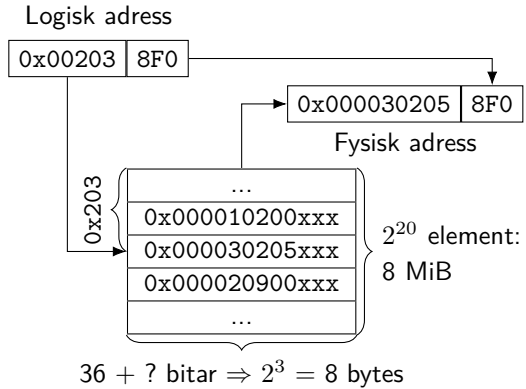
Paging: Hur stor blir page-table?

- 32-bit logiska adresser
- 32-bit fysiska adresser
- 4 KiB page och frame
- Page-table får inte plats i en page ⇒ Fragmentering!
- Kan vi välja en bra page-size?



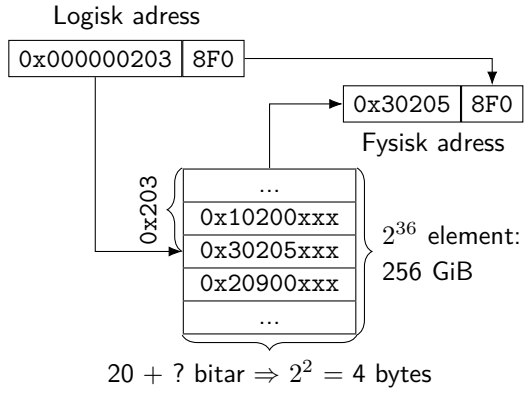
Paging: Hur stor blir page-table?

- 32-bit logiska adresser
- 48-bit fysiska adresser
- 4 KiB page och frame
- Page-table får inte plats i en page ⇒ Fragmentering!
- Kan vi välja en bra page-size?



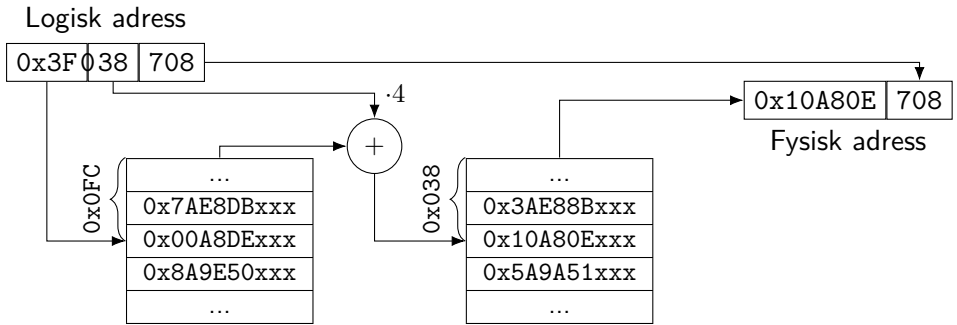
Paging: Hur stor blir page-table?

- 48-bit logiska adresser
- 32-bit fysiska adresser
- 4 KiB page och frame
- Page-table får inte plats i en page ⇒ Fragmentering!
- Kan vi välja en bra page-size?



Hierarkisk Page-table

Idé: Vi lagrar page-table i flera nivåer!



Kostnad av paging

Med 2 nivåer paging:

- Kostar 2 läsningar att slå upp adress
- Varje minnesaccess kostar som 3
- RAM är förhållandevis långsamt...

Lösning:

- TLB (Translation Lookaside Buffer)
- Cache för MMU-beräkningar (64–1024 element)
- Undviker minnesåtkomst om elementet finns

Kostnad av paging

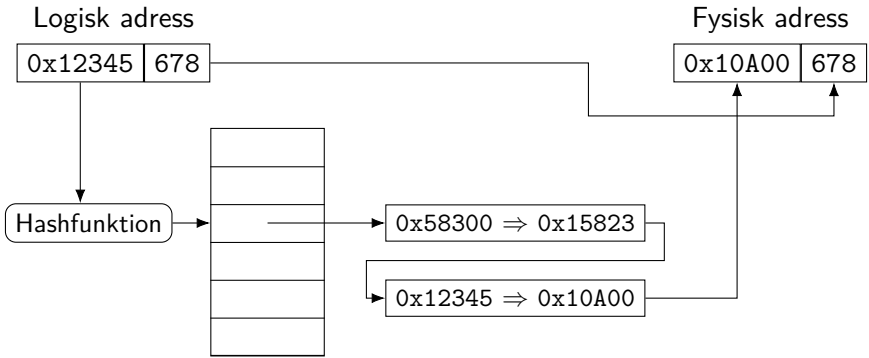
Effective Access Time:
Antag:

- $\alpha =$ hit-ratio, i %
- $t =$ RAM-access
- $\varepsilon =$ uppslagning i TLB
  Litet i förhållande till t

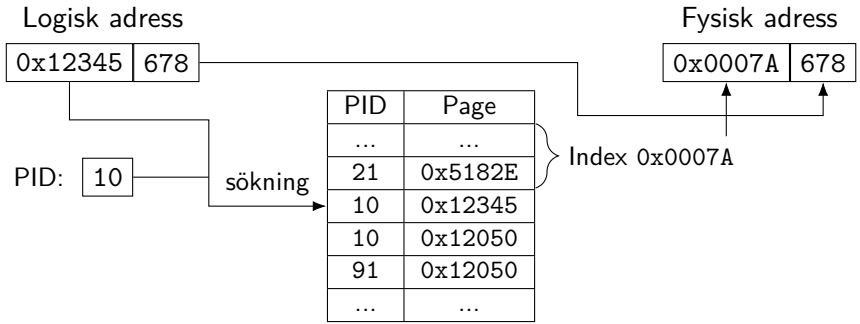
$$\text{EAT} = (t + \varepsilon)\alpha + (3t + \varepsilon)(1 - \alpha)$$

- $\alpha = 80\%$, $t = 100$ ns ger EAT $= 120$ ns
- $\alpha = 99\%$, $t = 100$ ns ger EAT $= 101$ ns

Hashad Page-table

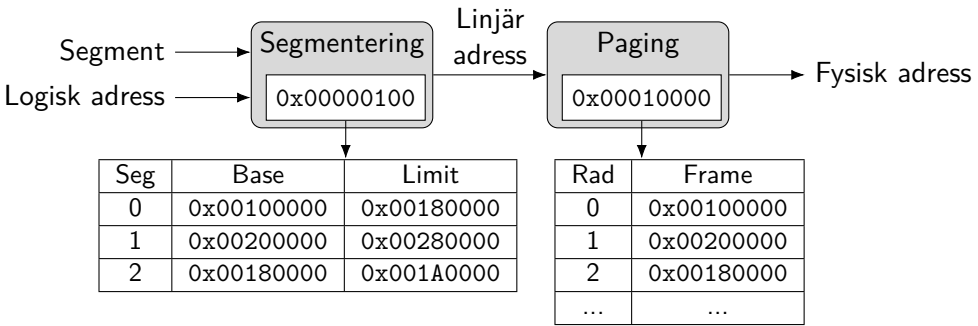


Inverterad Page-table



PT har ett element per *frame*. Svårt att dela minne mellan processer.

Segmentering + Paging (Intel x86)



Seg	Base	Limit
0	0x00100000	0x00180000
1	0x00200000	0x00280000
2	0x00180000	0x001A0000

Rad	Frame
0	0x00100000
1	0x00200000
2	0x00180000
...	...

Nästa vecka

På 64-bitarssystem har vi 256 TiB logisk adressrymd, men mycket mindre RAM.

- Varför är det användbart?
- Hur kan vi använda mer minne än vad som finns tillgängligt?

Filip Strömbäck

www.liu.se

LINKÖPING UNIVERSITY