Advanced Computer Architecture

Sep 9th 2022

Qiang Cao

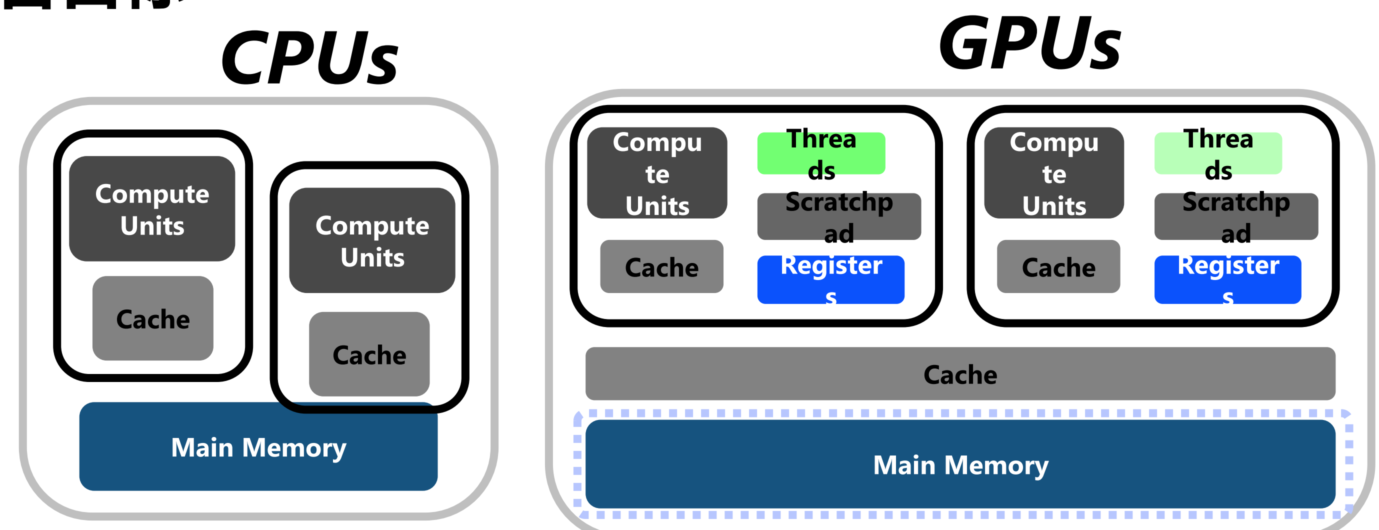
武汉光电国家研究中心存储部
华中科技大学计算机学院

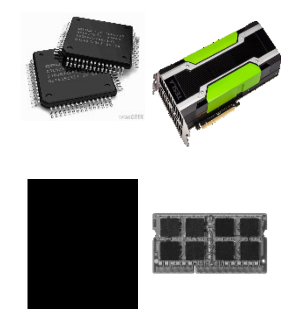
计算机系统结构定义

满足设计目标需求的计算机软硬部件的系统组织方式

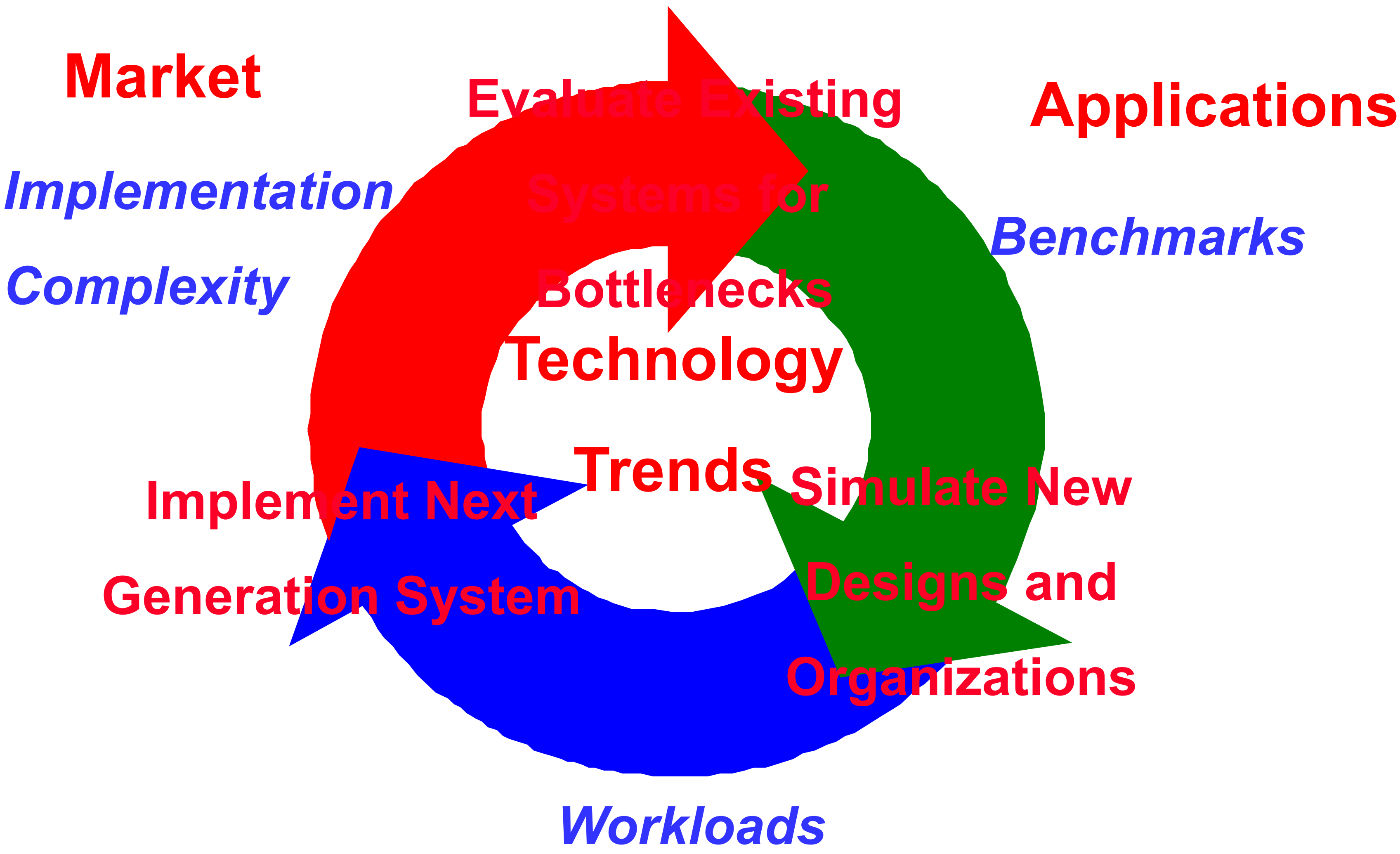
强调在多个部件之上组织系统的方式，以满足系统功能和性能要求

设计要求可以是特定功能、峰值性能或者平均性能、最长电池寿命、最低成本等功能及性能指标以及它们的组合目标





Computer Engineering Methodology

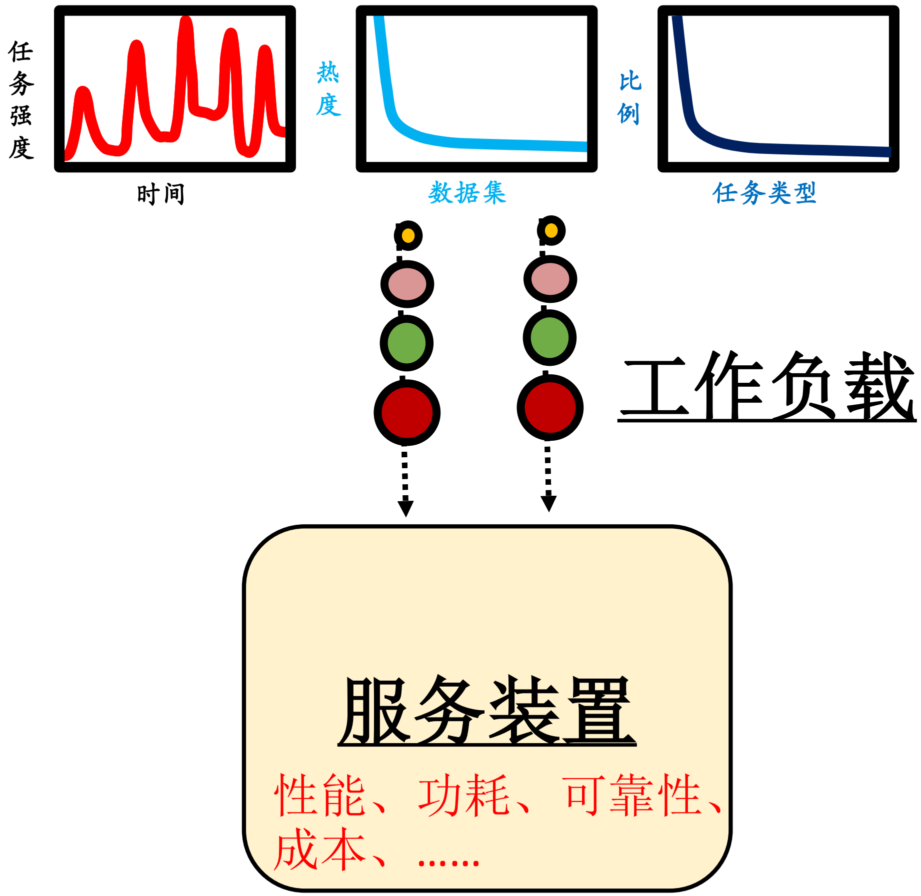


Six Design-Principles

1. **分层设计原则(Level, Interface，Stack)**
2. **并行性设计与挖掘原则(Parallelism)**
3. **局域性挖掘原则(Locality)**
4. **加快经常事件原则(Common-case Accelerating)**
5. **平衡设计原则(Balance, Tradeoff)**
6. **性能评价驱动设计原则(Performance Evaluation)**

Workloads

- **Workload-oriented design**



Metrics used to Compare Designs

- **Execution Time**
  - average and worst-case
  - Latency vs. Throughput
- **Energy and Power**
  - Also peak power and peak switching current
- **Reliability**
  - Resiliency to electrical noise, part failure
  - Robustness to bad software, operator error
- **Cost**
  - Die cost and system cost
- **Maintainability**
  - System administration costs
- **Compatibility**
  - Software costs dominate

What is Performance?

- Latency (or response time or execution time)
  - time to complete one task
- Bandwidth (or throughput)
  - tasks completed per unit time

Definition: Performance

- Performance is in units of things per sec
  - bigger is better
- If we are primarily concerned with response time

$$\text{performance}(x) = \frac{1}{\text{execution\_time}(x)}$$

" X is n times faster than Y" means

$$n = \frac{\text{Performance}(X)}{\text{Performance}(Y)} = \frac{\text{Execution\_time}(Y)}{\text{Execution\_time}(X)}$$

Performance: What to measure

- Usually rely on benchmarks vs. real workloads
- To increase predictability, collections of benchmark applications-- *benchmark suites* -- are popular
- SPECCPU: popular desktop benchmark suite
  – CPU only, split between integer and floating point programs
  – SPECint2000 has 12 integer, SPECfp2000 has 14 integer pgms
  – SPECCPU2006 to be announced Spring 2006
  – SPECSFS (NFS file server) and SPECWeb (WebServer) added as server benchmarks
- Transaction Processing Council measures server performance and cost-performance for databases
  – TPC-C Complex query for Online Transaction Processing
  – TPC-H models ad hoc decision support
  – TPC-W a transactional web benchmark
  – TPC-App application server and web services benchmark

Summarizing Performance

System	Rate (Task 1)	Rate (Task 2)
A	10	20
B	20	10

Which system is faster?

… depends who's selling

System	Rate (Task 1)	Rate (Task 2)	Average
A	10	20	15
B	20	10	15

Average throughput

System	Rate (Task 1)	Rate (Task 2)	Average
A	0.50	2.00	1.25
B	1.00	1.00	1.00

Throughput relative to B

System	Rate (Task 1)	Rate (Task 2)	Average
A	1.00	1.00	1.00
B	2.00	0.50	1.25

Throughput relative to A

Summarizing Performance over Set of Benchmark Programs

Arithmetic mean of execution times t_i (in seconds)

$$1/n \sum_i t_i$$

Harmonic mean of execution rates r_i (MIPS/MFLOPS)

$$n/ [\sum_i (1/r_i)]$$

- **Both equivalent to workload where each program is run the same number of times**
- **Can add weighting factors to model other workload distributions**

Normalized Execution Time and Geometric Mean

- **Measure speedup up relative to reference machine**

$$\text{ratio} = t_{Ref}/t_A$$

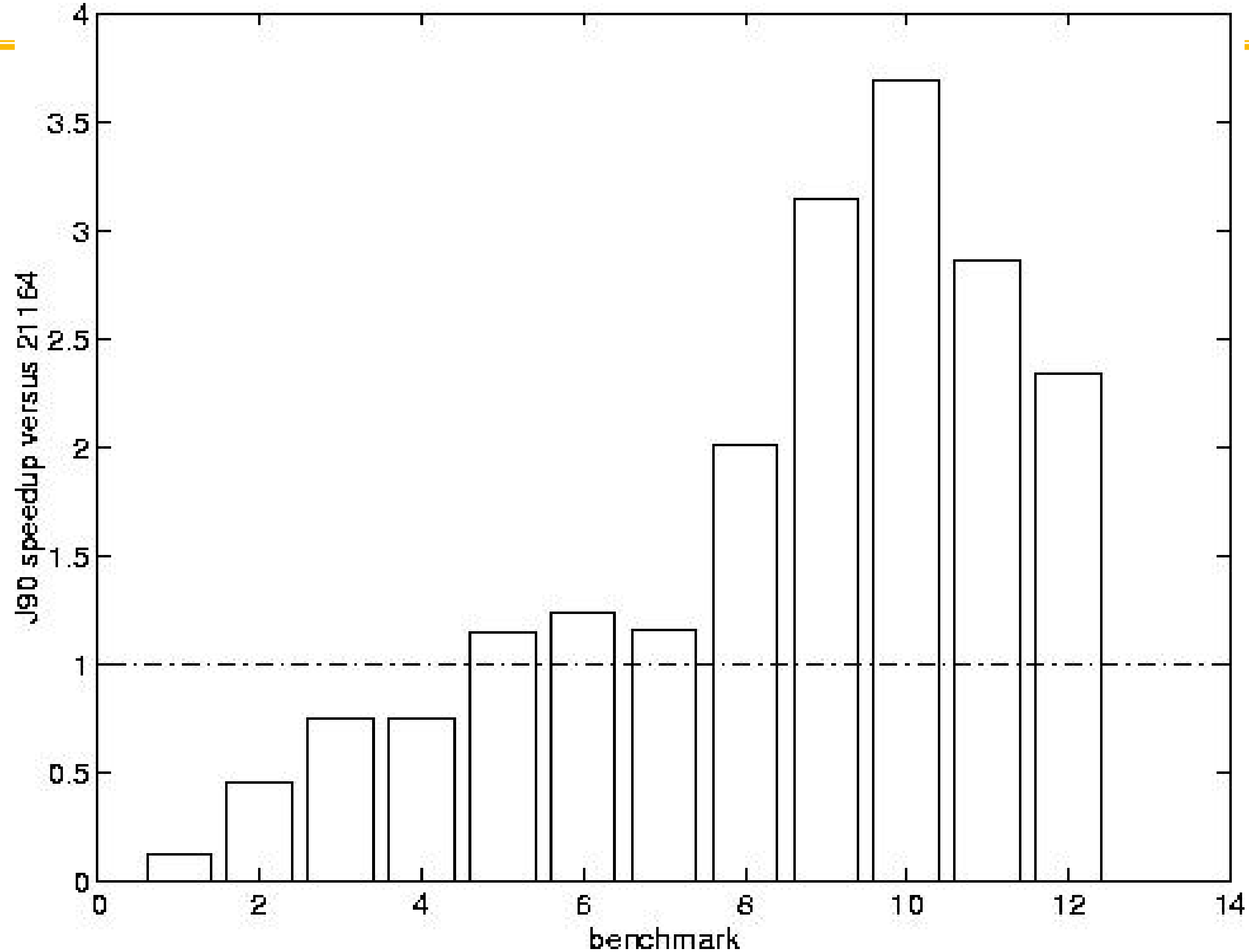
- **Average time ratios using geometric mean**

$$\sqrt[n]{\left(\prod_I \text{ratio}_i\right)}$$

- **Insensitive to machine chosen as reference**
- **Insensitive to run time of individual benchmarks**
- **Used by SPEC89, SPEC92, SPEC95, …, SPEC2006**

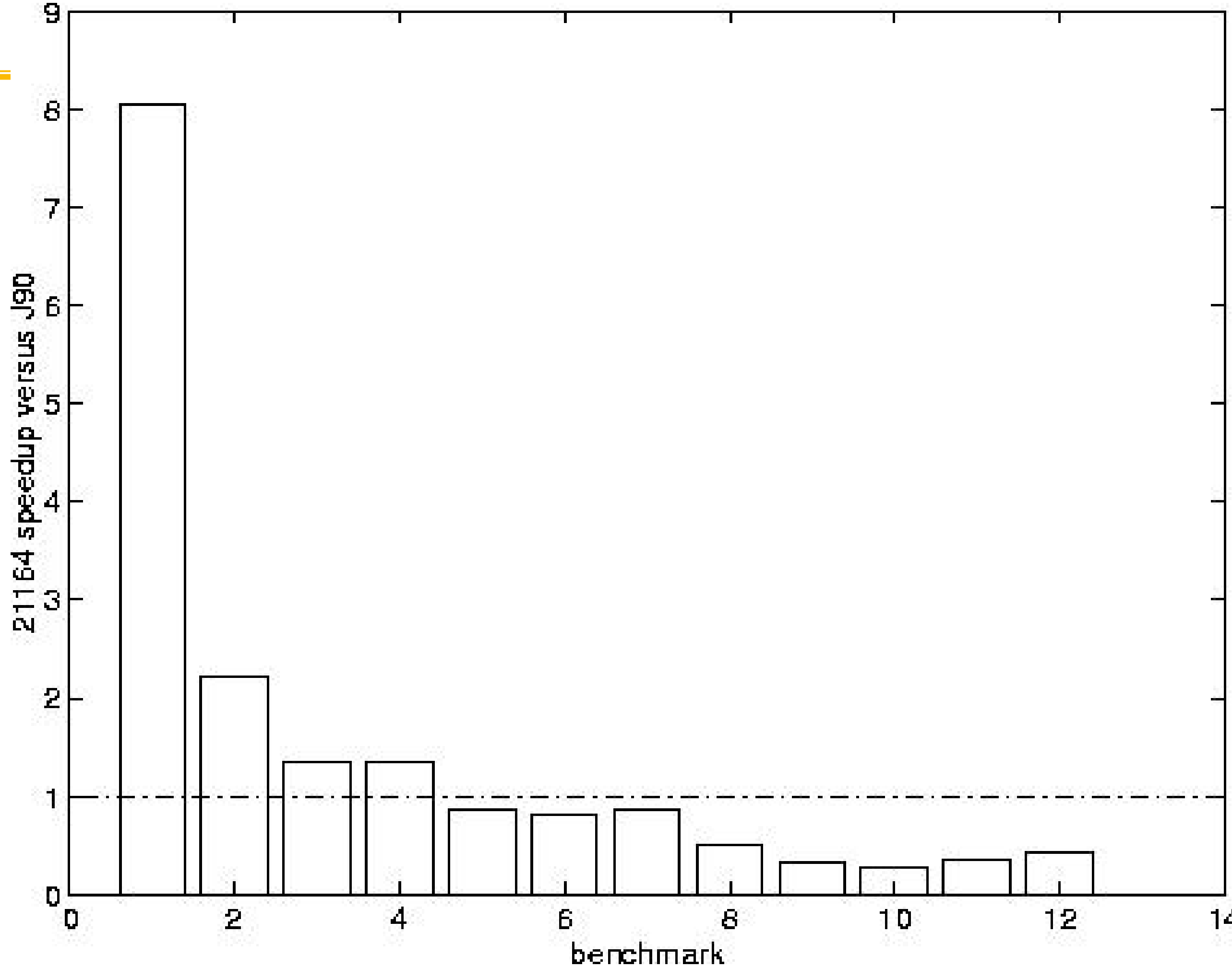
….. But beware that choice of reference machine can suggest what is “normal” performance profile:

Vector/Superscalar Speedup



- 100 MHz Cray J90 vector machine versus 300MHz Alpha 21164
- [LANL Computational Physics Codes, Wasserman, ICS'96]
- Vector machine peaks on a few codes????

Superscalar/Vector Speedup



- 100 MHz Cray J90 vector machine versus 300MHz Alpha 21164
- [LANL Computational Physics Codes, Wasserman, ICS'96]
- Scalar machine peaks on one code???

How to Mislead with Performance Reports

- Select pieces of workload that work well on your design, ignore others
- Use unrealistic data set sizes for application (too big or too small)
- Report throughput numbers for a latency benchmark
- Report latency numbers for a throughput benchmark
- Report performance on a kernel and claim it represents an entire application
- Use 16-bit fixed-point arithmetic (because it's fastest on your system) even though application requires 64-bit floating-point arithmetic
- Use a less efficient algorithm on the competing machine
- Report speedup for an inefficient algorithm (bubblesort)
- Compare hand-optimized assembly code with unoptimized C code
- Compare your design using next year's technology against competitor's year old design (1% performance improvement per week)
- Ignore the relative cost of the systems being compared
- Report averages and not individual results
- Report speedup over unspecified base system, not absolute times
- Report efficiency not absolute times
- Report MFLOPS not absolute times (use inefficient algorithm)

[David Bailey "Twelve ways to fool the masses when giving performance results for parallel supercomputers"]

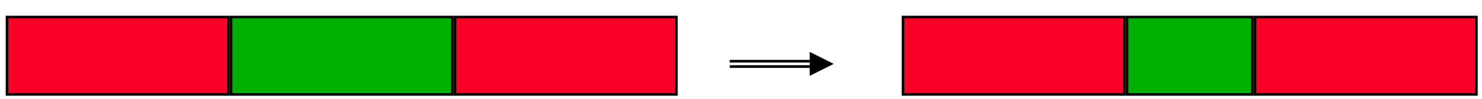
Amdahl's Law

$$ExTime_{new} = ExTime_{old} \times \left[\left(1 - Fraction_{enhanced}\right) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}} \right]$$

$$Speedup_{overall} = \frac{ExTime_{old}}{ExTime_{new}} = \frac{1}{\left(1 - Fraction_{enhanced}\right) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

Best you could ever hope to do:

$$Speedup_{maximum} = \frac{1}{\left(1 - Fraction_{enhanced}\right)}$$



Amdahl's Law example

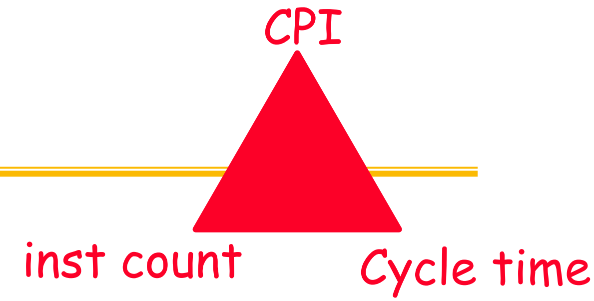
- **New CPU 10X faster**
- **I/O bound server, so 60% time waiting for I/O**

$$\text{Speedup}_{\text{overall}} = \frac{1}{\left(1 - \text{Fraction}_{\text{enhanced}}\right) + \dfrac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

$$= \frac{1}{\left(1 - 0.4\right) + \dfrac{0.4}{10}} = \frac{1}{0.64} = 1.56$$

- **Apparently, its human nature to be attracted by 10X faster, vs. keeping in perspective its just 1.6X faster**

Computer Performance



$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

	Inst Count	CPI	Clock Rate
Program	X		
Compiler	X	(X)	
Inst. Set.	X	X	
Organization		X	X
Technology			X

Cycles Per Instruction (Throughput)

"Average Cycles per Instruction"

CPI = (CPU Time * Clock Rate) / Instruction Count
= Cycles / Instruction Count

$$\text{CPU time} = \text{Cycle Time} \times \sum_{j=1}^{n} \text{CPI}_j \times \text{I}_j$$

$$\text{CPI} = \sum_{j=1}^{n} \text{CPI}_j \times \text{F}_j \qquad \text{where } \text{F}_j = \frac{\text{I}_j}{\text{Instruction Count}}$$

"Instruction Frequency"

Example: Calculating CPI bottom up

Run benchmark and collect workload characterization (simulate, machine counters, or sampling)

Base Machine (Reg / Reg)

Op	Freq	Cycles	CPI(i)	(% Time)
ALU	50%	1	.5	(33%)
Load	20%	2	.4	(27%)
Store	10%	2	.2	(13%)
Branch	20%	2	.4	(27%)
			1.5	

Typical Mix of instruction types in program

Design guideline: Make the common case fast

MIPS 1% rule: only consider adding an instruction of it is shown to add 1% performance improvement on reasonable benchmarks.

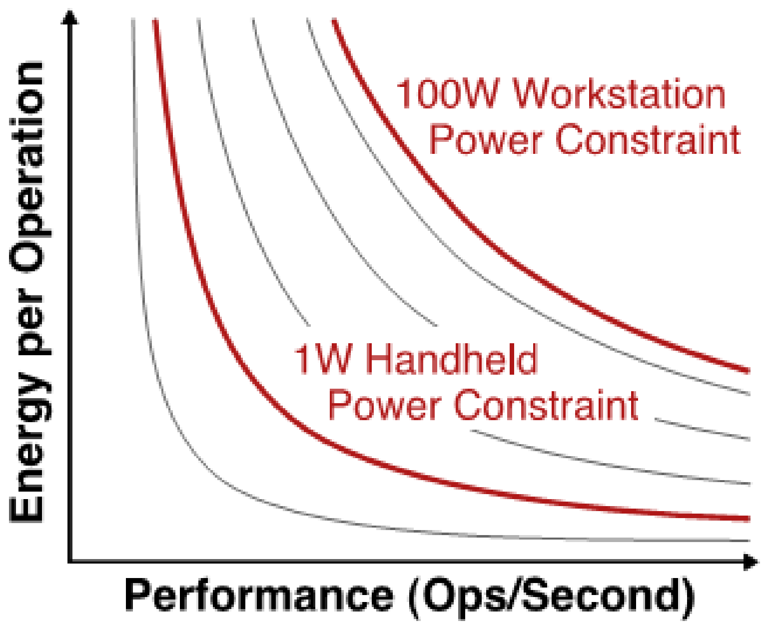
Power and Energy

- Energy to complete operation (Joules)
  - Corresponds approximately to battery life
  - (Battery energy capacity actually depends on rate of discharge)
- Peak power dissipation (Watts = Joules/second)
  - Affects packaging (power and ground pins, thermal design)
- di/dt, peak change in supply current (Amps/second)
  - Affects power supply noise (power and ground pins, decoupling capacitors)

Power Problem

$$\text{Power} = \frac{\text{Energy}}{\text{Second}} = \frac{\text{Energy}}{\text{Op}} \times \frac{\text{Ops}}{\text{Second}}$$

Power	Energy
Chip Packaging	Battery Life
Chip Cooling	Electricity Bill
System Noise	Mobile Device Weight
Case Temperature	
Data-Center Air Conditioning	



CMOS Power Equations

Power due to short-circuit current during transition

Dynamic power consumption

Power due to leakage current

$$P = ACV^2f + \tau AVI_{short}f + VI_{leak}$$

Reduce the supply voltage, V

$$I_{leak} \propto \exp\left(-\frac{qV_t}{kT}\right)$$

$$f_{max} \propto \frac{(V - V_t)^2}{V}$$

Reduce threshold V_t

CMOS Scaling

- **Historic CMOS scaling**
  - Doubling every two years (Moore's law)
    - » Feature size
    - » Device density
  - Device switching speed improves 30-40%/generation
  - Supply & threshold voltages decrease ($V_{dd,}$ V_{th})
- **Projected CMOS scaling**
  - Feature size, device density scaling continues
    - » ~10 year roadmap out to sub-10nm generation
  - Switching speed improves ~20%/generation
  - Voltage scaling tapers off quickly
    - » SRAM cell stability becomes an issue at ~0.7V V_{dd}

Dynamic Power

$$P_{dyn} \approx \sum_{i \in units} C_i V^2 A_i f$$

- Static CMOS: current flows when active
  - Combinational logic evaluates new inputs
  - Flip-flop, latch captures new value (clock edge)
- Terms
  - C: capacitance of circuit
    - wire length, number and size of transistors
  - V: supply voltage
  - A: activity factor
  - f: frequency
- Future: Fundamentally power-constrained

Reducing Dynamic Power

- Reduce capacitance
  - Simpler, smaller design (yeah right)
  - Reduced IPC
- Reduce activity
  - Smarter design
  - Reduced IPC
- Reduce frequency
  - Often in conjunction with reduced voltage
- Reduce voltage
  - Biggest hammer due to quadratic effect, widely employed
  - Can be static (binning/sorting of parts), and/or
  - Dynamic (power modes)
    - » E.g. Transmeta Long Run, AMD PowerNow, Intel Speedstep

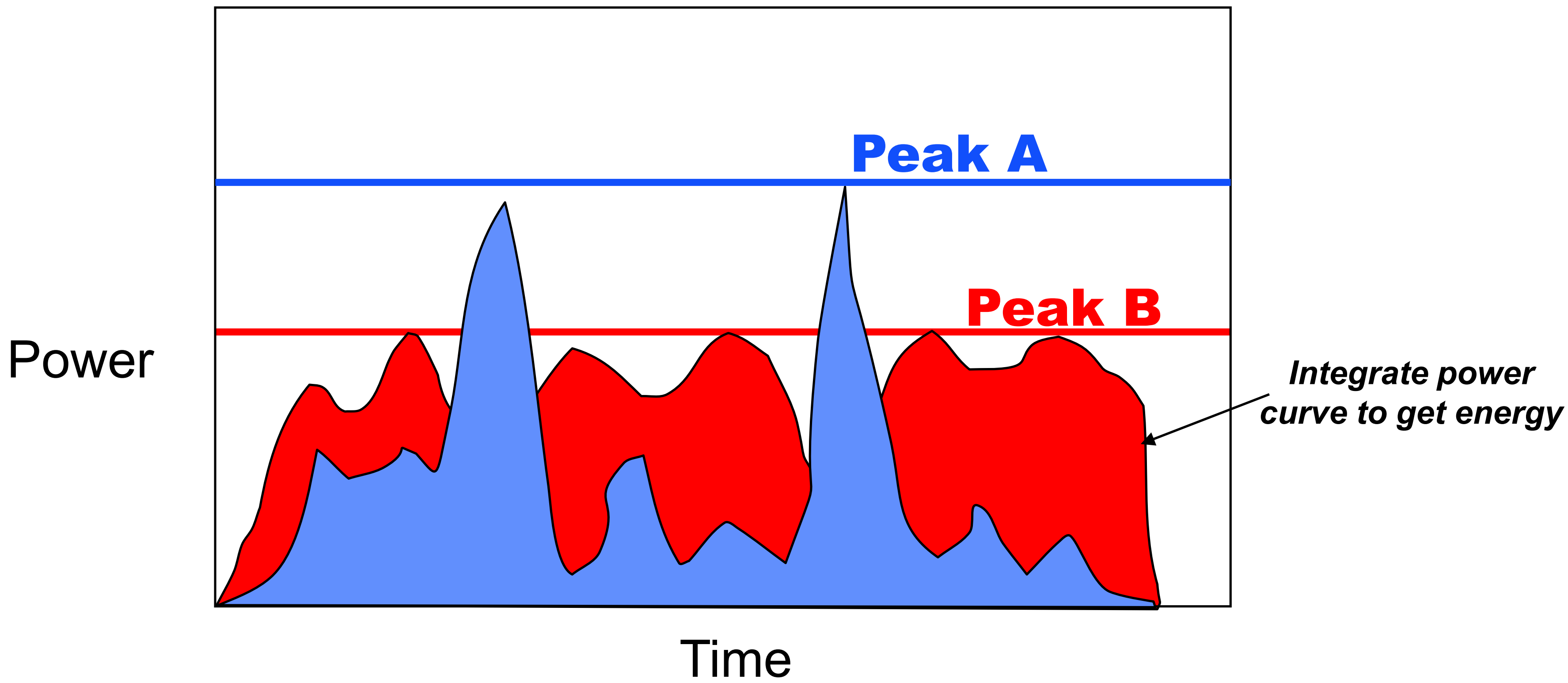
Frequency/Voltage relationship

- **Lower voltage implies lower frequency**
  - **Lower V_{th} increases delay to sense/latch 0/1**
- **Conversely, higher voltage enables higher frequency**
  - **Overclocking**
- **Sorting/binning and setting various V_{dd} & V_{th}**
  - **Characterize device, circuit, chip under varying stress conditions**
  - **Black art – very empirical & closely guarded trade secret**
  - **Implications on reliability**
    - » **Safety margins, product lifetime**
    - » **This is why *overclocking* is possible**

Frequency/Voltage Scaling

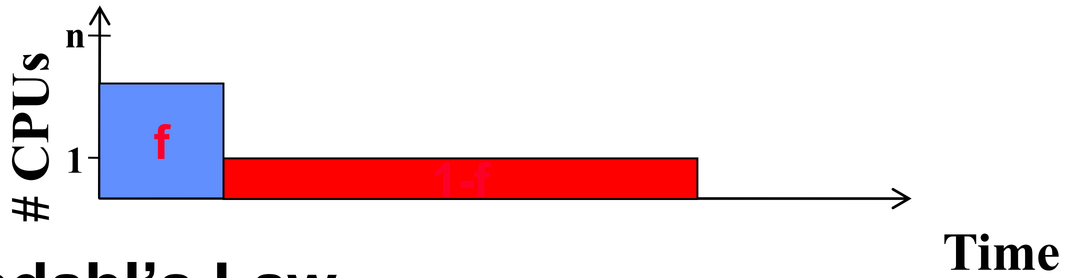
- Voltage/frequency scaling rule of thumb:
  - +/- 1% performance buys -/+ 3% power (3:1 rule)
- Hence, any power-saving technique that saves less than 3x power over performance loss is uninteresting
- Example 1:
  - New technique saves 12% power
  - However, performance degrades 5%
  - Useless, since 12 < 3 x 5
  - Instead, reduce f by 5% (also V), and get 15% power savings
- Example 2:
  - New technique saves 5% power
  - Performance degrades 1%
  - Useful, since 5 > 3 x 1
- Does this rule always hold?

Peak Power versus Lower Energy



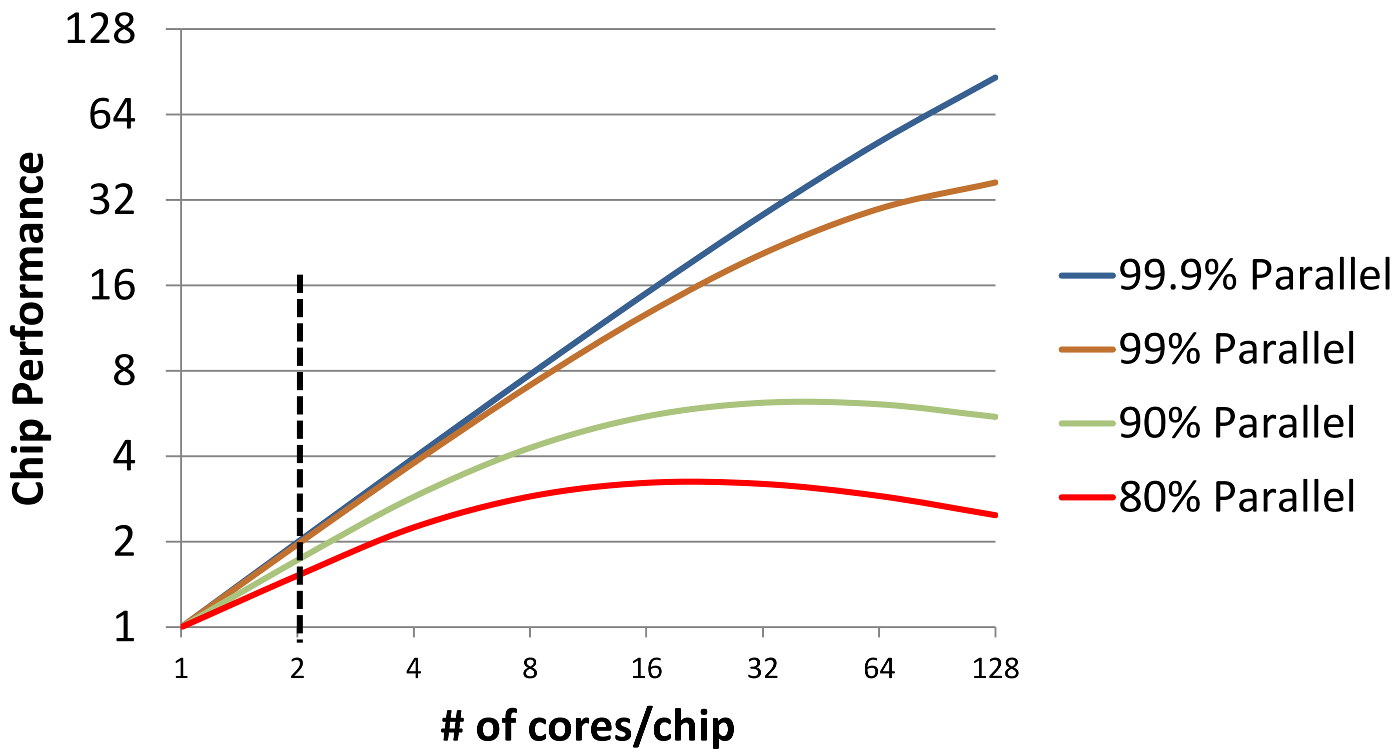
- System **A** has higher peak power, but lower total energy
- System **B** has lower peak power, but higher total energy

Fixed Chip Power Budget



- **Amdahl's Law**
  - Ignores (power) cost of n cores
- **Revised Amdahl's Law**
  - More cores → each core is slower
  - Parallel speedup < n
  - Serial portion (1-f) takes longer
  - Also, interconnect and scaling overhead

Fixed Power Scaling



- **Fixed power budget forces slow cores**
- **Serial code quickly dominates**

Schmoo图

A "Schmoo" plot for a Cell SPU ...

The energy equations: $E_{0\text{->}1} = \frac{1}{2} C V_{dd}^2$ $E_{1\text{->}0} = \frac{1}{2} C V_{dd}^2$

1 Joule of energy is dissipated by a 1 Amp current flowing through a 1 Ohm resistor for 1 second.

Also, 1 Joule of energy is 1 Watt (1 amp into 1 ohm) dissipating for 1 second.

Operating point Q

Operating point P

Vdd (Volts) \ Freq (GHz)	2	2.2	2.4	2.6	2.8	3	3.2	3.4	3.6	3.8	4	4.2	4.4	4.6	4.8	5	5.2
1.3	48C 4W	49C 4W	50C 5W	50C 6W	51C 6W	52C 7W	53C 7W	54C 7W	55C 8W	56C 8W	57C 9W	58C 9W	59C 10W	60C 10W	61C 10W	63C 11W	61C
1.2	39C 2W	39C 3W	40C 3W	41C 4W	42C 4W	42C 4W	43C 5W	44C 5W	45C 5W	45C 5W	46C 6W	47C 6W	47C 7W	48C	49C		
1.1	32C 2W	33C 2W	33C 3W	35C 3W	35C 3W	36C 3W	36C 4W	37C 4W	37C 4W	38C 4W	38C 4W	39C	39C				
1	28C 2W	28C 2W	29C 2W	29C 2W	30C 2W	30C 3W	30C 3W	31C 3W	31C 3W	31C 3W	32C						
0.9	25C 1W	26C 1W	26C 1W	26C 2W	27C 2W	27C 2W	27C								Failed		

Operating point R

Each square shows chip temperature (C) and power (W)

Concepts from Probability Theory

Probability density function: pdf

$f(t) = \text{prob}[t \le x \le t + dt] \,/\, dt = dF(t) \,/\, dt$

Cumulative distribution function: CDF

$F(t) = \text{prob}[x \le t] = \int_0 f(x)\, dx$

Expected value of x

$E_x = \int_{-\infty}^{+\infty} x\, f(x)\, dx = \sum_k x_k f(x_k)$

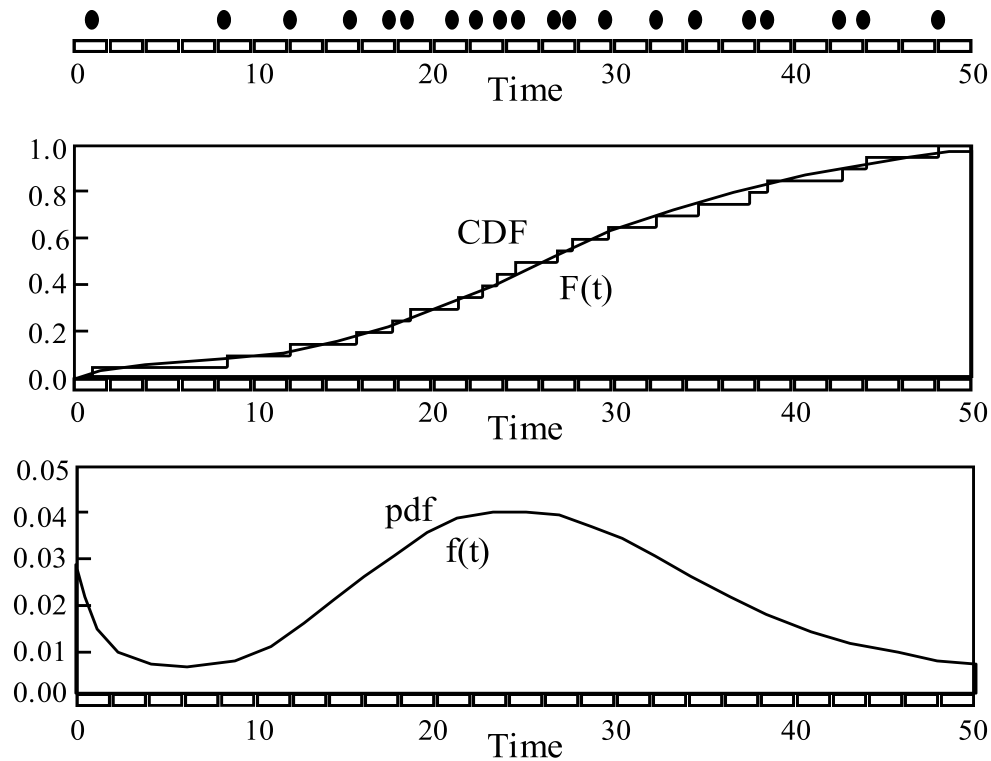
Variance of x

$\sigma_x^2 = \int_{-\infty}^{+\infty} (x - E_x)^2 f(x)\, dx$
$\quad = \sum_k (x_k - E_x)^2 f(x_k)$

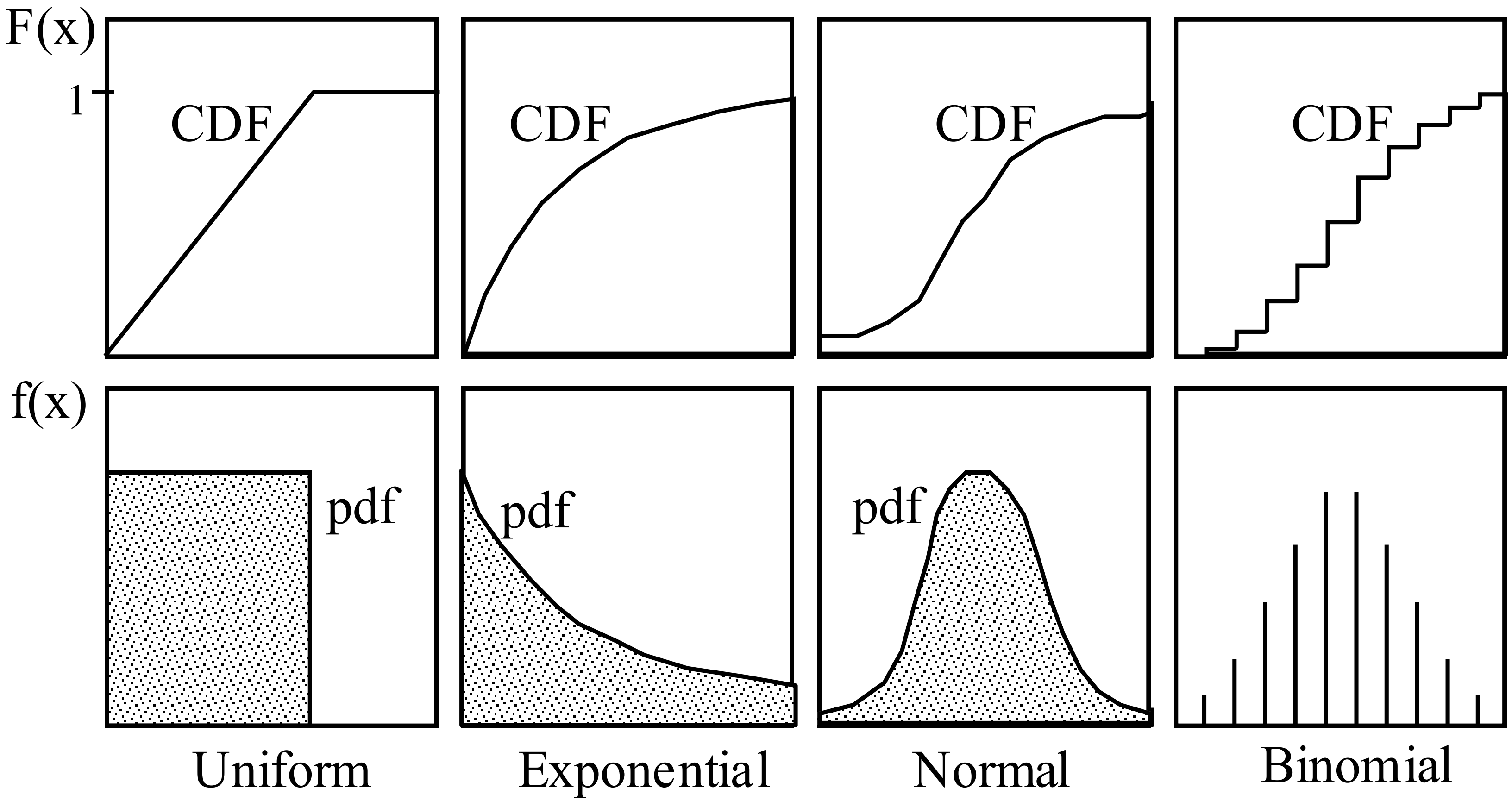
Covariance of x and y

$\psi_{x,y} = E[(x - E_x)(y - E_y)]$
$\quad = E[x\,y] - E_x E_y$

Lifetimes of 20 identical systems

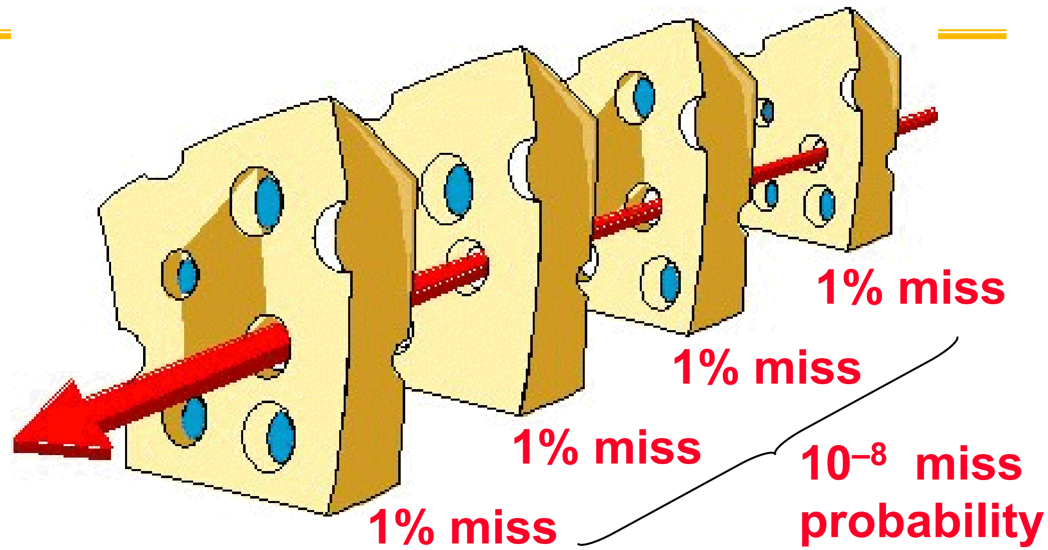


Some Simple Probability Distributions



Layers of Safeguards

With multiple layers of safeguards, a system failure occurs only if warning symptoms and compensating actions are missed at every layer, which is quite unlikely



Is it really?

The computer engineering literature is full of examples of mishaps when two or more layers of protection failed at the same time

Multiple layers increase the reliability significantly only if the “holes” in the representation above are fairly randomly and independently distributed, so that the probability of their being aligned is negligible

Dec. 1986: ARPANET had 7 dedicated lines between NY and Boston;
A backhoe accidentally cut all 7 (they went through the same conduit)

Reliability and MTTF

Reliability: $R(t)$

Probability that system remains in the "Good" state through the interval $[0, t]$

$R(t + dt) = R(t)\,[1 - z(t)\,dt]$

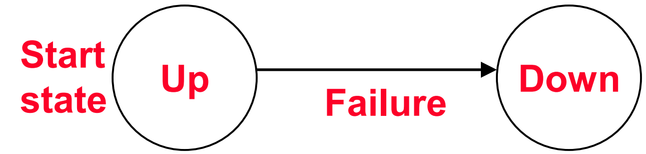
Hazard function

$R(t) = 1 - F(t)$ CDF of the system lifetime, or its unreliability

Constant hazard function $z(t) = \lambda \;\Rightarrow\; R(t) = e^{-\lambda t}$ (system failure rate is independent of its age)

Exponential reliability law

Two-state nonrepairable system



Mean time to failure: MTTF

$$\text{MTTF} = \int_0^{+\infty} t\,f(t)\,dt = \int_0^{+\infty} R(t)\,dt$$

Expected value of lifetime

Area under the reliability curve (easily provable)

Failure Distributions of Interest

Discrete versions

Exponential: $z(t) = \lambda$
$R(t) = e^{-\lambda t}$ MTTF = $1/\lambda$

Geometric
$R(k) = q^k$

Rayleigh: $z(t) = 2\lambda(\lambda t)$
$R(t) = e^{(-\lambda t)^2}$ MTTF = $(1/\lambda)\sqrt{\pi}/2$

Weibull: $z(t) = \alpha\lambda(\lambda t)^{\alpha-1}$
$R(t) = e^{(-\lambda t)^\alpha}$ MTTF = $(1/\lambda)\,\Gamma(1 + 1/\alpha)$

Discrete Weibull

Erlang:
Gen. exponential MTTF = k/λ

Gamma:
Gen. Erlang
(becomes Erlang for b an integer)

Normal:
Reliability and MTTF formulas are complicated

Binomial

Availability, MTTR, and MTBF

(Interval) Availability: $A(t)$
Fraction of time that system is in the "Up" state during the interval $[0, t]$

Steady-state availability: $A = \lim_{t\to\infty} A(t)$

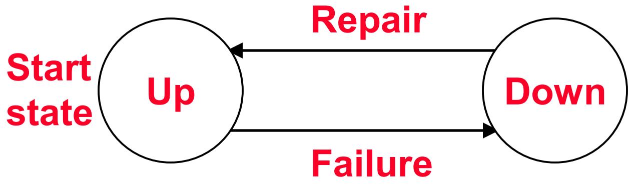
Pointwise availability: $a(t)$
Probability that system available at time t

$A(t) = (1/t) \int_0^t a(x)\, dx$

Availability = Reliability, when there is no repair

Fig. 2.5

Two-state repairable system



Availability is a function not only of how rarely a system fails (reliability) but also of how quickly it can be repaired (time to repair)

$$A = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}} = \frac{\text{MTTF}}{\text{MTBF}} = \frac{\mu}{\lambda + \mu}$$

Repair rate
$1/\mu$ = MTTR
(Will justify this equation later)

In general, $\mu \gg \lambda$, leading to $A \cong 1$

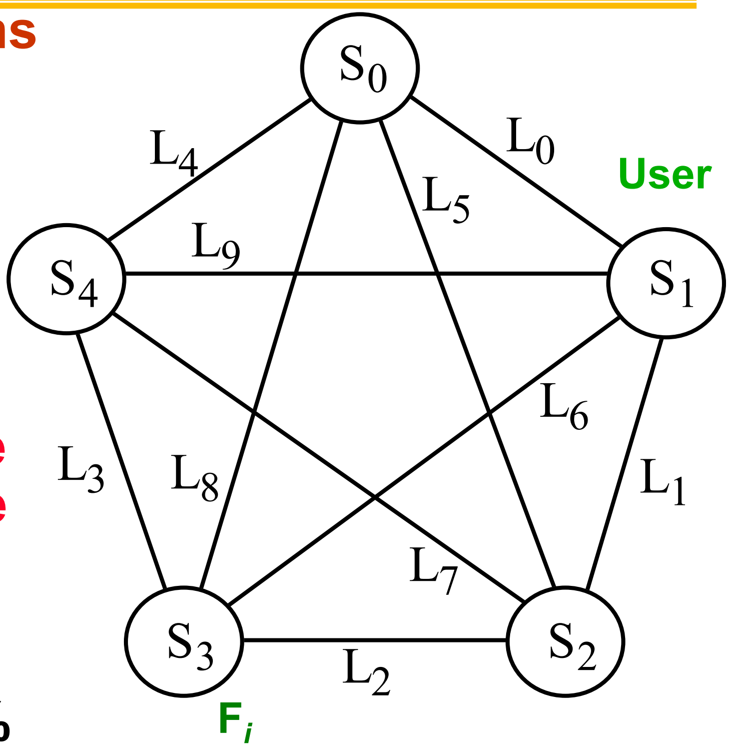
A Motivating Case Study

Data availability and integrity concerns

Distributed DB system with 5 sites
Full connectivity, dedicated links
Only direct communication allowed
Sites and links may malfunction
Redundancy improves availability

***S*: Probability of a site being available**
***L*: Probability of a link being available**

Single-copy availability = SL
Unavailability = $1 - SL$
= $1 - 0.99 \times 0.95 = 5.95\%$



Data replication methods, and a challenge

File duplication: home / mirror sites
File triplication: home / backup 1 / backup 2
Are there availability improvement methods with less redundancy?

Data Duplication: Home and Mirror Sites

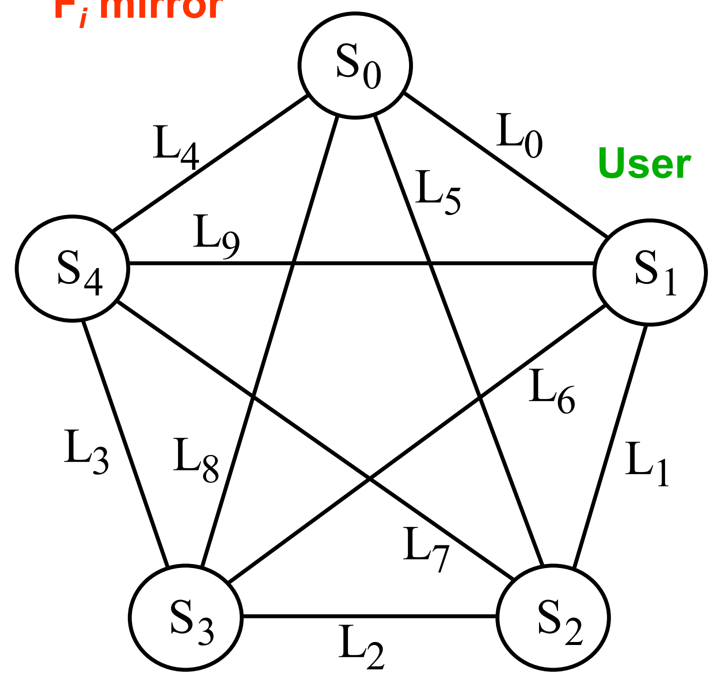
S: Site availability   e.g., 99%
L: Link availability   e.g., 95%

$A = SL + (1 - SL)SL$

Primary site can be reached (for the first SL)

Primary site inaccessible (for $(1 - SL)$)

Mirror site can be reached (for the second SL)



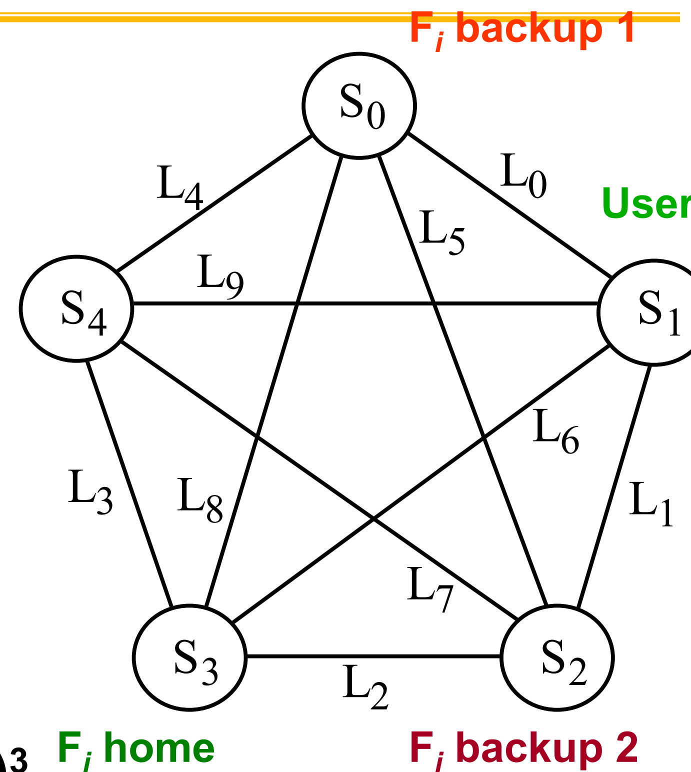
Duplicated availability $= 2SL - (SL)^2$
Unavailability $= 1 - 2SL + (SL)^2$
$= (1 - SL)^2 = 0.35\%$

Data unavailability reduced from 5.95% to 0.35%

Availability improved from ≈ 94% to 99.65%

Data Triplication: Home and Two Backups

S: Site availability e.g., 99%
L: Link availability e.g., 95%



$$A = SL + (1 - SL)SL + (1 - SL)^2 SL$$

Primary site can be reached

Primary site inaccessible

Backup 1 can be reached

Primary and backup 1 inaccessible

Backup 2 can be reached

Triplicated avail. $= 3SL - 3(SL)^2 - (SL)^3$
Unavailability $= 1 - 3SL + 3(SL)^2 + (SL)^3$
$= (1 - SL)^3 = 0.02\%$

Data unavailability reduced from 5.95% to 0.02%

Availability improved from ≈ 94% to 99.98%

Data Dispersion: Three of Five Pieces

$A = (SL)^4 + 4(1 - SL)(SL)^3 + 6(1 - SL)^2(SL)^2$

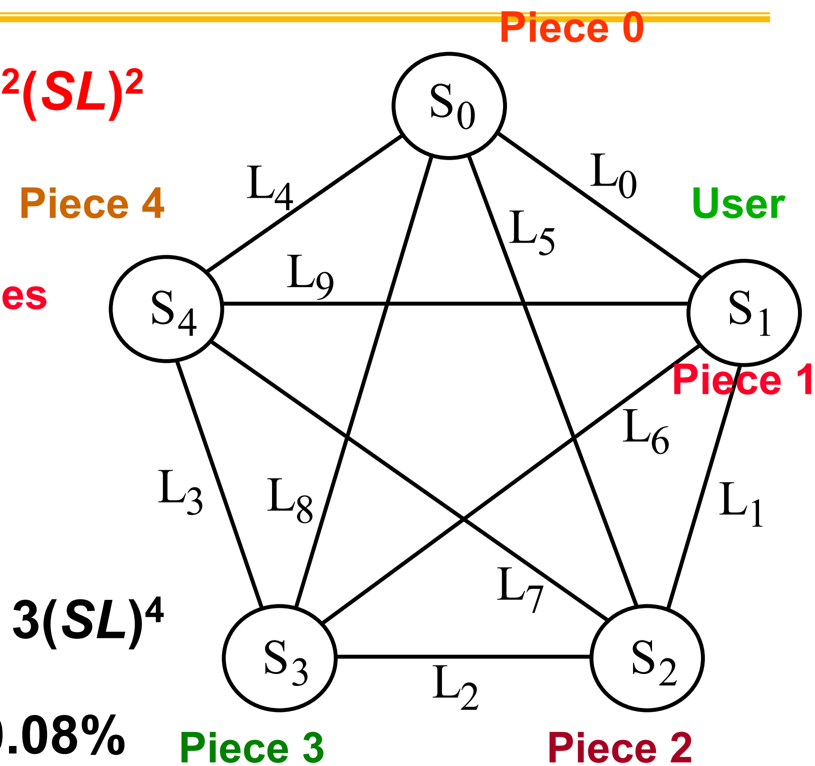
All 4 pieces can be reached

Exactly 3 pieces can be reached

Only 2 pieces can be reached

S: Site availability e.g., 99%

L: Link availability e.g., 95%



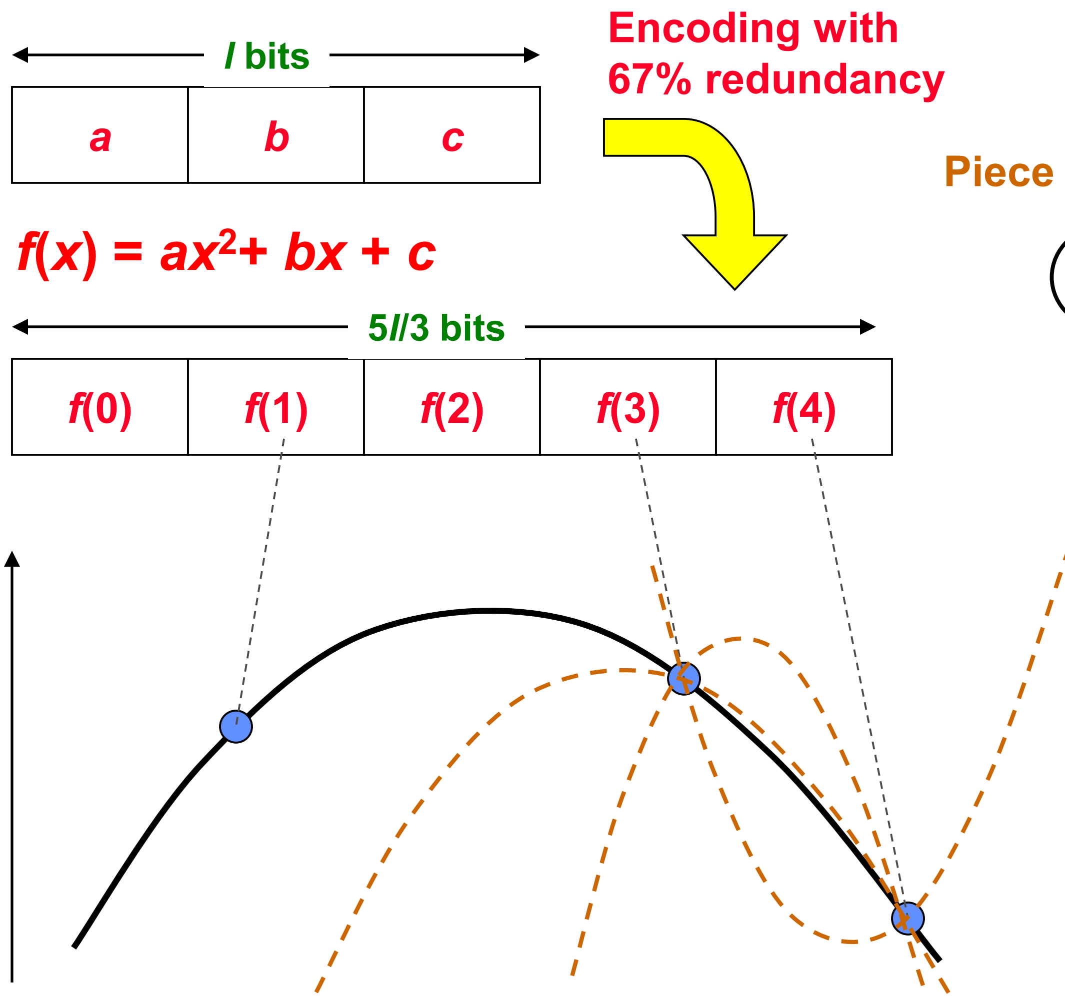
Dispersed avail. = $6(SL)^2 - 8(SL)^3 + 3(SL)^4$

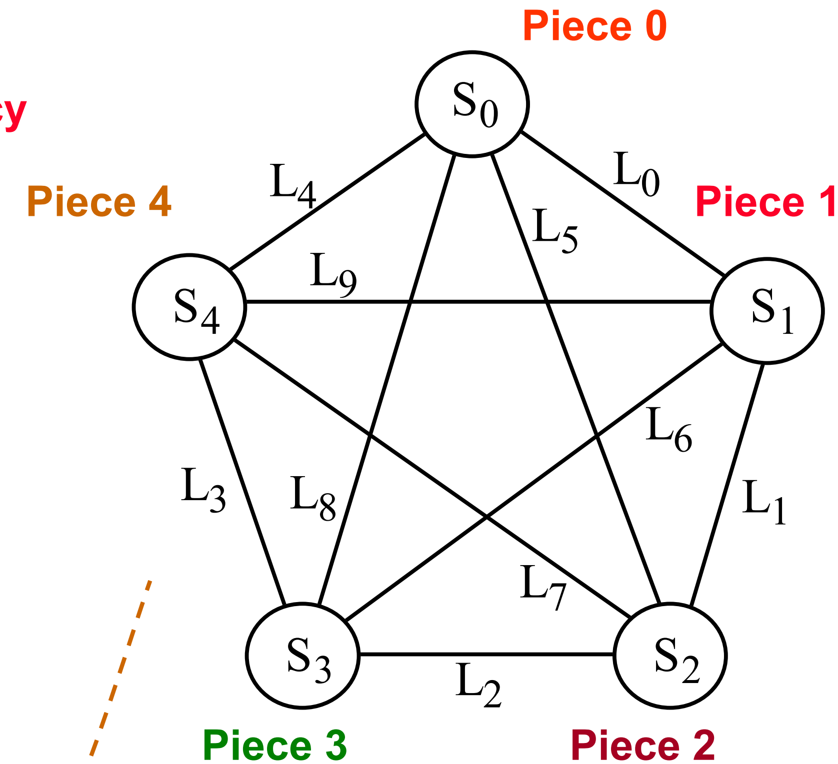
Availability = 99.92%

Unavailability = 1 – Availability = 0.08%

Scheme →	Nonredund.	Duplication	Triplication	Dispersion
Unavailability	5.95%	0.35%	0.02%	0.08%
Redundancy	0%	100%	200%	67%

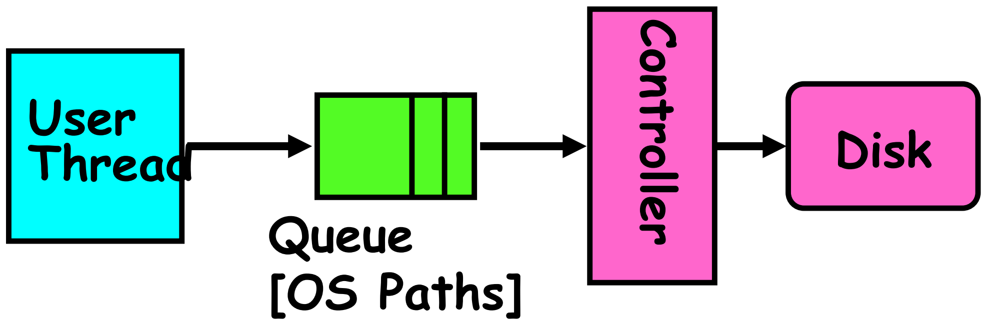
Dispersion for Data Security and Integrity



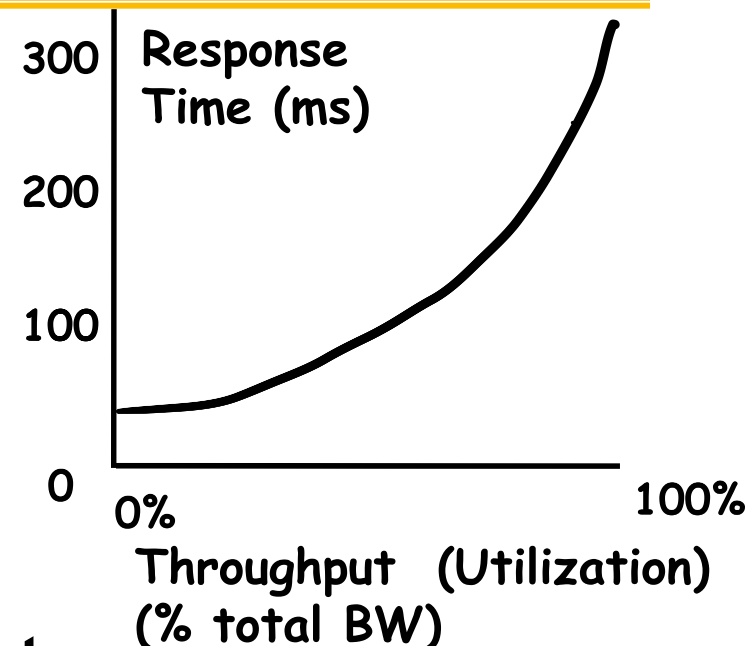


Note that two pieces would be inadequate for reconstruction

Disk I/O Performance

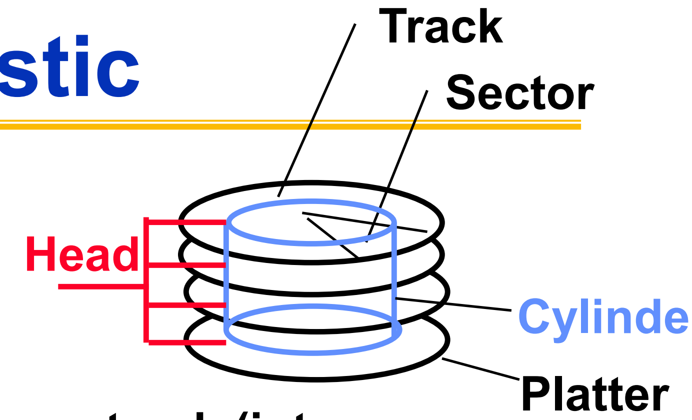


Response Time = Queue+Disk Service Time

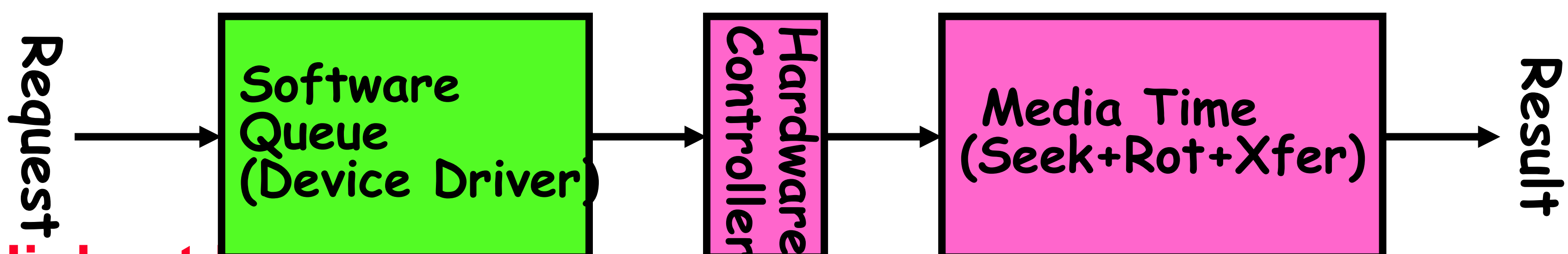


- **Performance of disk drive/file system**
  - Metrics: Response Time, Throughput
  - Contributing factors to latency:
    - » Software paths (can be loosely modeled by a queue)
    - » Hardware controller
    - » Physical disk media
- **Queuing behavior:**
  - Can lead to big increase of latency as utilization approaches 100%

Magnetic Disk Characteristic



- Cylinder: all the tracks under the head at a given point on all surface
- Read/write data is a three-stage process:
  - Seek time: position the head/arm over the proper track (into proper cylinder)
  - Rotational latency: wait for the desired sector to rotate under the read/write head
  - Transfer time: transfer a block of bits (sector) under the read-write head
- Disk Latency = Queueing Time + Controller time + Seek Time + Rotation Time + Xfer Time



- Highest Bandwidth:
  - transfer large group of blocks sequentially from one track

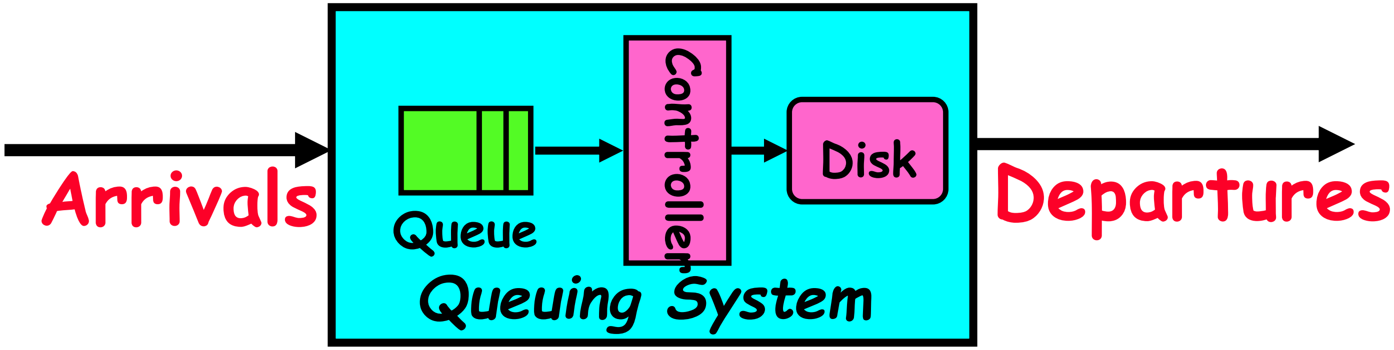
Disk Time Example

- **Disk Parameters:**
  - Transfer size is 8K bytes
  - Advertised average seek is 12 ms
  - Disk spins at 7200 RPM
  - Transfer rate is 4 MB/sec
- **Controller overhead is 2 ms**
- **Assume that disk is idle so no queuing delay**
- **Disk Latency =**
  Queuing Time +
  Seek Time + Rotation Time + Xfer Time + Ctrl Time
- **What is Average Disk Access Time for a Sector?**
  - Ave seek + ave rot delay + transfer time + controller overhead
  - 12 ms + [0.5/(7200 RPM/60s/M)] $\times$1000 ms/s +
    [8192 bytes/(4$\times 10^6$ bytes/s)] $\times$1000 ms/s + 2 ms
  - 12 + 4.17 + 2.05 + 2 = 20.22 ms
- **Advertised seek time assumes no locality: typically 1/4 to 1/3 advertised seek time: 12 ms => 4 ms**

Typical Numbers of a Magnetic Disk

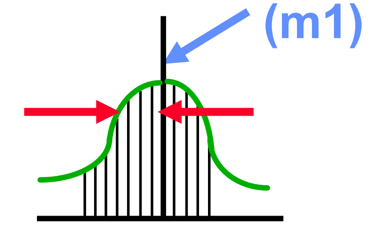
- **Average seek time as reported by the industry:**
  - Typically in the range of 4 ms to 12 ms
  - Due to locality of disk reference may only be 25% to 33% of the advertised number
- **Rotational Latency:**
  - *Most* disks rotate at 3,600 to 7200 RPM (Up to 15,000RPM or more)
  - Approximately 16 ms to 8 ms per revolution, respectively
  - An average latency to the desired information is halfway around the disk: 8 ms at 3600 RPM, 4 ms at 7200 RPM
- **Transfer Time is a function of:**
  - Transfer size (usually a sector): 1 KB / sector
  - Rotation speed: 3600 RPM to 15000 RPM
  - Recording density: bits per inch on a track
  - Diameter: ranges from 1 in to 5.25 in
  - Typical values: 2 to 50 MB per second
- **Controller time?**
  - Depends on controller hardware—need to examine each case individually

Introduction to Queuing Theory



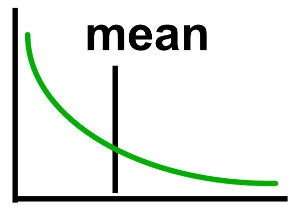
- **What about queuing time??**
  - Let's apply some queuing theory
  - Queuing Theory applies to long term, steady state behavior ⇒ Arrival rate = Departure rate
- **Little's Law:**
  Mean # tasks in system = arrival rate x mean response time
  - Observed by many, Little was first to prove
  - Simple interpretation: you should see the same number of tasks in queue when entering as when leaving.
- **Applies to any system in equilibrium, as long as nothing in black box is creating or destroying tasks**
  - Typical queuing theory doesn't deal with transient behavior, only steady-state behavior

Background: Use of random distributions



Distribution of service times

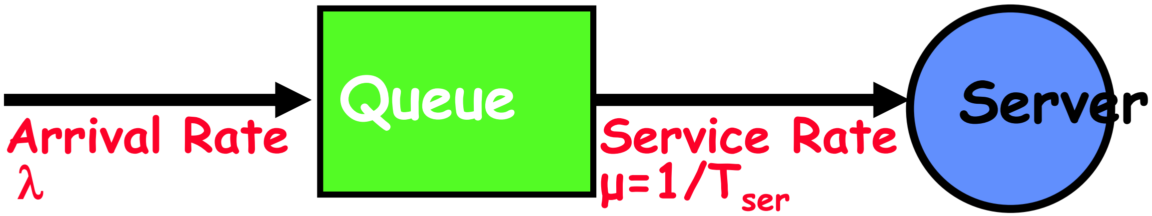
- **Server spends variable time with customers**
  - Mean (Average) m1 = $\Sigma p(T) \times T$
  - Variance $\sigma^2 = \Sigma p(T) \times (T\text{-}m1)^2 = \Sigma p(T) \times T^2\text{-}m1 = E(T^2)\text{-}m1$
  - Squared coefficient of variance: $C = \sigma^2/m1^2$
    Aggregate description of the distribution.
- **Important values of C:**
  - No variance or deterministic $\Rightarrow$ C=0
  - "memoryless" or exponential $\Rightarrow$ C=1
    - » Past tells nothing about future
    - » Many complex systems (or aggregates) well described as memoryless
  - Disk response times $C \approx 1.5$ (majority seeks < avg)



Memoryless

- **Mean Residual Wait Time, m1(z):**
  - Mean time must wait for server to complete current task
  - Can derive m1(z) = ½m1×(1 + C)
    - » Not just ½m1 because doesn't capture variance
  - C = 0 $\Rightarrow$ m1(z) = ½m1; C = 1 $\Rightarrow$ m1(z) = m1

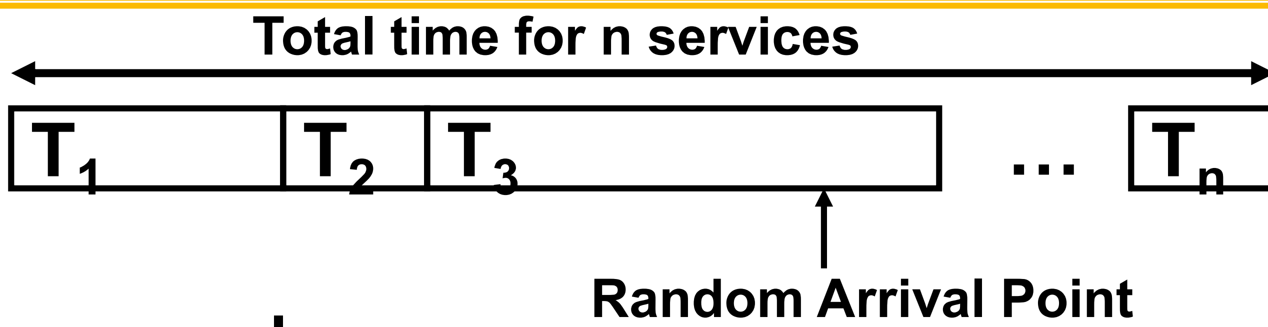
A Little Queuing Theory: Mean Wait Time



- **Parameters that describe our system:**
  - λ: mean number of arriving customers/second
  - T_{ser}: mean time to service a customer ("m1")
  - C: squared coefficient of variance = $\sigma^2/m1^2$
  - μ: service rate = $1/T_{ser}$
  - u: server utilization ($0 \le u \le 1$): $u = \lambda/\mu = \lambda \times T_{ser}$
- **Parameters we wish to compute:**
  - T_q: Time spent in queue
  - L_q: Length of queue = $\lambda \times T_q$ (by Little's law)
- **Basic Approach:**
  - Customers before us must finish; mean time = $L_q \times T_{ser}$
  - If something at server, takes m1(z) to complete on avg
    - » Chance server busy = u $\Rightarrow$ mean time is u $\times$ m1(z)
- **Computation of wait time in queue (T_q):**

$$T_q = L_q \times T_{ser} + u \times m1(z)$$

Mean Residual Wait Time: m1(z)



- **Imagine n samples**
  - **There are $n \times P(T_x)$ samples of size T_x**
  - **Total space of samples of size T_x:** $T_x \times n \times P(T_x) = n \times T_x P(T_x)$
  - **Total time for n services:** $\sum_x n \times T_x P(T_x) = n \times T_{ser}$
  - **Chance arrive in service of length T_x:** $\dfrac{n \times T_x P(T_x)}{n \times T_{ser}} = \dfrac{T_x P(T_x)}{T_{ser}}$
  - **Avg remaining time if land in T_x:** $\frac{1}{2} T_x$
  - **Finally: Average Residual Time m1(z):**

$$\sum_x \left(\frac{1}{2} T_x\right)\left(\frac{T_x P(T_x)}{T_{ser}}\right) = \frac{1}{2}\frac{E(T^2)}{T_{ser}} = \frac{1}{2} T_{ser}\left(\frac{\sigma^2 + T_{ser}{}^2}{T_{ser}{}^2}\right) = \frac{1}{2} T_{ser}(1 + C)$$

A Little Queuing Theory: M/G/1 and M/M/1

- **Computation of wait time in queue (T_q):**

  $T_q = L_q \times T_{ser} + u \times m1(z)$ Little's Law

  $T_q = \lambda \times T_q \times T_{ser} + u \times m1(z)$

  $T_q = u \times T_q + u \times m1(z)$ Defn of utilization (u)

  $T_q \times (1 - u) = m1(z) \times u \Rightarrow T_q = m1(z) \times u/(1\text{-}u) \Rightarrow$

  $T_q = T_{ser} \times \frac{1}{2}(1+C) \times u/(1 - u)$
- **Notice that as $u \to 1$, $T_q \to \infty$!**
- **Assumptions so far:**
  - System in equilibrium; No limit to the queue: works First-In-First-Out
  - Time between two successive arrivals in line are random and memoryless: (M for C=1 exponentially random)
  - Server can start on next customer immediately after prior finishes
- **General service distribution (no restrictions), 1 server:**
  - Called M/G/1 queue: $T_q = T_{ser} \text{ x } \frac{1}{2}(1+C) \text{ x } u/(1 - u))$
- **Memoryless service distribution (C = 1):**
  - Called M/M/1 queue: $T_q = T_{ser} \text{ x } u/(1 - u)$

A Little Queuing Theory: An Example

- **Example Usage Statistics:**
  - User requests 10 x 8KB disk I/Os per second
  - Requests & service exponentially distributed (C=1.0)
  - Avg. service = 20 ms (From controller+seek+rot+trans)
- **Questions:**
  - How utilized is the disk?
    - » Ans: server utilization, $u = \lambda T_{ser}$
  - What is the average time spent in the queue?
    - » Ans: T_q
  - What is the number of requests in the queue?
    - » Ans: L_q
  - What is the avg response time for disk request?
    - » Ans: $T_{sys} = T_q + T_{ser}$
- **Computation:**

λ (avg # arriving customers/s) = 10/s

T_{ser} (avg time to service customer) = 20 ms (0.02s)

u (server utilization) = $\lambda \times T_{ser}$ = 10/s x .02s = 0.2

T_q (avg time/customer in queue) = $T_{ser} \times u/(1 - u)$
= 20 x 0.2/(1-0.2) = 20 x 0.25 = 5 ms (0 .005s)

L_q (avg length of queue) = $\lambda \times T_q$=10/s x .005s = 0.05

T_{sys} (avg time/customer in system) =$T_q + T_{ser}$= 25 ms