Individuelle Softwarelösungen

# *Die neue Kollegin ist da*

## Der Weg der Software-Entwicklung bis zur KI

IKS-Thementag 2024

Jan Radeck, Jörg Vollmer

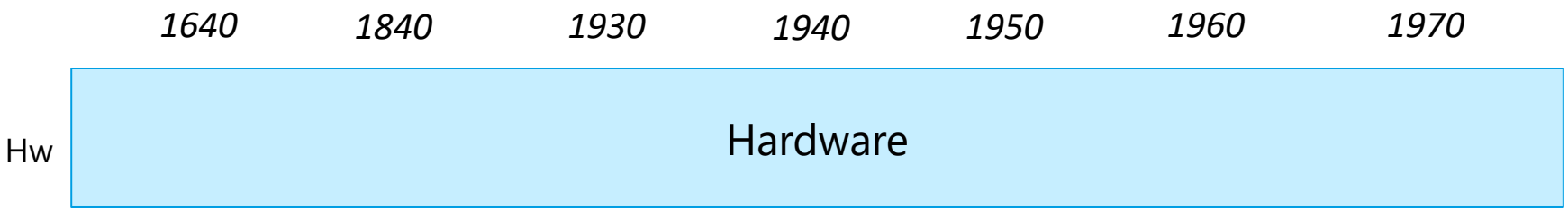
19.06.2024

# Agenda

IKS
Individuelle Softwarelösungen

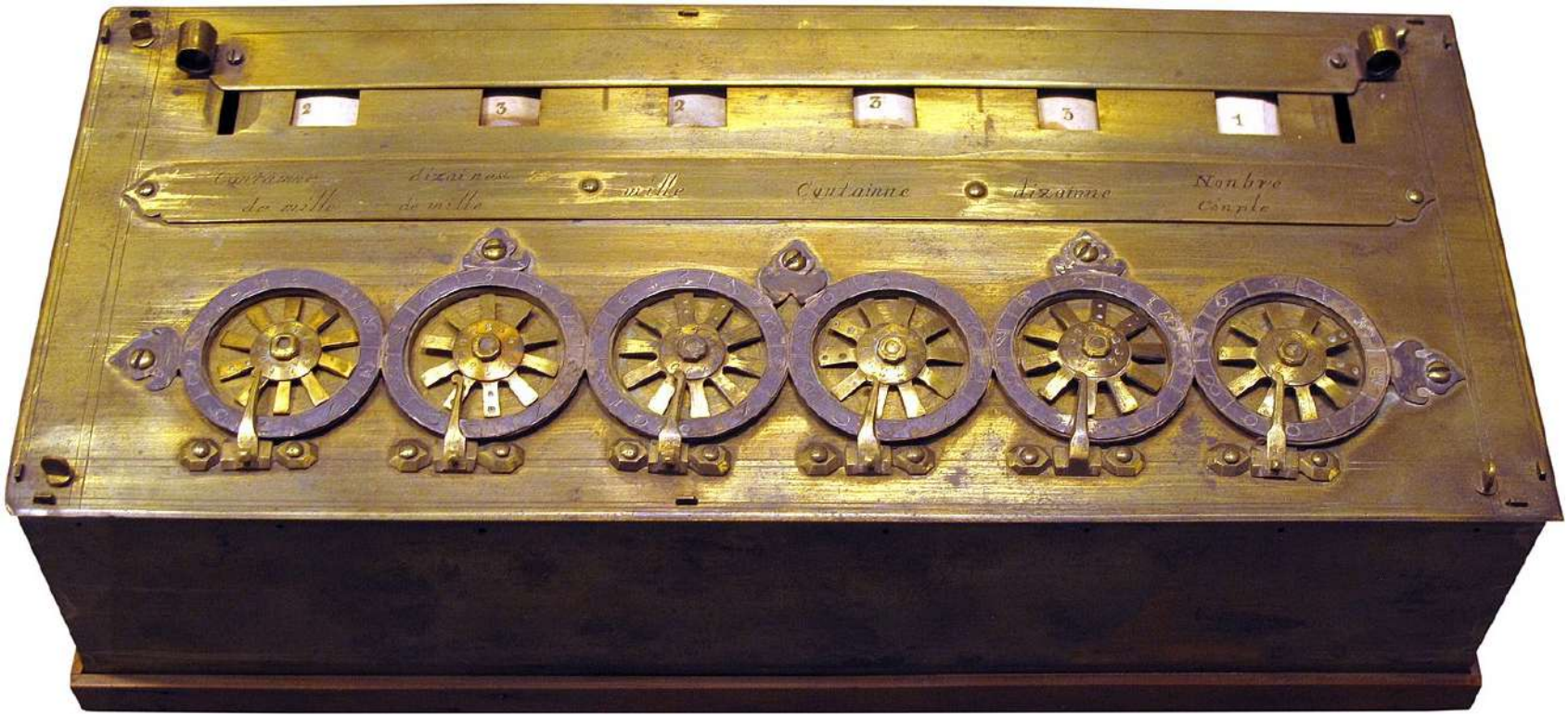
- Von der Pascaline zur denkenden Maschine
  - Hardware
  - Software
  - Entwicklungsprozess
  - KI
- Die neue Kollegin ist da, und jetzt?

# *Timeline*



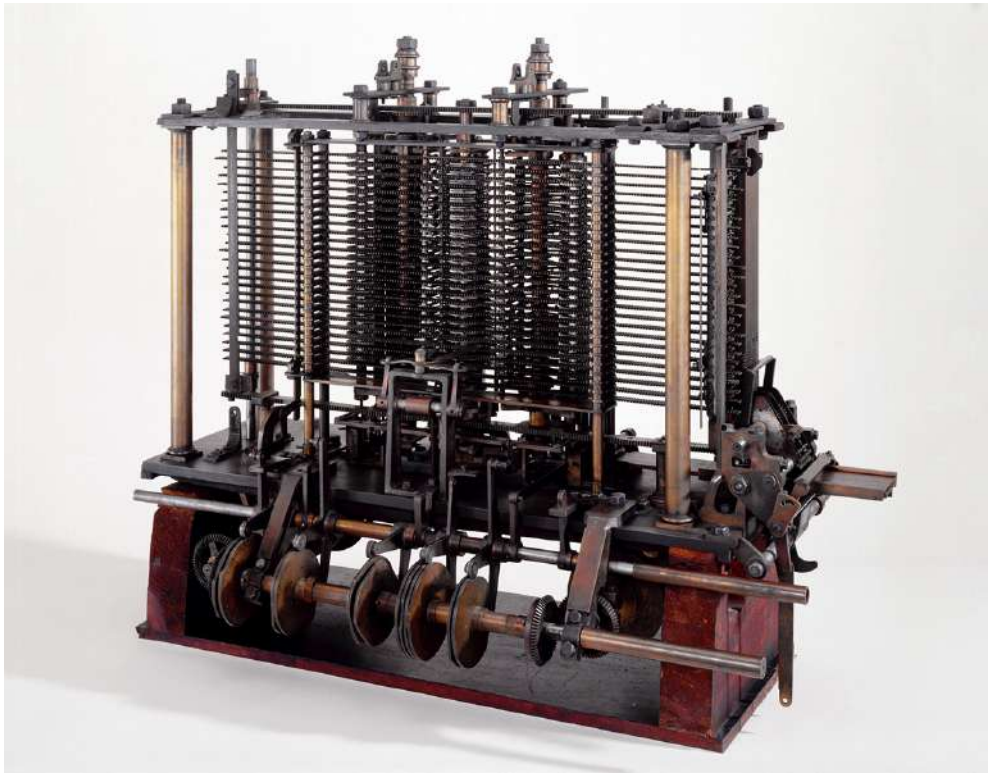
# 1642: Die Pascaline, Blaise Pascal

IKS
Individuelle Softwarelösungen



# *1837: Analytical Engine, Charles Babbage*

IKS
Individuelle Softwarelösungen



# *1843: Die ersten Computer-Programme, Ada Lovelace*

IKS
Individuelle Softwarelösungen

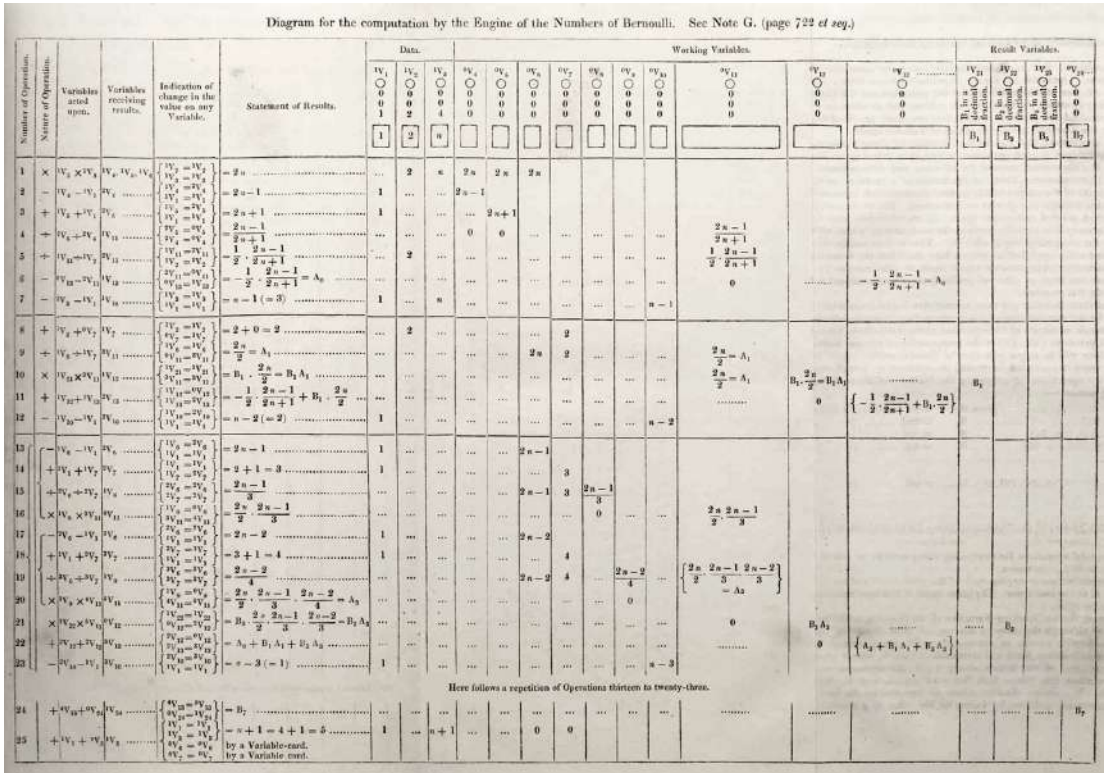
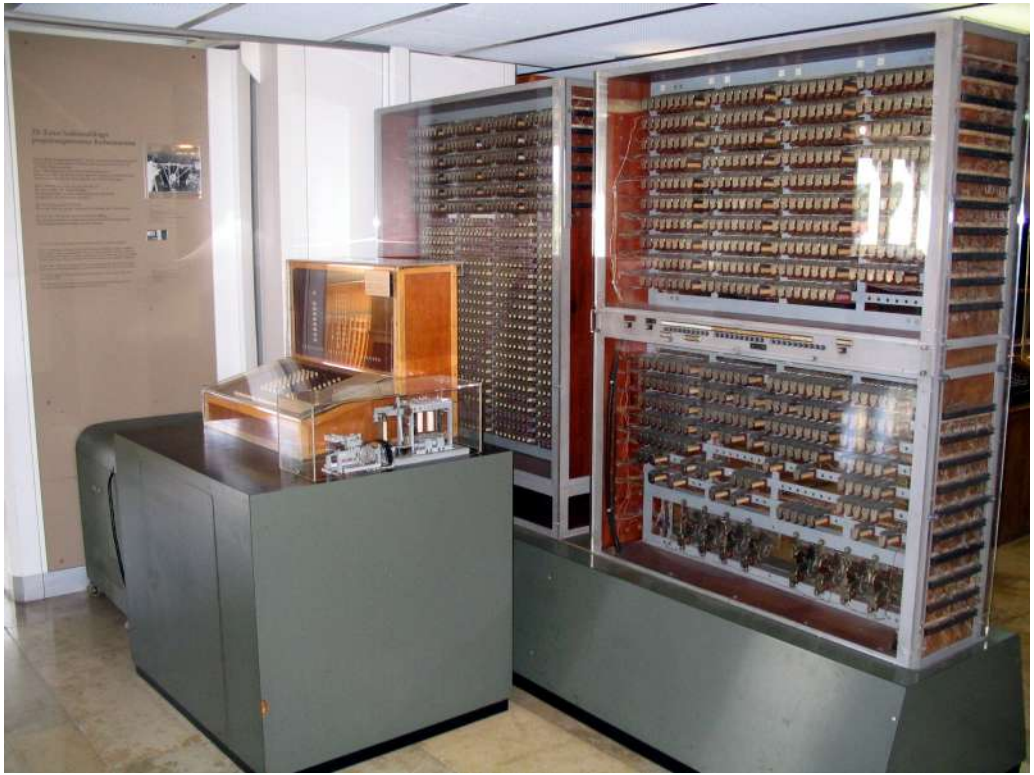


Diagram for the computation by the Engine of the Numbers of Bernoulli. See Note G. (page 722 *et seq.*)

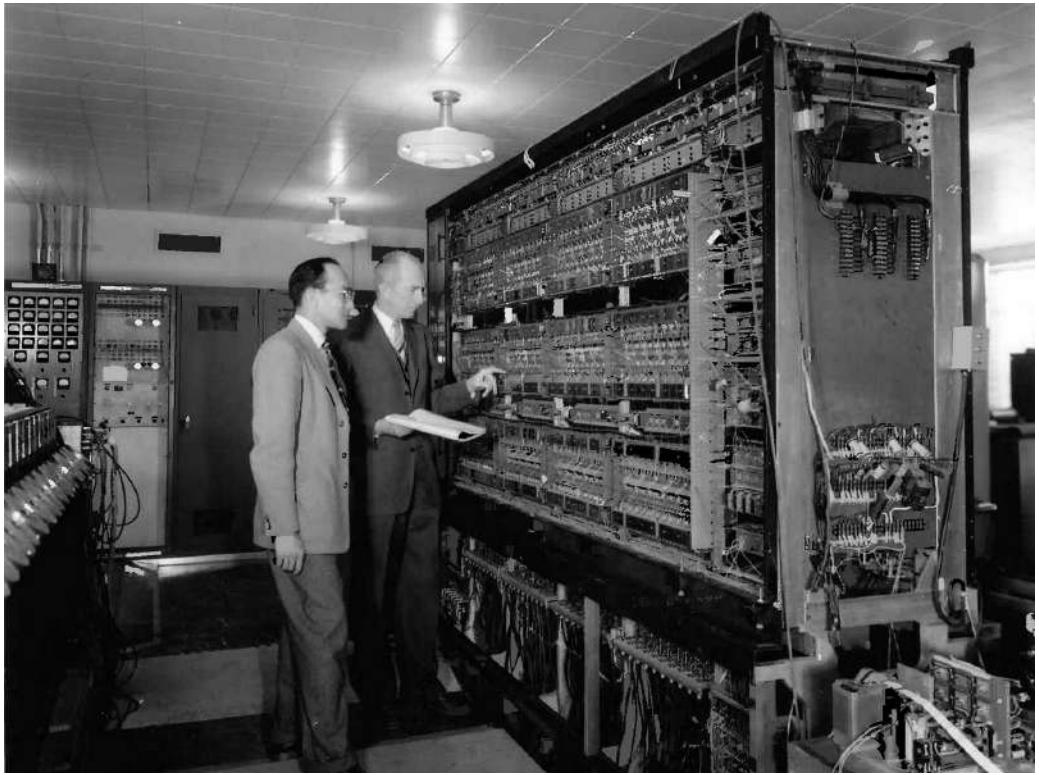
Here follows a repetition of Operations thirteen to twenty-three.

# 1930er: Das Relais, verwendet in der Z3



# 1940er: Die Elektronenröhre, ENIAC, Universität von Pennsylvania

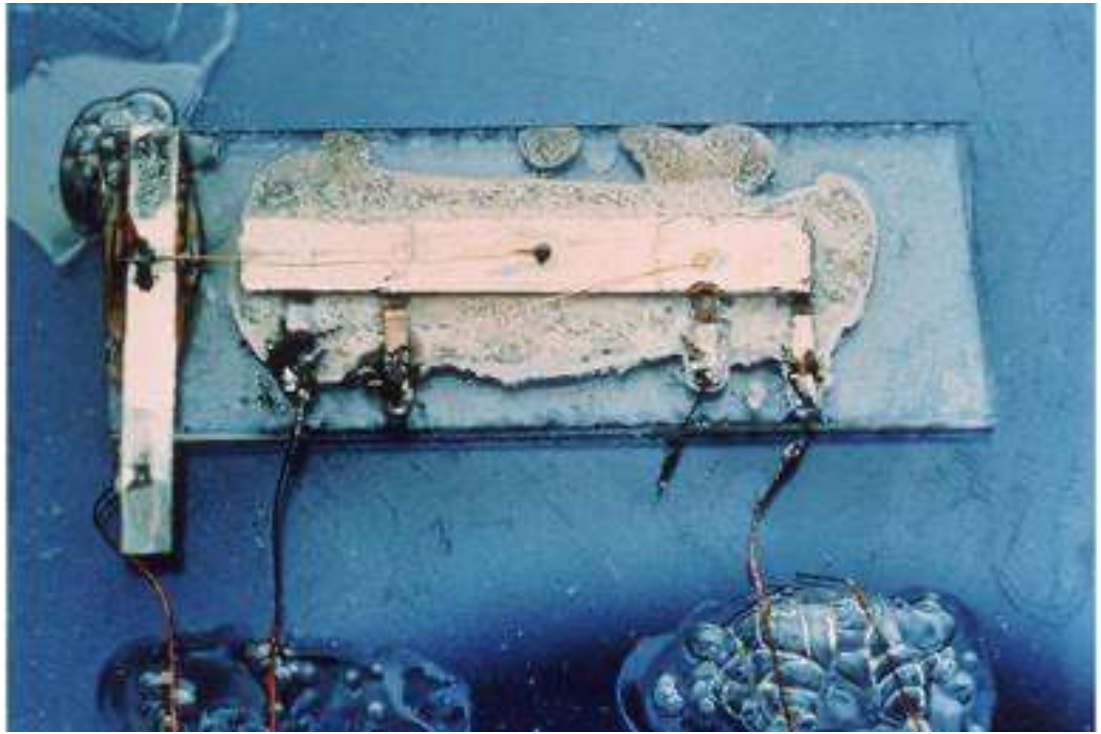
IKS
Individuelle Softwarelösungen

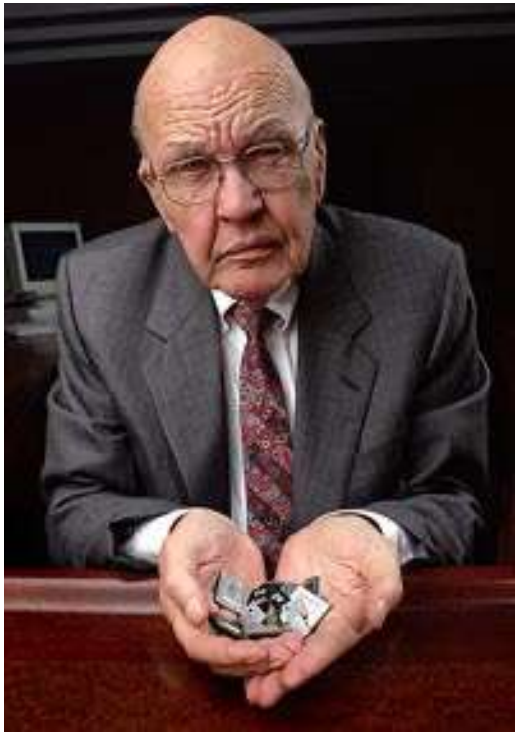


Die neue Kollegin ist da

# 1950er: Der erste Transistor-Computer: TX-0, MIT

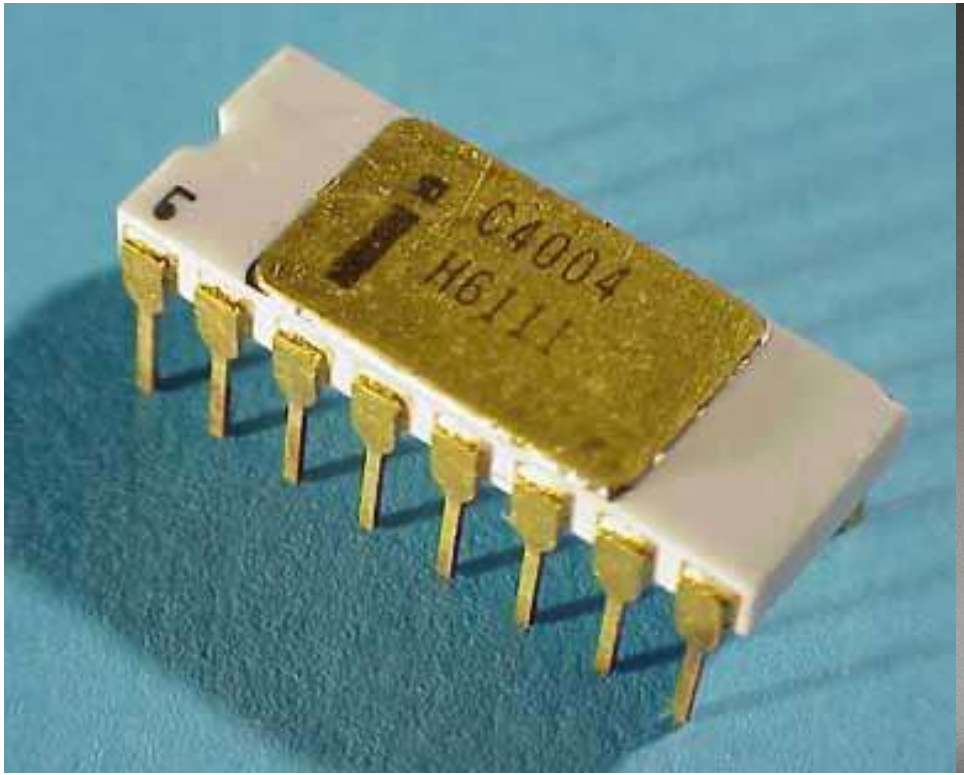

# 1960er: Der erste IC, Texas Instruments





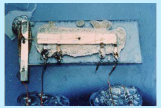
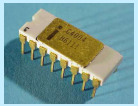
Jack Kilby (Nobelpreis 2000)

# 1970er: Der erste Mikroprozessor, Intel 4004




# Timeline

IKS
Individuelle Softwarelösungen

| | 1640 | 1840 | 1930 | 1940 | 1950 | 1960 | 1970 |
|---|---|---|---|---|---|---|---|
| Hw | Pascaline  | Analytical Engine  | Zuse: Z3  | ENIAC  | TX-0  | IC  | Microprozessor  |
| Sw | Software | | | | | | |

# 1950er: Maschinen-Code

IKS
Individuelle Softwarelösungen

```
11100000 00000010                   E0 02
11110000 00000010                   F0 02
11100000 00000001                   E0 01
11110000 00000010                   F0 02
10100010 00000010                   A2 02
11001010                            CA
11110000 00000010                   F0 02
11101100 00010000 00000000          EC 10 00
11110000 00000010                   F0 02
11100000 00000001                   E0 01
11110000 00000010                   F0 02
10010000 11110000                   90 F0
00111000                            38
01100000                            60
00011000                            18
01100000                            60
```

# 1960er: Assembler

IKS
Individuelle Softwarelösungen

```
start:
      LDA number
      JSR is_prime
      BCS not_prime
is_prime:
      CPX #2
      BEQ prime_done
      CPX #1
      BEQ prime_done
      LDX #2
check_divisible:
      DEX
      BEQ prime_done
      CPX number
      BEQ prime_done
      CPX #1
      BEQ prime_done
      BCC check_divisible
      SEC
      RTS
prime_done:
      CLC
      RTS
```

# 1960er: Assembler

IKS
Individuelle Softwarelösungen

```
start:
      LDA number            ; Lade die zu testende Zahl in das Akkumulator-Register
      JSR is_prime          ; Rufe die Unterfunktion auf, um zu prüfen, ob die Zahl eine Primzahl ist
      BCS not_prime         ; Wenn das Carry-Bit gesetzt ist, ist die Zahl keine Primzahl
is_prime:
      CPX #2                ; Vergleiche die Zahl im Akkumulator-Register mit 2
      BEQ prime_done        ; Wenn die Zahl gleich 2 ist, ist sie eine Primzahl
      CPX #1                ; Vergleiche die Zahl im Akkumulator-Register mit 1
      BEQ prime_done        ; Wenn die Zahl gleich 1 ist, ist sie keine Primzahl
      LDX #2                ; Setze X auf 2, um die Teilbarkeit durch alle Zahlen von 2 bis zur Hälfte der zu testenden Zahl zu überprüfen
check_divisible:
      DEX                   ; Dekrementiere X, um die nächste Zahl zu überprüfen
      BEQ prime_done        ; Wenn X gleich 0 ist, wurde keine Teilbarkeit gefunden, die Zahl ist eine Primzahl
      CPX number            ; Vergleiche die Zahl im Akkumulator-Register mit der aktuellen X-Zahl
      BEQ prime_done        ; Wenn die Zahl gleich der aktuellen X-Zahl ist, ist sie eine Primzahl (nicht teilbar)
      CPX #1                ; Vergleiche die Zahl im Akkumulator-Register mit 1
      BEQ prime_done        ; Wenn die Zahl gleich 1 ist, ist sie keine Primzahl
      BCC check_divisible   ; Wenn die Zahl nicht teilbar ist, setze die Schleife fort
      SEC                   ; Setze das Carry-Bit, um anzuzeigen, dass die Zahl nicht prim ist
      RTS                   ; Rückkehr zur aufrufenden Stelle
prime_done:
      CLC                   ; Lösche das Carry-Bit, um anzuzeigen, dass die Zahl prim ist
      RTS                   ; Rückkehr zur aufrufenden Stelle
```

# 1960er: Fortran I

```
     READ(5,2) NUM
2    FORMAT(I)
     I = 2
3    IF(I**2-NUM) 4,5,5
4    PRINT 6
     STOP
5    IF(I-1) 6,7,7
6    I = I + 1
     GOTO 3
7    PRINT 8
     STOP
8    FORMAT(1H1, ' ', I5)
     END
```

# 1960er: Cobol

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 Num PIC 9(5).
01 IsPrime PIC X.

PROCEDURE DIVISION.
    Accept-Num.
    Perform Prime-Number-Test.
    STOP RUN.

Accept-Num.
    ACCEPT Num.
```

```
Prime-Number-Test.
    IF Num <= 1
        MOVE "N" TO IsPrime
    ELSE
        PERFORM Check-Divisors UNTIL Num
<= Sqrt(Num)
        IF IsPrime IS INITIAL
            MOVE "Y" TO IsPrime
        END-IF
    END-IF.

Check-Divisors.
    DIVIDE Num BY Num - 1 GIVING
Remainder
    IF Remainder = 0
        MOVE "N" TO IsPrime
    END-IF
    SUBTRACT 1 FROM Num.
```

# 1960er: Algol 60

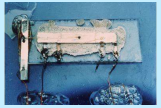
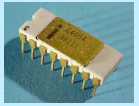
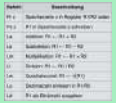
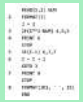
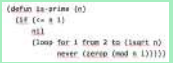
IKS
Individuelle Softwarelösungen

```
PROCEDURE is_prime_number(n);
    INTEGER n, i;
    is_prime := TRUE;
    FOR i := 2 STEP 1 UNTIL sqrt(n) DO
        IF n MOD i = 0 THEN
            is_prime := FALSE;
            EXIT;
        FI;
    OD;
END;
```

# 1960er: Lisp

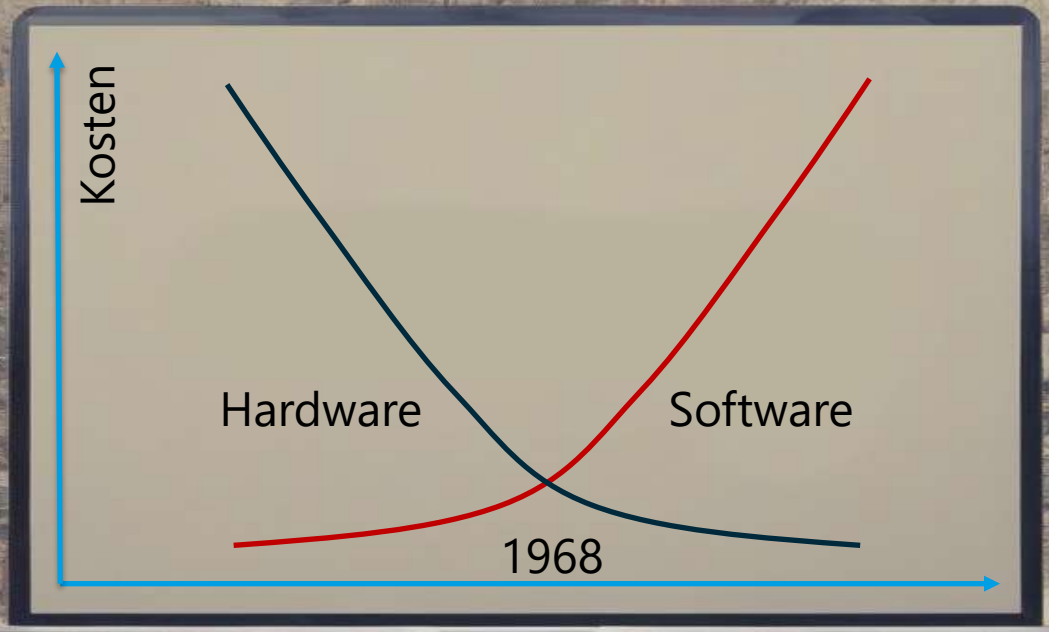
```
(defun is-prime (n)
  (if (<= n 1)
      nil
      (loop for i from 2 to (isqrt n)
            never (zerop (mod n i)))))
```

# Timeline

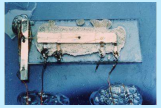
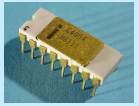
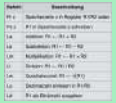
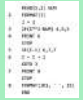
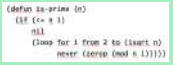
| | 1640 | 1840 | 1930 | 1940 | 1950 | 1960 | 1970 |
|---|---|---|---|---|---|---|---|
| Hw | Pascaline  | Analytical Engine  | Zuse: Z3  | ENIAC  | TX-0  | IC  | Mikroprozessor  |
| Sw | | Ada Lovelace  | Arithm. Befehle  | Assembler | | Fortran  Algol  | Lisp  |
| EP | Entwicklungsprozess | | | | | Strukturierte Program-mierung | |


1960er: Die Gewinner sind
Cobol
Fortran
Assembler
schade
1
2
3

# 1970er: 1. Software-Krise



# Timeline

| | 1640 | 1840 | 1930 | 1940 | 1950 | 1960 | 1970 |
|---|---|---|---|---|---|---|---|
| Hw | Pascaline  | Analytical Engine  | Zuse: Z3  | ENIAC  | TX-0  | IC  | Mikroprozessor  |
| Sw | | Ada Lovelace  | Arithm. Befehle  | Assembler | | Fortran  Algol  | Lisp  |
| EP | | | | | | Strukturierte Program-mierung | |
| KI | Künstliche Intelligenz | | | | | | |

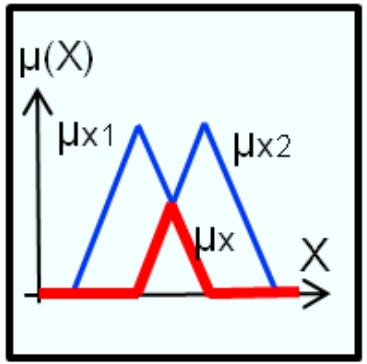
# *Thema KI: Warum ist das bloß so schwierig für Computer?*

IKS
Individuelle Softwarelösungen

- Computer können **gut „exakt“**
  - rechnen: billionenfach schneller als Menschen
  - Extrem schnell miteinander kommunizieren: milliardenfach schneller
  - immens viele Daten exakt speichern, und exakt vergleichen
- Computer konnten **schlecht “ähnlich“**
  - sehen und Dinge dabei erkennen
  - hören und dabei Dinge erkennen
  - ähnliche Dinge ausmachen und klassifizieren
- Menschliche Intelligenz zeichnet sich dadurch aus, eine **Hierarchie** von Konzepten zu verstehen

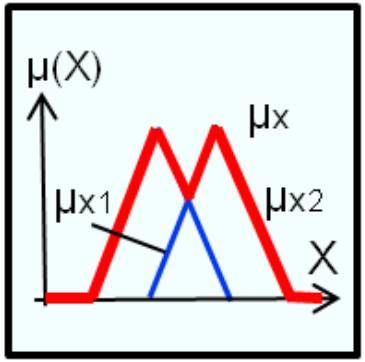
# 1960er: Fuzzy-Logic (Lotfi A. Zadeh)

**UND-Verknüpfung**
**min-Operator**



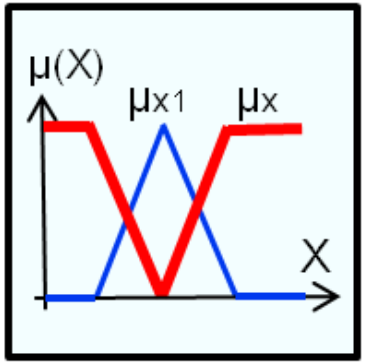

$\mu_x(X) = \min [\mu_{x1}(X), \mu_{x2}(X)]$
$= (\mu_{x1} \cap \mu_{x2})(X)$

**ODER-Verknüpfung**
**max-Operator**



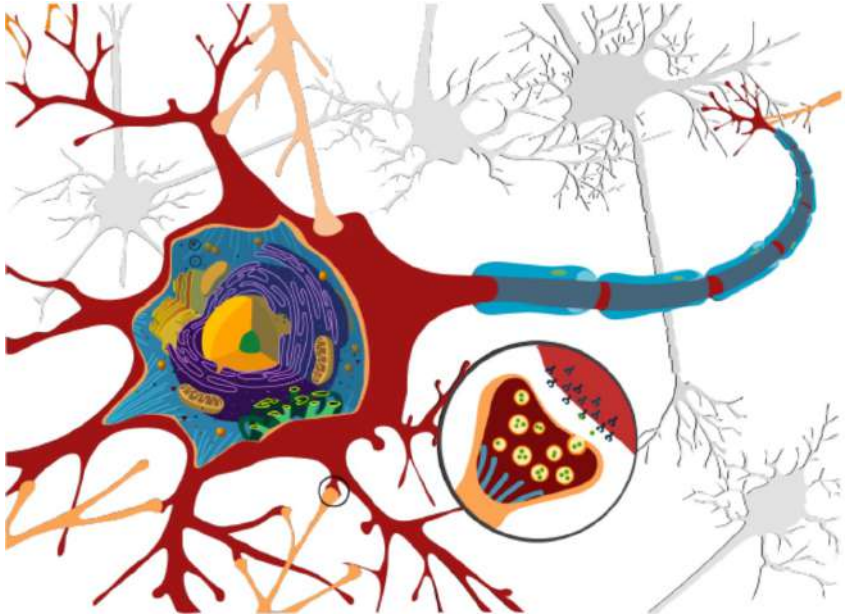

$\mu_x(X) = \max [\mu_{x1}(X), \mu_{x2}(X)]$
$= (\mu_{x1} \cup \mu_{x2})(X)$

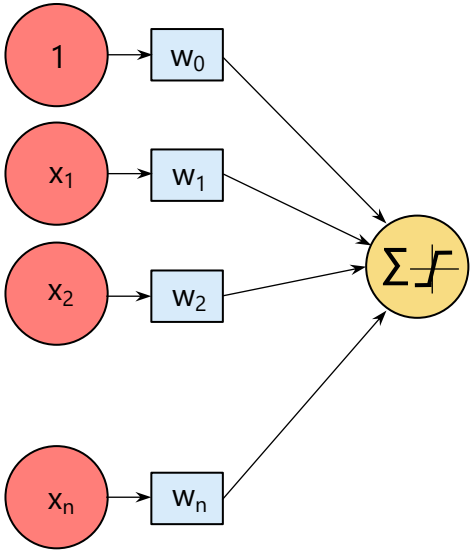
**NICHT-Funktion**
**Komplement-Operator**




$\mu_x(X) = 1 - \mu_{x1}(X)$

# 1960er: Das Perzeptron (Frank Rosenblatt)






Hebb'sche Lernregel

# *1970er: Der 1. KI-Winter*

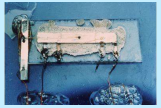
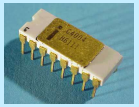
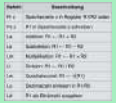
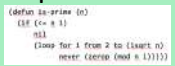
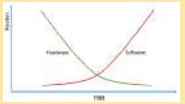
**Verarbeitung komplexerer neuronaler Netze?**

**Aus Hardware-Sicht unmöglich!**

neuroflash

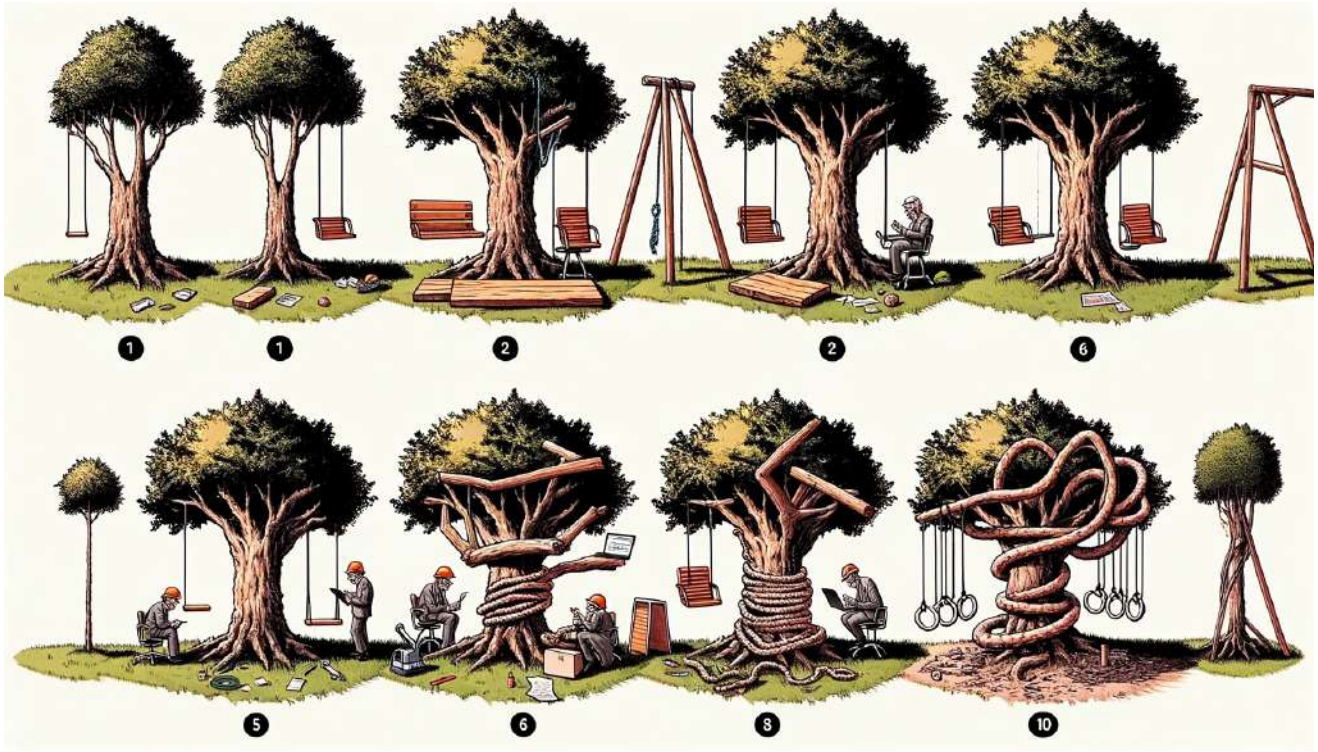
# Timeline

IKS
Individuelle Softwarelösungen

| | 1640 | 1840 | 1930 | 1940 | 1950 | 1960 | 1970 |
|---|---|---|---|---|---|---|---|
| Hw | Pascaline  | Analytical Engine  | Zuse: Z3  | ENIAC  | TX-0  | IC  | Mikroprozessor  |
| Sw | | Ada Lovelace  | Arithm. Befehle  | Assembler | | Fortran  Algol  | Lisp  |
| EP | | | | | | | Strukturierte Program-mierung 1. Softwarekrise  |
| KI | | | | | | | Perzeptron  Fuzzylogik  1. KI-Winter  |

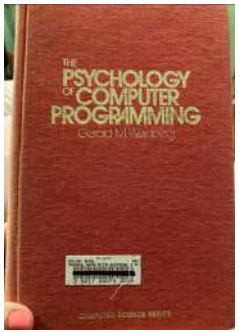
# 1970er: Warum ist Software nur so teuer?

IKS
Individuelle Softwarelösungen

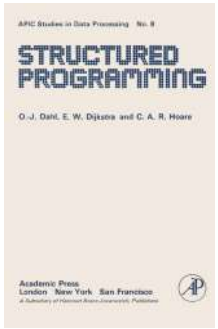


# 1970er: Software-Entwicklung wie ein Ingenieur

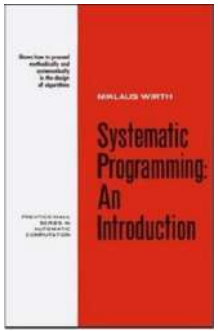

1970



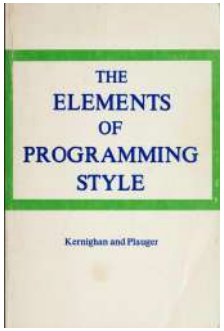

1971



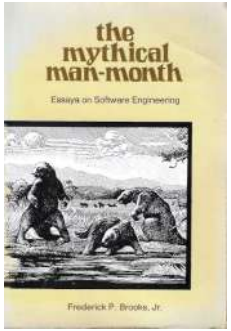

1972



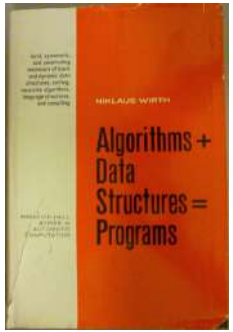

1973



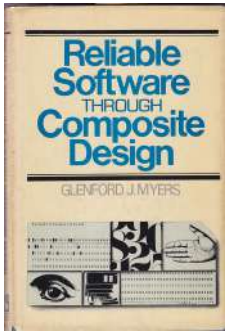

1974



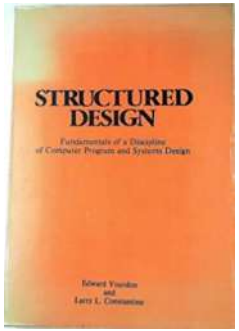

1975



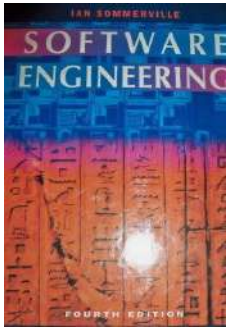

1976




1975




1978



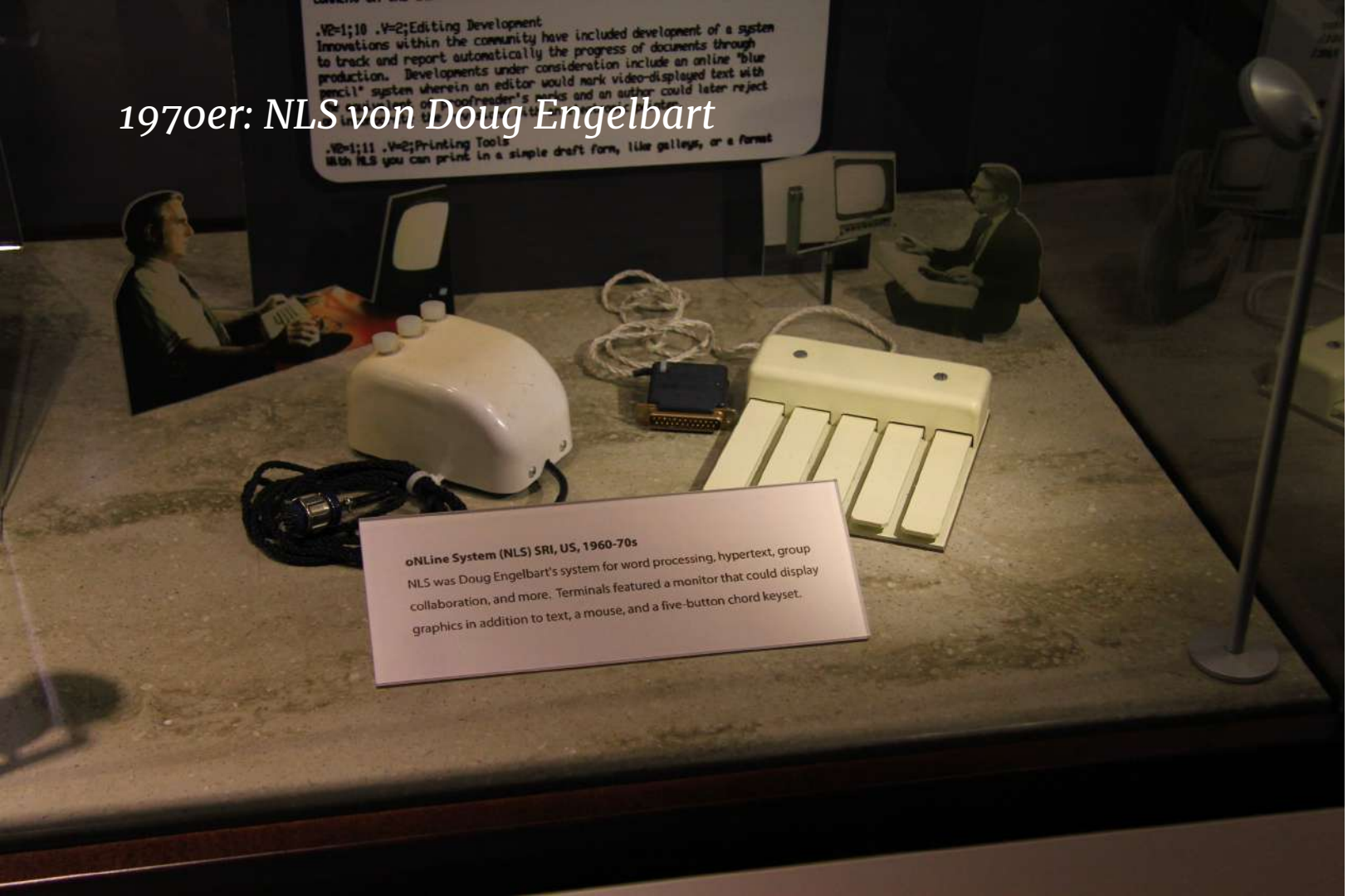

1982

# 1970er: Alternative KI-Forschung

IKS
Individuelle Softwarelösungen

- **Modellierung der Welt**
- **Expertensysteme**: Komplexe Entscheidungen
  - Regelbasierte Systeme: Prolog(!)
  - Fuzzylogik
  - Bayesianische Netze
  - Wissensrepräsentation: Frames, Schemata
  - Ontologien
- **Genetische Algorithmen**: Komplexe Optimierungen
- **Interaktion zwischen Menschen und Computern**

# *1970er: NLS von Doug Engelbart*



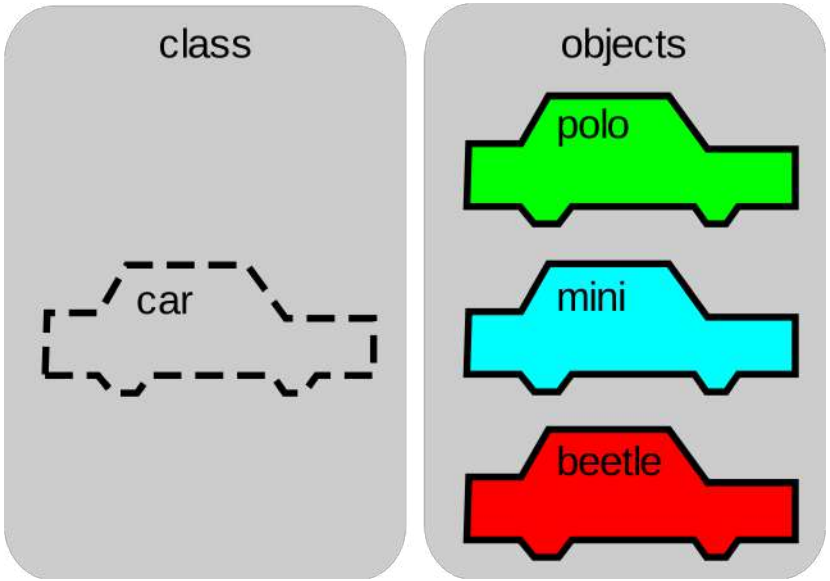

IKS
Individuelle Softwarelösungen


Ablage Bearbeiten Inhalt Spezial
Der Macintosh® Finder™
Larry, John, Steve und Bruce
©Apple Computer, Inc.
Macintosh Classic

# 1970er: Objektorientierung (auch gut für GUIs)

IKS Individuelle Softwarelösungen

**Modellierung der Welt**




1. OOP
2. OOD
3. OOA
4. OOM
5. OOSE
6. Booch
7. OMT

1. (Modula, C)
2. Simula
3. Smalltalk
4. C++
5. Objective C

# 1970er: Smalltalk (Kay, Ingalls, Goldberg)

IKS
Individuelle Softwarelösungen

```
isPrime: n
    ^(n <= 1)
        ifTrue: [false]
        ifFalse: [((2 to: (n sqrt) ceiling)
            allSatisfy: [:it | n \\ it ~= 0])].
```

# 1970er: Relationale Datenbanken (E. F. Codd), SQL

IKS
Individuelle Softwarelösungen


Schueler

| SNR | Name | Vorname |
|---|---|---|
| 6431 | Euler | Leo |
| 8151 | Gauß | Carl |
| 8152 | Gödel | Kurt |
| 7687 | Klein | Felix |

Kursbelegung

| SNR | CNR |
|---|---|
| 8151 | 13 |
| 6431 | 13 |
| 8151 | 24 |
| 8151 | 31 |

Kurs

| CNR | Name | LK |
|---|---|---|
| 13 | I LK2 | 1 |
| 24 | M LK | 1 |
| 31 | E LK | 1 |
| 26 | M GK | 0 |

```
SELECT Schueler.Name, Kurs.Name FROM Schueler
JOIN Kursbelegung ON Schueler.SNR = Kursbelegung.SNR
JOIN Kurs ON Kursbelegung.CNR = Kurs.CNR
```

*Information Retrieval* P. BAXENDALE, Editor

**A Relational Model of Data for Large Shared Data Banks**

E. F. Codd
*IBM Research Laboratory, San Jose, California*

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

Existing noninferential, formatted data systems provide users with tree-structured files or slightly more general network models of the data. In Section 1, inadequacies of these models are discussed. A model based on n-ary relations, a normal form for data base relations, and the concept of a universal data sublanguage are introduced. In Section 2, certain operations on relations (other than logical inference) are discussed and applied to the problems of redundancy and consistency in the user's model.

KEY WORDS AND PHRASES: data bank, data base, data structure, data organization, hierarchies of data, networks of data, relations, derivability, redundancy, consistency, composition, join, retrieval language, predicate calculus, security, data integrity
CR CATEGORIES: 3.70, 3.73, 3.75, 4.20, 4.22, 4.29

**1. Relational Model and Normal Form**

1.1. Introduction

This paper is concerned with the application of elementary relation theory to systems which provide shared access to large banks of formatted data. Except for a paper by Childs [1], the principal application of relations to data systems has been to deductive question-answering systems. Levein and Maron [2] provide numerous references to work in this area.

In contrast, the problems treated here are those of *data independence*—the independence of application programs and terminal activities from growth in data types and changes in data representation—and certain kinds of *data inconsistency* which are expected to become troublesome even in nondeductive systems.

The relational view (or model) of data described in Section 1 appears to be superior in several respects to the graph or network model [3, 4] presently in vogue for noninferential systems. It provides a means of describing data with its natural structure only—that is, without superimposing any additional structure for machine representation purposes. Accordingly, it provides a basis for a high level data language which will yield maximal independence between programs on the one hand and machine representation and organization of data on the other.

A further advantage of the relational view is that it forms a sound basis for treating derivability, redundancy, and consistency of relations—these are discussed in Section 2. The network model, on the other hand, has spawned a number of confusions, not the least of which is mistaking the derivation of connections for the derivation of relations (see remarks in Section 2 on the "connection trap").

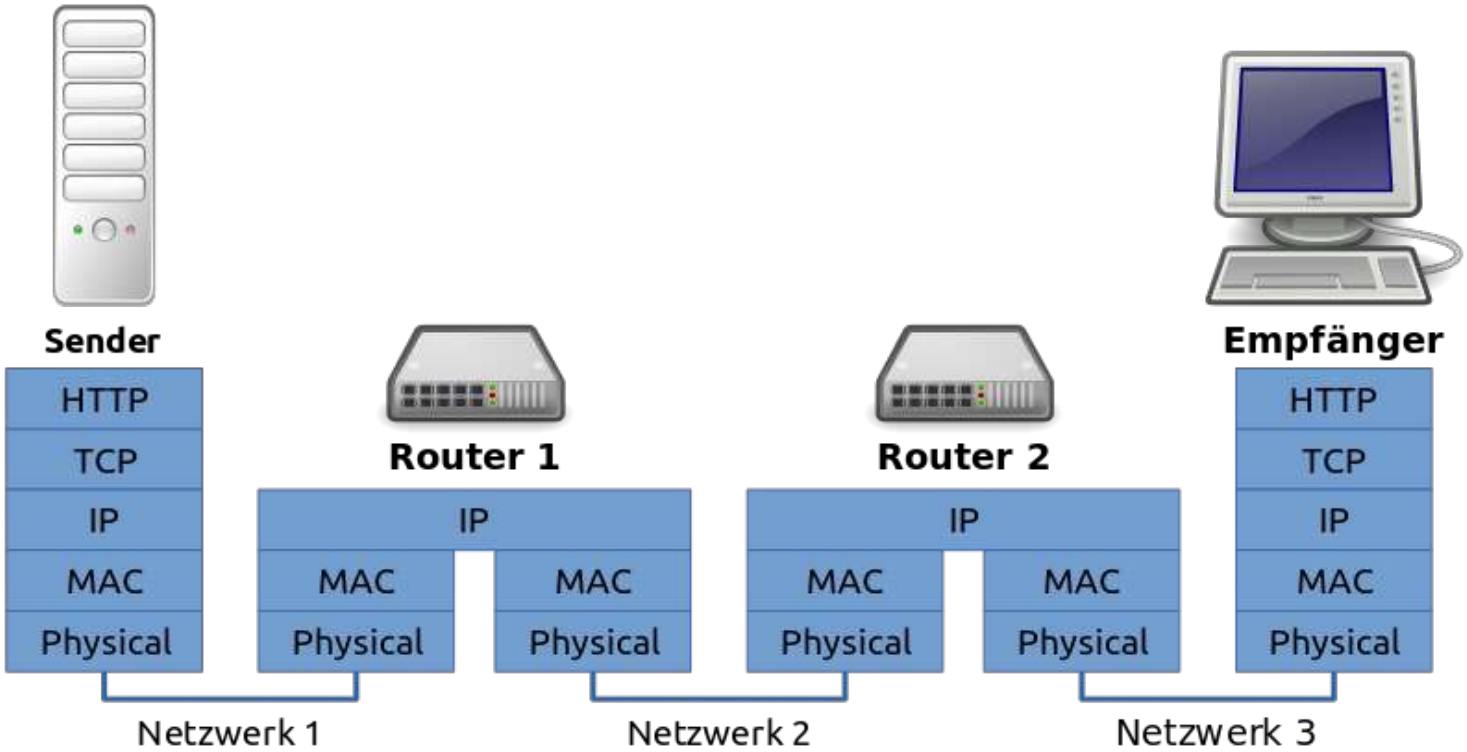
Finally, the relational view permits a clearer evaluation of the scope and logical limitations of present formatted data systems, and also the relative merits (from a logical standpoint) of competing representations of data within a single system. Examples of this clearer perspective are cited in various parts of this paper. Implementations of systems to support the relational model are not discussed.

1.2. Data Dependencies in Present Systems

The provision of data description tables in recently developed information systems represents a major advance toward the goal of data independence [5, 6, 7]. Such tables facilitate changing certain characteristics of the data representation stored in a data bank. However, the variety of data representation characteristics which can be changed *without logically impairing some application programs* is still quite limited. Further, the model of data with which users interact is still cluttered with representational properties, particularly in regard to the representation of collections of data (as opposed to individual items). Three of the principal kinds of data dependencies which still need to be removed are: ordering dependence, indexing dependence, and access path dependence. In some systems these dependencies are not clearly separable from one another.

1.2.1. *Ordering Dependence.* Elements of data in a data bank may be stored in a variety of ways, some involving no concern for ordering, some permitting each element to participate in one ordering only, others permitting each element to participate in several orderings. Let us consider those existing systems which either require or permit data elements to be stored in at least one total ordering which is closely associated with the hardware-determined ordering of addresses. For example, the records of a file concerning parts might be stored in ascending order by part serial number. Such systems normally permit application programs to assume that the order of presentation of records from such a file is identical to (or is a subordering of) the

# 1970er: Netzwerke, TCP/IP (Cerf, Kahn)



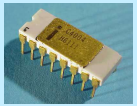
# 1970er: Unix (Thompson, Ritchie)

IKS
Individuelle Softwarelösungen

# Timeline

IKS
Individuelle Softwarelösungen

| 1970 | 1980 | 1990 | 2000 |
|---|---|---|---|

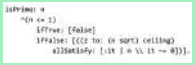
Mikroprozessor 

Netzwerke 

Der PC 

Die Maus 

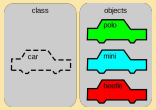
Pascal C SQL Modula2 Smalltalk



Software-Technik 

Unix 

RDBs TCP/IP 

OOP 

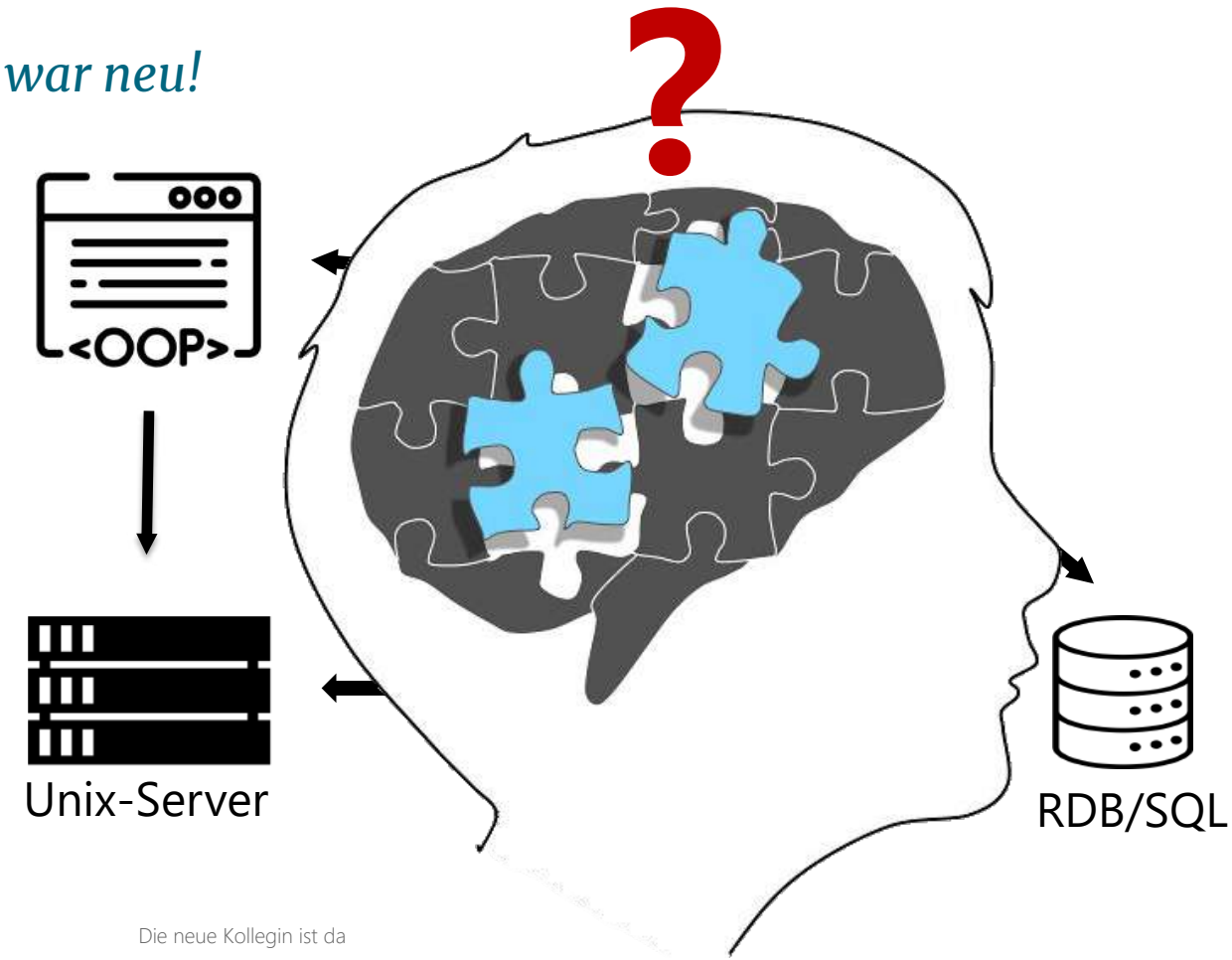

GUIs 

Genetische Algorithmen

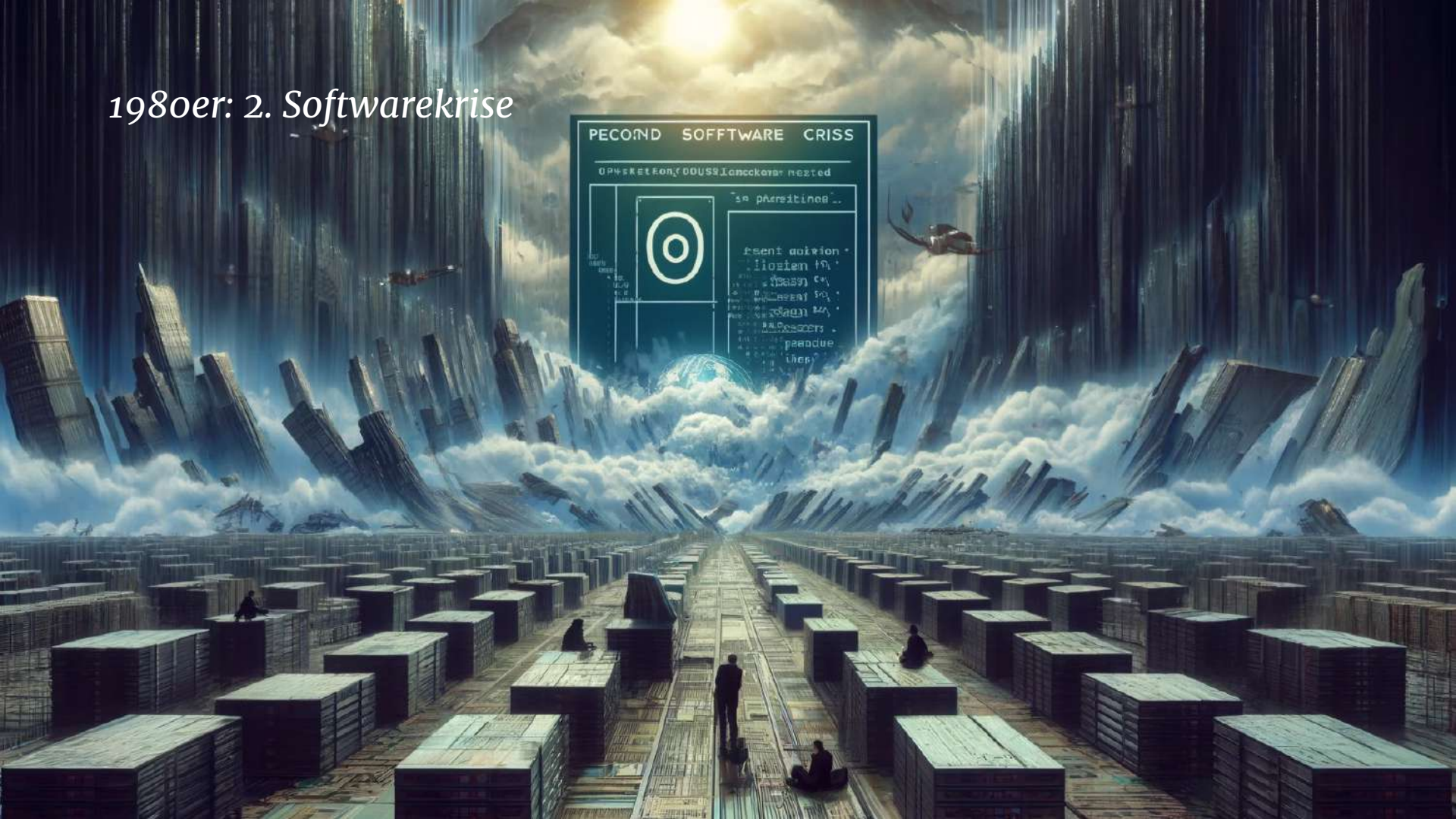
Wissens-repräsen-tation

Experten-systeme

# *Zu viel war neu!*



# 1980er: 2. Softwarekrise



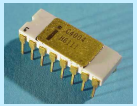
# *1980er: 2. KI-Winter*

Lösung für komplexe Probleme?

Erwartungen wurden erneut enttäuscht!

# Timeline

1970 1980 1990 2000

Mikroprozessor 

Netzwerke 

Der PC 

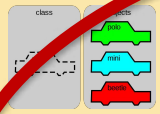
Die Maus 

Pascal C SQL Modula2 Smalltalk C++

Software-Technik 

Unix 

RDBs TCP/IP

OOP 

GUIs 

2. Softwarekrise 

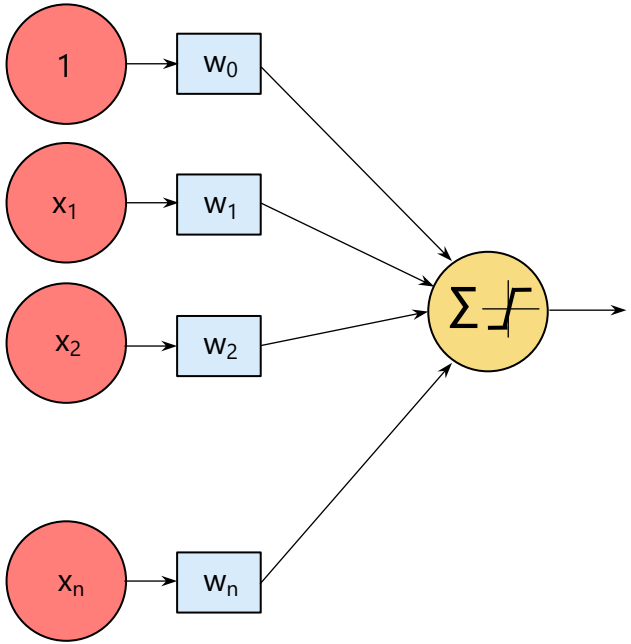
Genetische Algorithmen

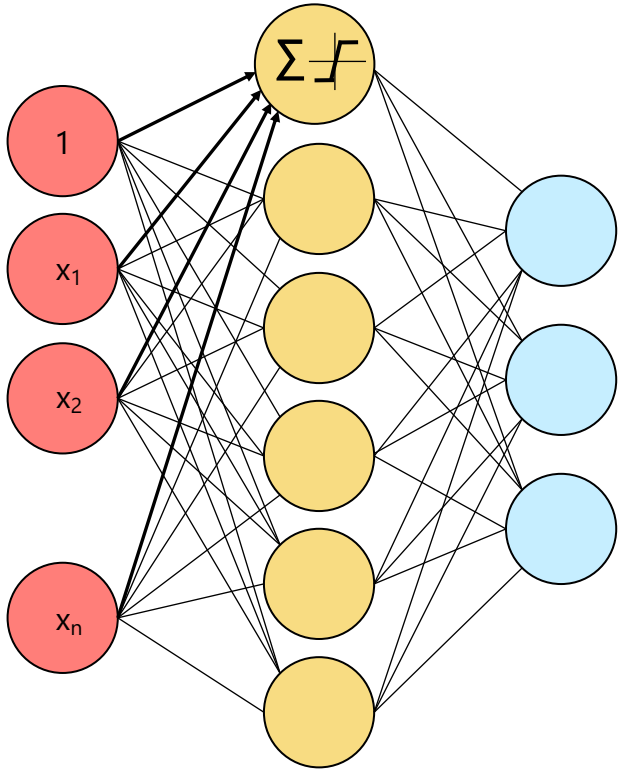
Wissens-repräsen-tation

Experten-systeme

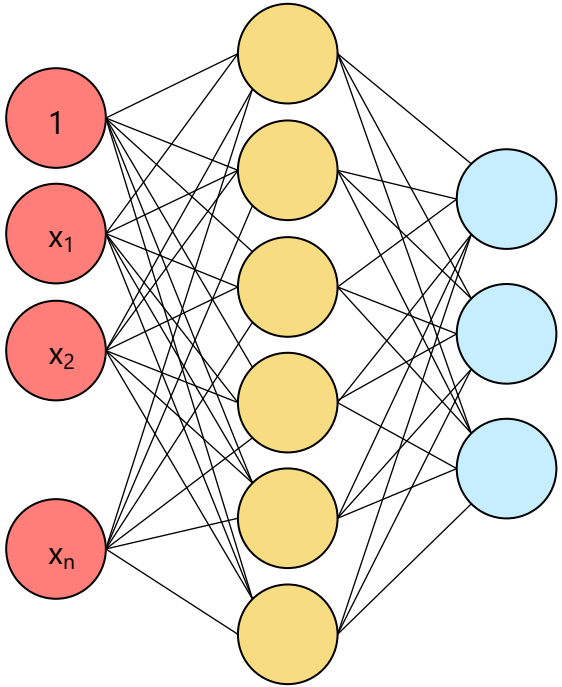
2. KI-Winter 

# 1980er: Backpropagation (Werbos, Rumelhart, Hinton, Williams)

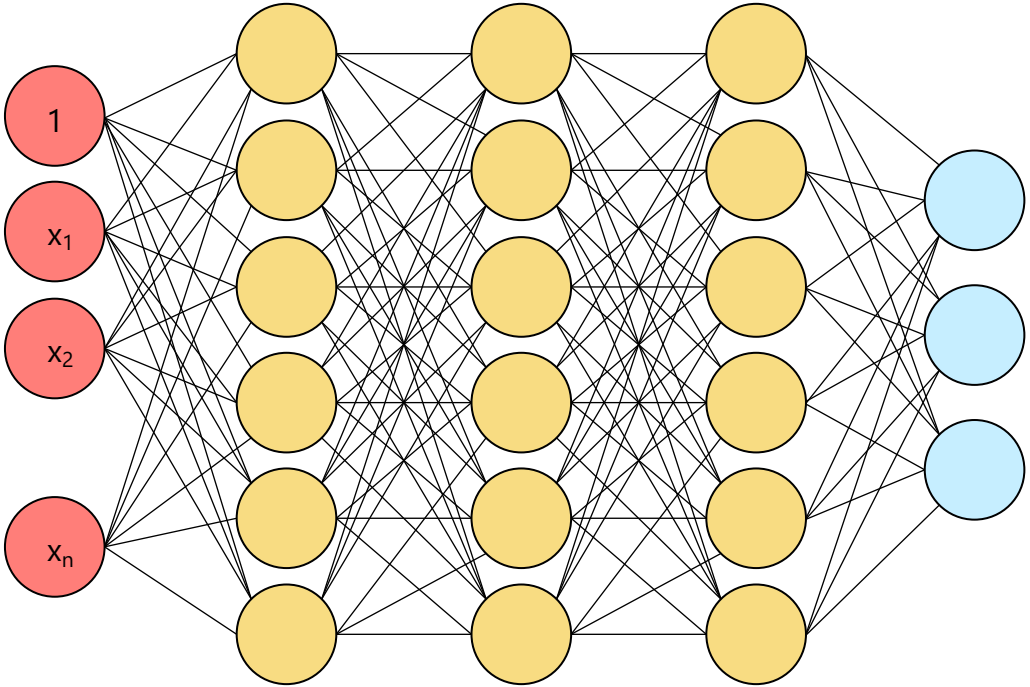


# 1990er: Deep Learning

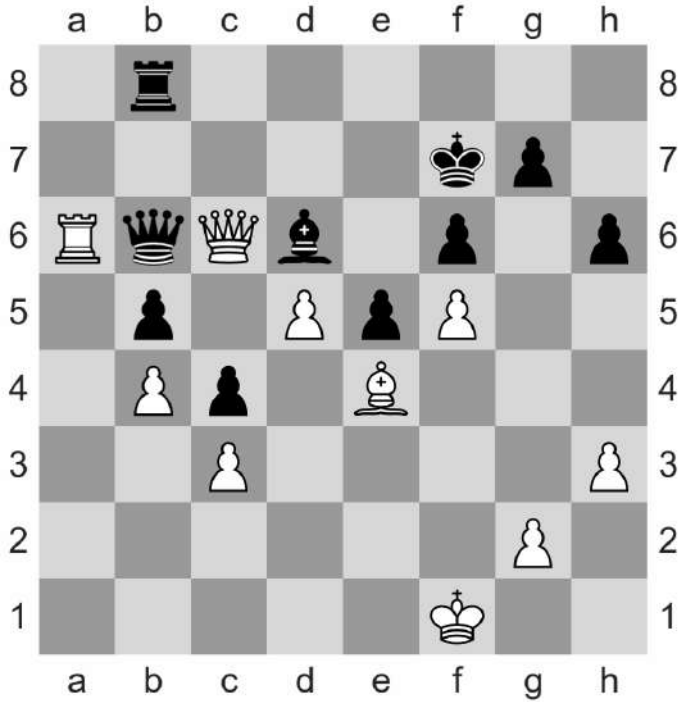



einfaches NN




tiefes NN

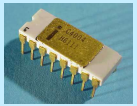
# *1997: Deep Blue*

# Timeline

IKS
Individuelle Softwarelösungen

1970 1980 1990 2000

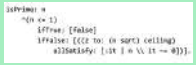
Mikroprozessor 

Netzwerke 

Der PC 

Die Maus 

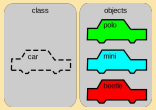
Pascal C SQL Modula2 Smalltalk C++



Software-Technik 

Unix 

RDBs TCP/IP

OOP 

GUIs 

2. Softwarekrise 

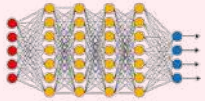
Genetische Algorithmen

Wissens-repräsen-tation

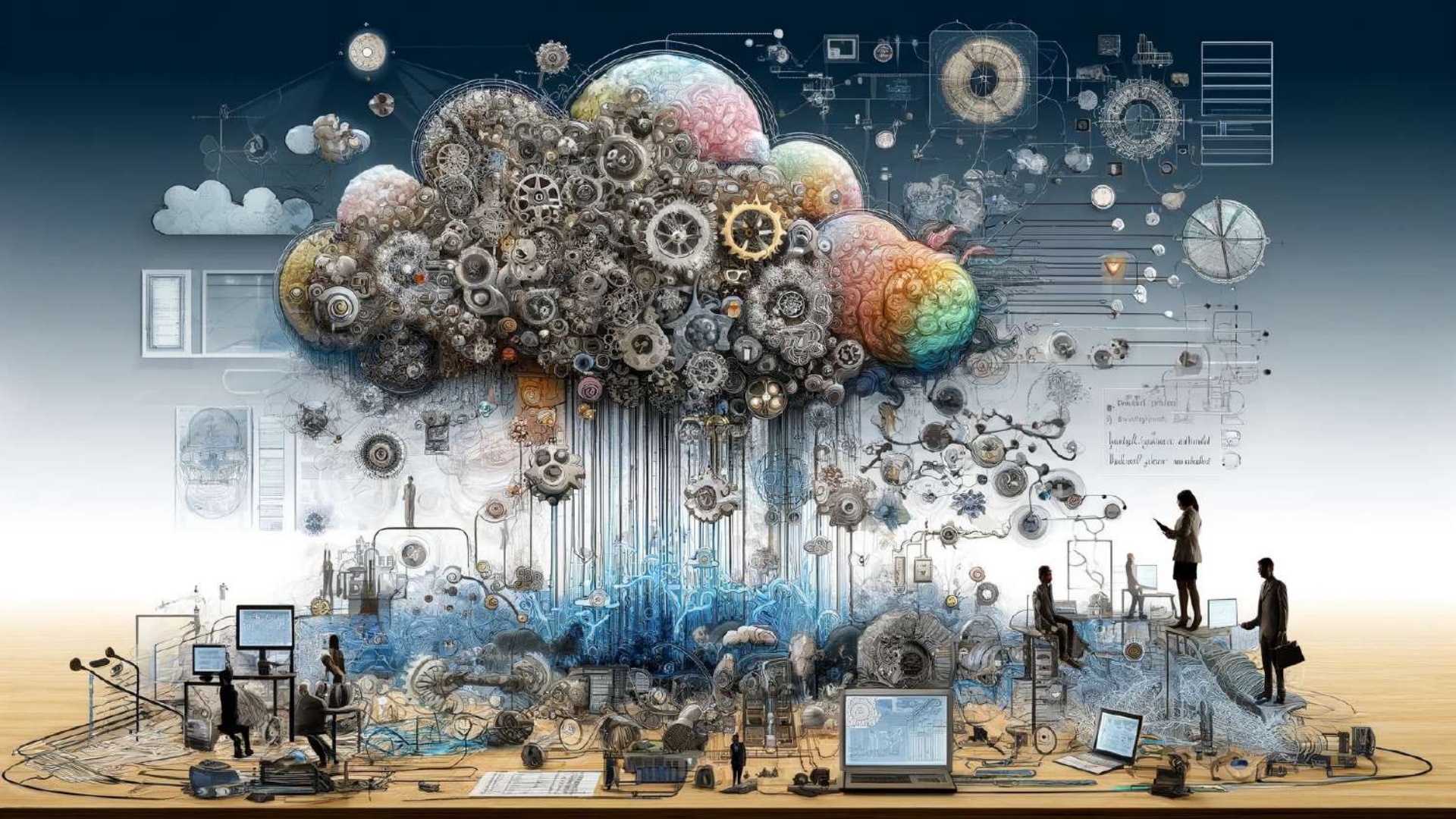
Experten-systeme

2. KI-Winter 

Back propagation

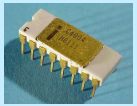
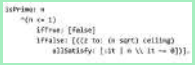
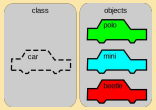
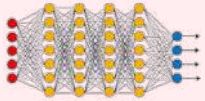
"Deep“ Learning 

Deep Blue 



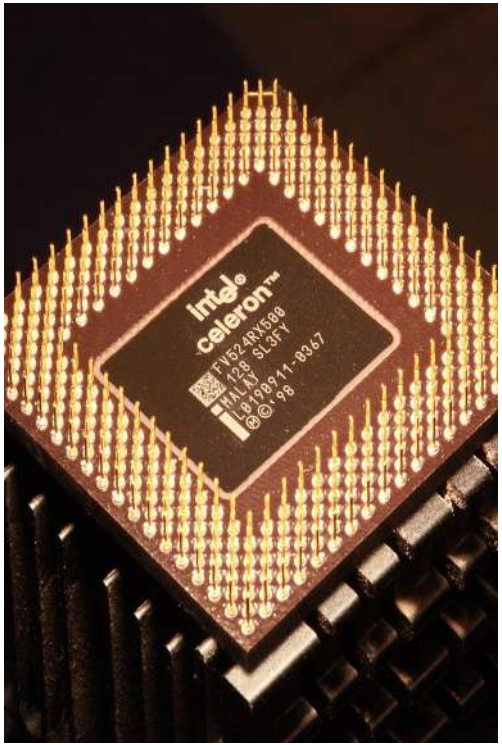
# Timeline

IKS
Individuelle Softwarelösungen

| 1970 | 1980 | 1990 | 2000 |
|---|---|---|---|
| Mikroprozessor  | Netzwerke  | Der PC  | Die Maus  | |
| Pascal | C | SQL | Modula2 | Smalltalk  | C++ | Python  | Ruby  | Java/-Script   | C#  |
| Software-Technik  | Unix  | RDBs  | TCP/IP | OOP  | GUIs  | 2. Softwarekrise  | V-Modell  | Open Source | UML RUP | Agile  | TDD | CI/CD |
| Genetische Algorithmen | Wissens-repräsen-tation | Experten-systeme | 2. KI-Winter  | Back propagation | "Deep“ Learning  | Deep Blue  |

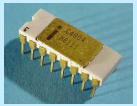
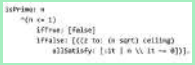
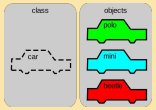
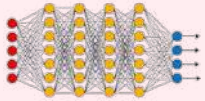

1990er: World Wide Web
Sir Tim Berners-Lee

# 1990er: Der moderne PC im Job und zu Hause



# Timeline

IKS
Individuelle Softwarelösungen

| 1970 | 1980 | 1990 | 2000 |
|---|---|---|---|
| Mikroprozessor  | Netzwerke  | Der PC  | Die Maus  | Internet-Boom  | PC, Flachbildschirm  |
| Pascal | C | SQL | Modula2 | Smalltalk  | C++ | Python  | Ruby  | Java/-Script   | C#  |
| Software-Technik  | Unix  | RDBs  | TCP/IP | OOP  | GUIs  | 2. Softwarekrise  | V-Modell  | Open Source | UML RUP | Agile  | TDD | CI/CD |
| Genetische Algorithmen | Wissens-repräsen-tation | Experten-systeme | 2. KI-Winter  | Back propagation | "Deep“ Learning  | Deep Blue  |

# 2010er: Das Smartphone

# 2010er: Leistungsfähige Multicore-CPUs und GPUs



2020er: Heutige Rechenzentren, Cloud Computing


2020er:
Der erste kommerzielle
Quantencomputer

# Hardware

IKS
Individuelle Softwarelösungen

| Jahr | Technik | Beispiel | Leistung | Speicher |
|---|---|---|---|---|
| 1642 | Mechanik | Pascaline, Blaise Pascal | | |
| 1930 | Relais | Konrad Zuse: Z3 | 1 | 176 |
| 1940 | Vakuumröhren | ENIAC: Erster digitaler Computer | 100 | 2000 |
| 1950 | Transistoren | Point-Contact Transistor, Bell | 1.000 | 10.000 |
| 1960 | Integrierte Schaltkreise | µA702, Fairchild Semiconductor | 10.000 | 100.000 |
| 1970 | Mikroprozessoren | Intel 4004, 4Bit, 740 kHz | 1.000.000 | 1.000.000 |
| 2000 | Smartphones | iPhone, Apple | 1.000.000.000.000 | 10.000.000.000 |
| 2010 | Cloud-Computing | AWS, Azure, GCP | quasi unbegrenzt | quasi unbegrenzt |
| 2020 | Quanten-Computing | IBM Q System One | nicht vergleichbar | nicht vergleichbar |

100 *Jahrzehnt*

# Timeline

IKS
Individuelle Softwarelösungen

2000 2010 2020 2030

Multicore CPUs GPUs

Megazentren

Smartphone

Quanten Computing

# 2000er: Scala

IKS
Individuelle Softwarelösungen

```
def isPrime(n: Int): Boolean = {
    if (n <= 1) false
    else !(2 to math.sqrt(n).toInt).exists(n % _ == 0)
}
```

# 2010er: Kotlin

IKS
Individuelle Softwarelösungen

```
fun isPrime(n: Int) =
    if (n <= 1) false
    else (2..sqrt(n)).none { it divides n }
```

# 2020er: Comment Driven Development (CDD)

IKS
Individuelle Softwarelösungen

```
/**
 * Testet eine Zahl auf Primzahleigenschaft in Python
 */

def is_prime(number):
    if number <= 1: return False
    for i in range(2, int(sqrt(number))+1):
        if number % i == 0:
            return False
    return True
```

# *Wie rede ich mit Maschinen?*

Level 1

### *1950er: Maschinen-Code*

```
11100000 00000010
11110000 00000010
11100000 00000001
11110000 00000010
10100010 00000010
11001010
11110000 00000010
11101100 00010000 00000000
11110000 00000010
11100000 00000001
11110000 00000010
10010000 11110000
00111000
01100000
00011000
01100000
```

Level 2

### *1960er: Assembler*

```
start:
      LDA number
      JSR is_prime
      BCS not_prime
is_prime:
      CPX #2
      BEQ prime_done
      CPX #1
      BEQ prime_done
      LDX #2
check_divisible:
      DEX
      BEQ prime_done
      CPX number
      BEQ prime_done
      CPX #1
      BEQ prime_done
      BCC check_divisible
      SEC
      RTS
prime_done:
      CLC
      RTS
```

Level 3

### *1960er: Cobol*

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 Num PIC 9(5).
01 IsPrime PIC X.

PROCEDURE DIVISION.
    Accept-Num.
    Perform Prime-Number-Test.
    STOP RUN.

Accept-Num.
    ACCEPT Num.

Prime-Number-Test.
    IF Num <= 1
        MOVE "N" TO IsPrime
    ELSE
        PERFORM Check-Divisors UNTIL Num
<= Sqrt(Num)
        IF IsPrime IS INITIAL
            MOVE "Y" TO IsPrime
        END-IF
    END-IF.

Check-Divisors.
    DIVIDE Num BY Num - 1 GIVING
Remainder
    IF Remainder = 0
        MOVE "N" TO IsPrime
    END-IF
    SUBTRACT 1 FROM Num.
```

Level 3

### *1960er: Fortran I*

```
      READ(5,2) NUM
2     FORMAT(I)
      I = 2
3     IF(I**2-NUM) 4,5,5
4     PRINT 6
      STOP
5     IF(I-1) 6,7,7
6     I = I + 1
      GOTO 3
7     PRINT 8
      STOP
8     FORMAT(1H1, ' ', I5)
      END
```

Level 3

### *1980er: Python*

```
def is_prime(number):
    if number <= 1: return False
    for i in range(2, int(sqrt(number))+1):
        if number % i == 0:
            return False
    return True
```

Level 4

### *2000er: Scala*

```
def isPrime(n: Int): Boolean = {
    if (n <= 1) false
    else !(2 to math.sqrt(n).toInt).exists(n % _ == 0)
}
```

Level 4

### *2010er: Kotlin*

```
fun isPrime(n: Int) =
    if (n <= 1) false
    else (2..sqrt(n)).none { it divides n }
```

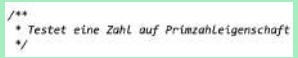
Level 5

### *2020er: Comment Driven Development (CDD)*

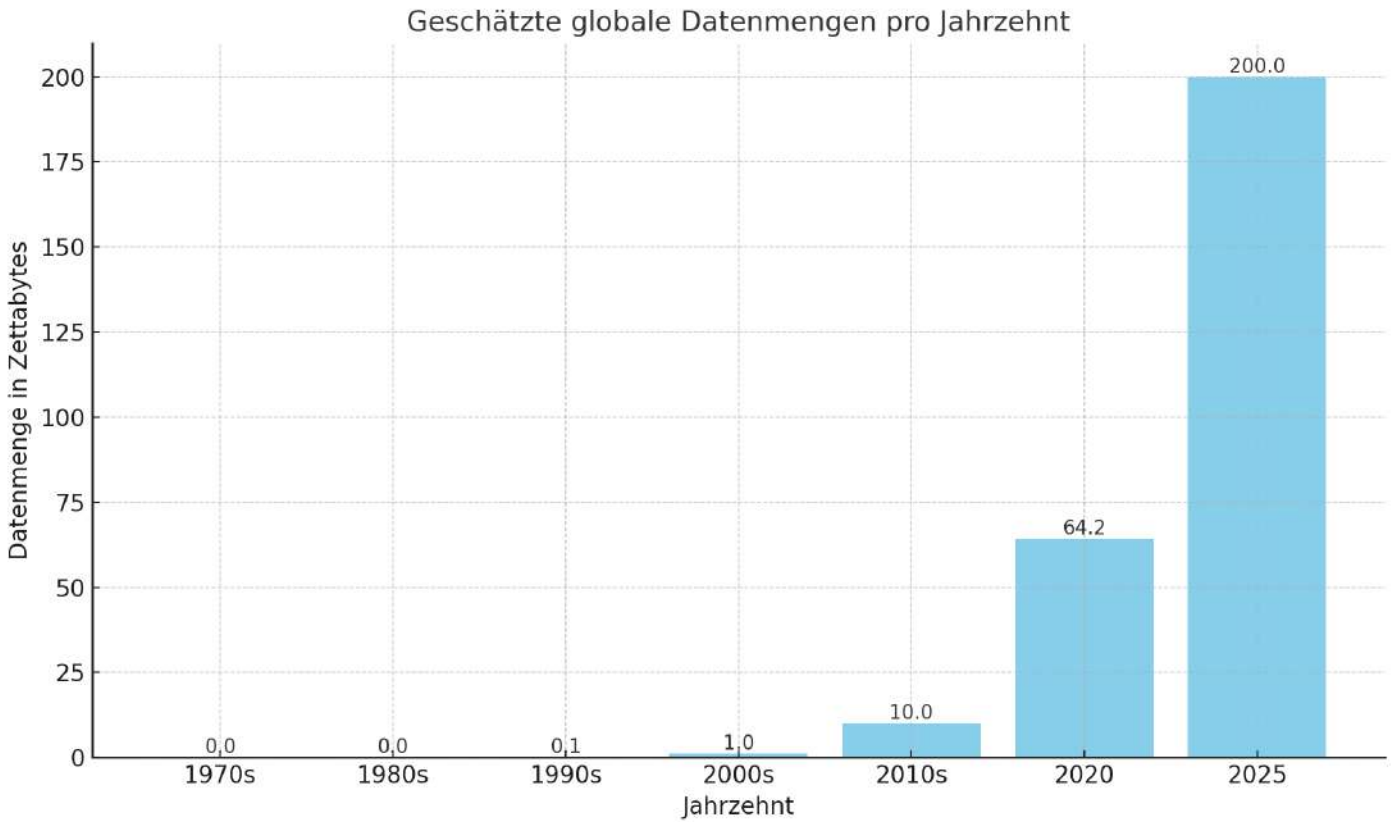
```
/**
 * Testet eine Zahl auf Primzahleigenschaft
 */
```

# Timeline

IKS
Individuelle Softwarelösungen

| 2000 | 2010 | 2020 | 2030 |
|---|---|---|---|
| Multicore CPUs GPUs | Megazentren  | Smartphone  | Quanten Computing  |
| Scala  | Go, Rust, Julia | TypeScript Kotlin  | Low Code | Natürliche Sprache  |
| | | | |
| | | | |

# 2010er: BigData




# Timeline

Multicore
CPUs
GPUs

Megazentren

Smartphone

Quanten Computing

Scala

Go,
Rust,
Julia

TypeScript Kotlin

```
fun isPrime(n: Int) =
    if (n <= 1) false
    else (2..sqrt(n)).none { it divides n }
```

Low Code

Natürliche Sprache

```
/**
 * Testet eine Zahl auf Primzahleigenschaft
 */
```

SOA
DDD

Google
Facebook
Apple
Amazon

Cloud

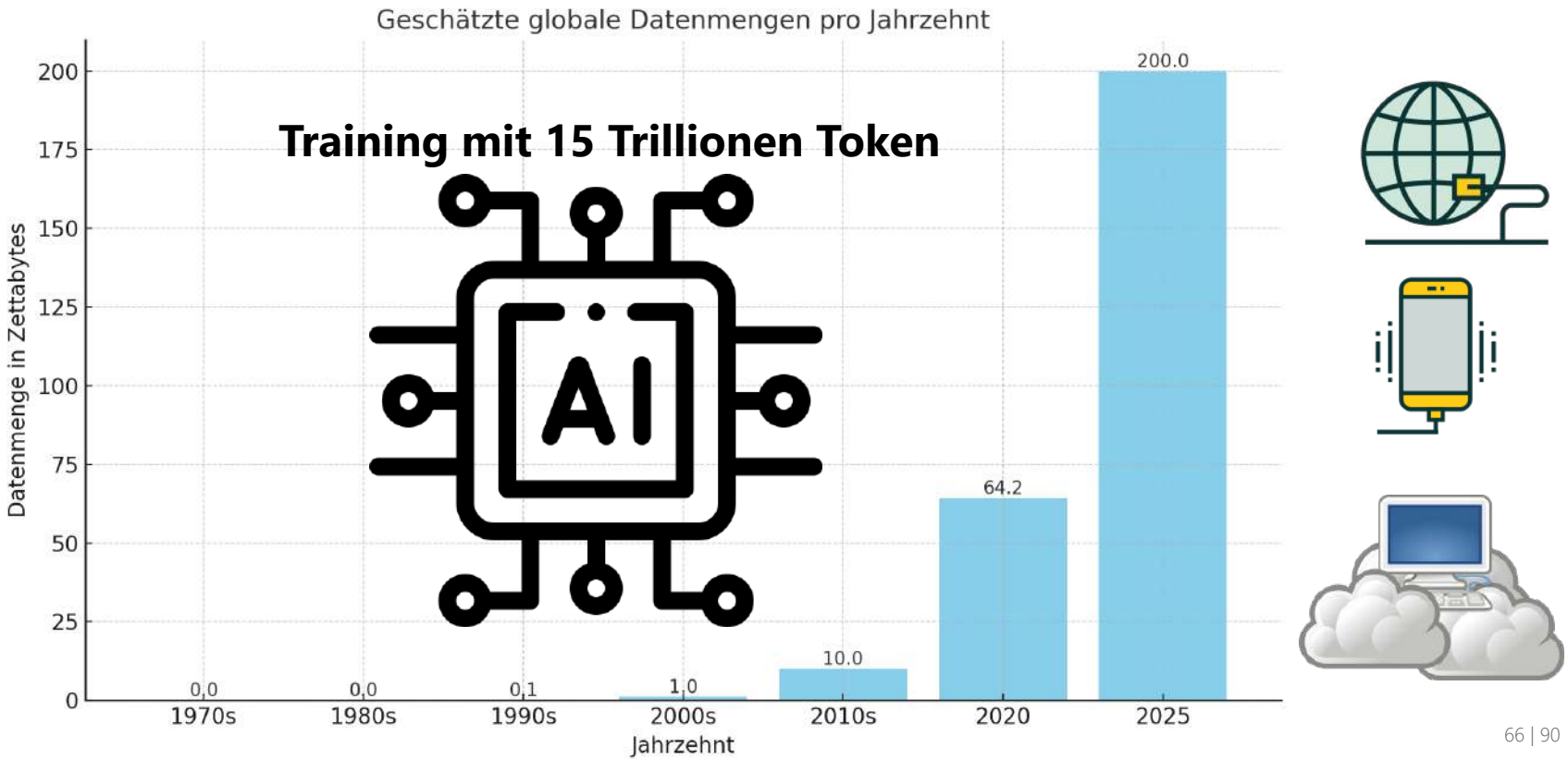
Big Data

Blockchain

Container

DevOps

# 2010er: BigData



# 2006: Unsupervised pre-training

IKS
Individuelle Softwarelösungen

## A fast learning algorithm for deep belief nets *

**Geoffrey E. Hinton** and **Simon Osindero**
Department of Computer Science University of Toronto
10 Kings College Road
Toronto, Canada M5S 3G4
{hinton, osindero}@cs.toronto.edu

**Yee-Whye Teh**
Department of Computer Science
National University of Singapore
3 Science Drive 3, Singapore, 117543
tehyw@comp.nus.edu.sg

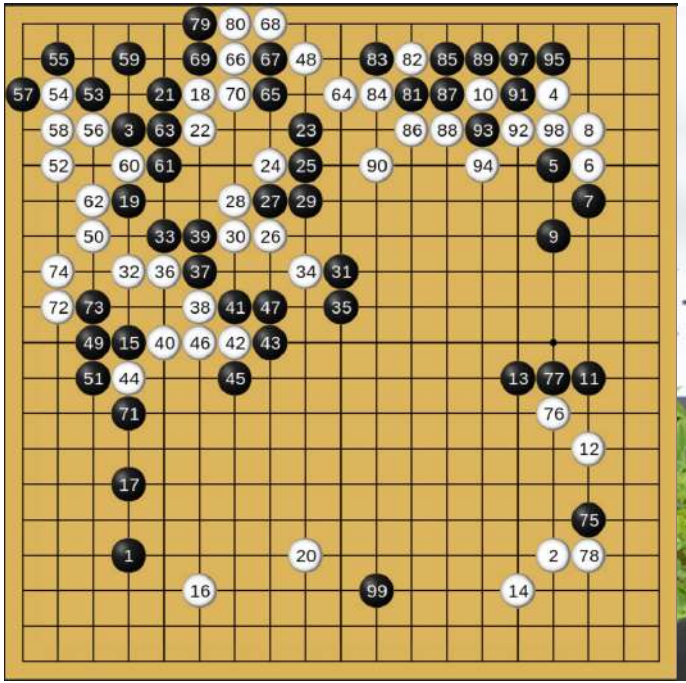
### Abstract

We show how to use "complementary priors" to eliminate the explaining away effects that make inference difficult in densely-connected belief nets that have many hidden layers. Using complementary priors, we derive a fast, greedy algorithm that can learn deep, directed belief networks one layer at a time, provided the top two layers form an undirected associative memory. The fast, greedy algorithm is used to initialize a slower learning procedure that fine-tunes the weights using a contrastive version of the wake-sleep algorithm. After fine-tuning, a network with three hidden layers forms a very good generative model of the joint distribution of handwritten digit images and their labels. This generative model gives

remaining hidden layers form a directed acyclic graph that converts the representations in the associative memory into observable variables such as the pixels of an image. This hybrid model has some attractive features:

1. There is a fast, greedy learning algorithm that can find a fairly good set of parameters quickly, even in deep networks with millions of parameters and many hidden layers.
2. The learning algorithm is unsupervised but can be applied to labeled data by learning a model that generates both the label and the data.
3. There is a fine-tuning algorithm that learns an excellent generative model which outperforms discriminative methods on the MNIST database of hand-written digits.

# 2016 : AlphaGo siegt gegen Lee Sedol



- überwachtes Lernen
- unüberwachtes Lernen
- Deep Neural Networks
  - Policy Network
  - Value Network
- Reinforcement Learning

# 2017: Transformer

## Attention Is All You Need

**Ashish Vaswani*** Google Brain avaswani@google.com

**Noam Shazeer*** Google Brain noam@google.com

**Niki Parmar*** Google Research nikip@google.com

**Jakob Uszkoreit*** Google Research usz@google.com

**Llion Jones*** Google Research llion@google.com

**Aidan N. Gomez*** † University of Toronto aidan@cs.toronto.edu

**Łukasz Kaiser*** Google Brain lukaszkaiser@google.com

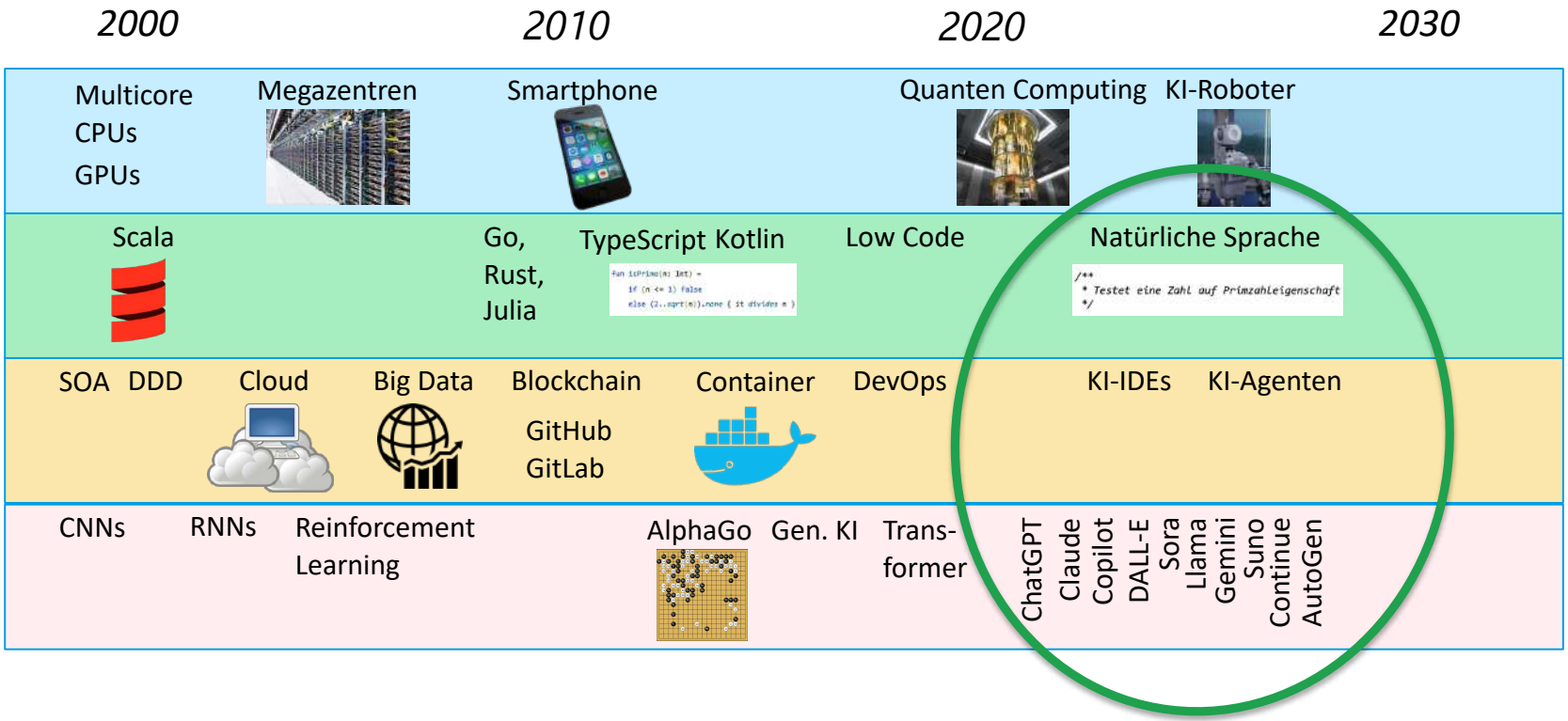
**Illia Polosukhin*** ‡ illia.polosukhin@gmail.com

### Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer,

# Timeline

IKS
Individuelle Softwarelösungen



# Timeline

IKS Individuelle Softwarelösungen

2000 2010 2020 2030

Multicore CPUs GPUs

Megazentren

Smartphone

Quanten Computing

KI-Roboter

Scala

Go, Rust, Julia

TypeScript Kotlin

```
fun isPrime(n: Int) =
    if (n <= 1) false
    else (2..sqrt(n)).none { it divides n }
```

Low Code

Natürliche Sprache

```
/**
 * Testet eine Zahl auf Primzahleigenschaft
 */
```

SOA DDD

Cloud

Big Data

Blockchain

GitHub
GitLap

Container

DevOps

KI-IDEs

3. Softwarekrise?

CNNs

RNNs

Reinforcement Learning

AlphaGo

Gen. KI

Trans-former

ChatGPT
Claude
Copilot
DALL-E
Sora
Llama
Gemini
Suno
Continue
AutoGen

3. KI-Winter?

# 2024: KANs

IKS
Individuelle Softwarelösungen

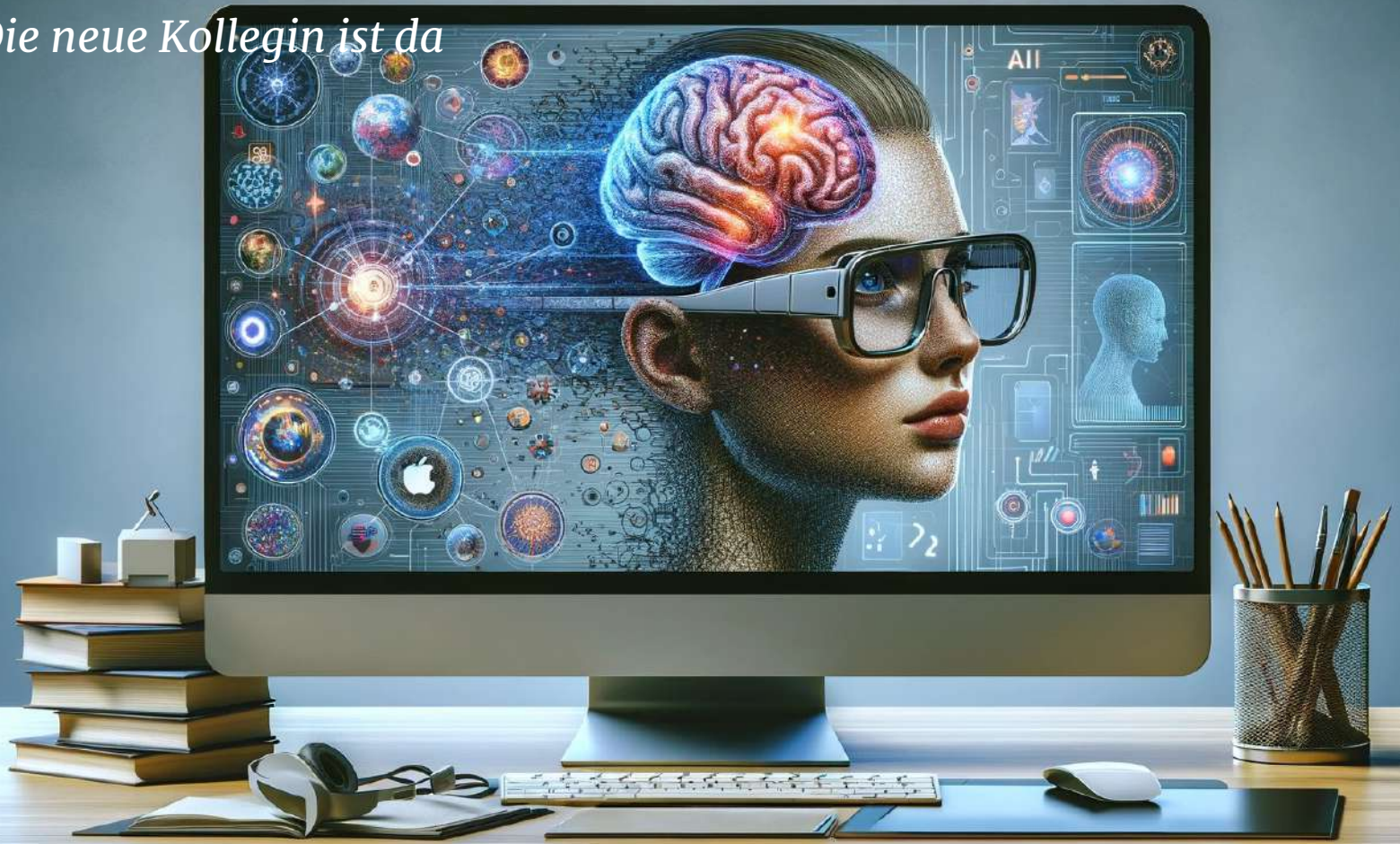
## KAN: Kolmogorov–Arnold Networks

**Ziming Liu**[1,4*] **Yixuan Wang**[2] **Sachin Vaidya**[1] **Fabian Ruehle**[3,4]
**James Halverson**[3,4] **Marin Soljačić**[1,4] **Thomas Y. Hou**[2] **Max Tegmark**[1,4]
[1] Massachusetts Institute of Technology
[2] California Institute of Technology
[3] Northeastern University
[4] The NSF Institute for Artificial Intelligence and Fundamental Interactions

### Abstract

Inspired by the Kolmogorov-Arnold representation theorem, we propose Kolmogorov-Arnold Networks (KANs) as promising alternatives to Multi-Layer Perceptrons (MLPs). While MLPs have *fixed* activation functions on *nodes* ("neurons"), KANs have *learnable* activation functions on *edges* ("weights"). KANs have no linear weights at all – every weight parameter is replaced by a univariate function parametrized as a spline. We show that this seemingly simple change makes KANs outperform MLPs in terms of accuracy and interpretability. For accuracy, much smaller KANs can achieve comparable or better accuracy than much larger MLPs in data fitting and PDE solving. Theoretically and empirically, KANs possess faster neural scaling laws than MLPs. For interpretability, KANs

*Die neue Kollegin ist da*

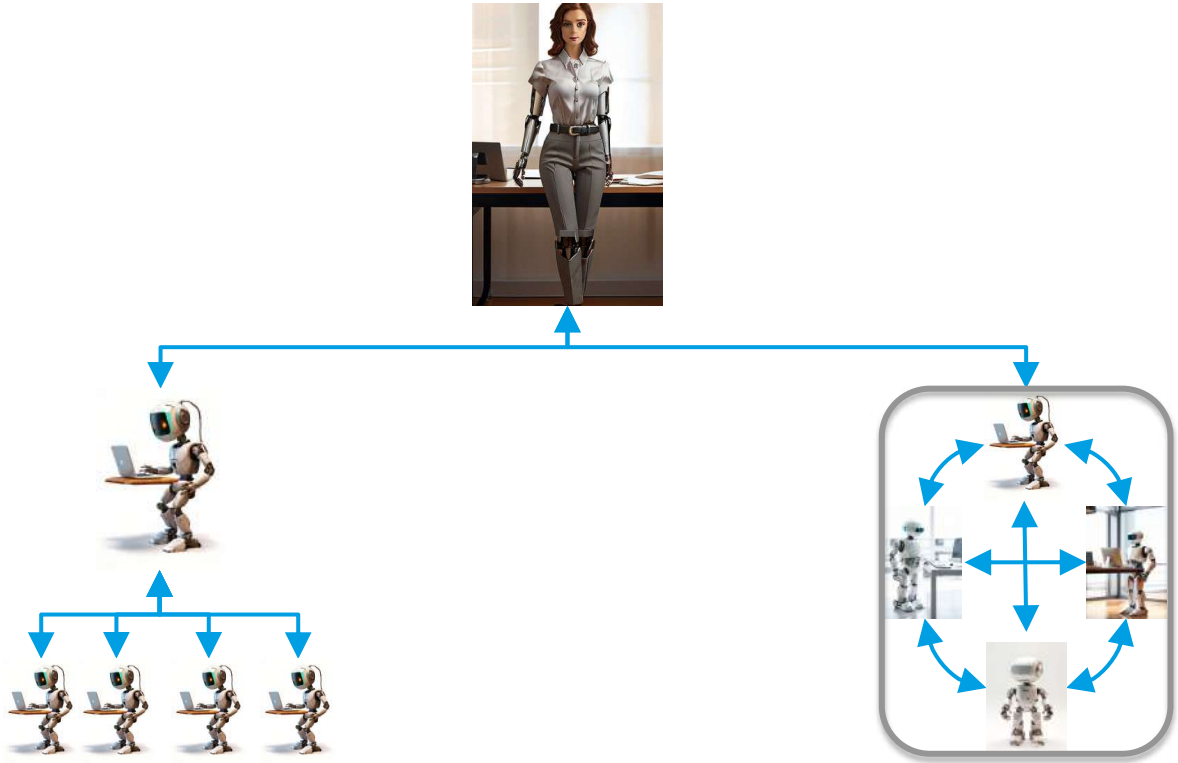


# *KI – Werkzeug oder Kollegin?*

IKS
Individuelle Softwarelösungen


Die neue Kollegin ist da

# *Die neuen Kollegin steht für ein Team neuer Kolleginnen*



# *Die neue Kollegin ist da*

Die neue Kollegin ist da

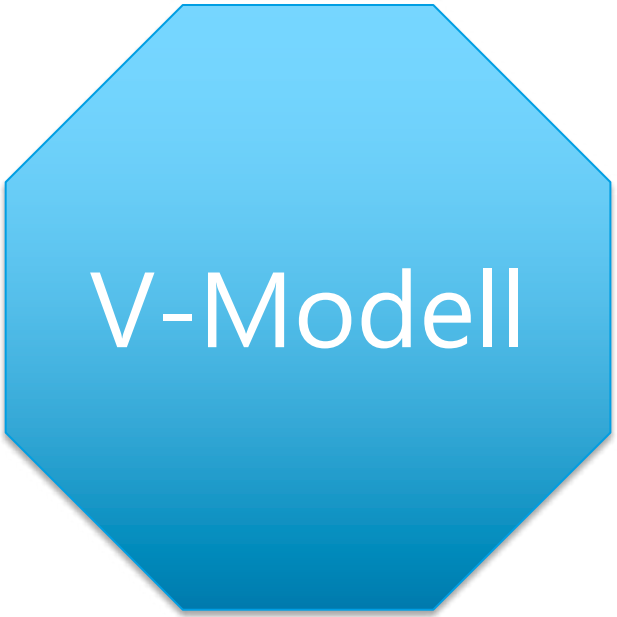
# *Die neuen Kollegen – Hype oder bleibende Innovation?*

IKS
Individuelle Softwarelösungen

# *Wie arbeiten Menschen zusammen?*

IKS
Individuelle Softwarelösungen

# *Klassische Zusammenarbeit*

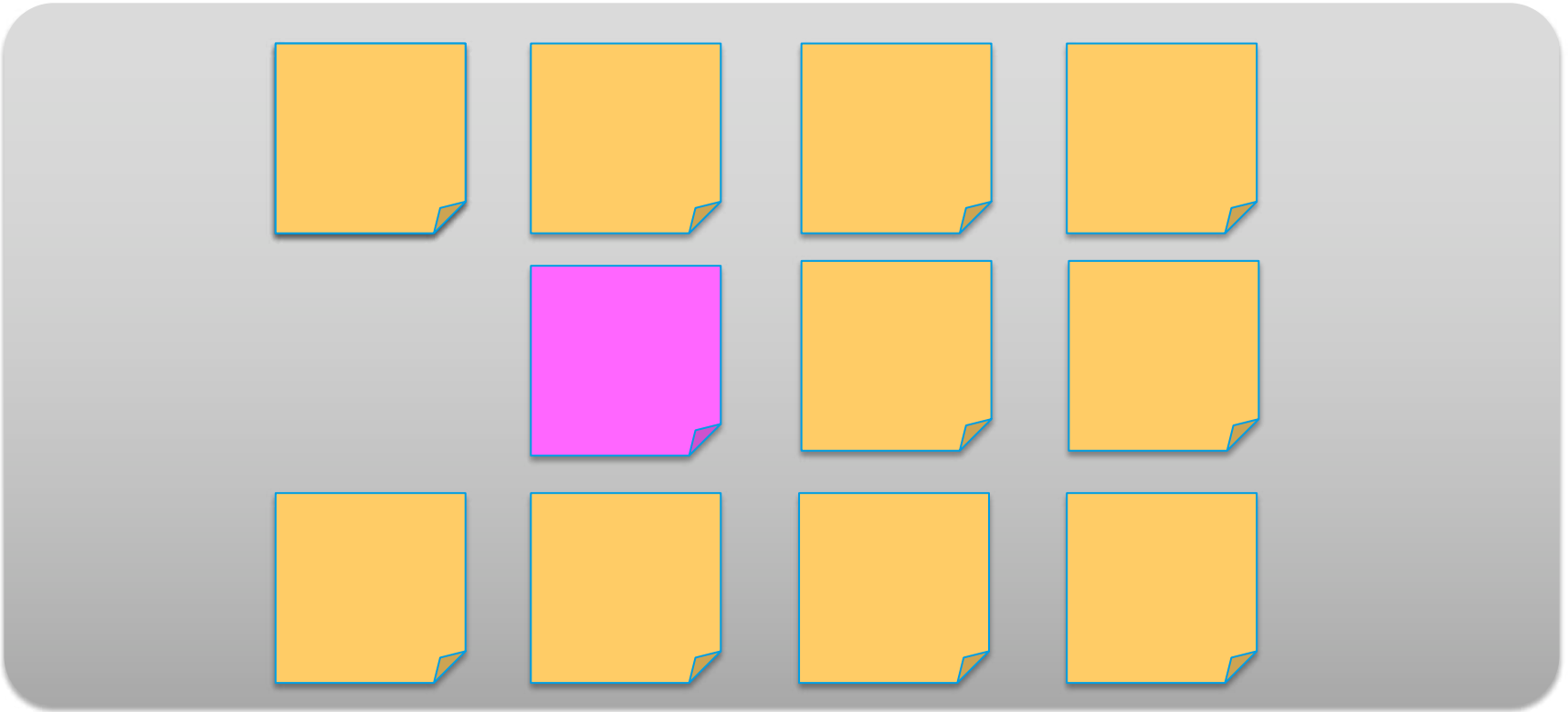



# *Klassisch: Analyse*

IKS
Individuelle Softwarelösungen

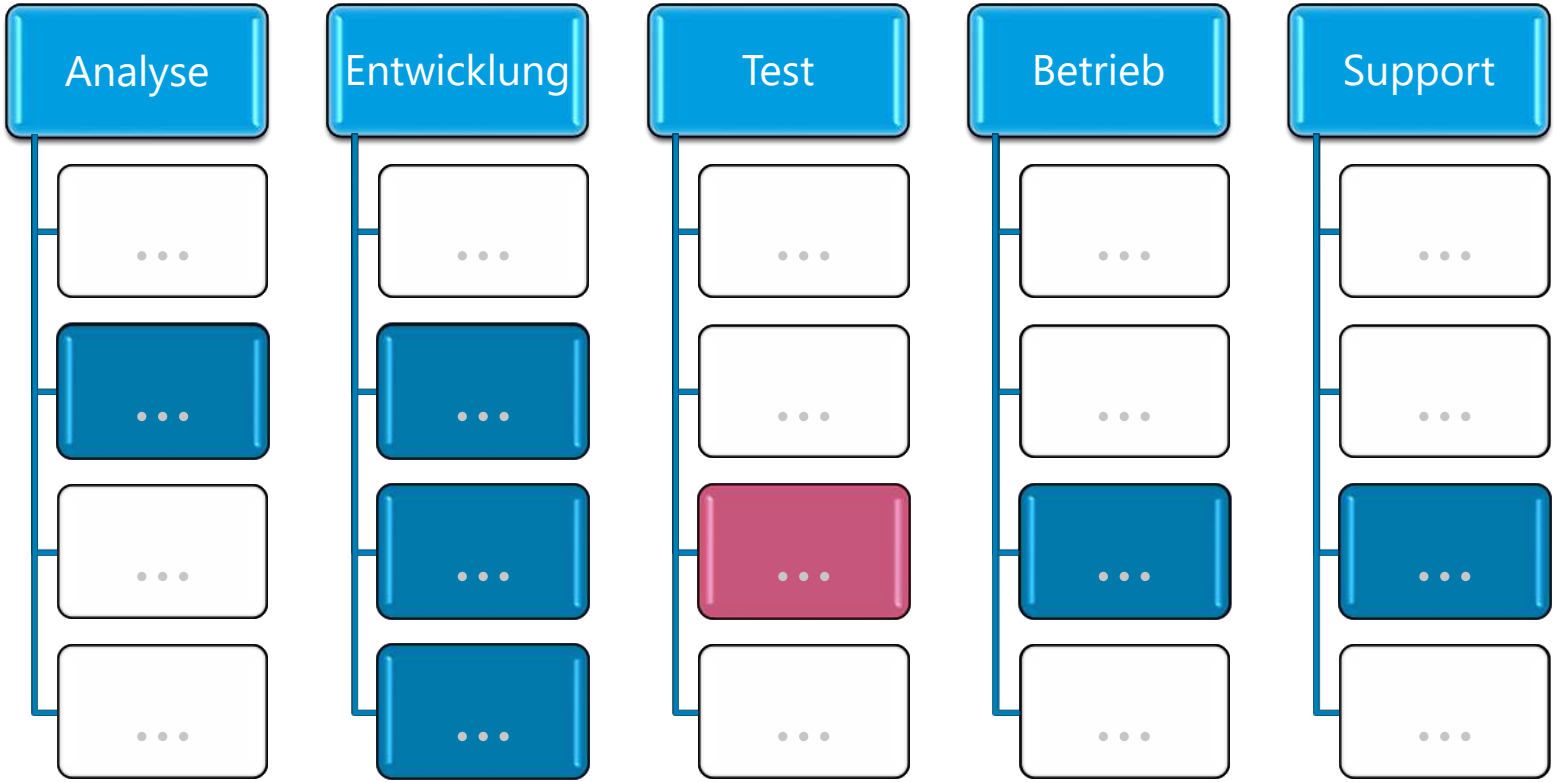
Die neue Kollegin ist da

# *Modern: Event Storming*

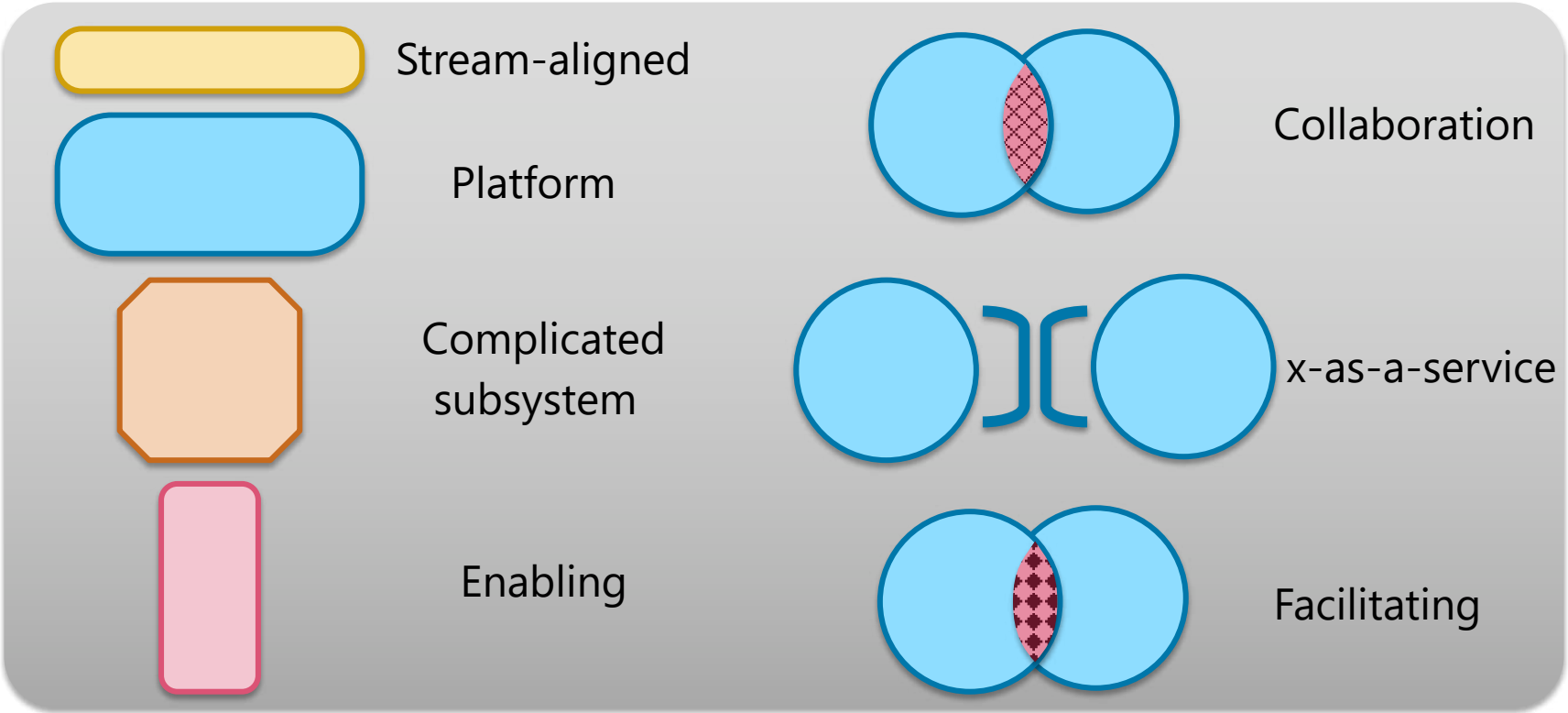


# *Klassisch: Projekt kontra Organisation*

IKS
Individuelle Softwarelösungen



# *Modern: Team Topologies*

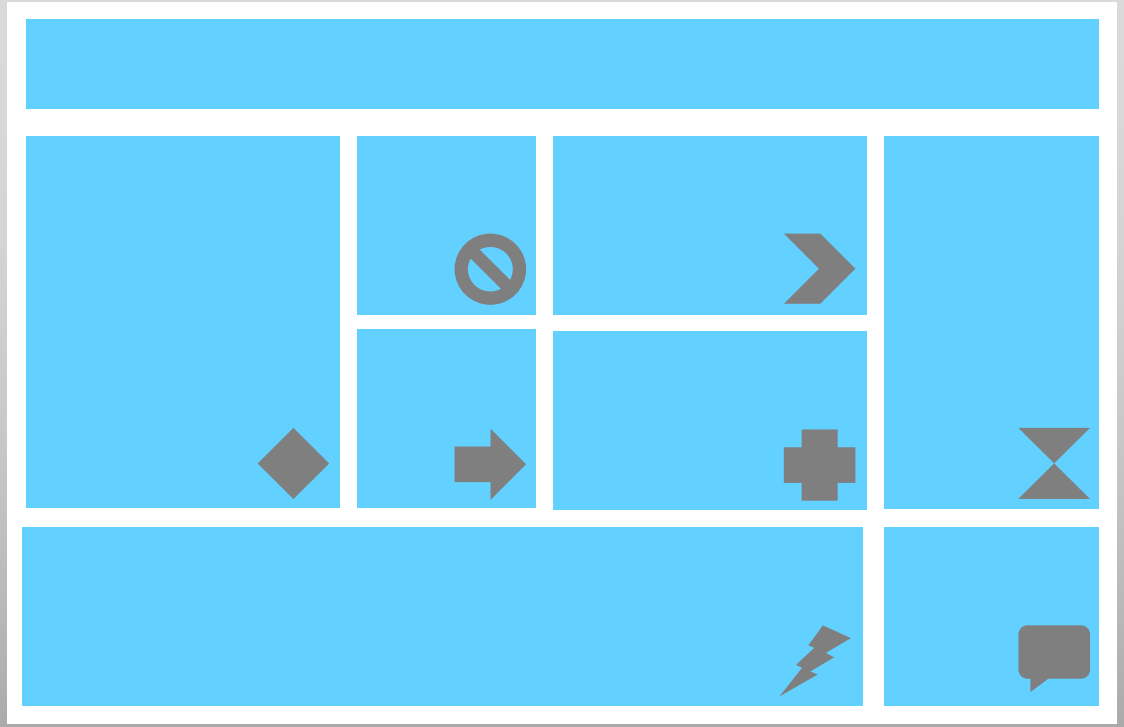


# *Klassisch: Der „Plan“*

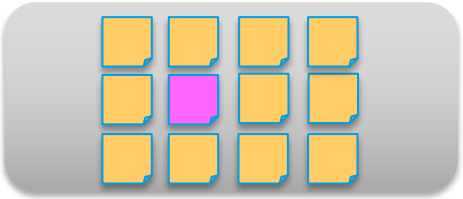
IKS
Individuelle Softwarelösungen

# *Modern: Der Canvas*

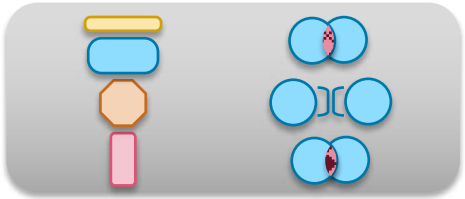
IKS
Individuelle Softwarelösungen



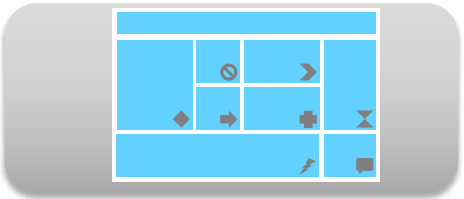
# *Was waren die Auslöser?*



Event Storming



Team Topologies



Canvas

Niemand hat mehr das gesamte Wissen

Im Team schneller „liefern“ können

Selbstständig und eigenverantwortlich arbeiten

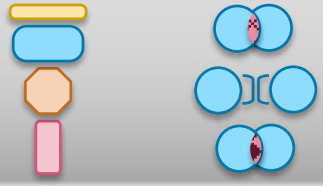
# *Worauf basieren sie?*

IKS
Individuelle Softwarelösungen

Event Storming

Niemand hat mehr das gesamte Wissen

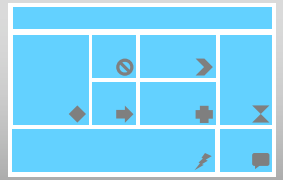
Durch einfache Methoden kann jeder mitarbeiten



Team Topologies

Im Team schneller „liefern“ können

Aufgaben und Zusammenarbeit sind klar geregelt



Canvas

Selbstständig und eigenverantwortlich arbeiten

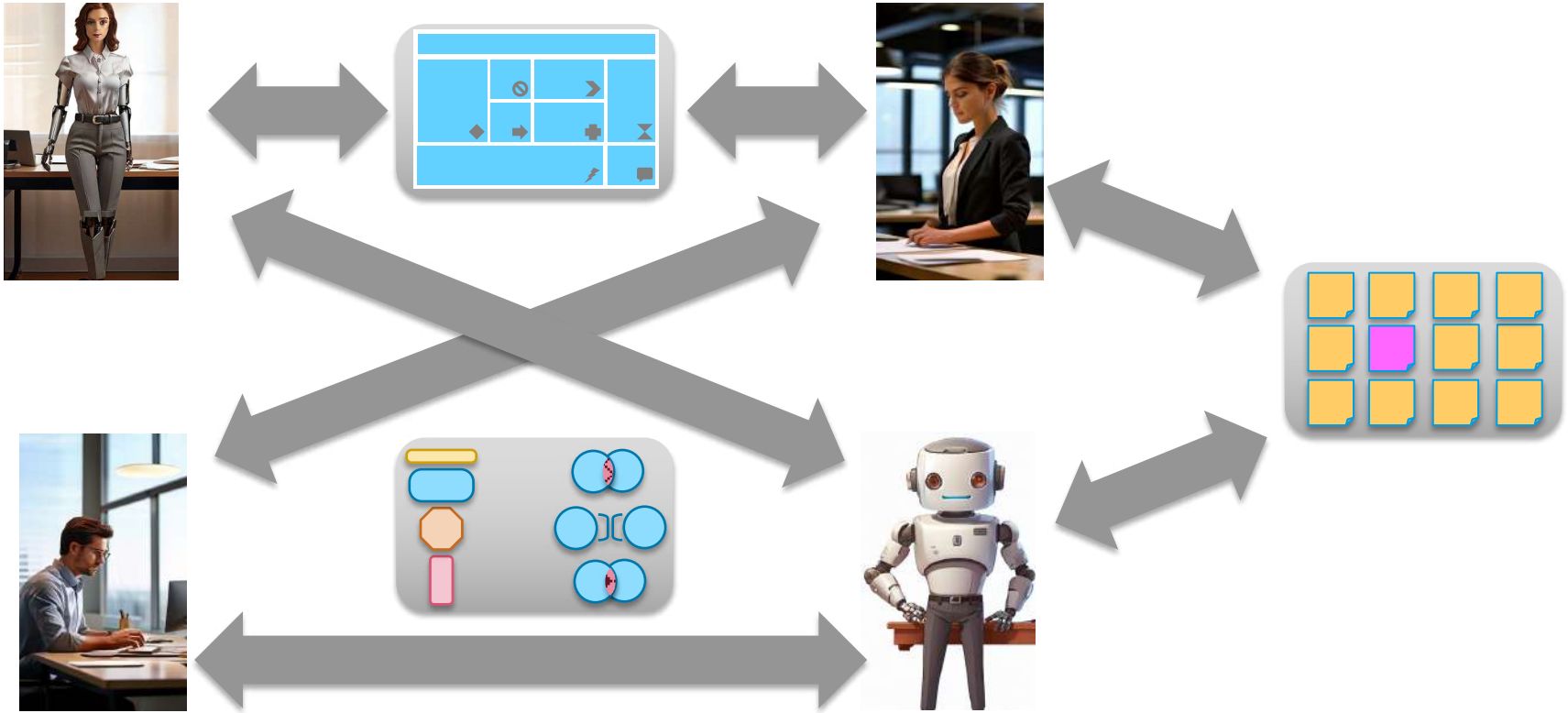
Alles Wesentliche stets im Blick

# *Hype oder bleibende Innovation?*

IKS
Individuelle Softwarelösungen

# *Die Methoden für Menschen helfen auch mit den neuen Kollegen*

IKS
Individuelle Softwarelösungen



# *Vielen Dank!*

IKS
Individuelle Softwarelösungen

*www.iks-gmbh.com*

# *Abbildungsverzeichnis*

IKS
Individuelle Softwarelösungen

- Pascaline:
  - https://commons.wikimedia.org/wiki/File:Pascaline-CnAM_823-1-IMG_1506-white.jpg
- Analytical Mashine
  - https://de.wikipedia.org/wiki/Analytical_Engine#/media/Datei:Babbages_Analytical_Engine,_1834-1871._(9660574685).jpg
  - https://de.wikipedia.org/wiki/Charles_Babbage#/media/Datei:Engaving_of_Charles_Babbage_from_Mechanics_Magazine.jpg
- Ada Lovelace
  - https://en.wikipedia.org/wiki/Note_G#/media/File:Diagram_for_the_computation_of_Bernoulli_numbers.jpg
  - https://en.wikipedia.org/wiki/Note_G#/media/File:Ada_Lovelace_portrait.jpg
- Zuse
  - https://de.wikipedia.org/wiki/Zuse_Z3#/media/Datei:Elektromagnetischerspeicher_zuse_relais.jpg
  - https://de.wikipedia.org/wiki/Zuse_Z3#/media/Datei:Z3_Deutsches_Museum.JPG
- ENIAC
  - https://www.flickr.com/photos/epitti/2586212612 (Creative Commons, Attribution 2.0 Generic)
  - https://upload.wikimedia.org/wikipedia/commons/c/ca/AVIDAC_--_First_Argonne_Computer_%281953%29.jpg

# *Abbildungsverzeichnis*

- TX-o:
  - https://commons.wikimedia.org/wiki/File:KT3107_transistors.jpg
  - https://miro.medium.com/v2/resize:fit:1400/format:webp/1*O1_WWZn7uiPtDdZrd_i7qQ.jpeg
- Der erste IC
  - https://commons.wikimedia.org/wiki/File:4-%D0%B1%D1%96%D1%82%D0%BD%D0%B8%D0%B9_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D0%BE%D1%80_Intel_4004.jpg
  - https://upload.wikimedia.org/wikipedia/commons/1/18/Busicom_overall_1.jpg
- Die Gewinner sind
  - KI-generiert von DALL-E
- Die Gewinner sind
  - KI-generiert von DALL-E
- Fuzzy-Logic
  - https://de.wikipedia.org/wiki/Fuzzylogik#/media/Datei:Fuzzy-Operatoren.png

# *Abbildungsverzeichnis*

IKS
Individuelle Softwarelösungen

- Perzeptron
  - https://upload.wikimedia.org/wikipedia/commons/2/2d/PerceptronOverview.svg
- Der 1. KI-Winter
  - KI-generiert von neuroflash
- Warum ist Software nur so teuer?
  - KI-generiert von DALL-E
- NLS:
  - https://de.wikipedia.org/wiki/ON-Line_System#/media/Datei:ON-Line_System_(NLS),_SRI_(1960-1970s)_-_three-button_mouse_and_chord_keyboard_-_Computer_History_Museum.jpg
- Macintosh
  - https://de.wikipedia.org/wiki/Grafische_Benutzeroberfl%C3%A4che#/media/Datei:Mac_Classic.jpg
- Objektorientierung
  - https://commons.wikimedia.org/wiki/File:CPT-OOP-objects_and_classes.svg#/media/File:CPT-OOP-objects_and_classes.svg

# *Abbildungsverzeichnis*

IKS
Individuelle Softwarelösungen

- Netzwerke
  - https://commons.wikimedia.org/wiki/File:IP-Paket_Routing_%C3%BCber_Netzwerke.svg
- Unix
  - https://www.flickr.com/photos/rudolf_schuba/153225000/in/photostream/ (Creative Commons, Attribution 2.0 Generic)
  - https://commons.wikimedia.org/wiki/File:Tux.svg
- Zu vieles neu:
  - https://commons.wikimedia.org/wiki/File:Server_icon_CC0.svg
  - https://commons.wikimedia.org/wiki/File:Database-icon.svg
  - https://www.vectorstock.com/royalty-free-vector/outline-object-oriented-programming-icon-isolated-vector-28254018
  - https://commons.wikimedia.org/wiki/File:Noun_project_network_icon_1365244_cc.svg
  - https://commons.wikimedia.org/wiki/File:Brain_with_a_head_outline_and_grey_and_blue_puzzle_pieces.jpg
- 2. Softwarekrise
  - KI-generiert von DALL-E

# *Abbildungsverzeichnis*

IKS
Individuelle Softwarelösungen

- 2. KI-Winter
  - KI-generiert von DALL-E
- Timeline
  - https://vectorportal.com/de/vector/winter-landschaft-vektor-bild/27130
  - https://commons.wikimedia.org/wiki/File:Database-icon.svg
- Deep Blue
  - https://de.wikipedia.org/wiki/Deep_Blue#/media/Datei:Deep_Blue.jpg
  - https://commons.wikimedia.org/wiki/File:Kasparov-23.jpg
- Komplexität
  - KI-generiert von DALL-E
- World Wide Web
  - KI-generiert von DALL-E

# *Abbildungsverzeichnis*

IKS
Individuelle Softwarelösungen

- Der moderne PC
  - https://www.pexels.com/de-de/foto/schreibtisch-buro-tisch-technologie-7238759
  - https://www.flickr.com/photos/e2/393217813
- Das Smartphone
  - https://www.trustedreviews.com/wp-content/uploads/sites/54/2023/11/Best-smartphone-3-920x518.jpg
  - https://commons.wikimedia.org/wiki/File:Apple_Devices.jpg
- Multicore
  - https://www.flickr.com/photos/3336/27830149309 (Creative Commons, Attribution 2.0 Generic)
  - https://commons.wikimedia.org/wiki/File:Intel_CPU_Core_i7_6700K_Skylake_top.jpg
- Rechenzentren
  - KI-generiert von DALL-E
- Quantencomputer
  - KI-generiert von DALL-E

# *Abbildungsverzeichnis*

IKS
Individuelle Softwarelösungen

- BigData
  - https://commons.wikimedia.org/wiki/File:Internet-icon.png
  - https://commons.wikimedia.org/wiki/File:Mobile-phone.png
  - https://commons.wikimedia.org/wiki/File:Cloud_computing_icon.svg
- Timeline:
  - https://commons.wikimedia.org/wiki/File:Docker-svgrepo-com.svg
  - https://picryl.com/media/darpa-big-data-a1d68a
  - https://bostondynamics.com/blog/electric-new-era-for-atlas/
  - https://commons.wikimedia.org/wiki/File:Big_data_%2821036%29_-_The_Noun_Project.svg
- AlphaGo
  - https://upload.wikimedia.org/wikipedia/commons/3/38/AlphaGo_Fan_Huiren_aurka_(2).png
  - https://citaty-slavnych.sk/media/authors/lee-sedol.600w.webp
- Die neue Kollegin ist da
  - KI-generiert von DALL-E

# *Abbildungsverzeichnis*

# *Abbildungsverzeichnis*

IKS
Individuelle Softwarelösungen