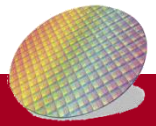
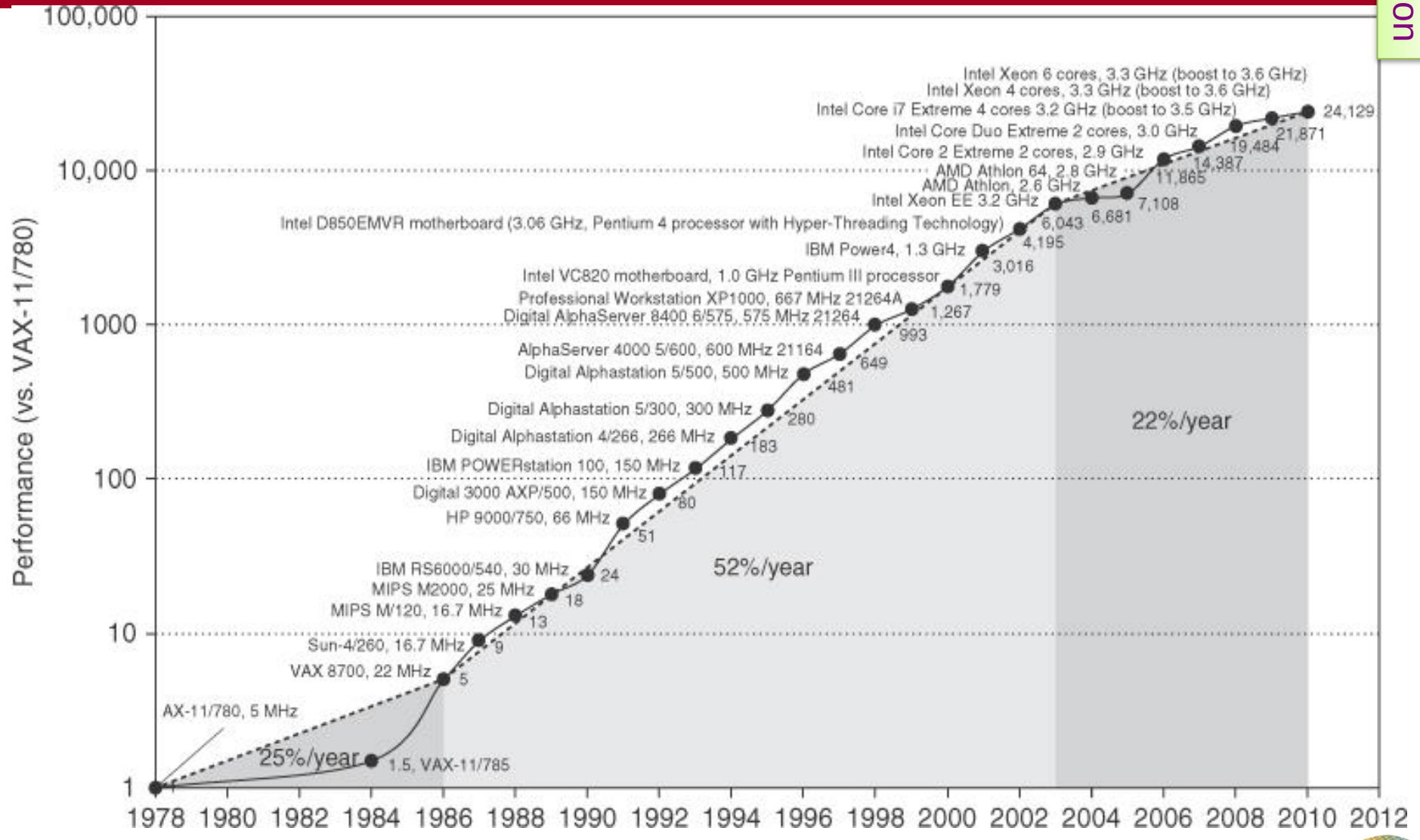
Fundamentals of Quantitative Design and Analysis

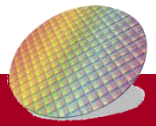


Trend of Computer Performance



Computer Technology

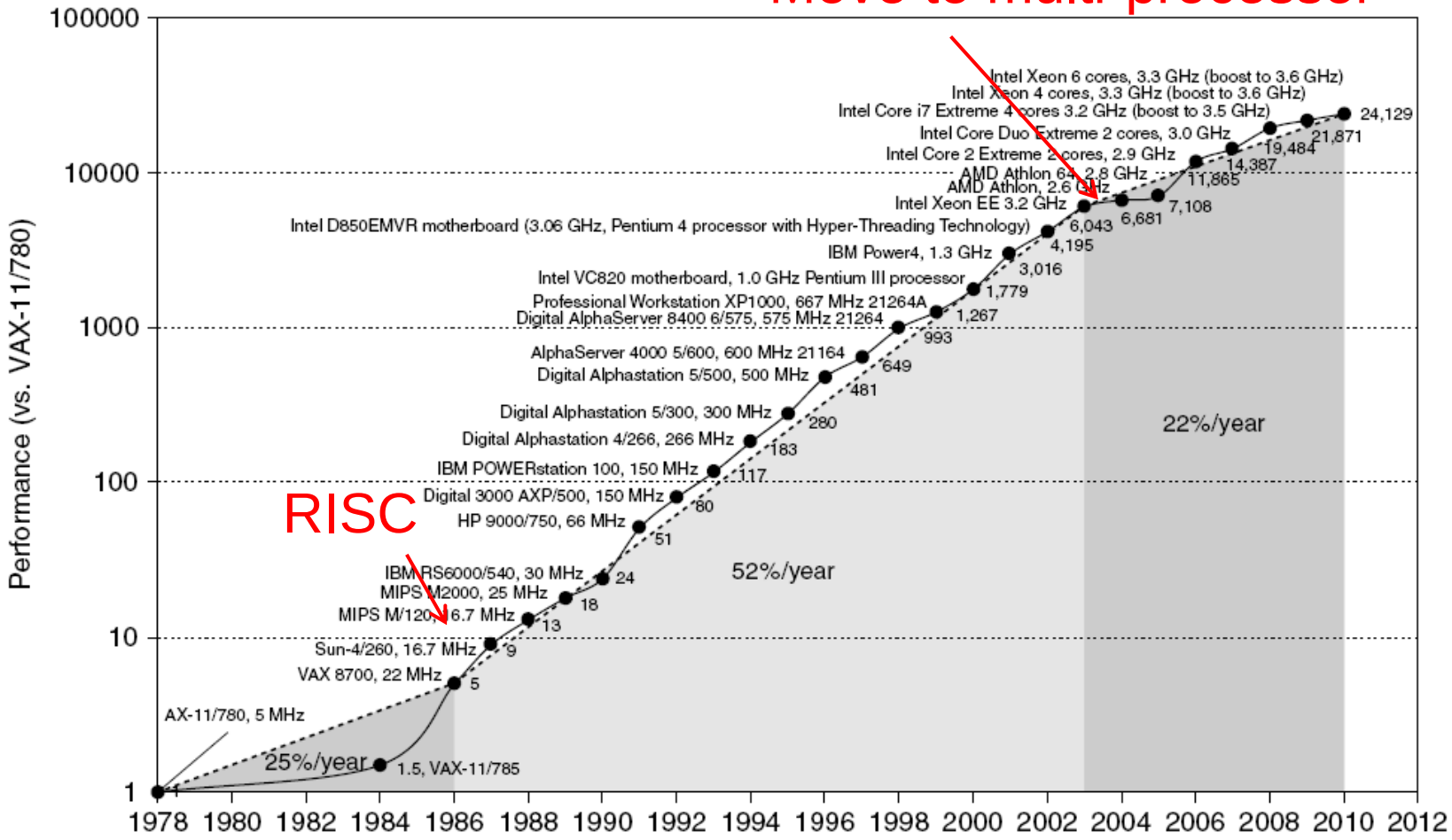
- Performance improvements:
  - –Improvements in semiconductor technology
    - Feature size, clock speed
  - –Improvements in computer architectures
    - Enabled by high-level language compilers (no need to consider object code compatibility) and vender-independent OS (e.g. UNIX, lower cost of new architecture)
    - Lead to RISC architectures
  - –Together have enabled:
    - Lightweight computers
    - Productivity-based managed/interpreted programming languages



Single Processor Performance

Power and ILP constraint

Move to multi-processor

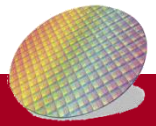


Technology driven

Technology & computer architectural and organizational improvement

Current Trends in Architecture

- Cannot continue to leverage Instruction-Level parallelism (ILP) (Chapter 3)
  - Single processor performance improvement ended in 2003
- New models for performance:
  - Data-level parallelism (DLP) (Chapter 4)
  - Thread-level parallelism (TLP) (Chapter 5)
  - Request-level parallelism (RLP) (Chapter 6)
- These require explicit restructuring of the application
  - Multithread programming
  - Parallel programming
  - Distributed programming



Application Level vs. Architectural Level Parallelism

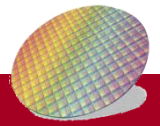
Application Level Parallelism

Data-Level Parallelism (DLP)
- Data can be operated at the same time

Task-Level Parallelism (TLP)
- Task that can operate independently and largely in parallel

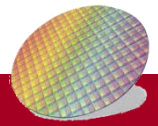
Architectural parallelism

Instruction-Level Parallelism (ILP)
Vector architectures/Graphic Processor Units (GPUs)
Thread-Level Parallelism
Request-Level Parallelism



Application Level vs. Architectural Level Parallelism

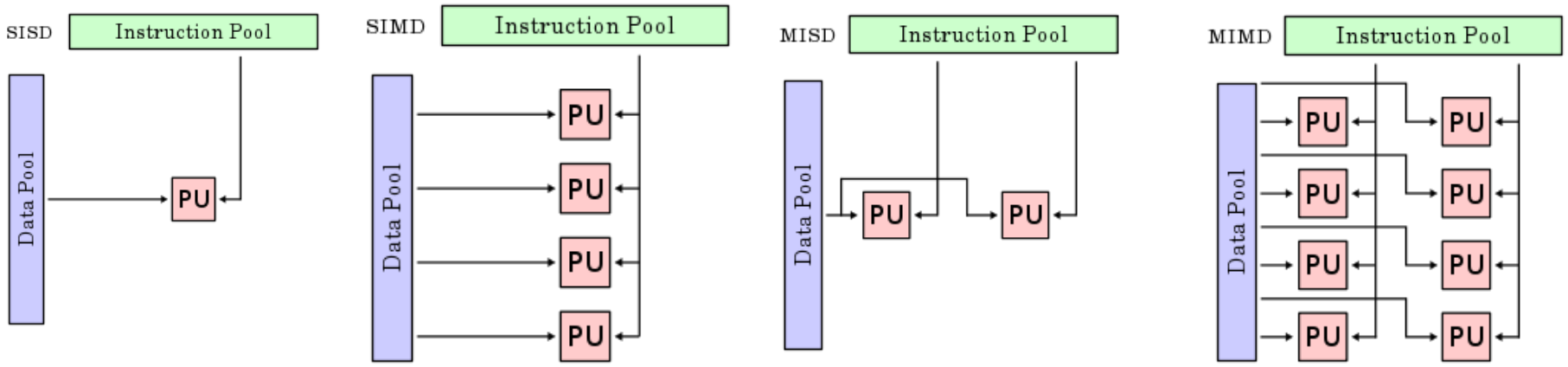
Architectural parallelism	Achieve DLP or TLP
Instruction-Level Parallelism (ILP)	Exploit DLP by pipeline, or speculative execution
Vector architectures/Graphic Processor Units (GPUs)	Exploit DLP by single instruction multiple data (SIMD)
Thread-Level Parallelism	Exploit either DLP or TLP (Task-level parallelism) by tightly coupled parallel threads
Request-Level Parallelism	Exploit Task-level parallelism mostly by largely decoupled tasks



Flynn's Taxonomy

Proposed by Michael J. Flynn's (Stanford University Professor)

	Single Instruction Stream	Multiple Instruction Stream
Single Data Stream	SISD	MISD (No commercial implementation)
Multiple Data Stream	SIMD includes • Vector architectures • Multimedia extensions • Graphics processor units	MIMD • Tightly-coupled MIMD • Loosely-coupled MIMD



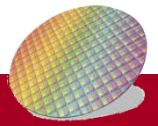
Source: Wiki

Defining Computer Architecture

- Old vs. Real Computer Architecture

"Old" view of computer architecture:
Instruction Set Architecture (ISA) design

- Class of ISA
- Memory addressing
- Addressing mode
- Type and sizes of operands
- Operations
- Control flow instructions
- Instruction encoding



Defining Computer Architecture

- Old vs. Real Computer Architecture

"Real" computer architecture:

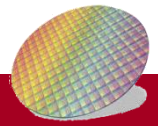
- Specific requirements of the target machine
- Design to maximize performance within constraints: cost, power, and availability

"Old" view of computer architecture:

Instruction Set Architecture (ISA) design

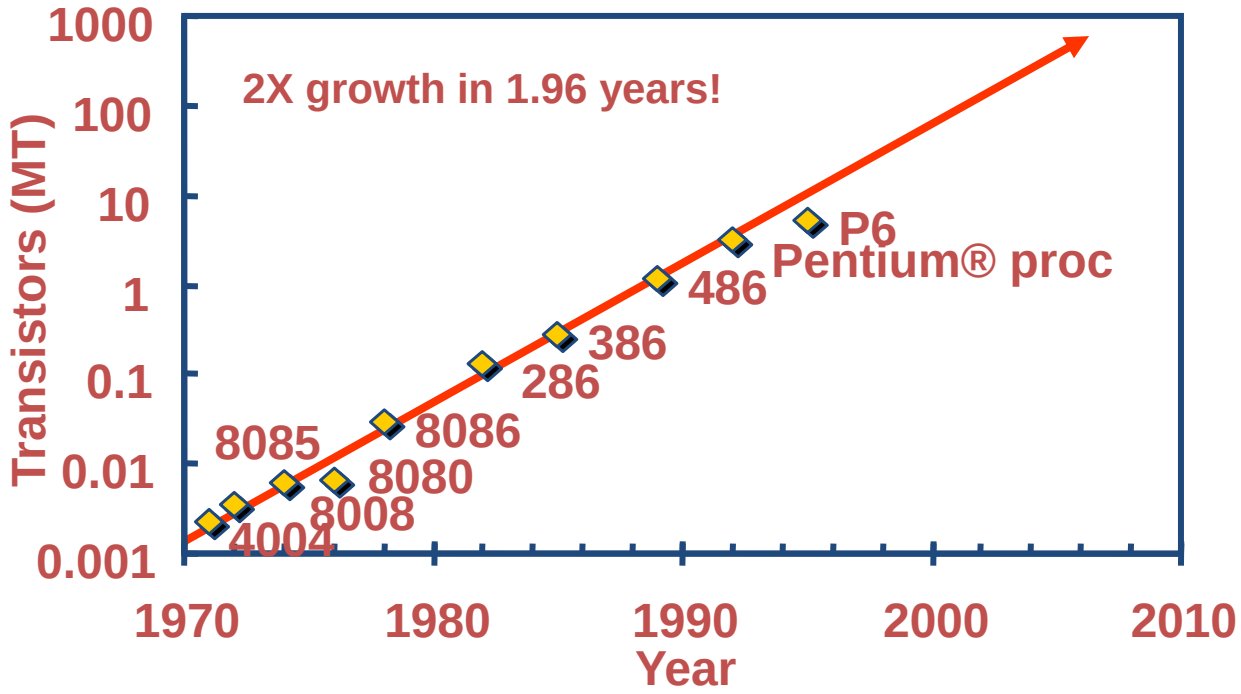
- registers, memory addressing, addressing modes, instruction operands, available operations, control flow instructions, instruction encoding

Computer Architecture includes ISA, microarchitecture, hardware



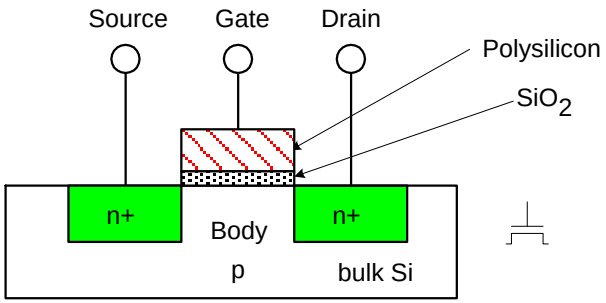
Moore's Law

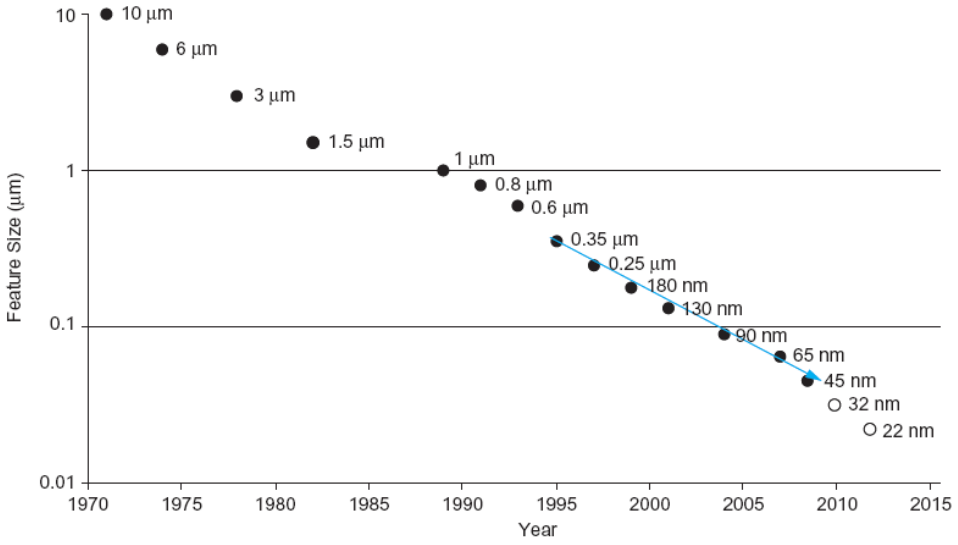
- In 1965, Gordon Moore predicted that the number of transistors that can be integrated on a die would double every 18 to 24 months (i.e., grow exponentially with time).



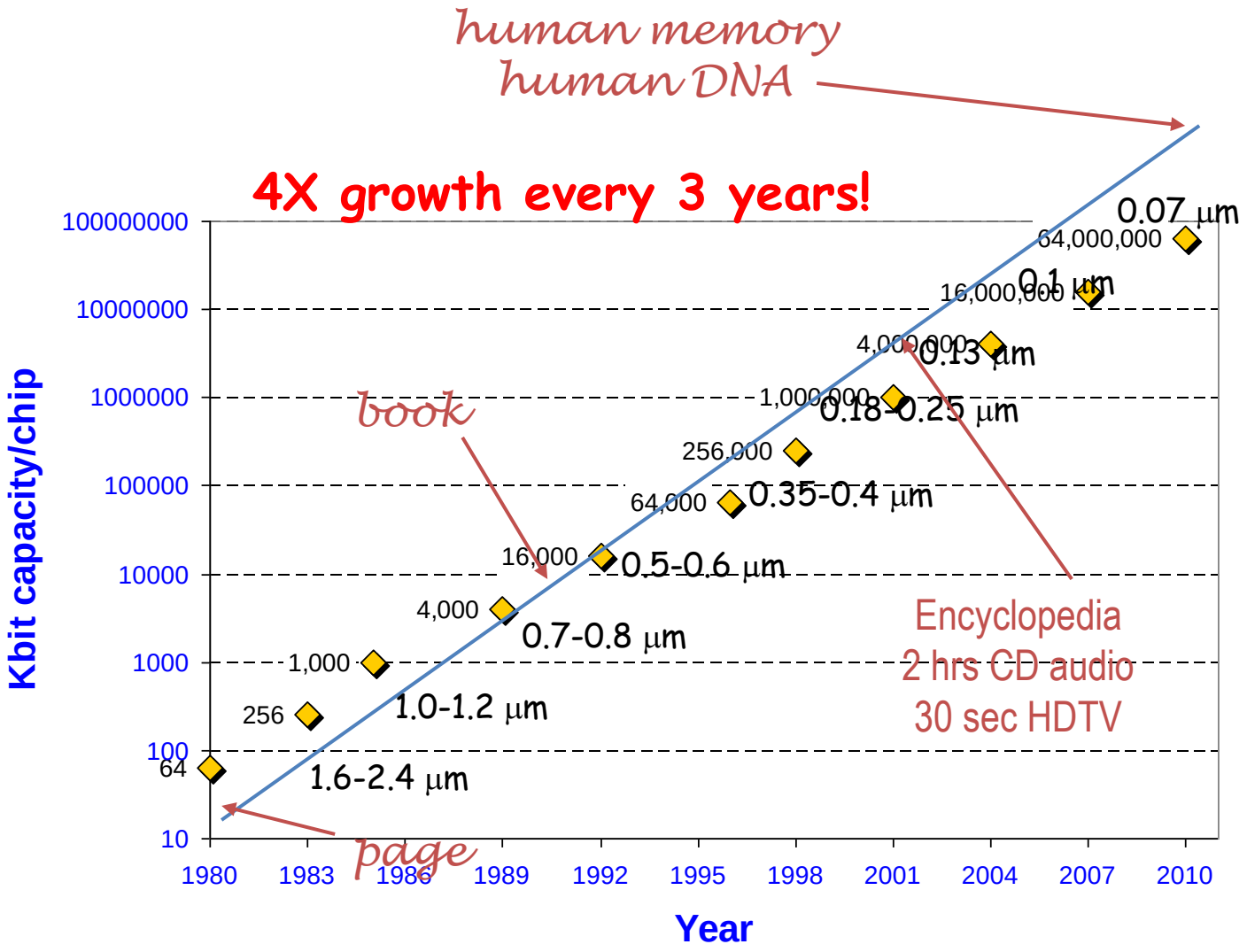
Feature Size

- Feature Size
  - Minimum size of transistor or wire in x or y dimension
  - Minimum feature size shrinking 30% every 2-3 years
  - Transistor Performance improves about linearly with decreasing feature size



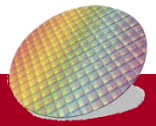


Evolution in DRAM Chip Capacity

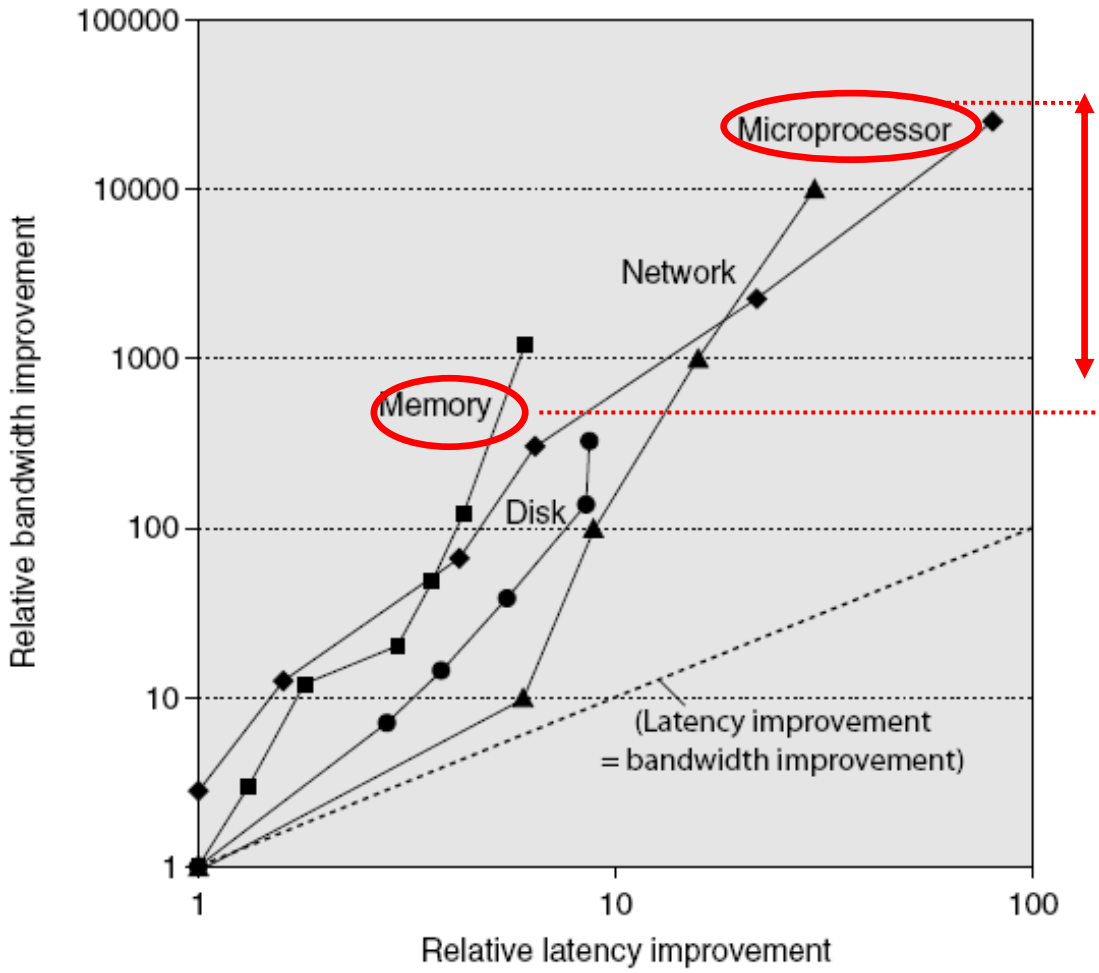


Bandwidth vs. Latency

- Bandwidth or throughput
  - Total work done in a given time
  (e.g. MB/s for disk transfer)
  - 10,000-25,000X improvement for processors
  - 300-1200X improvement for memory and disks
- Latency or response time
  - Time between start and completion of an event
  - 30-80X improvement for processors
  - 6-8X improvement for memory and disks
  - Much less than bandwidth



Bandwidth and Latency



Gap is increasing

Rule of thumb: bandwidth grows by at least the square of the improvement in latency

$$\Delta BW \propto \Delta Latency^2$$

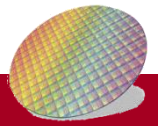
Log-log plot of bandwidth and latency milestones

Power and Energy: A system perspective

- Power consumption: how much energy is consumed per operation (or per unit time) and how much heat the circuit dissipates
- From the viewpoint of a system designer, there are three primary concerns:
  - Peak power: maximum power.
  - Sustained power consumption, widely called the thermal design power (TDP)
  - The third factor for both (Designers and Users) is Energy and Energy efficiency.

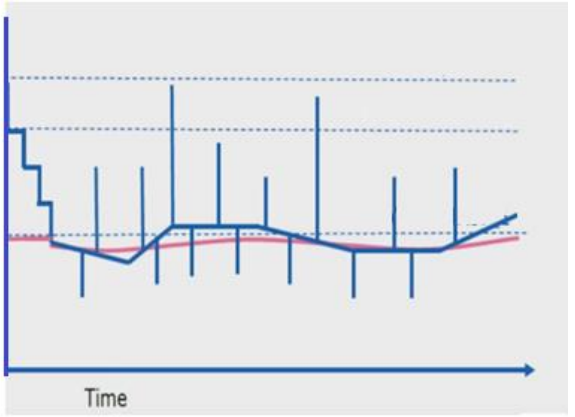
Power: is simply Energy per Unit time.
1 watt = 1 Joule/second
1 Joule = $6.24 * 10^{18}$ eV

- Energy is power integrated over time. Energy = Power * time
- Energy is the capacity to do work.
- Power is the rate at which work is done, or energy is transmitted.

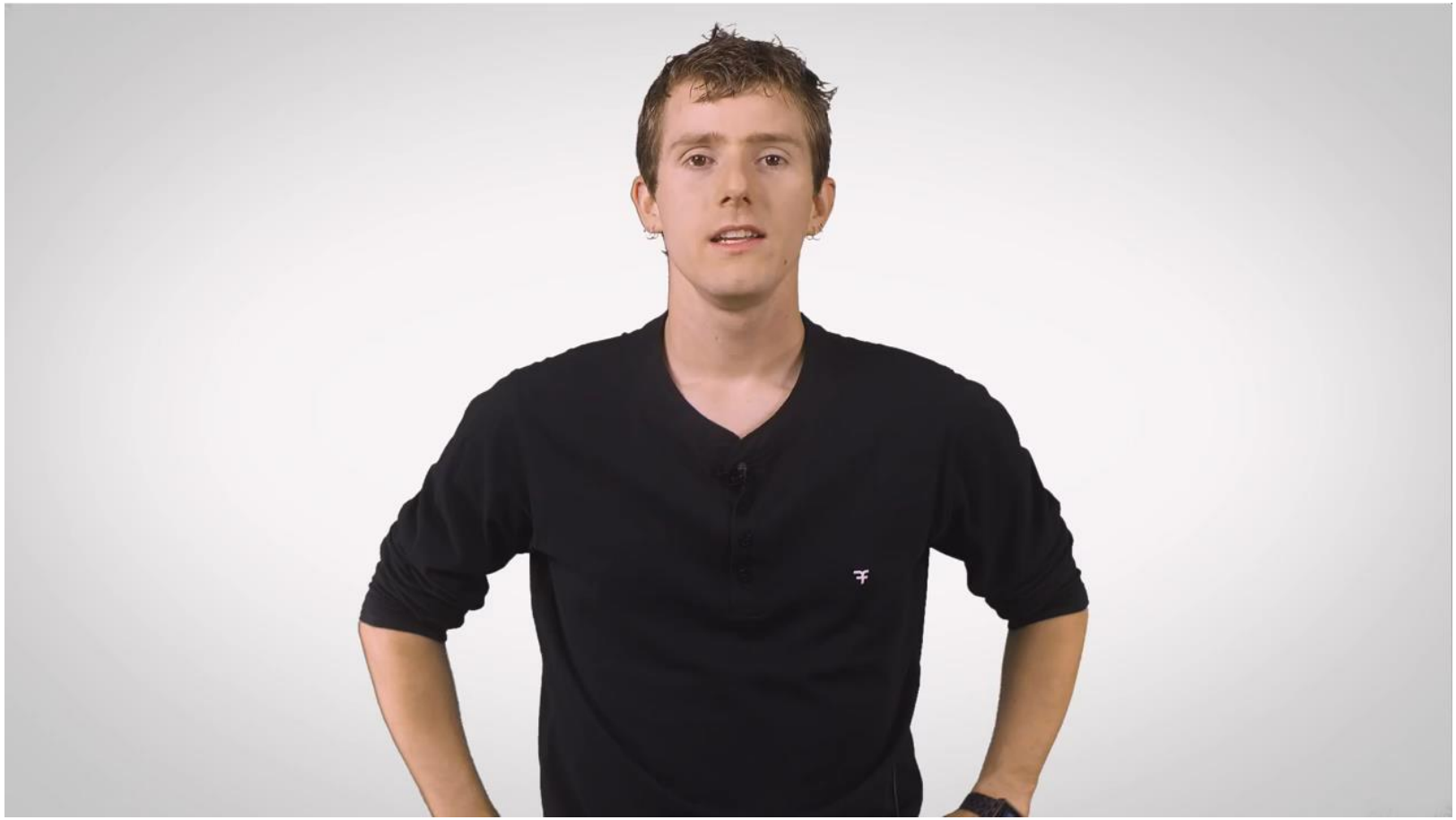


Peak power and Sustained power (TDP)

- Peak power: maximum power consumption
  - Used to determine supply line sizing, packaging
- Average Power : power consumed during a given computation
  - Used to determine battery lifetime
- Sustained Power consumption: also called Thermal Design Power (TDP)
  - Not a direct measure of power consumption
  - Maximum amount of thermal power (heat) can generate
  - Normally, 1.5X of Average Power
  - Determine an appropriate cooling systems
  - If cooling system has heat removal efficiency lower than Processor's TDP => overheat => processor fails
- See Video in the next slide



TDP



Start at 1:59

Energy and Energy efficiency ---Dynamic Energy and Power

- Dynamic energy: Transistor switch from 0 -> 1 or 1 -> 0

$$Energy_{dynamic} = \frac{1}{2} \times Capacitive\ load \times Voltage^2$$

$$E_{dyn} = \frac{1}{2} \times CV^2$$

- Dynamic power

$$Power_{dynamic} = \frac{1}{2} \times Capacitive\ load \times Voltage^2 \times Freq$$

$$P_{dyn} = \frac{1}{2} \times CV^2 f$$

- To reduce power, we can reduce Capacitive load, Voltage and Freq
- Reducing clock rate reduces power, not energy

Energy and Energy efficiency ---Example

- Example: Some microprocessors today are designed to have adjustable voltage, so a 15% reduction in voltage may result in a 15% reduction in frequency. What would be the impact on dynamic energy and on dynamic power?

$$\frac{E_{new}}{E_{old}} = \frac{(Voltage * 0.85)^2}{Voltage^2} = 0.72$$

$$\frac{P_{new}}{P_{old}} = 0.72 \times \frac{(Freq * 0.85)}{(Freq)} = 0.61$$

Energy is 72% of the original, and Power is 61% of the original

Energy and Energy efficiency ---Power-Delay Product

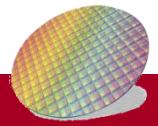
- Power-delay product (PDP)
  - Product of power and delay
  - PDP = P * t_p (i.e., power * delay)
  - PDP measures the energy consumed per switching event
- This metric consider power and delay, which is actually Energy
- Normally, a design with lower PDP is better

e.g. Design A consumes 50W to finish a task in 10s. Design B consumes 60W to finish the same task in 8s. Which design is better?

$PDP_A = 50 * 10 = 500$

$PDP_B = 60 * 8 = 480$

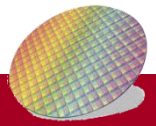
B has low PDP



Example

- Example: Processor A may have a 20% higher average power consumption than Processor B, but A executes the task in only 70% of the time needed by B, which processor is better in term of energy consumption?

Answer: A, its Energy consumption $=1.2 \times 0.7 = 0.84$
which is clearly better



Power-Delay Product may be misleading

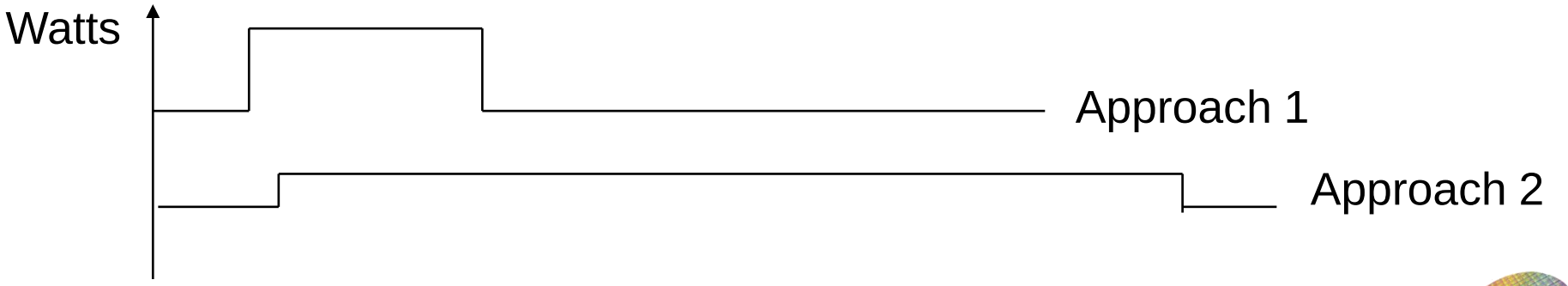
- PDP is useful, but it may be misleading => A design with low PDP may be a design with low power but has very LONG delay

e.g. Design A consumes 0.1W to finish a task in 1000s. Design B consumes 60W to finish the same task in 8s. Which design is better?

$PDP_A = 0.1 * 1000 = 100$

$PDP_B = 60 * 8 = 480$

A has low PDP, but runtime may be unacceptable



A better Metric: Energy-Delay Product

- Energy-delay product (EDP) = Energy * Delay

  =PDP * Delay = Power * $Delay^2$

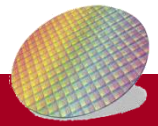
- A metric that accounts for both energy and performance
  - Weight performance more heavily than PDP

e.g. Design A consumes 50J to finish a task in 10s. Design B consumes 60J to finish the same task in 8s. Which design is better?

EDP_A= 50 *10 =500

EDP_B = 60 *8 =480

B is better



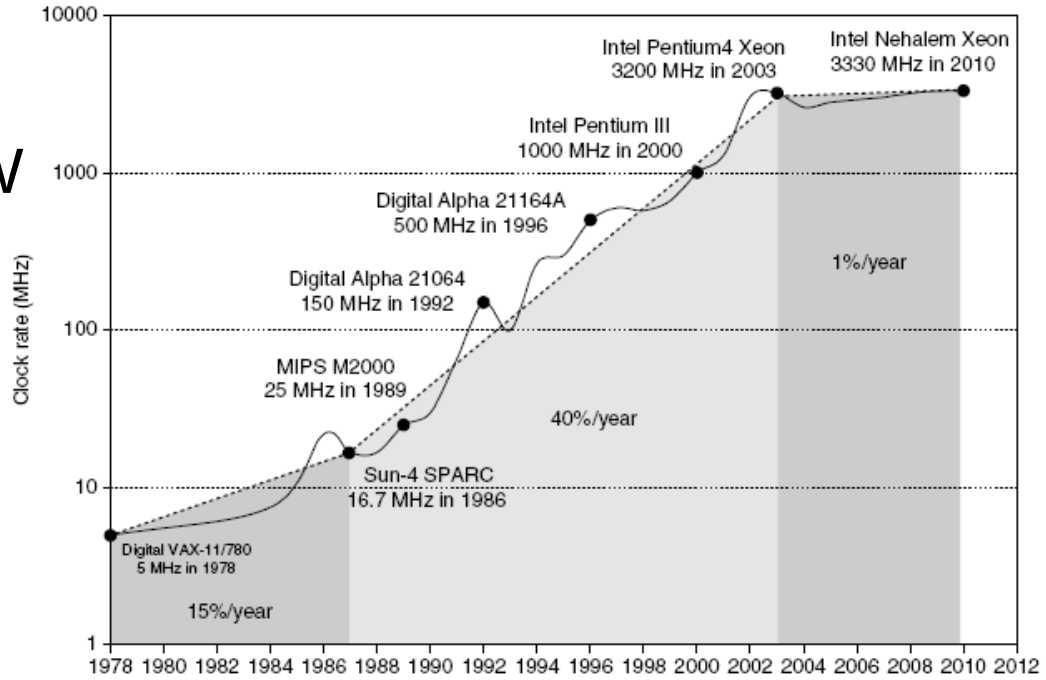
Power as design constraint

Intel 80386 ~ 2 W

3.3 GHz Intel Core i7 130 W

Heat must be dissipated from 1.5 x 1.5 cm chip

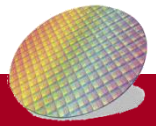
With increased power density, it is challenging to remove heat and keep the processor cool.



- Power becomes a critical design constraint
  - Choosing processor and cooling system

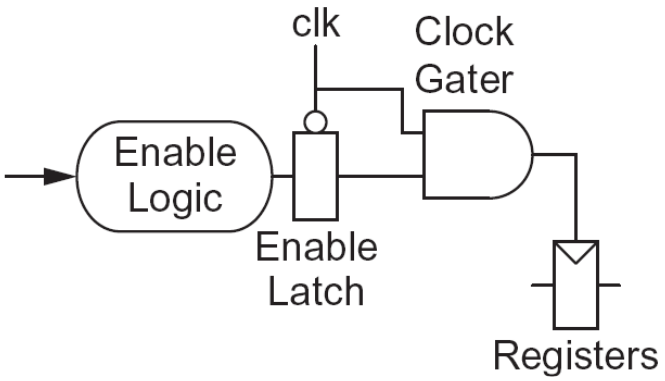
Techniques to Reduce Power

- Techniques for reducing power:
  - Do nothing well (Clock gating)
  - Dynamic Voltage-Frequency Scaling (DVFS)
  - Make typical case energy efficient
  - Overclocking, turning off cores



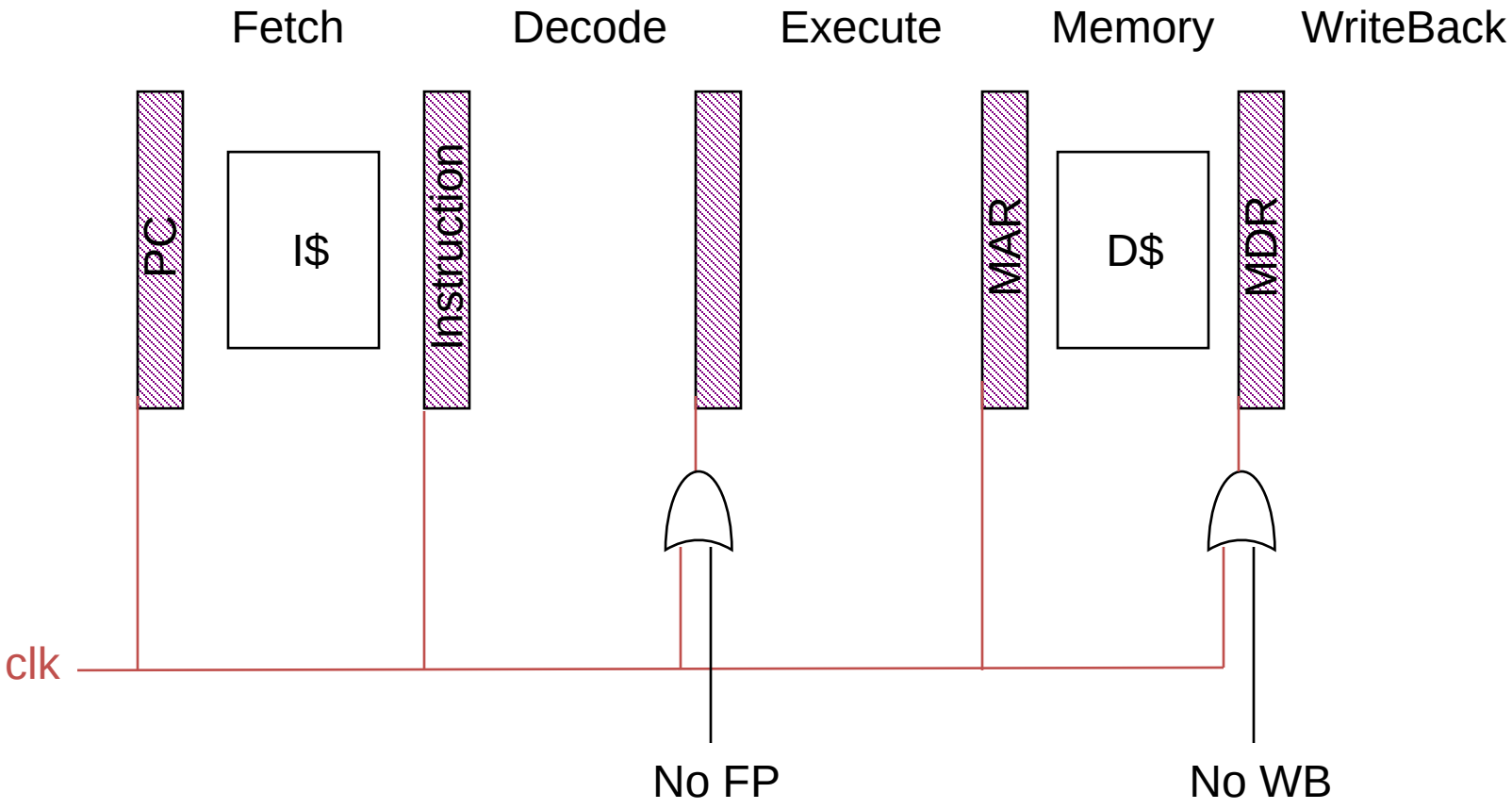
Technique 1 - Clock Gating

- Turn off the clock to registers in unused blocks
  - Reduce clock activity
  - Eliminates all switching activity in the block
- Overhead: Need to determine whether circuits will be used
  - Turn off if the circuits if they are idle



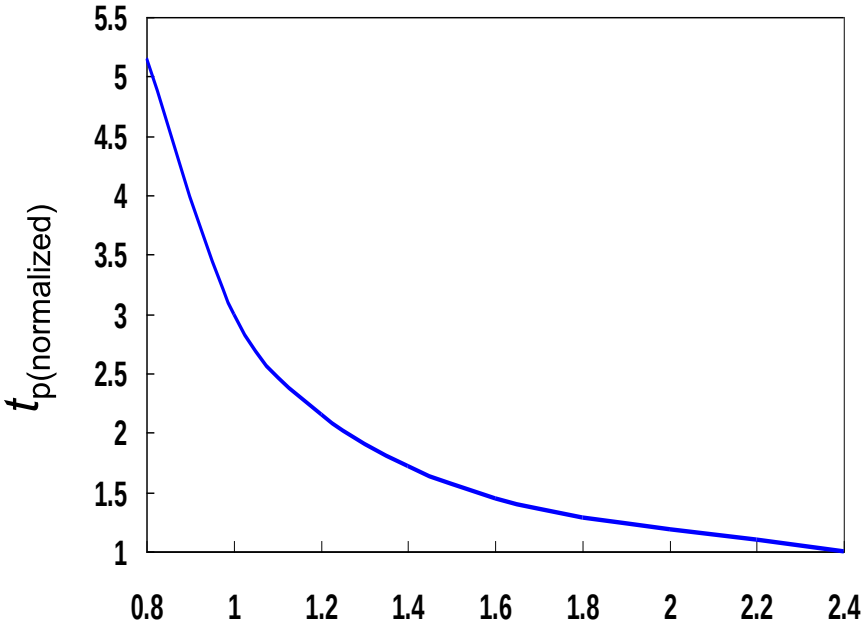
Clock Gating in a Pipelined Datapath

- For idle units (e.g., floating point units in Exec stage, WB stage for instructions with no write back operation)



Recall: Dynamic Power as a Function of V_{DD}

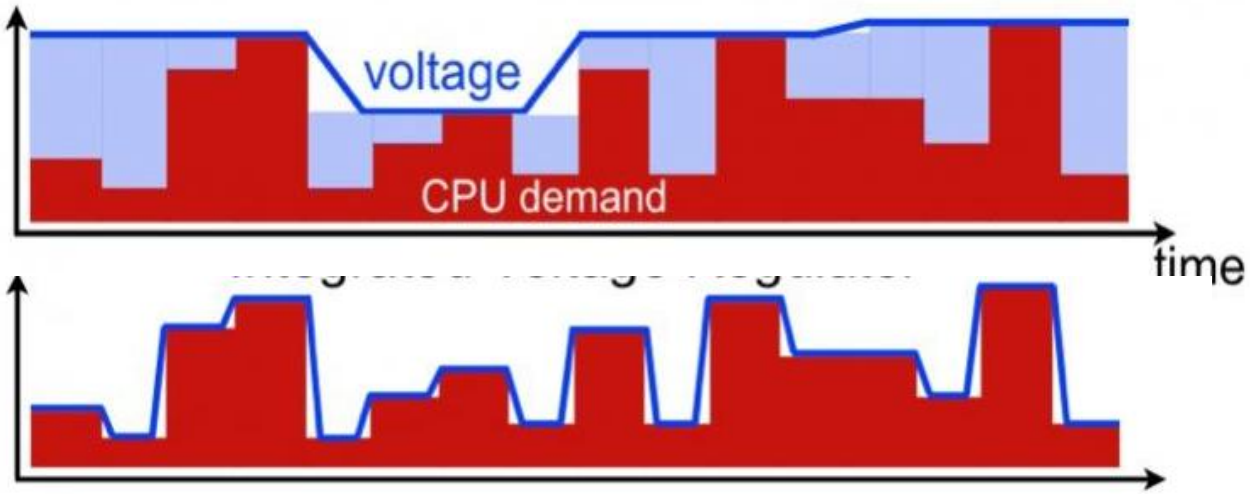
- Decreasing the V_{DD} decreases dynamic energy consumption (quadratically)
- But, increases gate delay (decreases performance)



If lower V_{DD} are used at run time, the clock frequency must also be reduced.

Technique 2 - DVFS

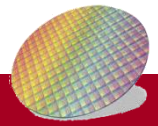
- Dynamic Voltage and Frequency Scaling
  - Lower voltage -> lower power
  - Higher voltage -> higher power
- Better to run at the lowest supply voltage and frequency that meets the timing constraints
  - Maximize energy and power reduction



Save more energy & power

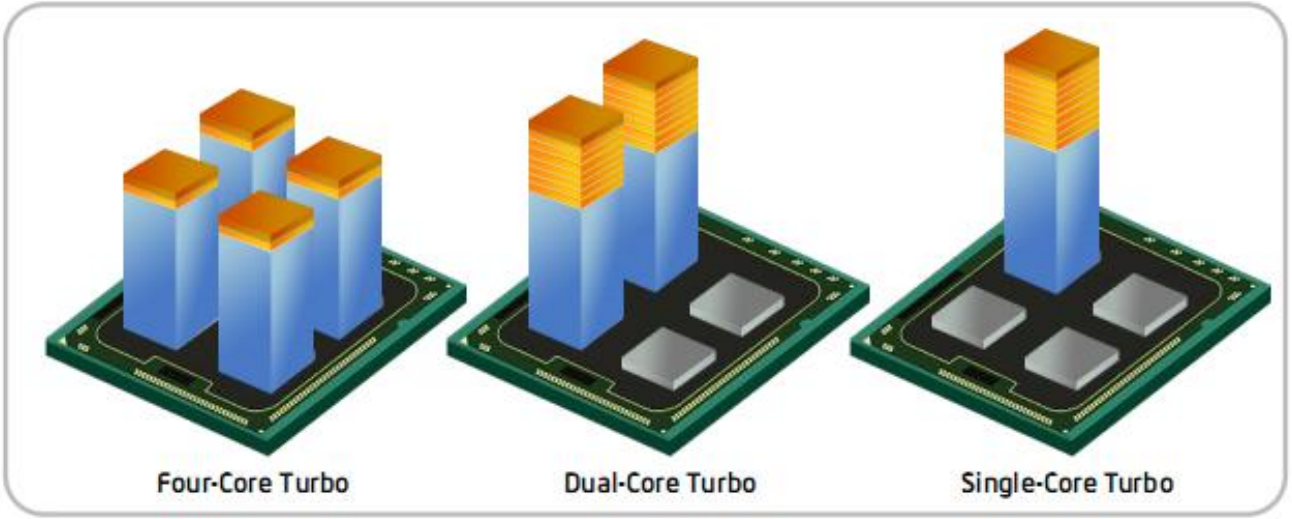
Technique 3 – Make typical case energy efficient

- Make typical case energy efficient
  - –Hence, more energy can be saved.
- e.g. Many components, such as disks, are often idle
  - –Enter lower power states to save energy
- e.g. Turn the power of unused cache
- e.g. Low power state for DRAM, disks
  - Use lower voltage modes to save energy
  - Disk to spins at lower rates to save energy



Technique 4 - Overclocking

- Run at a higher clock rate for a short time on a few cores until temperature starts to rise.
- Finish tasks earlier to save energy
  - Also called race to halt policy
- E.g. Turbo mode

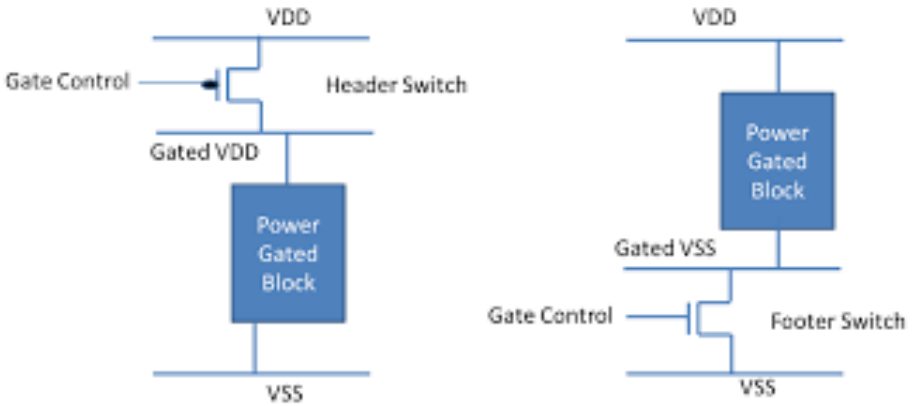


Static Power

- Static power consumption: leakage current flows even when a transistor is off
  - Scales with number of transistors

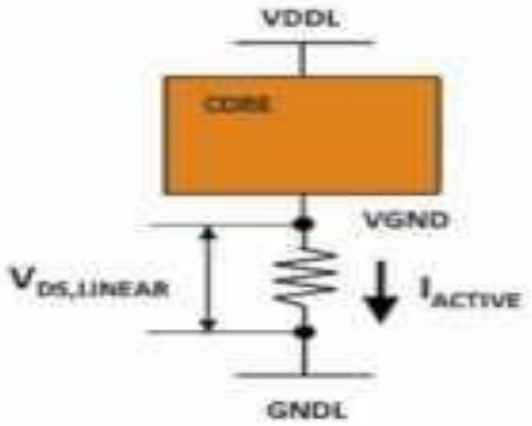
$$P_{static}=Current_{static} \times Voltage$$

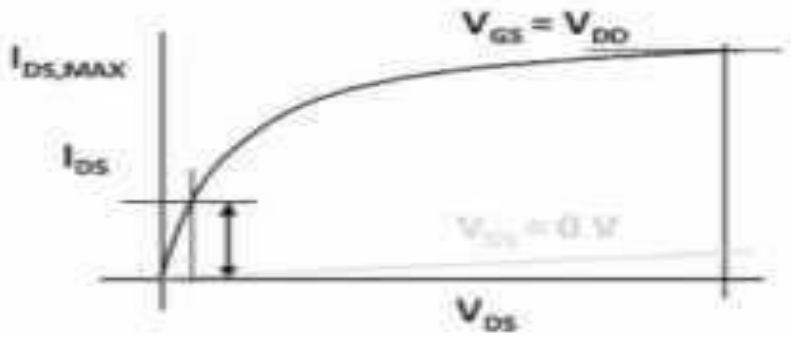
- An effective method to reduce static power: power gating



Power Gating

Normal Operation Mode



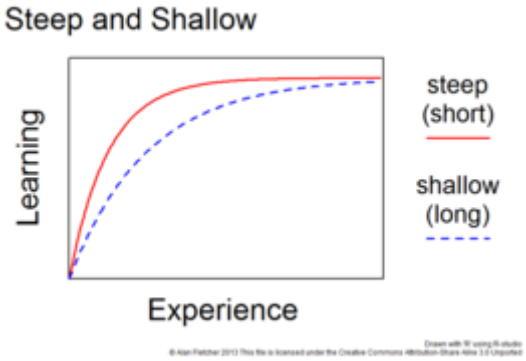


Trends in Cost

The Impact of Time, Volume, and Commoditization

- Cost decreases over time
  - Cost driven down by learning curve
  - Learning curve improves Yield
  - Designs that have twice the yield will have half the cost.
  - DRAM: price closely tracks cost
  - Microprocessors: Prices also drop over time, but, because they are less standardized than DRAMs, the relationship between price and cost is more complex.
- Volume
  - As a rule of thumb, some designers have estimated that cost decreases about 10% for each doubling of volume.
- Commoditization
  - Many vendors ship virtually identical products ➔ the market is highly competitive.

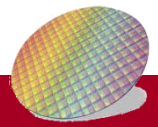
Learning Curve



Total Cost of Integrated Circuits

- Total Cost = Fixed Cost (Non Recurring) + Variable Cost (Recurring)
- Fixed Cost (also called non-recurring expenses, NRE)
  - Fixed cost to produce the design
    - design effort and verification effort, mask generation
    - Influenced by the design complexity and designer productivity
  - More pronounced for small volume products
- Variable Cost(also called recurring expenses)
  - proportional to product volume
  - silicon processing
    - also proportional to chip area
  - assembly (packaging)
  - Test

$$Cost\ per\ IC = Variable\ cost\ per\ IC + \frac{fixed\ cost}{volume}$$



Variable (Recurring) Costs

- Cost: an important factor of the success of the product

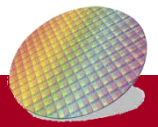
$$\text{Variable Cost} = \frac{\text{Cost of die} + \text{Cost of testing} + \text{Cost of packaging}}{\text{Final Test Yield}}$$

$$\text{Cost of die} = \frac{\text{Cost of wafer}}{\text{Dies per wafer} \times \text{Die yield}}$$

Example 1: A company produced spent 2M dollars as fixed cost and 4M dollars as variable cost to manufacture 1M chips, please find the cost of per IC.

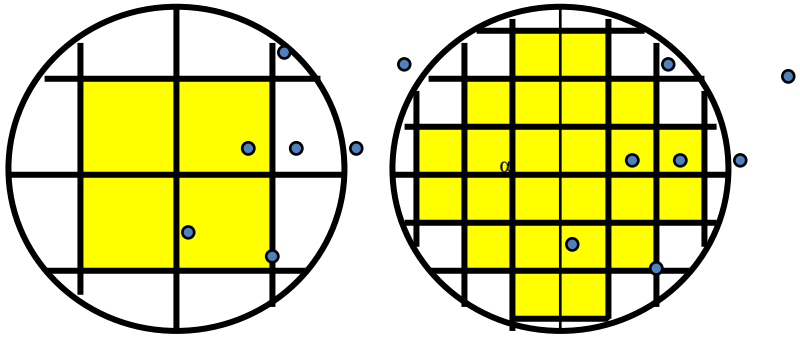
$$Cost\ per\ IC = Variable\ cost\ per\ IC + \left(\frac{fixed\ cost}{volume}\right)$$

$$= \frac{4M}{1M} + \frac{2M}{1M} = 6$$



Die Yield

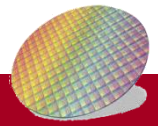
$$\text{die per wafer} = \frac{\pi \times (\text{wafer diameter}/2)^2}{\text{die area}} - \frac{\pi \times \text{wafer diameter}}{\sqrt{2 \times \text{die area}}}$$



Yield estimation: Bose-Einstein Formula

$$Die\ yield = Wafer\ yield \times \frac{1}{(1 + Defect\ per\ unit\ area\ \times Die\ area)^N}$$

- Wafer yield ~ 100%
- Defects per unit area : a measure of random manufacturing defects that occurs
  - 0.016-0.057 defects per square cm (2010)
- N is a process-complexity factor that depends on the manufacturing process
  - N = 11.5-15.5 (40 nm, 2010)

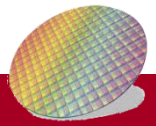


Die per wafer example

- Find the number of dies per 300 mm (30cm) wafer for a die
  - (1) for dies that are 1.5cm on a side and
  - (2) for dies that are 1.0cm on a side

$$\text{Dies per wafer} = \frac{\pi \times (30/2)^2}{2.25} - \frac{\pi \times 30}{\sqrt{2 \times 2.25}} = \frac{706.9}{2.25} - \frac{94.2}{2.12} = 270$$

$$\text{Dies per wafer} = \frac{\pi \times (30/2)^2}{1.00} - \frac{\pi \times 30}{\sqrt{2 \times 1.00}} = \frac{706.9}{1.00} - \frac{94.2}{1.41} = 640$$



Yield example

Q1: Find the die yield, assuming a defect density of 0.031 per cm^2 and N is 13.5

(1) For dies that are 1.5cm on a side

(2) For dies that are 1.0 cm on a side

$$\text{Die Yield} = \frac{1}{(1 + 0.031 \times 2.25)^{13.5}} = 0.40$$

$$\text{Die Yield} = \frac{1}{(1 + 0.031 \times 1.00)^{13.5}} = 0.66$$

Q2: Follow the above question, find the number of good dies per wafer

(1) For dies that are 1.5cm on a side

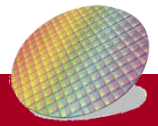
(2) For dies that are 1.0 cm on a side

$$\text{Num of Good Die} = 270 \times 0.4 \approx 108$$

$$\text{Num of Good Die} = 640 \times 0.66 \approx 422$$

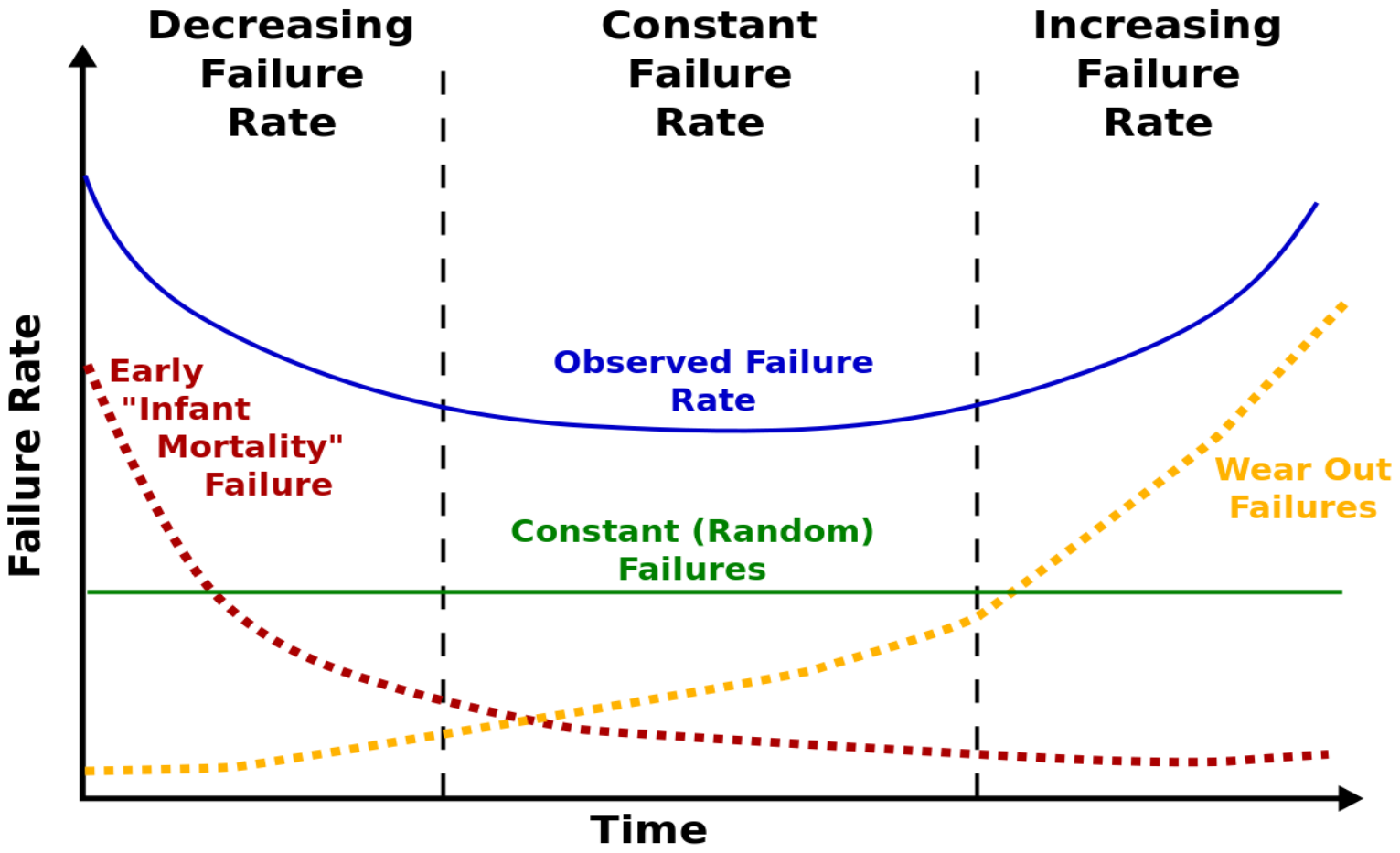
Area increase will cause number of good die decrees significantly => significantly increase cost, in fact

$$\text{cost of die} \propto \text{f(die area)}^{3 \text{ or } 4}$$



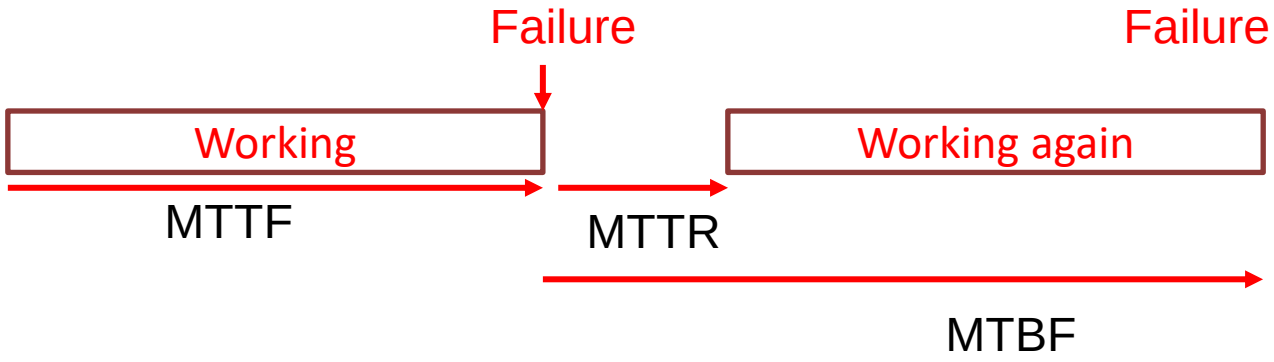
Dependability

- Failure rate bathtub curve
  - Earlier failure rate: decreasing
  - Random failure rate: constant
  - Wear-out failure rate: increasing



Measure of Dependability

- Measure of Dependability
  - Module Reliability and Module availability
- Module Reliability: a measure of continuous service from a reference initial instant.
  - Method to measure reliability
    - Mean Time to failure (MTTF) vs. Mean time to repair (MTTR)
    - Mean time between failures (MTBF) the sum of MTTF+MTTR



- FIT: Failure in Time: failure per billion hours

Relationship between MTTF & FIT

- MTTF: Mean Time to failure
- Failure in Time (FIT)
  - Represents the rate of failures
  - Stand for number of failure per billion hours

$$100\ \text{FIT} = \frac{100}{1{,}000{,}000{,}000} = \frac{1}{10{,}000{,}000}$$

One failure in 10M hours

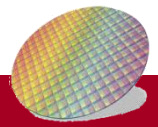
- MTTF can be converted FIT, and vice versa

Example: A device has MTTF of 1,000,000 hours, find its FIT

1,000,000-hour MTTF means there are 1 failure in 1,000,000 hour
=> 1000 failures in 1000,000,000

=> 1000 FIT

$$\frac{1}{1000000} = \frac{1000}{1000000000}$$



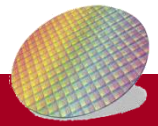
MTTF example

- A disk system with the following components, find its FIT and MTTF
  - 10 disks, each rated at 1000000-hour MTTF
  - 1 ATA controller, 500,000-hour MTTF
  - 1 power supply, 200,000-hour MTTF
  - 1 fan, 200,000-hour MTTF
  - 1 ATA cable, 1000000 hour MTTF

$$Failure\ rate_{system} = 10 \times \frac{1}{1M} + \frac{1}{500K} + \frac{1}{200K} + \frac{1}{200K} + \frac{1}{1000K}$$

$$= \frac{10 + 2 + 5 + 5 + 1}{1000000} = \frac{23}{1000000} = \frac{23000}{1000000000} = 2300\ FIT$$

$$MTTF = \frac{1}{Failure\ rate} = 43500\ hours$$ (just under 5 years)

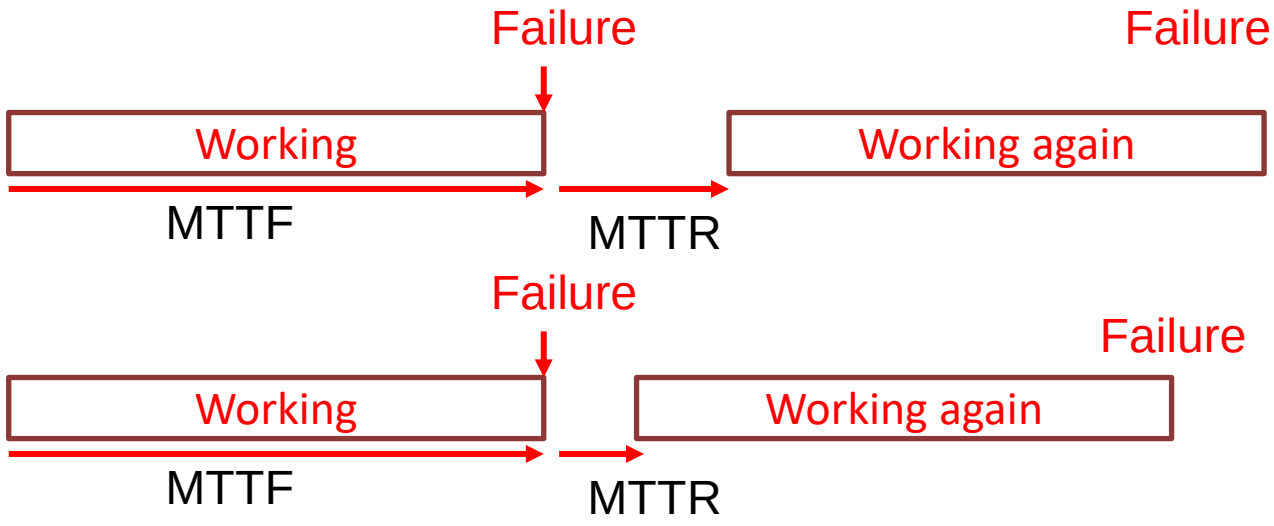


Modular Availability

- Module availability: a measure of the service with respect to the alternation between of the two stats of accomplishment

$$Module\ availability = \frac{MTTF}{(MTTF + MTTR)}$$

Which one has better availability?



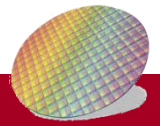
Measuring Performance

- Typical Performance metrics:
  - Response time or Execution time: The time between the start and the completion of an event
  - Throughput: the total amount of work done in a given time

So far, we compare performance using execution time

X is *n* times faster than Y $\Leftrightarrow$ $\frac{\text{Perf}_X}{\text{Perf}_Y} = \frac{\text{Execution time}_Y}{\text{Execution time}_X} = n$

Note: For warehouse-scale computer: throughput (the total amount of work done in a given time) are more important



Type of Benchmarks

- Benchmarks: applications or programs used to measure performance
- Types of Benchmarks
  - Kernels: small, key pieces of real application
    - e.g. matrix multiplication
  - Toy programs: 100-line simple programs
    - e.g. sorting
  - Synthetic benchmarks : fake programs that try to match the behavior of really application
    - Dhrystone
  - Benchmark suites
    - Collections of real applications
    - But hard to test ALL applications, a subset is used
    - E.g. SPEC CPU 2006

Not recommended as results may be falsified.

SPEC2006 benchmark description	Benchmark name by SPEC generation				
	SPEC2006	SPEC2000	SPEC95	SPEC92	SPEC89
GNU C compiler					gcc
Interpreted string processing			perl		espresso
Combinatorial optimization		mcf			li
Block-sorting compression		bzip2		compress	eqntott
Go game (AI)	go	vortex	go	sc	
Video compression	h264avc	gzip	ijpeg		
Games/path finding	astar	eon	m88ksim		
Search gene sequence	hmmer	twolf			
Quantum computer simulation	libquantum	vortex			
Discrete event simulation library	omnetpp	vpr			
Chess game (AI)	sjeng	crafty			
XML parsing	xalancbmk	parser			
CFD/blast waves	bwaves				fpppp
Numerical relativity	cactusADM				tomcatv
Finite element code	calculix				doduc
Differential equation solver framework	dealII				nasa7
Quantum chemistry	gamess				spice
EM solver (freq/time domain)	GemsFDTD			swim	matrix300
Scalable molecular dynamics (~NAMD)	gromacs		apsi	hydro2d	
Lattice Boltzman method (fluid/air flow)	lbm		mgrid	su2cor	
Large eddie simulation/turbulent CFD	LESlie3d	wupwise	applu	wave5	
Lattice quantum chromodynamics	milc	apply	turb3d		
Molecular dynamics	namd	galgel			
Image ray tracing	povray	mesa			
Spare linear algebra	soplex	art			
Speech recognition	sphinx3	equake			
Quantum chemistry/object oriented	tonto	facerec			
Weather research and forecasting	wrf	ammp			
Magneto hydrodynamics (astrophysics)	zeusmp	lucas			
		fma3d			
		sixtrack			

SPEC89: 10

SPEC92:6

SPEC95:8

SPEC2000:21

SPEC2006: 29

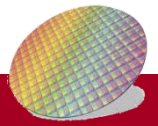
For SPEC 2006

: 12 integer

- 9 in C, 3 in C++

:17 FP

- 6 in Fortrain
- 4 in C++
- 3 in C
- 4 in mixed C



SPECRatio

- SPECRatio: Comparing a computer's performance to a reference computer

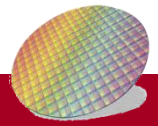
$$\text{SPECRatio}_A = \frac{\text{Performace}_A}{\text{Performace}_{REF}} = \frac{\text{Time}_{Ref}}{\text{Time}_A}$$

Benchmarks	Ultra 5 (sec)	Opteron (Sec)	SPECRatio
Wupwise	1600	51.5	31.06

$$\frac{\text{Performace}_{Opteron}}{\text{Performace}_{ULTRA}} = \frac{\text{Time}_{ULTRA}}{\text{Time}_{Opteron}} = \frac{1600}{51.5} = 31.06$$

- Compare two computers using SPECRatio

$$\frac{\text{Performace}_A}{\text{Performace}_B} = \frac{\frac{\text{Performance}_A}{\text{Performance}_{Ref}}}{\frac{\text{Performance}_B}{\text{Performance}_{Ref}}} = \frac{\text{SPECRatio}_A}{\text{SPECRatio}_B} = \frac{\text{Time}_B}{\text{Time}_A}$$



SPECRatio of a benchmark suite

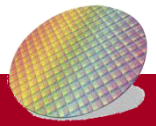
Benchmarks	Ultra 5 (sec)	Opteron (Sec)	SPECRatio
Wupwise	1600	51.5	31.06
Swim	3100	125.0	24.73
Mgrid	1800	98.0	18.37
Applu	2100	94.0	22.34
Mesa	1400	64.6	21.69
Galgel	2900	86.4	33.57
Art	2600	92.4	28.13
Equake	1300	72.6	17.92
Facerec	1900	73.6	25.80
Ammp	2200	136.0	16.14
Lucas	2000	88.8	22.52
Fma3d	2100	120.0	17.48
sixtrack	1100	123.0	8.95
apsi	2600	150.0	17.36
G. M.			20.886

If a benchmark suite contains many benchmarks, the overall SPECratio is the geometric mean of each SPECratio

$$SPECRatio_{overall} = \sqrt[n]{\prod_{i=1}^{n} SPECRatio_i}$$

$$= \sqrt[14]{31.06 \times 24.73 \ldots \times 17.36}$$

$$= 20.86$$



Quantitative Principles of Computer Design (1)

- 4 principles

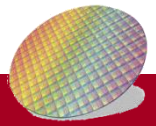
Take Advantage of Parallelism

Principle of Locality

Focus on the Common Case

Processor Performance Equation

- Principle 1:Take Advantage of Parallelism
  - Can be done at different level
  - e.g. multiple processors
  - e.g. pipelining (CPU design)
  - e.g. carry-lookahead adder (ALU design)
- Principle 2: Principle of Locality
  - 90% of execution spent in 10% of codes
  - Reuse of data and instructions
  - Temporal locality and Spatial locality



Quantitative Principles of Computer Design (2)

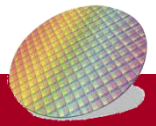
- Principle 3: Focus on the Common Case: improve the common case will achieve better improvement
  - Beware of Amdahl's Law: Possible improvement is limited by the amount that the improved feature is used

$$\text{Speedup}_{\text{overall}} = \frac{\text{Execution time}_{\text{old}}}{\text{Execution time}_{\text{new}}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

e.g.: a CPU spends 40% of time on computing and 60% of time on I/O. Assume a new CPU has 10X faster on computing, find $\text{Speedup}_{\text{overall}}$

$$Speedup_{overall} = \frac{1}{(1 - 0.4) + \frac{0.4}{10}} = 1.56$$

Speedup is 1.56



Quantitative Principles of Computer Design (3)

- Example: multiplication accounts for 80s of 100s. How much improvement in multiplication performance to get 4x overall?

Multiplication account for 80/100=80%
of execution time

$$4 = \frac{1}{(1 - 0.8) + \frac{0.8}{X}}$$

X = 16

multiplication needs to be 16x faster

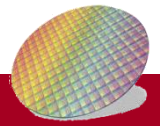
Quantitative Principles of Computer Design (4) Processor Performance Equation

- CPU Time for a program =

$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$
$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

$$\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}}$$

$$\text{CPU Time} = \text{Clock Cycles} \times \text{Clock Cycle Time}$$
$$= \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$$
$$= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$



CPI in More Detail

- If different instruction classes take different numbers of cycles

$$\text{Clock Cycles} = \sum_{i=1}^{n} (IC_i \times CPI_i)$$

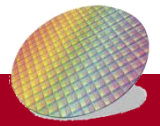
$$\text{CPU Time} = \sum_{i=1}^{n} (IC_i \times CPI_i) \times \text{CycleTime}$$

- Weighted average CPI (Effective CPI)

$$CPI = \frac{\text{Clock Cycles}}{IC}$$

$$\text{Effective CPI} = \sum_{i=1}^{n} (CPI_i \times Prob_i)$$

$Prob_i$:Probability of Instruction i



CPI Example

- Alternative compiled code sequences using instructions in classes A, B, C, find effective CPI (Avg. CPI) for instruction sequence 1 and 2

Instruction Class	A	B	C
CPI for class	1	2	3
Inst. Count in sequence 1	2	1	2
Inst. Count in sequence 2	4	1	1

- Sequence 1: IC = 5
- Prob(A) = 2/5 =0.4
- Prob(B) = 1/5 = 0.2
- Prob(C) = 2/5 = 0.4

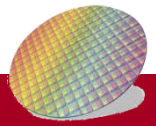
Avg. CPI = 1*0.4+2*0.2+ 3*0.4

= 2.0

- Sequence 2: IC = 6
- Prob (A) = 4/6
- Prob (B)= 1/6
- Prob (C) = 1/6

- Avg. CPI = 1*4/6 + 2* 1/6+ 3*1/6

= 1.5

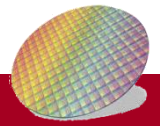


CPI Example

- Alternative compiled code sequences using instructions in classes A, B, C, find average CPI for instruction sequence 1 and 2

Instruction Class	A	B	C
CPI for class	1	2	3
Inst. Count in sequence 1	2	1	2
Inst. Count in sequence 2	4	1	1

- Sequence 1: IC = 5
  - Clock Cycles
    = 2×1 + 1×2 + 2×3
    = 10
  - Avg. CPI = 10/5 = 2.0
- Sequence 2: IC = 6
  - Clock Cycles
    = 4×1 + 1×2 + 1×3
    = 9
  - Avg. CPI = 9/6 = 1.5



Backup Slides