CPUs, GPUs, and the Rise of Software Parallelism

Joel Svensson's PhD Thesis Defense
Göteborg, Sweden

Prof. Stephen A. Edwards

Columbia University

December 16, 2013

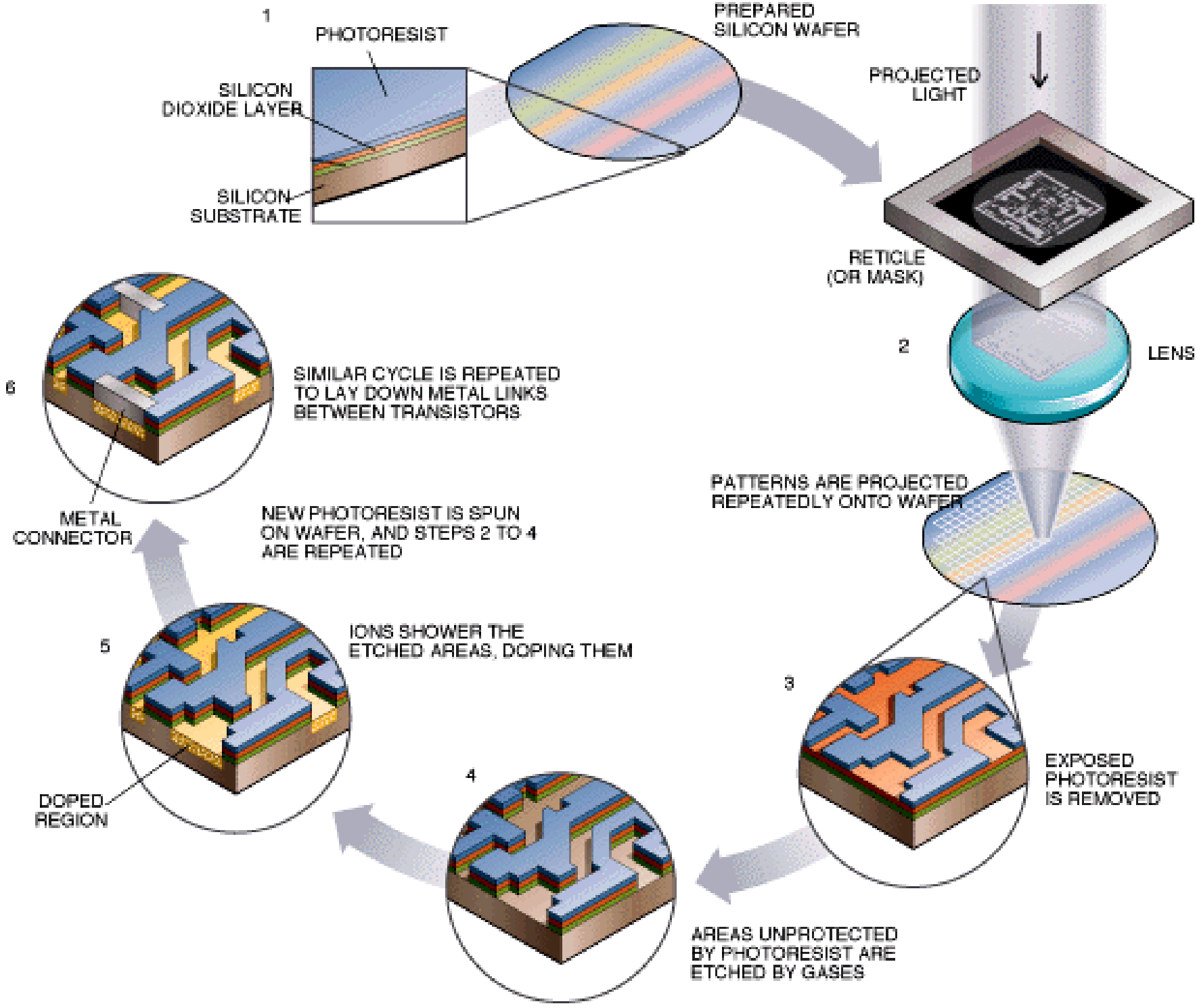
Sand and Silicon

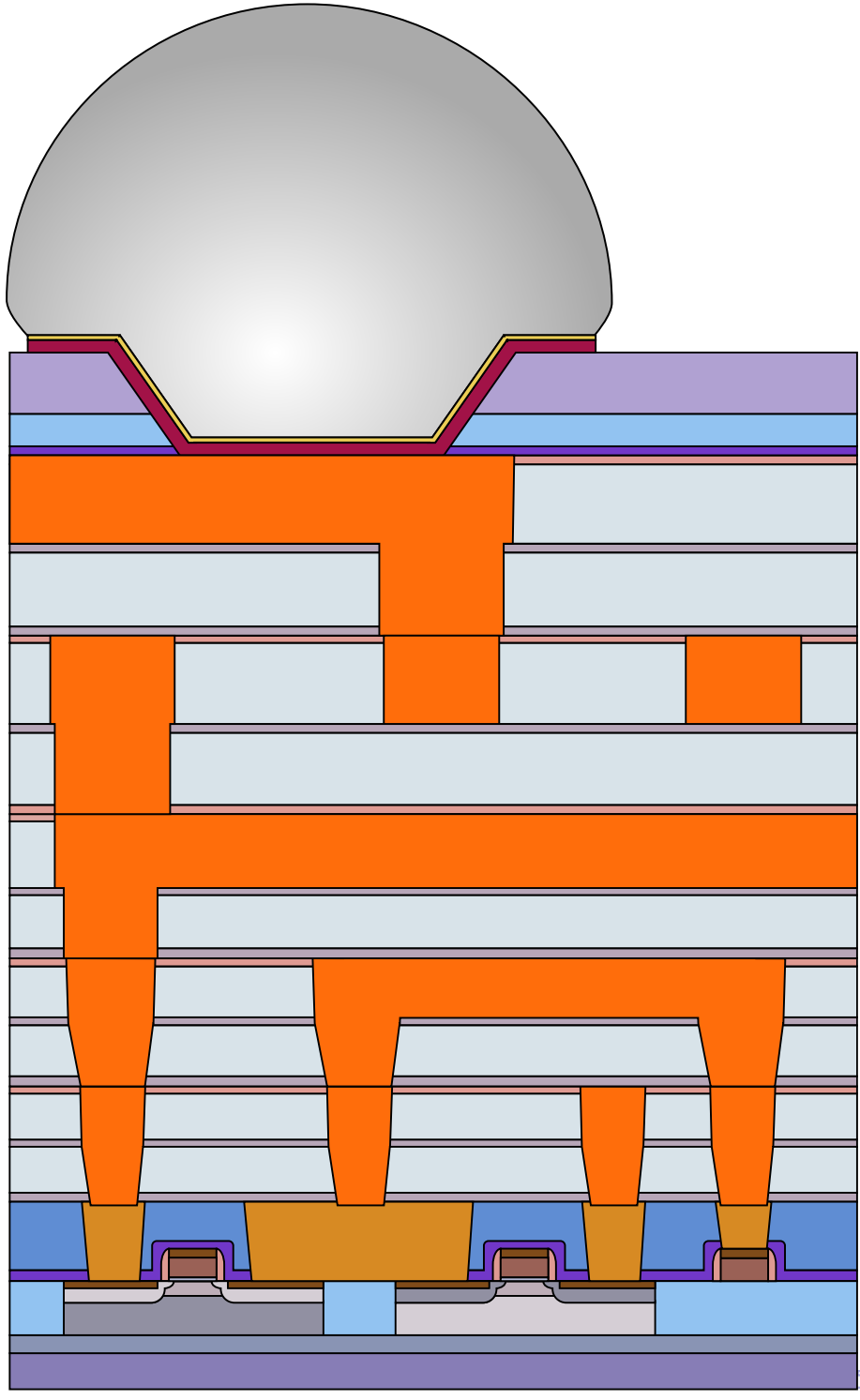
Silica a.k.a. SiO_2 a.k.a. Quartz

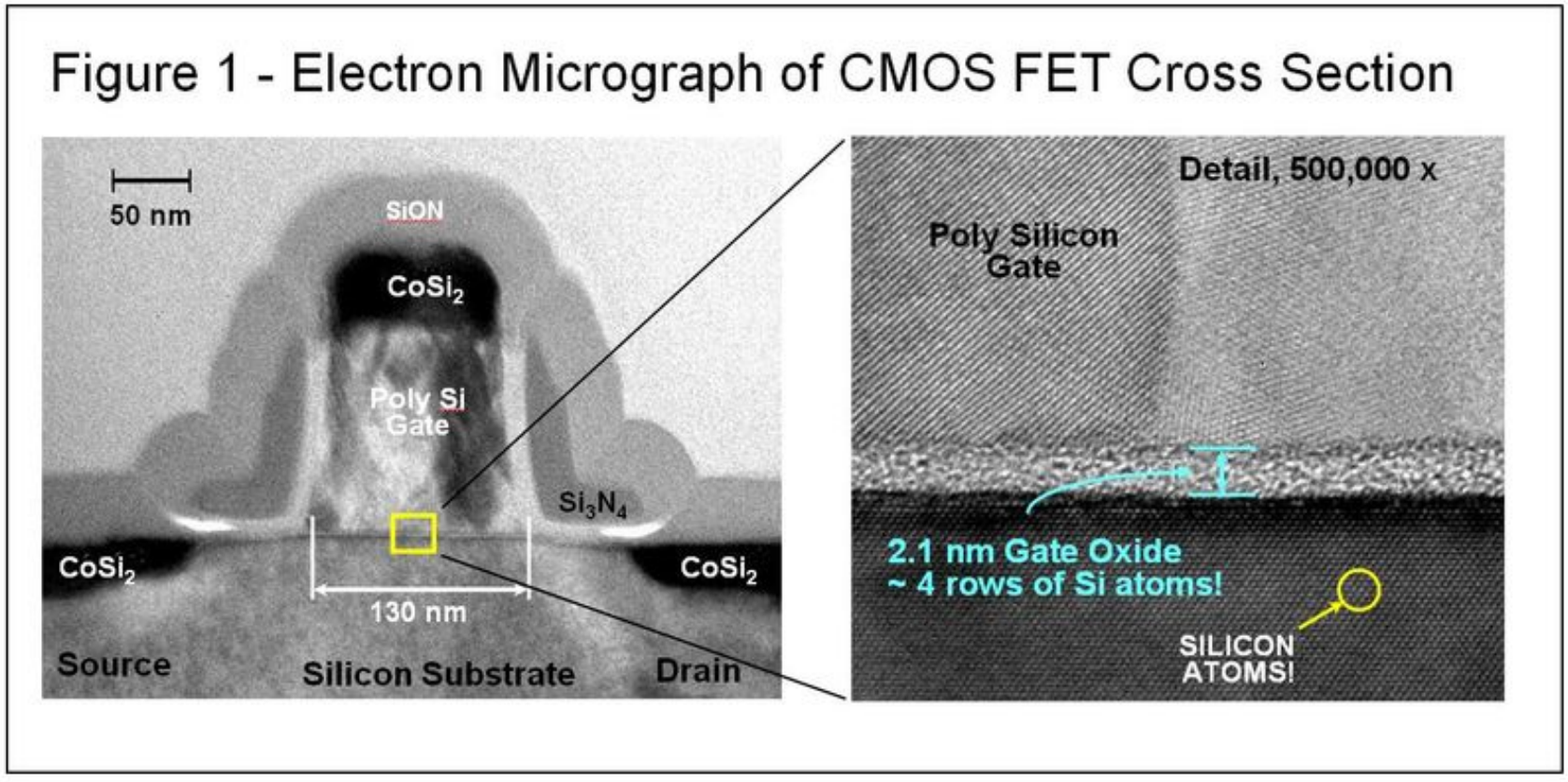
$$SiO_2 + 2\,C \rightarrow Si + 2\,CO$$

Elemental, amorphous silicon

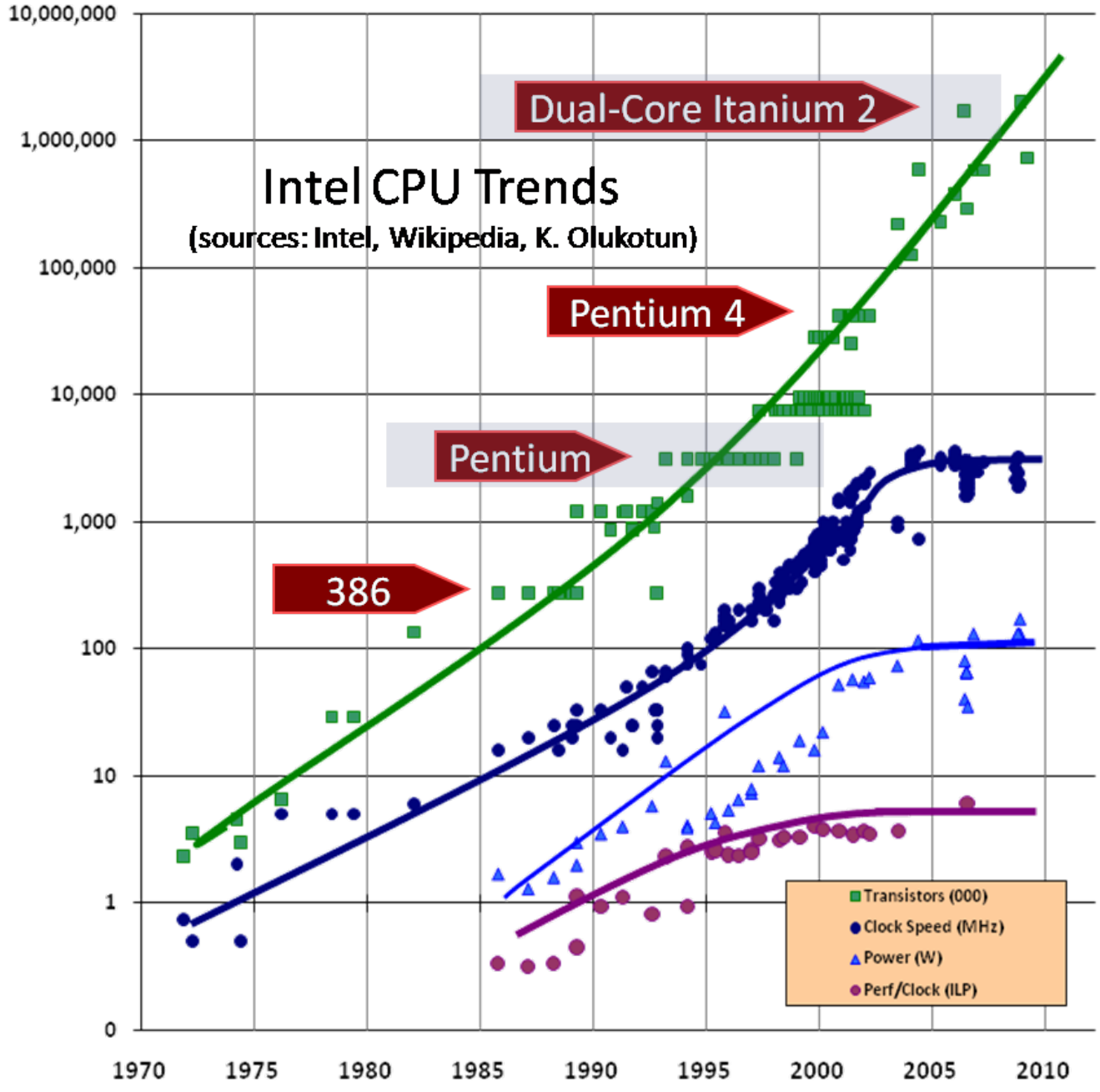
Monocrystalline Silicon Ingot


1
PHOTORESIST
SILICON DIOXIDE LAYER
SILICON SUBSTRATE
PREPARED SILICON WAFER
PROJECTED LIGHT
RETICLE (OR MASK)
2
LENS
PATTERNS ARE PROJECTED REPEATEDLY ONTO WAFER
3
EXPOSED PHOTORESIST IS REMOVED
4
AREAS UNPROTECTED BY PHOTORESIST ARE ETCHED BY GASES
5
IONS SHOWER THE ETCHED AREAS, DOPING THEM
DOPED REGION
NEW PHOTORESIST IS SPUN ON WAFER, AND STEPS 2 TO 4 ARE REPEATED
6
SIMILAR CYCLE IS REPEATED TO LAY DOWN METAL LINKS BETWEEN TRANSISTORS
METAL CONNECTOR

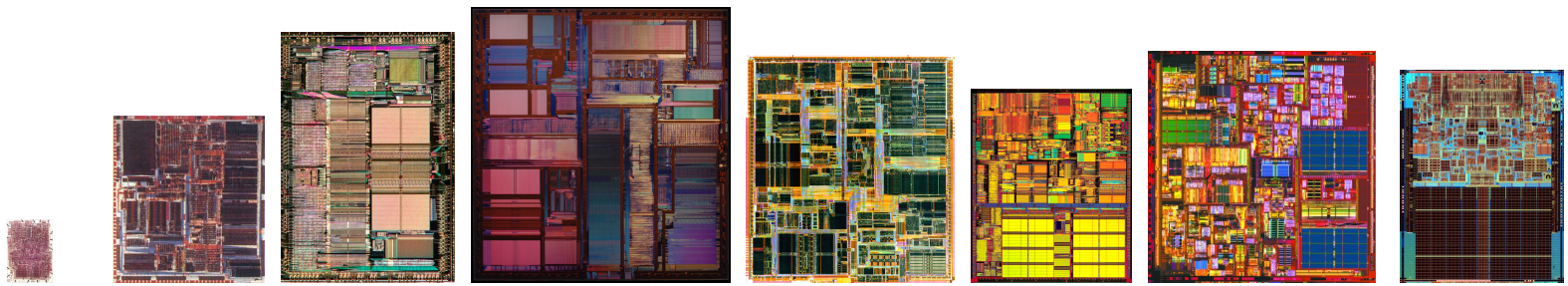




Source: The Energy Zarr Blog

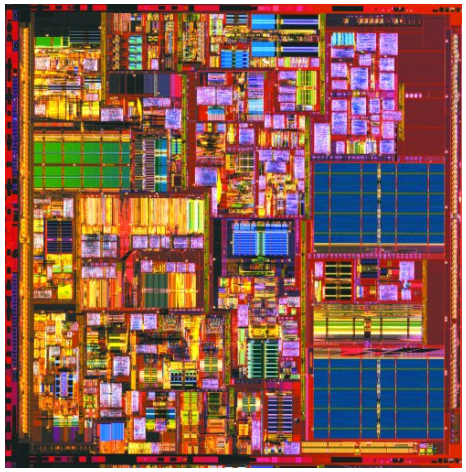

Intel CPU Trends
(sources: Intel, Wikipedia, K. Olukotun)
Dual-Core Itanium 2
Pentium 4
Pentium
386
Transistors (000)
Clock Speed (MHz)
Power (W)
Perf/Clock (ILP)
10,000,000
1,000,000
100,000
10,000
1,000
100
10
1
0
1970
1975
1980
1985
1990
1995
2000
2005
2010

Intel Processors to Scale

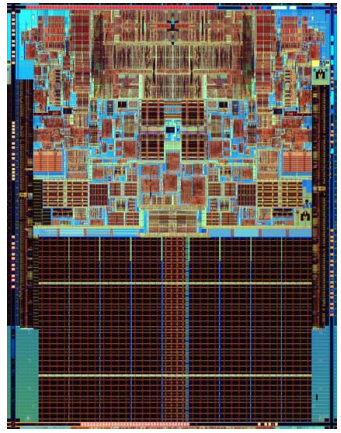


4004 1971 | ’386 1987 | ’486 1989 | P I 1993 | P II 1997 | P III 1999 | P IV 2000 | Core 2 2006

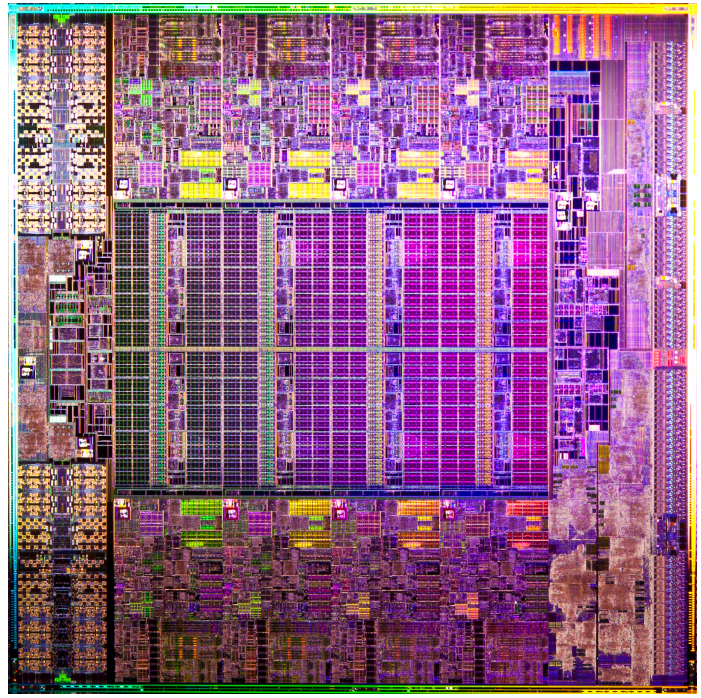
What Happened in 2005?



Pentium 4
2000
1 core
Transistors: 42 M

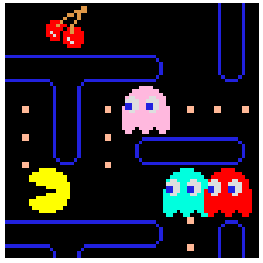


Core 2 Duo
2006
2 cores
291 M



Xeon E5
2012
8 cores
2.3 G

Meanwhile, in the Graphics World...



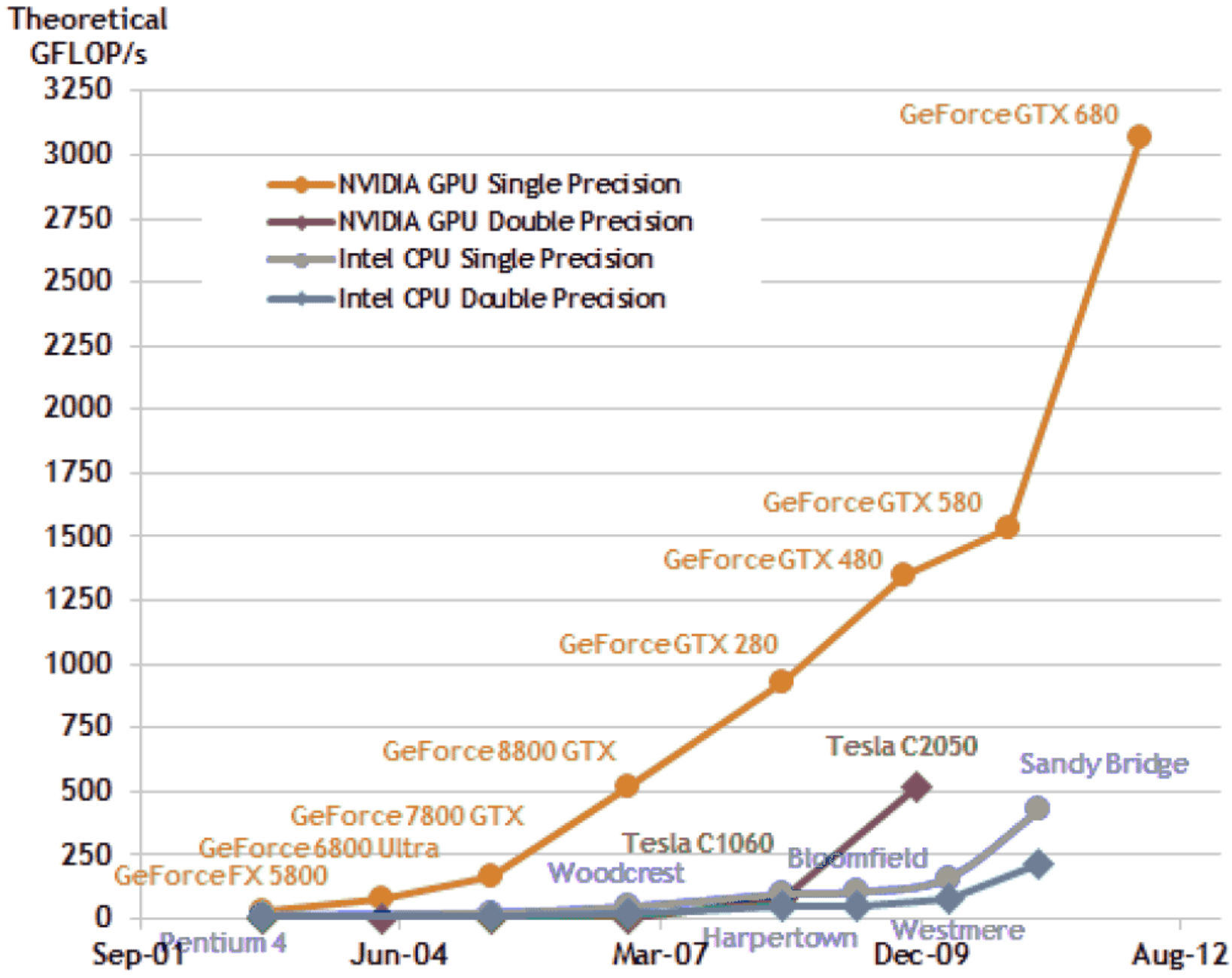
1980: Pac-Man

5 sprites

1988: Sega Genesis
Sonic the Hedgehog
20 sprites

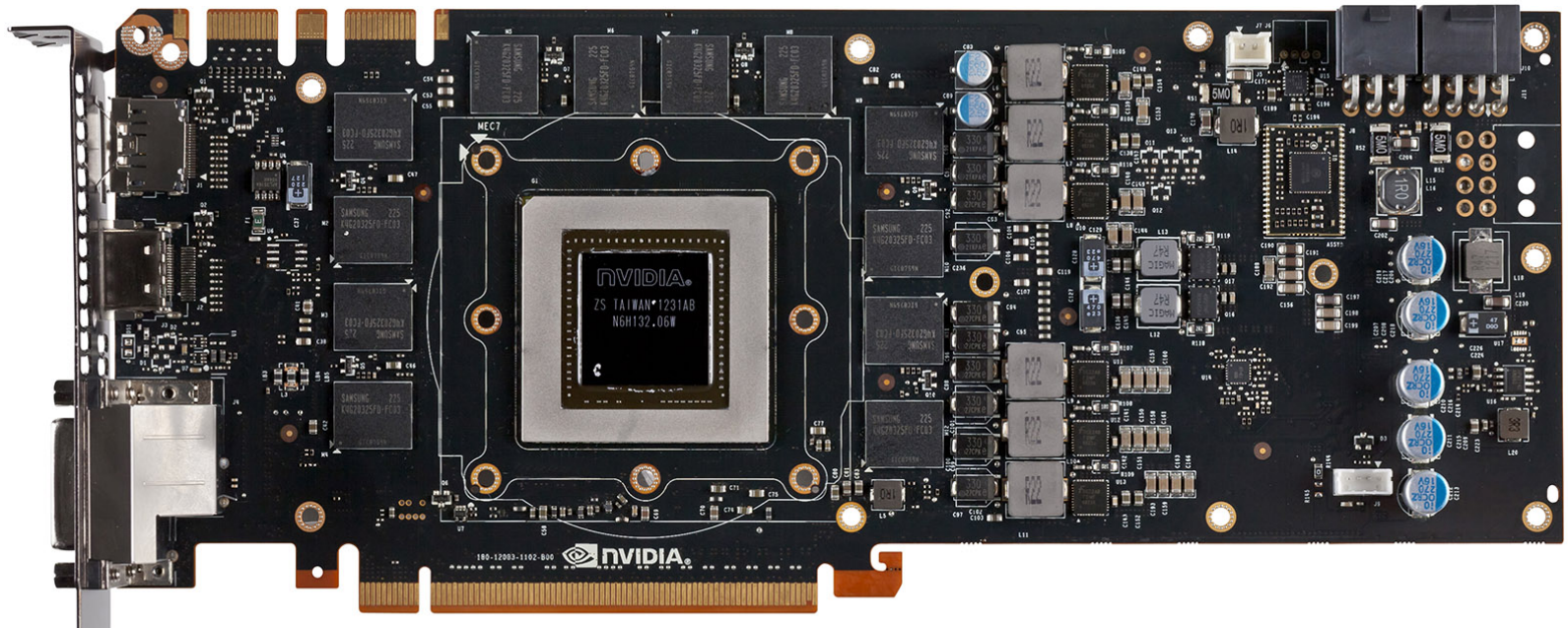
2000: PlayStation 2
Final Fantasy X
2 VPUs
66.2 GFLOP/s

2006: PlayStation 3
Gran Turismo 5
24 pixel shaders
176 GFLOP/s


Theoretical GFLOP/s
3250
3000
2750
2500
2250
2000
1750
1500
1250
1000
750
500
250
0
NVIDIA GPU Single Precision
NVIDIA GPU Double Precision
Intel CPU Single Precision
Intel CPU Double Precision
GeForce GTX 680
GeForce GTX 580
GeForce GTX 480
GeForce GTX 280
GeForce 8800 GTX
GeForce 7800 GTX
GeForce 6800 Ultra
GeForce FX 5800
Tesla C2050
Tesla C1060
Sandy Bridge
Bloomfield
Woodcrest
Harpertown
Westmere
Pentium 4
Sep-01
Jun-04
Mar-07
Dec-09
Aug-12

The NVIDIA GTX Titan/GK110 GPU

The NVIDIA GTX Titan/GK110 GPU



The NVIDIA GTX Titan/GK110 GPU

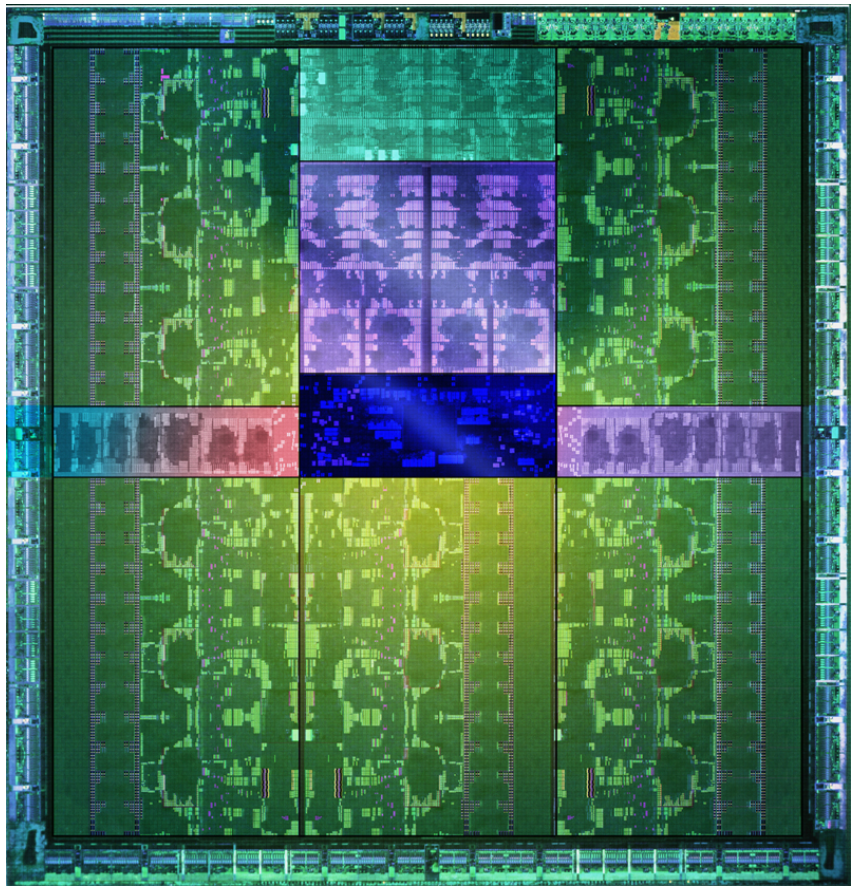
Speed	4.5	TFLOP/s
Frequency	876	MHz
Power	250	W
Transistors	7	G
Area	561	mm^2
Cores	2688	

Memory

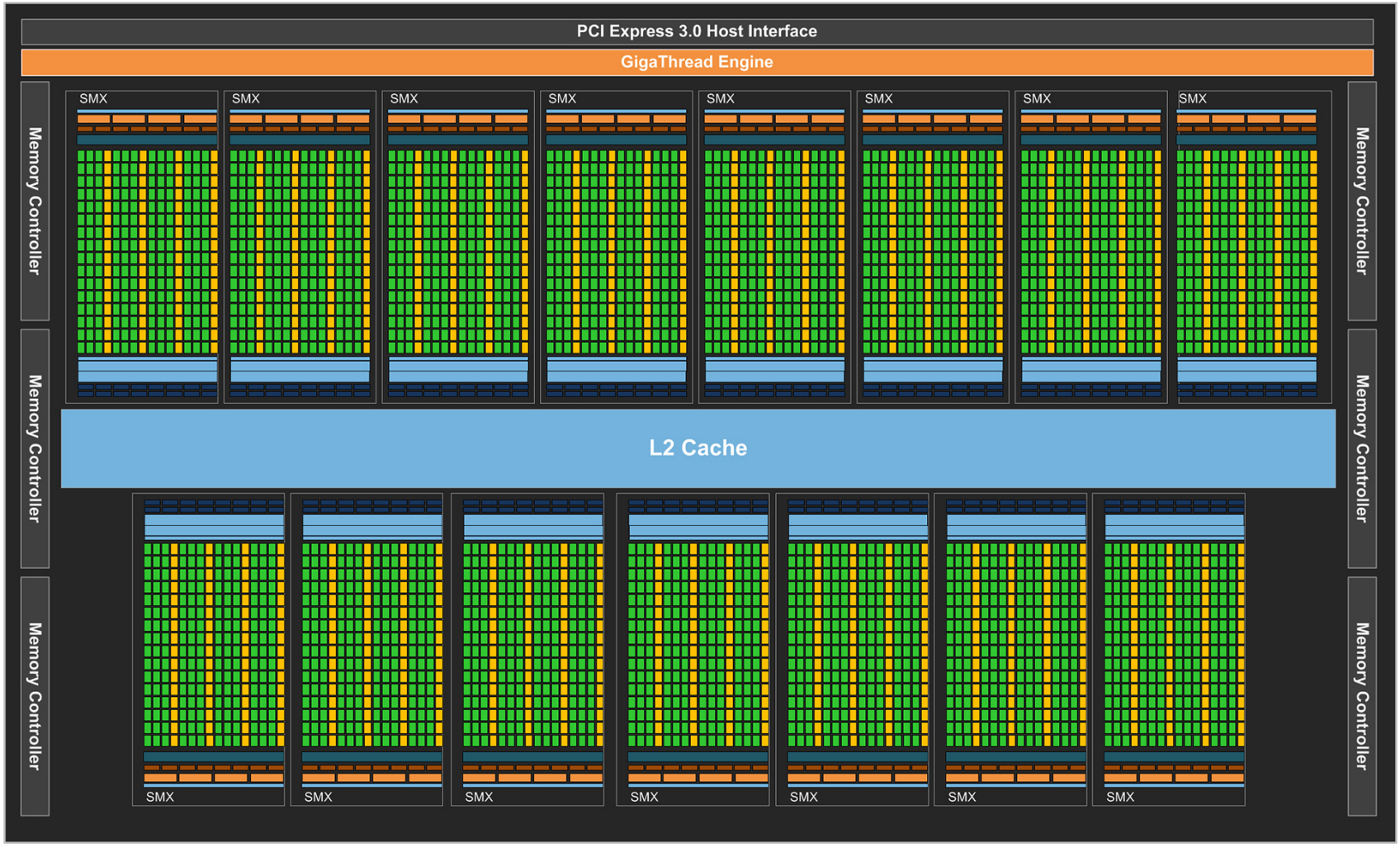
Size	6	Gb
Bus width	384	bits
Clock	1.5	GHz
Bandwidth	288	Gb/s

Price

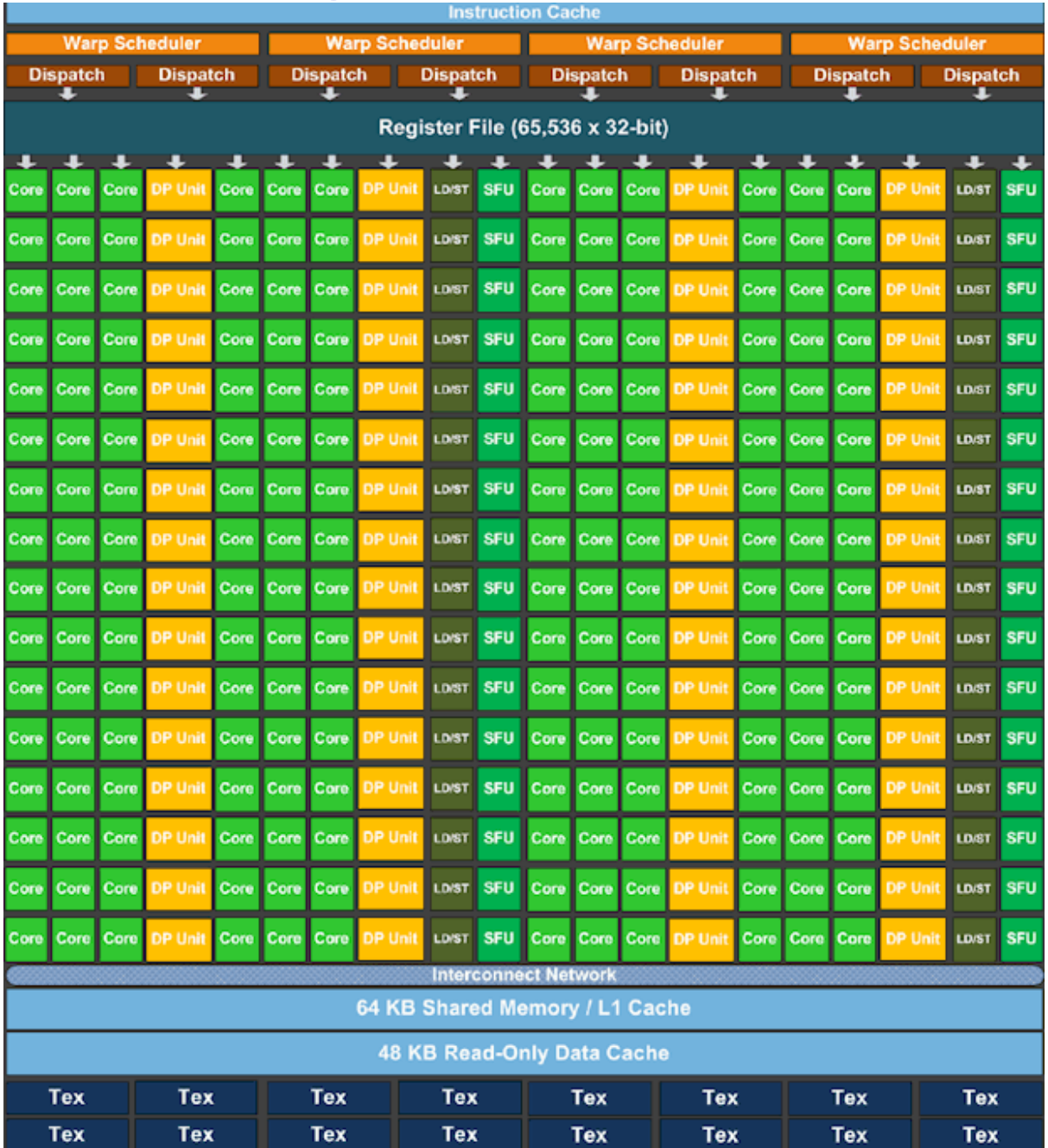
6500	kr
$1000	
€730	



GK 110 Block Diagram



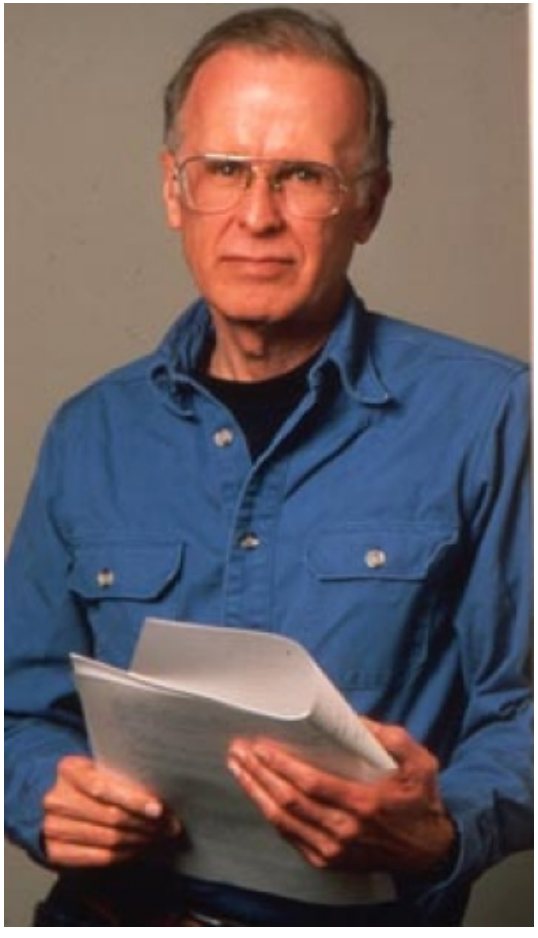
SMX Unit Block Diagram



Parallel Programming is Today's Challenge

Can Programming Be Liberated from the von Neumann Style?

John Backus, 1977 Turing Award Lecture



Best regards Walter
John Backus

Programmer's
Reference Manual
October 15, 1956

THE FORTRAN
AUTOMATIC CODING SYSTEM
FOR THE
IBM 704 EDPM®

This manual supersedes all earlier information about the FORTRAN system. It describes the system which will be made available during late 1956, and is intended to permit planning and FORTRAN coding in advance of that time. An Introductory Programmer's Manual and an Operator's Manual will also be issued.

APPLIED SCIENCE DIVISION
AND PROGRAMMING RESEARCH DEPT.
International Business Machines Corporation
590 Madison Ave., New York 22, N. Y.

WORKING COMMITTEE

J. W. BACKUS
R. J. BEEBER
S. BEST
R. GOLDBERG
H. L. HERRICK
R. A. HUGHES
University of California Radiation Laboratory, Livermore, Calif.
I. B. MITCHELL
R. A. NELSON
R. NUTT
United Aircraft Corp., East Hartford, Conn.
D. SAYRE
P. B. SHERIDAN
H. STERN
I. ZILLER

Key developer of FORTRAN, the first compiled language

Popular Programming Languages Aren't Mathematical

Math	Java

If $x = y$ then $f(x) = f(y)$	`x = 5;`
	`y = 5;`
	`add(x);`
	`add(y);`

Popular Programming Languages Aren't Mathematical

Math	Java

If $x = y$ then $f(x) = f(y)$	`x = 5;`
	`y = 5;`
	`add(x);`
	`add(y);`

If $y = z$ and $z = x$,
$y = x$
	assume z=5 and x=3
	`y = z;`
	`z = x;`
	y now 5; z now 3

Popular Programming Languages Aren't Mathematical

Math	Java

If $x = y$ then $f(x) = f(y)$	`x = 5;`
	`y = 5;`
	`add(x);`
	`add(y);`

If $y = z$ and $z = x$,
$y = x$
	assume z=5 and x=3
	`y = z;`
	`z = x;`
	y now 5; z now 3

If $x = x + 1$, world has ended	`x = x + 1; // add one`

Fibonacci Sequence Generator in C

```
int *fibArray(int n) {
  int a = 0, b = 1;
  int *fibs = malloc((sizeof int) * n);
  for (int i = 0; i < n; i++) {
    int temp = a + b;
    fibs[i] = a;
    a = b;
    b = temp;
  }
  return fibs;
}
```

Fibonacci Sequence Generator in Haskell

```
fibArray :: Int → [Int]
fibArray n = f 0 1 0
  where
    f a b i = if i < n then a : f b (a+b) (i+1)
                       else []
```

Fibonacci Sequence Generator in Haskell

```
fibs :: [Int]
fibs = 0 : 1 : [ a + b | (a, b) ← zip fibs (tail fibs) ]
```

fibs = 0 1

tail fibs = 1

zip fibs (tail fibs) = (0,1)

[a+b | (a,b) ← zip fibs (tail fibs)]
= 1

Fibonacci Sequence Generator in Haskell

```
fibs :: [Int]
fibs = 0 : 1 : [ a + b | (a, b) ← zip fibs (tail fibs) ]
```

```
                fibs =   0       1       1

           tail fibs =   1       1

zip fibs (tail fibs) = (0,1)   (1,1)

 [a+b | (a,b) ← zip fibs (tail fibs)]
                      =   1       2
```

Fibonacci Sequence Generator in Haskell

```
fibs :: [Int]
fibs = 0 : 1 : [ a + b | (a, b) ← zip fibs (tail fibs) ]
```

fibs =	0	1	1	2	3	5	8
tail fibs =	1	1	2	3	5	8	13
zip fibs (tail fibs) =	(0,1)	(1,1)	(1,2)	(2,3)	(3,5)	(5,8)	(8,13)
[a+b | (a,b) ← zip fibs (tail fibs)] =	1	2	3	5	8	13	21

A Four-Function Calculator in Three Slides

```
{ module Scanner where }

%wrapper "basic"

tokens :-
      $white+       ;                          -- Ignore spaces
      [0-9]+        { \s → Int (read s) }      -- Numbers, e.g., 42
      [\+\-\*\/\(\)] { \s → Sym (head s) }     -- Symbols + - * / ( )

{
  data Token = Int Int    -- Numbers and
             | Sym Char   -- Symbols
}
```

A Four-Function Calculator in Three Slides

```
{ module Main where
import Scanner }
%name parse
%tokentype { Token }
%token '+'  { Sym '+' }    '-'  { Sym '-' }    '*'  { Sym '*' }
       '/'  { Sym '/' }    '('  { Sym '(' }    ')'  { Sym ')' }
       Int  { Int $$ }
%left '+' '-'     -- Do add and subtract second
%left '*' '/'     -- Do multiply and divide first
%%

Expr : Expr '+' Expr { Bin $1 Add $3 } -- The Grammar
     | Expr '-' Expr { Bin $1 Sub $3 }
     | Expr '*' Expr { Bin $1 Mul $3 }
     | Expr '/' Expr { Bin $1 Div $3 }
     | '(' Expr ')'  { $2 }                -- Allow parentheses
     | Int           { Lit $1 }
```

A Four-Function Calculator in Three Slides

```
{
data Op = Add | Sub | Mul | Div                    -- Operators

data Expr = Bin Expr Op Expr | Lit Int             -- Operations & numbers

eval (Bin e1 op e2) =
  let (e1', e2') = (eval e1, eval e2) in           -- Evaluate children
      case op of Add →e1' + e2'                    -- then the operator
                 Sub → e1' − e2'
                 Mul →e1' ∗ e2'
                 Div → e1' `div` e2'
eval (Lit i) = i                                   -- Just a number

main = getContents >>=                             -- Read the input and...
          putStrLn . show . eval . parse . alexScanTokens

happyError _ = error "Parse error"
}
```