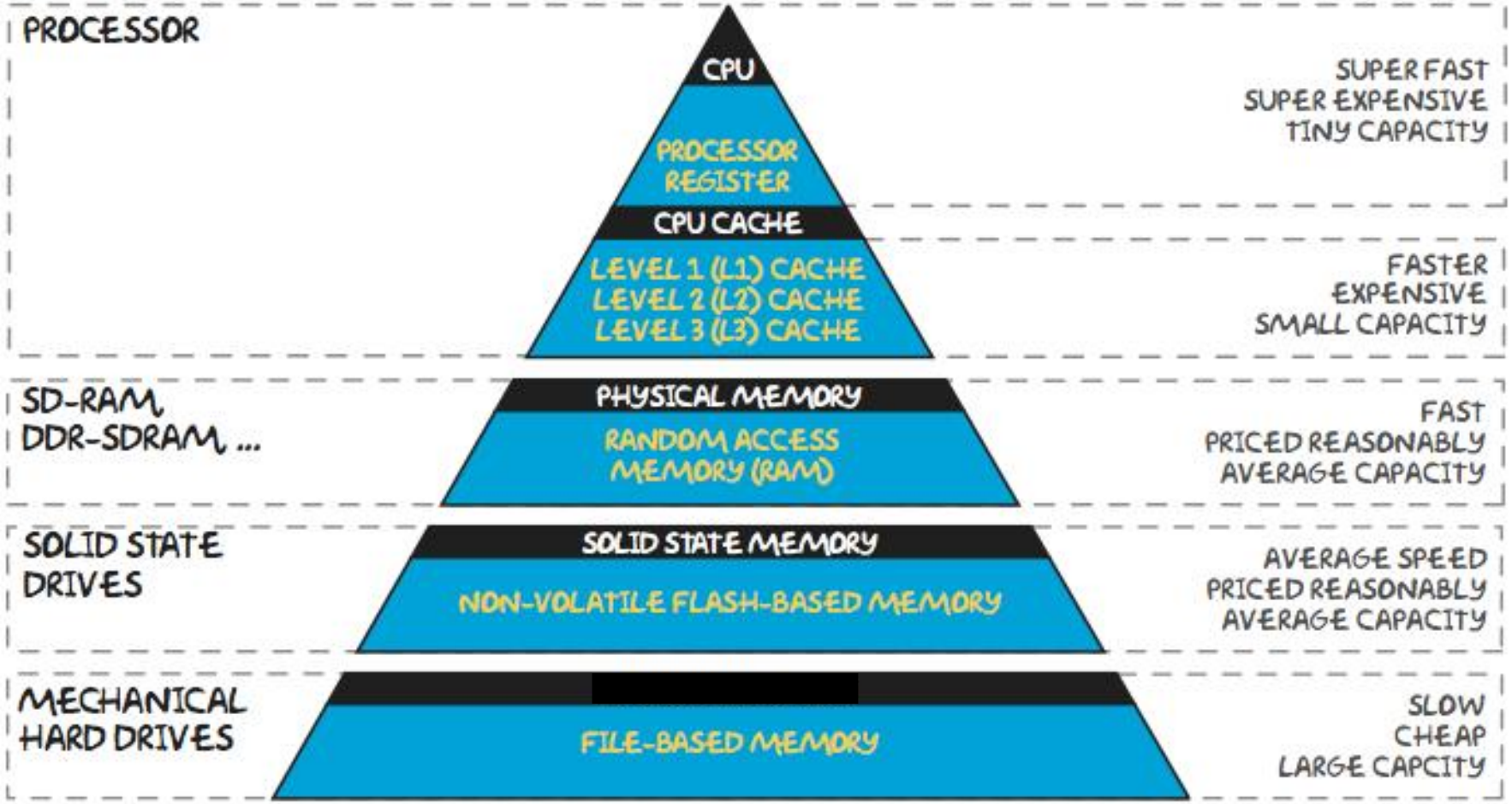
DATORARKITEKTUR

Minne

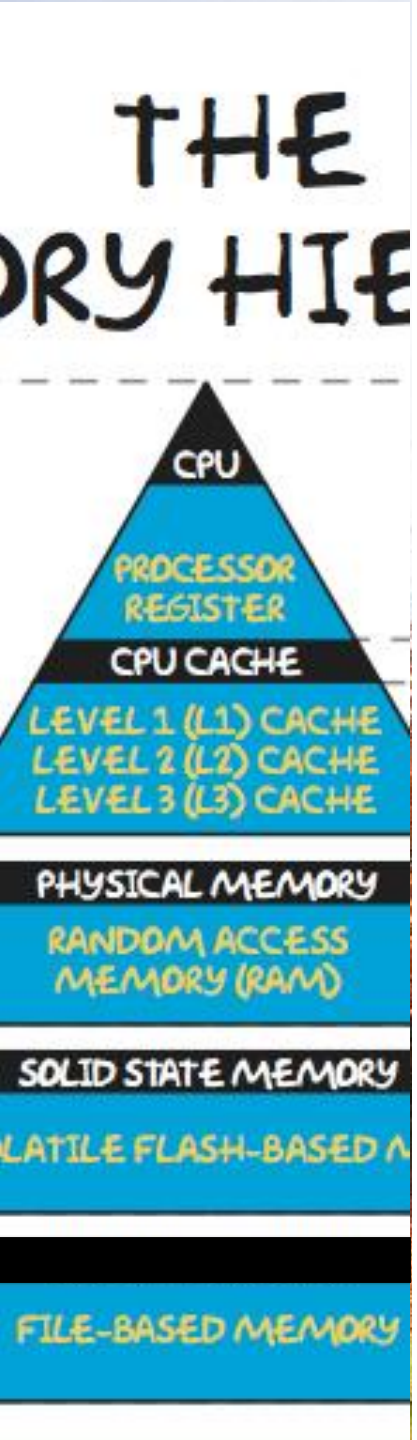
HÖGSKOLAN VÄST

Dr. Andreas de Blanche
andreas.de-blanche@hv.se

THE MEMORY HIERARCHY



Dr. Andreas de Blanche
DAK101, Minne, 2022



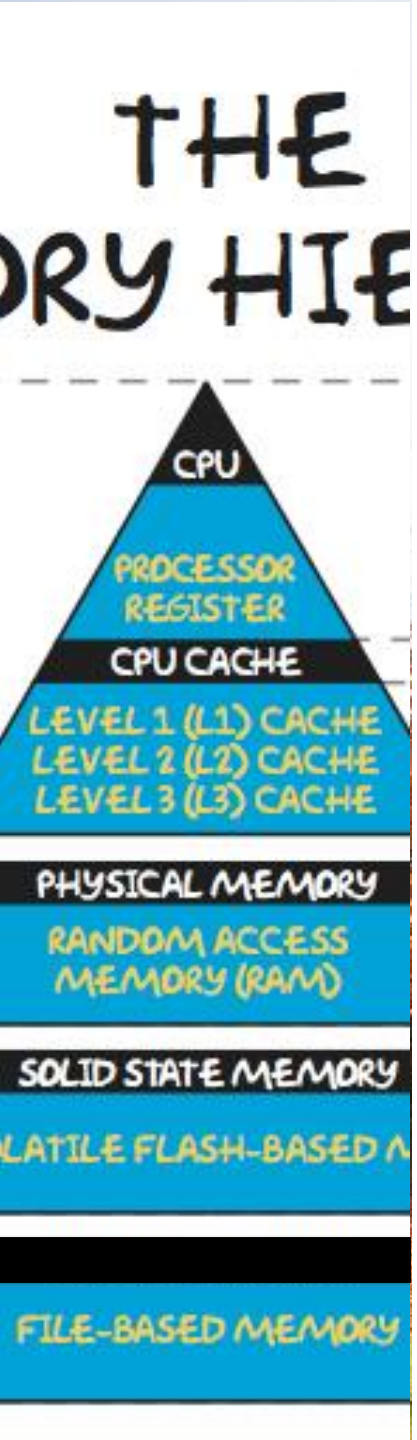
CACHE

HÖGSKOLAN VÄST

- Hur ofta finns den data processor behöver i cachen?

Väldigt ofta

- Data som just använts återanvänds
- Minnesaccesser sker ofta till samma minnesområden
- Minnet läses ofta sekventiellt



CACHE LINE

HÖGSKOLAN VÄST

- Data hanteras i "*cache lines*"
  - En cache line i Raspberry Pi är 32 byte (ARM11)
  - En cache line i en x86-64 är 64 byte
- Det sparas även en *cache tag* per *cache line*.
  - Innehåller adressen
- Sedan sparas det även två extra bitar
  - valid bit
  - dirty bit

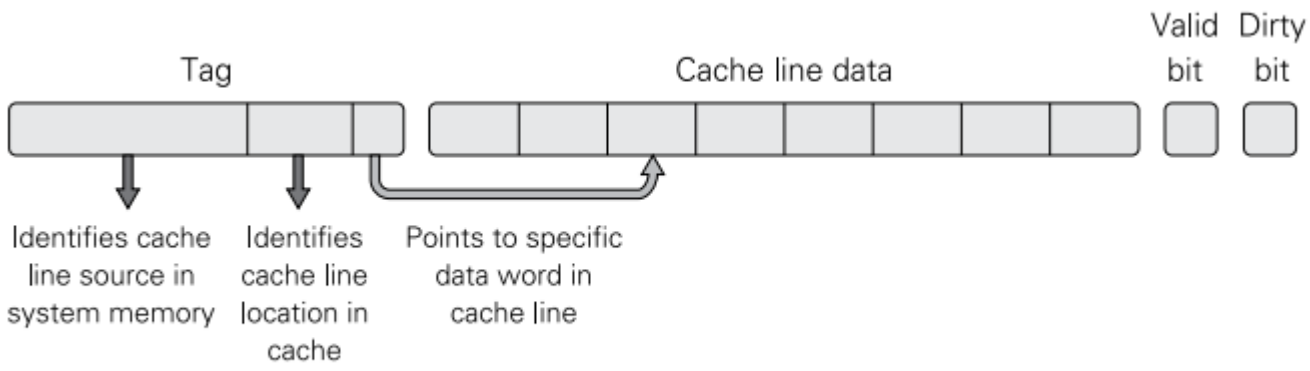
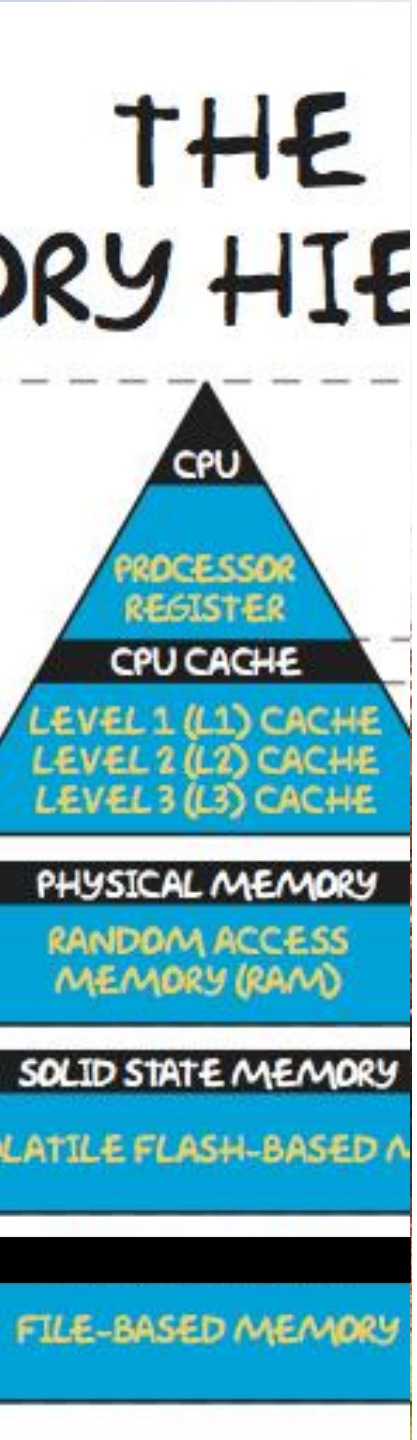


FIGURE 3-9: Cache line structure



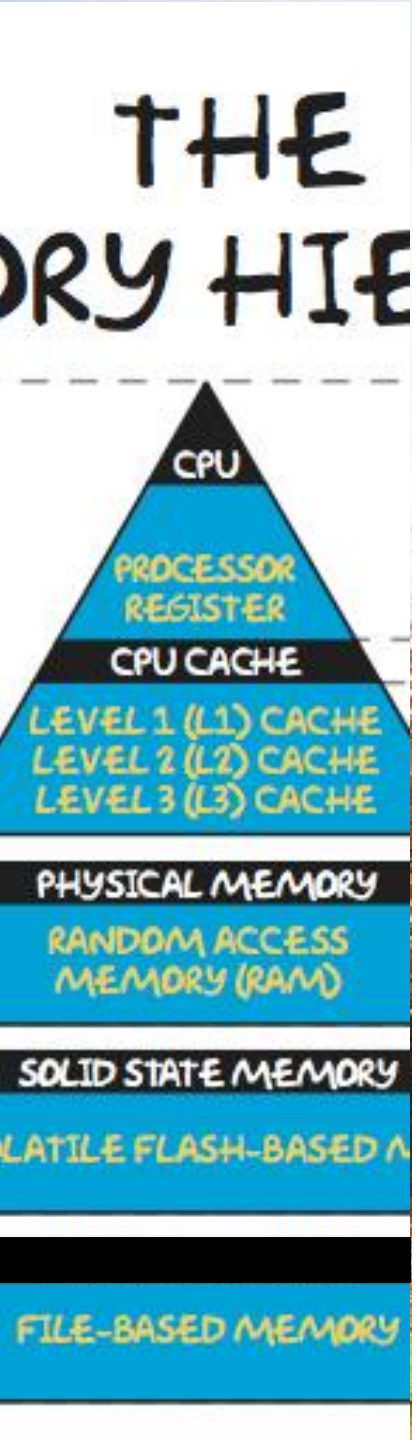
CACHE MAPPING

HÖGSKOLAN VÄST

Cachen är mycket mindre än arbetsminnet.

- Raspberryn har 16,384 byte L1 cache
- 32 byte stora cache lines
- Alltså kan 512 cache lines sparas i L1 cache samtidigt

Detta ställer till ett problem!



Hur vet vi om en cache line finns i L1?

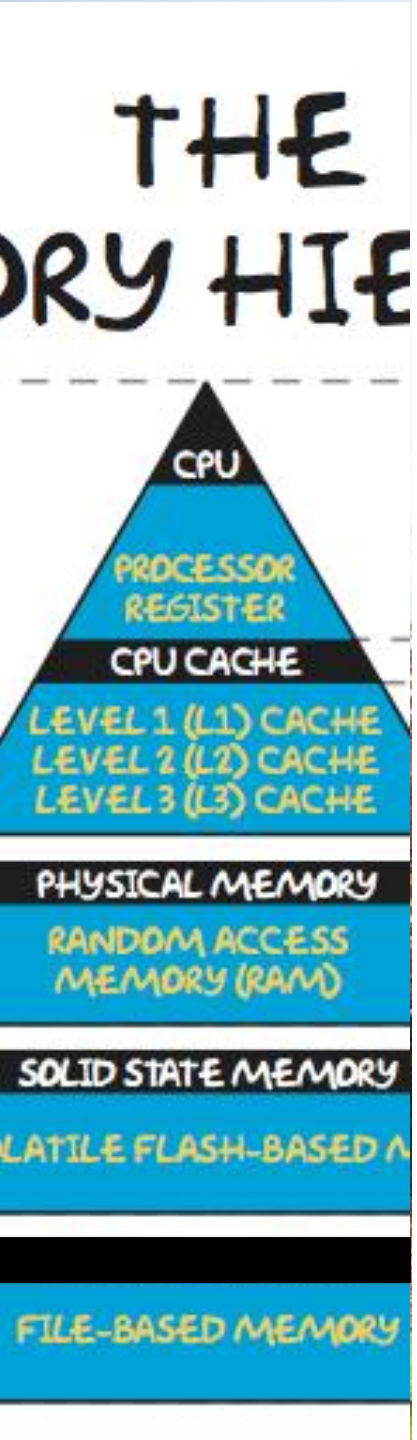
HÖGSKOLAN VÄST

- Spara den på en särskild plats.
- Söka efter den.

Det enklaste sättet är att den fysiska minnesadressen avgör var i L1 som cache linen kan finnas. (direct mappning)

Ram minnet är 1GB stort

Det finns alltså 65 536 ramar (cache lines) per L1 cache position (512 st).



65K PER CACHE ADRESS

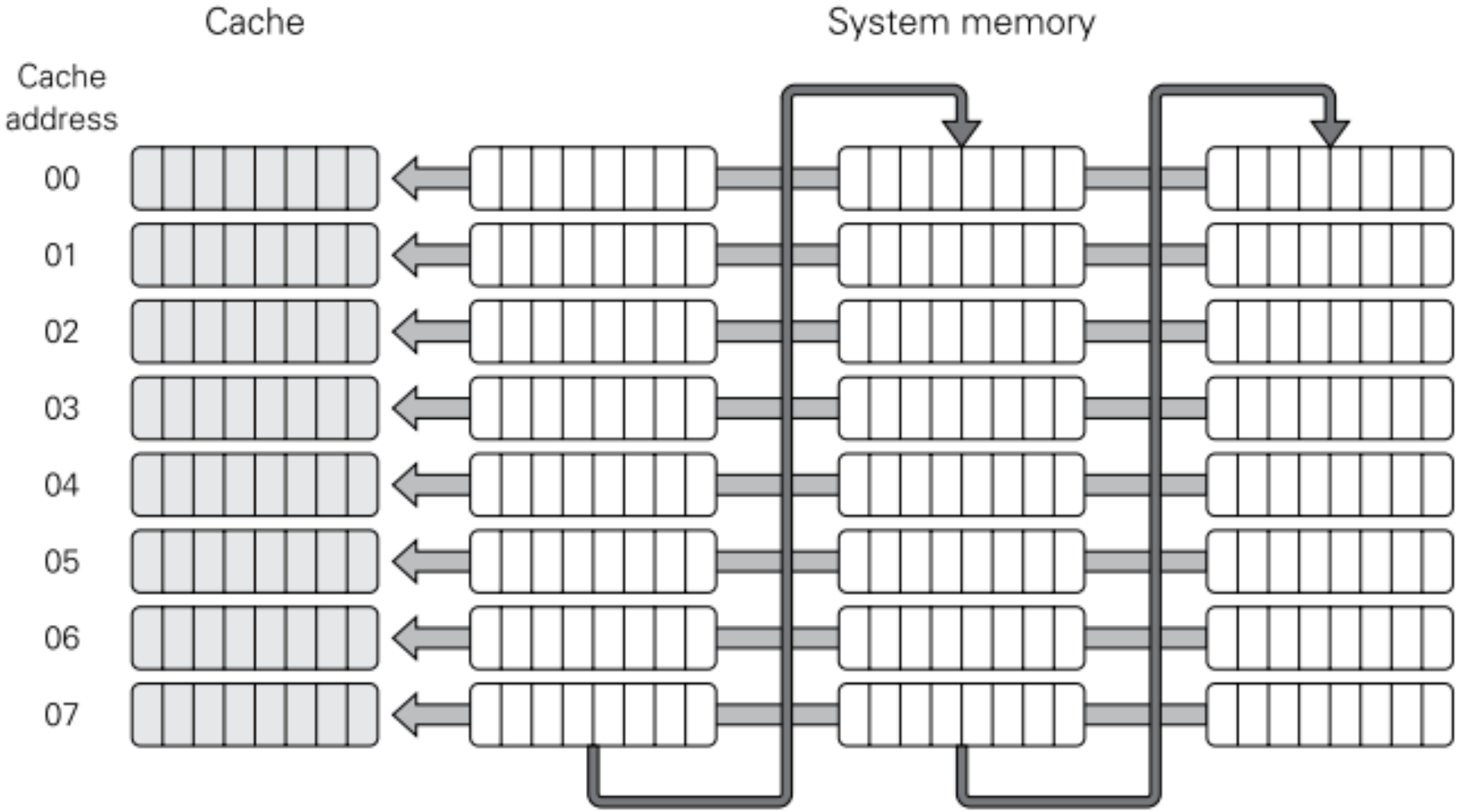
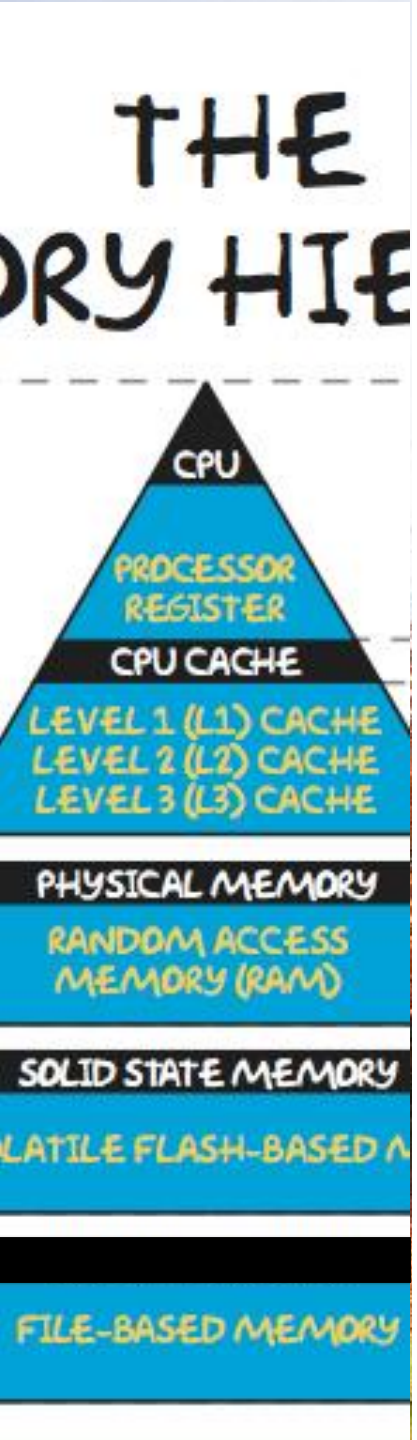


FIGURE 3-10: Direct cache mapping

Att räkna ut rätt position går superfort, *modulo n*.



ASSOCIATIV MAPPNING

HÖGSKOLAN VÄST

Om en cache line kan sparas i vilket block som helst.

Då används utbytesalgoritmer.

- First in first out (FIFO)
- Least recently used (LRU)
- Random

Problemet med AM är att det ofta **tar tid och plats** att ta reda på var man ska placera den nya sidan

Eller se om en sida finns i cachen.

SET-ASSOCIATIVE CACHE

- Det finns fyra alternativa platser att lägga en cache line på
- Enkel beräking, snabb replacement policy

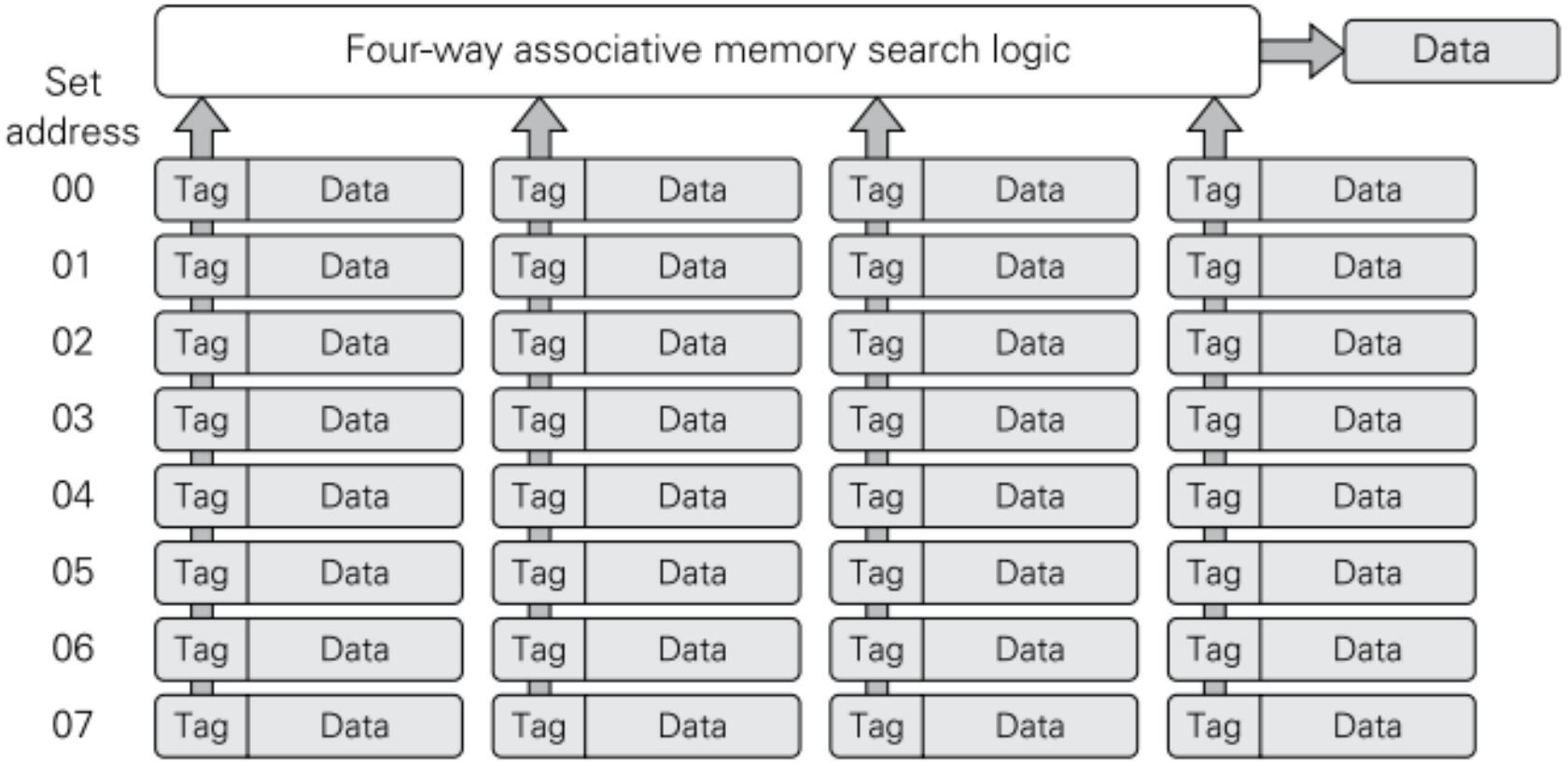


FIGURE 3-11: Set-associative cache mapping

- Raspberry Pi (Arm)
  - 4 way set-ass.
- Intel i7
  - L1i 8-way set-ass.
  - L1d 8-way set-ass.
  - L2 4-way set-ass.
  - L3 16-way set-ass.

UPPDATERA ARBETSMINNET

HÖGSKOLAN VÄST

Det finns två olika tekniker

- Write through
  - skrivning sker direkt till underliggande nivå.

  + Mindre chans för dataförlust

  + lättare att synkronisera flera cores

- Write back
  - skrivning av “dirty” pages sker när sidan slängs ut (evicted).

  + Färre skrivningar

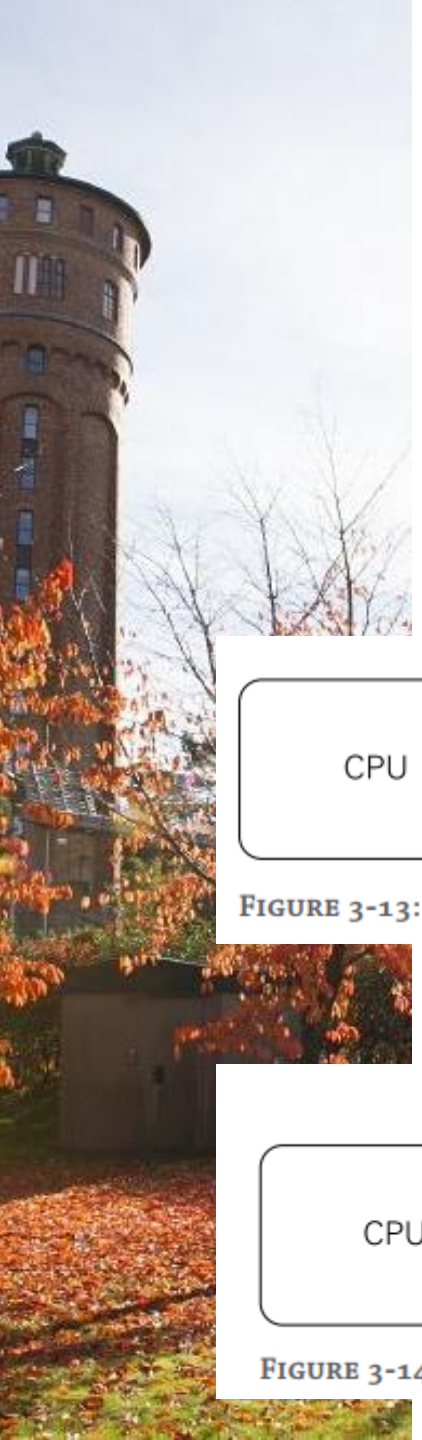
VIRTUELLT MINNE

HÖGSKOLAN VÄST

1. Minnesskydd
   - Varje process får sin egen minnesarea
   - Börjar på minnesadress 0.

2. Pageing (swapping)
   - Alla sidor behöver inte finnas i minnet
   - Vi kan alltså köra program som använder mer RAM än vad vi har

3. Defragmentering
   - Är inget problem



MEMORY MANAGEMENT UNIT

- MMU översätter virtuella till fysiska addresser.

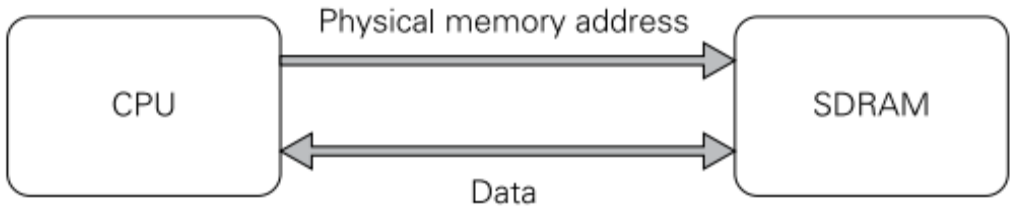


FIGURE 3-13: Direct use of physical memory addresses

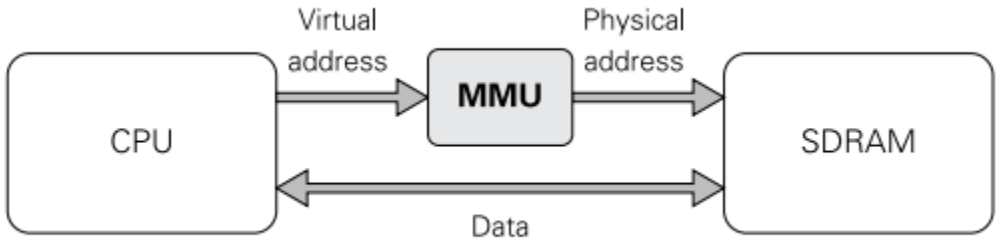


FIGURE 3-14: An MMU intermediating virtual and physical addresses

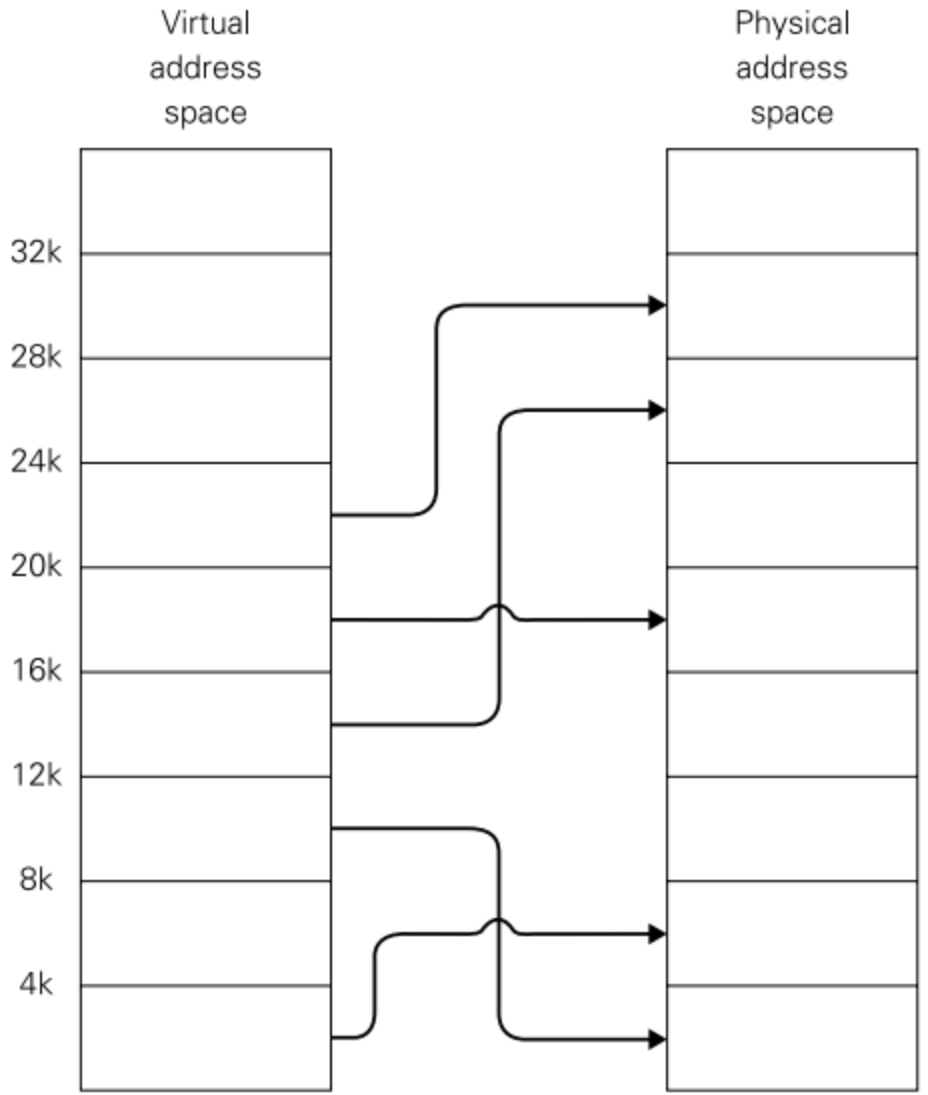


FIGURE 3-15: Stitching the virtual address space together out of 4KB blocks of physical memory

SIDTABELLEN – PAGE TABLE

HÖGSKOLAN VÄST

En sidtabell som talar om vilka sidor en process har allokerat.

En sidtabell per process.

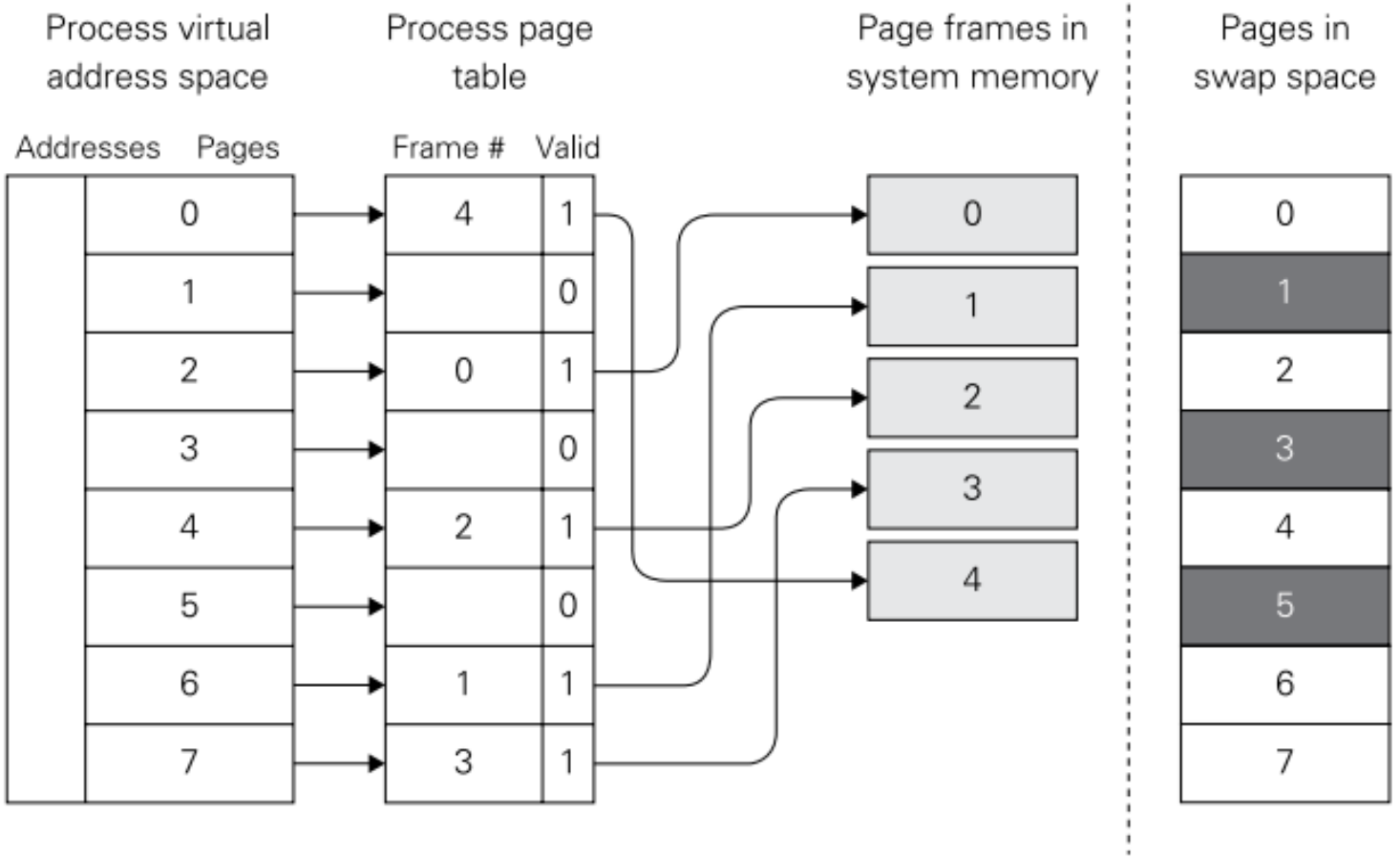


FIGURE 3-12: How virtual memory paging works

Translation lookaside buffer TLB

Lite överkurs

- Sidtabellen är stor och uppslagnigar i den tar tid.
- Uppslagningar cachas därför i en TLB.
- TLB är relativt små men pga. lokalitet och att de pekar på stora sidor fungerar det ändå bra.

Intel i7 (Skylake)

TLB	4k sidor	4k+2MB sidor	2/4 MB sidor	1G sidor
ITLB	128		8 per thread	
DTLB	64		32 st	4 st
STLB		1536 st		16 st