Machine Called Computer

Part 3

- More on "Abstraction"

References:

1. Computer Organization and Design & Computer Architecture, Hennessy and Patterson (slides are adapted from those by the authors)

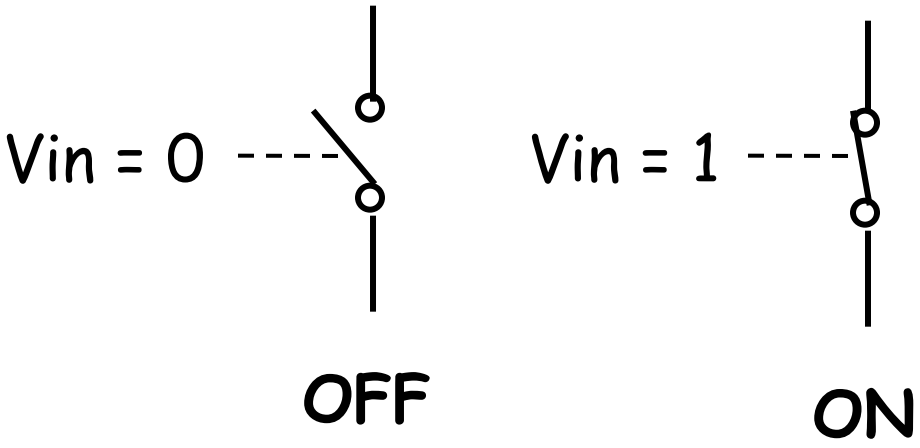
Design processor using 10^9 transistors

(How do we handle complexity?)

cpu -> processor로 용여 변화(multicore로 변화되어서)

3-Terminal Digital Switches (반복)

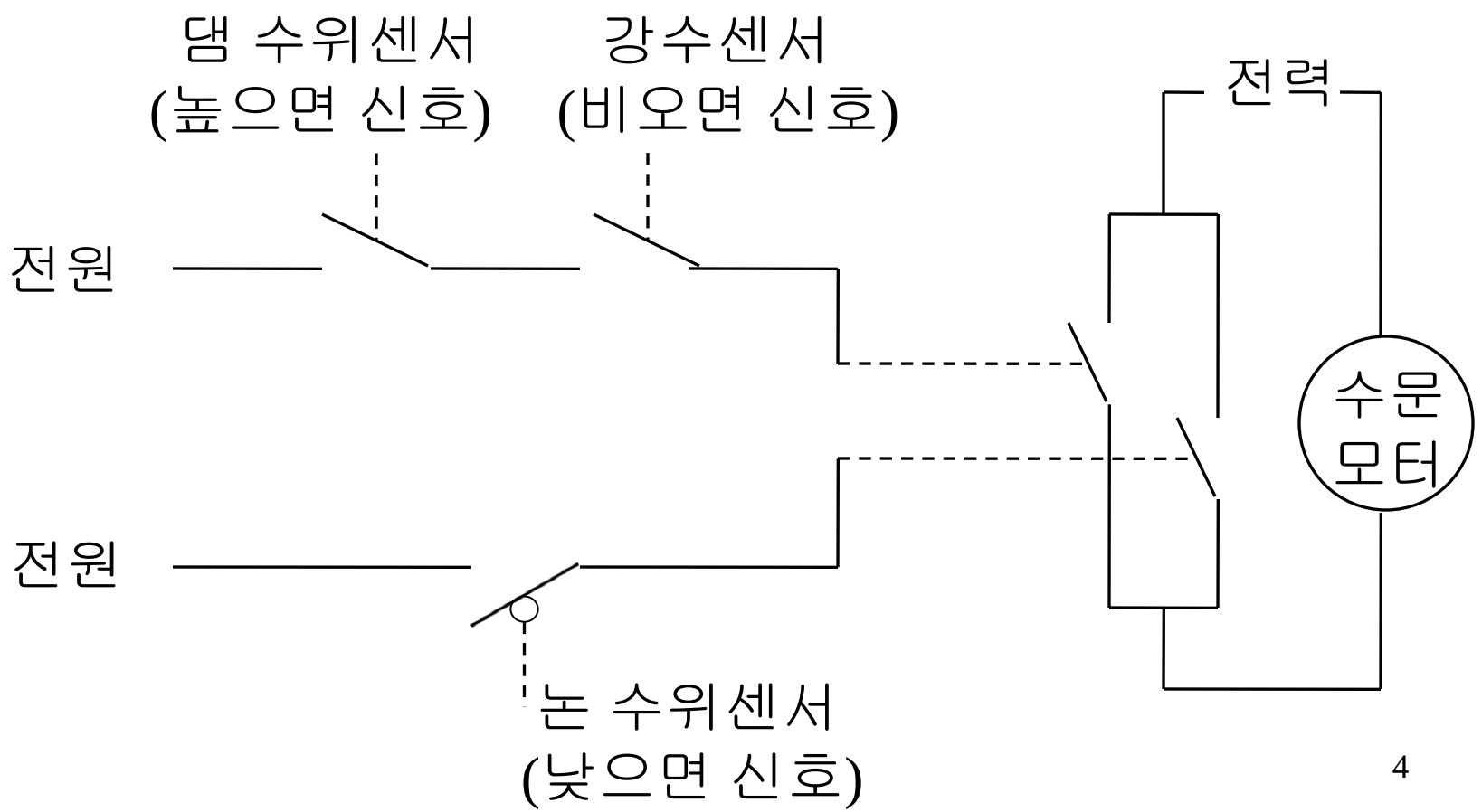
- Need for three-terminal switching device
  - Control signal, flow between the remaining two
    - Digital switch (ON, OFF)



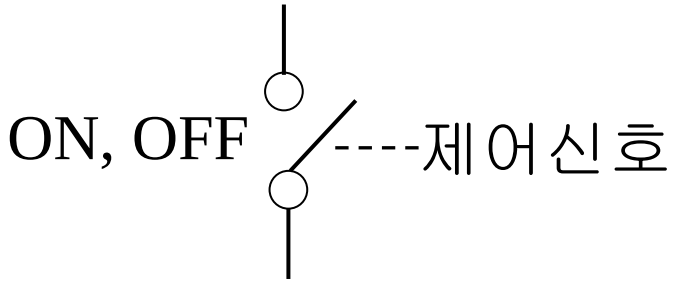
- High = 2^V = "1" = True, Low = 0^V = "0" = False

Automaton (반복)

- Self-operating machine ("sensors" and digital "switches")



3-Terminal Digital Switches
(구현 기술)



- None in mechanical era
- Electromagnetic relay (릴레이)
  - Invented in 1835 (speed: 10^{-3} second)
- Vacuum tube (진공관; speed: 10^{-6} second)
  - Invented in 1906; first commercial use in 1920
  - 라디오, TV, 오디오, 전화설비, ENIAC, …
- Transistor – dream device (speed: 10^{-11} second)
  - Invented in 1947; 실용화에 10년 걸림
  - Small, fast, reliable, energy-efficient, inexpensive
  - Integrated Circuits 형태로 제작 가능

Electro-Mechanical Relay (반복)

- Invented in 1835, switching speed: order of milliseconds

Image of electromagnetic relays:
http://en.wikipedia.org/wiki/File:Relay.jpg
Image of electromagnetic relays:
http://en.wikipedia.org/wiki/File:Relay_symbols.svg

Electron or Vacuum Tube (반복)

- Invented in 1906 (speed: order of microseconds)
- First commercial electron tube by RCA in 1920
  - Radio, TV, Audio, telephone networks, ENIAC

Image of electronic vacuum tubes:
http://en.wikipedia.org/wiki/File:SE-300B-70W.jpg
Image of electronic vacuum tubes:
http://en.wikipedia.org/wiki/File:Triode_tube_schematic.svg

Electronic Switch – Transistor

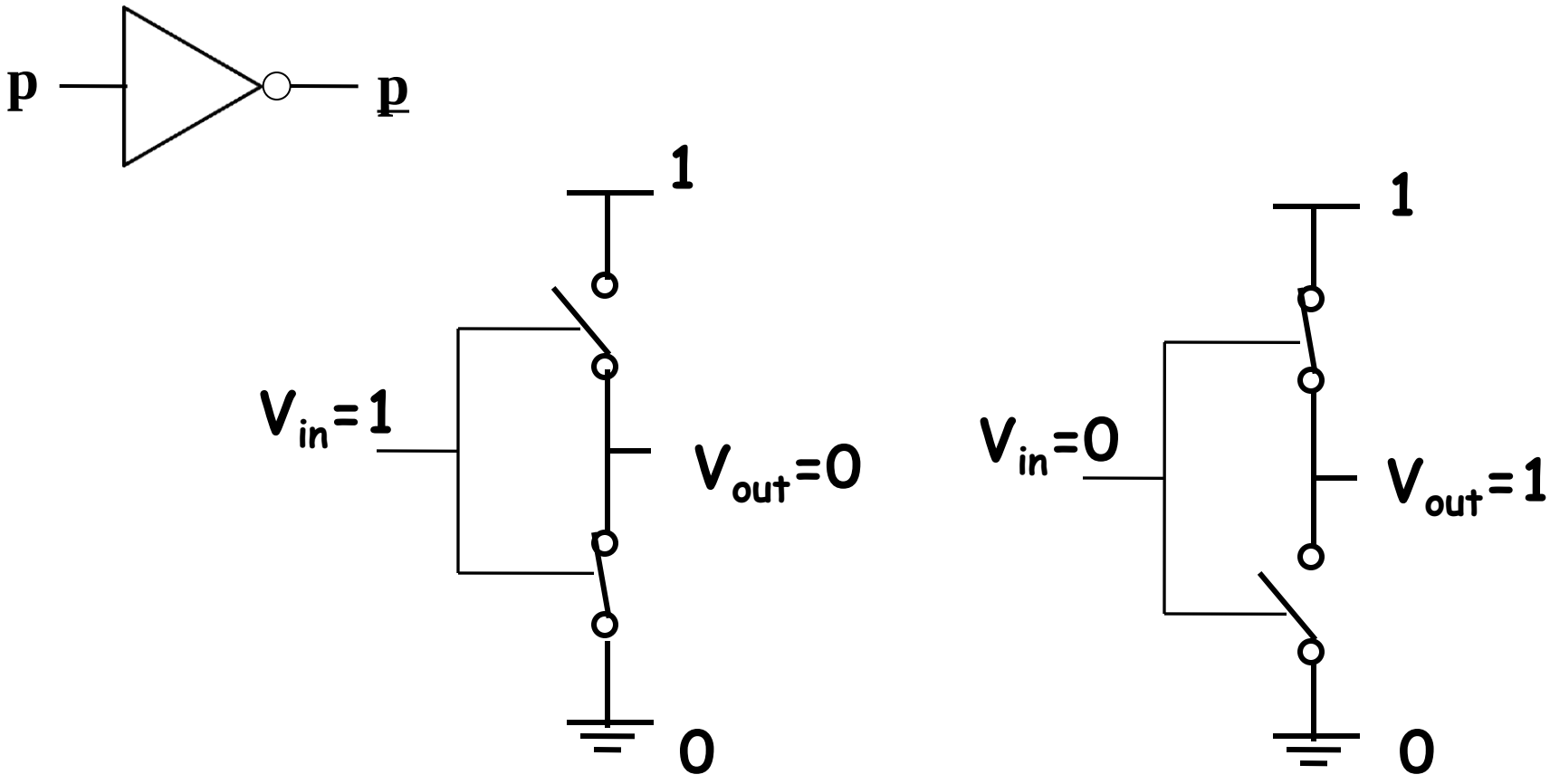
- Solid-state semiconductor device
  - "Transistor" by Bell Labs. in 1947 (c.f., ENIAC in 1946)
  - Integrated circuits in 1958

Image of transistors:
http://en.wikipedia.org/wiki/File:Transistorer_(croped).jpg
Image of ICs (Integrated Circuits):
http://en.wikipedia.org/wiki/File:Microchips.jpg

How to implement AND, OR, NOT

(move to gate-level of abstraction)
(from transistor-level of abstraction)

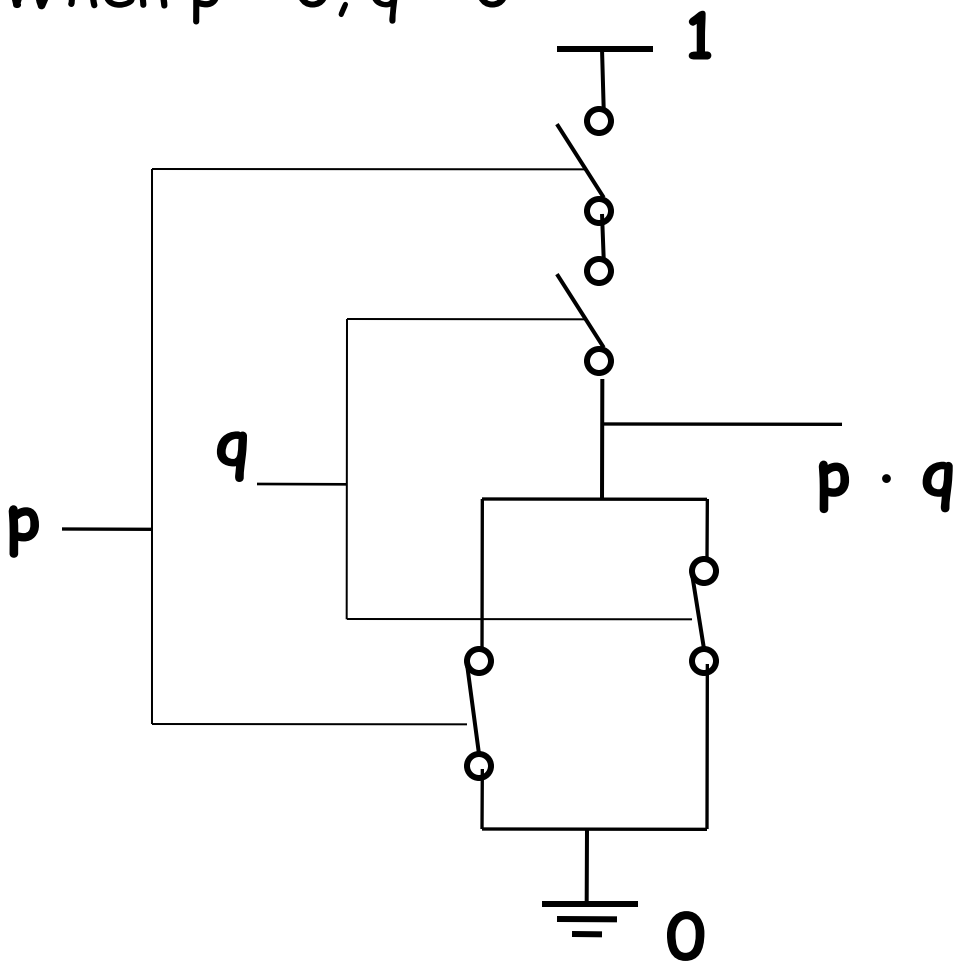
NOT Gate (Inverter)

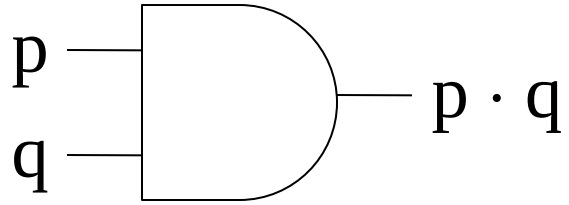


- High = 2^V = "1" = True, Low = 0^V = "0" = False

AND Gate

- High = 2^V = "1" = True, Low = 0^V = "0" = False
- When p = 0, q = 0

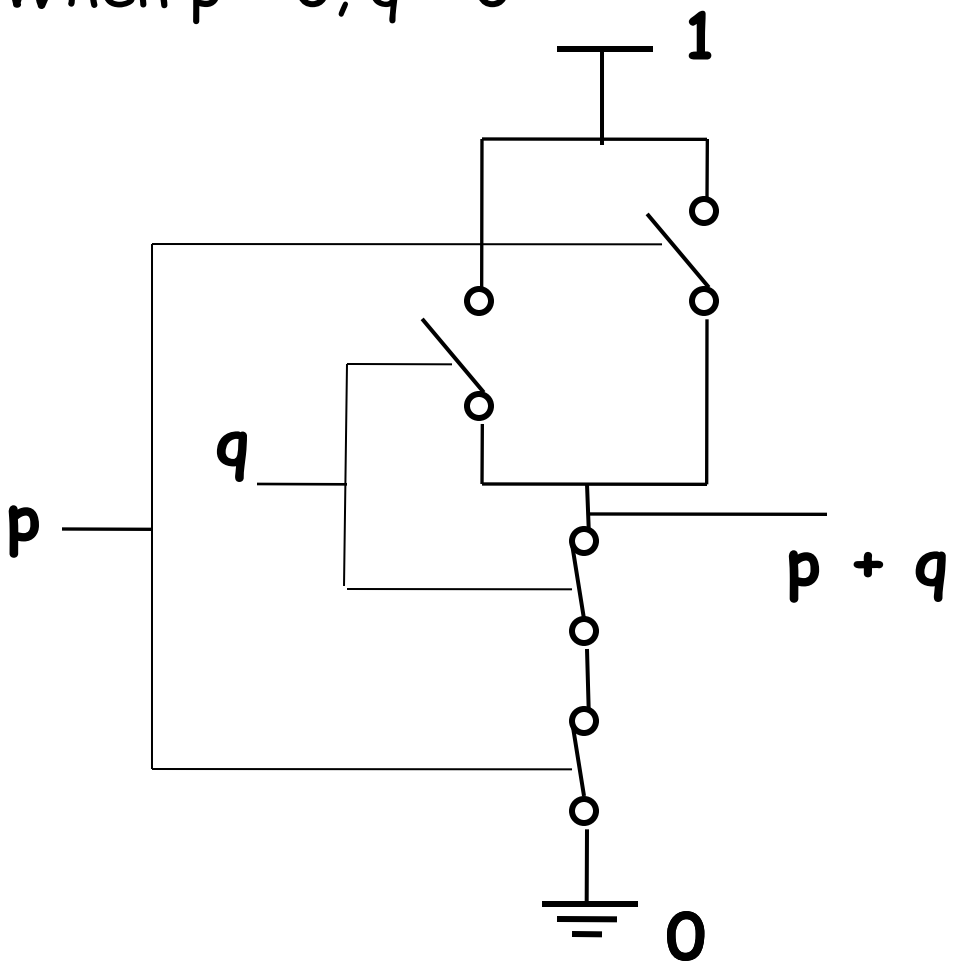


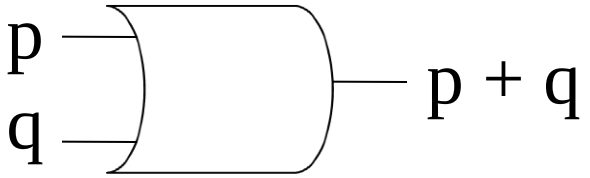


p	q	p · q
1	1	1
1	0	0
0	1	0
0	0	0

OR Gate

- High = = 2^V = "1" = True, Low = 0^V = "0" = False
- When p = 0, q = 0





p	q	p + q
1	1	1
1	0	1
0	1	1
0	0	0

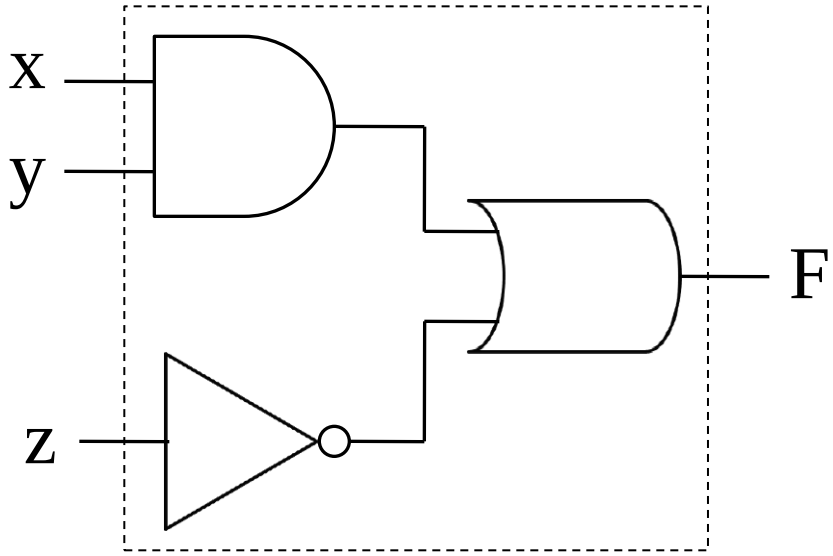
Gate-Level of Abstraction (반복)

- Real world example
  - x: 댐의수위가 높다, y: 비가 온다, z: 논에 물이 충분하다
  - F: 댐의 수문을 연다

x	y	z	F
1	1	1	1
1	1	0	1
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	1
0	0	1	0
0	0	0	1

Truth Table

- $F = x \cdot y + \underline{z}$
  - Simplest form?



Logic Diagram

Hardware Abstraction (반복)

- You have AND, OR, NOT
- What did you learn from "디지털논리설계" 교과목?
  - Designing useful functional units
    - Combinational circuits: decoder, adder, (ALU)
    - Sequential circuits: register, counter, memory, (CPU)
  - What is more important: notion of abstraction
    - Use AND, OR, NOT, to design simple functional unit (e.g., decoder)
      - You know only its interface (i.e., how to use it)
    - Then use it to build more complex functional unit

† This cycle is repeated indefinitely

Is transistor a simple thing?

(abstraction of a complex thing)

CMOS NAND Gate

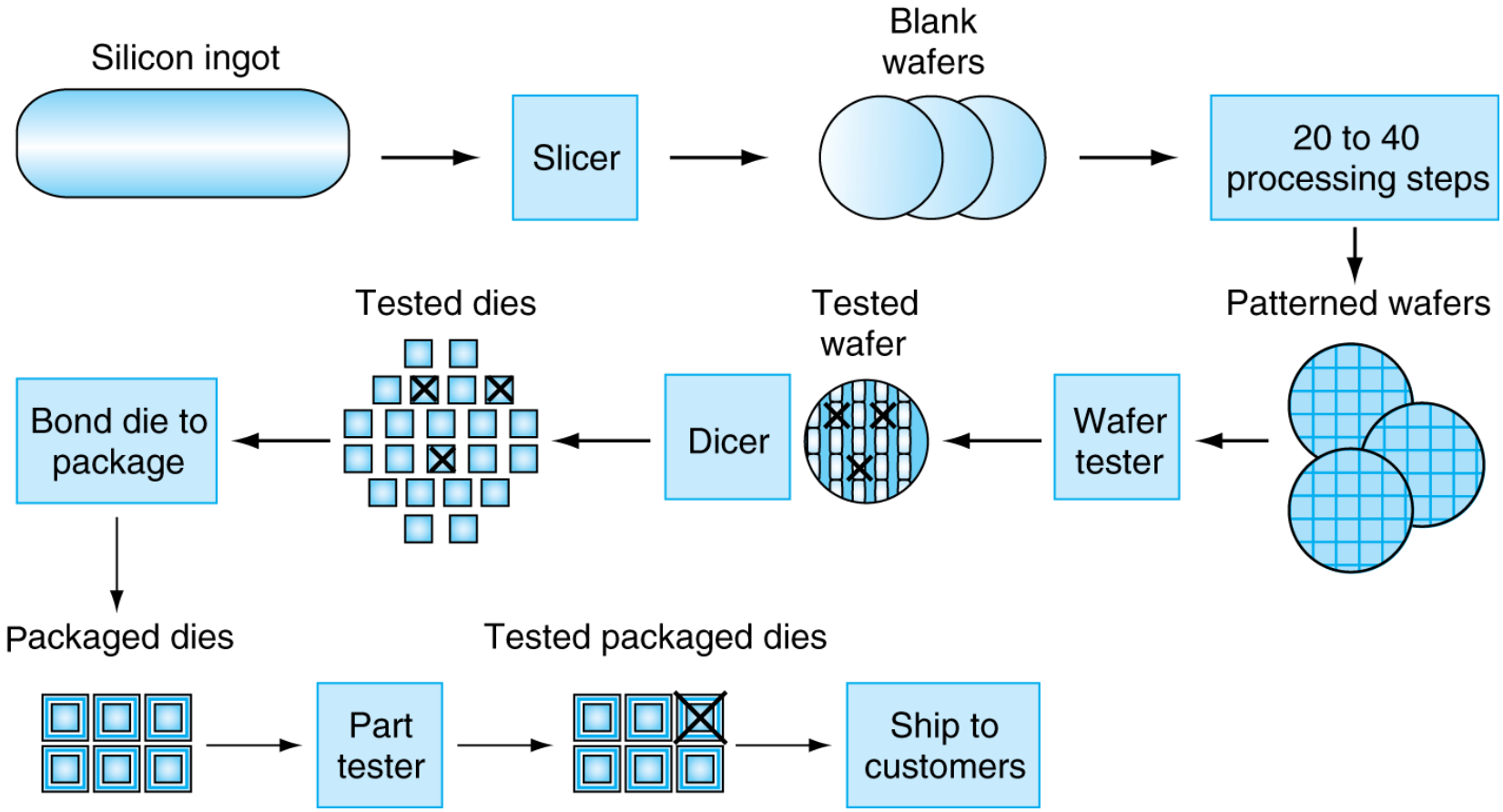
Image of CMOS NAND gate:
http://en.wikipedia.org/wiki/File:CMOS_NAND.svg

Image of CMOS NAND layout:
http://en.wikipedia.org/wiki/File:CMOS_NAND_Layout.svg

Image of CMOS transistor pair:
http://en.wikipedia.org/wiki/File:Cmos_impurity_profile.PNG

Manufacturing ICs

(Hennessy and Patterson slide, Computer Organization and Design, Morgan Kaufmann)

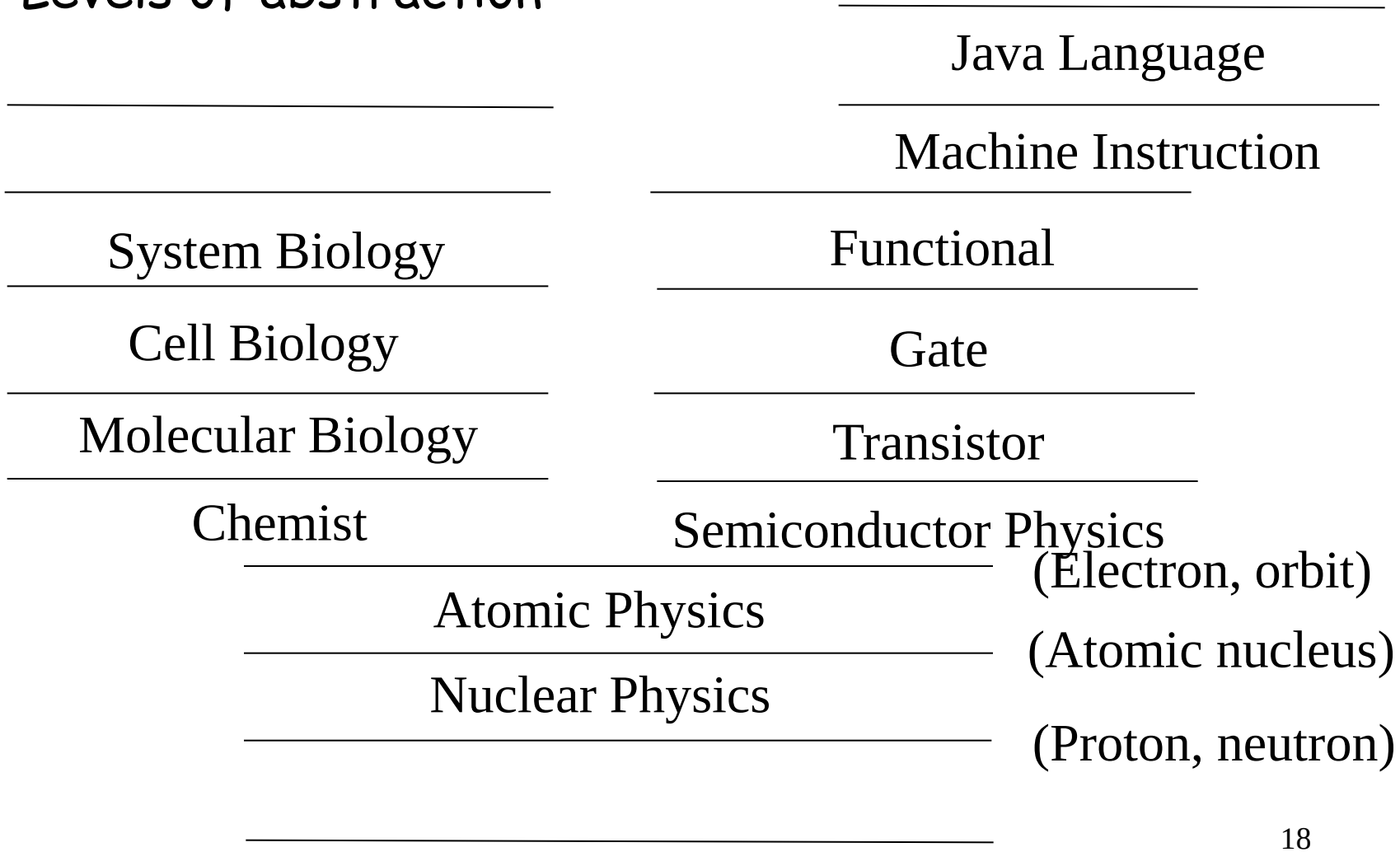


- Yield: proportion of working dies per wafer

Science and Engineering

(GEB by Hofstadter, AI by Winston)

- Levels of abstraction



Abstraction in Software

C (or High-Level) Programming

- Basic building block
  - Statement (like sentence in human writing)
    - "atoms" (that have meanings)
- Compiler translate statements into CPU instructions

Statements

- Compiler support variety of statements for programmers
  - Variable declaration statement

```
int  a, b, c, d, i, j=0;      // colored: special symbols
float  x, y, z=3.5;           // statement end with ;
```

  - Assignment statement

```
a = 3;
```

  - Arithmetic and assignment statement

```
a = (b*3) – (c/d);
```

  - Conditional statement

```
if (i > 0)  x = x*1.1;            // if statement
if (i > 0)  x = x*1.1;            // if-else statement
   else  x = x*0.9;                  (indentation)
```

Statements

- Loop statement

```
a = 0;                                  // summation
for (i = 1; i < 5; i = i + 1)
    a = a + i;
```

- Compound statement

```
{ multiple statements }        // treat as single
```

- Function call statement

```
printf("hello, world!\n");      // call OS service
```

-
-
-

C Programming

- We have statements
  - Can write algebraic equation
  - Have English-like control structure
    - Can forget about hardware-level details
- Are we ready to handle million lines of source code?
  - Need design paradigm to reduce complexity
    - We need function

Small C Program – Function

(from The C Programming Language by Kernighan and Ritchie)

```
#include <stdio.h>
int power(int, int);                /* function declaration */
main()                              /* test power function */
{   int i;
    for (i = 0; i < 10; i++)
        printf("%d %d \n", i, power (2,i));  // function call statement
    return 0;
}
int power (int base, int n)   /* power: raise base to n-th power */
{   int i, p;
    p = 1;
    for (i = 1; i <= n; ++i)
        p = p * base;
    return p;
}
```

interface

implementation

Function: Key Abstraction

- Why define functions?
  - Write once, call many times (from different locations)
- Once we define function, all users need to know is
  - Function interface: "int power (int, int)"

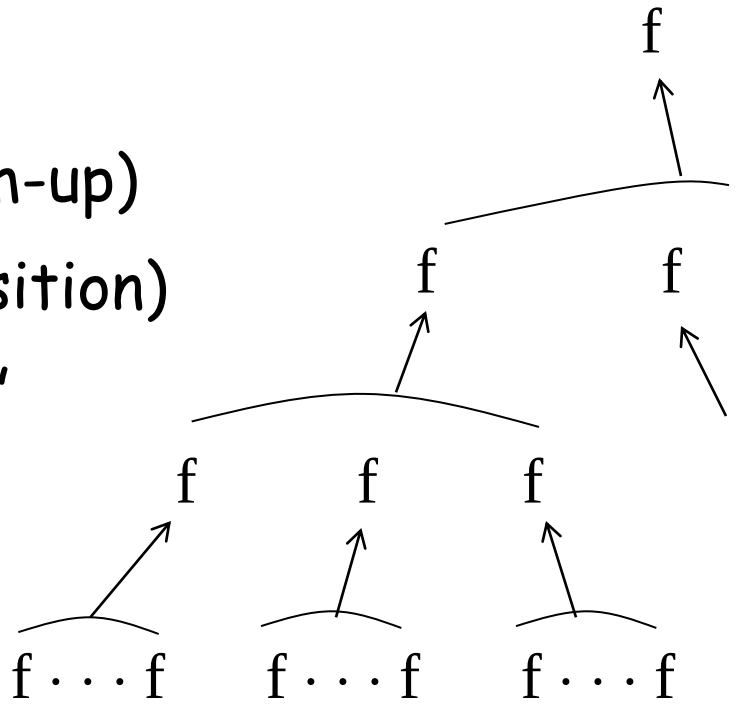
    a = power (2, 3);
  - Don't have to know about implementation
- Function "int power (int, int)": single abstract operation
  - Function become like statement
    - † Function has name, naming is abstracting
- Function abstraction
  - Critical to deal with program largeness and complexity

Hierarchical Function Abstraction

- Hierarchical bottom-up function abstraction
  - Critical to deal with complexity
- Notion of program structure
- Design perspective
  - Top-down (rather than bottom-up)
  - Modular design (i.e., decomposition)
  - Keep "dividing and conquering"

high level abstraction 개념 o vs machine abstraction 개념 x

library - 관련된 function 모임

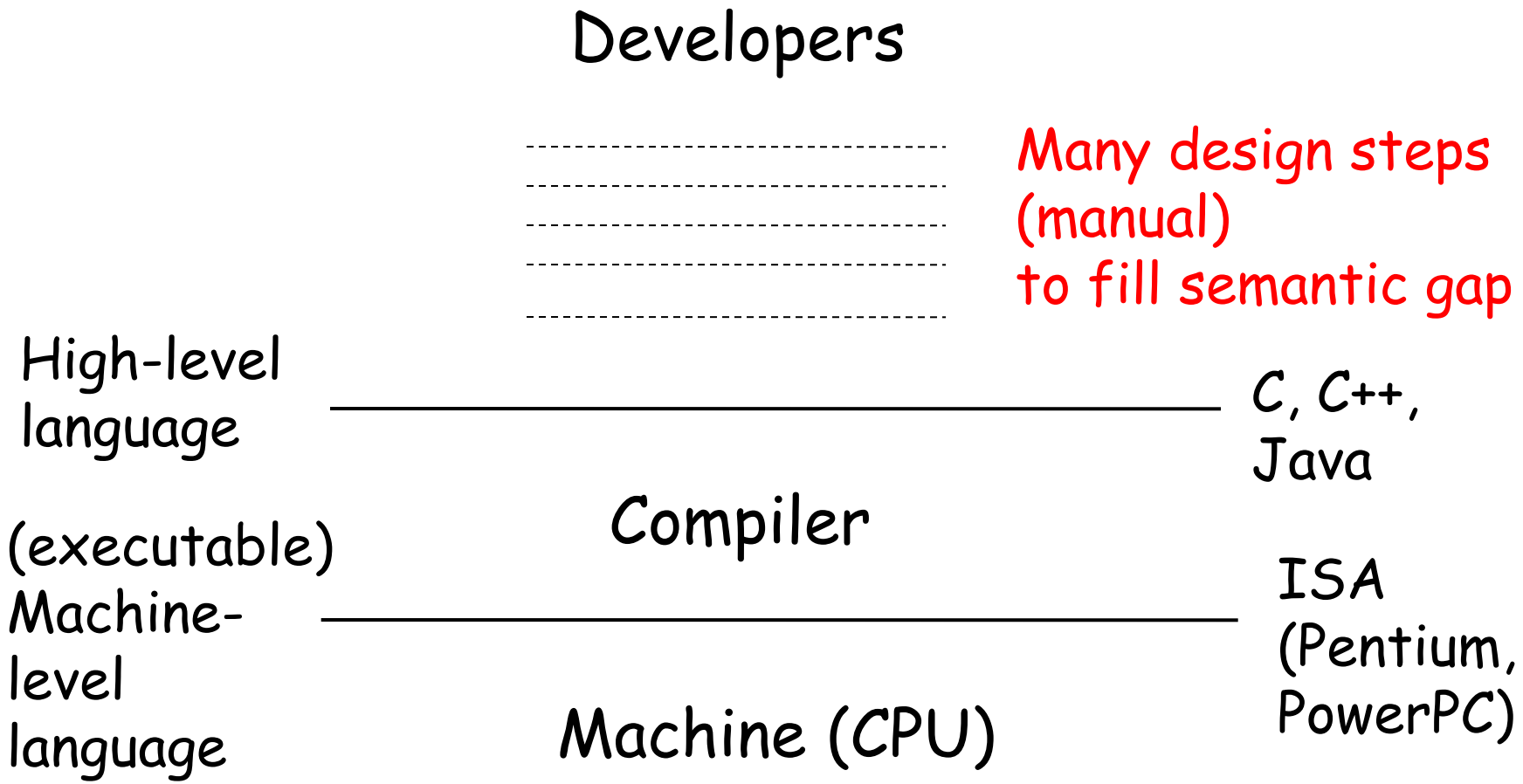


Primitive-Composition-Abstraction

공학의 진리

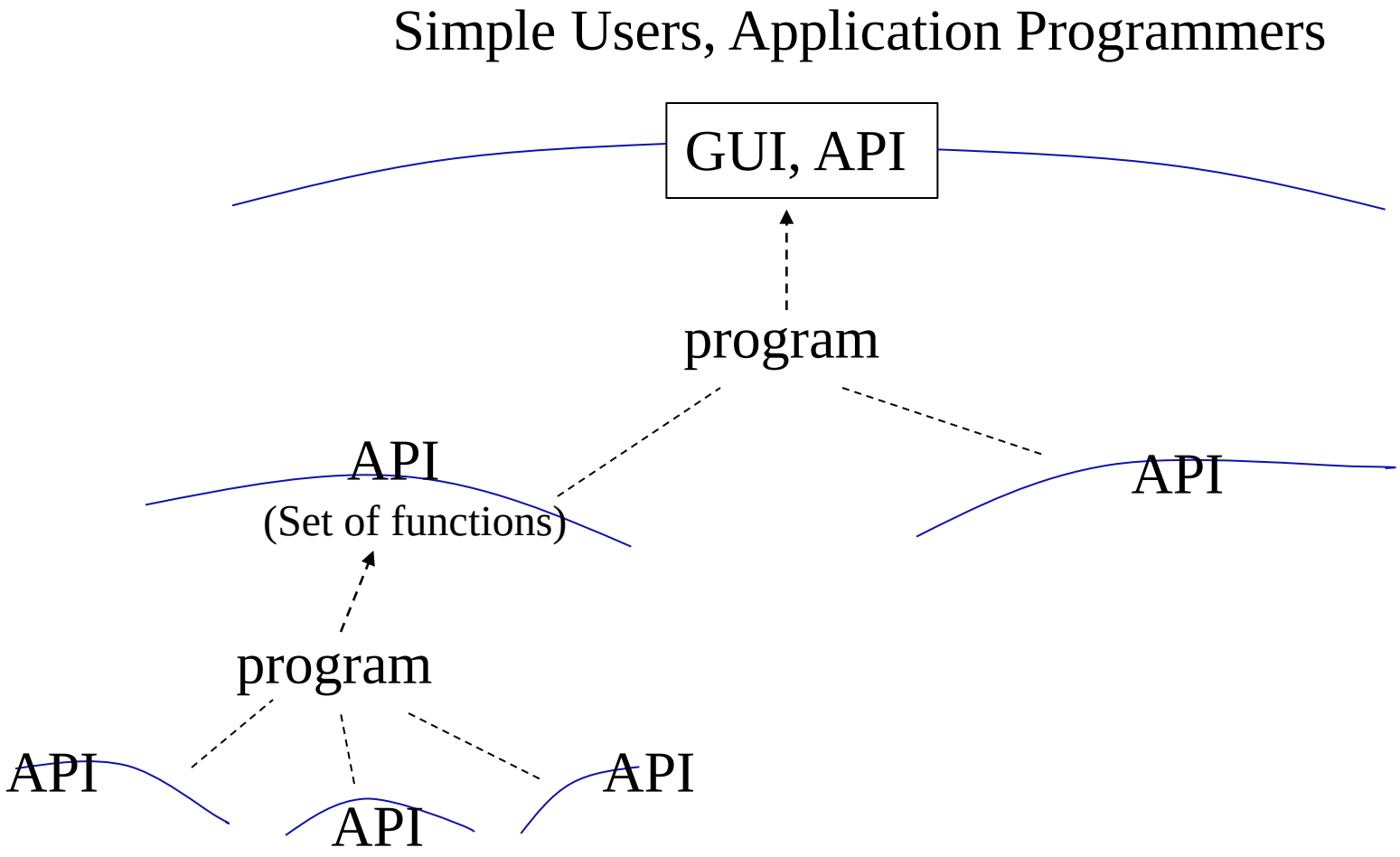
- Fundamental paradigm in high-level programming
  - Primitives: statements
  - Composition: build function using statements
  - Abstraction
    - Given its interface, can use function
    - Function (an abstract statement) become primitive
- Function: abstraction building mechanism
- What is high-level programming?
  - Hierarchically build abstraction
    - † True in all engineering

Million Lines of Source Code



Software Design (e.g., OS)

- Hierarchy: buy library (or API), build on top of it



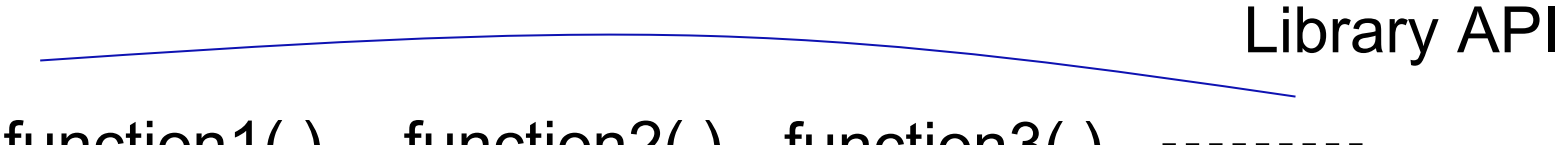
What is library?

- Collection of related functions (e.g., math library)
  - API (Application Programming Interface):

    int power(int, int);, double sin(double);, ….
  - Compiled code: power.o + sin.o + …

Programmer:

- Build more complex function using library functions
- Don't have to know how library functions are implemented



Software Architecture

- Software architecture (or program structure)
- What is it?

의미 : set of key boundaries

ISA(instruction set architecture)
하드웨어 / 소프트웨어
ISA는 두 개의 key boundary를 다루기 때문에 architecture이다.

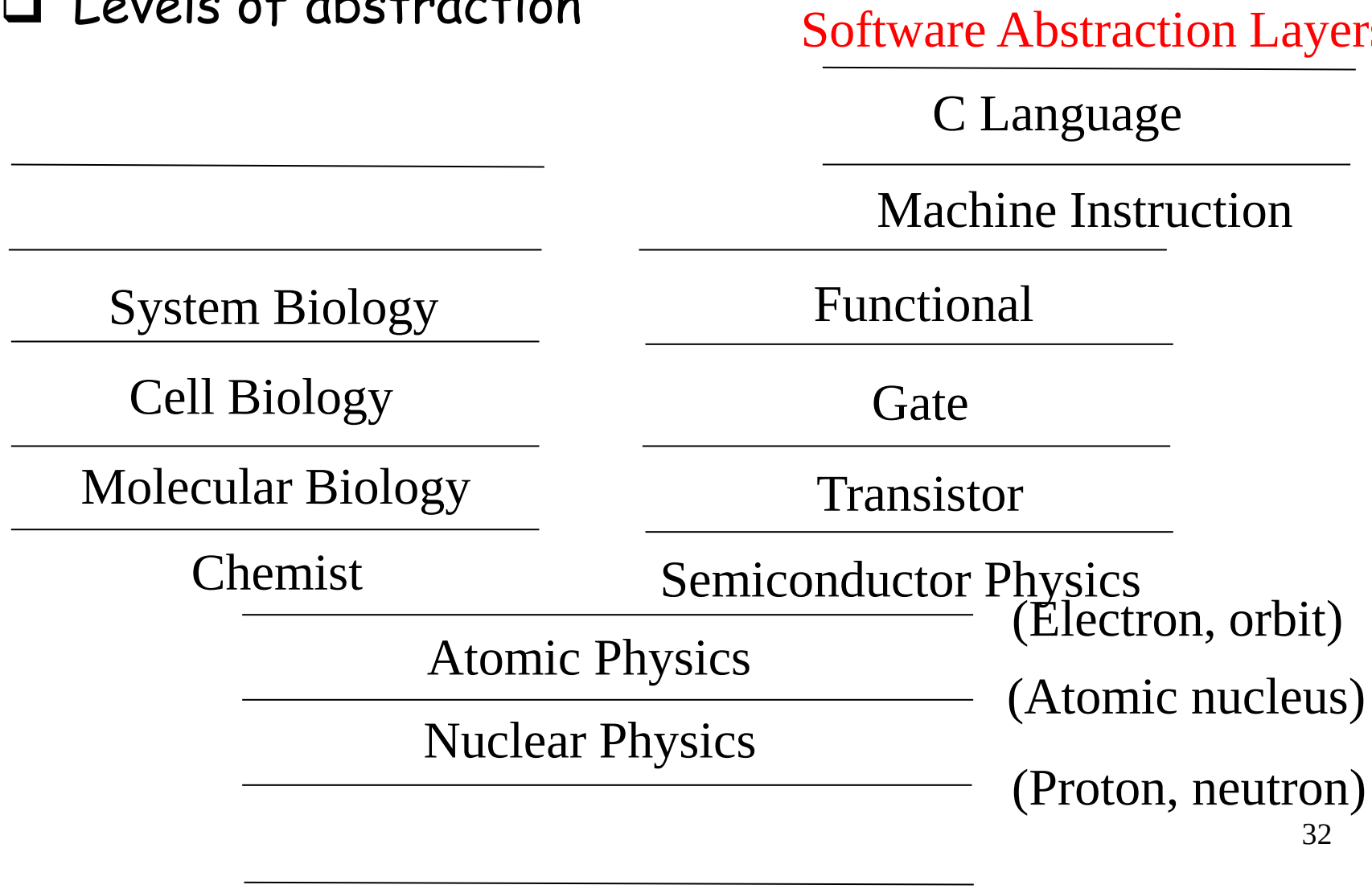
  - Set of key boundaries
    - Identification of modules
    - Their interface
  - Hierarchical: all the way down to lowest library
- To think about
  - Architect vs. programmer
    - "대한민국에는 소프트웨어 아키텍트가 없다"?

† The same applies to hardware or any engineering area

Science and Engineering

(GEB by Hofstadter, AI by Winston)

- Levels of abstraction



Abstractions Provided by Languages

- C language
  - Statements
  - Abstraction builder
    - Function for processing
    - Array/structure for data
- Limitation of C
  - Separate abstraction of data and processing
- OOP
  - Object-based abstraction
  - Object contain both processing and data

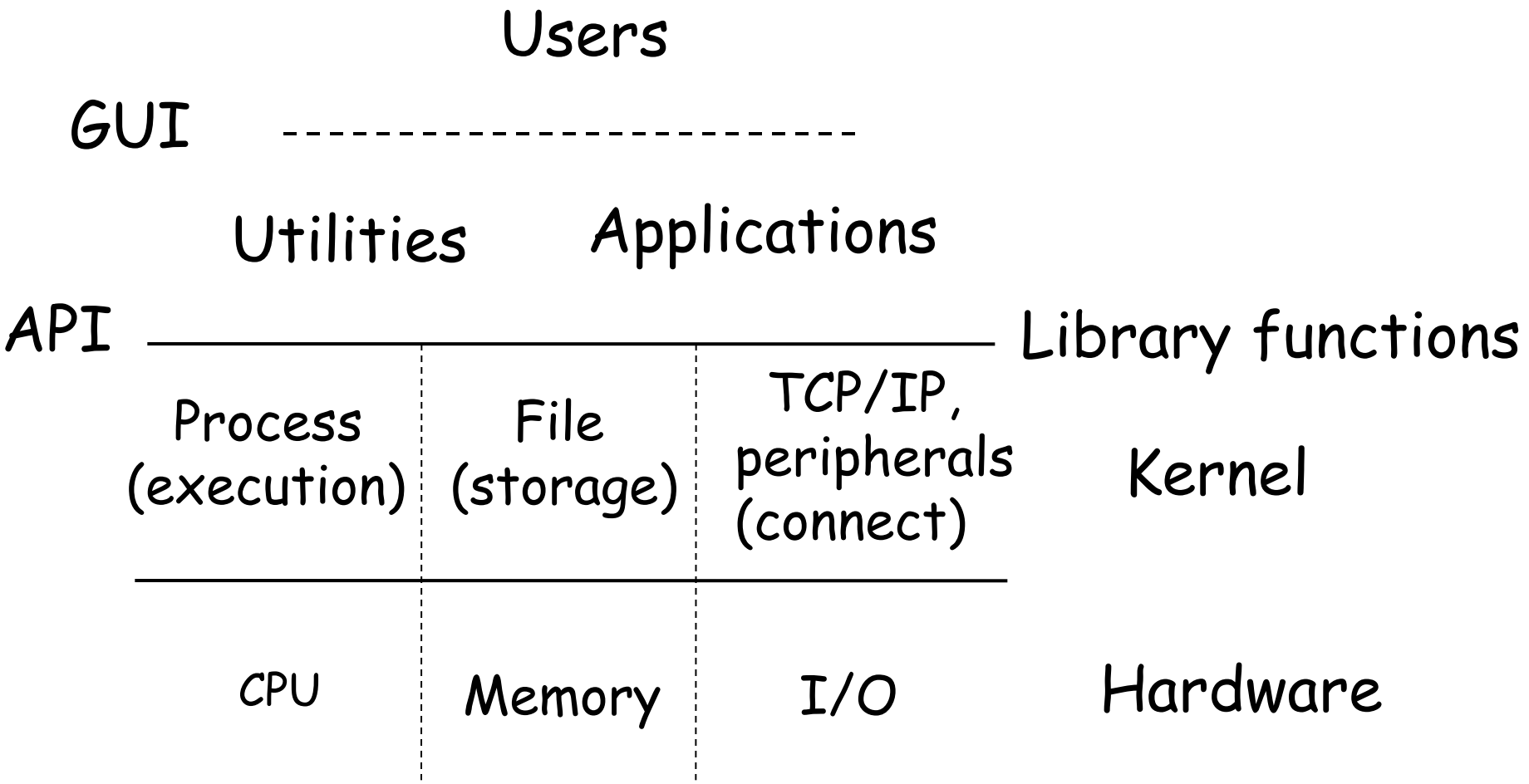
CSE Major: Abstraction Builder

- Problem solving create new abstraction (or service)
  - Web, search engine, SNS, Kakao talk
  - RISC-style ISA in 1980s

OS Abstraction

(이 부분은 참고자료임)
- 어떤 구체적인 문제들을 풀었나?
- 어떤 solution 들을 만들어 내었나?

What is OS?


Users
GUI
Utilities
Applications
API
Library functions
Process (execution)
File (storage)
TCP/IP, peripherals (connect)
Kernel
CPU
Memory
I/O
Hardware

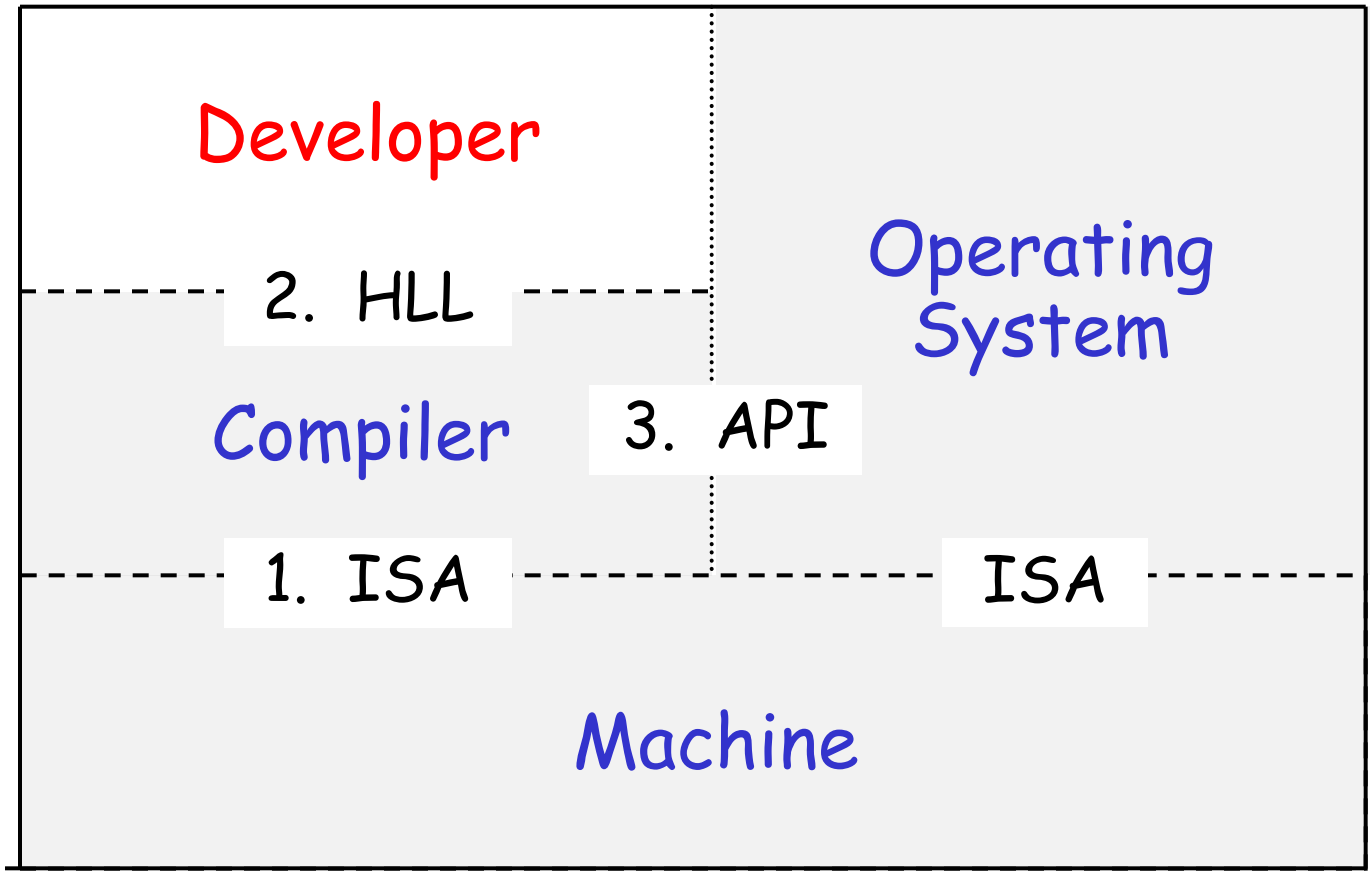
What is OS?

- Make hardware easy to use by providing library
  - CPU (program execution)
    - process_create(), process_kill(), …
  - Memory (storage)
    - file_copy(), delete_folder(), file_rename(), …
  - I/O (connectivity)
    - Socket("naver.com", 80), monitor_write(), …
- GUI, utilities (common functions for all users)
- 공유자원의 사용관리 및 보호 (응용 프로그램간의 조정)

† Programmer: OS API 이용하여 Applications 개발

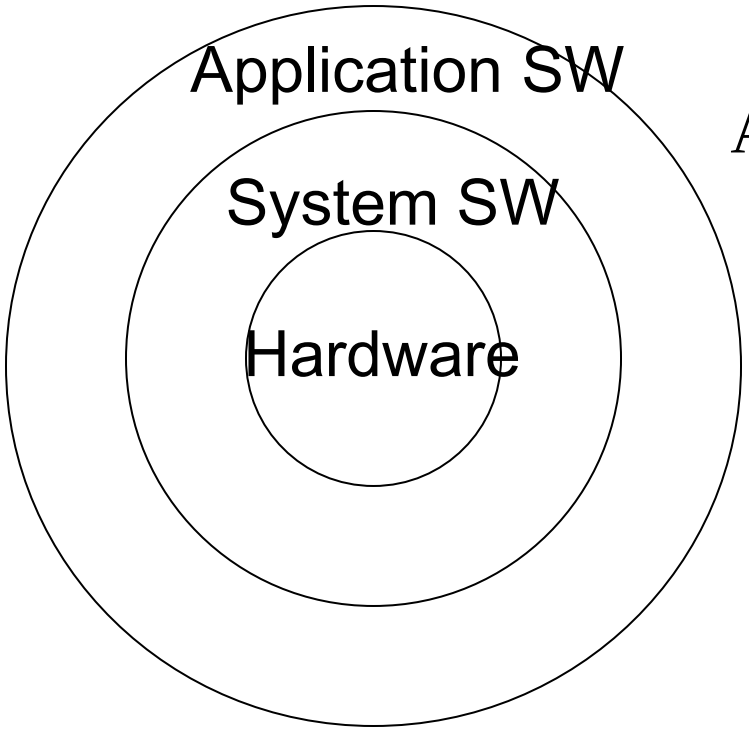
Using OS API: Third Major I/F

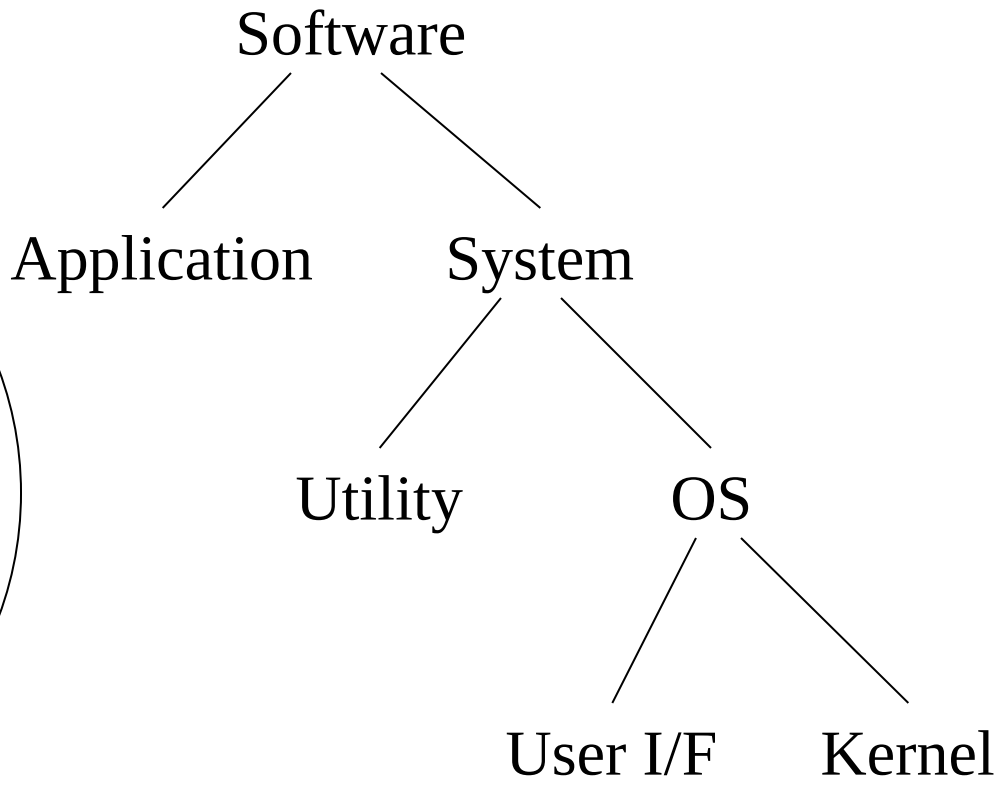
- Three interfaces, three key products and their services



What is OS?

- Map to previous figure





Summary

- How do we build complex processor?
  - Abstractions in hardware design
- Abstractions in software design
- OS abstractions