Informatica

2e semester: les 5

lineaire datastructuren - chips

Jan Lemeire

Informatica 2e semester

februari – mei 2023

VRIJE UNIVERSITEIT BRUSSEL

Vandaag

1. Stapel (hfst 3)
2. Wachtrij (hfst 3)
3. Klasse-oefening
4. Interface-oefening
5. Gelinkte lijst (hfst 6)
6. Deel III/5: chips

Stapel

Stack

De stapel of 'stack'

- Twee operaties: dingen opleggen en afnemen

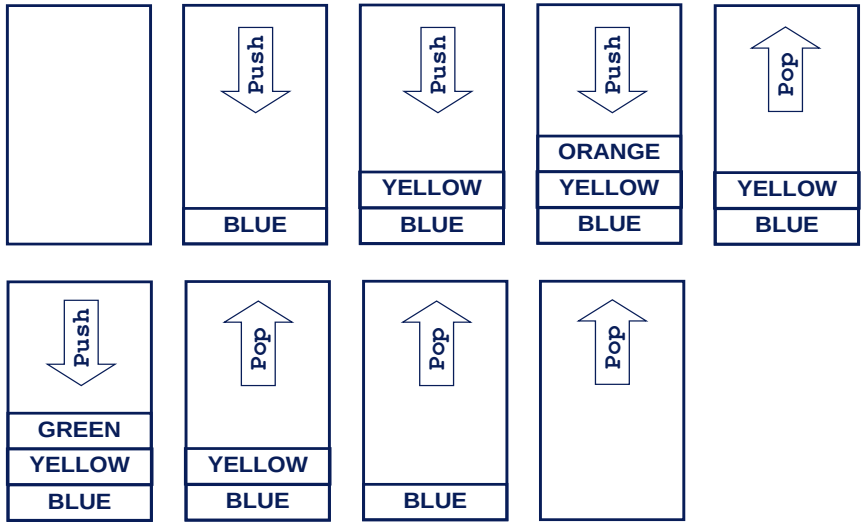


Fig.1.5. Last-in, first-out stack

- *last-in, first-out* (lifo) structuur

‘interface’ van stack

- Wat willen we er mee doen?
  - Pop
  - Push
  - isEmpty
  - isFull
- Headers van de methods (interface):
  - `void push(int element)`
  - `int pop()`
  - `boolean isEmpty()`
  - `boolean isFull()`

```
import java.nio.BufferOverflowException;
import java.nio.BufferUnderflowException;

public class Stack {
    private int[] data;
    private int pointer=0; // wijst naar het eerste vrije element

    public Stack(int capacity){
          data = new int[capacity];
    }
    public void push(int element){
          if (pointer == data.length) // check of vol
            throw new BufferOverflowException();
         data[pointer] = element;
         pointer++;
    }
    public int pop(){
        if (pointer == 0) // check of leeg
            throw new BufferUnderflowException();
        pointer--;
        return data[pointer];
    }
    public boolean isEmpty(){
         return pointer == 0;
    }
    public boolean isFull(){
         return pointer == data.length;
    }
```

Vraagjes

- Je zou ook een pointer kunnen bijhouden naar het bovenste element van de stack, waarbij je met -1 aanduidt dat hij leeg is.
  - Wat verandert er aan de code?
- Voeg een *size()* methode toe die aangeeft hoeveel elementen er op de stack zijn.
- Nu werkt ie enkel met integers. Hoe generiek maken, t.t.z. bruikbaar voor elk type?
  - zie volgende datastructuur

Waarom niet gewoon een array gebruiken?

1. Code van stack komt overal in je programma terecht
2. Moeilijk om fouten met stack op te sporen
3. Je geeft niet aan wat je met de array wilt doen
4. Een collega-programmeur kan de array of pointer aanpassen
   - ➡ Consistentie om zeep helpen

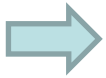
   Voorbeeld van het **Encapsulatieprincipe**

Je programma wordt dan:

```
...
int[] data = new int[capacity];
int pointer=0;
...

if (pointer == data.length)
    throw new BufferOverflowException();
data[pointer] = element;
pointer++;

...
if (pointer == 0)
    throw new BufferUnderflowException();
pointer--;
element = data[pointer];
...
```



```
Stack stack = new Stack(3);
```

```
stack.push(element);
```

```
element = stack.pop();
```

Encapsulatie

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"

Wachtrij
Queue

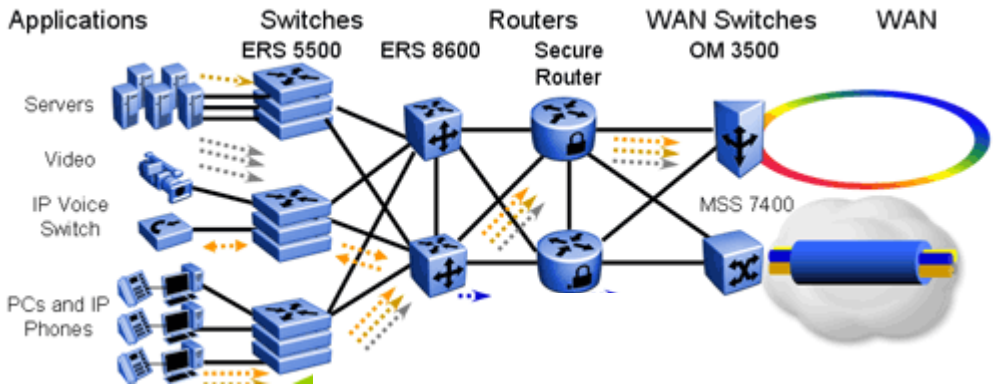
Wachtrij of Fifo-queue

"What would the lady at the back like?"

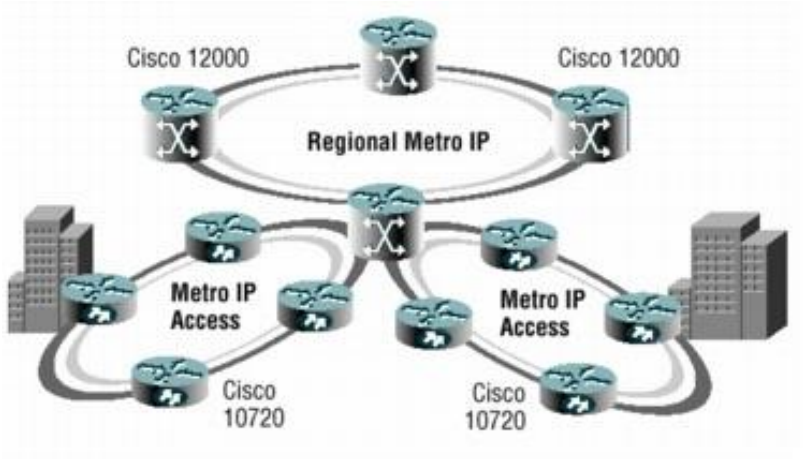
- *Fifo = First-in, first out*

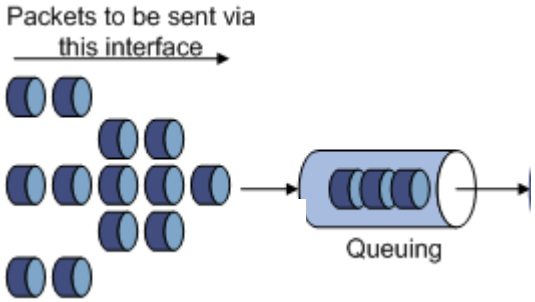
Internetknopen

Zie deel III hoofdstuk 8



- ***Switch:*** knooppunt
- ***Router:*** bepaalt ook route van pakketje





Als queue vol: **packetdrop**

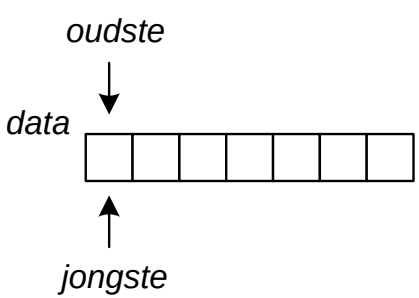
- Verlies van gegevens
- Moeten opnieuw verstuurd worden

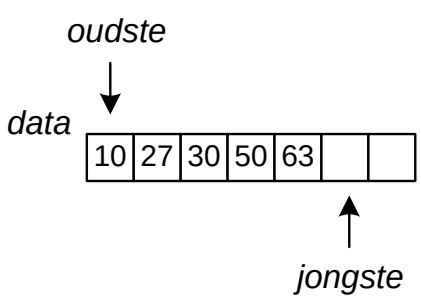
Fifo-queue

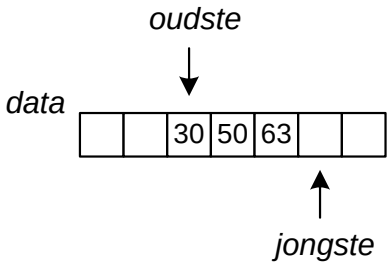
- Basisoperaties:
  - add()
  - get()

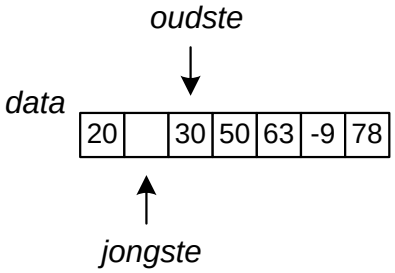
Implementatie met array

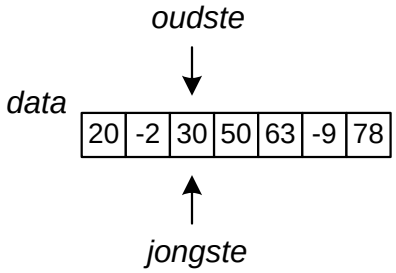
- 2 pointers nodig











- en ... boolean voor te weten of vol/leeg

```
public class FIFOQueue<T>{
	private T[] data;
	private int oudste=0; // pointer naar oudste element (als niet leeg)
	private int jongste=0;//pointer naar de plaats voor het volgend element
	boolean full = false; // als queue vol is
	public FIFOQueue(int capacity){
		data = (T[]) new Object[capacity];
	}
	public void add(T waarde){
		if (full)
			throw new BufferOverflowException();
		data[jongste] = waarde;
		jongste++;
		if (jongste == data.length)
			jongste = 0;
		if (jongste == oudste)
			full = true;
	}
	public T get(){
		if (isEmpty())
			throw new BufferUnderflowException();
		T waarde = data[oudste];
		data[oudste] = null;
		oudste++;
		if (oudste == data.length)
			oudste = 0;
		if (full)
			full = false;
```

```
public boolean isEmpty(){
   return !full && jongste == oudste;
}
public boolean isFull(){
   return full;
}
```

Priorityqueue

- Een wachtrij waarbij niet het oudste element wordt teruggegeven, maar het element met de hoogste prioriteit
- Java heeft hiervoor een klasse PriorityQueue<E> in zijn bibliotheek.
  - De poll()-methode neemt het element met de hoogste prioriteit.
- De prioriteit van de elementen is gebaseerd op een orderelatie die gedefinieerd is op de elementen.
  - Zoals we gaan zien in 7.2 en 8.5 kan dit in Java via de natuurlijke ordening of via een specifieke ordening.

Klasse-oefening

4. Een **object** is een *instantiatie* van een klasse [`new`], waarbij elk object eigen waarden voor elk attribuut heeft. Bij het instantiëren wordt de constructor van de klasse uitgevoerd.
   - De **default constructor** heeft geen parameters en een lege implementatie. Als er geen andere constructor bestaat, wordt de default constructor automatisch door Java aangemaakt.
   - Met `super(...)` op de *eerste lijn van je constructor* roep je één van de constructors van de superklasse op. Indien afwezig, wordt de default constructor opgeroepen. Deze moet dan wel bestaan voor de superklasse.
   - Op de *eerste lijn van een constructor* kan met `this(...)` een andere constructor van dezelfde klasse opgeroepen worden.

Voer de code van de main() stap-voor-stap uit en geef aan welke code uitgevoerd wordt

```
1. public class KlasseOefening2 {
2.       /** PROGRAMMA */
3.       public static void main(String[] args) {
4.                Kl1 o1 = new Kl1(5);
5.                Kl2 o2 = new Kl2(7);
6.                Kl3 o3 = new Kl3();
7.                o1.f();
8.                o2.f();
9.                o3.f();
10.      }
11. }
12. // welke klassen hebben de default constructor? ........
13. class Kl1 {
14.      int a;
15.      Kl1(){ // op welke lijn(en) wordt dit opgeroepen? ........
16.          this.a=5;
17.      }
18.      Kl1(int a){ // op welke lijn(en) wordt dit opgeroepen? ........
19.          this.a=a;
20.      }
21.      void f(){  // op welke lijn(en) wordt dit opgeroepen? ........
22.      }
23. }
24. class Kl2 extends Kl1 {
25.      Kl2(int x){ // op welke lijn(en) wordt dit opgeroepen? ........
26.          super(x);
27.      }
28.      void f(){  // op welke lijn(en) wordt dit opgeroepen? ........
29.      }
30. }
31. class Kl3 extends Kl1 {
32.      int b;
33. }
```

Oefening interfaces

7. Een **abstracte methode** heeft *geen implementatie* (deel tussen accolades), enkel een header [`abstract`]. De eerste lijn eindigt met een punt-komma.
8. Een **interface** is een klasse met *enkel abstracte methodes* [`interface`].
   - Een interface heeft geen constructor en geen attributen (al zijn statische attributen wel mogelijk).
   - Een klasse mag meerdere interfaces als superklasse hebben [`implements`]: "de klasse implementeert de interface".
   - Een concrete klasse moet alle abstracte methodes implementeren. Anders blijft het een abstracte klasse en kan je er geen objecten van maken.

Oefening op interfaces

```
// a. Welke concrete klasses zijn correct (zijn echt concreet)?
// b. Hoe maak ik ze concreet?
// c. Welke methoden zou ik mogen weglaten opdat het toch nog concrete klassen blijven?

interface I1 {
	void f();
	void g(int x);
}
interface I2 {
	int h();
	int k(int a);
}

class Cl1 implements I1{
	public void f(){
		...
	}
	public void g(){
		...
	}
	public int h(){
		...
	}
}
class Cl2 extends Cl1 implements I2 {
	public void f(){
		...
	}
	public void g(int x){
		...
	}
}
```

Gelinkte lijst
Linked list

Jan hoofdstuk 5_edit Anke_2 [Compatibility Mode] - Microsoft Word

Home Insert Page Layout References Mailings Review View Developer Add-Ins

Times New Roman 12

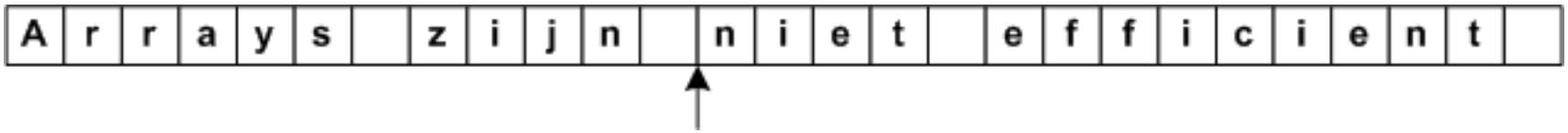
Paste Clipboard Font Paragraph Quick Styles Change Styles Editing Styles

Hoofdstuk 6

Gelinkte lijsten

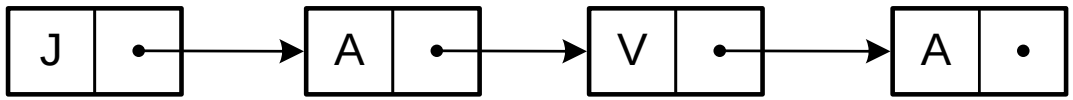
Arrays zijn niet echt efficiënt als je veelvuldig elementen wilt toevoegen en verwijderen. Stel dat we een gesorteerde lijst van namen bijhouden. Als we iemand willen toevoegen, moet dit op de juiste plek gebeuren. Voor zo'n *insert* moet er een plaatsje vrijgemaakt worden. Als je dit met een array doet, moeten alle elementen erna, naar achter worden opgeschoven. Omgekeerd geldt dat als je een element verwijdert, de elementen erna weer

Page: 1 of 9 Words: 2.220 Dutch (Belgium) 101%



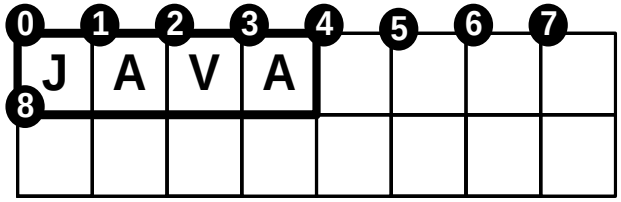
Linked List

- Flexibel toevoegen en verwijderen

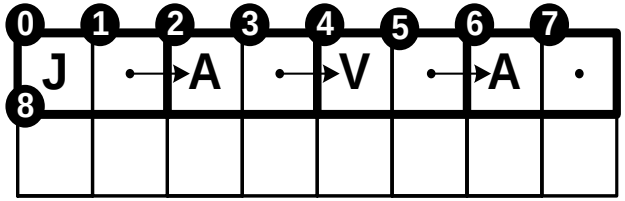


Geheugengebruik

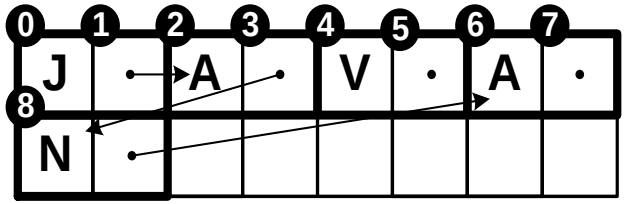
- Array



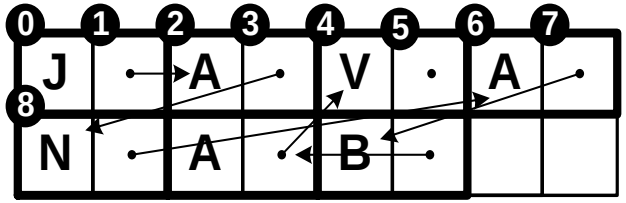
- Linked List



Geordend in 't begin



Door elkaar na enkele veranderingen



```
public class LinkedList <T> {
    class Link<T>{
            T data;
            Link<T> next;

            Link(T data){
                    this.data = data;
                    this.next = null;
            }
            Link(T data, Link<T> next){
                    this.data = data;
                    this.next = next;
            }
    }

    Link<T> first;
    public LinkedList(){
            first = null;
    }
}
```

Java's LinkedList

Constructor Summary

LinkedList()

Constructs an empty list.

LinkedList(Collection<? extends E> c)

Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.

Method Summary

boolean **add**(E e)

Appends the specified element to the end of this list.

Toevoegen van een bepaald element aan het einde

```
public void append(T data){
        Link<T> el = new Link<T>(data);
        if (first == null){
                first = el;
        } else {
                // zoek laatste link
                Link<T> last = first;
                while (last.next != null)
                        last = last.next;
                last.next = el;
        }
}
```

Printen van lijst

```
public String toString(){
        Link<T> link = first;
        String str = "[";
        boolean eerste = true;
        while (link != null){
                if (eerste)
                        eerste = false;
                else
                        str+=", ";
                str += link.data;
                link = link.next; // ga naar volgende
        }
        str+="]";
        return str;
}
```

Vinden van een bepaald element

```
public T get(int index){
        int i=0;
        Link<T> link = first;
        while (link != null && i < index){
                link = link.next;
                i++;
        }
        if (link == null)
                return null;
        else
                return link.data;
}
```

- *Nadeel*: heel de lijst moet doorlopen worden
- Bij arrays kan je onmiddelijk naar de index springen!

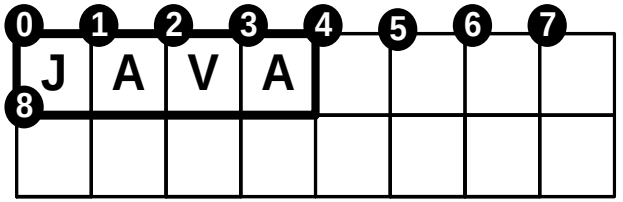
Vinden van het *i*de element

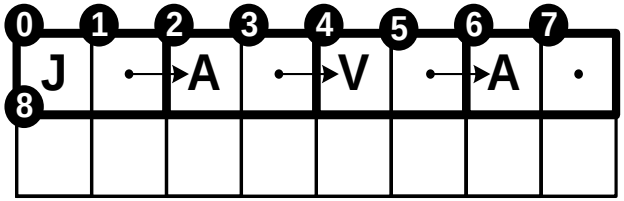
- Array

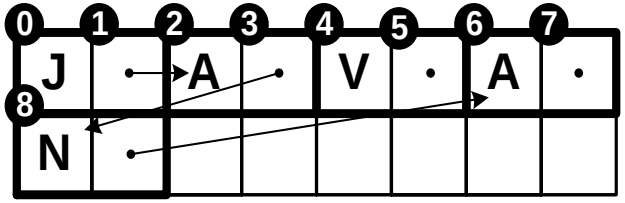
= beginvakje + i

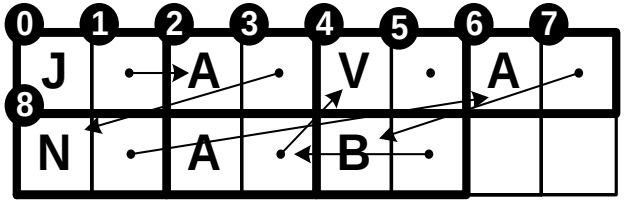
- Linked List

Je bent van in het begin niet zeker waar het i-de element zich bevindt in het geheugen, zeker nadat elementen zijn toegevoegd en verwijderd





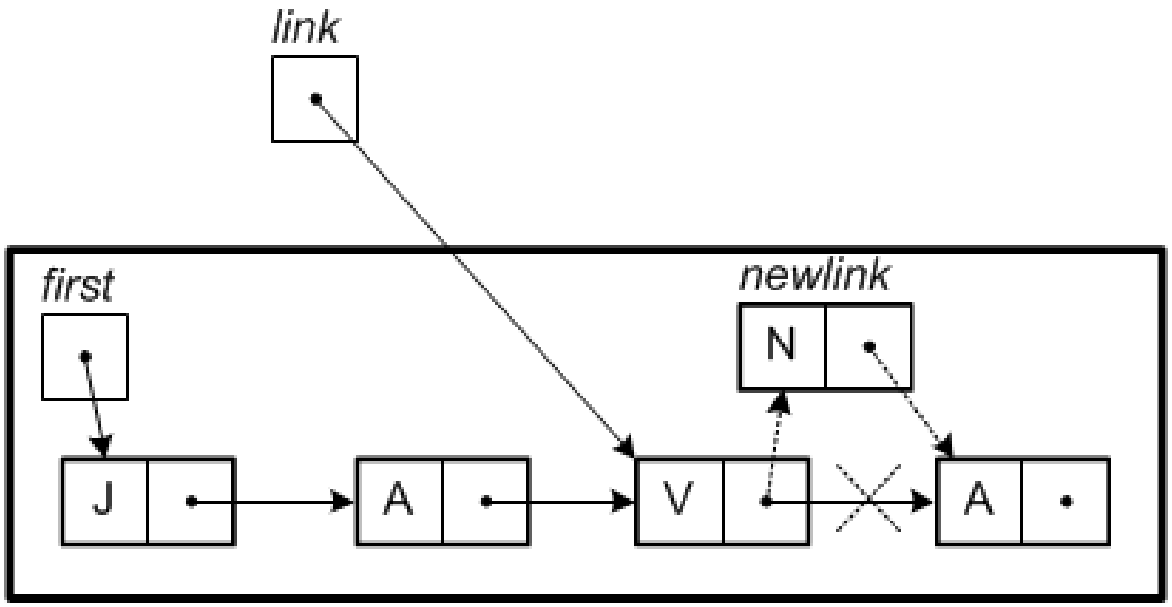




Inlassen van een element

```
public void insert(T data, int index){
    Link<T> newlink = new Link<T>(data);
    if (index == 0){
            // wordt eerste element
            newlink.next = first;
            first = newlink;
    } else {
            int i=0;
            Link<T> link = first;
            while (link != null && i < index - 1){
                    link = link.next;
                    i++;
            }
            if (link == null)
                    throw new IllegalArgumentException("Insert at
"+index+" impossible; list has only "+(i+1)+" elements.");
            newlink.next = link.next;
            link.next = newlink;
     }
}
```

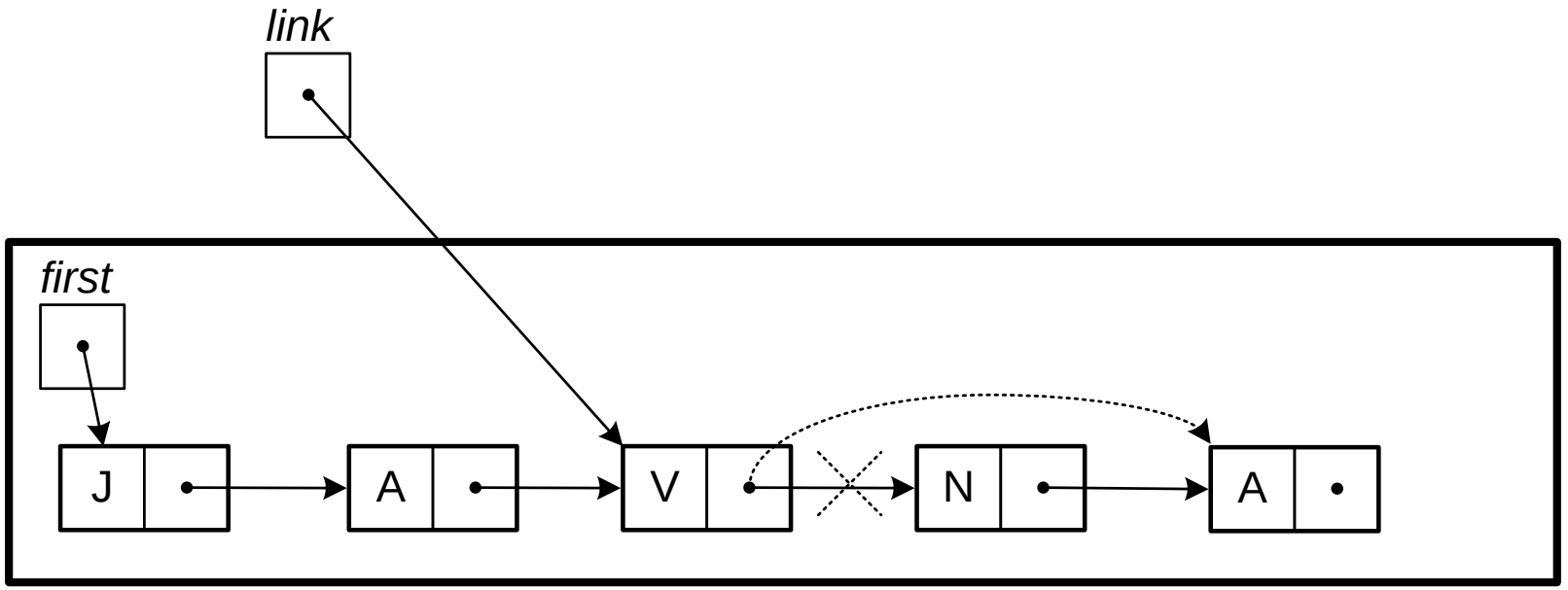
Inlassen van een element



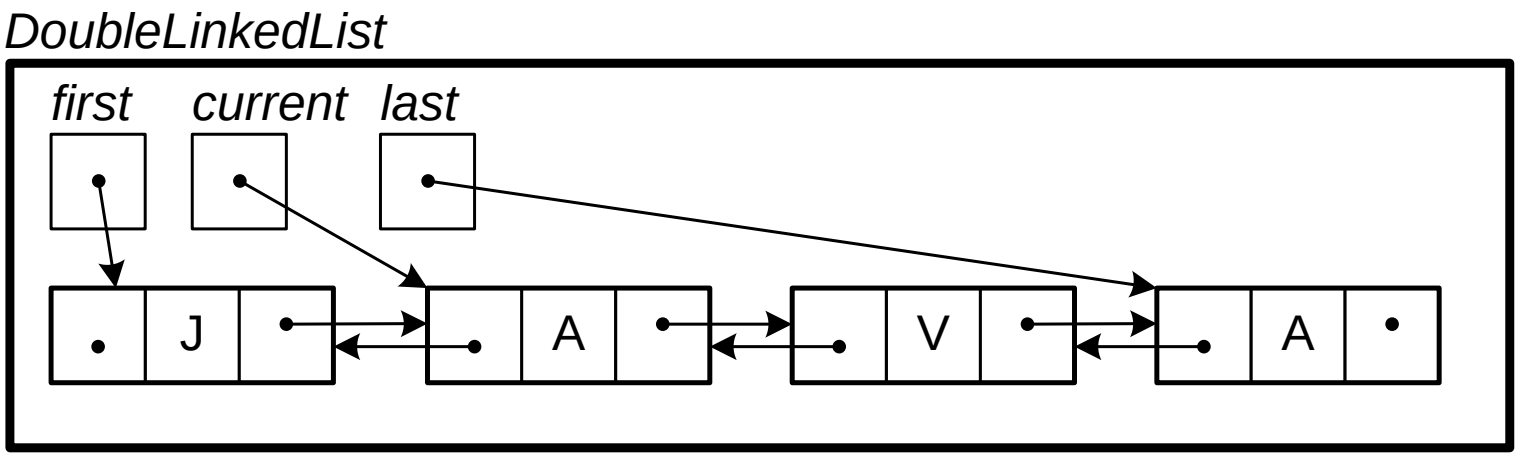
Verwijderen van een element

```
  public void remove(int index){
        if (index == 0){
              if (first == null)
                    throw new IllegalArgumentException("Removal
of first element impossible; list has no elements.");
              first = first.next;
        } else {
              int i=0;
              Link<T> link = first;
              while (link != null && i < index - 1){
                    link = link.next;
                    i++;
              }
              if (link == null || link.next == null)
                    throw new IllegalArgumentException("Removal
of element "+index+" impossible; list has only "+(i+1)+"
elements.");
              link.next = link.next.next;
        }
  }
```

Verwijderen van een element



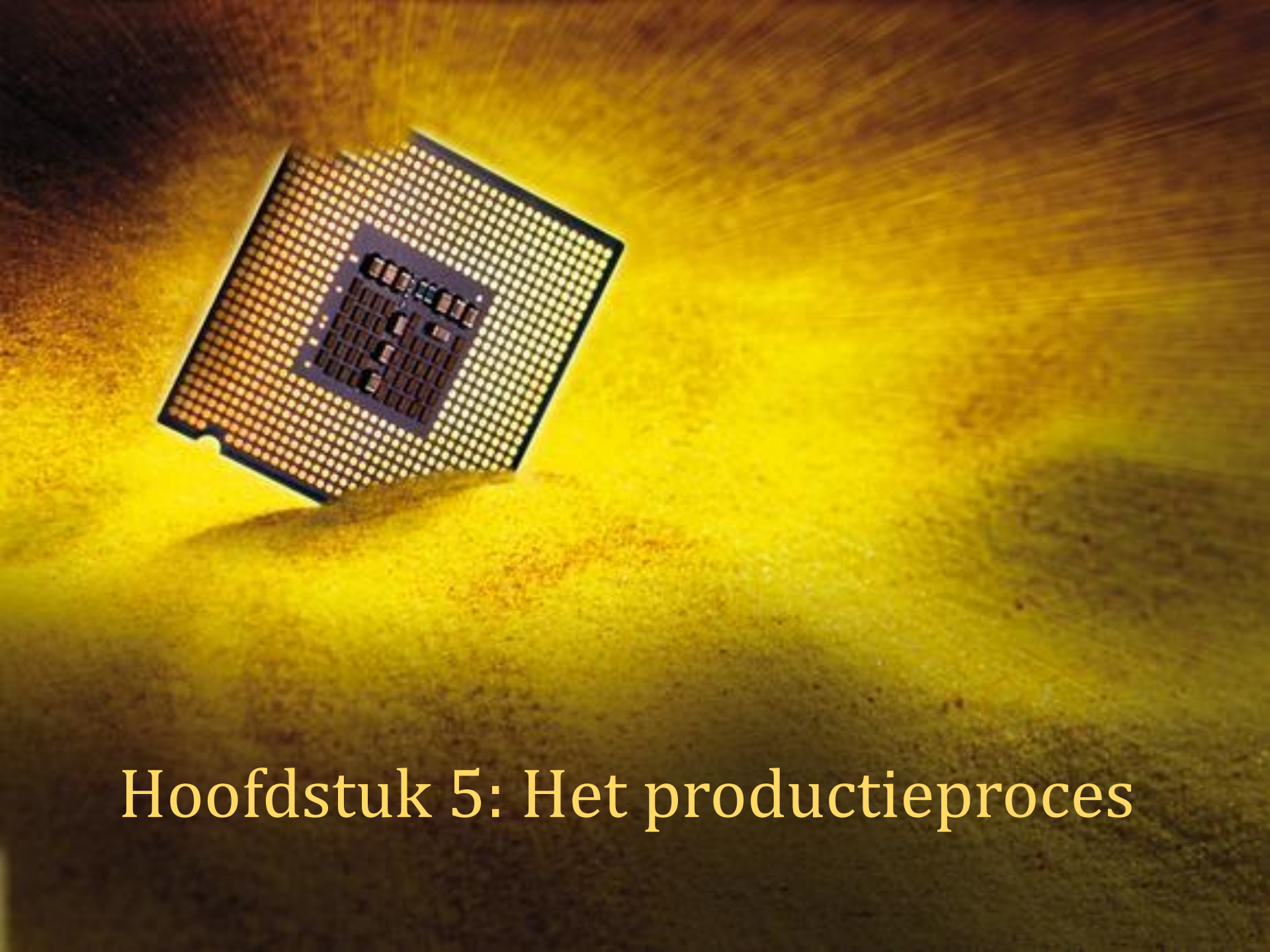
Dubbele gelinkte lijst



- *Voordelen*:
  - einde gemakkelijk te vinden
  - houdt huidige positie bij
- *Toepassing*: word

Niet te kennen

Hoofdstuk 4: Computerarchitectuur



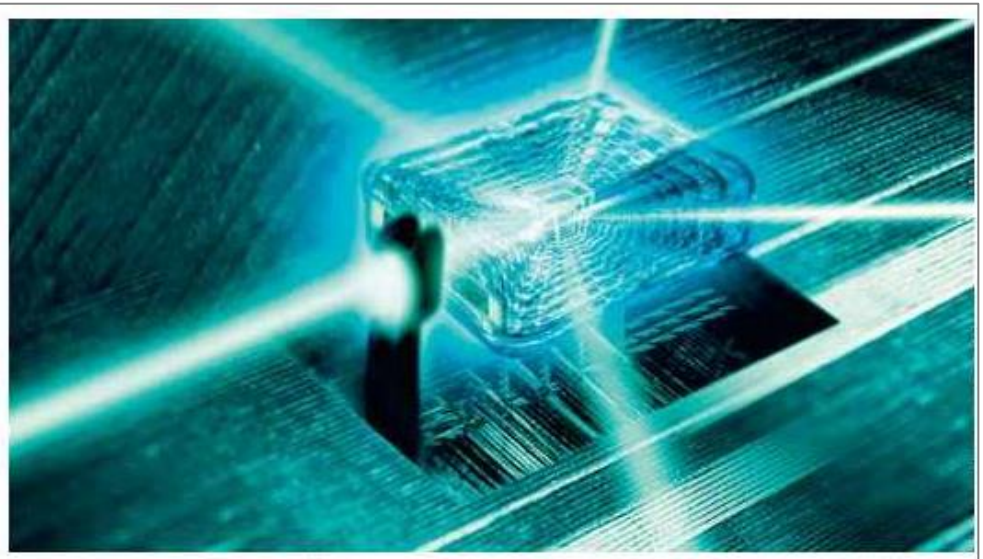
Hoofdstuk 5: Het productieproces

Waar zijn de chips?

Natuurkunde

Licht opsluiten

Door licht in cirkels te buigen kan het in piepkleine volumes verdwijnen.



FOTONICA *Lichtdeeltjes maken snellere computers mogelijk dan elektronica.*

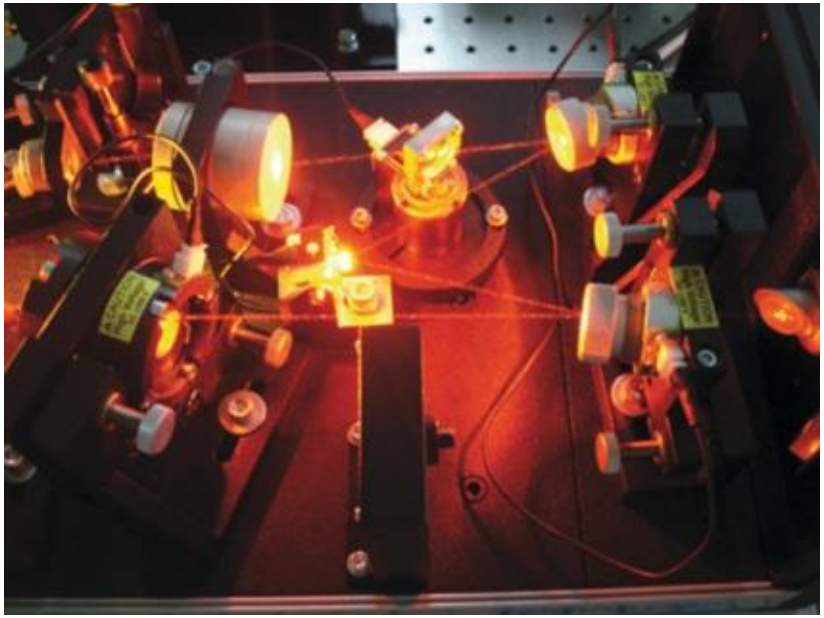
Omdat er niets is wat sneller beweegt dan licht, hopen wetenschappers van licht gebruik te kunnen maken om informatie over te dragen. Want dat zou dan nog sneller kunnen dan met de klassieke elektrische systemen van nu het geval is.

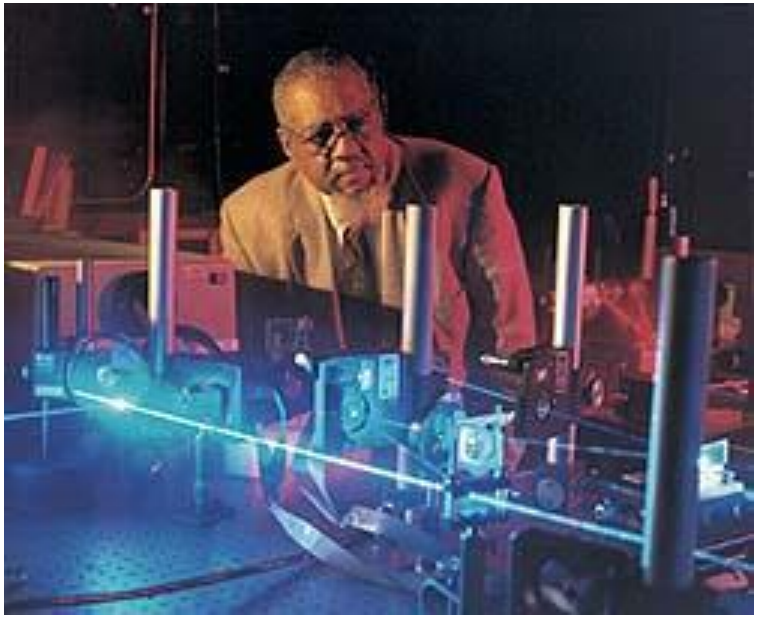
Maar lichtdeeltjes zijn minder gemakkelijk te controleren dan de elektronen uit onze dagelijkse elektronica, waardoor ze vooral op heel kleine schaal (zoals in de computerchipindustrie) moeilijk te gebruiken zijn. Daar komt echter stilaan verandering in. Fysicus Vincent Ginis van de Vrije Universiteit Brussel beschrijft met een aantal collega's in de *New Journal of Physics* een manier om licht op te sluiten in héél kleine eenheden, waardoor een verdere miniaturisatie van de fotonica (als tegenhanger van de elektronica) mogelijk wordt.

Hun idee is geïnspireerd door de recente ontwikkeling van een 'onzichtbaarheidsmantel', waarbij licht op zo'n vloeiende manier rond een voorwerp gebogen wordt dat het onzichtbaar wordt. Door dat principe om te keren kan licht op een vloeiende manier in cirkels worden gebogen, zodat het in piepkleine volumes kan worden opgesloten.

Hoe kleiner de componenten, hoe meer er op een chip kunnen.

Optische computer?





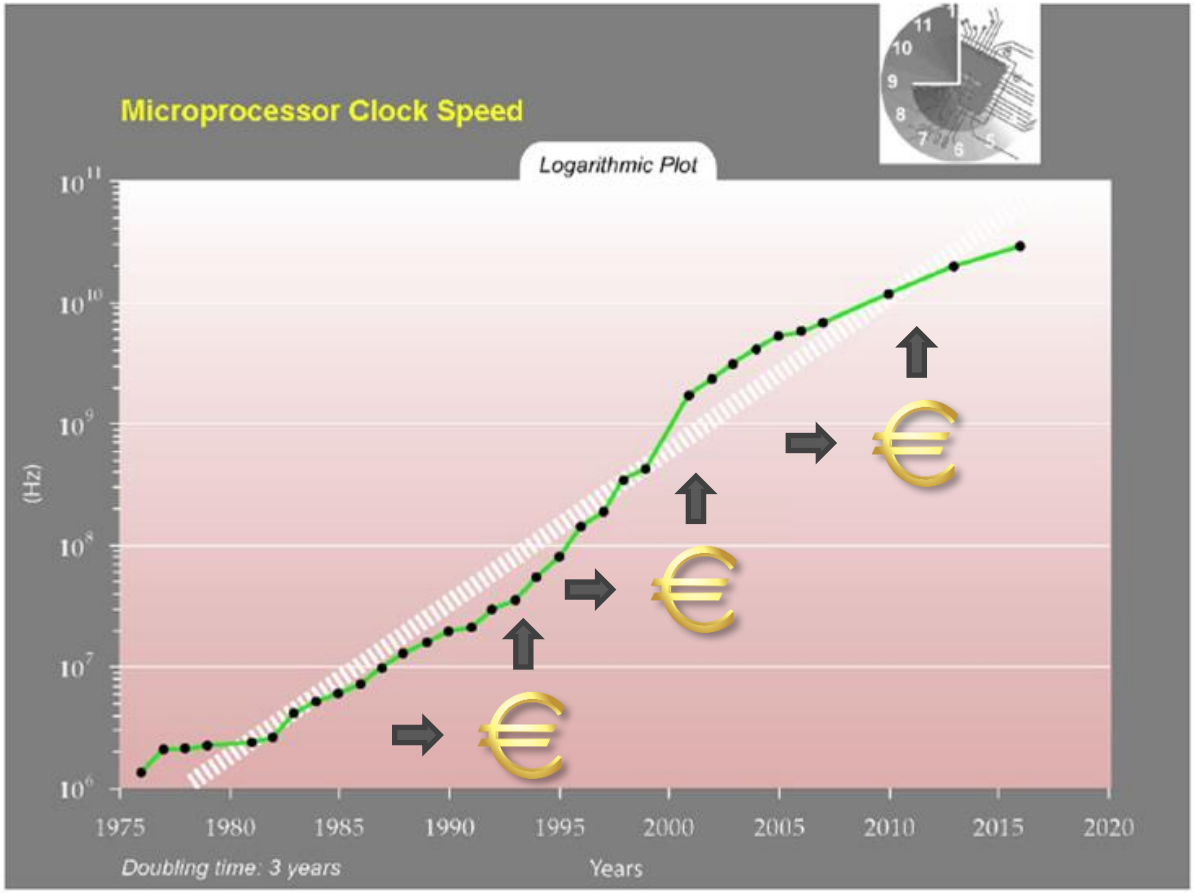
- *Maar…*

Ook aan de VUB wordt er onderzoek gedaan naar een optische computer. Ik zie echter twee grote problemen waardoor deze waarschijnlijk nooit op de markt zal komen.

Probleem 1: Electrische computer heeft al een hele ontwikkelingsproces doorgemaakt

Een alternatief moet wedijveren met de huidige processorkracht , de ontwikkeling van deze werd gefinancierd door de verkoop van minder krachtige processoren

Een alternatief heeft deze mogelijkheid niet …



Aanverwant thema: de beste technologie is niet de beste optie voor de markt

- De eerste iPhone of iPad die Apple op de markt bracht was niet het beste wat Apple te bieden had, maar ze was wel goed genoeg om de markt te veroveren.
- Zo kon Apple later met een verbeterde versie uitpakken (bvb betere camera) en weer langs de kassa passeren…
  - Tip: wacht minstens tot 2e versie

Probleem 2

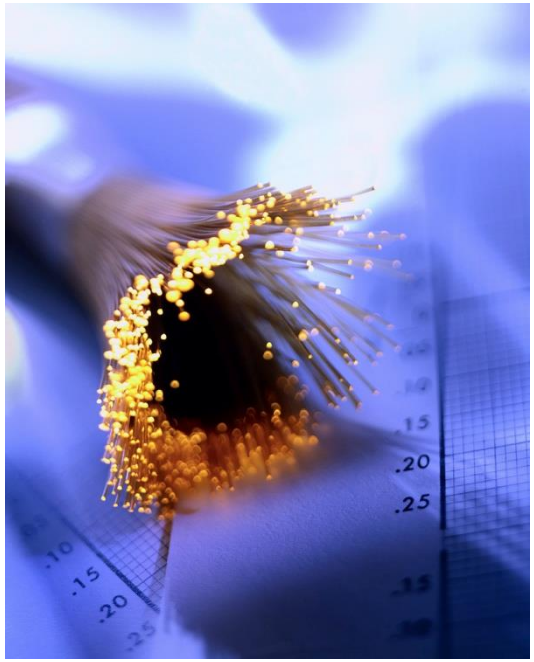
Geen goedkoop, eenvoudig productieproces

- lenzen zijn foutgevoelig, niet te miniaturiseren

Electrische computer heeft dat wel, gebruik makend van micro-electronica

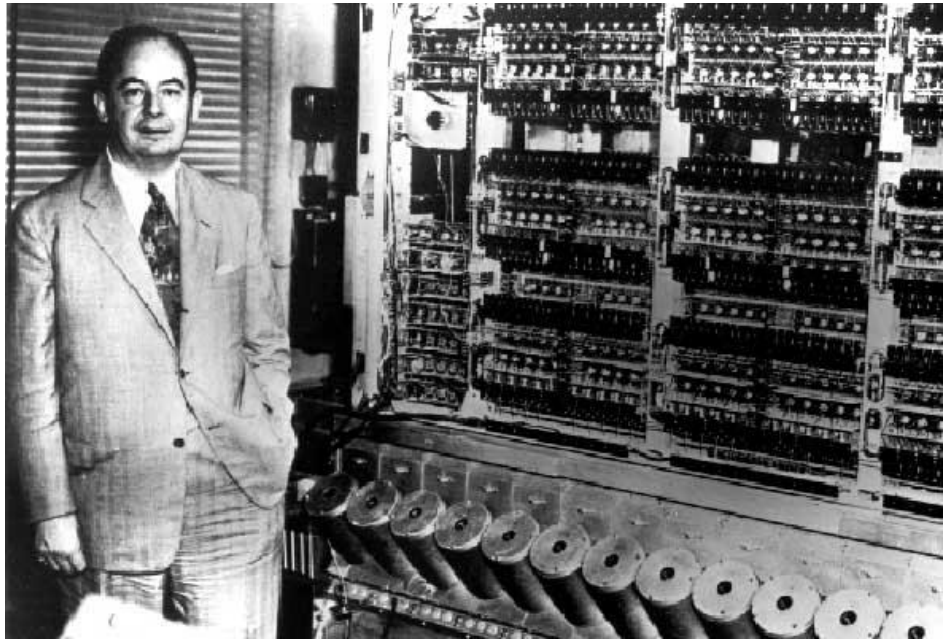
Wel: communicatienetwerken gaan via licht

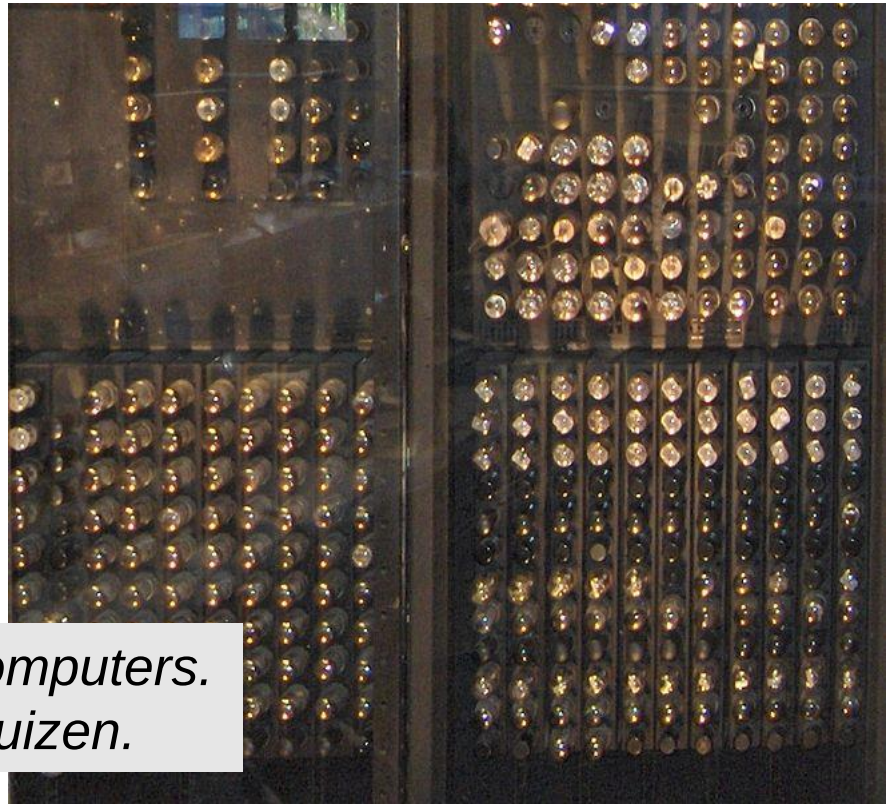
- **Glasvezel**



- The world's cable map: http://www.cablemap.info/
- Ook voor communicatie **in** computersystemen

De doorbraak van de computertechnologie

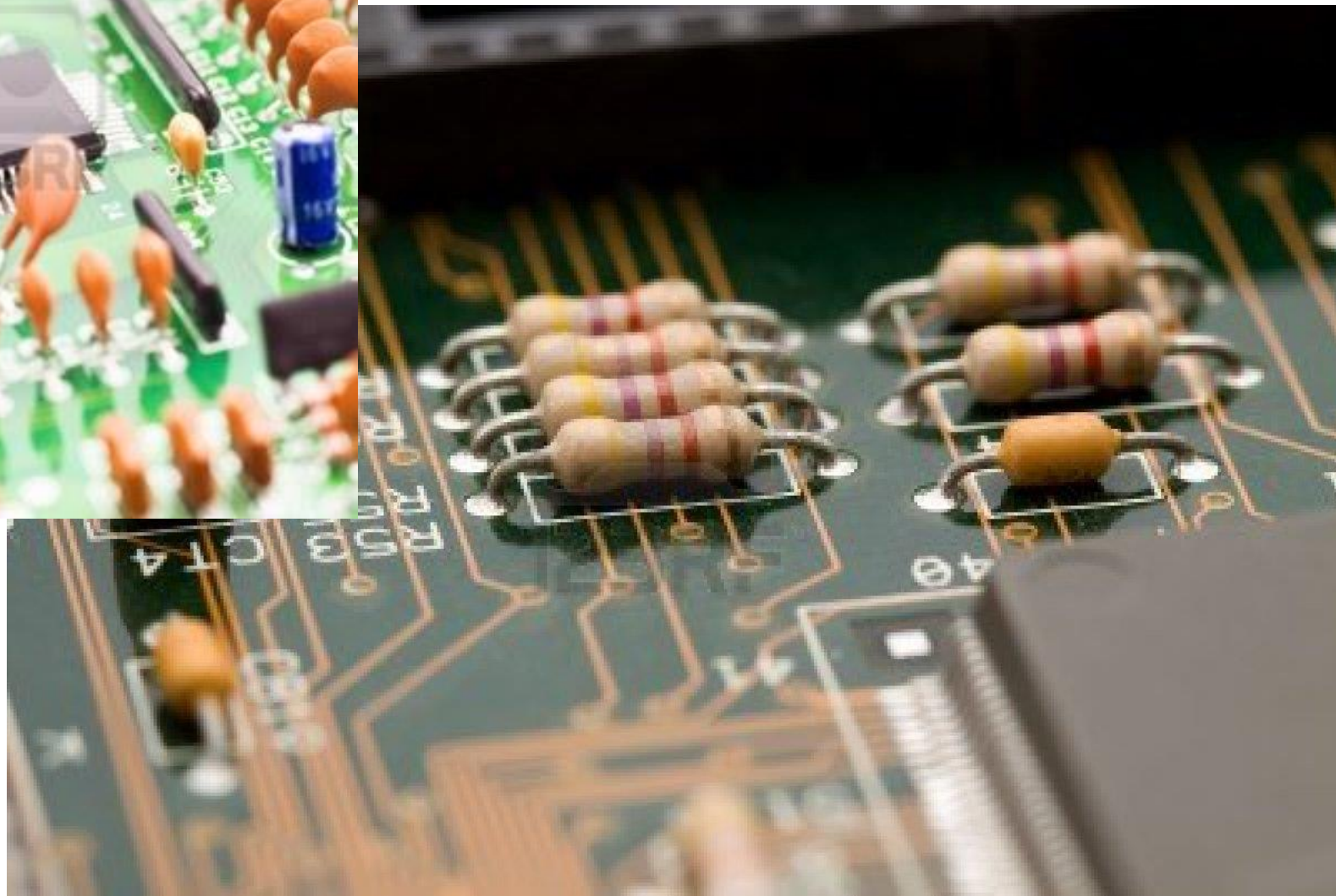




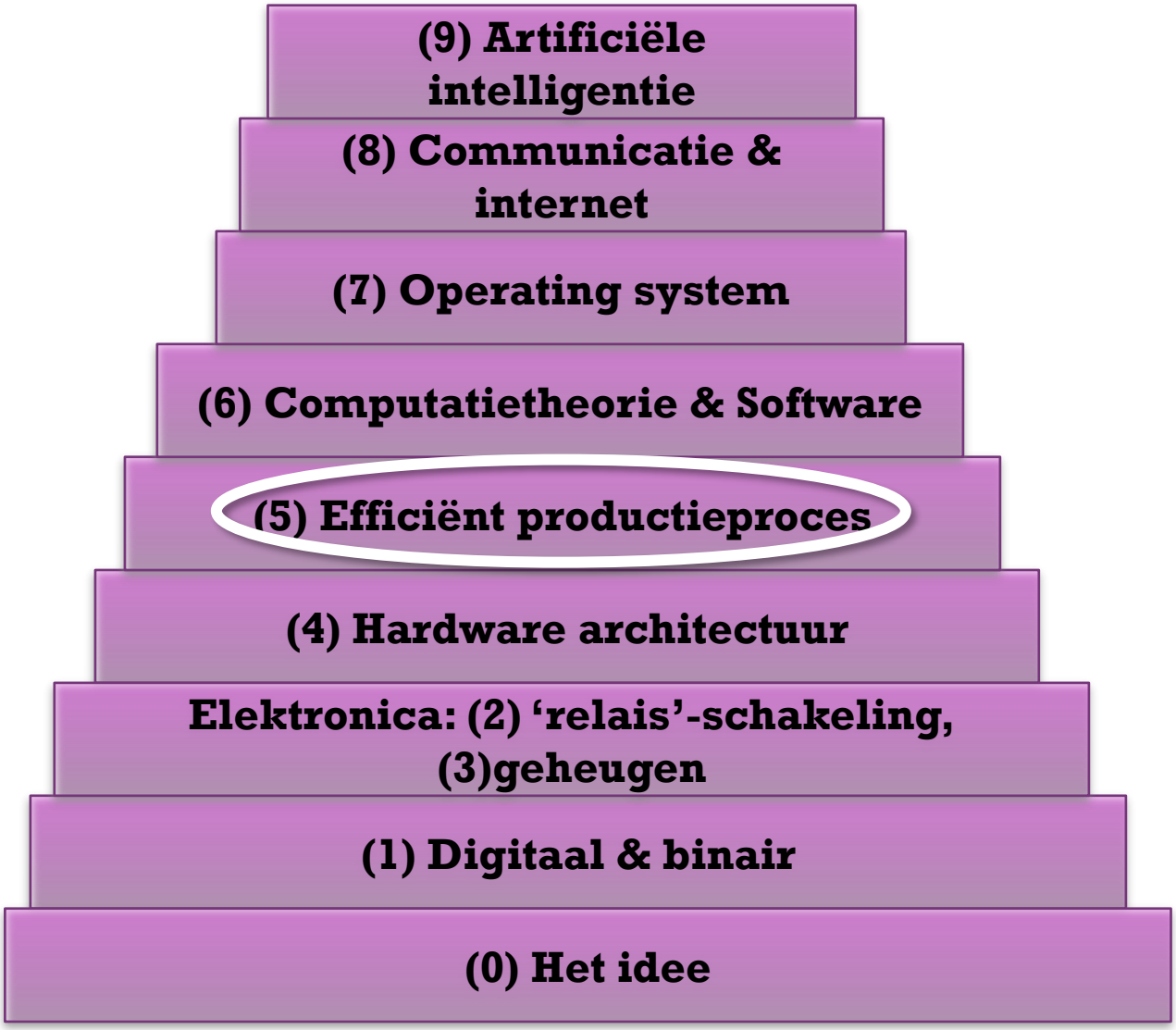
Von Neuman met één van de eerste computers.
Rechts de achterkant met de vacuümbuizen.

De vacuümbuizen worden vervangen door chips

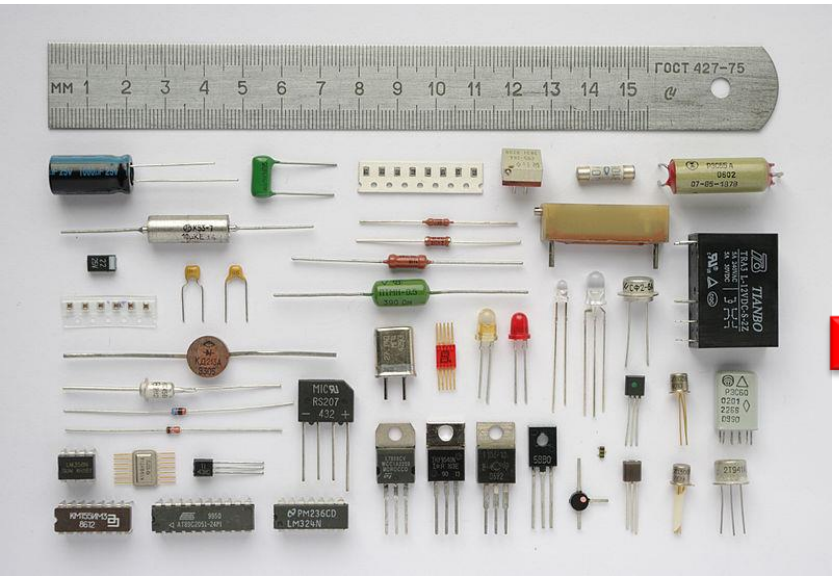
Componenten op een bordje
met ingebakken connectielijnen

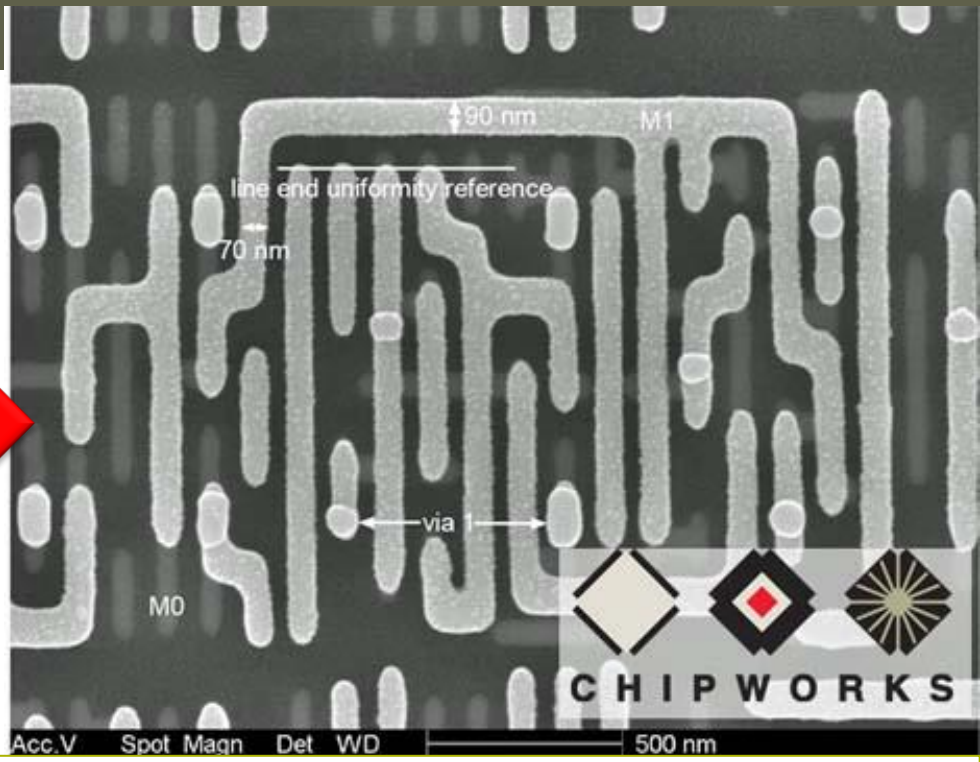


Waarmaken van Leibniz's droom



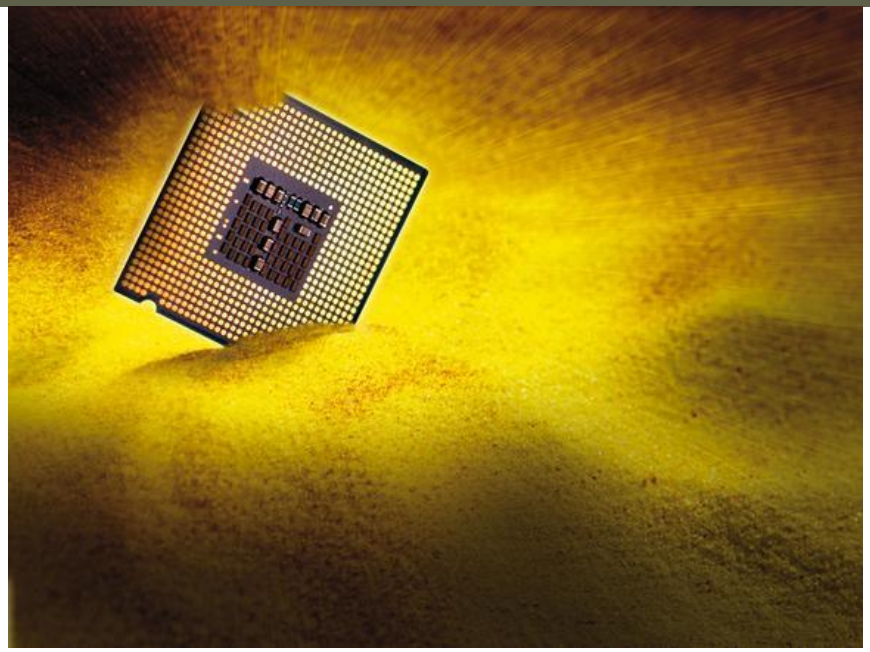
Alle componenten op een **chip**



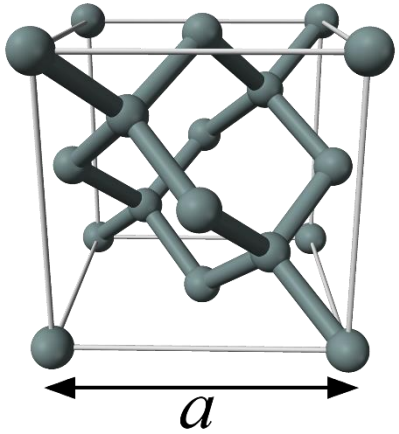


In plaats van losse componenten die geconnecteerd moeten worden, wordt er 1 plaatje gemaakt waarvan delen ‘gemuteerd’ zijn tot geleiders of halfgeleiders. Zo worden er weerstanden, capaciteiten en transistoren ‘ingebakken’ in het plaatje alsook de connectoren die de componenten connecteren. Er wordt gestart met een ‘wafer’ van silicium (element Si – atoomnummer 14). Vervolgens wordt er adhv mallen geëtst in het wafer en worden er andere elementen geïnjecteerd (dopering genoemd) zodat een geleider of halfgeleider ontstaat.

Zand…



bevat het wonderlijke Silicium, de basis van de chip



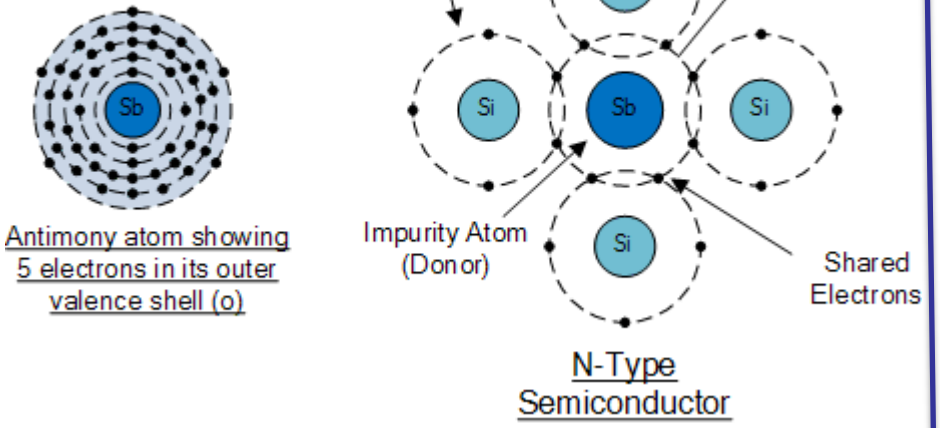
a = 0.5431nm

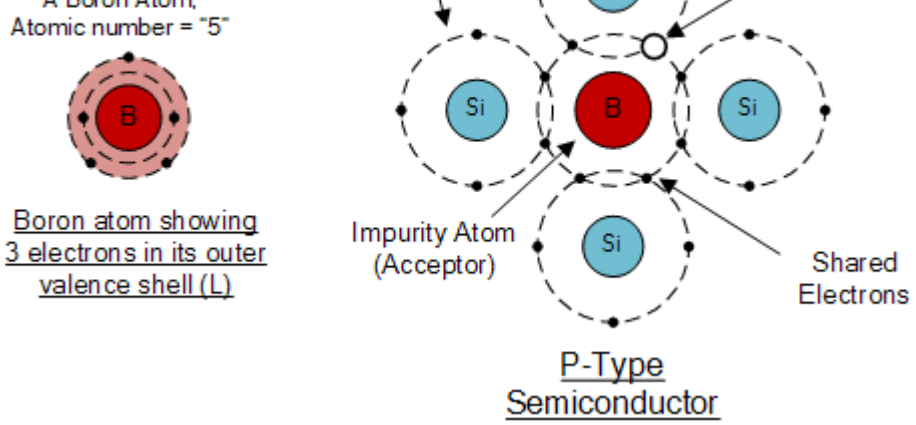
Halfgeleiders

Periodic Table of Semiconductors

Elements Group 13	Elements Group 14	Elements Group 15
3-Electrons in Outer Shell (Positively Charged)	4-Electrons in Outer Shell (Neutrally Charged)	5-Electrons in Outer Shell (Negatively Charged)
(5) Boron (B)	(6) Carbon (C)	
(13) Aluminium (Al)	(14) Silicon (Si)	(15) Phosphorus (P)
(31) Gallium (Ga)	(32) Germanium (Ge)	(33) Arsenic (As)
		(51) Antimony (Sb)

Doperen met neven-atomen





Antimoon

- 'Voelt' 9 electronen rond zich

=> *Eentje te veel om OK te zijn…*

N

Boron

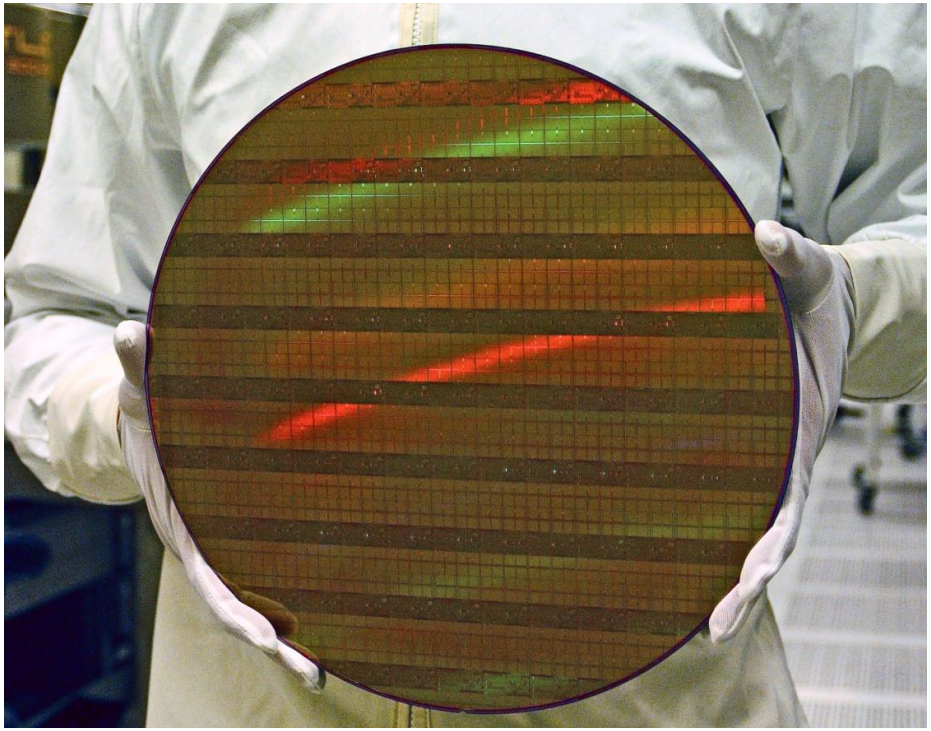
- 'Voelt' 7 electronen rond zich

=> *Eentje te weinig…*

P

Silicium-’wafer’

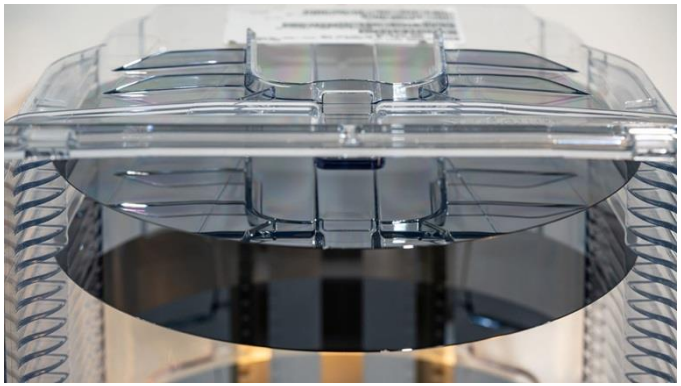
- Principieel ‘eenvoudig’ productieproces
- te miniaturiseren tot enkele nanometers
- https://www.asml.com/en/technology/all-about-microchips/how-microchips-are-made

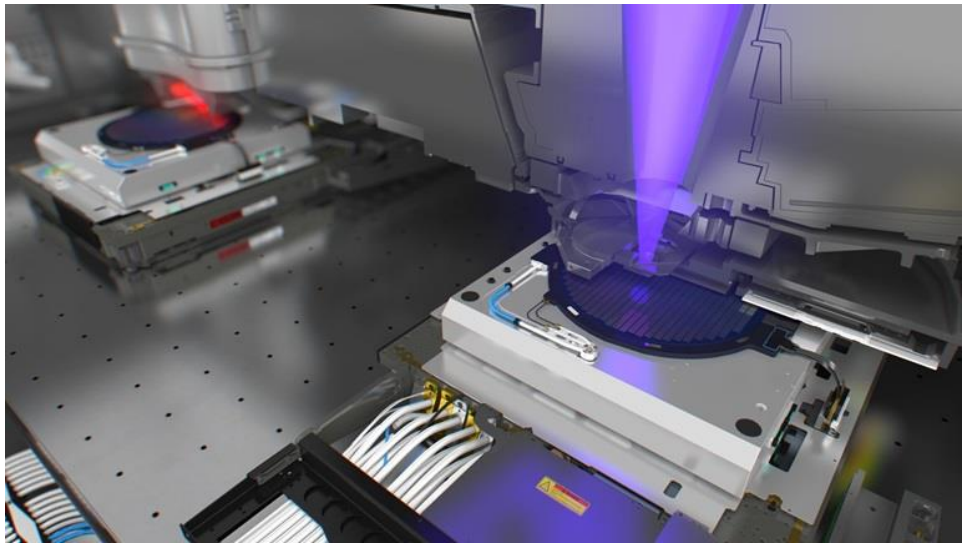


Gefabriceerde wafer bevat tientallen chips

Productieproces chip

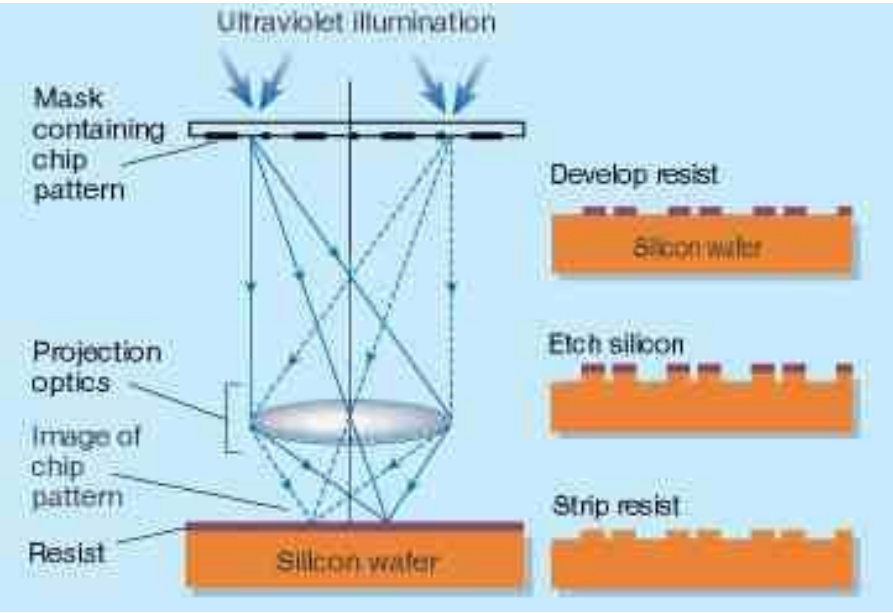
- Stap 0: Silicium wordt gesneden tot een ronde plaat
- Stap 1: aanbrengen van lichtgevoelig materiaal (resist) op het Silicium
- Stap 2: wegbranden met een laser van het lichtgevoelig materiaal volgens een patroon (mal of masker).





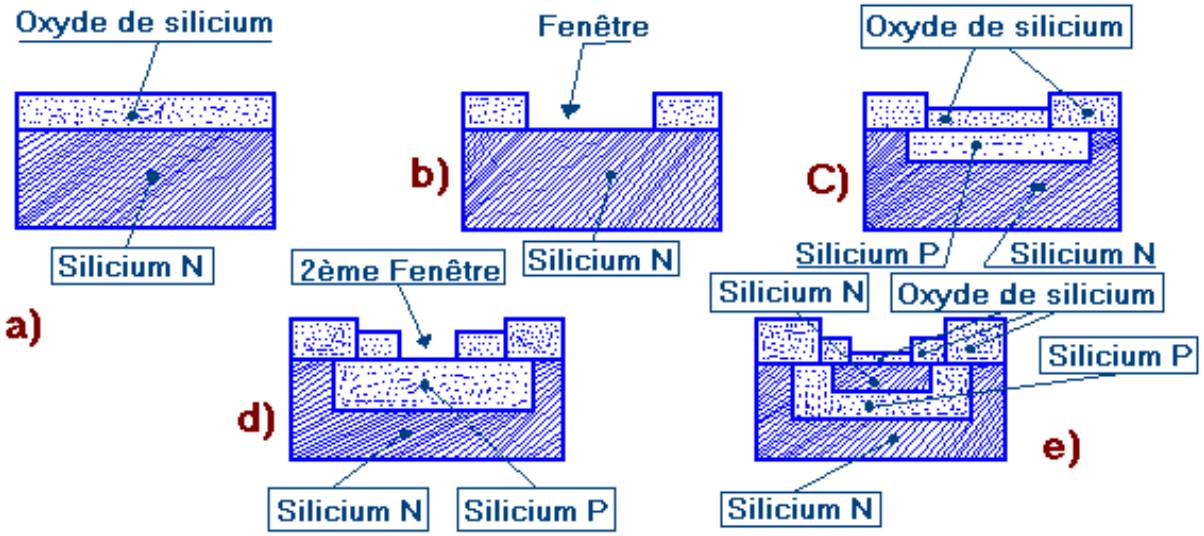
Etsen en vervangen

- Stap 3a: een laagje silicium wordt afgeschraapt (etsen)
- het weggeschraapt laagje silicium wordt vervangen door ander materiaal (door gasmoleculen te laten neerslaan)
- De rest van het resist wordt verwijderd



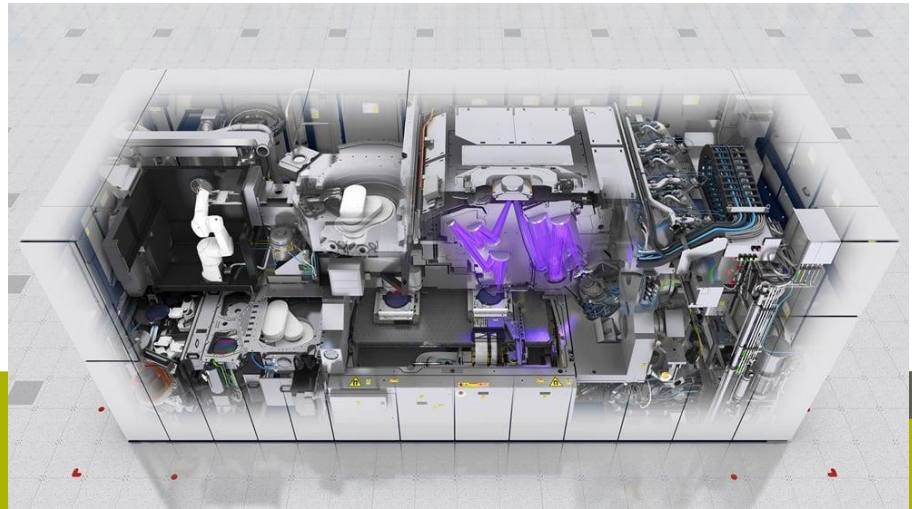
Doperen

- Stap 3b: ionen worden in het rooster geschoten
- De ionen bepalen de werking van de halfgeleider (N of P-materiaal op slide).
  - N-ionen: 5 elektronen op buitenste schil (rechts van Si in periodieke tabel)
  - P-ionen: 3 elektronen op buitenste schil (links van Si in periodieke tabel)

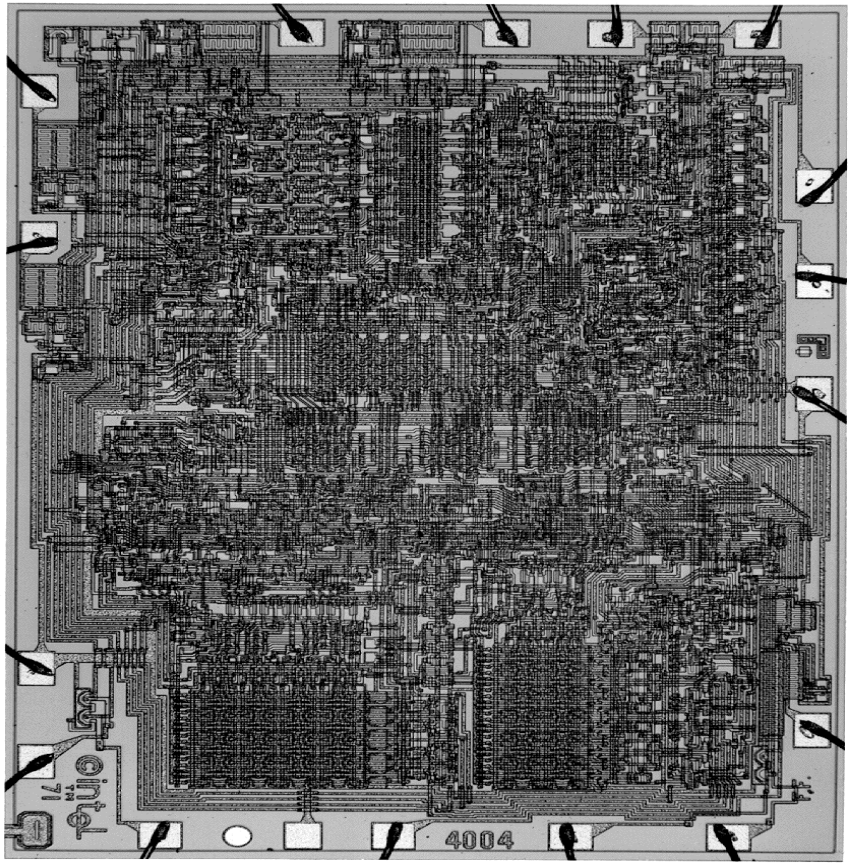


Laag per laag

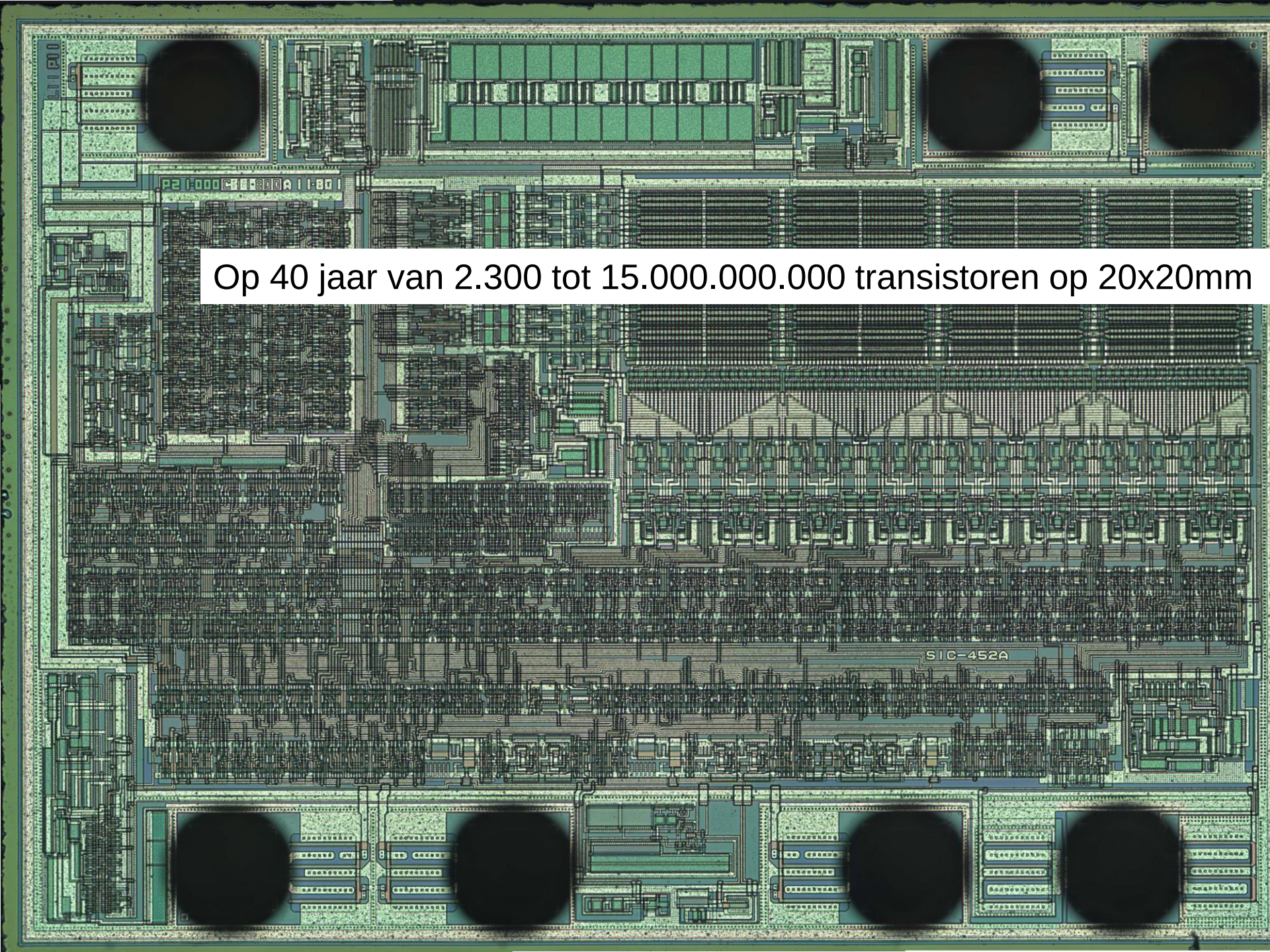
- Dit procede wordt tot 100x herhaald om het schema met de elektronische componenten laag per laag in de chip te produceren.
- De chips worden dan uit de wafer gesneden met een diamanten ‘zaag’.
- Vervolgens krijgt elke chip een beschermende package.
- We bekomen een geïntegreerd circuit waarbij alles in 1 plaatje zit.



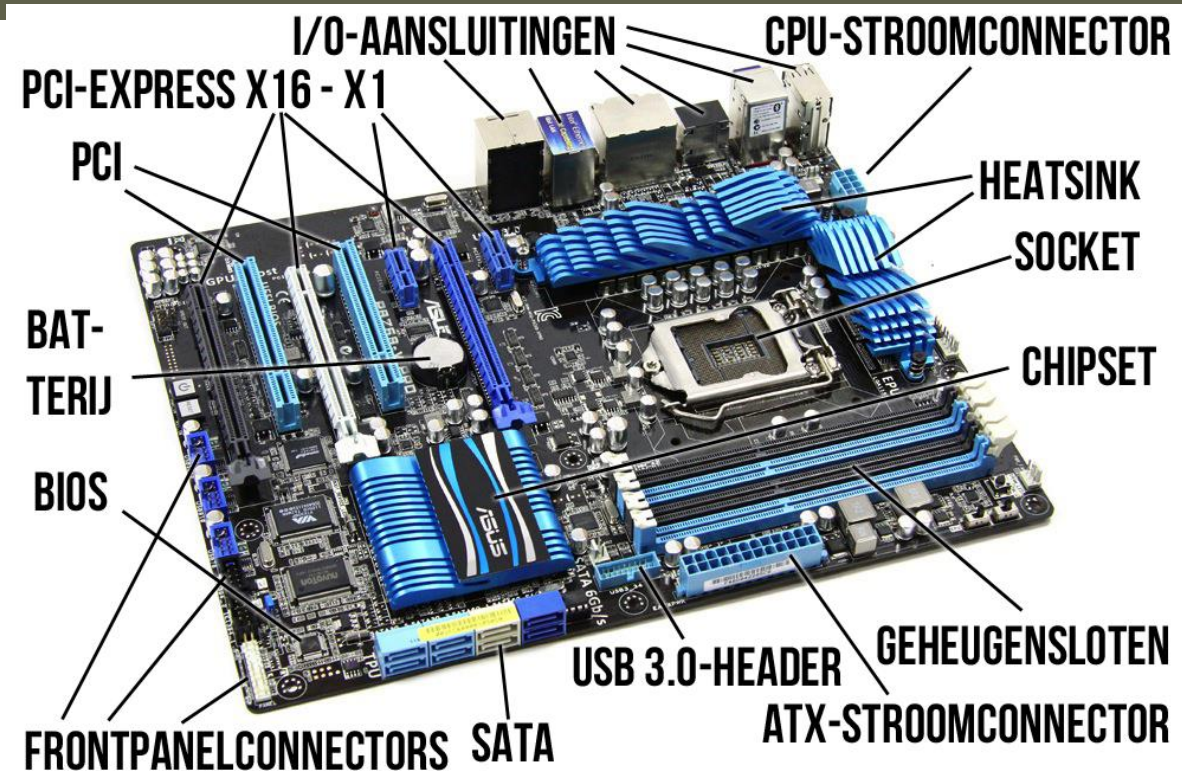
Een ‘oudtje’: de Intel 4004 microprocessor (1971)



1000 transistoren op
1MHz klokfrekwentie

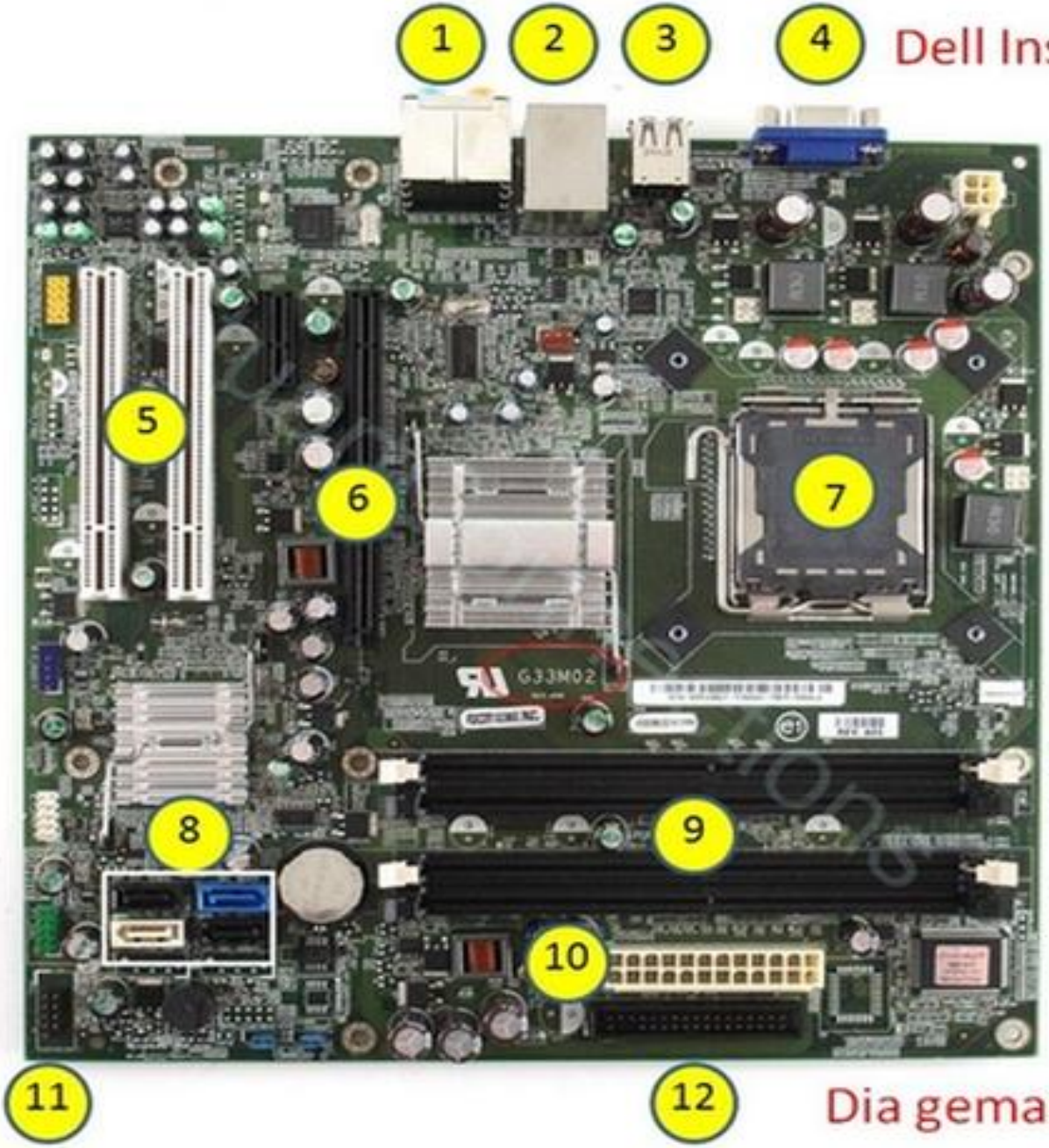

Op 40 jaar van 2.300 tot 15.000.000.000 transistoren op 20x20mm

Moederbord van je PC



De processor bevindt zich op het moederbord van je PC (schema: in de socket), waar het geconnecteerd wordt met het RAM-geheugen (schema: geheugensloten) en de connecties naar de buitenwereld (muis, toetsenbord en USB, PCI voor andere devices). Verder: de ‘heatsink’ neemt de warmte weg, de BIOS bevat de basisinstellingen voor de opstart van je computer.

Dell Inspiron 530 Moederbord



1 - Aan sluiting geluid

2 - Internet aansluiting

3 - 2 USB aansluitingen

4 - Video op moederbord

5 - Extra pci kaart aansluitingen

6 - Videokaart aansluiting

7 - Plaats van de Processor

8 - 4 Disks Sata aansluitingen

9 - plaats 4 geheugenkaartje

10 - Voeding aansluiting

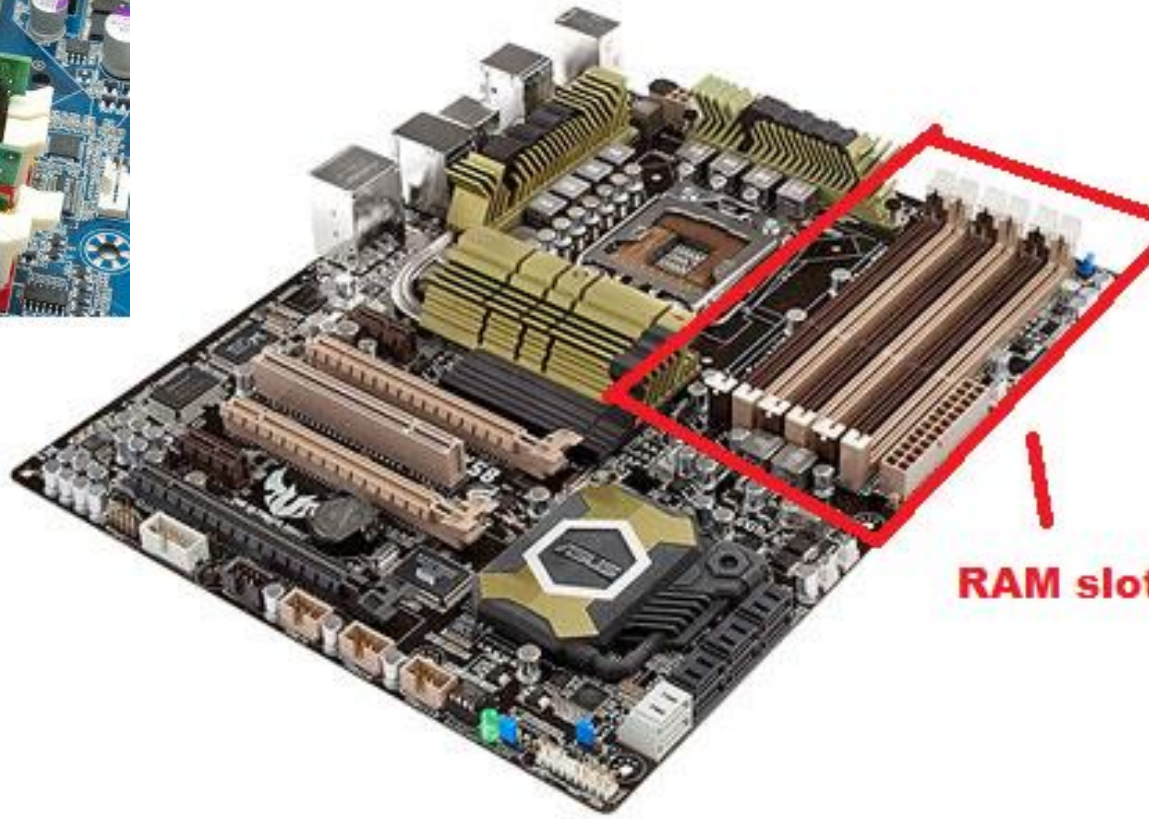
11 - Naar USB voorkant Pc

12 - Aansluiting oude diskettes

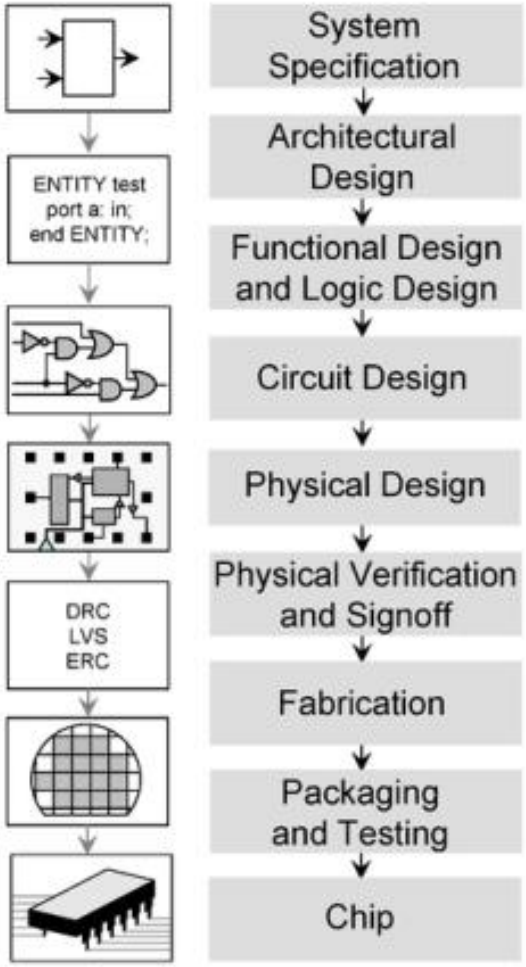
Dia gemaakt in PowerPoint 2007

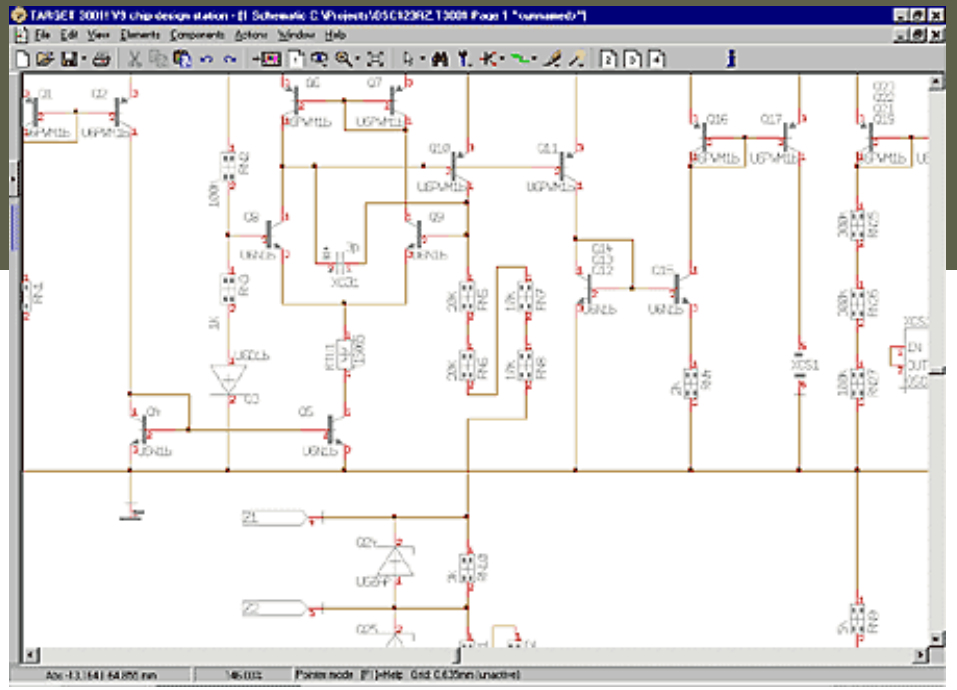
Zelf toevoegen van RAM-geheugen

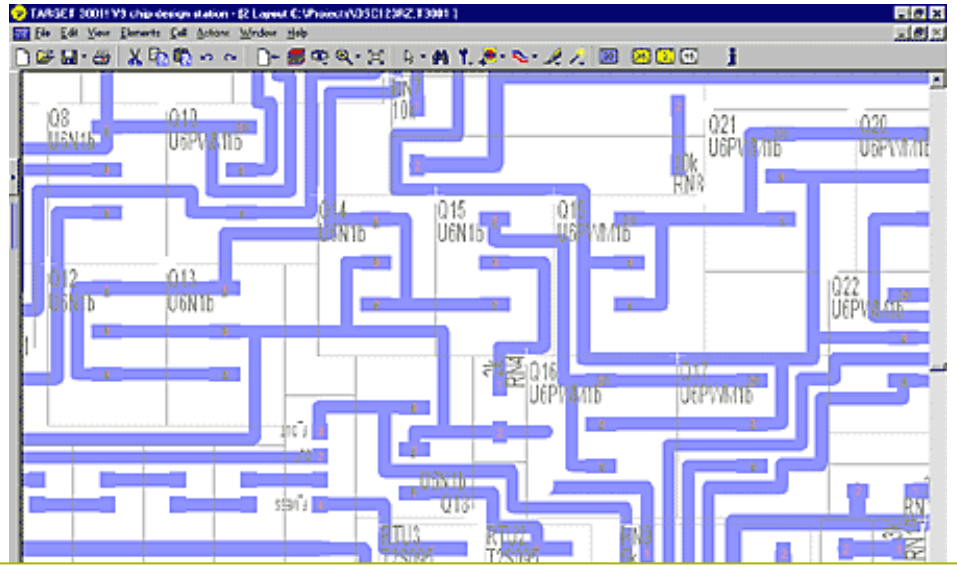
www.shutterstock.com · 12654331



Chip design

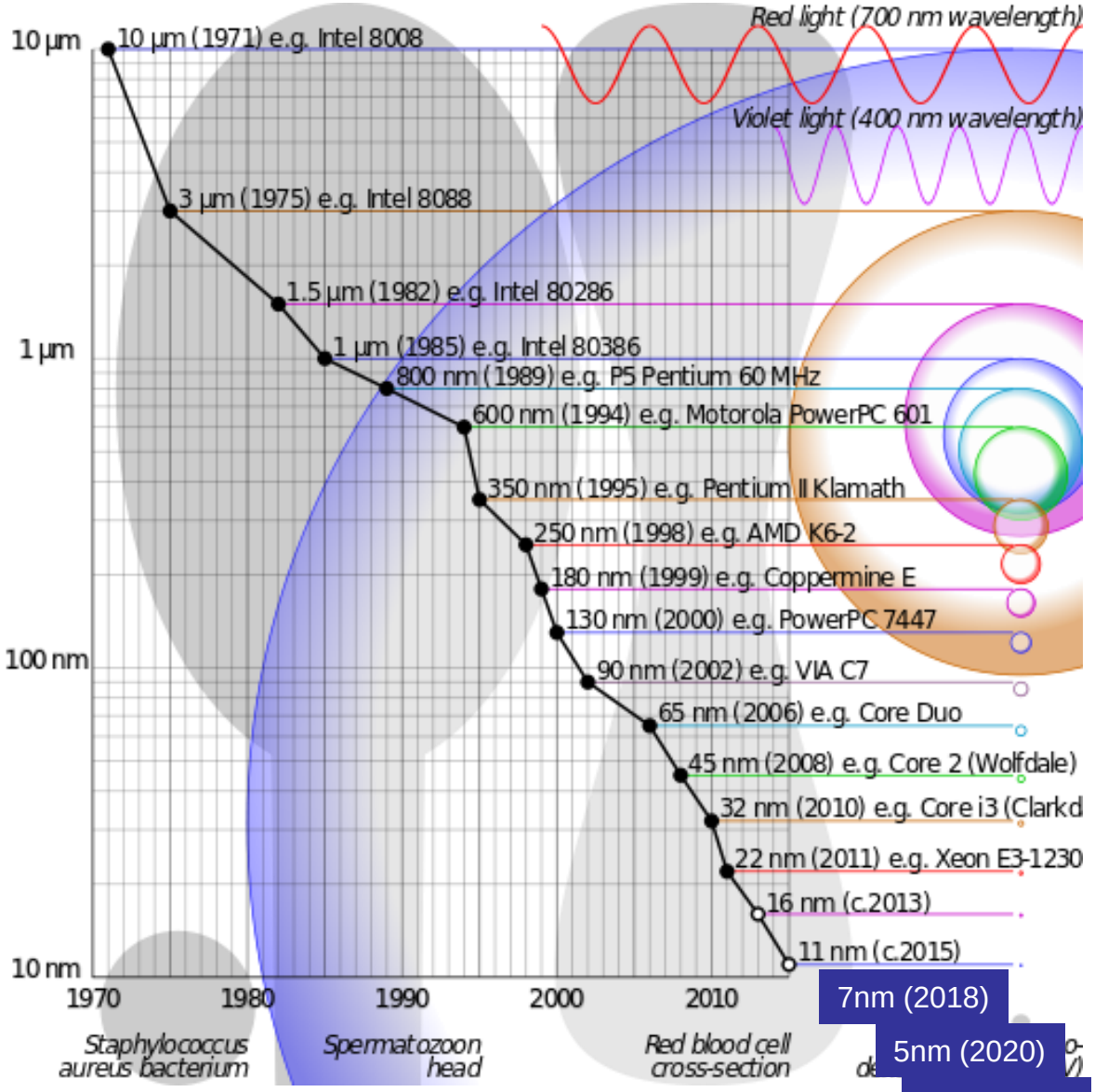






Het aanmaken van een chip gebeurt volautomatisch vertrekkende van je ontwerp van het electronisch circuit. Voor het ontwerp bestaat modern software.

Miniaturisatie



7nm (2018)

5nm (2020)

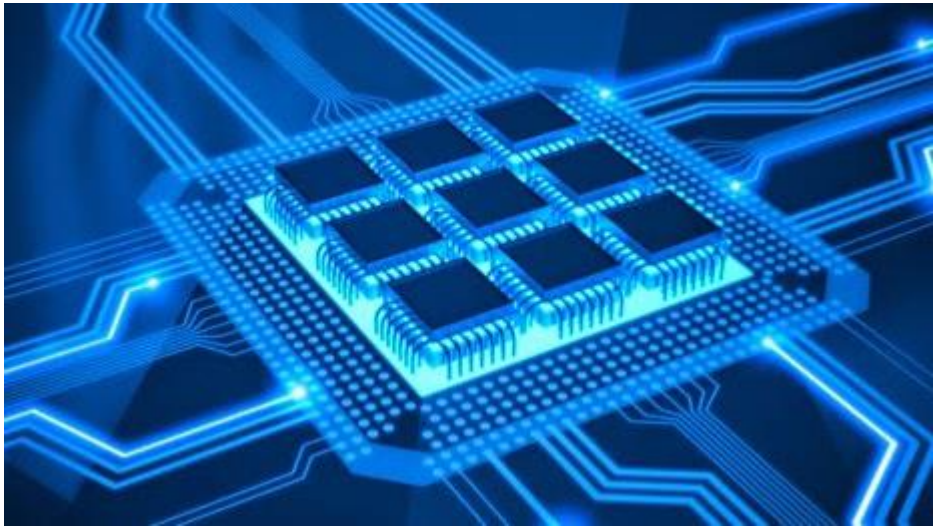
3nm (2022)

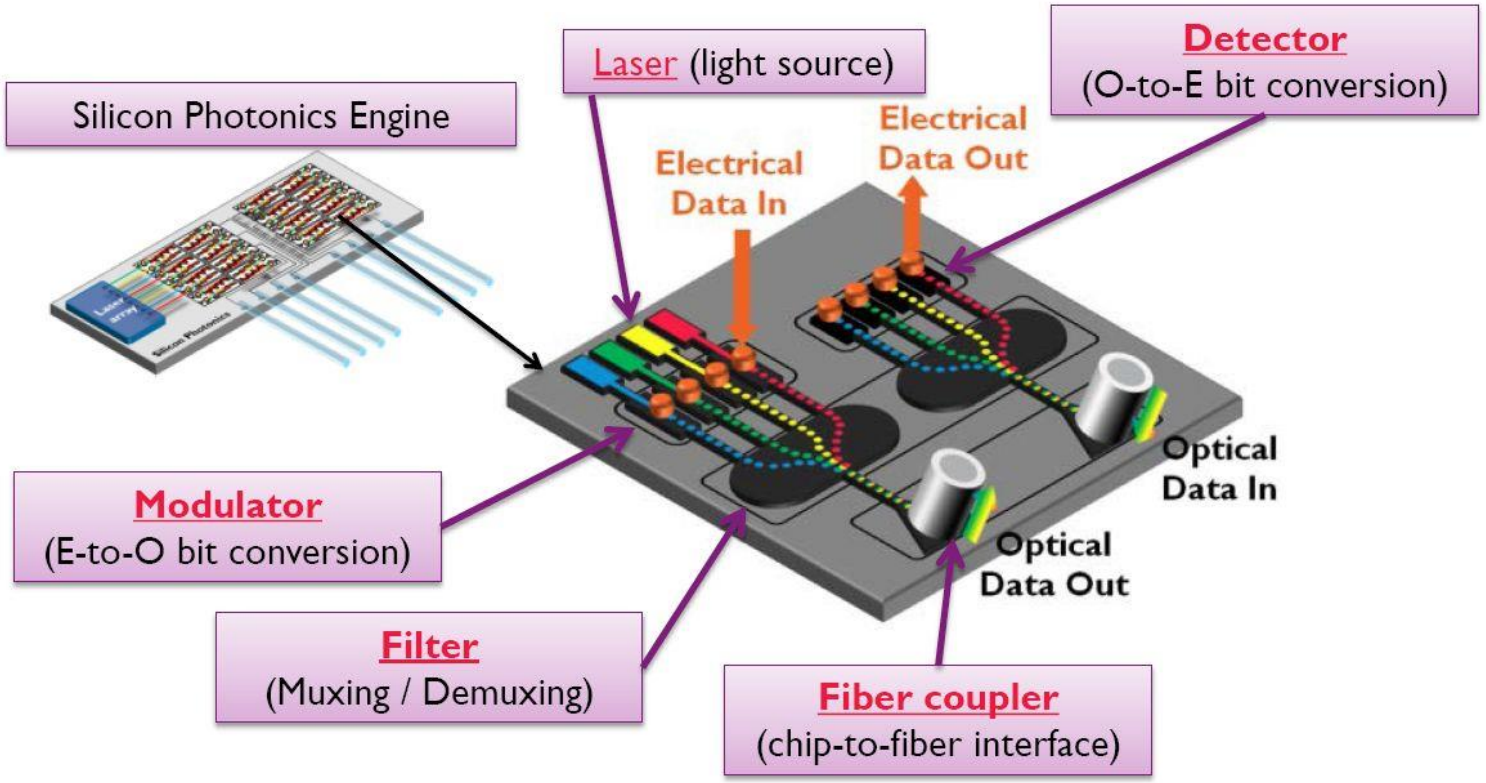
2nm (2024)

Grootte van de Componenten
≈ 10 x precisie
Dus:
3 nm -> 30 nm
≈ 60 atomen breed

Wèl toekomst voor licht in de computer

Photonic Integrated Circuit: geen lichtprocessor, maar wel licht die in **dezelfde chip** interageert met de electronen.
Kan wel niet zo klein gemaakt worden: de processor blijft werken met elektronen, de communicatie via licht





Interuniversitair Micro-Elektronica Centrum is het grootste onafhankelijke Europese onderzoekscentrum op het gebied van micro-elektronica, nanotechnologie

Laatste machine van Fablab kostte 800 miljoen euro

TER INFO: HALFGELEIDERS

Halfgeleiders

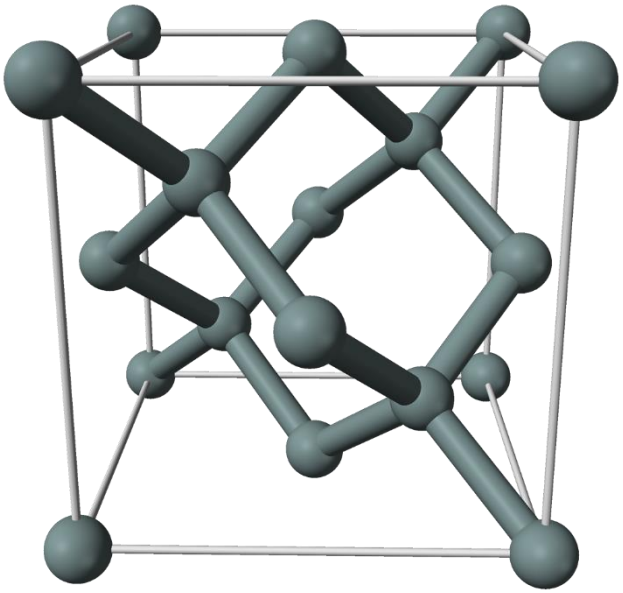
- Gedragen zich als *geleider* of *isolator*
  - Afhankelijk van spanning over component
    - **Diode**
  - Afhankelijk van een andere spanning
    - **Transistor**
    - Werkt als versterker!

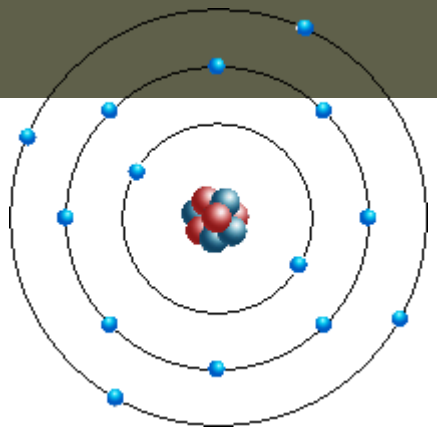
Halfgeleiders

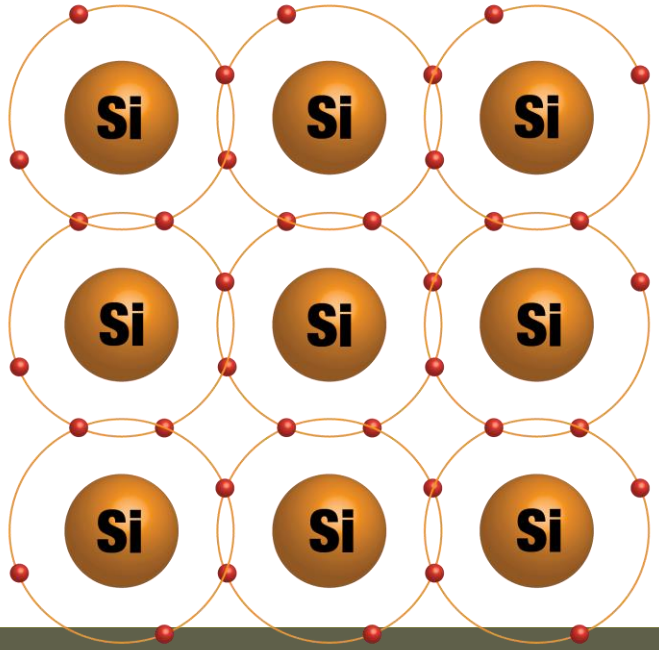
Periodic Table of Semiconductors

Elements Group 13	Elements Group 14	Elements Group 15
3-Electrons in Outer Shell (Positively Charged)	4-Electrons in Outer Shell (Neutrally Charged)	5-Electrons in Outer Shell (Negatively Charged)
(5) Boron (B)	(6) Carbon (C)	
(13) Aluminium (Al)	(14) Silicon (Si)	(15) Phosphorus (P)
(31) Gallium (Ga)	(32) Germanium (Ge)	(33) Arsenic (As)
		(51) Antimony (Sb)

Silicium atoom en kristal





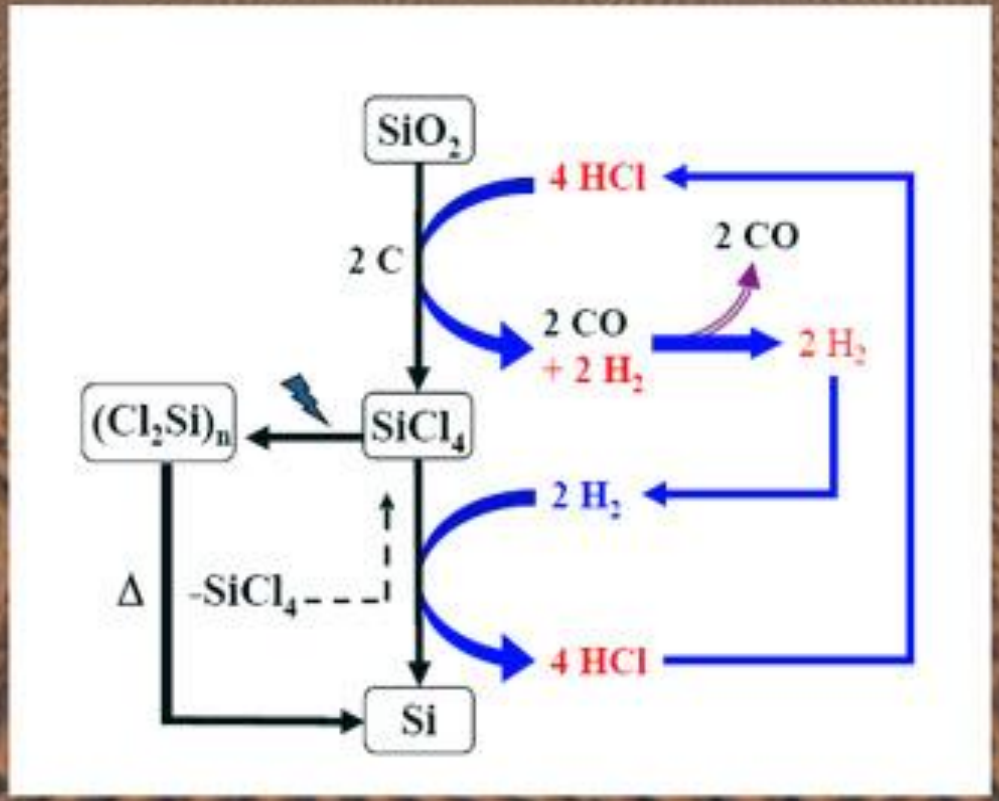


- ‘Voelt’ 8 electronen rond zich

=> *Neutraal (‘OK’)*

SiO_2

Si

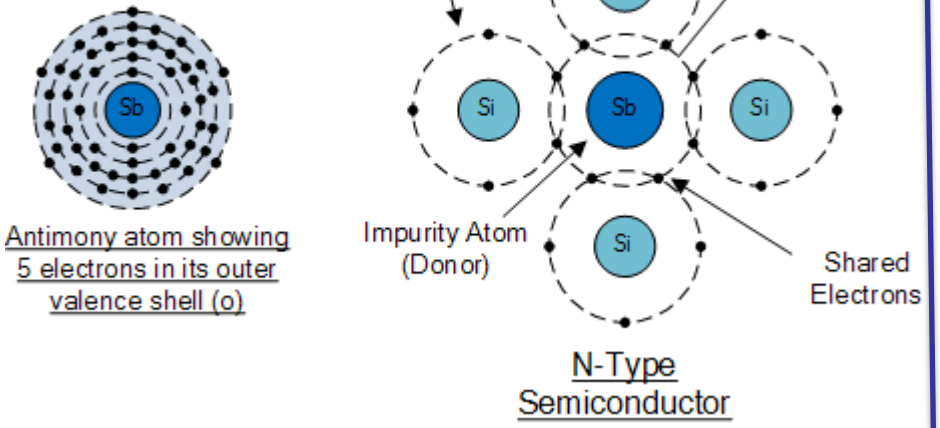


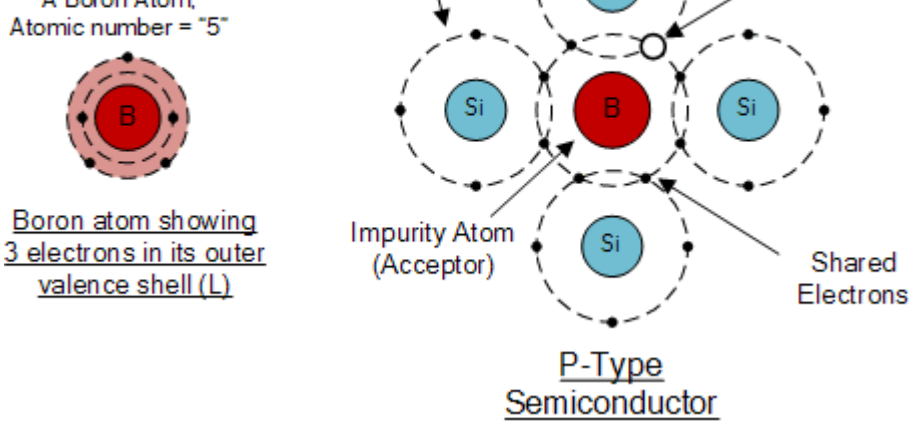
Elektronen kunnen het atoom ontspringen

http://www.pveducation.org/pvcdrom/band-gap

Een gat (hole) wordt opgevuld door een naburig electron -> het lijkt net alsof het gat zich verplaatst

Doperen met neven-atomen





Antimoon

- 'Voelt' 9 electronen rond zich

=> *Eentje te veel om OK te zijn…*

N

Boron

- 'Voelt' 7 electronen rond zich

=> *Eentje te weinig…*

P

Gedopeerd silicium geeft meer electronen of holen

https://www.pveducation.org/pvcdrom/pn-junctions/doping

Diffusie

https://www.pveducation.org/pvcdrom/pn-junctions/diffusion

- Elektronen en holen bewegen zich willekeurig rond
- Hoe hoger de temperatuur, hoe meer en hoe sneller

Een elektrisch veld stuurt de elektronen/holen in een bepaalde richting: drift

https://www.pveducation.org/pvcdrom/pn-junctions/drift

- Door de diffusiecomponent zal het elektron niet 100% het elektrisch veld volgen

Een NP-junctie

https://www.pveducation.org/pvcdrom/pn-junctions/formation-of-a-pn-junction

- Het tegen elkaar plakken van N-gedopeerd silicium met P-gedopeerd silicium zorgt voor een depletielaag of ontruimingslaag waarover een spanning van 0.6 Volt komt te staan.
- De spanning houdt elektronen en holen tegen om van de ene naar de andere kant te springen.

Spanning over een diode

https://www.pveducation.org/pvcdrom/pn-junctions/bias-of-pn-junctions

- Een spanning over de diode zal ofwel de depletielaag versterken en de diode doen **sperren** ofwel zorgen dat de spanning over de depletielaag overwonnen kan worden, de diode is in **doorlaat**.