Instruction Set Specification

Defining the programming contract

Where we are

- Instructions defined abstractly
- Grouped into types
- Types are assigned values
- Next step is designing instruction formats
- Assign codes for operations

Field type: Instruction type

- Usually fixed in position for all types
  - Makes decoding easier
- Typically 2 to 4 bits (4 to 16 types)
- Can have a type that indicates additional bits specify a sub-type
  - Example: 3-bit type, value 7 indicates next 2 bits extend the type

Field type: Opcode

- Often starts in same position for all types, but may differ in length
  - Example: Many more ALU ops than branch ops
- Typically 3 to 6 bits
  - May have values that indicate additional bits extend the opcode (typically used when type field is small)

Field type: Register addr

- $\log_2(R)$ bits, where R= # registers
- One field per operand
- Not every instruction type needs 3 fields
  - Two or one address architectures use fewer
  - Some operations may have 4 fields
- Need not be adjacent

Field type: Immediate

- Made up of “left-over” bits
- Size may vary with format
- Usually sign extended by operations

Other Fields

- Condition (may be in branch only, or predicate for all)
- Set/Don’t set condition (a la Arm)
- Shift amount (how many positions)
- “Micro-op” formatting

Endianness

- Byte order within a word
- Given value 0x12345678, what order should the bytes (12, 34, 56, 78) appear in a word?
- 12 is high order, 78 is low order

Endianness

If you say:

Byte address:	0	1	2	3
Value:	12	34	56	78

Then you are in favor of Little-endian addressing
(low order byte comes at the end)

Endianness

If you say:

Byte address:	0	1	2	3
Value:	78	56	34	12

Then you are in favor of Big-endian addressing
(high order byte comes at the end)

What About Strings?

Given ASCII string (8-bit subset of Unicode): “ABCD”

Normal placement is first character in low-order byte,
last character in high-order byte

What About Strings?

Given ASCII string (8-bit subset of Unicode): “ABCD”
Little endian byte address 3 holds the low-order byte, so all of you “Little-endians” end up with:

Byte address:	0	1	2	3
Value:	D	C	B	A

First character, ‘A’, is in low order byte,
last character, ‘D’, is in high order byte

What About Strings?

Given ASCII string (8-bit subset of Unicode): "ABCD"

Big endian byte address 0 holds the low-order byte,

so "Big-endians" end up with:

Byte address:	0	1	2	3
Value:	A	B	C	D

First character, 'A', is in low order byte,

last character, 'D', is in high order byte

Does Endianness Matter?

- Within an architecture, it’s just a matter of wiring, and everything is consistent
- Transferring files between systems of different endianness complicates data sharing

Modes

- Endianness can be modal
- Can have alternate instruction set modes
- Typical modes are security levels
  - User/Supervisor, additional levels for VM, etc.

Interrupts

- Asynchronously cause jump to handler
- Usually a low area of memory contains a table of jumps
- Interrupt type N jumps to Nth element of jump table, causing jump to handler (saves PC in separate return register)
- Need to be able to turn off
- Often supported by supervisor mode

Virtual Memory

- Generally requires supervisor mode
  - Privileged instructions to manage memory map
- Defines virtual to physical address mapping
- Can be paged, segmented, combination, or a hierarchy
- Not required for simulator project

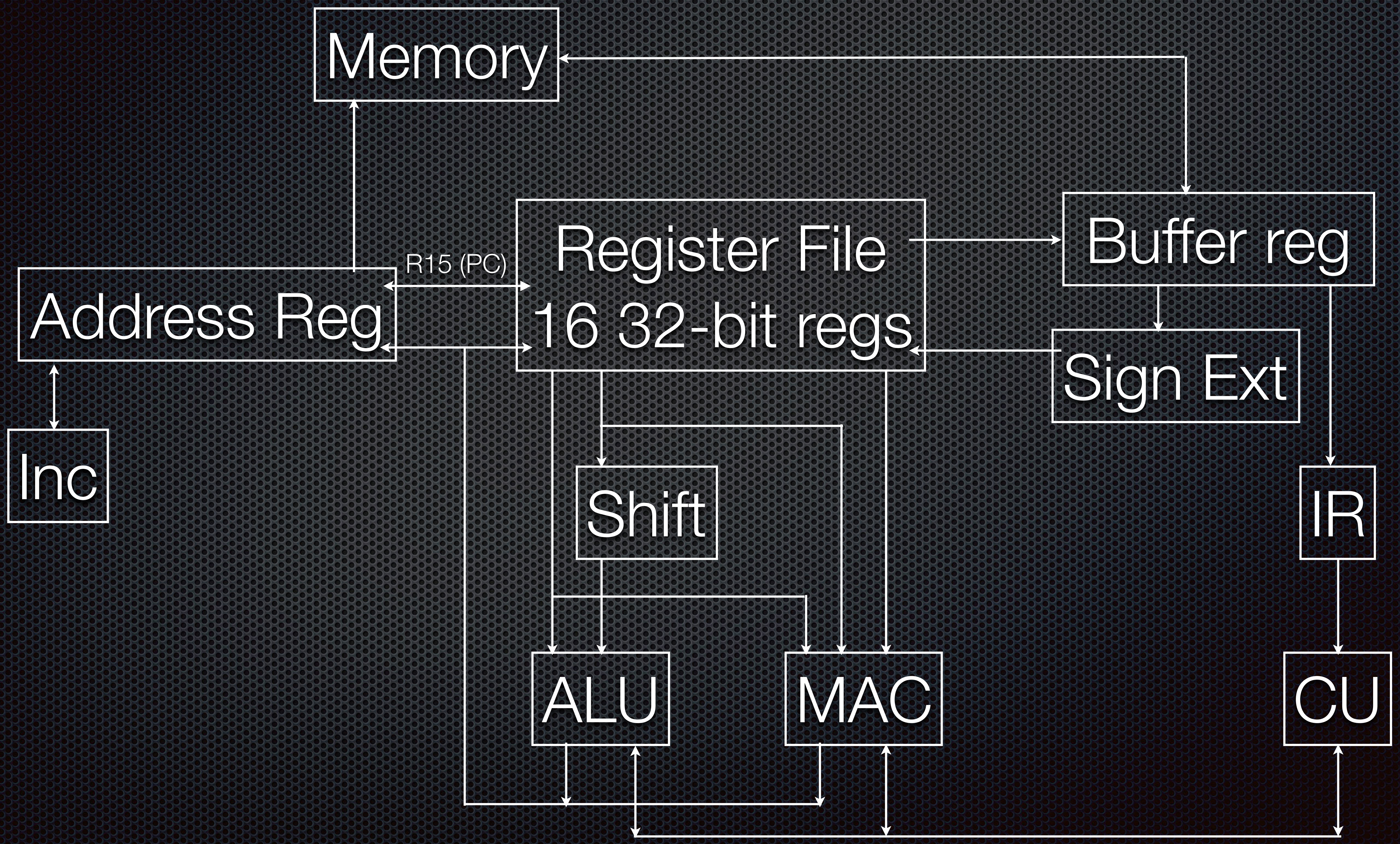
Microarchitecture Ops

- Sometimes need to interact with aspects of the microarchitecture
- Typically need ability to force cache lines to flush to memory for I/O
- May have branch hints for predictor warm-up
- May have privileged instructions for managing other cached state (TLB, branch predictor)

Arm Architecture

Example Embedded RISC Processor

ARM Organization



Arm Registers

- 32-bits each
- R0 - R12 General Purpose
- R13 Stack Pointer
- R14 Link Register (subroutine return)
- R15 Program Counter
- CPSR Status Register
  - Condition codes, overflow, etc.

ARM Data Types

- 8, 16, 32, 64-bit integers (depending on model)
- Floating point and vector supported by some versions
- Can be big endian or little endian, as set by user

ARM Instruction Formats

- Bits 31-28: Condition Code
- Bits 27-25: Instruction type
- Types 000 - 001:
  - Bits 24-21: Opcode
  - Bit 20: Condition code update flag
  - Bits 19-16: Source register 1
  - Bits 15-12: Destination register

Type 000

- Bits 6-5: Shift type
- Bits 3-0: Source register 2
- Immediate Shift Subset (Bit 4 = 0)
  - Bits 11-7: Shift amount
- Register Shift Subset (Bit 4 = 1)
  - Bits 11-8: Register with shift count
  - Bit 7 = 0

Other Types

- 001: Data processing with immediate operand (7-0) and rotate(11-8)
- 010: Load/store with immediate offset
- 011: Load/store with register offset
- 100: Load/store multiple
- 101: Branch or branch with link

Formats

	31 30 29 28	27	26	25	24	23	22	21	20	19 18 17 16	15 14 13 12	11 10 9 8	7	6 5	4	3 2 1 0
Data processing immediate shift	cond [1]	0	0	0	opcode				S	Rn	Rd	shift amount		shift	0	Rm
Miscellaneous instructions: See Figure A3-4	cond [1]	0	0	0	1	0	x	x	0	x x x x	x x x x	x x x x	x	x x	0	x x x x
Data processing register shift [2]	cond [1]	0	0	0	opcode				S	Rn	Rd	Rs	0	shift	1	Rm
Miscellaneous instructions: See Figure A3-4	cond [1]	0	0	0	1	0	x	x	0	x x x x	x x x x	x x x x	0	x x	1	x x x x
Multiplies: See Figure A3-3 Extra load/stores: See Figure A3-5	cond [1]	0	0	0	x	x	x	x	x	x x x x	x x x x	x x x x	1	x x	1	x x x x
Data processing immediate [2]	cond [1]	0	0	1	opcode				S	Rn	Rd	rotate	immediate			
Undefined instruction	cond [1]	0	0	1	1	0	x	0	0	x x x x	x x x x	x x x x	x	x x	x	x x x x
Move immediate to status register	cond [1]	0	0	1	1	0	R	1	0	Mask	SBO	rotate	immediate			
Load/store immediate offset	cond [1]	0	1	0	P	U	B	W	L	Rn	Rd	immediate				

More Formats

Load/store register offset	cond [1]	0 1 1	P	U	B	W	L	Rn	Rd	shift amount	shift	0	Rm
Media instructions [4]: See Figure A3-2	cond [1]	0 1 1	x x x x x x x x x x x x x x x x x x x x	1	x x x x								
Architecturally undefined	cond [1]	0 1 1	1 1 1 1 1	x x x x x x x x x x x x	1 1 1 1	x x x x							
Load/store multiple	cond [1]	1 0 0	P	U	S	W	L	Rn	register list				
Branch and branch with link	cond [1]	1 0 1	L	24-bit offset									
Coprocessor load/store and double register transfers	cond [3]	1 1 0	P	U	N	W	L	Rn	CRd	cp_num	8-bit offset		
Coprocessor data processing	cond [3]	1 1 1 0	opcode1	CRn	CRd	cp_num	opcode2	0	CRm				
Coprocessor register transfers	cond [3]	1 1 1 0	opcode1	L	CRn	Rd	cp_num	opcode2	1	CRm			
Software interrupt	cond [1]	1 1 1 1	swi number										
Unconditional instructions: See Figure A3-6	1 1 1 1	x x x x x x x x x x x x x x x x x x x x x x x x x x x x											

Condition Codes

Opcode [31:28]	Mnemonic extension	Meaning	Condition flag state
0000	EQ	Equal	Z set
0001	NE	Not equal	Z clear
0010	CS/HS	Carry set/unsigned higher or same	C set
0011	CC/LO	Carry clear/unsigned lower	C clear
0100	MI	Minus/negative	N set
0101	PL	Plus/positive or zero	N clear
0110	VS	Overflow	V set
0111	VC	No overflow	V clear
1000	HI	Unsigned higher	C set and Z clear
1001	LS	Unsigned lower or same	C clear or Z set
1010	GE	Signed greater than or equal	N set and V set, or N clear and V clear (N == V)
1011	LT	Signed less than	N set and V clear, or N clear and V set (N != V)
1100	GT	Signed greater than	Z clear, and either N set and V set, or N clear and V clear (Z == 0,N == V)
1101	LE	Signed less than or equal	Z set, or N set and V clear, or N clear and V set (Z == 1 or N != V)
1110	AL	Always (unconditional)	-
1111	-	See *Condition code 0b1111*	-

DP Instructions

Opcode	Mnemonic	Operation	Action
0000	AND	Logical AND	Rd := Rn AND shifter_operand
0001	EOR	Logical Exclusive OR	Rd := Rn EOR shifter_operand
0010	SUB	Subtract	Rd := Rn - shifter_operand
0011	RSB	Reverse Subtract	Rd := shifter_operand - Rn
0100	ADD	Add	Rd := Rn + shifter_operand
0101	ADC	Add with Carry	Rd := Rn + shifter_operand + Carry Flag
0110	SBC	Subtract with Carry	Rd := Rn - shifter_operand - NOT(Carry Flag)
0111	RSC	Reverse Subtract with Carry	Rd := shifter_operand - Rn - NOT(Carry Flag)
1000	TST	Test	Update flags after Rn AND shifter_operand
1001	TEQ	Test Equivalence	Update flags after Rn EOR shifter_operand
1010	CMP	Compare	Update flags after Rn - shifter_operand
1011	CMN	Compare Negated	Update flags after Rn + shifter_operand
1100	ORR	Logical (inclusive) OR	Rd := Rn OR shifter_operand
1101	MOV	Move	Rd := shifter_operand (no first operand)
1110	BIC	Bit Clear	Rd := Rn AND NOT(shifter_operand)
1111	MVN	Move Not	Rd := NOT shifter_operand (no first operand)

ARM Manual

- Available on 335 course page:
- Arm v7 Reference Manual
- https://people.cs.umass.edu/~weems/homepage/courses/cmpsci-335.html

Team Homework

- For Wednesday 2/13, complete ISA description, including register set description
  - Include format diagrams, encodings
  - Explain all instructions, esp. load/store, branch
  - Use ARM manual as guide for descriptions
- This is the draft of your ISA report

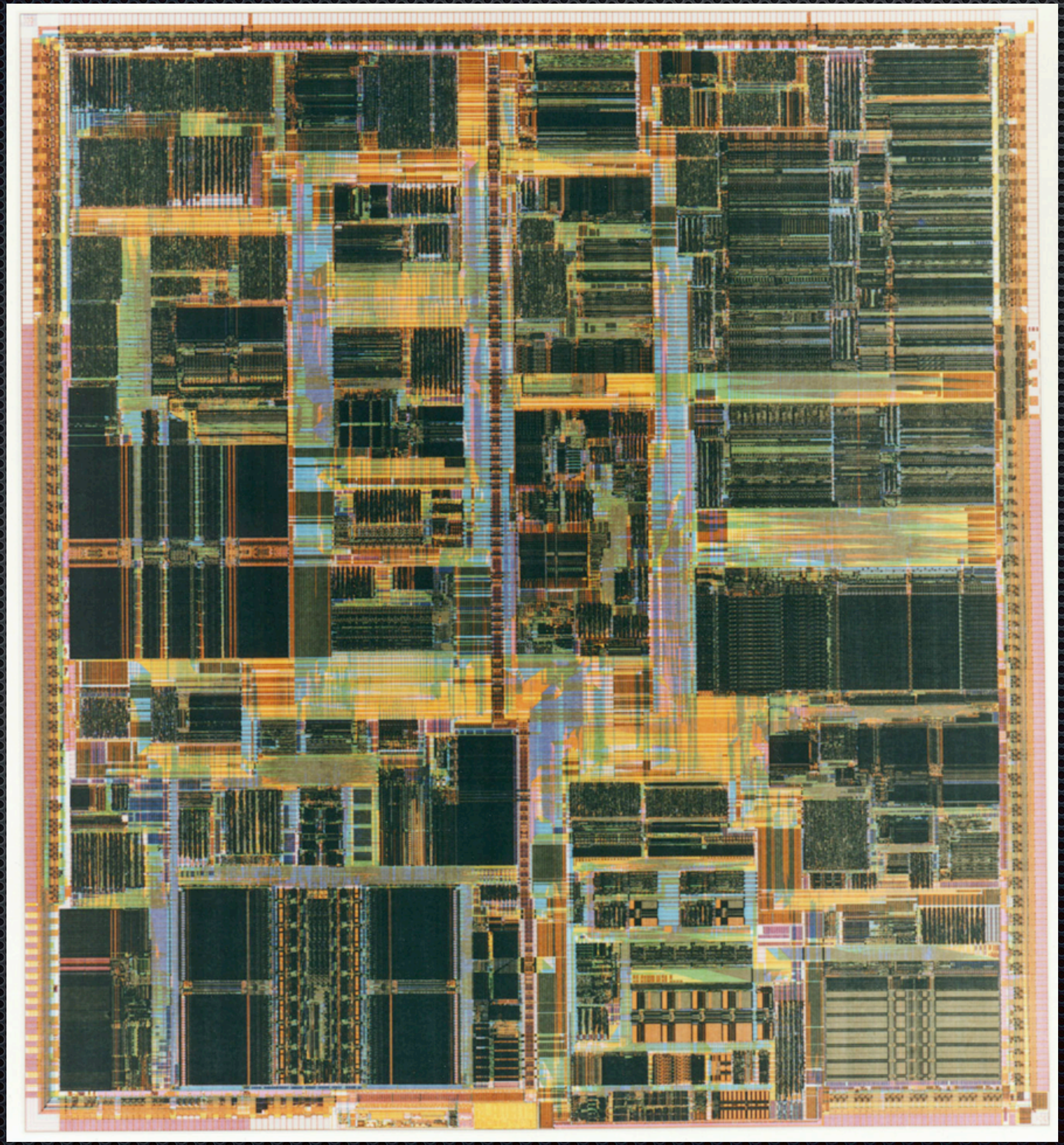
Teamwork

Remember to work as a team!

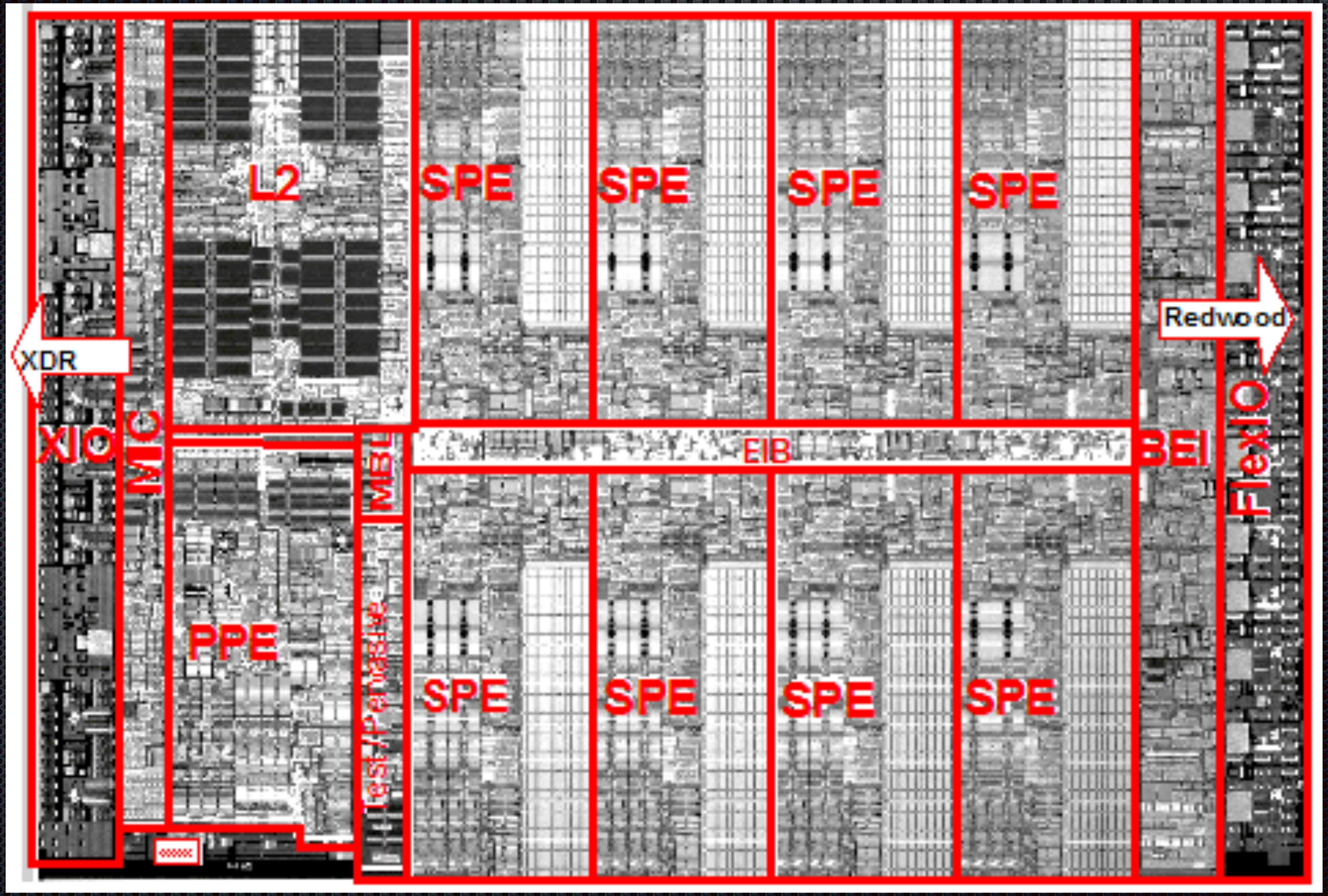
VLSI Technology

Fabrication and Structures

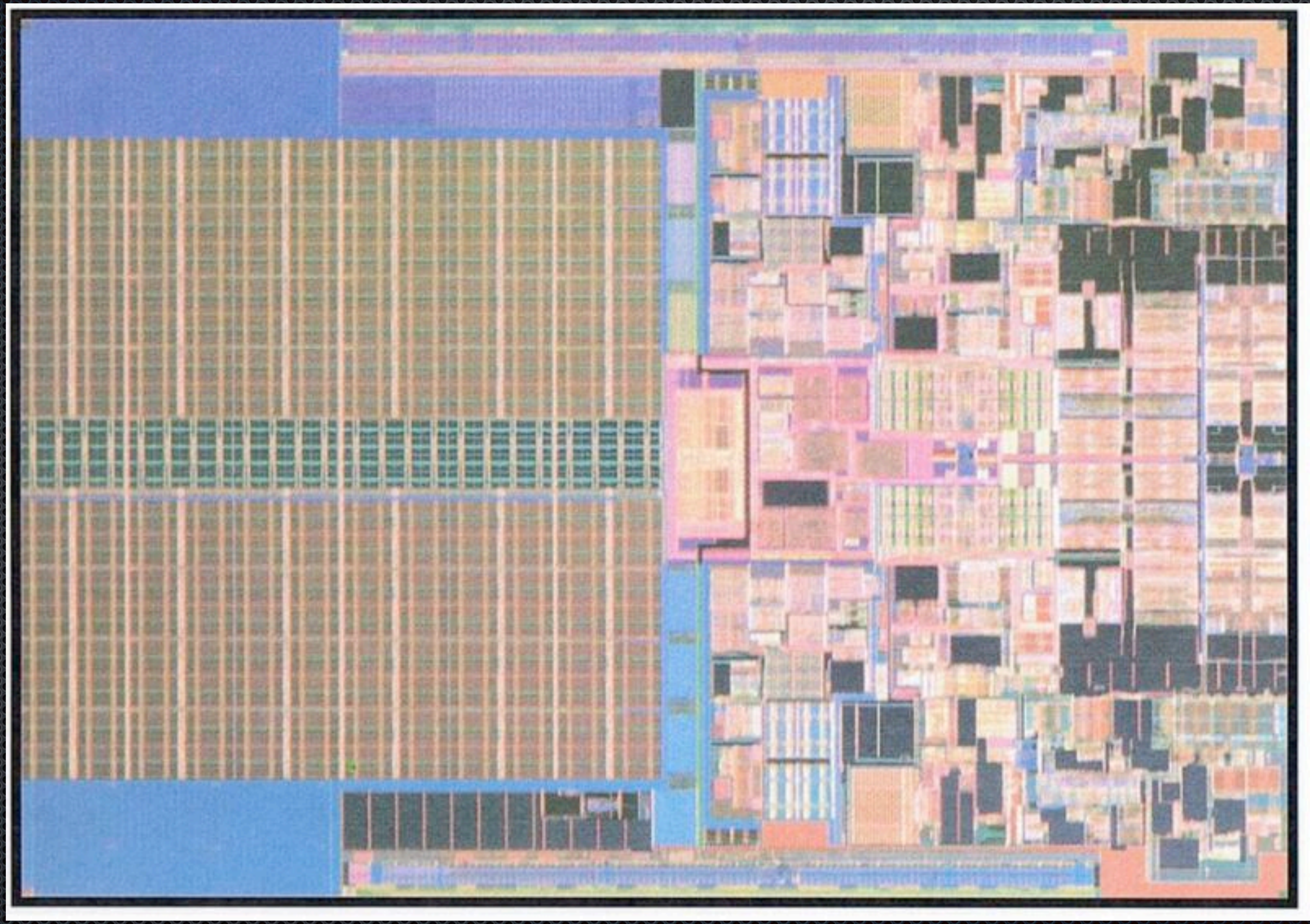
Chip Die Photo (P6)



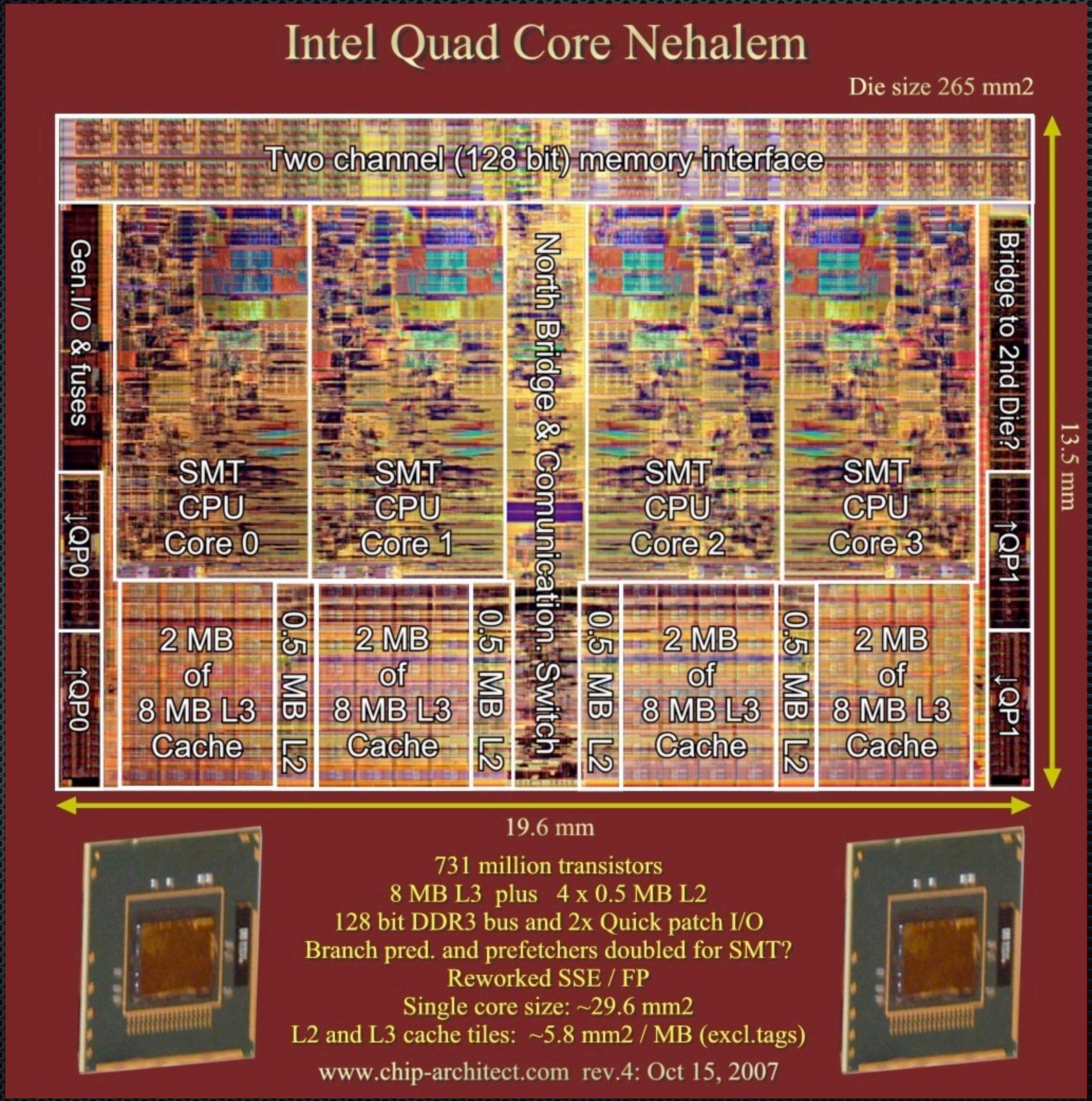
Another Chip (Cell)



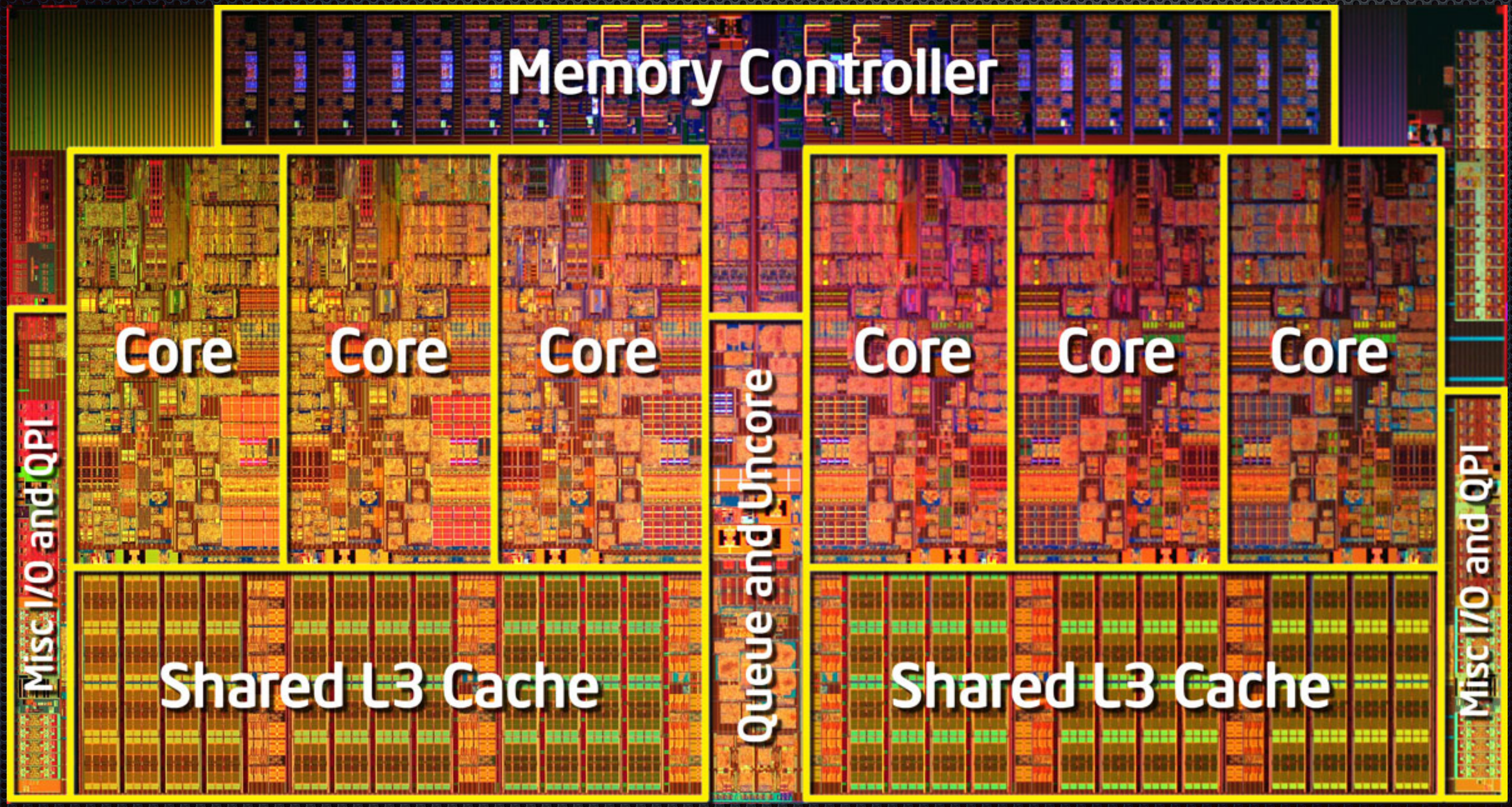
Dual-Core (Penryn)



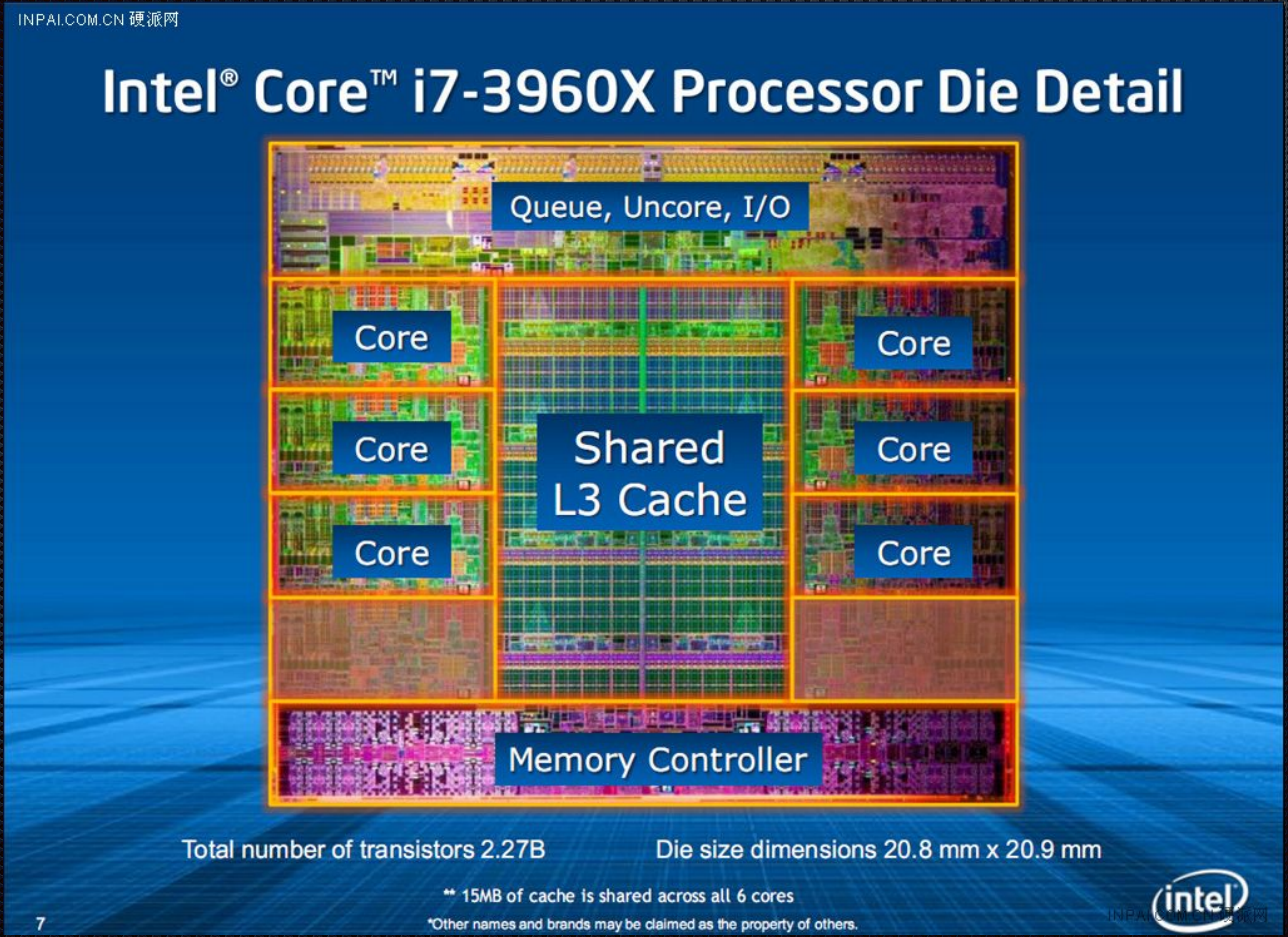
Quad-Core (Nehalem)



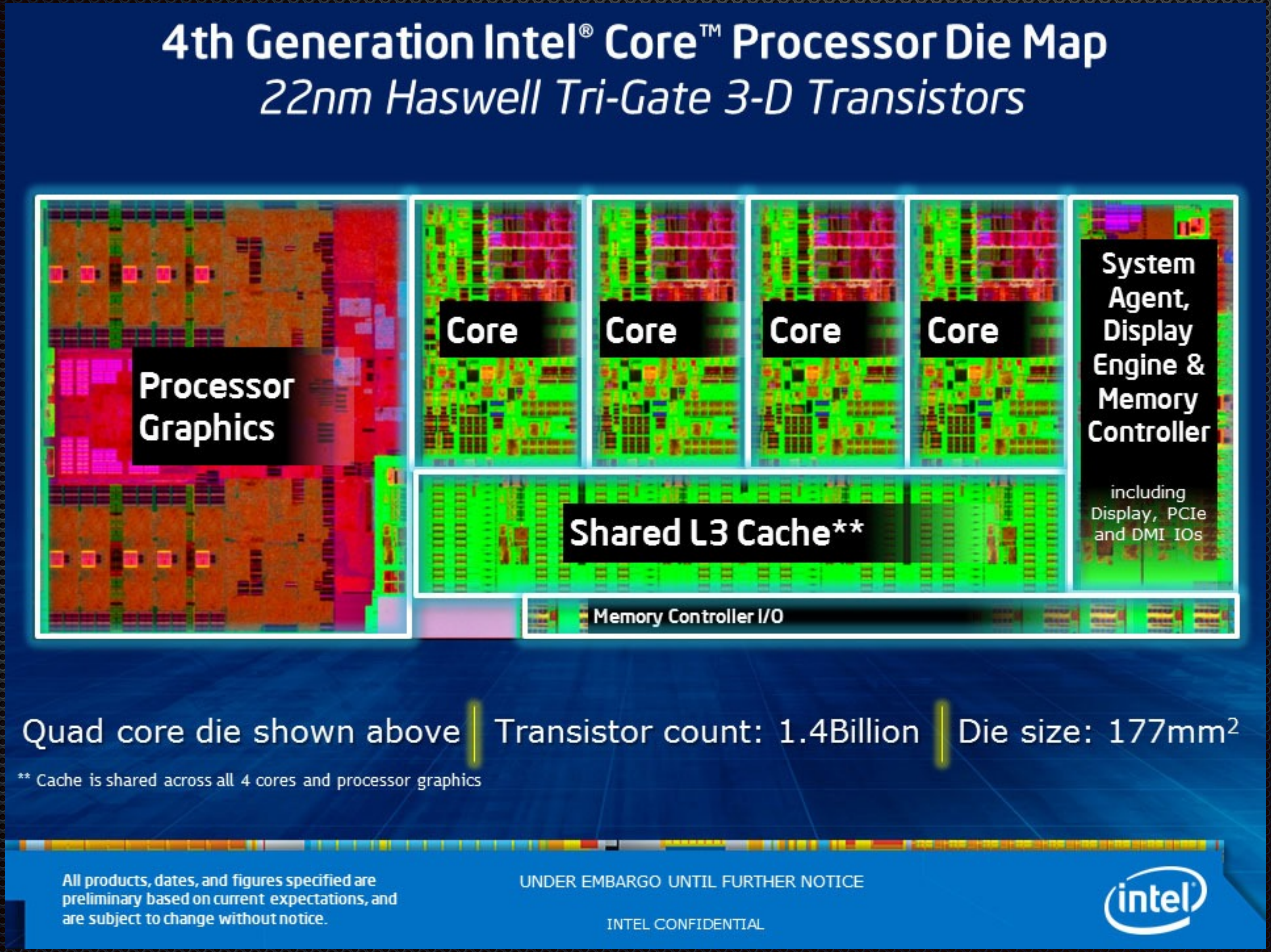
Six-core Gulftown



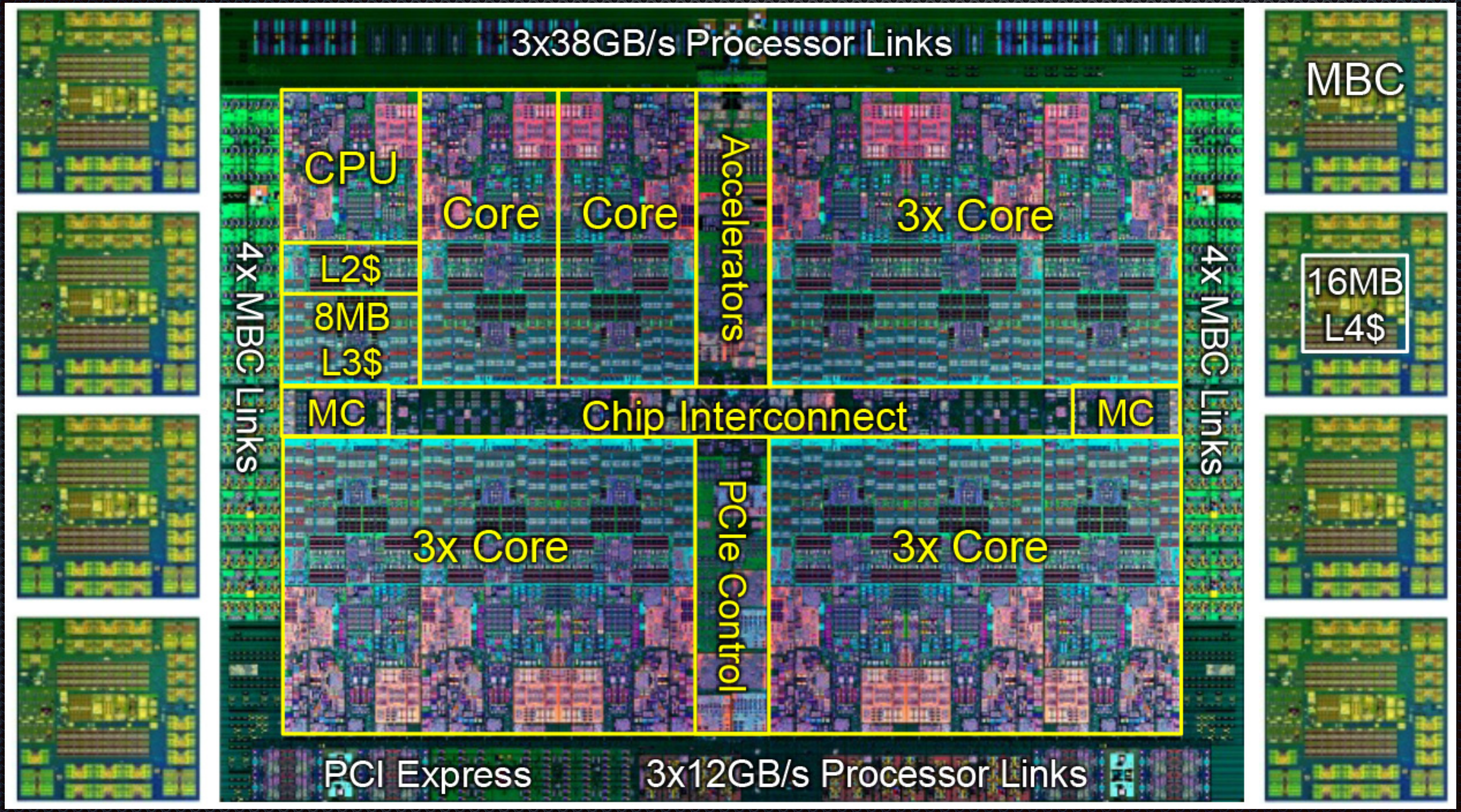
Six-core i7(Sandy Bridge)



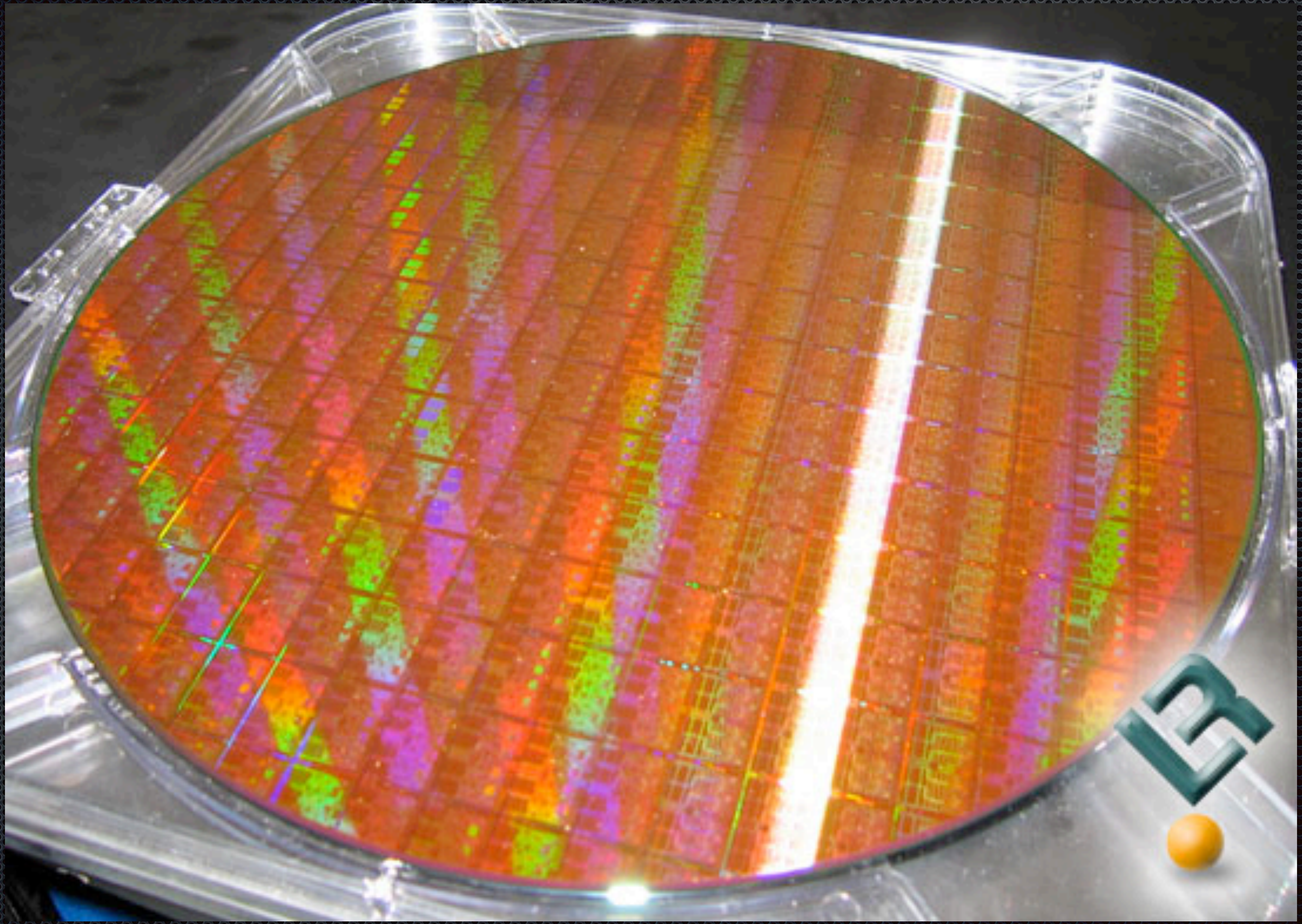
Quad-Core Haswell w/GPU



IBM Power 8 with 12 cores

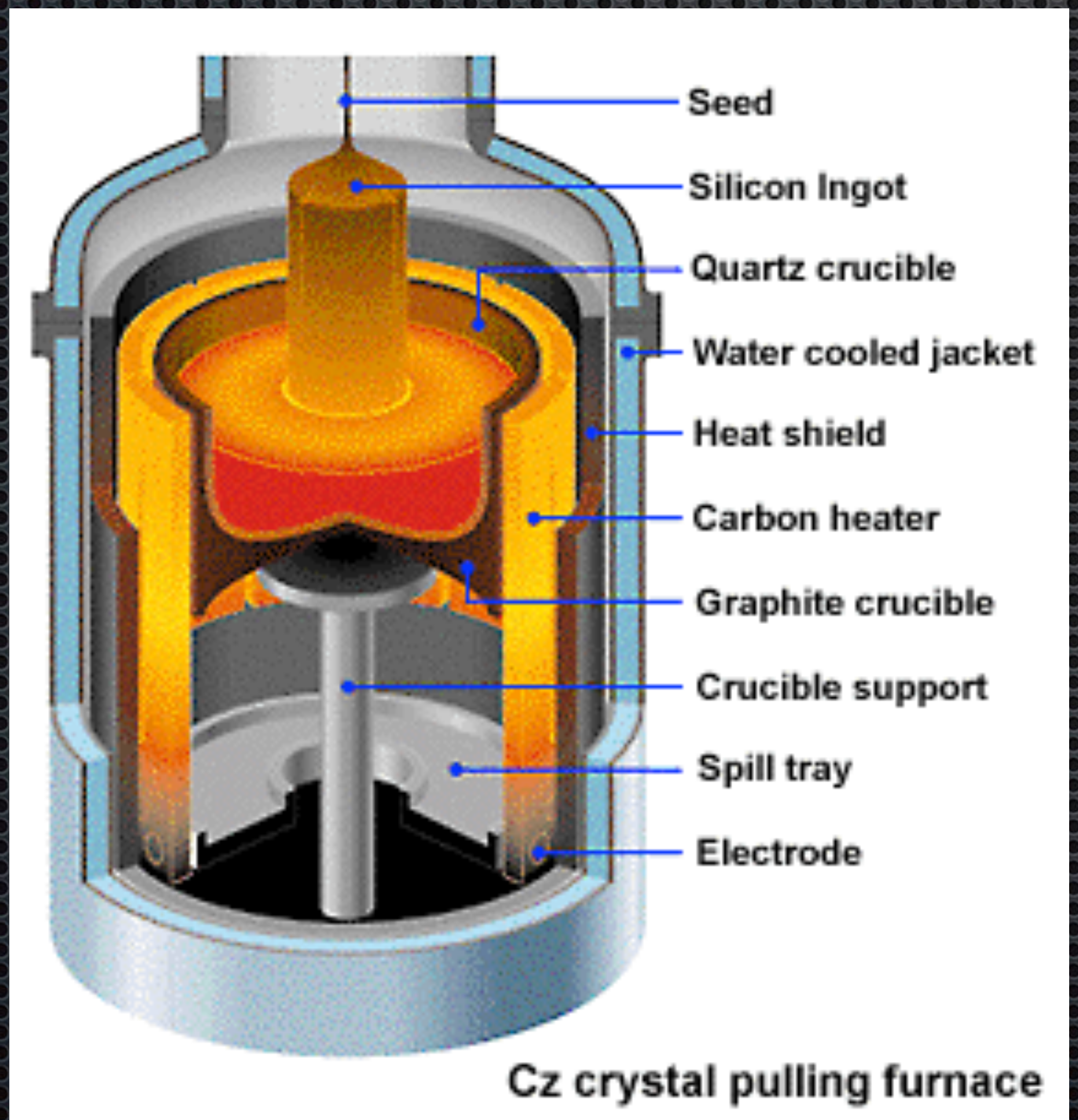


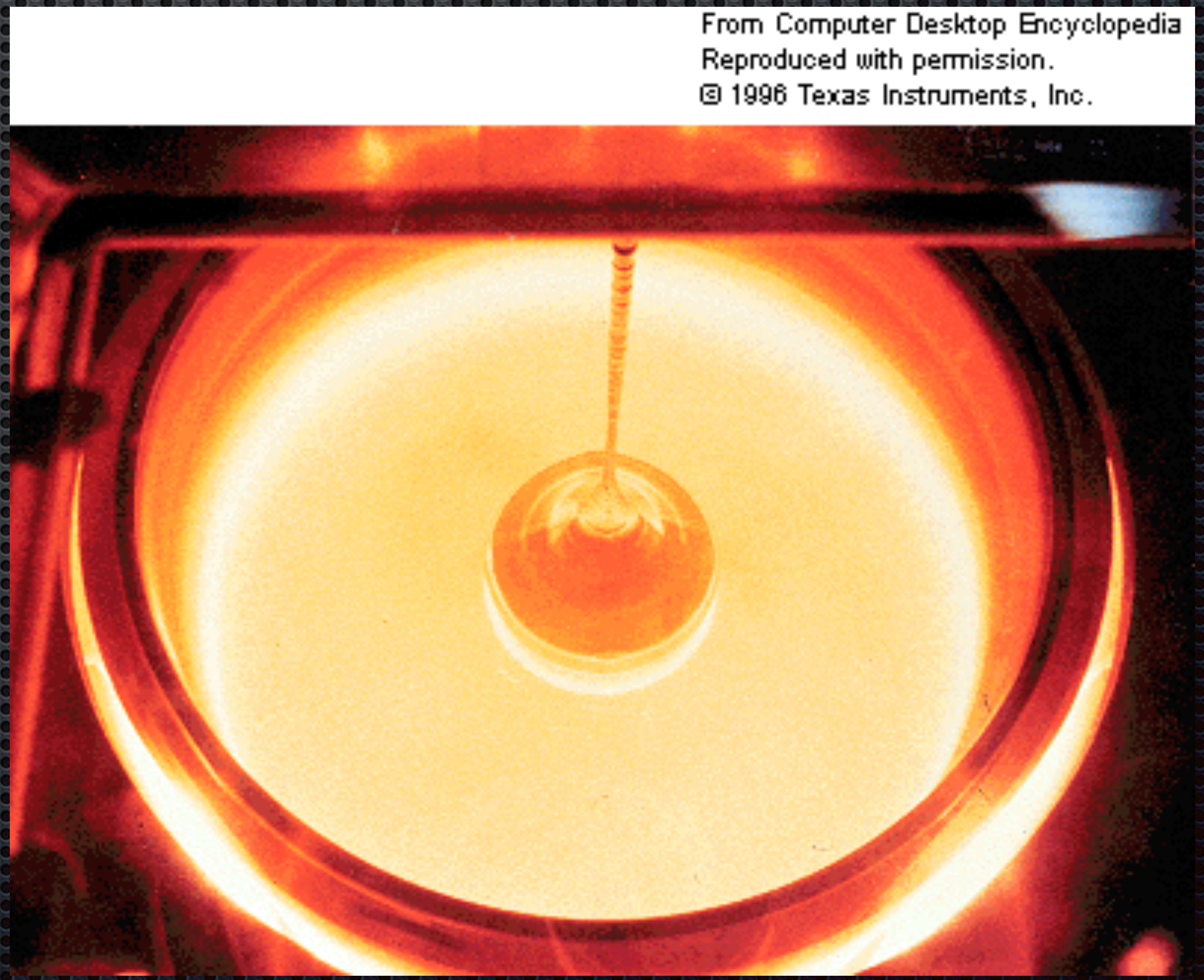
300mm Wafer



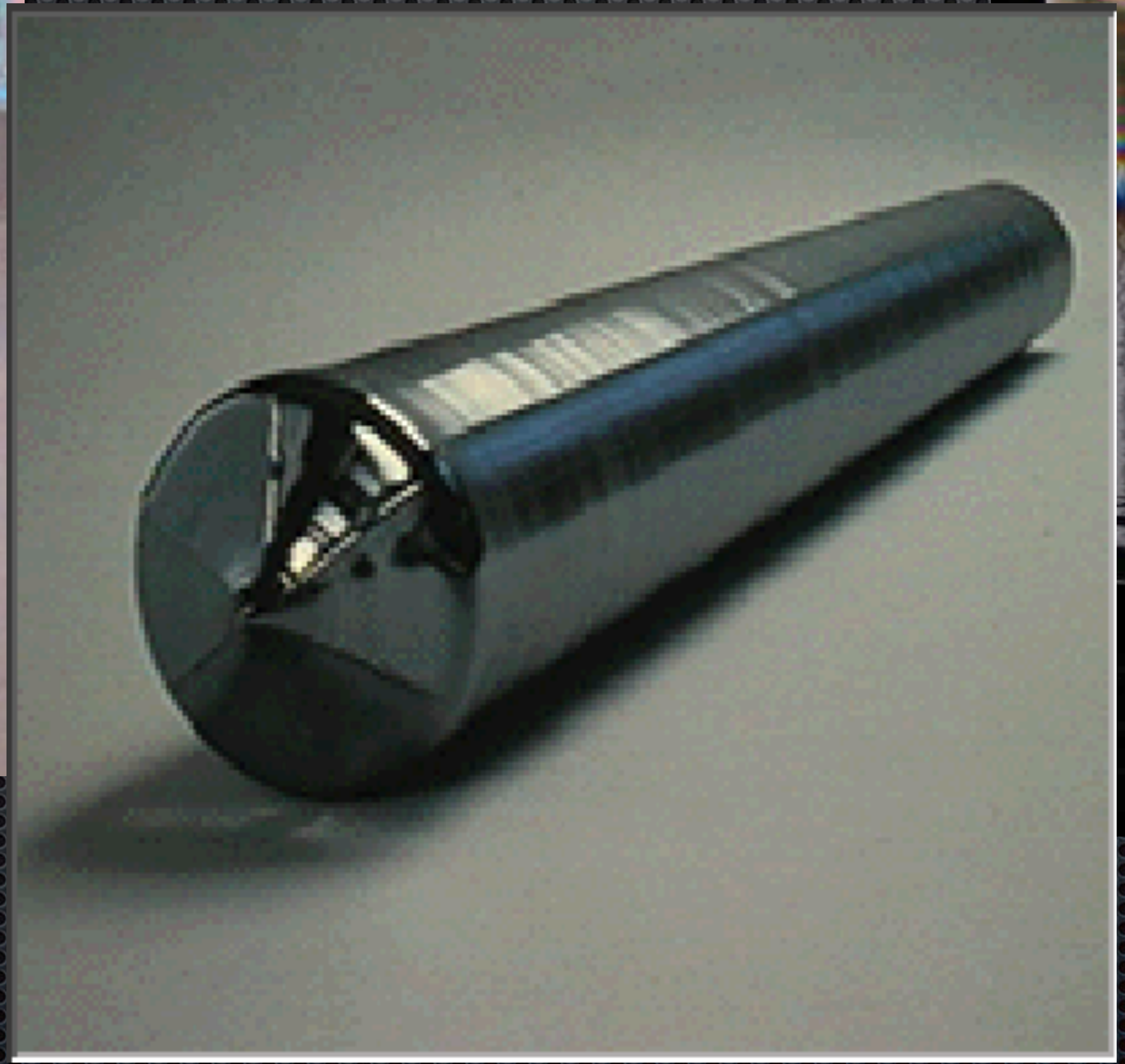
Fabrication Begins

- 99.9999999% pure Silicon, purified from old sand
- Slowly draw a single crystal boule (ingot) out of a melt, starting from a seed



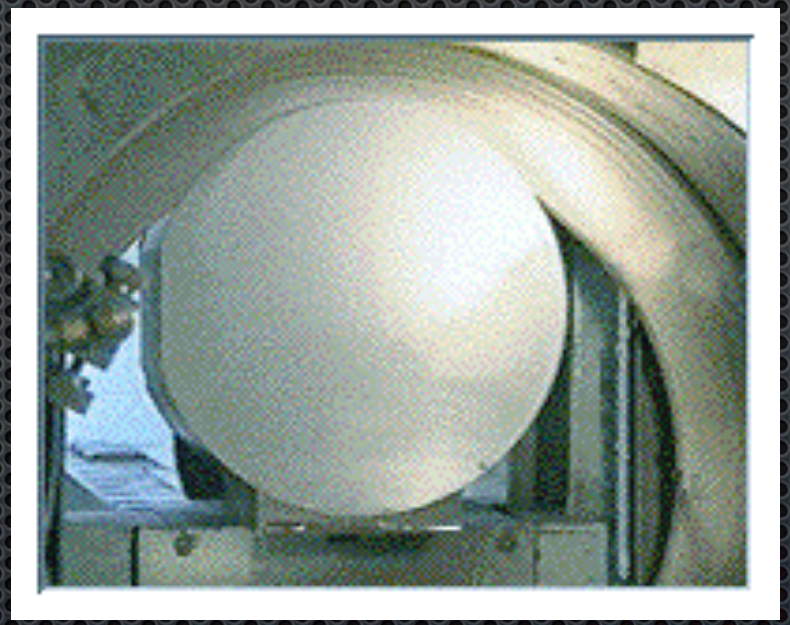
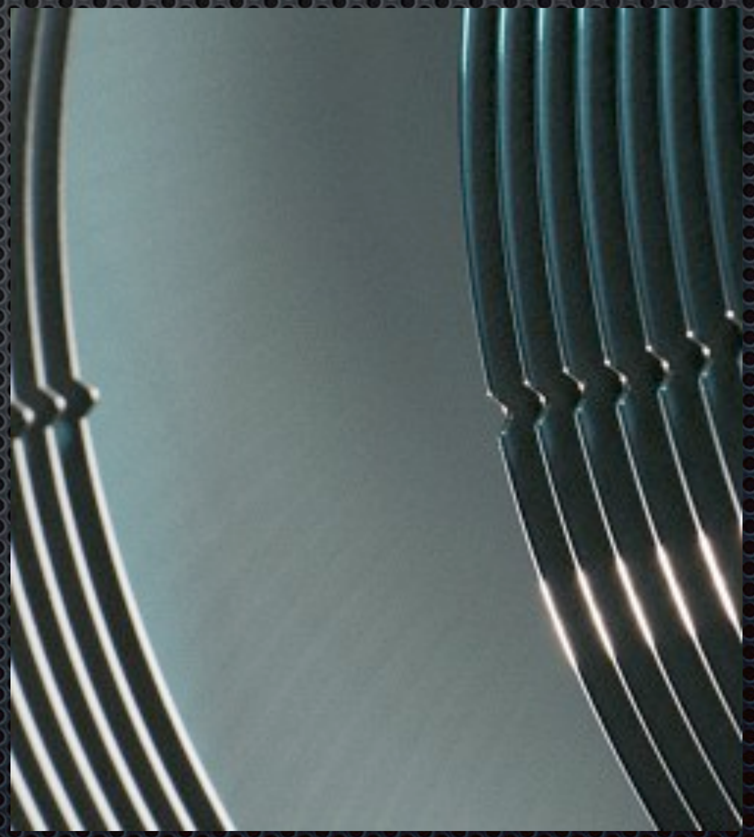
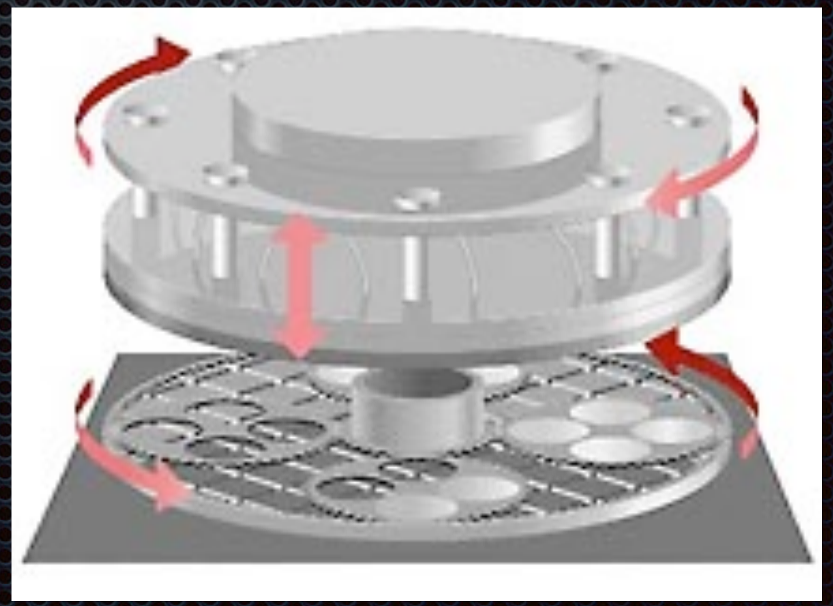


Silicon Boules



Turn into Wafers

- Lathe to a cylinder, machine flat or notch on one side
- Slice with inside diameter diamond saw, or parallel wire saw
- Polish, clean, and overcoat

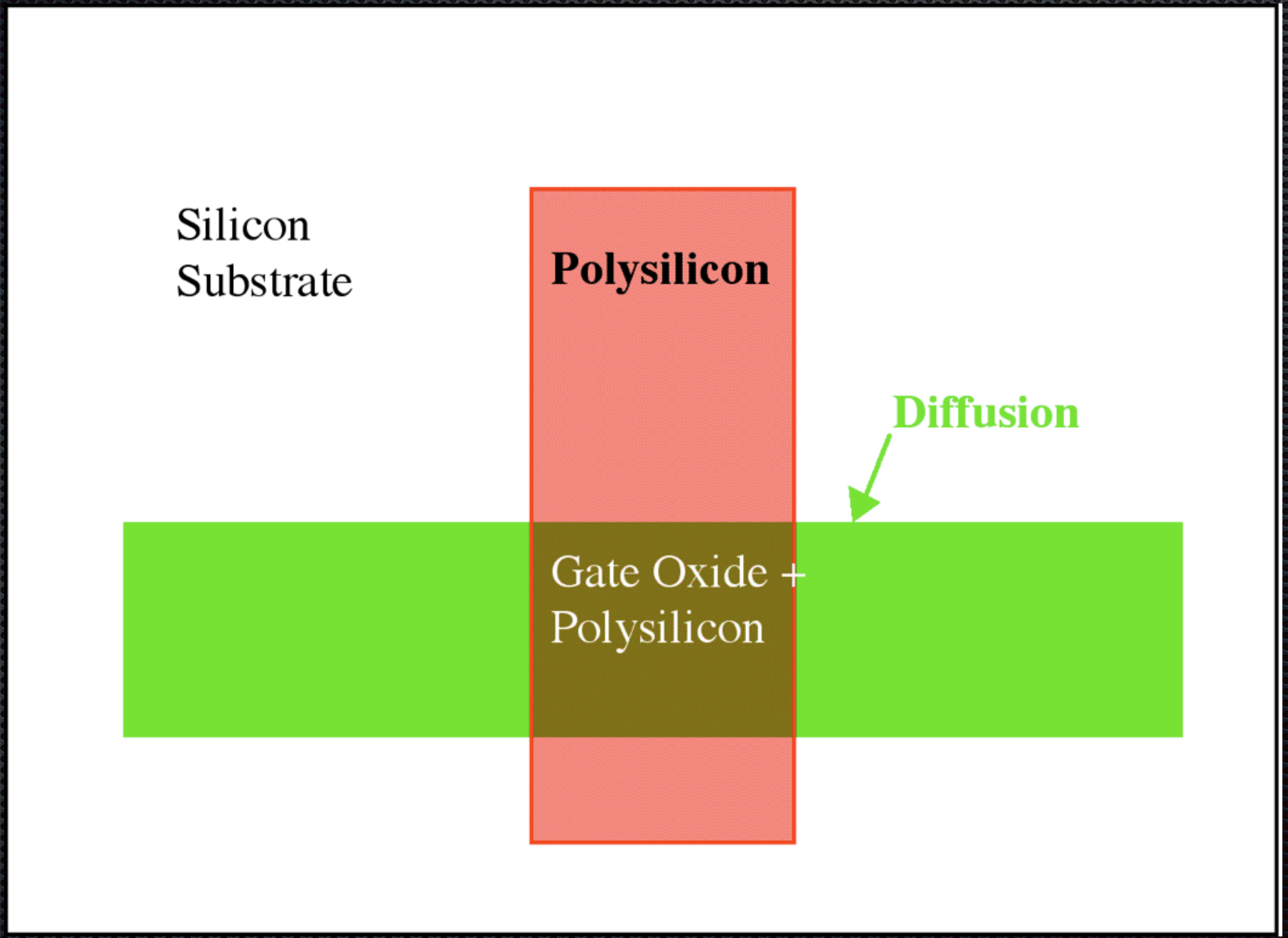





Processing Cycle

- Deposit a material
- Deposit a UV-sensitive photoresist
- Expose through a quartz/chrome mask
- Wash away unwanted resist
- Acid etch material into pattern
- Repeat many times (35 masks, 700 steps)

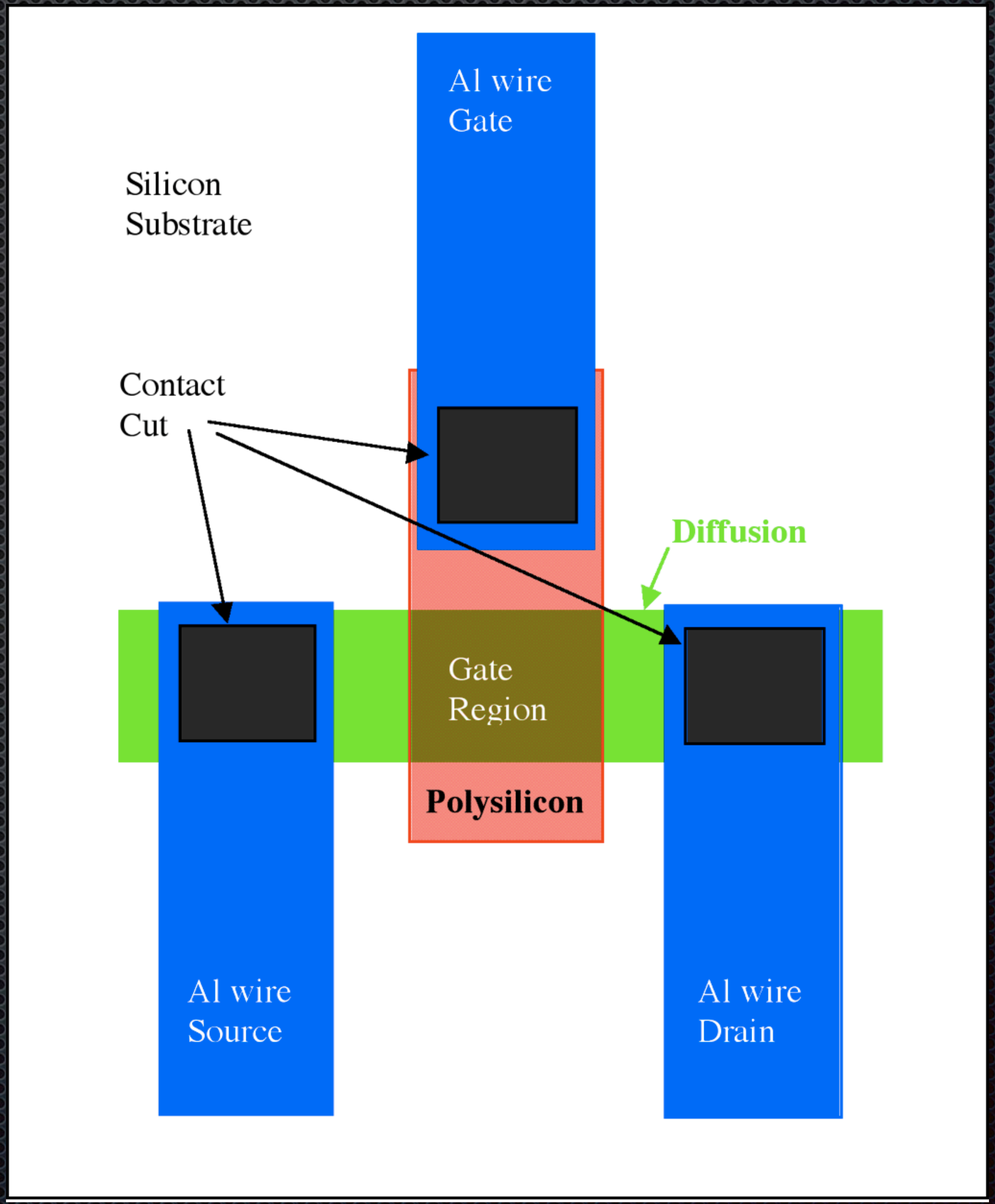


Typical Pattern

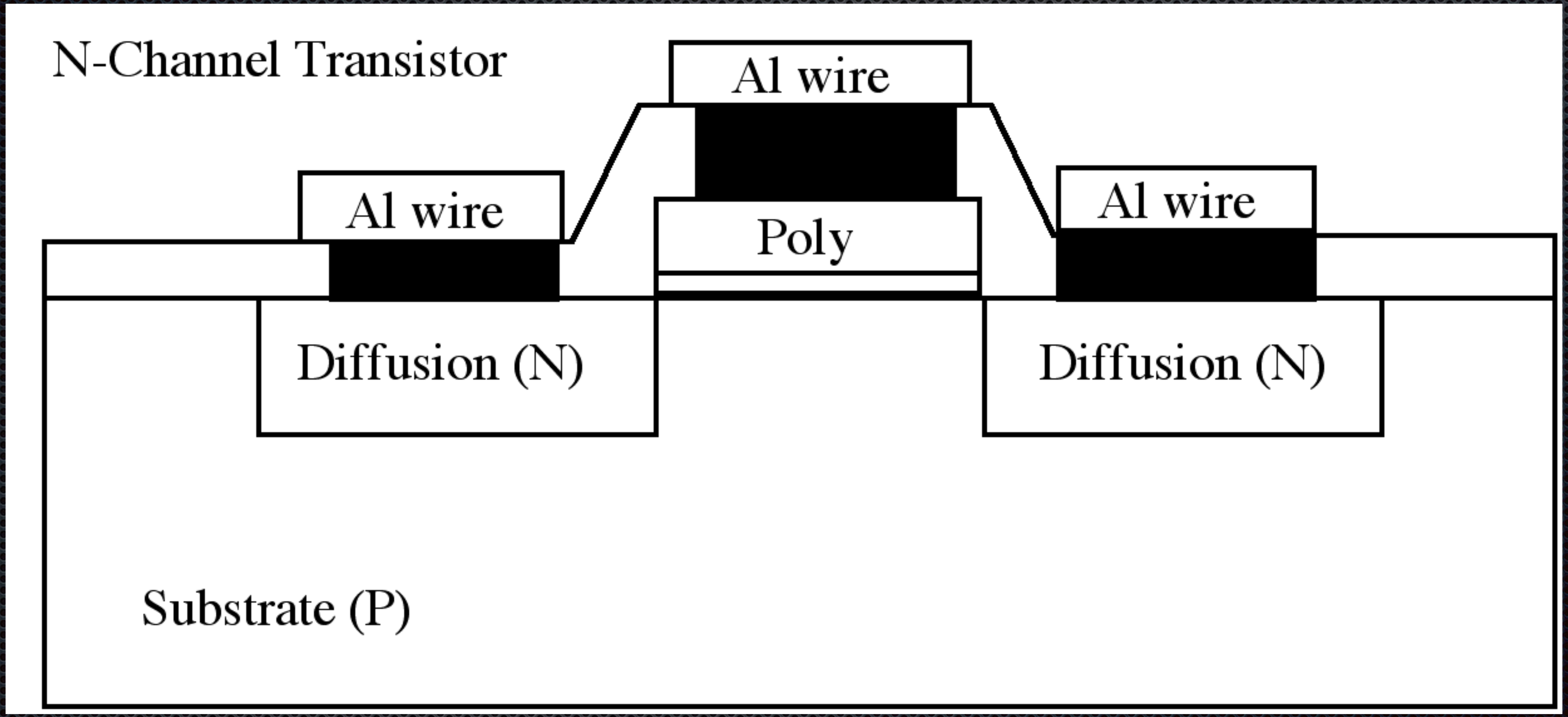


After More Steps

- Highly simplified layout
- Doesn't show wells, guard rings, multiple metal layers, etc.

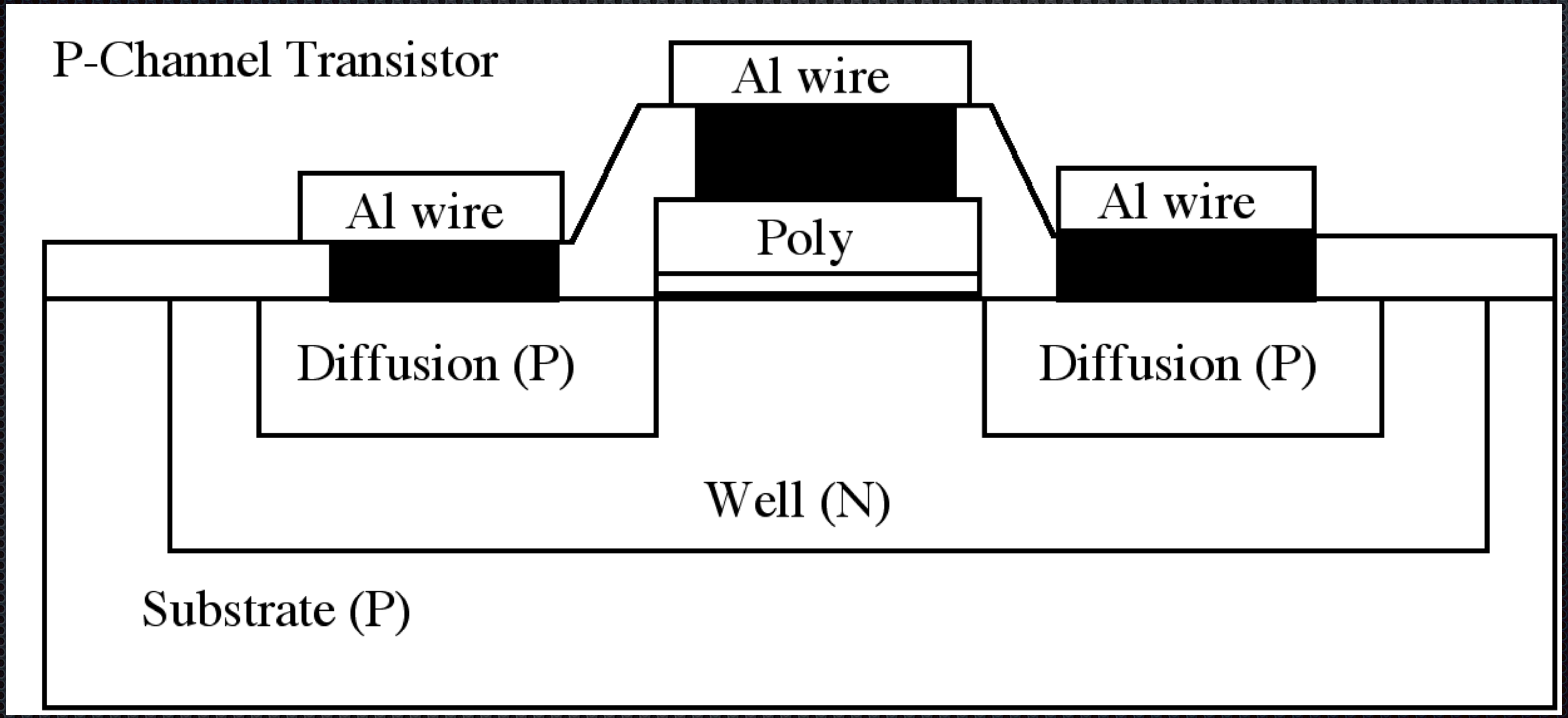


Cross Section
(NMOS transistor -- 1 = on)



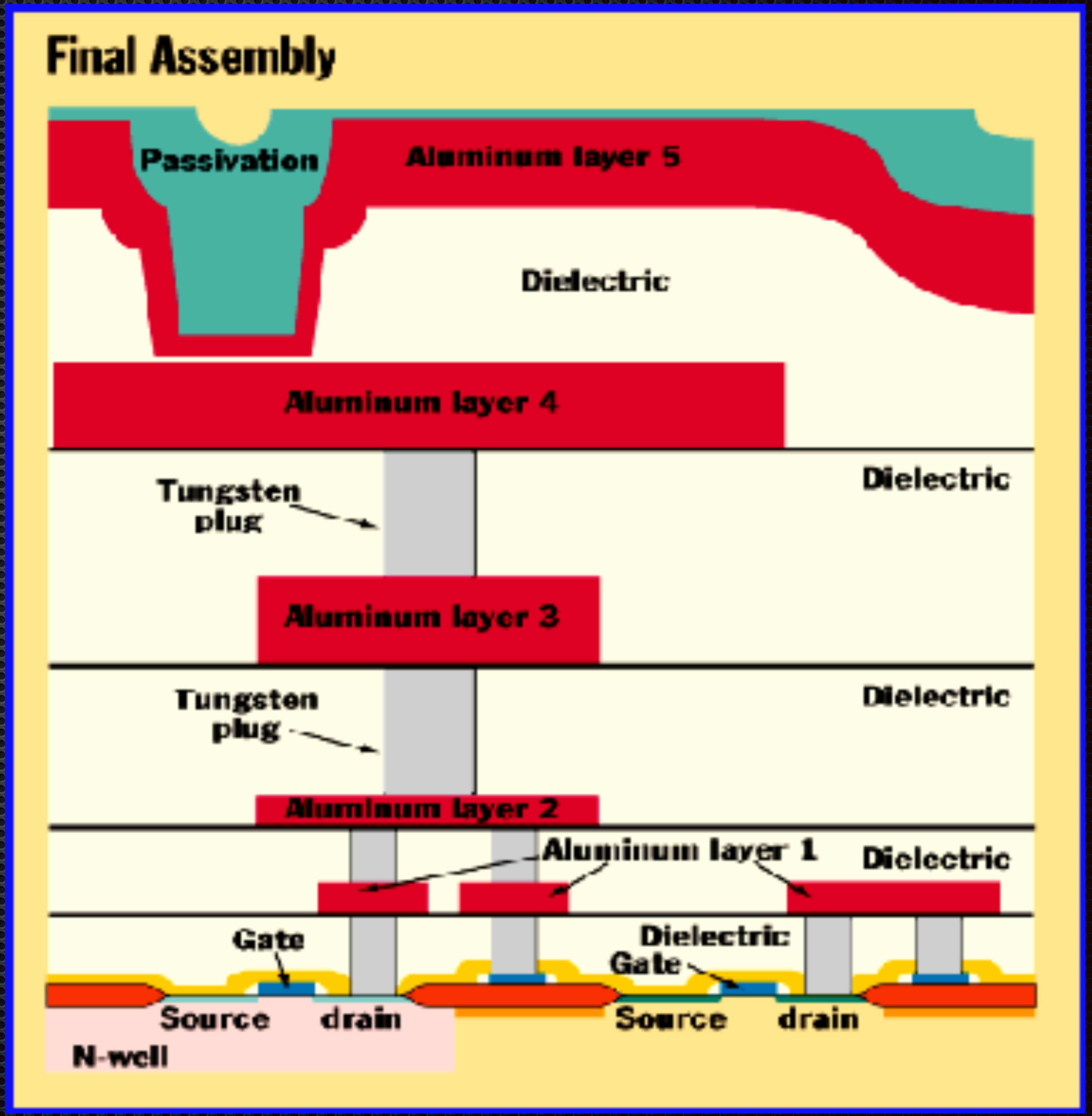
N-well PMOS Transistor

(0 = on)



Multiple Wire Layers

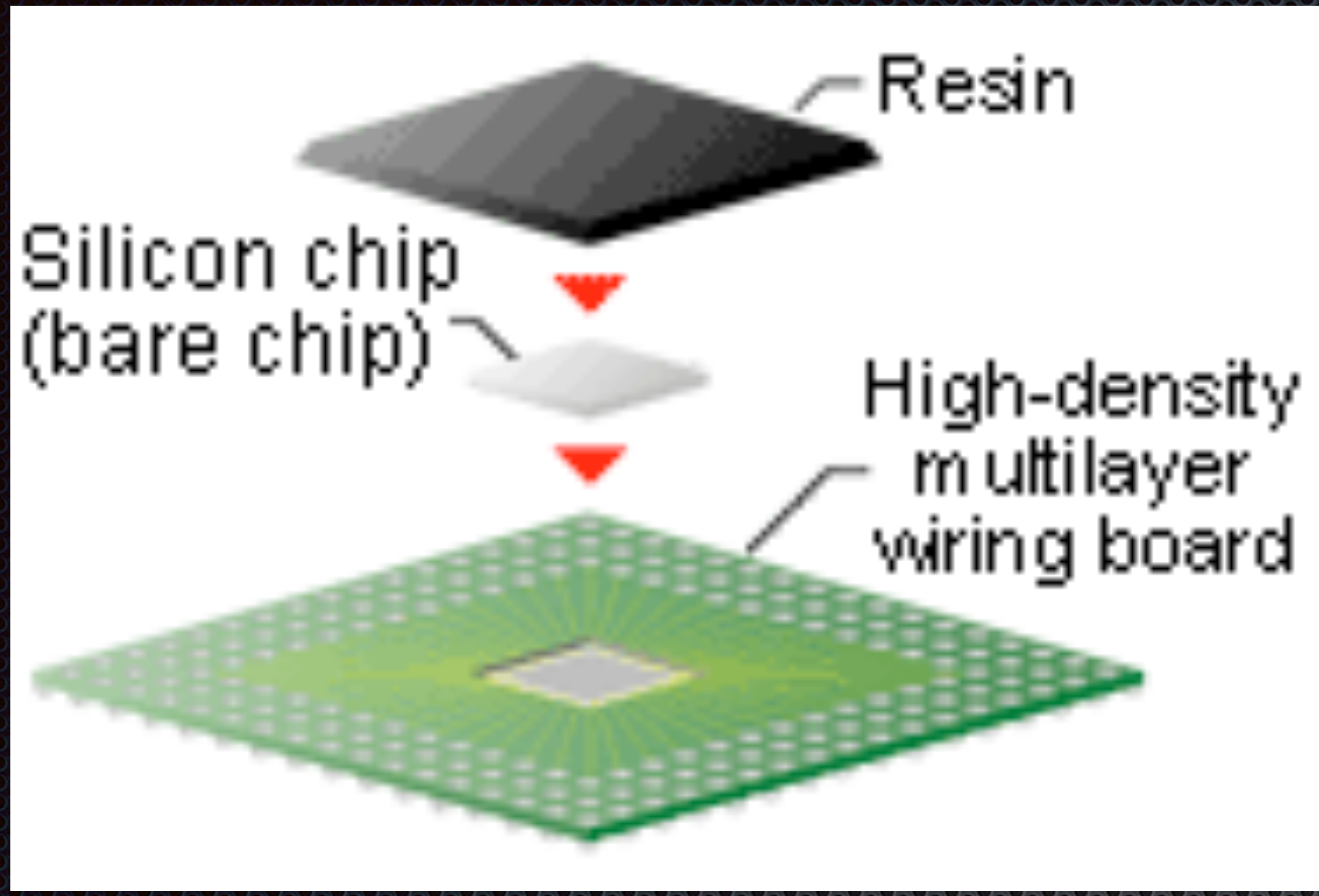
- Even with polishing, surface is uneven
- Higher layers must be thicker, and wires wider to be reliable. Reduces wiring density.
- Insulates heat-generating regions
- Up to 7 layers (4 in memory)

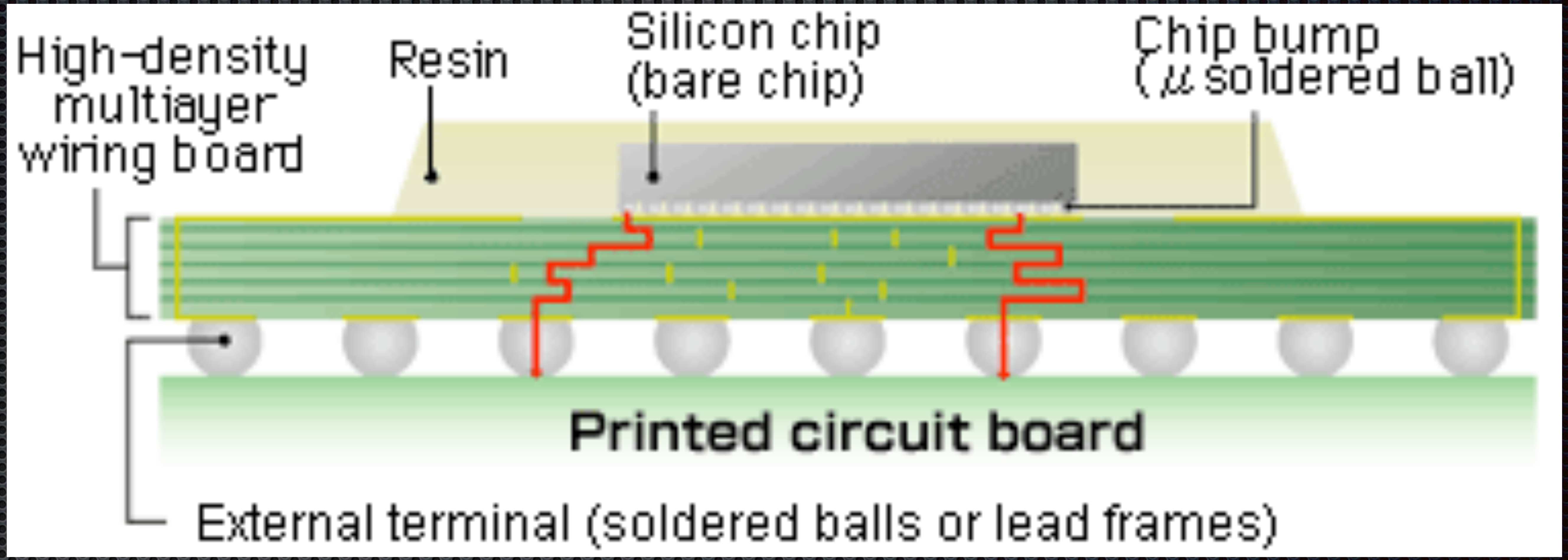


Bare Die Testing

- Optical inspection for defects
- Micro-probe tester contacts test pads, powers chip and runs test sequence
- Probes wear out, must be replaced
- Bad die are marked as rejects

Packaging





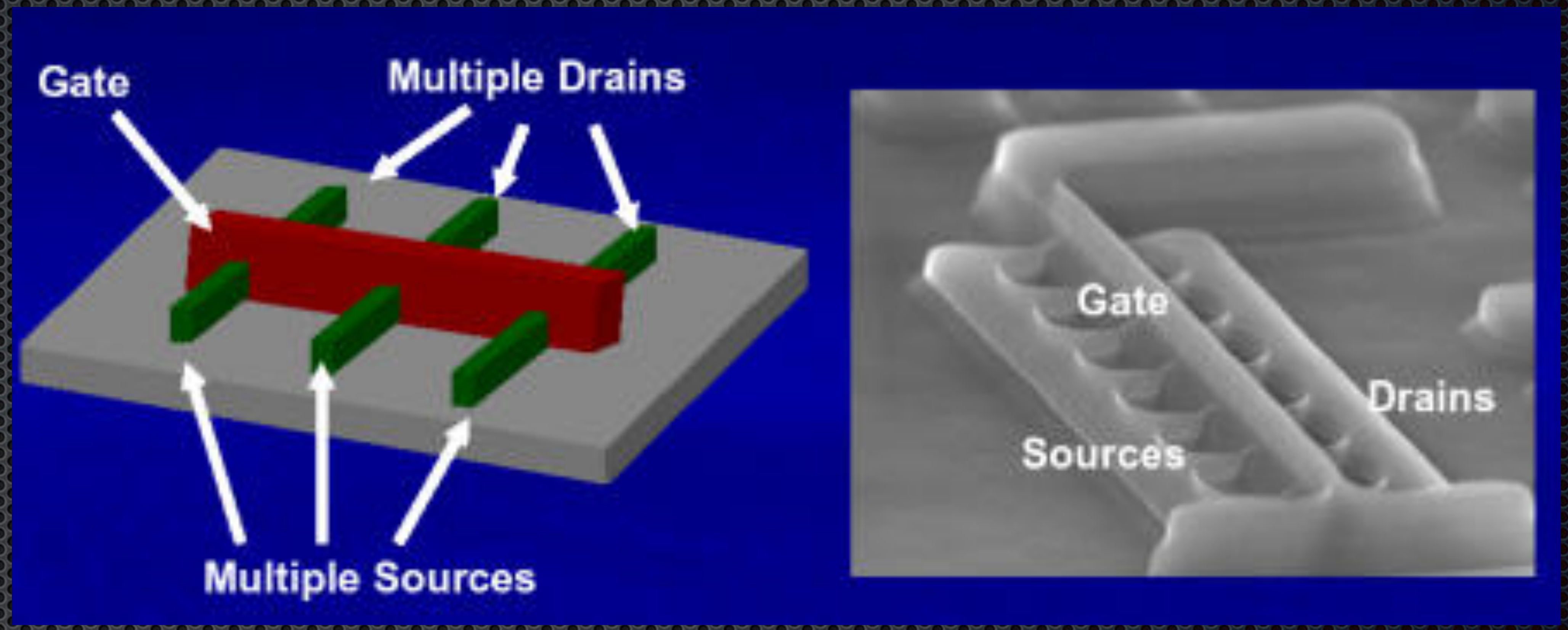
- Solder-ball or wire bonding
- Multi-level wiring in package
- Bare chip for heat sink contact in some cases

Packaged Chip Testing

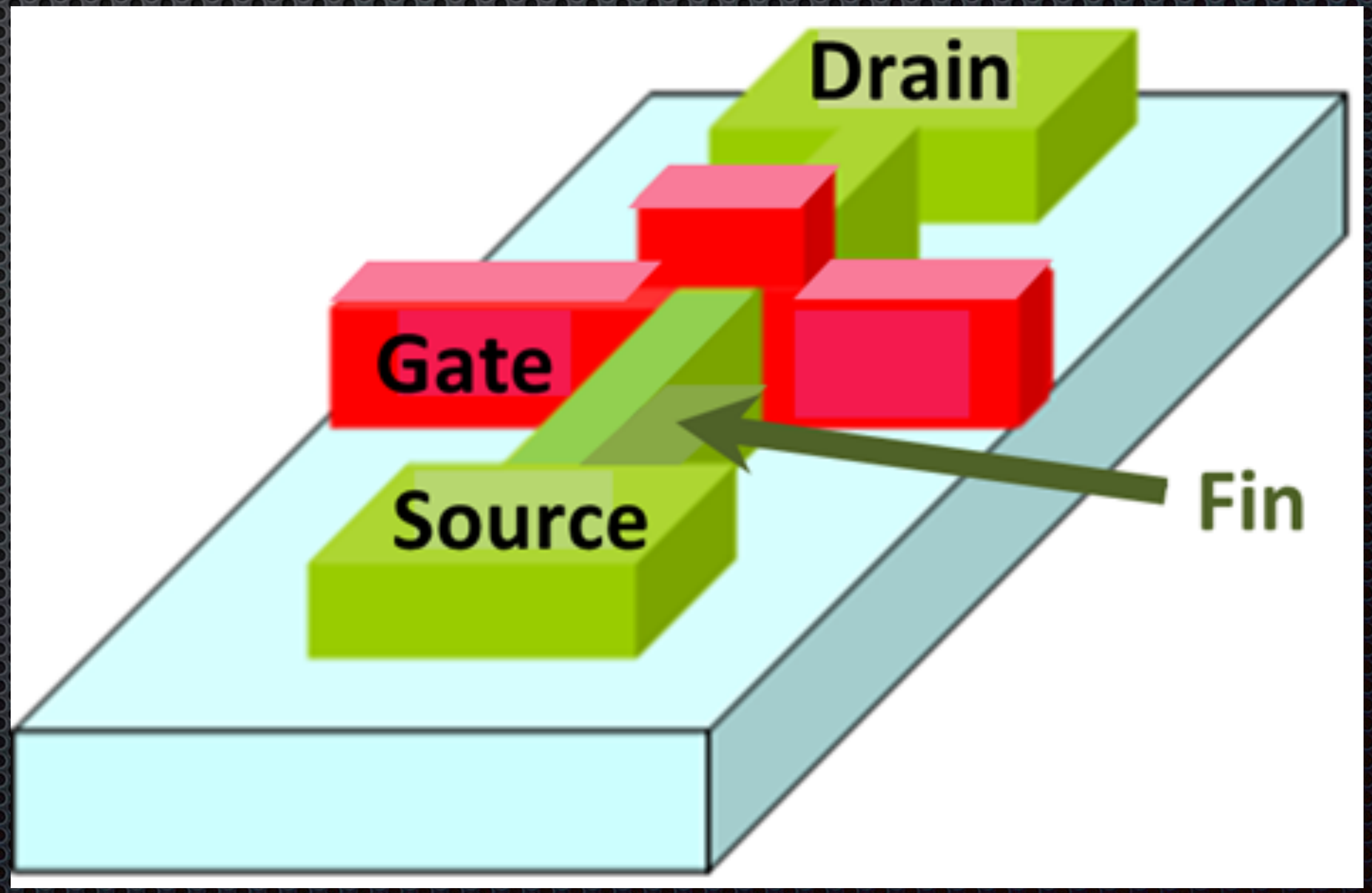
- Initial test (some fail in bonding)
- Performance grading test
- Power and thermal cycle tests
- Vibration tests (for some)
- Packages with bad chips recycled

Newer technologies

Intel Tri-gate
(in production)



Fin-FET



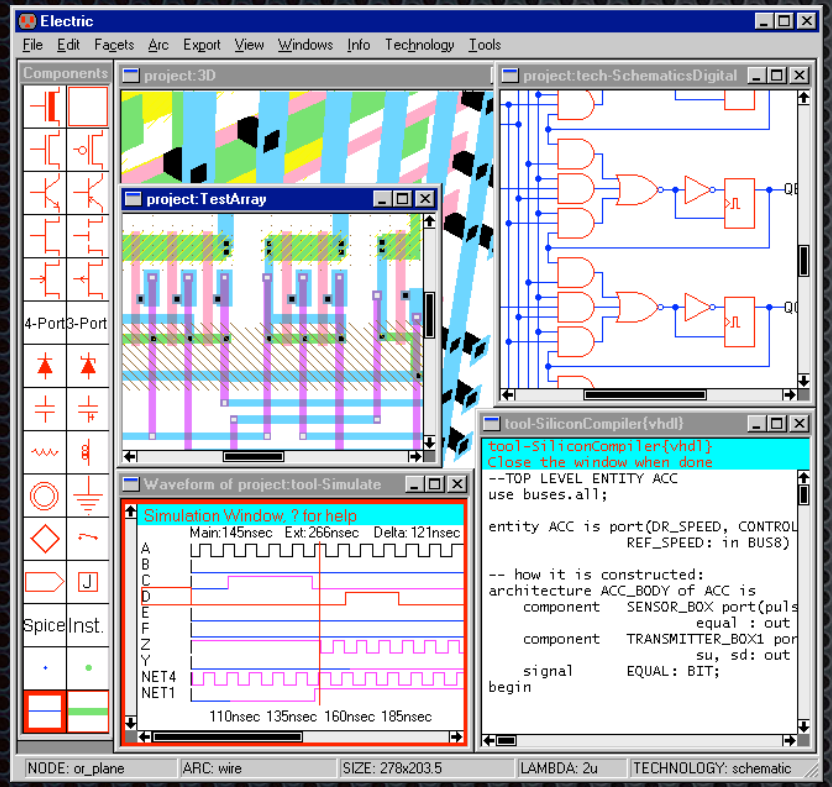
VLSI Cost Model

Estimating Cost of Chips and Systems for a Given Technology

Two Kinds of Cost

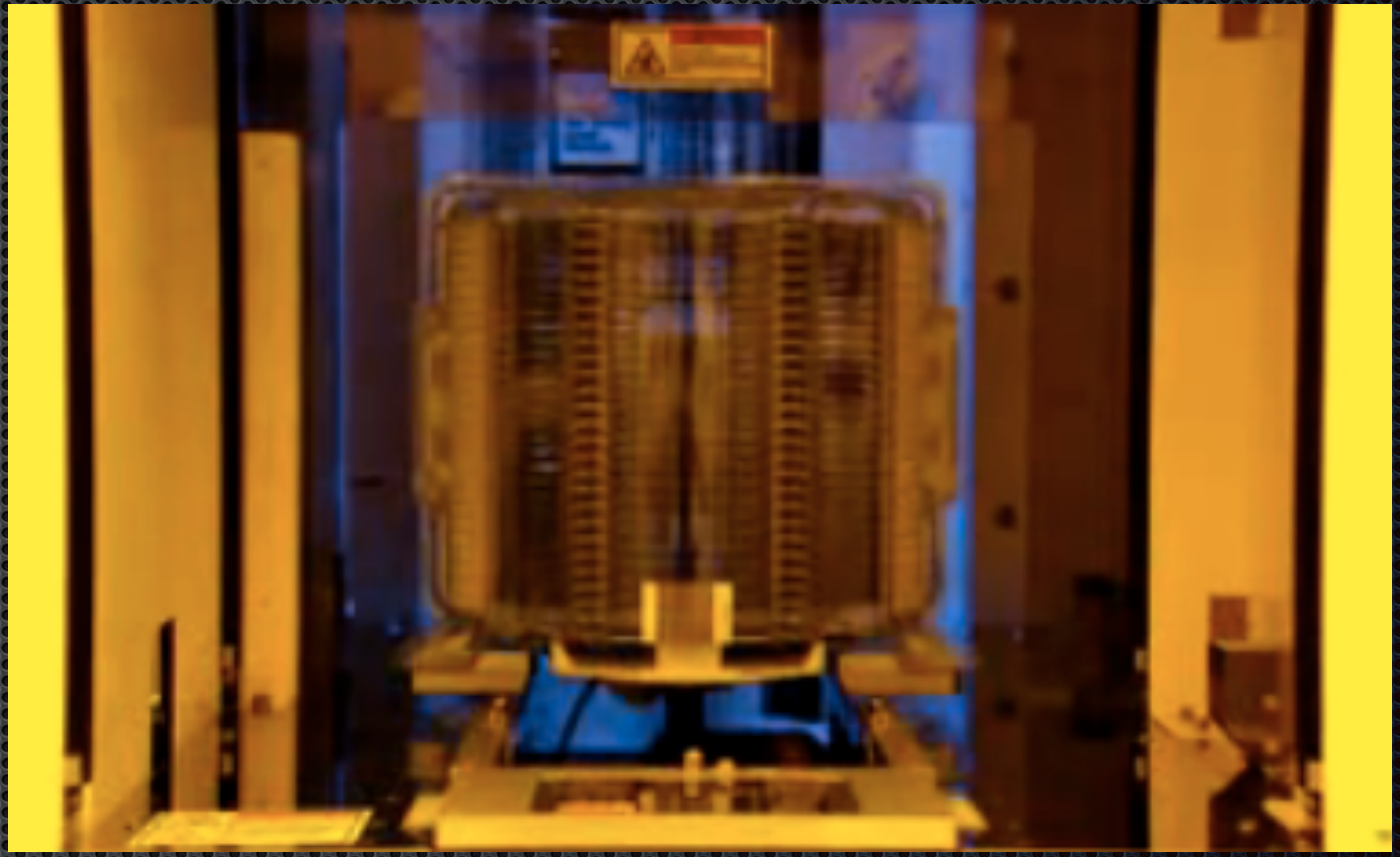
- Non-recurring (NRE)
- Recurring

Nonrecurring Cost



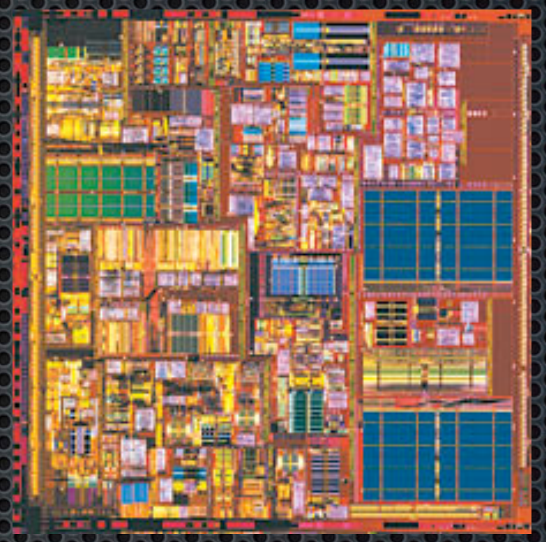
- Chip design ($800M),
- Plant capitalization ($8B),
- Mask set ($1.5M)

Recurring Cost

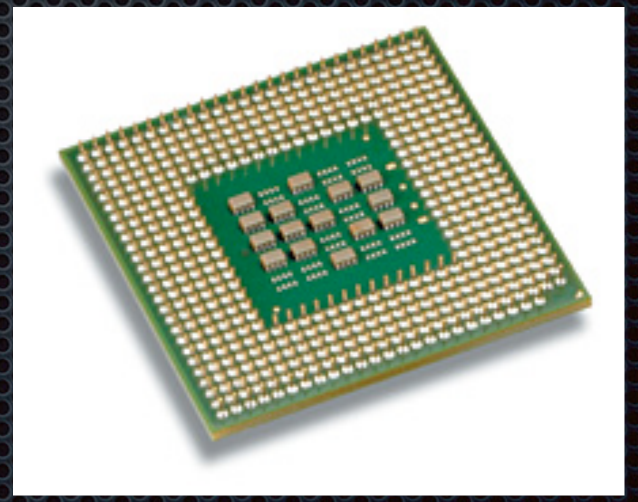


- Recurring: Manufacturing, packaging and testing, marketing, distribution, warranty, research, overhead, legal
- We will focus on manufacture, package, & test

Packaged Chip Cost



- $Cost_{package}$ and $Yield_{final}$ are given
- Other terms are computed



$$Cost_{IC} = \frac{Cost_{chip} + Cost_{test} + Cost_{package}}{Yield_{final}}$$

Cost of Chip

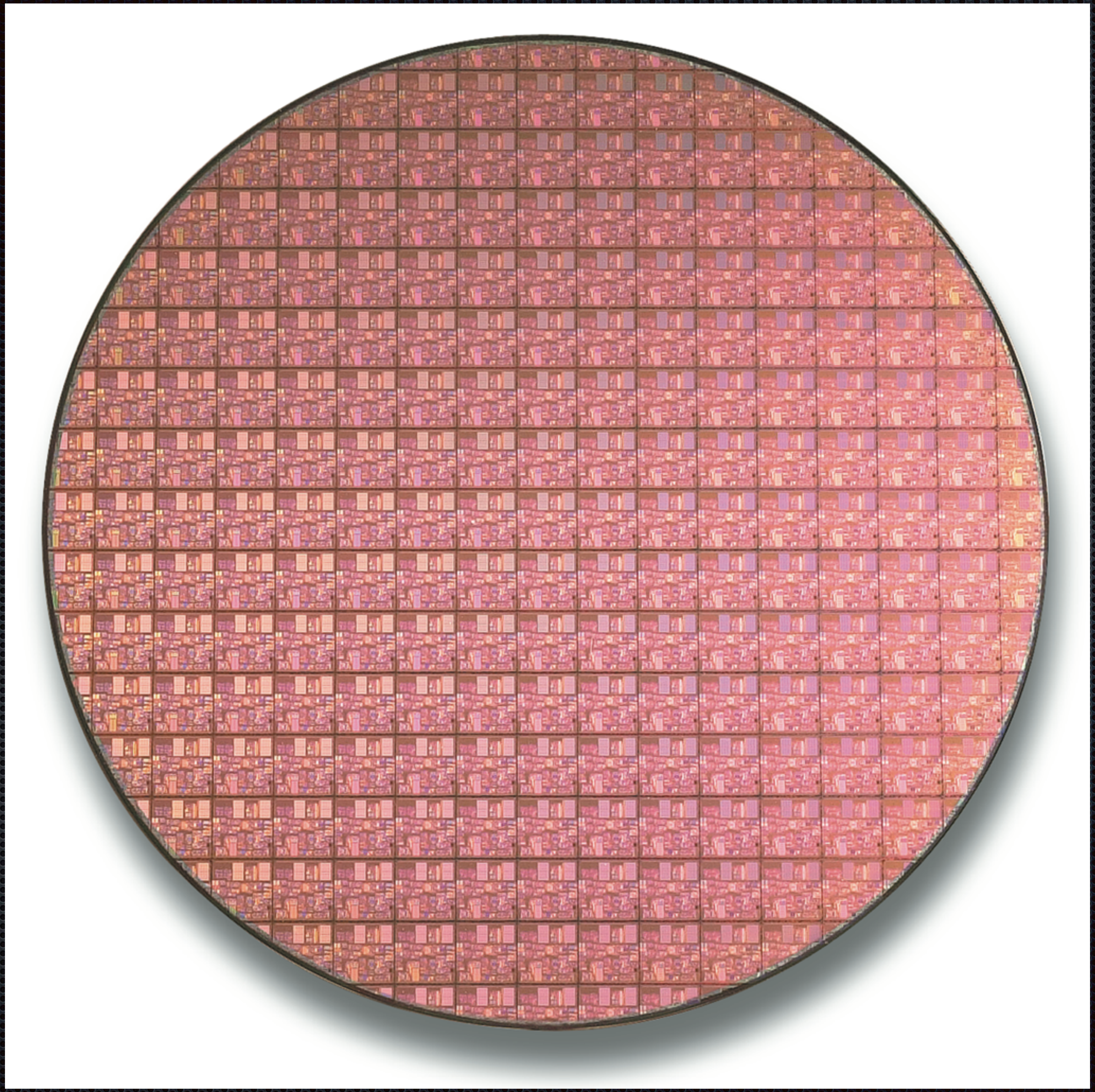
$$\text{Cost}_{\text{chip}} = \frac{\text{Cost}_{\text{wafer}}}{\text{Chips}_{\text{wafer}} \times \text{Yield}_{\text{die}}}$$

- $\text{Cost}_{\text{wafer}}$ is the cost of the finished wafer. Typically around $6000 to $8000
- Other terms are calculated

Chips Per Wafer

$$\text{Chips}_{\text{wafer}} = \left\lfloor \frac{\pi \times \left(\frac{\text{Diameter}_{\text{wafer}}}{2}\right)^2}{\text{Area}_{\text{chip}}} - \frac{\pi \times \text{Diameter}_{\text{wafer}}}{\sqrt{2 \times \text{Area}_{\text{chip}}}} - \text{TestSites}_{\text{wafer}} \right\rfloor$$

- Square peg in a round hole formula
- Chips per wafer less ones at edge



Chips Per Wafer

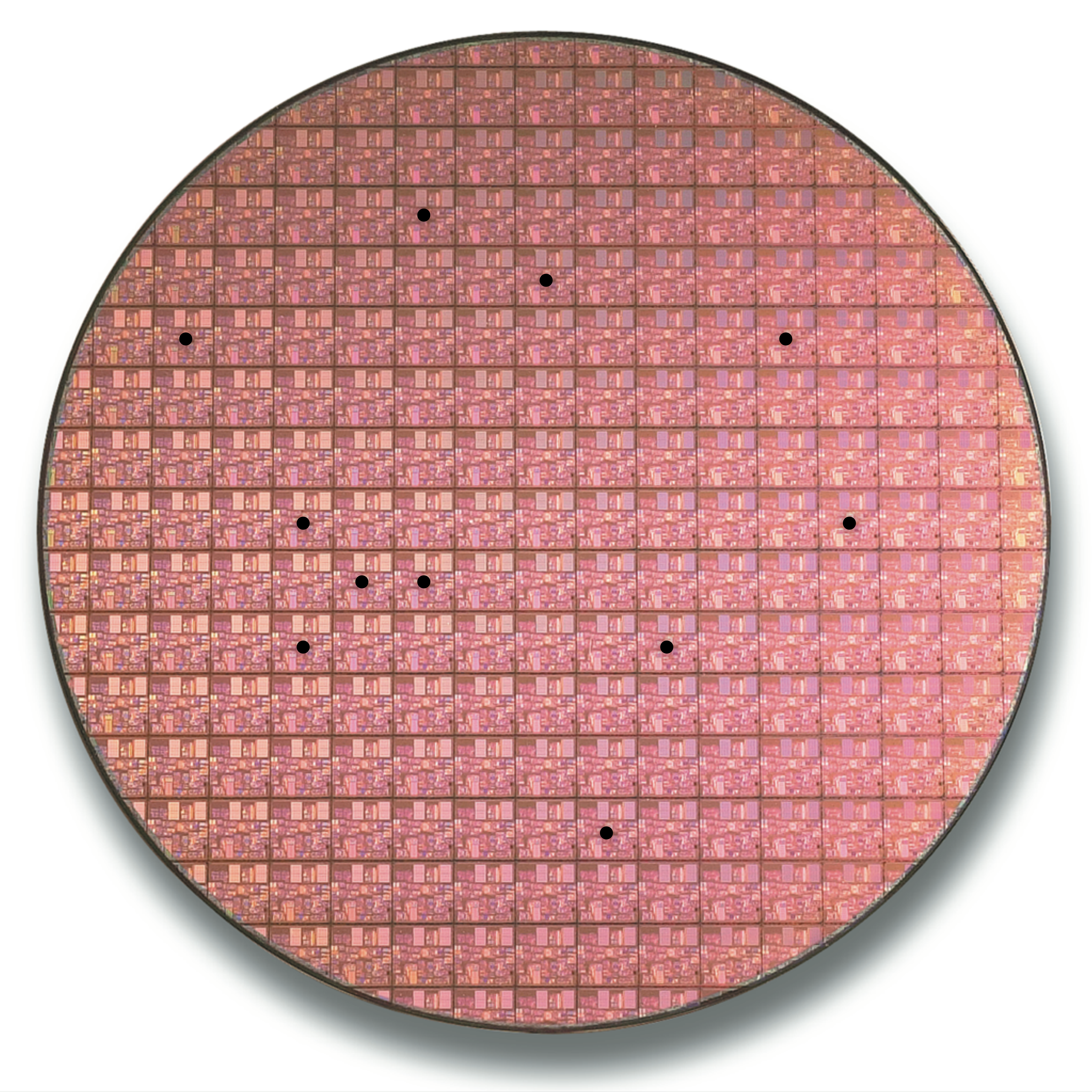
$$\text{Chips}_{\text{wafer}} = \left\lfloor \frac{\pi \times \left(\frac{\text{Diameter}_{\text{wafer}}}{2} \right)^2}{\text{Area}_{\text{chip}}} - \frac{\pi \times \text{Diameter}_{\text{wafer}}}{\sqrt{2 \times \text{Area}_{\text{chip}}}} - \text{TestSites}_{\text{wafer}} \right\rfloor$$

- Square peg in a round hole formula
- Chips per wafer less ones at edge

Die Yield

$$\text{Yield}_{\text{die}} = \text{Yield}_{\text{wafer}} \times \left(1 + \frac{\text{Defects}_{\text{unit-area}} \times \text{Area}_{\text{chip}}}{\text{P}}\right)^{-\text{P}}$$

- Fraction of good die. Depends on process.
- $\text{Defects}_{\text{unit-area}}$ and $\text{Yield}_{\text{wafer}}$ are given
- P is a process complexity factor



Die Yield

$$\text{Yield}_{\text{die}} = \text{Yield}_{\text{wafer}} \times \left(1 + \frac{\text{Defects}_{\text{unit-area}} \times \text{Area}_{\text{chip}}}{\text{P}}\right)^{-\text{P}}$$

- Fraction of good die. Depends on process.
- $\text{Defects}_{\text{unit-area}}$ and $\text{Yield}_{\text{wafer}}$ are given
- P is a process complexity factor

Test Cost

$$\text{Cost}_{\text{test}} = \frac{\text{Cost}_{\text{hour}} \times \text{Time}_{\text{test}}}{\text{Yield}_{\text{die}}}$$

- Testers are expensive to operate
- Cost must be amortized over good chips

Cost Practice

- $Cost_{wafer}$ = \$7000
- $Diameter_{wafer}$ = 300mm
- $Area_{chip}$ = 13.5 X 19.6 mm
- $Defects_{unit-area}$= 0.5 per cm^2
- $Yield_{wafer}$ = 0.999
- $Yield_{final}$ = 0.97
- P = 4.3
- $Cost_{hour}$ = \$1000
- $Time_{test}$ = 10 seconds
- $TestSites_{wafer}$ = 0
- Package Cost = \$12