Informatica

Les 4

arrays – encapsulatie - geheugen

Jan Lemeire

Informatica 2e semester

februari – mei 2023

VRIJE UNIVERSITEIT BRUSSEL

Vandaag

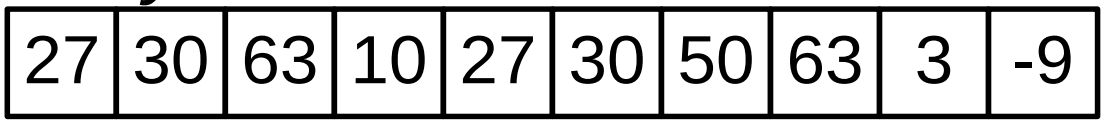
1. Arrays
2. Oefening
3. Algoritmes met arrays
4. Encapsulatie
5. Deel III: hoofdstuk 3 - Geheugen

Arrays

Boek II p. 5

Array

array



- **Array** ('rij') is een deel van het geheugen waarin een bepaald aantal elementen kunnen worden opgeslagen.
- Java: ***enkel elementen van eenzelfde type***
- **Grootte moet vastgelegd worden bij het aanmaken** (*reserveren*)
  - Aanmaken via **new**

```java
// ==== ARRAYS ====
int[] a; // declaratie van een array variabele
a = new int[10]; // aanmaken van array met grootte 10

a[0] = 10; // waarden toekennen
a[1] = 9;
a[2] = 8;
a[3] = 7;
a[4] = 6;
a[5] = 5;
a[6] = 4;
a[7] = 3;
a[8] = 2;
a[9] = 1;

// vullen van array
for (int i = 0;i < a.length;i++){
    a[i]=2*i; // index van 0 tot length-1
}
// printen van array
for (int i = 0;i < a.length;i++){
    System.out.print(a[i]+" ");
}
```

Oefening

```java
import java.util.Scanner;

public class ZeefVanErathostenes {
	/** PROGRAMMA */
	public static void main(String[] args) {
		System.out.print("Geef range: ");
		Scanner scanner = new Scanner(System.in);
		int N = scanner.nextInt();

		// maak array aan
		boolean[] array = new boolean[N];

		// initialiseer array
		for(int i=0;i<N;i++)
			array[i] = true;

		// de zeef
		for(int i=2;i<N;i++){
			if (array[i]){
				int m = i * 2;
				while(m < N){
					array[m] = false;
					m = m + i;
				}
			}
		}

		// print array
		System.out.print("XXX kleiner dan "+N+": ");
		for(int i=2;i<N;i++)
			if (array[i])
				System.out.print(i+", ");
		System.out.println();
	}
```

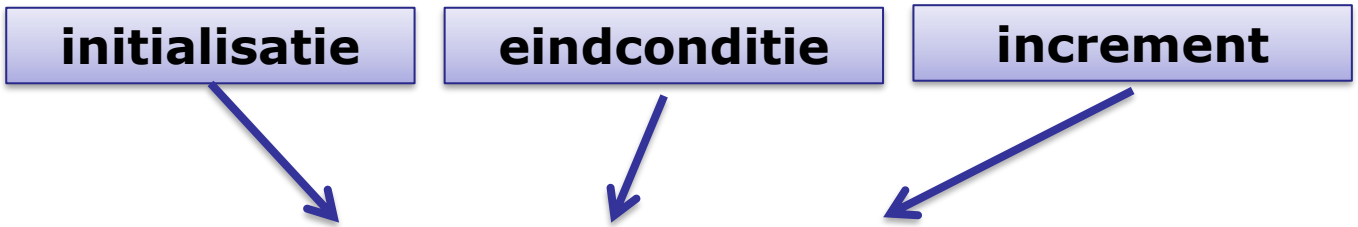
Accolade niet nodig, want maar 1 instructie

Output?

***Vraag 1**: wat als ik van 1 begin?*

***Vraag 2**: wat als ik van 0 begin?*

Java's for-lus




```
for(int i=0;i<100;i+=2){
        System.out.print(i+", ");
}
```

$i \mathrel{+}= 2 \quad \equiv \quad i = i + 2$

$i{+}{+} \quad \equiv \quad i \mathrel{+}= 1 \quad \equiv \quad i = i + 1$

als ik for-lus introduceer => equiv

while tonen

```
int i=0;
while(i<100){
        print
        i+=2;
}
```

```java
// ==== ARRAYS ====
int[] a; // declaratie van een array variabele
a = new int[100]; // aanmaken van array met grootte 100

// doorlopen van array
for (int i = 0;i < a.length;i++){
    a[i]=i; // index van 0 tot length-1
}

// aanmaken en vullen van array, krijgt een grootte van 3
String[] words = new String[]{"bla", "blo", "bli"};
System.out.println("'words' heeft " +words.length+" elementen");

// tweedimensionale array
int[][] matrix = new int[2][3];  // 2 rijen en 3 kolommen
matrix[1][1] = 5;
System.out.println("Matrix heeft " +matrix.length+" rijen en "
+matrix[0].length+" kolommen");
```

Array Algoritmes

Array algoritmes

- Zoek minimum in array
- Bereken totaal van array
- Bereken gemiddelde
- Tel het voorkomen van een element
- Check of de elementen gesorteerd zijn

```
/** PROGRAMMA */
public static void main(String[] args) {
 int lengte = 20, maxWaarde = 20;
 int[] array = randomArray(lengte, maxWaarde);

 System.out.println("Array       : "+Arrays.toString(array));
 System.out.println("Minimum     : "+minWaarde(array));
 System.out.println("Totaal      : "+totaal(array));
 System.out.println("Gemiddelde  : "+gemiddelde(array) +"
(ik verwacht hier "+((float)maxWaarde/2)+")");

 System.out.println("Aantal 3's  : "+aantal(array, 3)+"
(ik verwacht hier "+((float)lengte/maxWaarde)+")");

 System.out.println("Gesorteerd?   "+isGesorteerd(array));
 Arrays.sort(array);
 System.out.println("Gesorteerd?   "+isGesorteerd(array));
}
```

Arrays klasse

Algemene klasse met nuttige functies, zoals:

- `public static String toString(long[] a)`
- `public static void sort(int[] a)`
- `public static void sort(int[] a, int fromIndex, int toIndex)`
- Zie online documentatie

```
public static int[] randomArray(int length, int maxWaarde){
  int[] array = new int[length];
  Random r = new Random();
  for(int i=0;i<length; i++)
    array[i] = r.nextInt(maxWaarde);
  return array;
}
public static int minWaarde(int[] array){
  int min = array[0];
  for(int i=1;i<array.length; i++) // ik begin vanaf 1
    if (array[i] < min)
      min = array[i];
  return min;
}
public static int totaal(int[] array){
  int tot = 0;
  for(int i=0;i<array.length; i++)
    tot += array[i];
  return tot;
}
```

***Vraag**: wat als ik van 0 begin?*

Totaal berekenen van array

```
int tot = 0;
for(int i=0;i<array.length; i++)
    tot += array[i];
```

Alternatieve for-lus (a la Python):

```
int tot = 0;
for(int v: array)
    tot += v;
```

```
public static float gemiddelde(int[] array){
  int tot = totaal(array);
  return (float) tot / array.length;
}
public static int aantal(int[] array, int waarde){
  int n=0;
  for(int i=0;i<array.length; i++)
    if (array[i] == waarde)
      n++;
  return n;
}
public static boolean isGesorteerd(int[] array){
  for(int i=1;i<array.length; i++) // ik begin vanaf 1
    if (array[i] < array[i-1])
      return false;
  return true;
}
```

***Vraag**: wat als ik van 0 begin?*

documenteer de niet-triviale code.

Array Algoritmes vervolg

Zoeken in array I

```
public static int aantalIteraties = 0; // performantie

/** geeft index terug van eerste voorkomen van waarde.
Geeft -1 terug als niet gevonden */
public static int zoek(int[] array, int waarde){
  aantalIteraties = 0;
  for(int i=0;i<array.length; i++){
    aantalIteraties++;
    if (array[i] == waarde)
      return i;
  }
  return -1;
}
```

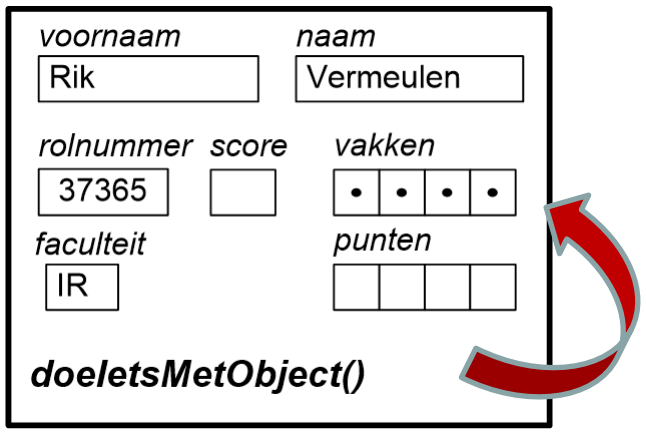
Lineaire zoektijd

```
System.out.println("Index van 3: "+zoek(array, 3)+"
(aantal zoekiteraties = "+aantalIteraties+", ik verwachtte
"+lengte/2+")");
```

static vs non-static

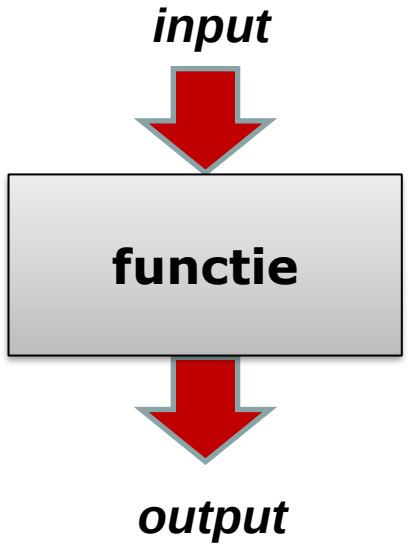
Niet in cursus
Hoort bij boek 1, p53 1.6.1

Gewone methode



Methode kan attributen bekijken of veranderen

statische methode



Methode gebruikt enkel input om output te berekenen.
Kan wel statische variabelen gebruiken.

Zoeken in array II


```
public static int zoekAlsGesorteerd(int[] array, int waarde){
  // eerst check ik of wel degelijk gesorteerd
  if (!isGesorteerd(array))
    return zoek(array, waarde);
  aantalIteraties = 0;
  int min = 0, max = array.length;
  int midden;
  do{
    aantalIteraties++;
    midden = (max + min) / 2;
    if (waarde == array[midden])
      return midden; // gevonden!
    if (waarde < array[midden])
      max = midden;
    else // waarde ligt voorbij midden
      min = midden;
  } while (min < max);
  return -1;
}
```

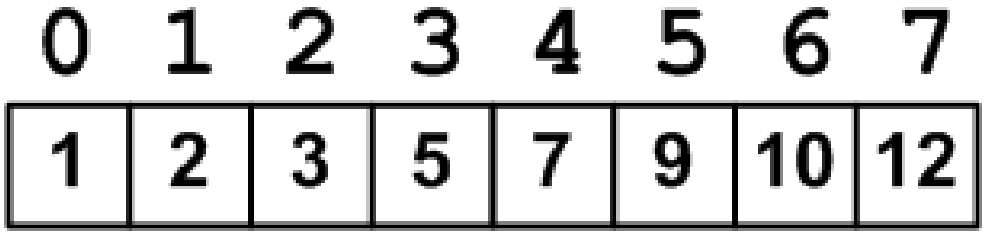
Logaritmische zoektijd

Hier is een fout!!!!
Verbeteren we zo meteen

Vraag: wat gebeurt er als het te zoeken getal het eerste element is?

Vorig programma loopt fout

Test het uit met het getal 6 te zoeken in volgende array:



Check dat het verbeterde programma van de volgende slide wel werkt.

Zoeken in array II

Niet in cursus

```
public static int zoekAlsGesorteerd(int[] array, int waarde){
  // eerst check ik of wel degelijk gesorteerd
  if (!isGesorteerd(array))
    return zoek(array, waarde);
  aantalIteraties = 0;
  int min = -1, max = array.length; // we zorgen ervoor dat
min en max niet de oplossing zijn
  int midden;
  do{
    aantalIteraties++;
    midden = (max + min) / 2;
    if (waarde == array[midden])
      return midden; // gevonden!
    if (waarde < array[midden])
      max = midden;
    else // waarde ligt voorbij midden
      min = midden;
  } while (max - min > 1); // zolang er elementen tussen min
                           // en max zijn
}
```

Logaritmische zoektijd

Afleiding logaritmische wet

- We 'kappen' de array steeds in 2
- In het slechtste geval moeten we zoeken tot we nog maar 1 element hebben (min==max)
- Dus:
  - n => n/2 => n/4 => ... => 1
  - Dus: 1*2*2*2*...*2 ≈ n ≈ 2^x (niet exact door afrondingen)
  - => aantal iteraties = x = $\log_2 n$
  - Voorbeeld n=1024 => 512 => 256 => 128 => 64 => 32 => 16 => 8 => 4 => 2 => 1;

    10 iteraties want $2^{10} = 1024$

Deel II: hoofdstuk 1

- overzicht

Datastructuur	Random access (opvragen i^{de} element)	Find (via naam)	Toevoegen / verwijderen nadat element gevonden is
Array	O(0) ++	O(n) O(log(n)) als gesorteerd	O(n) -
ArrayList	O(0) ++	O(n) O(log(n)) als gesorteerd	O(n) -
Linked list	O(n) -	O(n) --	O(0) ++
Binaire boom	n.v.t.	O(log(n)) +	O(log(n)) ++
Hashtabel	n.v.t.	O(0) ++	O(0) ++ Zolang binnen grootte

De ArrayList

- Houdt het aantal elementen bij die je er in gestoken hebt
  - Op te vragen met *size()*
- Maar heeft intern een grotere array
- Indien nodig, vergroot het automatisch zijn grootte
  - door een nieuwe array aan te maken

≈ de Python-lijst

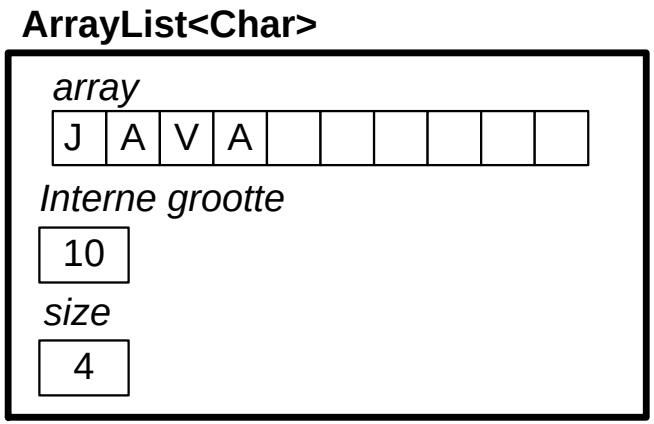
Array vs ArrayList

Array	ArrayList	Python list	Python tuple
Vaste grootte	*Flexibele grootte*	*Flexibele grootte*	*Vaste grootte*
Grootte te specifieren	*Initiele grootte facultatief*	*Niet nodig*	*Bij initialisatie*
`int[] array = new int[5];`	`ArrayList<Integer> arrayList = new ArrayList<Integer>();`	`list=[]`	*Creatie en initialisatie samen*
`array = new int[]{1, 2,3};`	*Initialisatie niet mogelijk*	`list=[1, 2, 3]`	`tuple = (1, 2, 3, 4, 5 )`
`x = array[2];`	`x = arrayList.get(2);`	`x=list[2]`	`x=tuple[2]`
`array[1] = 5;`	`arrayList.set(1, 5);`	`list[1]=5`	*Veranderen niet mogelijk*
Toevoegen niet mogelijk	`arrayList.add(7);`	`list.append(7)`	*Niet mogelijk*
`array.length`	`arrayList.size()`	`len(list)`	`len(tuple)`

Encapsulatie

Interne werking ArrayList?

- Weten we niet, zien we niet…
- Ook de attributen zijn verborgen
  - **`private`** in plaats van **`public`**
  - Moest de gebruiker ze kunnen aanpassen, zou dit kunnen leiden tot inconsistenties
  - Zo zou het object er uit kunnen zien van binnen:



Pijlers van object-georiënteerde programmeertalen

I. Encapsulatie

- **2.3 ArrayList p. 11**
- **3.1 Stapel-datastructuur p. 12**
- **6.2 Java's LinkedList p. 55**

II. Overerving (inheritance)

- 1.1.2 Studentvoorbeeld p. 19
- 1.3 Vriendenvoorbeeld p. 36
- 1.4.1 MyPanel p. 41
- 1.4.3 PainComponent overschrijven p. 44
- **4.3 FunctieMetAfgeleide-interface p. 25**

III. Polymorfisme en abstractie

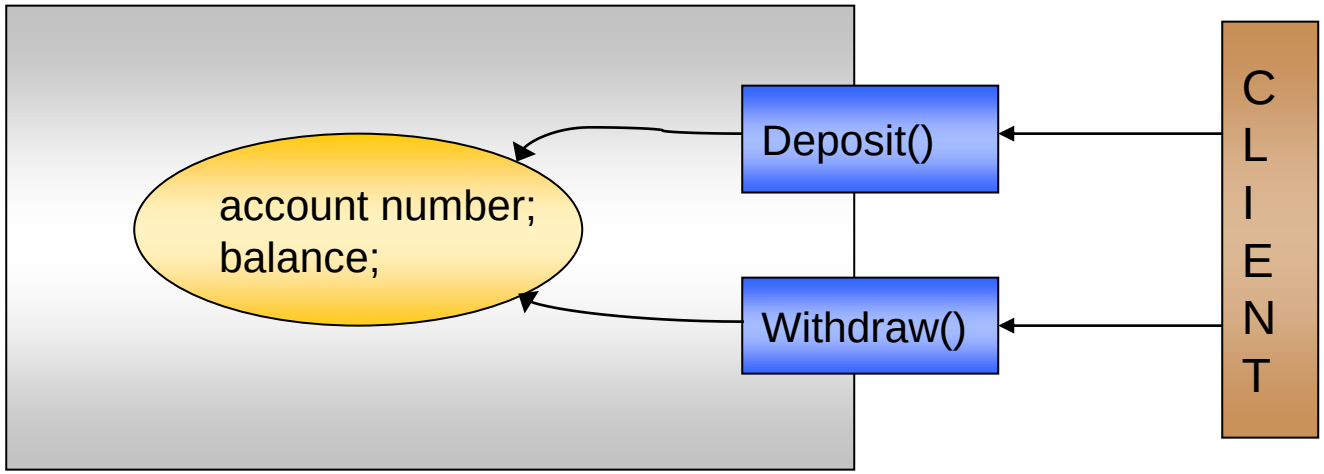
- 1.2.4 Set p. 32
- 1.2.5 Map p. 33
- 1.4.2 EventListener p. 43
- 1.6.4 Abstracte klassen p. 58
- **4.2 Functie-interface p. 21**
- **5.2.2 Backtracking & Breadth-first p. 35**
- **Addendum bij hoofdstuk 5 (zie website, is optioneel)**
  - **Abstract zoekalgoritme**
  - **Vergelijking van zoekalgoritmes**

Encapsulatie (Data hiding)

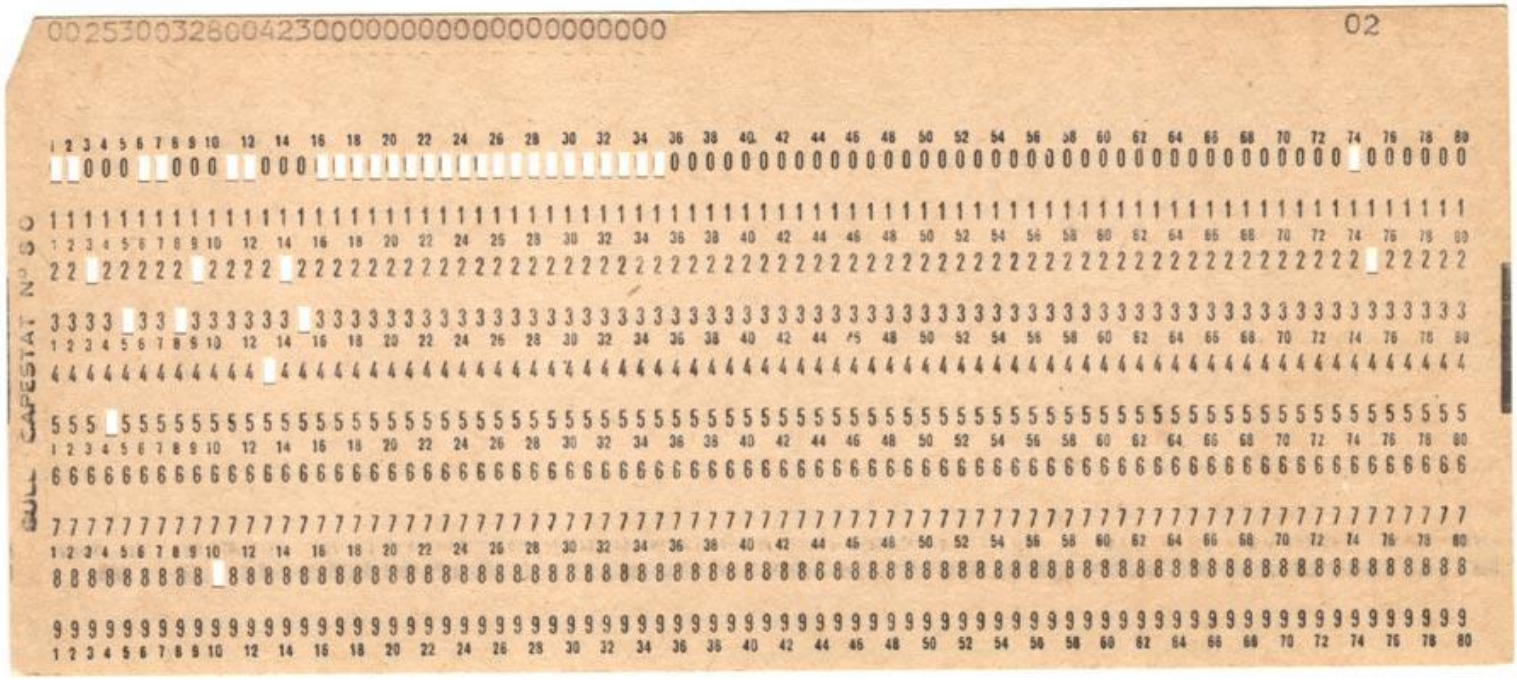
- Data en operaties zitten samen (in het object)
- Data maak je niet toegankelijk voor de buitenwereld, enkel operaties
  - Daarom *get()* en *set()*-methodes voor attributen
- De maker van de klasse heeft controle over wat de gebruiker met de data doet
- Als gebruiker weet je dat de maker ervoor gezorgd heeft de consistentie bewaard blijft
  - "je hoeft niet bang te zijn iets verkeerds te doen"

Encapsulatie: voorbeelden

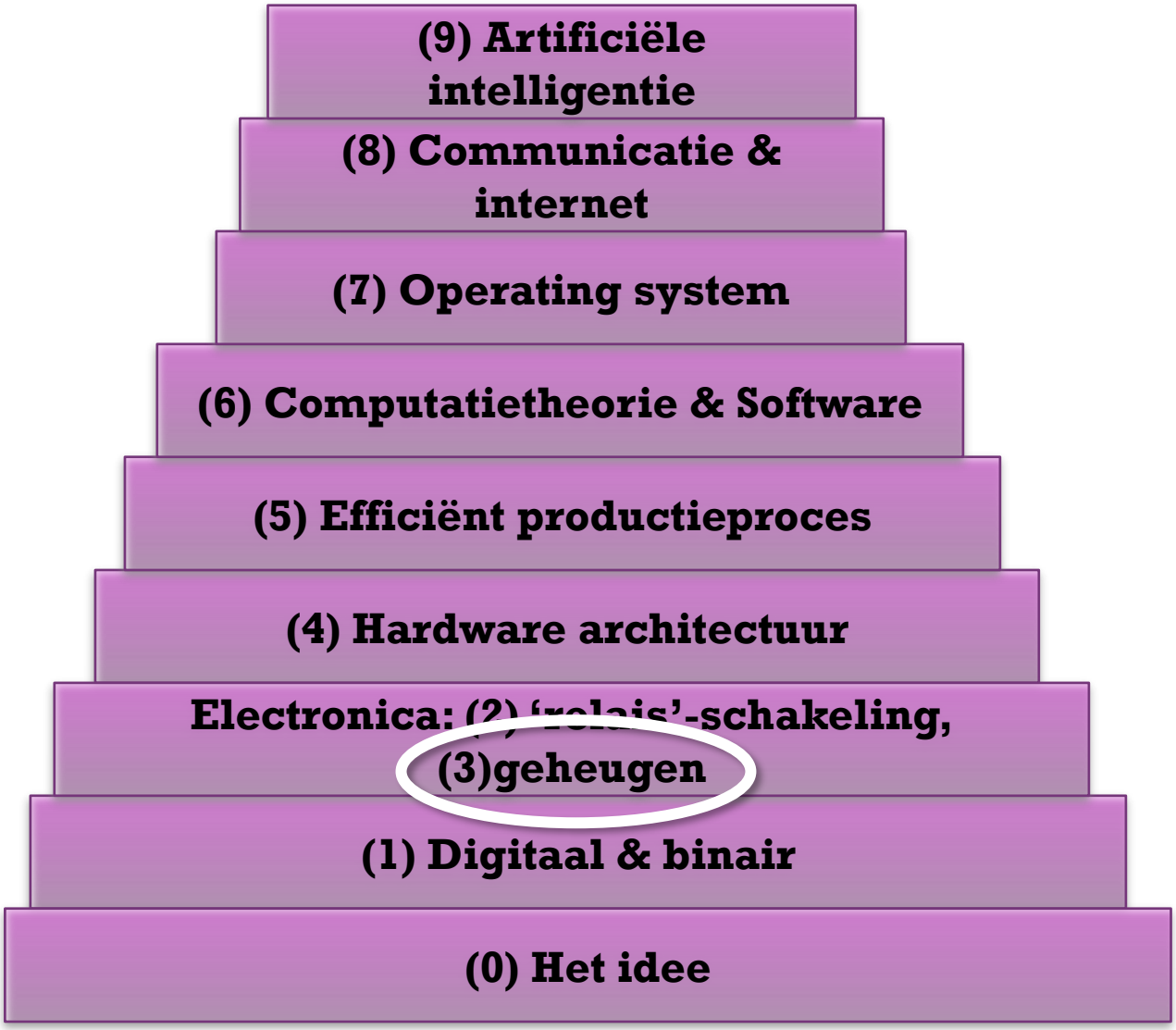
- *Voorbeeld 1*

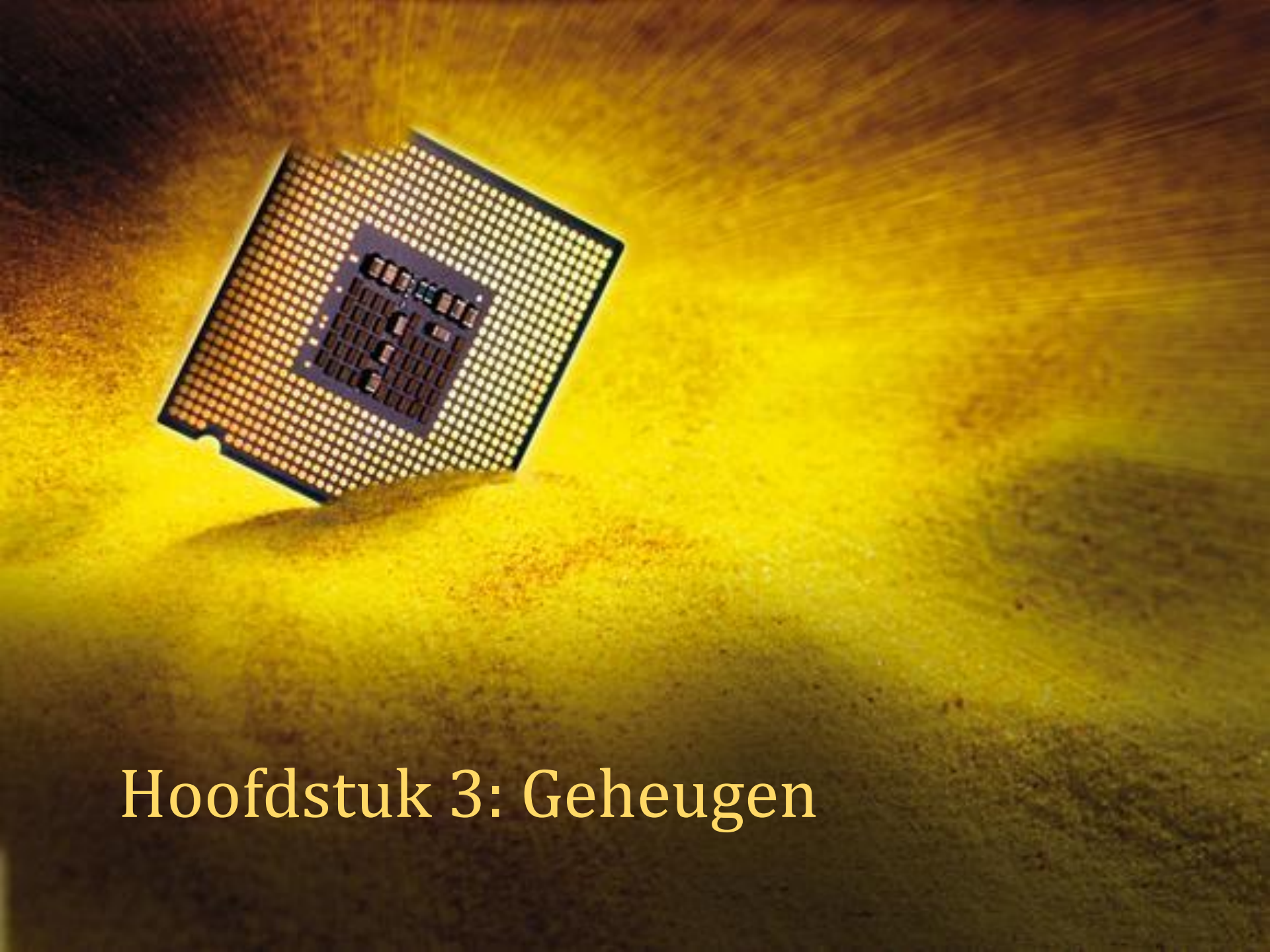


- *Voorbeeld 2*: toegang tot attributen via get() en set() methodes
  - Het object behoudt de controle over de gegevens


00253003280042300000000000000000000
02
BULL CAPESTAT Nº 80

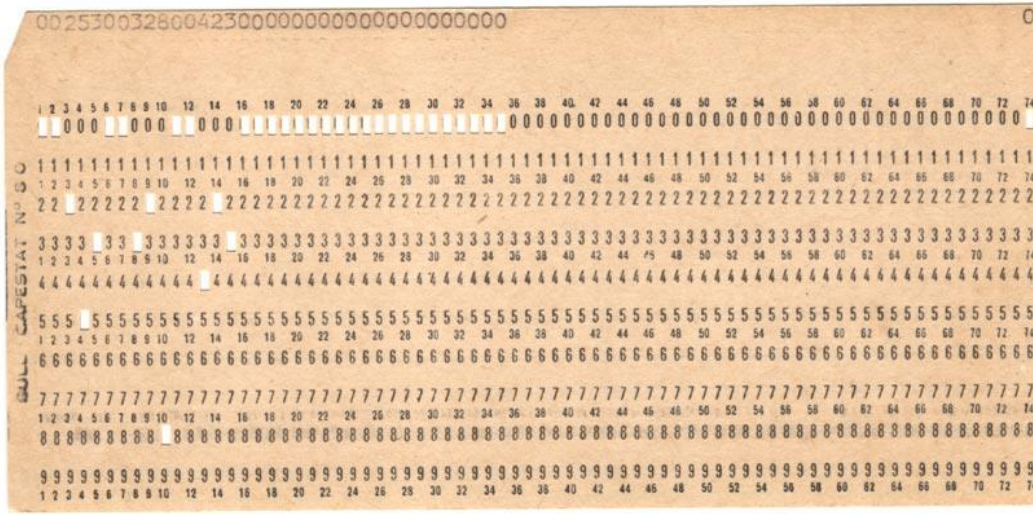
Waarmaken van Leibniz's droom





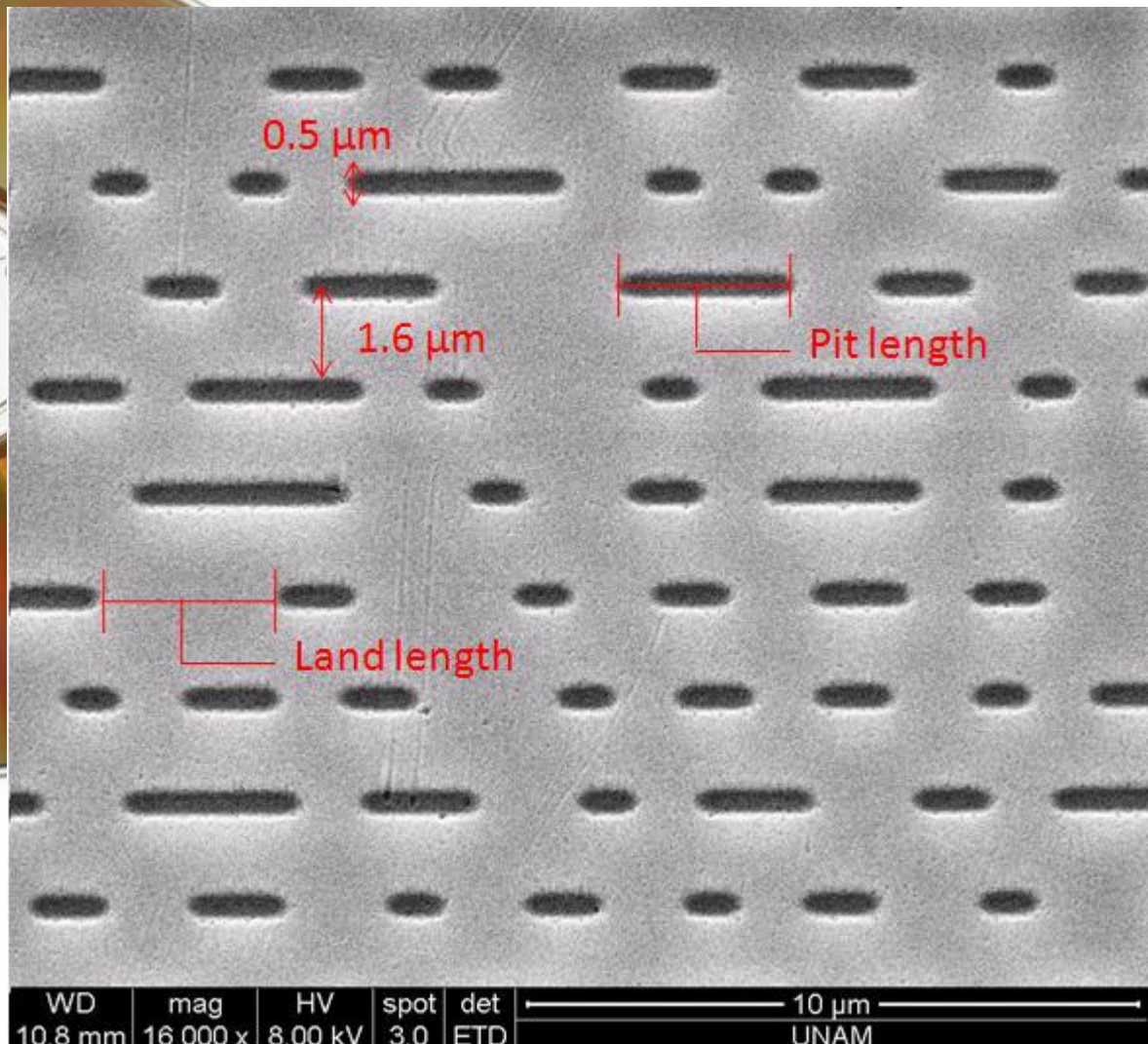
Hoofdstuk 3: Geheugen

Ponskaarten



- Beschrijven de programma’s

Moderne ponskaart?

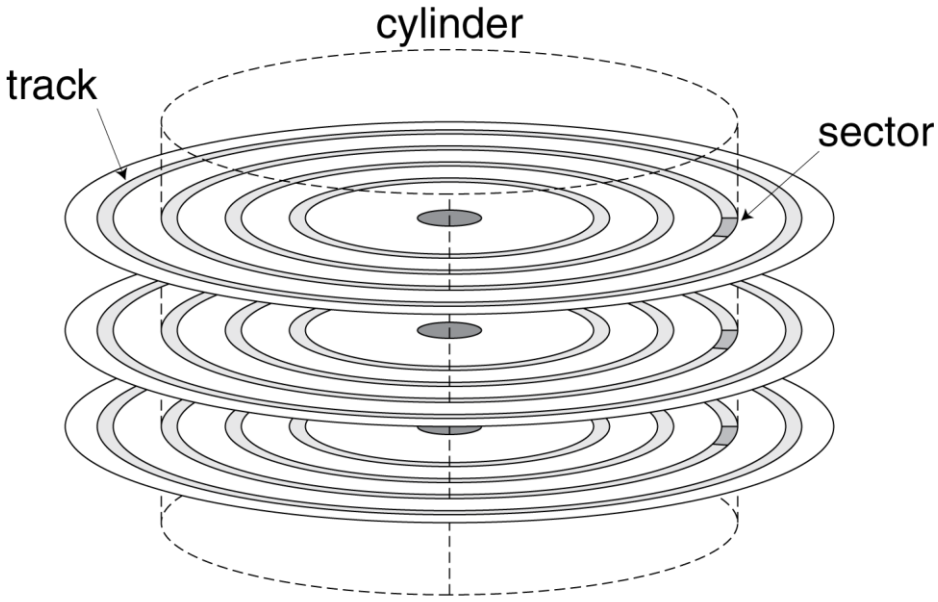


Magnetische banden

- zoals muziek- en videocassettes

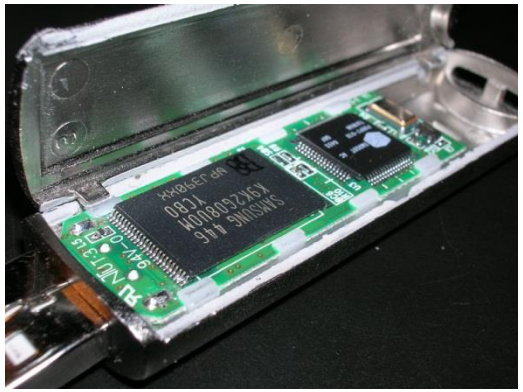
Harde schijf

- Non-volatile, rotating magnetic storage



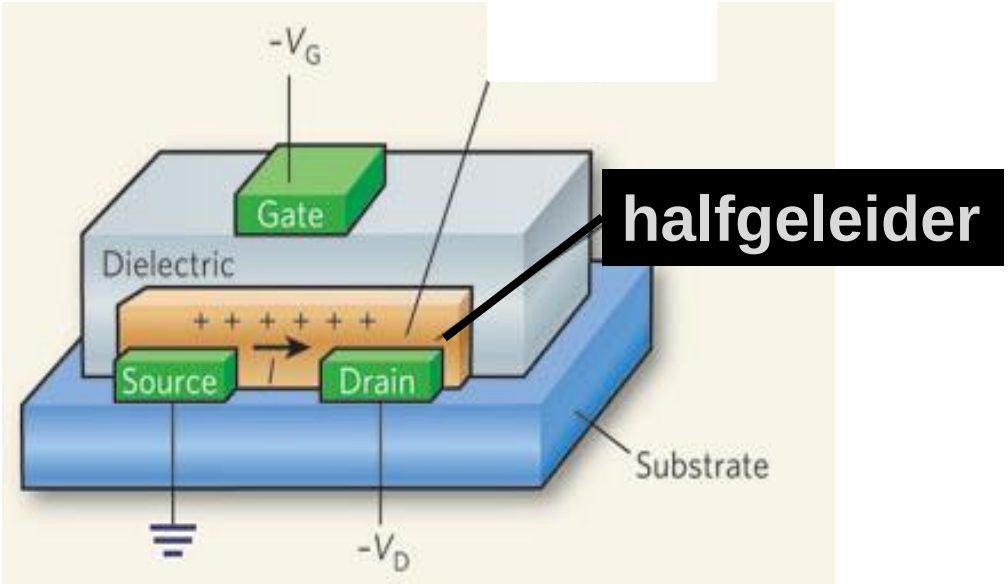
Flashgeheugen (SSD)

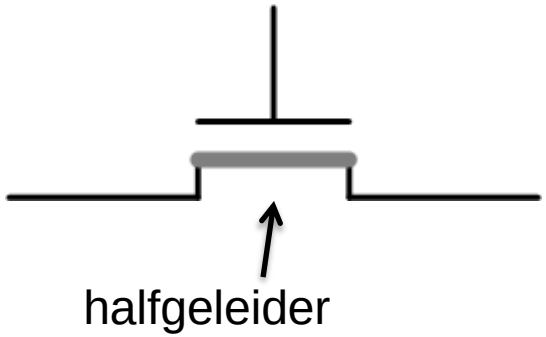
- Non-volatile <u>semiconductor</u> storage
  - 100× – 1000× faster than disk
  - Smaller, lower power, more robust
  - But more $/GB (between disk and DRAM)



Herhaling: Transistor

- De spanning $-V_G$ bepaalt de geleidbaarheid van de halfgeleider (semiconductor) tussen Source en Drain
  - Te gebruiken als versterker
  - Te gebruiken als relais, voor binaire operaties

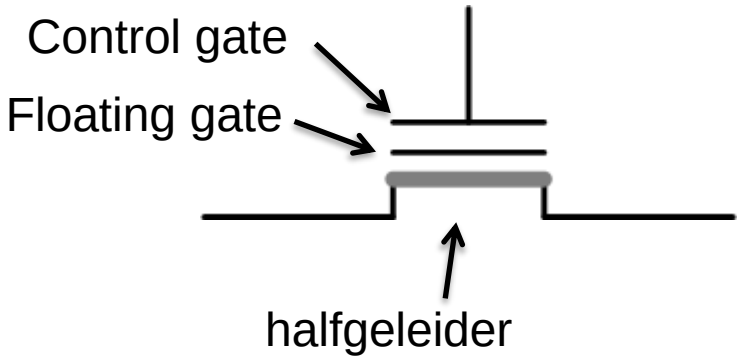




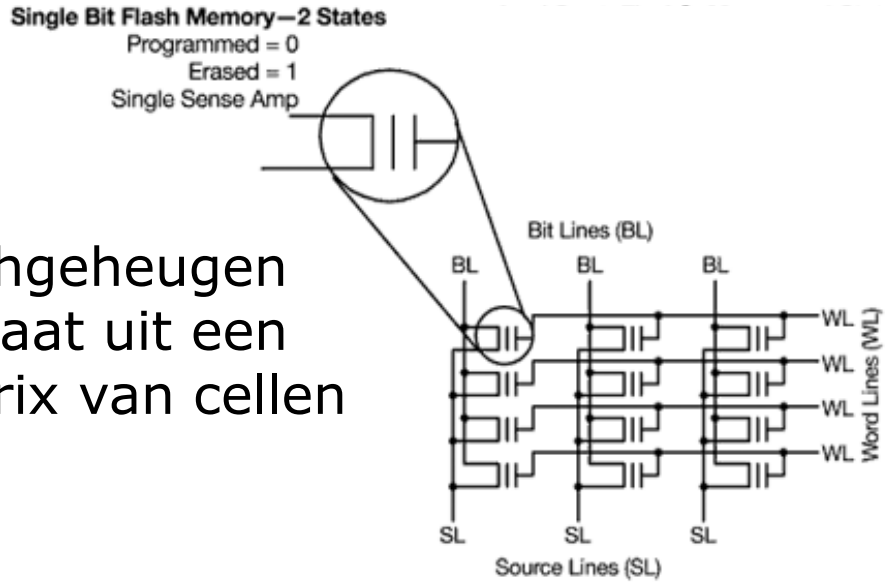
transistorwerking

Hoe werkt Flashgeheugen?

- Elektrische lading wordt bewaard in een ingekapselde geleider *(*de *floating gate).*
  - 50.000 elektronen zijn voldoende, maar door lekken gaan die toch verdwijnen. Zeker meer dan 10 jaar OK, maar hoe lang flashgeheugen veilig is, is nergens te vinden.
  - De floating gate bepaalt geleidbaarheid van halfgeleider

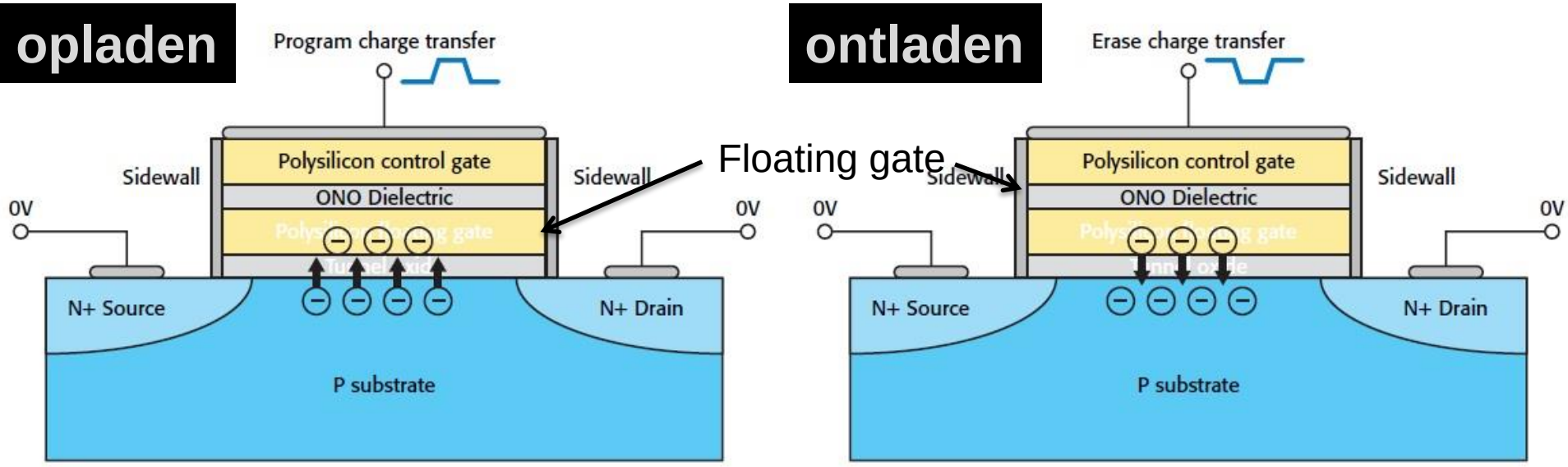


Flashgeheugen bestaat uit een matrix van cellen



Op- of ontladen flash bit

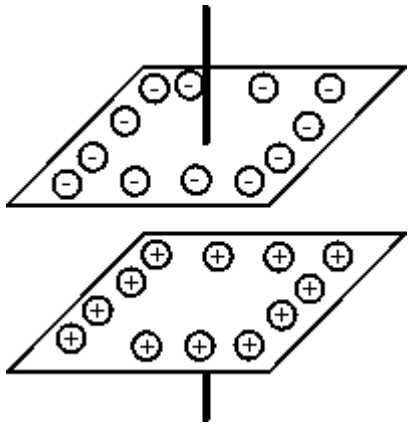
- Op- of ontladen van *floating gate* gebeurt door isolator via ***tunneling*** (kwantum-mechanisch effect)
- Geactiveerd via hoge spanning op *control gate*



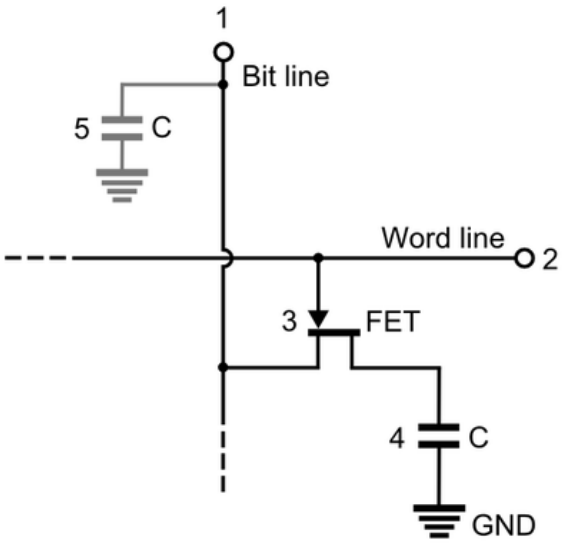
Permanent versus volatiel geheugen

- **Permanent**: “eeuwigdurend”
  - Optisch: Ponskaarten en CDs
  - Magnetisch: banden, diskettes en harde schijven
  - Flash-geheugen
  - Optisch & magnetisch enkel *sequentieel* leesbaar: je moet eerst ‘doorspoelen’ naar de juiste positie
- **Vluchtig of volatiel**: electriciteit nodig om gegevens aktief te behouden
  - Wordt gebruikt als *werkgeheugen*
  - Random Access Memory (RAM): elke byte kan onmiddellijk gelezen worden, is direct toegankelijk

Lading bewaren: condensator

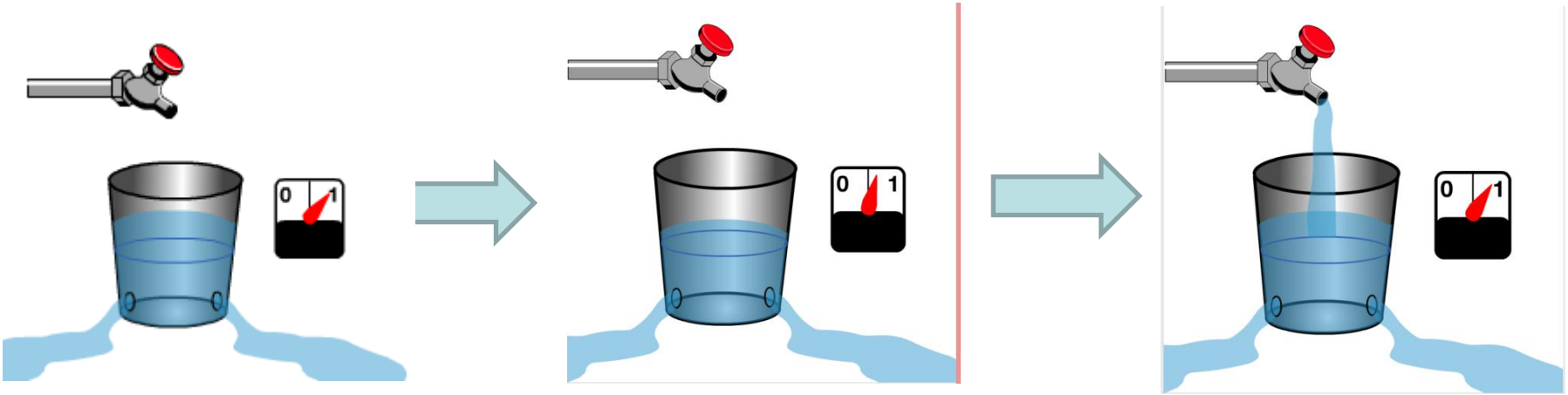


- Het geheugen dat de snelste transfer van bits van en naar de processor toelaat is dat type van geheugen waarbij elektrische schakelingen gebruikt worden waarbij 1 of 0 wordt voorgesteld als de aan- of afwezigheid van spanning over een geleider.
- Maar lading zal langzaam wegvloeien en ‹en…



DRAM-cel die 1 bit kan opslaan met behulp van een condensator (C) en een transistor (MOSFET of FET)

Volatiel geheugen: refreshen!

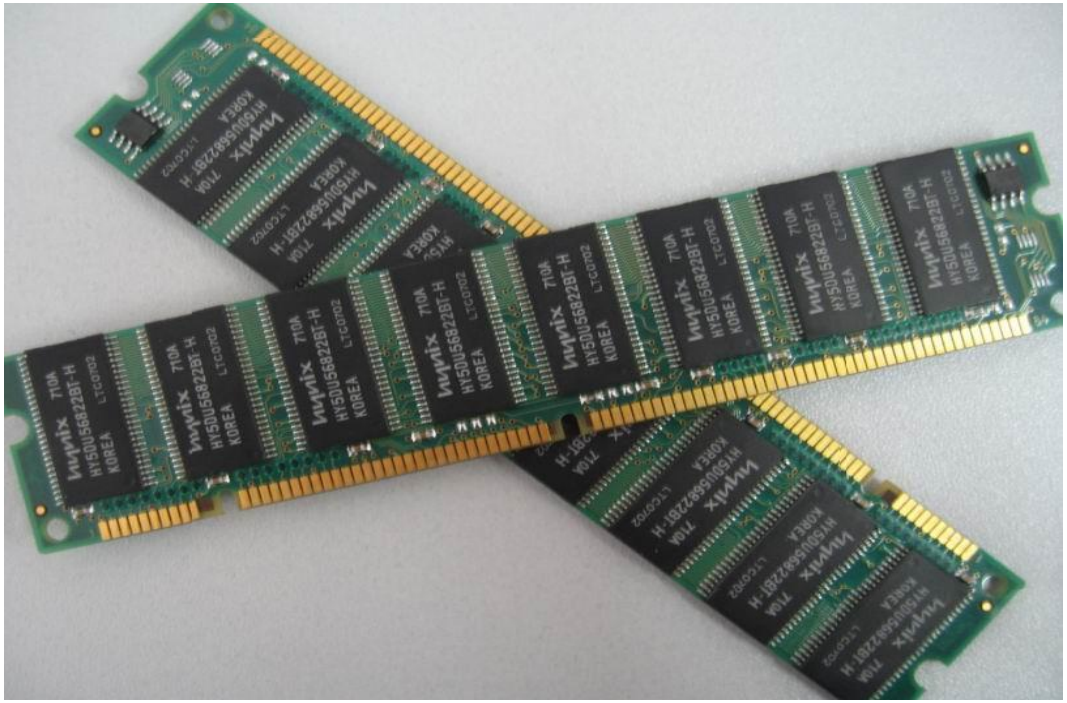


- Waarde van geheugen hou je bij op een condensator, maar die lading vloeit weg
- ➡ regelmatig weer opladen is noodzakelijk!
- **Dynamic Random Access Memory** (DRAM): elke cel slaat 1 bit op en bestaat uit een *capaciteit* voor de bitwaarde te bewaren, en een *transistor* voor refreshen/lezen/schrijven
- **Static Random Access Memory** (SRAM): een alternatief bestaande uit enkel transistors (*flipflops*). SRAM is sneller, maar neemt meer plaats in en is duurder.

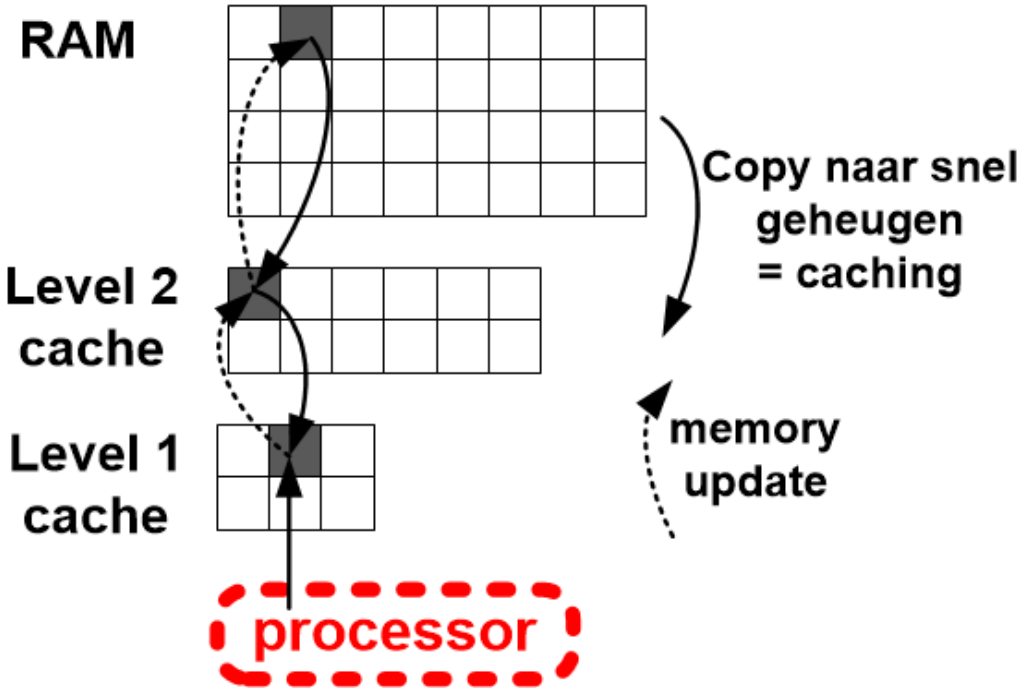
Werk- versus perifeer geheugen

- **Werk- of primair geheugen**: rechtsstreeks bruikbaar in je programma's
  - Elk programma krijgt een deel van het geheugen toegewezen om in te 'werken'
  - Heeft electriciteit nodig, bestaat enkel als computer opstaat, is dus vluchtig of volatiel, dus niet voor het opslaan van permanente gegevens
- **Perifeer of secundair geheugen**: om gegevens permanent op te slaan
  - Gebeurt via files
  - Vanuit programma lees en schrijf je *files*
  - Goedkoop, maar wel traag

Werkgeheugen: RAM

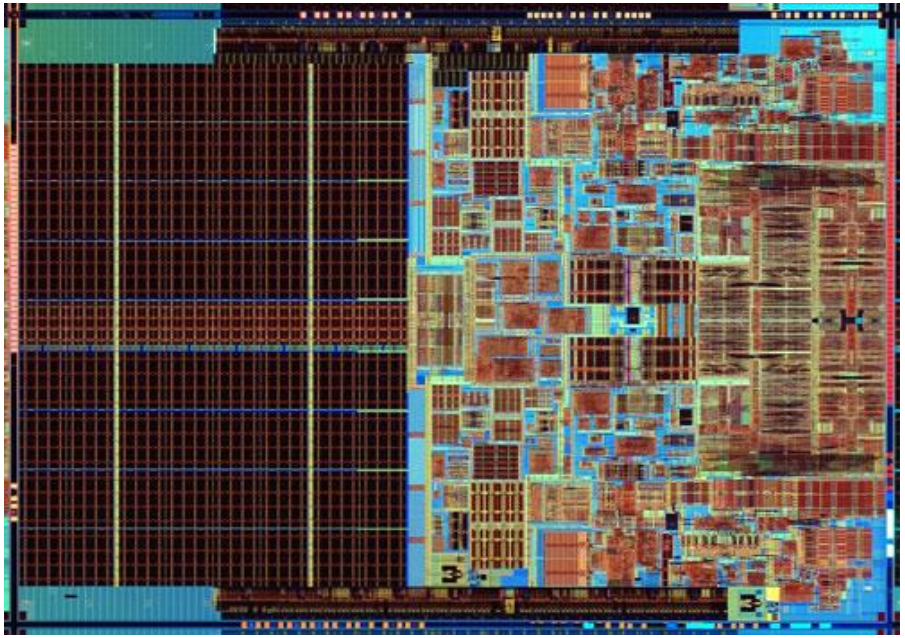


Caching (Level 1, Level 2)



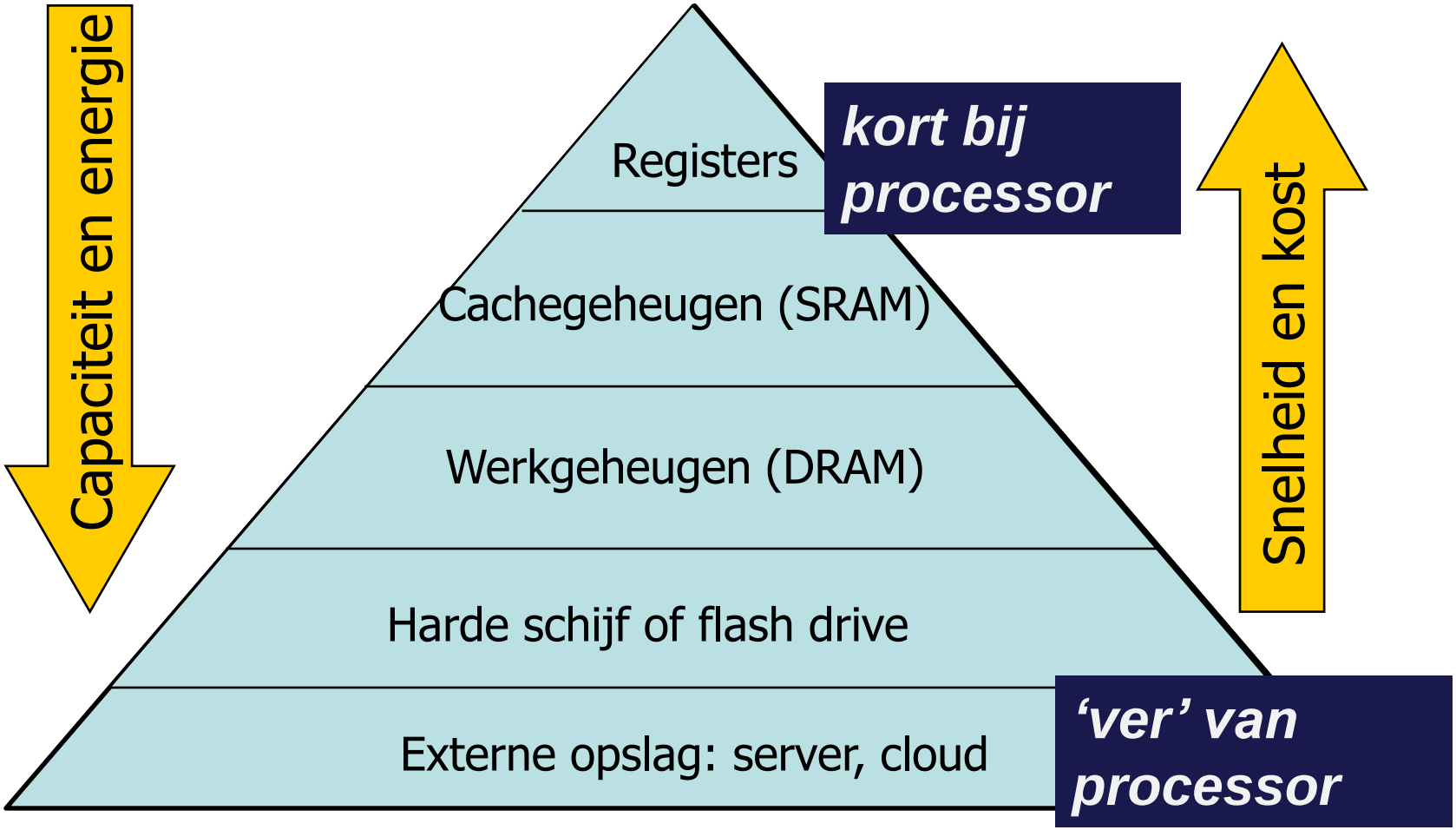
Caching bestaat uit het kopiëren van gegevens van traag geheugen naar snel geheugen. Na het kopiëren kunnen de gegevens snel gelezen en aangepast worden in het snel geheugen. Na aanpassing worden de gegevens eveneens aangepast in het origineel geheugen.
RAM-geheugen moet groot zijn (Gigabytes), maar kan daarom niet tegelijkertijd ook goedkoop en snel zijn. Dit lost men op door caching.

Processorchip



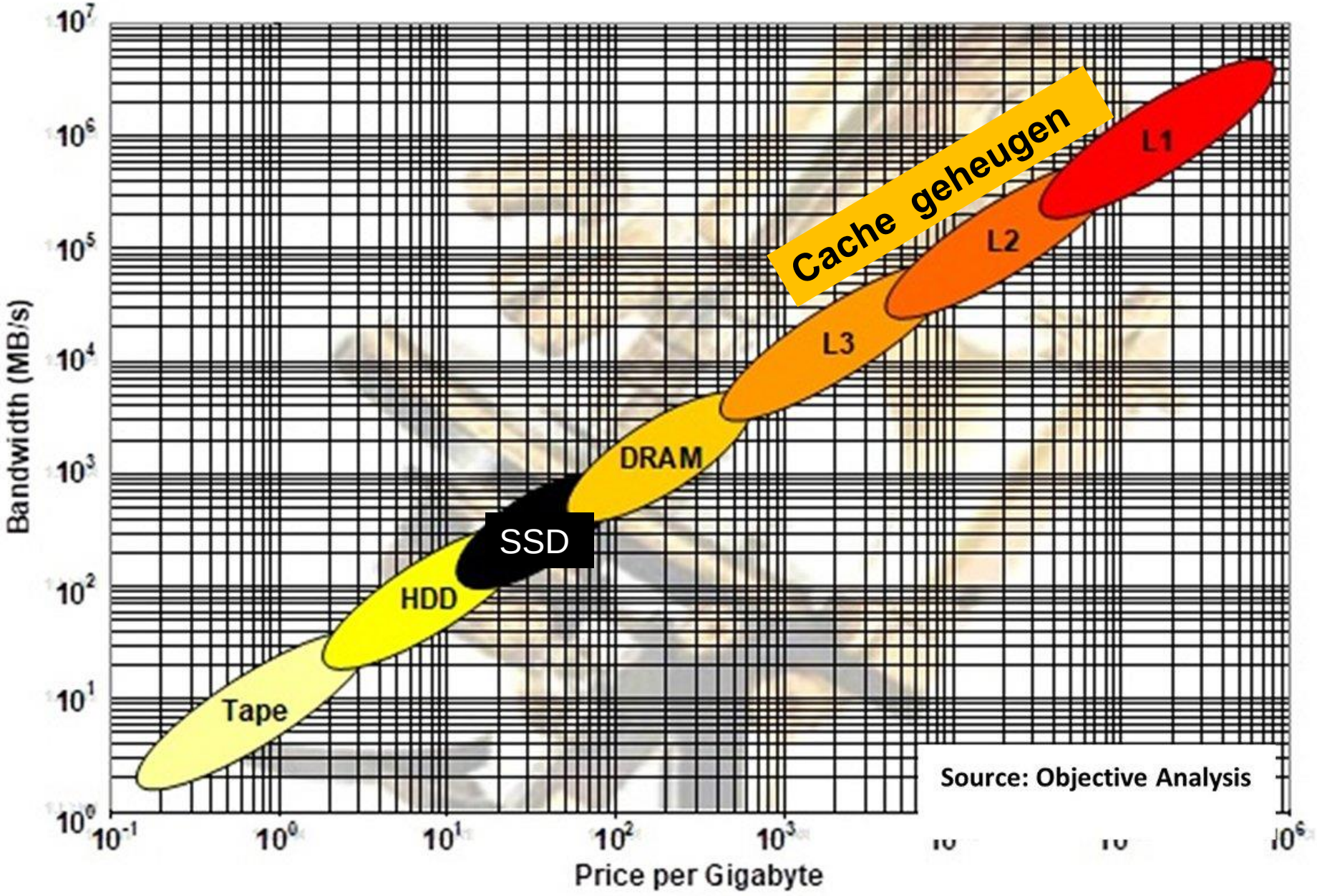
- Dual core CPU chip met aanduiding in blauw van *registers*.
  - Registers: zeer snel, klein geheugen voor het bijhouden van tussentijdse waarden van berekeningen.
- Het *cache geheugen* bevindt zich in het donkere linkergedeelte.

Geheugenhiërarchie



Geheugenhiërarchie

Ondanks het feit dat de von Neumann-architectuur het computergeheugen als één enkele component afbeeldt, gebruiken moderne computers een combinatie van geheugentypes, elk met z'n eigen karakteristieke performantie en kostprijs.
De trade-off tussen performantie en kost is afgebeeld op de volgende slide.
Aangezien de prijs van geheugens sterk afhankelijk is van de toegangstijd (dat is de tijd nodig om gememoriseerde informatie terug te vinden) wordt het geheugen meestal fysisch opgesplitst in delen met verschillende eigenschappen: duur, zeer snel geheugen voor wat snel bereikbaar moet zijn, en traag, goedkoper geheugen voor wat niet onmiddellijk nodig is. De snelle technologieën worden gebruikt voor registers en het primair werkgeheugen, terwijl de goedkope technologieën aangewend worden in de secundaire periferische geheugens.
Recentelijk is er de mogelijkheid tot ‘remote storage’, het extern opslaan van je gegevens in de cloud of op een server.


Cache geheugen
L1
L2
L3
DRAM
SSD
HDD
Tape
Source: Objective Analysis
Bandwidth (MB/s)
Price per Gigabyte
10^{7}
10^{6}
10^{5}
10^{4}
10^{3}
10^{2}
10^{1}
10^{0}
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}