Tildeling av minne til prosesser

Tildeling av minne til en prosess

- Når en prosess *opprettes* har den et *krav* til hvor mye minne som skal reserveres for prosessen
- Memory Management System (MMS) i OS må da:
  - Forsøke å finne et passende ledig minneområde som matcher kravet fra prosessen
  - Tildele dette området til prosessen
  - Merke minneområdet som “opptatt” slik at det beskyttes
- Hele minneområdet må frigis ved *terminering* av prosessen for å unngå *minnelekkasje*

Noen teknikker for tildeling av minne*

- Enkel, sammenhengende tildeling
- Partisjonert tildeling
- Virtuelt minne med paging
- Segmentert minnehåndtering

*: OS kan også *underveis* i eksekveringen av et program tildele *mer* minne ved behov og *frigi* minne som ikke lenger er i bruk (garbage collection). *Dynamisk* håndtering av minnet i runtime omtales ikke her.

Enkel, sammenhengende tildeling

- Den enkleste formen for minnehåndtering
- En liten del av minnet settes av til å kjøre OS
- Hele *resten* av minnet er bare tilgjengelig for *én* prosess om gangen

Problemer med enkel, sammenhengende tildeling

- Bare én prosess er i minnet hver gang, vanskelig å få til ekte multitasking
- Multitasking *kan* oppnås ved å dumpe hele RAM for nåværende prosess ut på disk, og deretter lese inn hele RAM-innholdet for neste prosess*
- Er for langsomt til å kunne brukes generelt
- Kan brukes i systemer der det oftest bare er én prosess, typisk embedded og real-time systemer

* Eldre Microsoft-systemer for PC (MS-DOS) brukte denne teknikken

Partisjonert tildeling

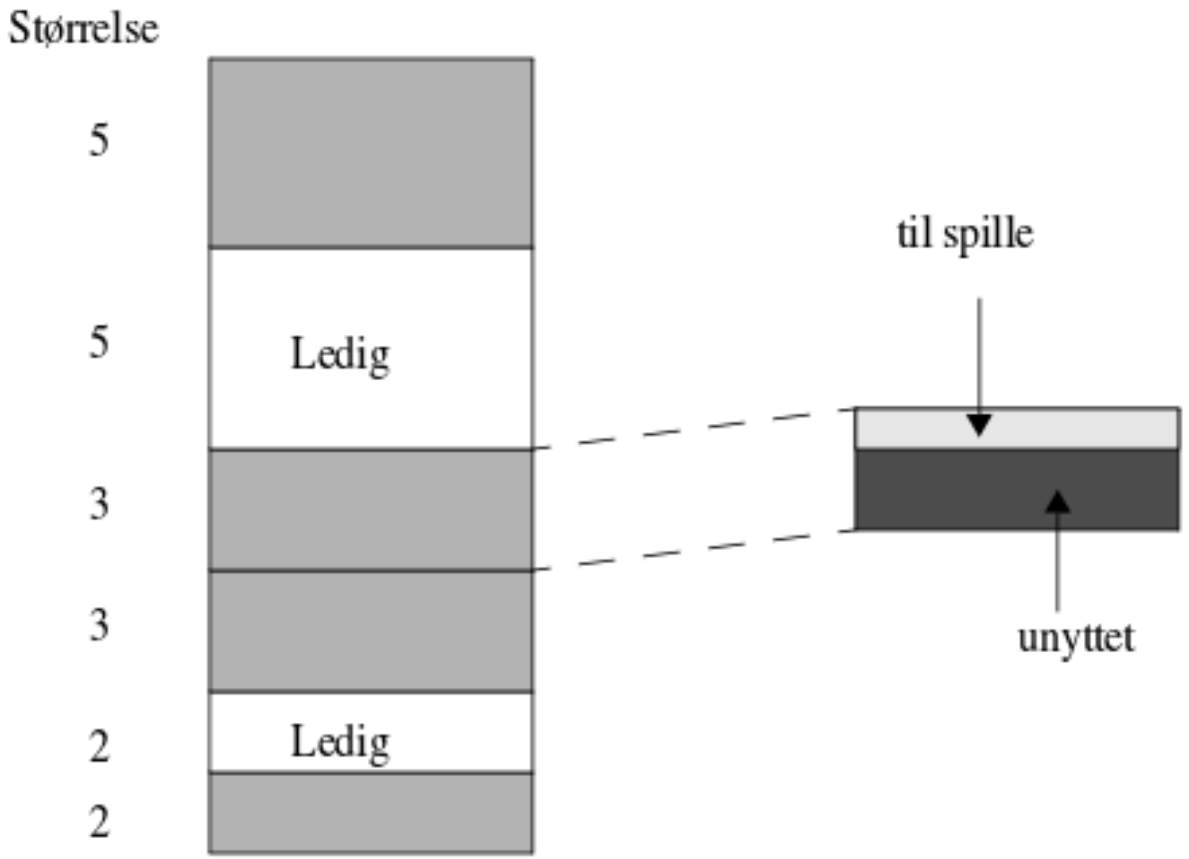
- Alle prosessene deler på det tilgjengelige minnet
- OS tildeler *ett* sammenhengende minneområde til hver prosess
- Hele minnet blir delt opp (partisjonert) i flere minne-områder av ulik størrelse
- To typer partisjonert tildeling av minne:
  1) Statisk (stiv) tildeling
  2) Dynamisk (fleksibel) tildeling

Statisk (stiv) tildeling

- Hele minnet deles opp i *faste* områder av ulik størrelse:
  - Oppdelingen gjøres ved oppstart av maskinen
  - Oppdelingen kan deretter ikke endres (er *statisk*)
- Minne tildeles til nye prosesser med bruk av en “minimum-fit”- strategi:
  - En prosess tildeles alltid *minste* ledige minne-område som er *større* enn minnekravet fra prosessen

Problemer med statisk tildeling

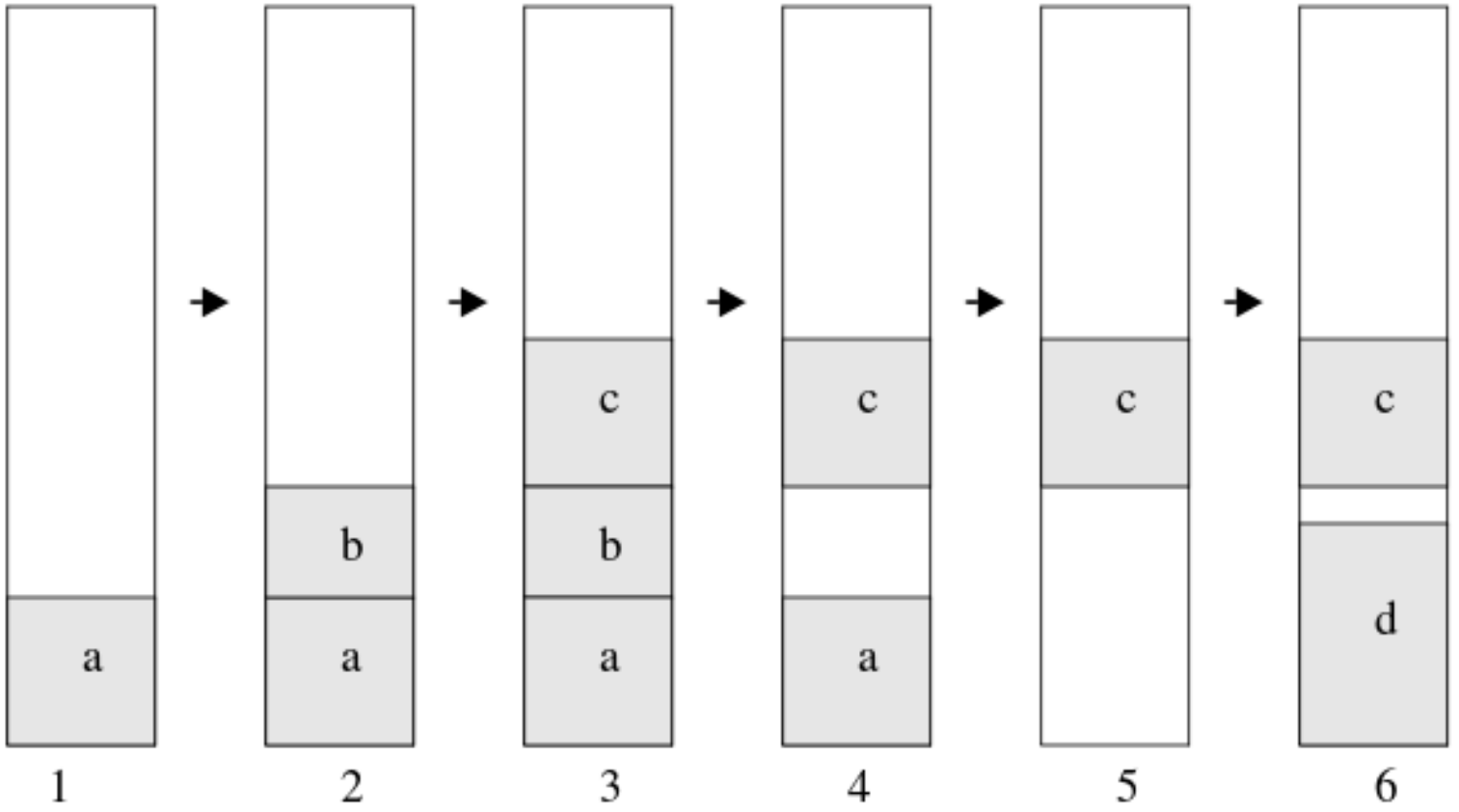
- Sløser alltid (!) med plass i minnet
- Max. antall prosesser = antall minneområder (!)
- Brukes ikke i moderne OS



Organisering av minnet i faste segmenter (stiv tildelingsstrategi). De uskraverte områdene er ledige. Til høyre ser at vi noe minne vil gå til spille fordi segmentet alltid er litt for stort

Dynamisk (fleksibel) tildeling

- Bruker *ikke* faste minneområder
- Tildelingsstrategi til nye prosesser:
  - Finn det *minste* ledige *sammenhengende* minneområdet som er *større* enn krav (minimum-fit)
  - Tildél *nøyaktig* det som prosessen ber om, fra *begynnelsen* av minneområdet som er funnet



Tildeling av segmenter med fleksibel størrelse.

Fordeler med dynamisk tildeling

- Mer fleksibelt enn statisk tildeling
- Ingen fast øvre grense for antall prosesser
- Vil tilpasse seg “automatisk” både til systemer med mange små prosesser, og til systemer med få store prosesser.

Problem med dynamisk tildeling: Fragmentering

- Blir stadig flere ledige *mindre* områder i minnet:
  - Hver gang vi setter av et minneområde blir det en “liten rest” igjen som ikke settes av til prosessen
  - Minnet deles opp i flere og flere små *fragmenter**
- Prosesser som krever mye minne kan måtte avvises:
  - Finnes ikke stort nok sammenhengende minneområde
  - Men *summen* av de små ledige minneområdene kan allikevel være *større* enn det prosessen krever

*: Fragmenteringsproblemet oppstår også på disker, når store filer blir spredt rundt i mange små områder

Mulig løsning: Defragmentering

- Problemet med fragmentering av minnet *kan* løses ved å gjøre *defragmentering*
- Går jevnlig gjennom hele minnet og finner alle ledige minneområder under en viss størrelse
- Slår sammen disse områdene til større områder ved å flytte rundt på alt som er lagret i minnet

Problemer med defragmentering

- Defragmentering er en tung prosess
- Krever relativt mye kjøretid
- Medfører endringer i tilstand for alle prosessene
- Systemet “fryser” mens defragmenteringen pågår
- Brukes for disker, men er *ikke* vanlig brukt i minnehåndtering

Bedre løsning: Paging

- Deler opp hele primærminnet i “småbiter” av fast størrelse
- Hver “småbit” kalles for en “page frame”
- Typisk størrelse er 4096 Bytes = 4 KByte
- Tilsvarer den minste mengden av data som kan flyttes mellom RAM og disk under skriving/ lesing av data

Minnetildeling med paging

- Minne tildeles som et antall hele “pages”
- Minneområdet til en prosess blir liggende *spredt* i minnet
- Fragmenteringsproblemet løses ved å bruke ledige “småbiter” av RAM til å sette sammen hele minneområdet
- Paging brukes alltid sammen med *virtuelt minne*

Virtuelt * minne

- Hver prosess tildeles et sammenhengende *tenkt* (virtuelt) minneområde – en lineær tabell med minne som kan indekseres fra start til slutt
- Prosessen aksesserer *bare* sitt eget minne ved å bruke *virtuelle minneadresser*, som er indekser i denne tabellen
- OS håndterer virtuelle minneadresser og tilordner dem til korrekte adresser i det *fysiske* primærminnet
- Egen *maskinvare* – memory management units – gjør automatisk *oversettelse* fra virtuell til fysisk minneaddresse

* *virtual* (in computing): not physically existing but made by software to appear to do so

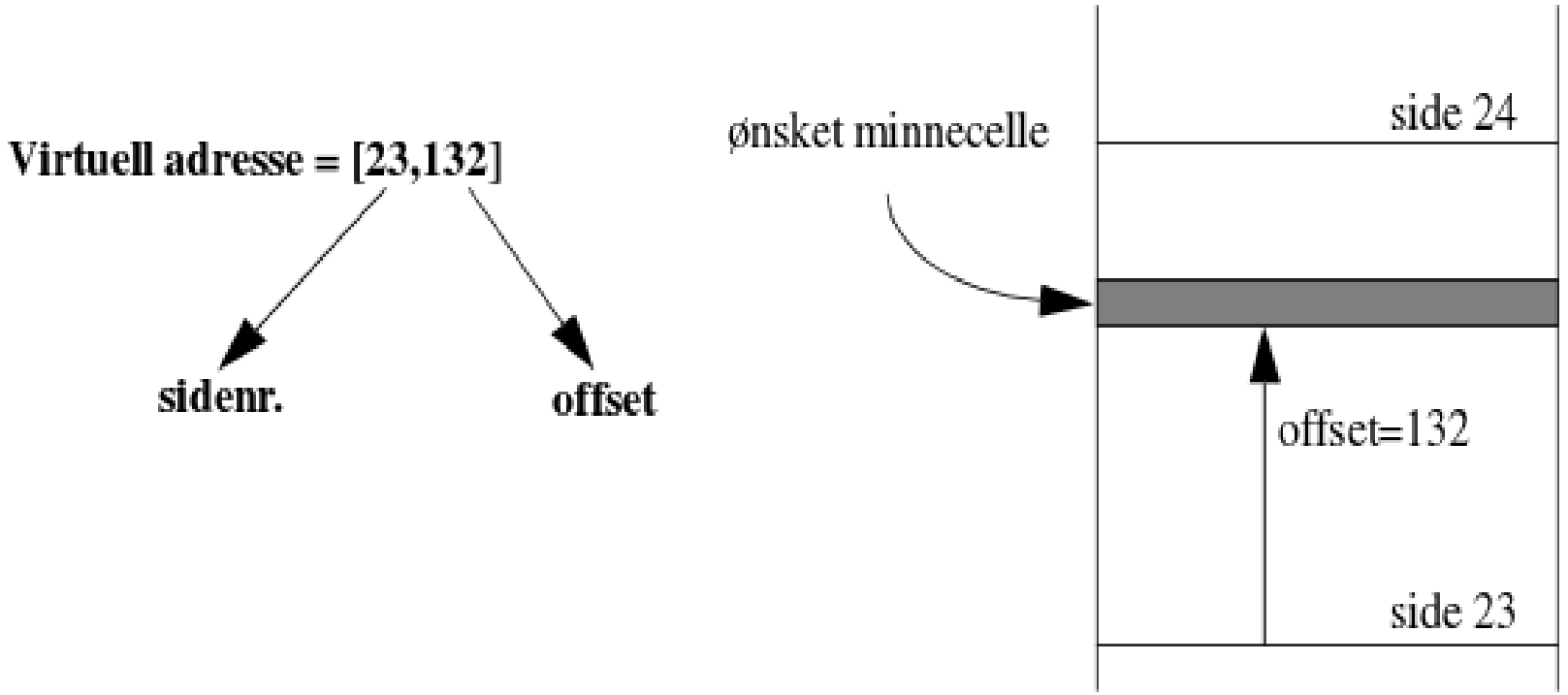
Noen fordeler med virtuelt minne

- Alle prosesser forholder seg kun til et eget (virtuelt) lineært adresserom, enklere for programmerere
- OS håndterer og skjuler den fysiske oppdelingen av minnet i for applikasjonene
- Bedre sikkerhet, prosessene “kjenner ikke til” minnet utenfor eget tildelt virtuelt minneområde

Virtuelt minne og paging

- Virtuelle minneområder deles inn i *pages* (“sider”)
- Page: En sammenhengende blokk med virtuelt minne*
- Det virkelige primærminnet deles opp i *page frames* (“siderammer”) – et område av det fysiske minnet som kan lagre inneholdet i én enkelt virtuell page
- Hver virtuell page tilordnes en fysisk page frame
- OS-ets *minnestyring* sørger for at en prosess som adresserer en virtuell page får lest fra/skrevet til riktig page frame

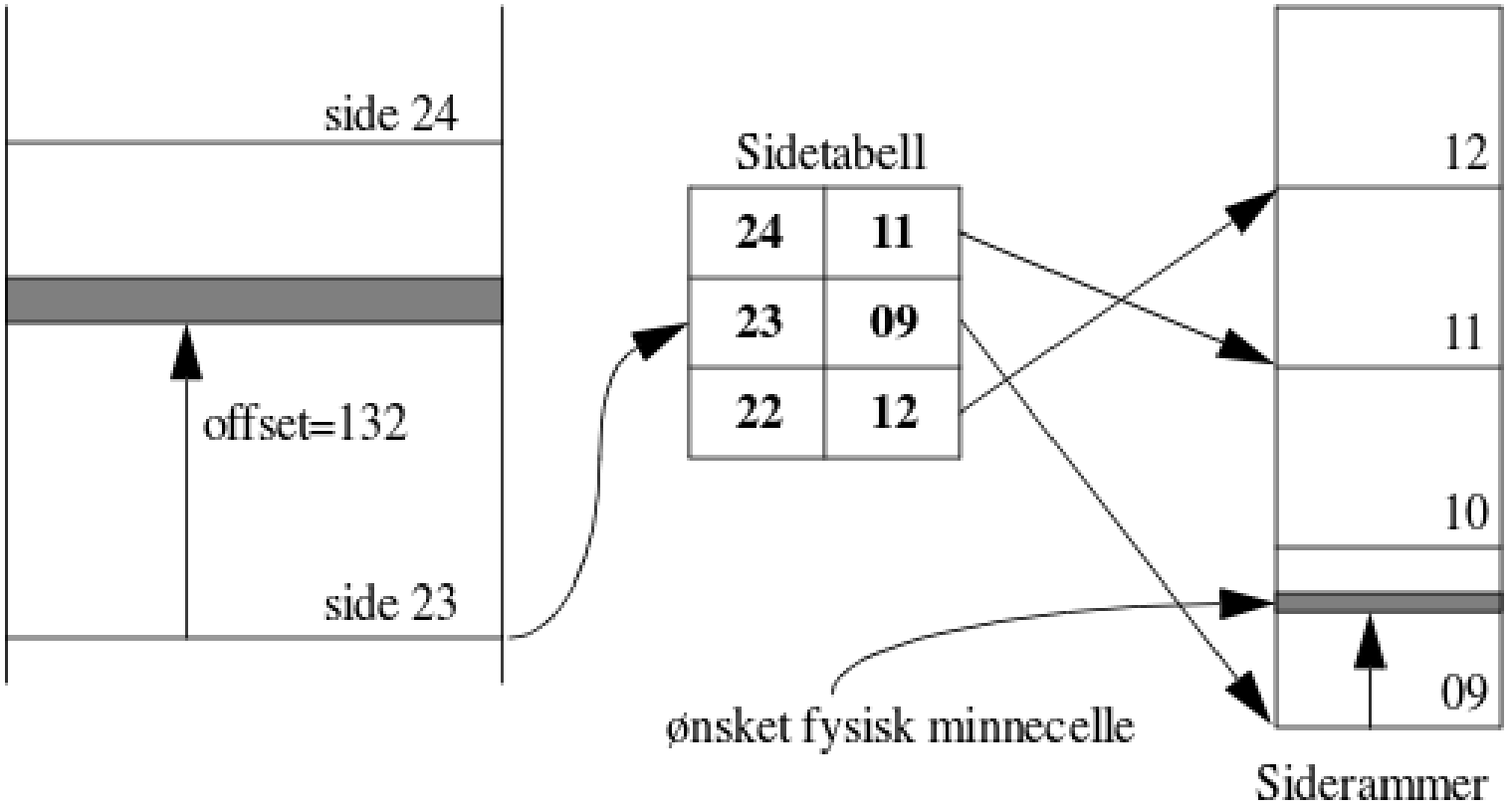
*: Størrelsen på en page er typisk 4K på Linux-systemer



En virtuell adresse og en virtuell minnecelle

Page tables (“sidetabeller”)

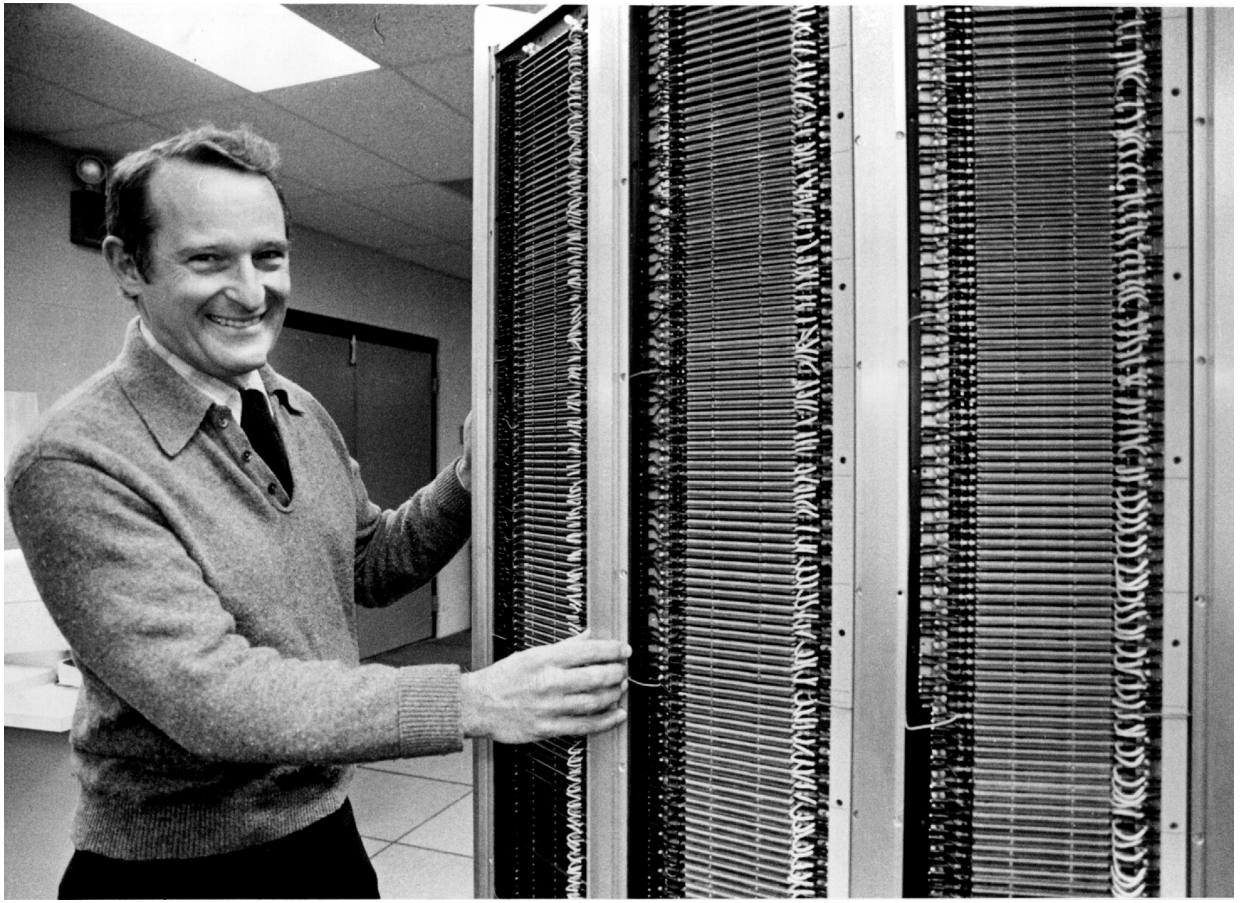
- Brukes av MMS for å oversette korrekt fra prosessenes virtuelle minneadresser til fysiske adresser i primærminnet
- Hvert element i tabellen lagrer informasjon om én virtuell page
- OS kan ha én stor pagetabell, én tabell for hver prosess, én for hvert segment av RAM, eller kombinasjoner av dette



Bruk av sidetabell. Innholdet i den skraverte minnecellen i det virtuelle minnet (til venstre) ligger lagret i den skraverte minnecellen i det fysiske minnet (til høyre).

Virtuelt minne er “ubegrenset”

- Alle maskiner har et begrenset fysisk primærminne
- Typisk 8-32 GB RAM for en relativt standard PC/Mac
- Operativsystemet kan håndtere *mer* virtuelt minne enn det fysisk er plass til i RAM
- Gjøres ved å lagre virtuelt minne som det ikke er plass til i fysisk RAM på sekundærminne (typisk en disk) i stedet

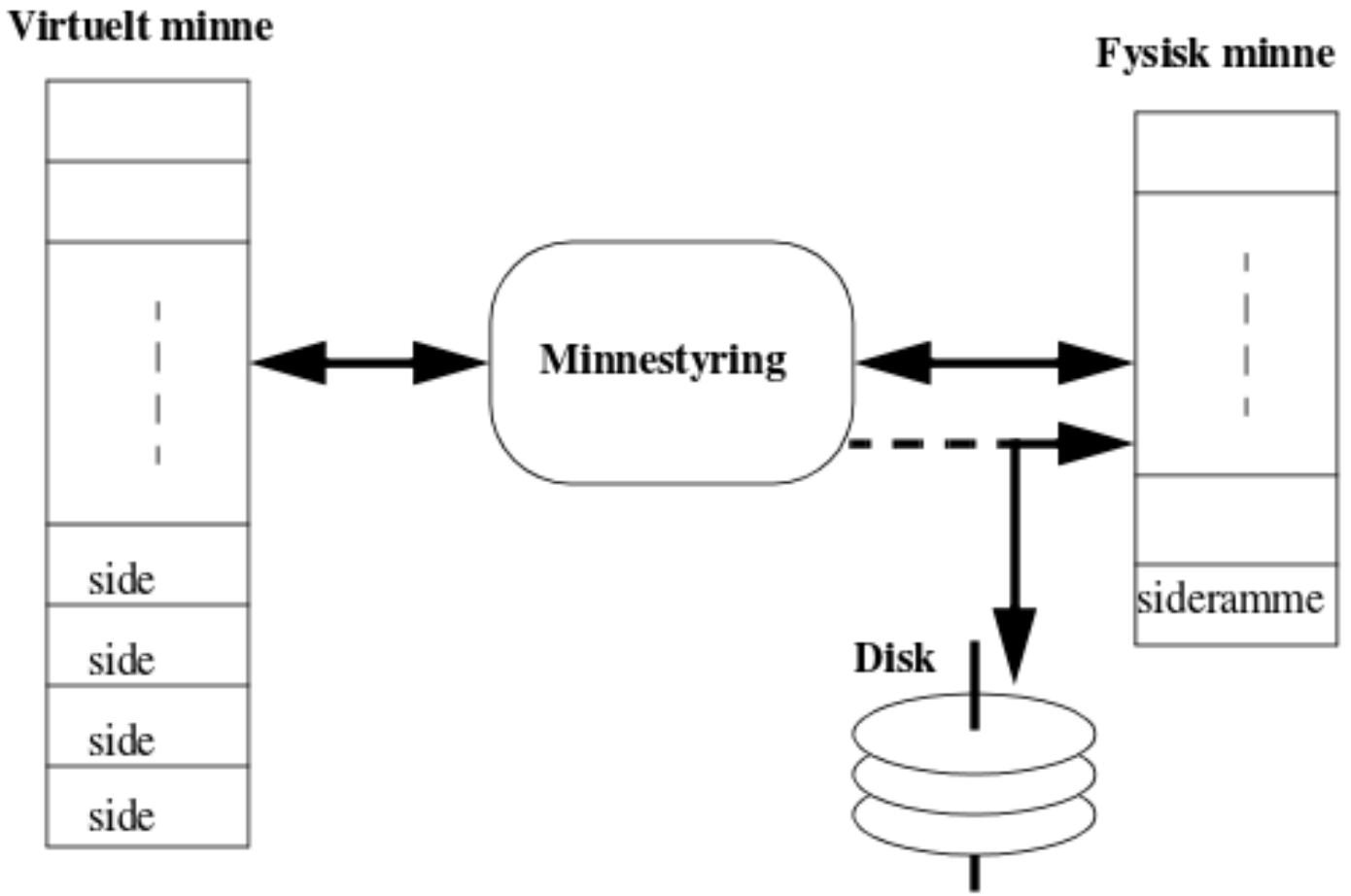


"Memory is like an orgasm. It's a lot better if you don't have to fake it."

– *Seymour Cray*, *“the father of supercomputing”, commenting on virtual memory*

Paging og swapping

- Hvis primærminnet ikke har *plass* til alt det virtuelle minnet som er satt av til prosessene, bruker MMS *disken* til midlertidig å lagre innholdet av RAM
- Det er alltid hele page frames med RAM som lagres
- En page i virtuelt minne vil lagres i sin helhet fysisk *enten* i en page frame i primærminnet *eller* på disk.
- Prosessen med å lese/skrive en page frame med RAM til/fra disk kalles for *swapping*



Sammenhengen mellom sider, virtuelt minne og fysisk minne

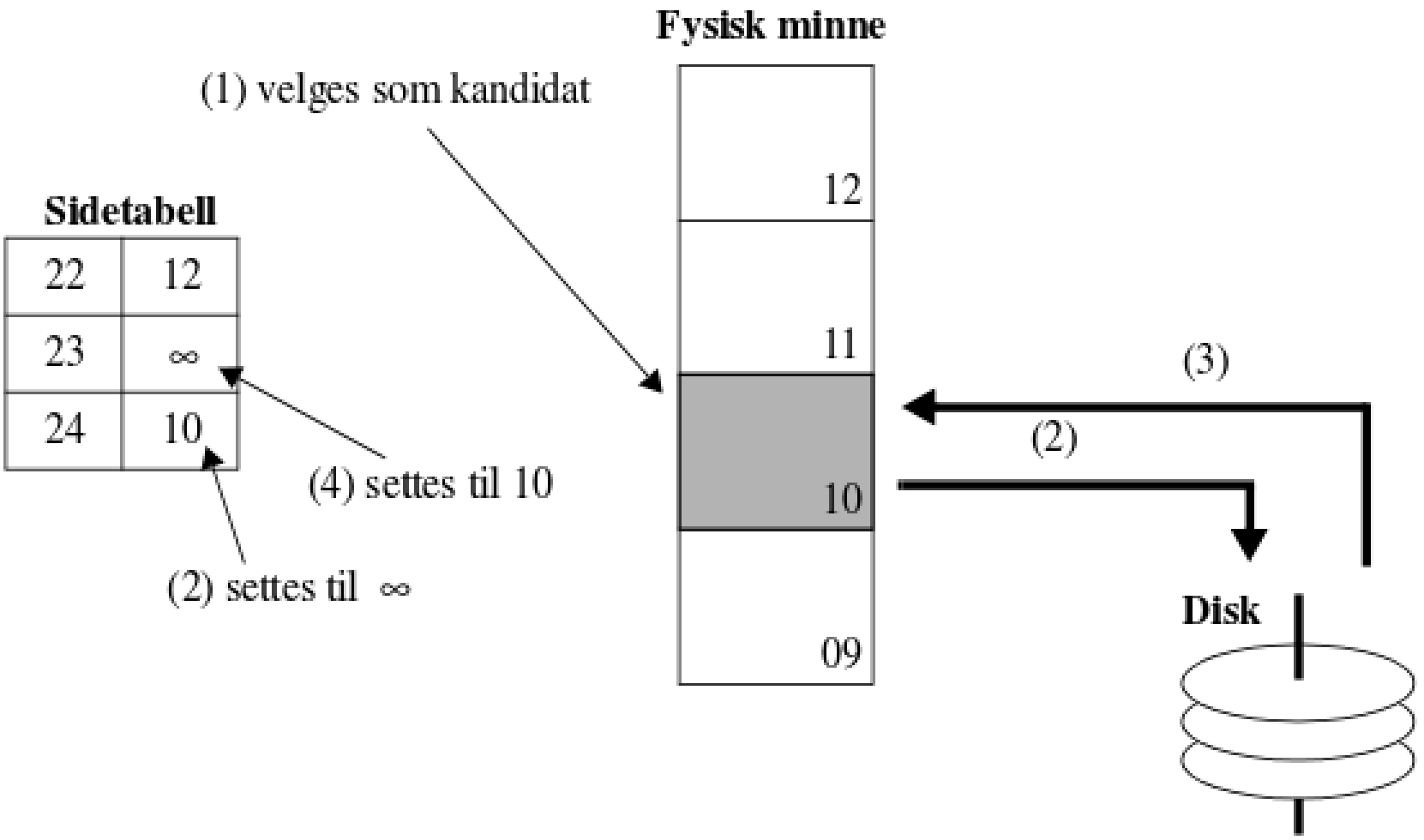
Page faults

- Pagetabellen lagrer også info. om status: En page ligger *enten* i RAM *eller* den er swappet ut til disk
- Hvis en page er i RAM, inneholder pagetabellen den fysiske RAM-adressen til page frame
- Hvis en page ikke er i RAM, vil lesing/skriving generere et avbrudd som kalles *page fault* :
  - Starter en avbruddsrutine som henter riktig page frame fra disk og lagrer den korrekt i RAM
  - Er *mye* mer tidkrevende enn å lese data fra RAM

Håndtering av page faults

- OS leser *fra* disk og *inn i* RAM den page frame som inneholder ønsket virtuell page
- Pagetabellen oppdateres med den fysiske RAM-adressen til innlest frame
- Hvis det ikke er plass i RAM til pageframe som skal leses fra disk, må det gjøres en “page replacement” *
  - En page i minnet velges ut for “replacement”
  - Innholdet i utvalgt page skrives til disk
  - Ønsket page leses fra disk og erstatter den som nettopp er skrevet ut

*: På norsk: “Skifte av sideinnhold”



Skifte av sideinnhold mellom disklager og sideramme

Page replacement algoritmer

- OS må ha en strategi – en algoritme – for å finne ut *hvilken* page i RAM som skal flyttes ut på disk når minnet er fullt og det må gjøres en “page replacement”
- Algoritmen må prøve å minimere antall (tidkrevende) page faults, og bør selv også være rask og effektiv
- Mulige valgstrategier:
  - Page “lengst i minnet” tas ut (First-In-First-Out)
  - “Minst brukte page” tas ut (krever “bokholderi”)
  - Prøv å *forhindre* nye page faults (“ta med naboer” etc.)

Trashing

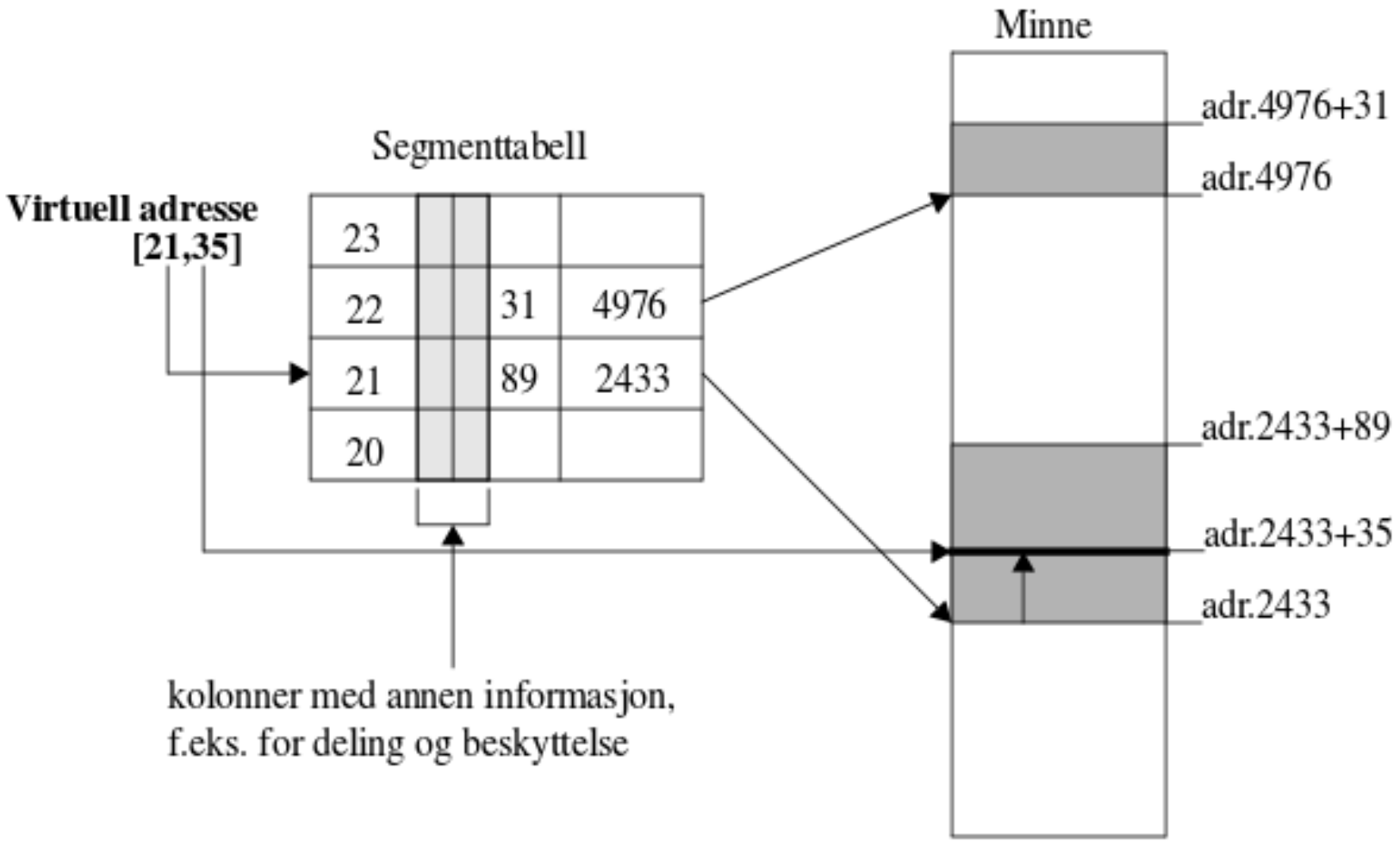
- Mange page faults kan gi en “flaskehals” i systemet
- “Trashing”:
  - For mange prosesser bruker for mye RAM
  - Maskinen bruker alt for mye tid på å overføre page frames frem og tilbake mellom RAM og disk
  - Systemet blir svært tregt eller ubrukelig
- Trashing kan unngås ved å:
  - Sette en øvre grense for antall prosesser
  - Kjøpe mer RAM...

Segmentert minnehåndtering

- *Gir ikke* en prosess *ett* sammenhengende minneområde
- Det virtuelle minnet består i stedet av *flere* sammenhengende områder av ulik størrelse, *minnesegmenter*
- En prosess kan ha *vilkårlig mange* segmenter
- Et segment knyttes ofte til en logisk samling av data:
  - Koden som svarer til en metode i et program
  - Data som ligger lagret i en tabell/array i et program
- Minnesegmenter kan håndteres indivividuelt mht. tildeling, (*bedre*) beskyttelse, deling og frigivelse

Implementasjon av segmentert minne

- Prosessen ser minnet i form av en *segmenttabell*, som for hvert minnesegment inneholder:
  - Starten på segmentet i minnet
  - Lengden på segmentet
  - Tilgangsinformasjon (delt/beskyttet...)
  - Statusinformasjon (i RAM / swappet ut til disk etc.)
- Segmentert minne kan implementeres både med og uten bruk av paging
- Moderne CPU-er støtter segmentert minne i *hardware*



Bruk av segmenttabell. Kolonnen til høyre peker til starten på det angjeldende segmentet, og kolonnen nest lengst til høyre angir lengden på segmentet.

Segmentert minne med paging

- Segmenttabellen inneholder adresser i det virtuelle minnet
- Disse virtuelle minneadressene brukes igjen av systemet i en egen pagetabell for hvert segment, som for vanlig virtuelt minne
- Segmentering blir “bare” et ekstra lag med abstraksjon i minnehåndteringen
- Paging og page faults håndteres på samme måte som for vanlig virtuelt minne
- Vanlig løsning i moderne OS og maskinvare

Minnehåndtering i Linux for Intel-CPU

- Standard Linux bruker:
  - Segmentert minnehåndtering med paging
  - 4096 bytes pages
- Page replacement algoritme:
  - Hver page frame har en verdi (en “age”) som indikerer hvor ønskelig det er at den forblir i RAM
  - “age” oppdateres jevnlig av MMS
  - “age” øker ved mye aksess til page frame, avtar ellers
  - Velger pages med lav “age” for swapping

Kommandoer for å se minnebruk i Linux

- Hele systemet:

```
cat /proc/meminfo
top
free -m
vmstat
ps aux --sort -rss
gnome-system-monitor *
```

- Enkeltprosesser:

```
ps v PID
cat /proc/PID/maps
```

*: Fint verktøy for f.eks. å overvåke stresstesting av et OS

En stresstest av Linux

- Utsetter systemet for ekstrem/unormal belastning
- Vil se om minne-, prosess- og feilhåndtering er robust
- Bruker et program som krever svært mye minne:
  - En Java-implementasjon av Ackermanns funksjon
  - Funksjonen har ekstremt dyp rekursjon og brukes bl.a. i standard testing (“benchmarking”) av kompilatorer
- Kjører shellscript som starter programmet “hele tiden”
- Observerer systemet etterhvert som prosesstabellen og minnet blir fullt, og alle CPU’ene har maks belastning *

*: Med bruk av `gnome-system-monitor`