
EE282 Lecture 2: Statistics for Architects

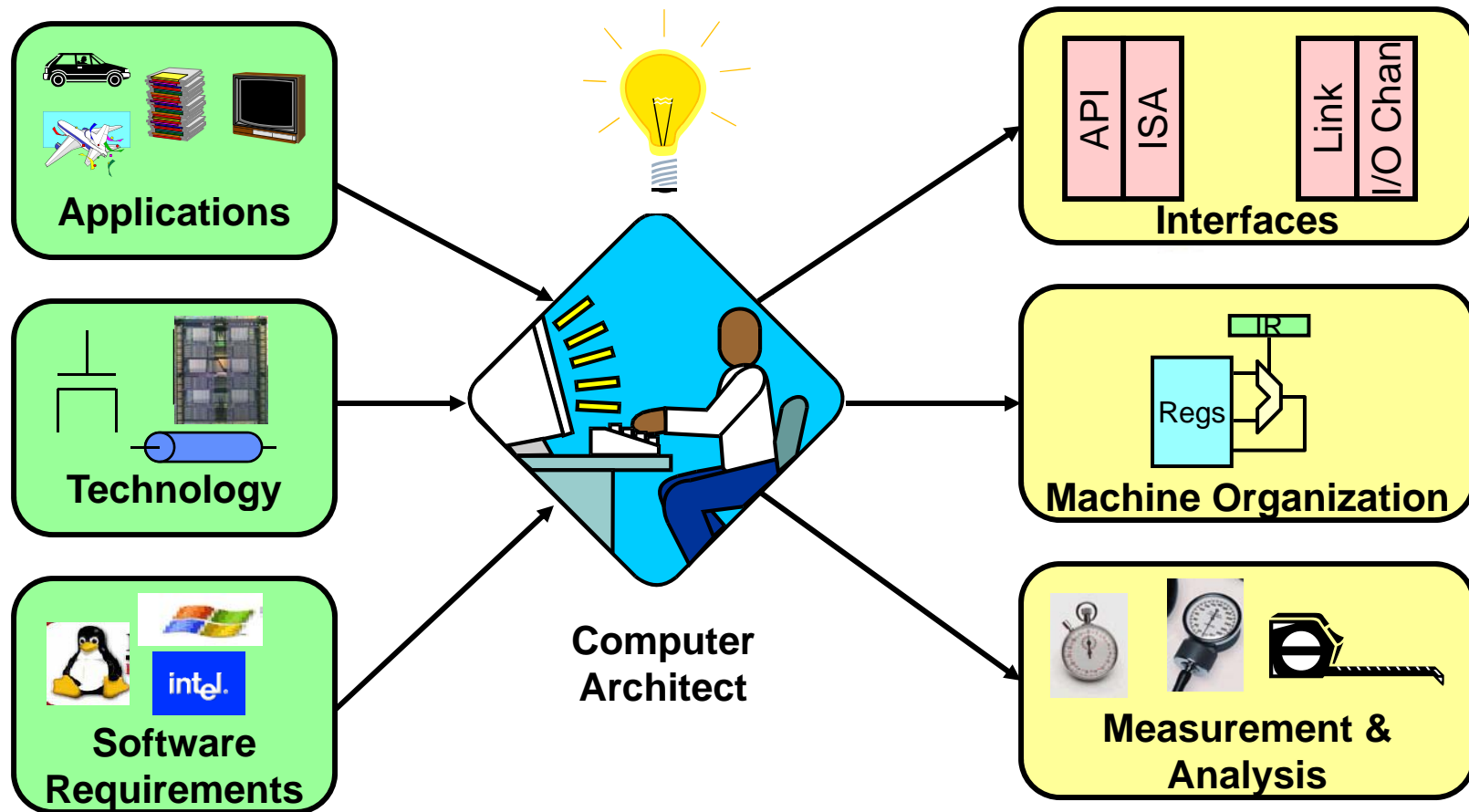
Department of Electrical Engineering
Stanford University

<http://eeclass.stanford.edu/ee282>

Announcements

- Double-check your prerequisites
- Double-check you can make the quiz/HW/PA deadlines
 - Quiz 2 is on 12/8th...
- Reading assignment for each lecture is posted online
 - Book chapters/section, required & optional papers
- Start searching for a HW/PA group
 - HW1 out in a week
- Do not try to register with <http://eeclass.stanford.edu/ee282> yet
 - We'll let you know when registration is open
- Review session: Fri 9/26, 11am – 11.50am, Skilling 193
 - Processor architecture review

Summary of Last Lecture



The science/art of constructing efficient systems for computing tasks

Key Architecture Techniques (1)

- Pipelining
 - What is it and what does it work?
 - Does it improve latency or bandwidth?
- Parallelism processing
 - What makes it possible?
 - What should be aware of?
- Out-of-order execution
 - In what order should you execute the instructions in a program?
- Speculation
 - Why would speculation be useful?

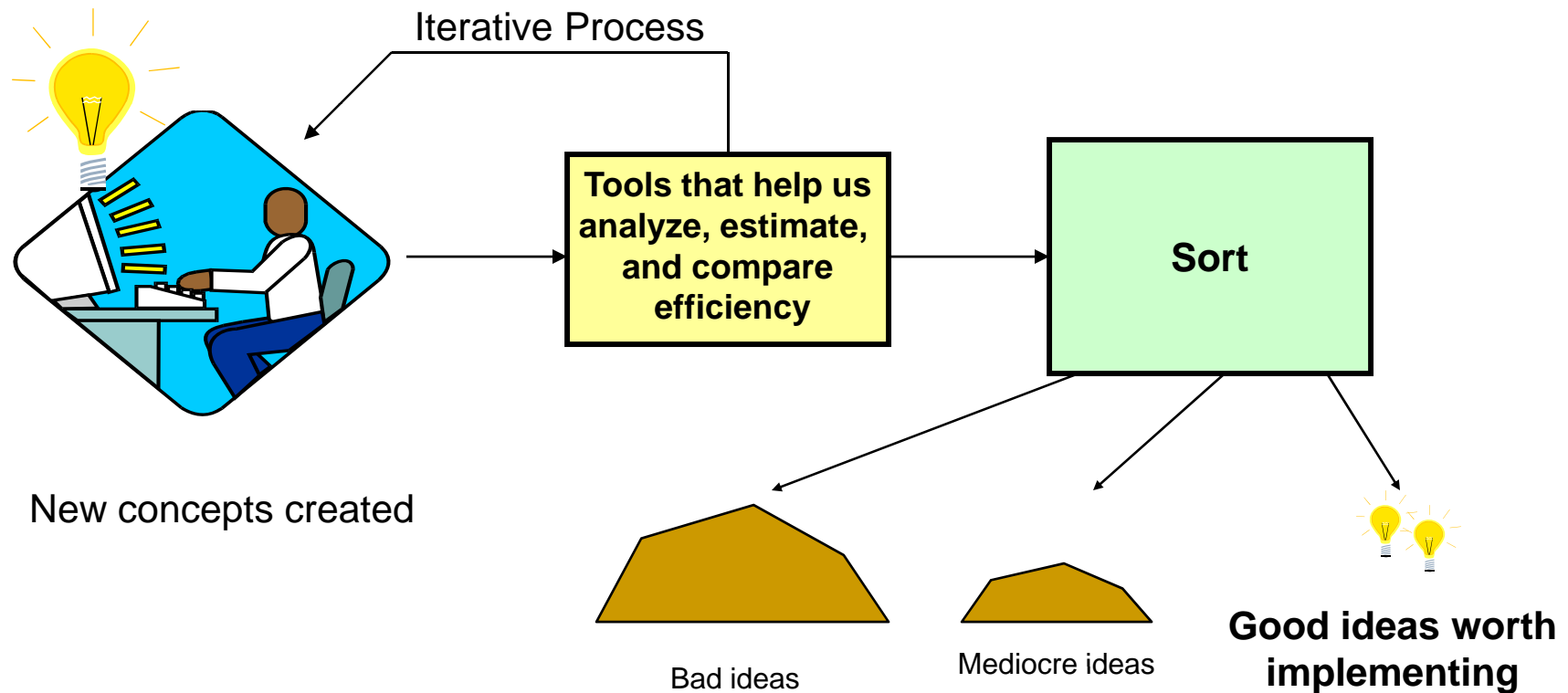
Key Architecture Techniques (2)

- Caching
 - Why do caches reduce memory access latency?
 - Are caches applicable only to processor design?
- Indirection
 - How do you find your doctor's number?
 - Examples in system design?
- Amortization
 - How can you amortize the high cost of memory accesses?
 - Other examples?

Today's Menu

- Benchmarks
- Averages & pitfalls
- Basic statistics for architects

Architects use a Quantitative Approach



Benchmarks

- Ideally, evaluate a system with the real applications
- Benchmark suite: the substitute to the real workload
 - A collection of programs for evaluation and comparisons
- Components of a benchmark
 - The programs
 - In some agreed form: specification, source code, binary, ...
 - The input and output datasets
 - The measurements rules
 - How to compile, run, summarize, measure, ...
 - Even with very strict rules, people often cheat
 - The metrics

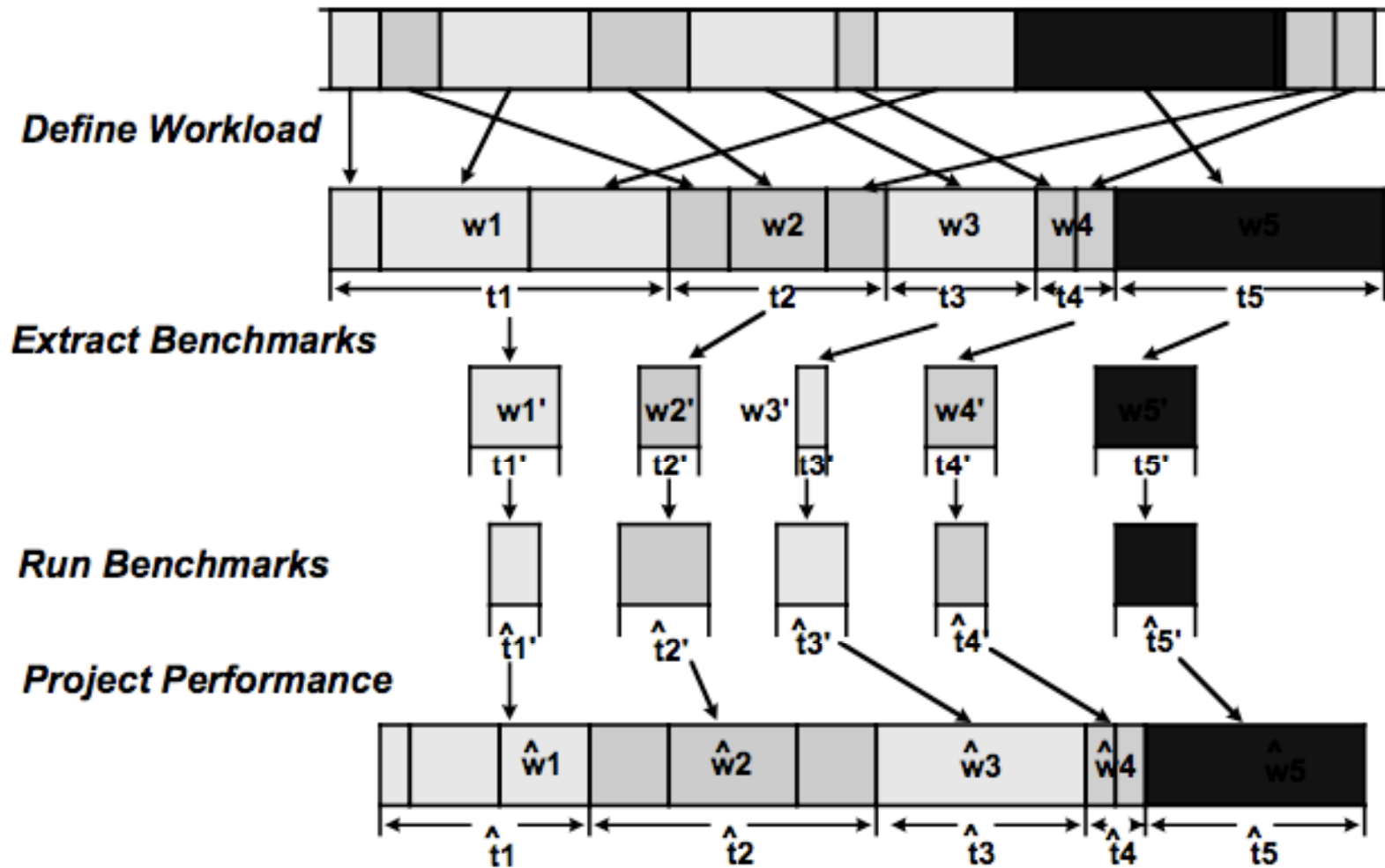
Who Benchmarks and Why

- Computer designers
 - To evaluate new HW/SW systems
- Software designers
 - To understand where to focus effort on OS, compiler, or apps work
- Computer buyers
 - To evaluate potential purchases, capacity planning, ...
- Computer magazines or benchmarks
 - To sell advertisements
- Industry consortia
 - SPEC, TPC, EEMBC, ...
 - To define meaningful benchmarks & avoid coercion/waste of silly ones
- Researchers
 - To publish, keep students busy... 😊

Benchmarking Steps (Ideal)

1. Define workload
2. Extract benchmarks from applications
3. Choose performance metric
4. Execute benchmarks on target machine(s)
5. Project workload performance on target machines & summarize results

Benchmarking Process



Choosing Benchmark Programs

- Benchmark types
 - Kernels: small key pieces of real applications
 - Toy benchmarks: 100-line programs
 - Synthetic benchmarks: fake programs designed to stress a component
 - Real applications: applications for one or more domains
 - Harder to cheat with real applications
 - Representative because they are real
 - But require more work and can get dated
- How to cheat the benchmark
 - Sub-setting (run half the programs or datasets)
 - Use benchmark-specific optimizations
 - Run the benchmark for the wrong applications domain
 - Use older version of the benchmark
 - Summarize performance creatively (e.g., wrong weights)

Examples of Benchmark Suites

- Desktop/workstation: SpecCPU (INT & FP)
- Scientific computing: Linpack, SpecOMP, SpecHPC, SPLASH, NAS ...
- Embedded benchmarks: EEMBC, Dhrystone, ...
- Enterprise computing
 - TCP-C, TPC-W, TPC-H
 - SpecJbb, SpecSFS, SpecMail, Streams,...
 - MinuteSort, PennySort, Joulesort, ...
- Other
 - 3Dmark, ScienceMark, Winstone, iBench, AquaMark, ...
- Caveats
 - Your system will be as good as your benchmarks
 - Make sure you know what the benchmark is designed to measure
 - Performance is not the only metric for computing systems
 - Predicting the real-world programs/datasets for 3 years from now

Designers Paradox

- Consider 2 application domains and 3 computer designs

Application Domain	Computer 1 time (sec.)	Computer 2 time (sec.)	Computer 3 time (sec.)
Domain 1	10	100	20
Domain 2	100	10	20
Total Time	110	110	40

- Computer 3 gives best overall performance
 - ***BUT WON'T SELL***
 - Customers in domain 1 will choose Computer 1 and customers in domain 2 will choose Computer 2

The SPEC CPU Benchmark Suite

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	jpeg		
Games/path finding	astar	eon	m88ksim		
Search gene sequence	hammer	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	deall				nasa7
Quantum chemistry	gams				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	ampp			
Magneto hydrodynamics (astrophysics)	zeusmp	lucas			
		fma3d			
		sixtrack			

© 2007 Elsevier, Inc. All rights reserved.

Summarizing Performance

- Combining different benchmark results into 1 number
 - Sometimes misleading, always controversial...and inevitable
 - We all like quoting a single number
- 3 types of means
 - Arithmetic: for times
 - Harmonic: for rates
 - Geometric: for ratios
- Remember: benchmark results are samples of a population
 - Distributions
 - Confidence intervals

(Weighted) Arithmetic Mean

$$\frac{1}{n} \sum_{i=1}^n (Weight_i) \cdot Time_i$$

	Machine A	Machine B	Speedup (B over A)
Prog. 1 (sec)	1	10	0.1
Prog. 2 (sec)	1000	100	10
Mean (50/50)	500.5	55	9.1
Mean (75/25)	250.75	32.5	7.7

- If you know your exact workload (benchmarks & relative frequencies), this is the right way to summarize performance.

(Weighted) Harmonic Mean

$$HM = \frac{n}{\sum_{i=1}^n \frac{(Weight_i)}{Rate_i}}$$

- Exactly analogous, but for averaging rates (work / unit time).

Geometric mean: used for ratios

$$GM = \left(\prod_{i=1}^n Ratio_i \right)^{\left(\frac{1}{n}\right)}$$

- Used by SPEC CPU suite. To avoid questions of how to weight benchmarks, normalize Machine A's performance on each benchmark i to the performance of some reference machine *Ref*:

$$SPECRatio_i = \frac{Time_i, MachineA}{Time_i, Ref}$$

and report GM of ratios as final result.

Pros and Cons of Geometric Mean

- *Pros:* Ratio of means = mean of ratios

$$GM\left(\frac{X}{Y}\right) = \frac{GM(X)}{GM(Y)}$$

- *Cons:*
 - No intuitive physical meaning
 - Can't be related back to execution time

Means Revisited

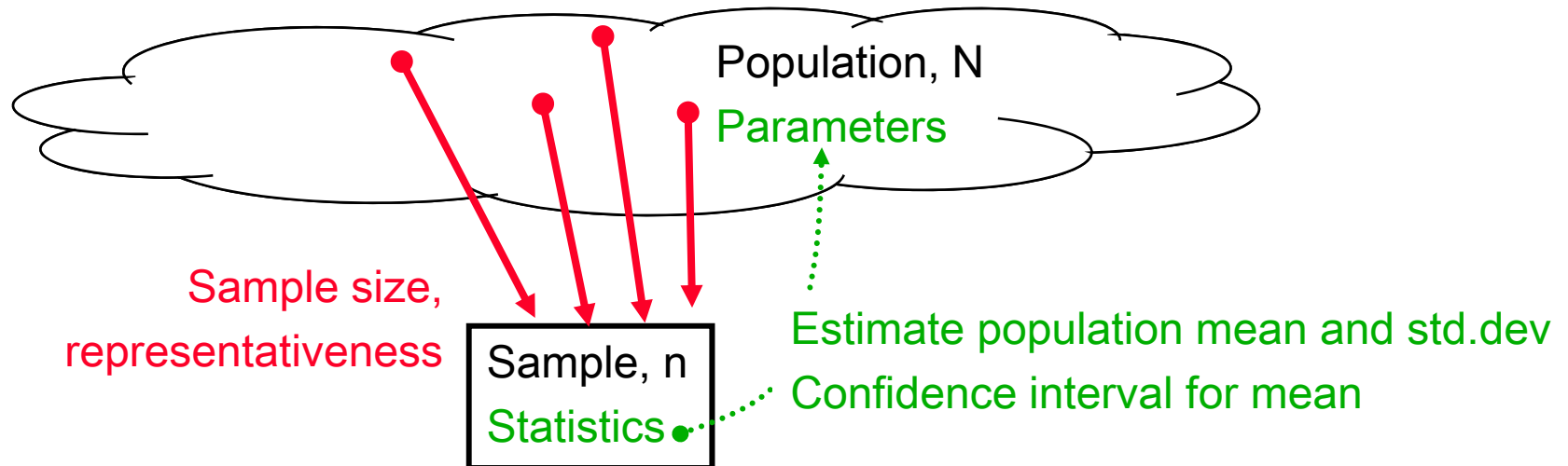
- Geometric mean gives equal reward for speeding up all benchmarks
 - Easier to speedup programs with more inherent parallelism
 - \Rightarrow the already fast programs get faster
- Harmonic mean gives greater reward for speeding up the slow benchmarks
 - Consistent with Amdahl's law
 - But you can pay for parallelism \Rightarrow Hmean at a “disadvantage”
 - Will become a greater issue with parallel benchmarks
- Arithmetic mean gives greater reward for speeding up already-fast benchmark

Statistics for Architects

- Means are nice, but they don't tell you the whole truth
 - More info when you run 1,000 programs on a machine
 - More info when you run one program on 1,000 machine configurations
- Next few slides: basic tools for statistics for computer architectus
 - How to observe large collections of experiment results
 - How to represent large collections of experiment results
 - Modified from J. Mashey, "Summarizing Performance is No Mean Feat"
- Note: take a stats class for the full story...

Populations and Samples

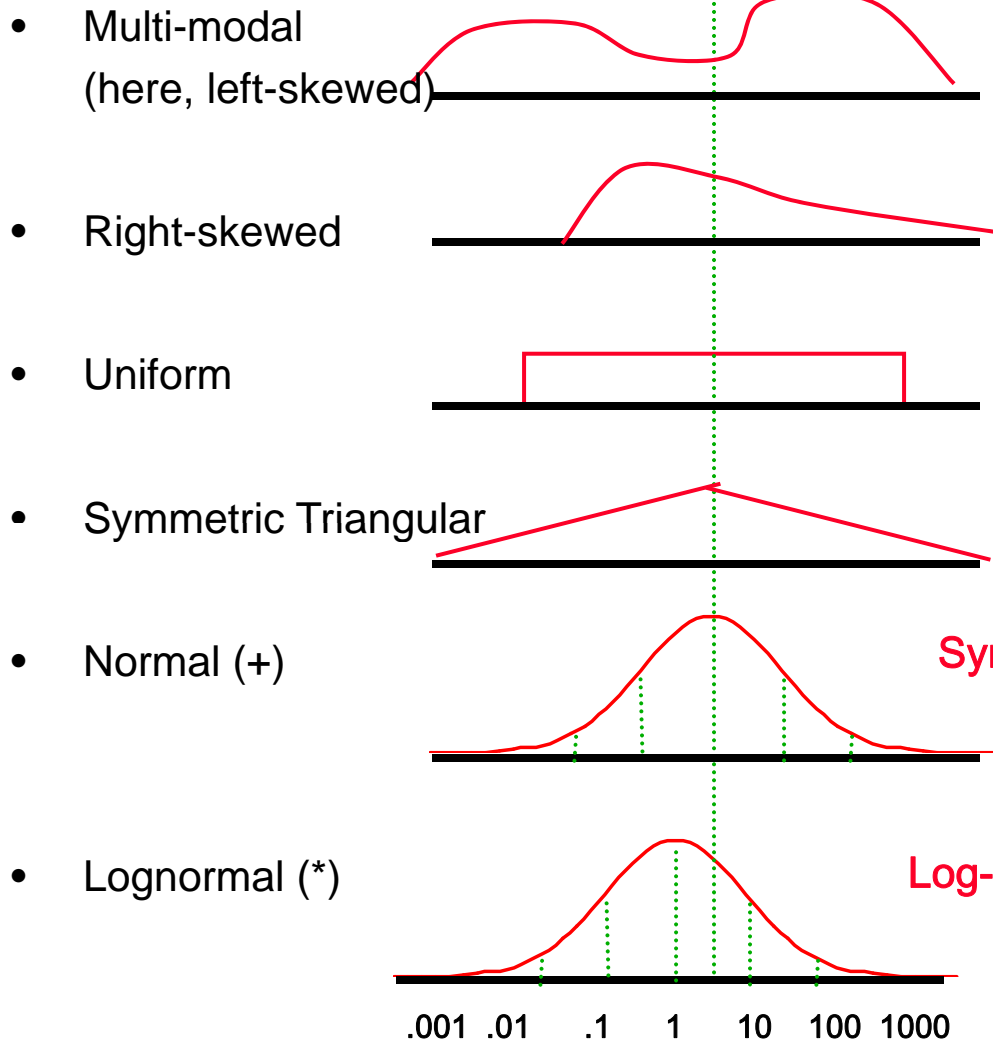
- *Population*: set of observations measured for ALL members of group
 - Forms a *distribution*
 - Uncertainty: individual measurement errors
- *Sample*: subset of population
 - Compute *statistics*
 - Extra uncertainty: small samples or selection bias



Basic Assumptions

- Measurements are repeatable
 - Same program + input gives same performance
 - Valid for most programs/machines – worth verifying
 - Watch out for non-deterministic programs
- Choice of input doesn't change relative performance of different machines
 - Usually true – counterexample?
- Number of benchmarks in suite (sample size) is large enough to yield good conclusions
 - Confidence intervals help verify this
- Benchmarks are representative and not a biased sample
 - Can only address qualitatively

Data Distributions with Same Arithmetic Mean



- Uncertain
- Awful, but hope...
- OK, not much central tendency
- Good, more central tendency
- Terrific! Statistics toolkit
- Terrific! Statistics toolkit

Symmetric

Log-symmetric



General Distribution Descriptions

- *Mean*: measure of central tendency, 1st moment
- *Variance*: measure of dispersion, 2nd moment
 - The amount of variation in a distribution
- *Standard deviation*: measure of dispersion, same scale as Mean
 - Average distance from mean of samples
- *Excel functions at left, when exist (OpenOffice.org Calc mostly same)*

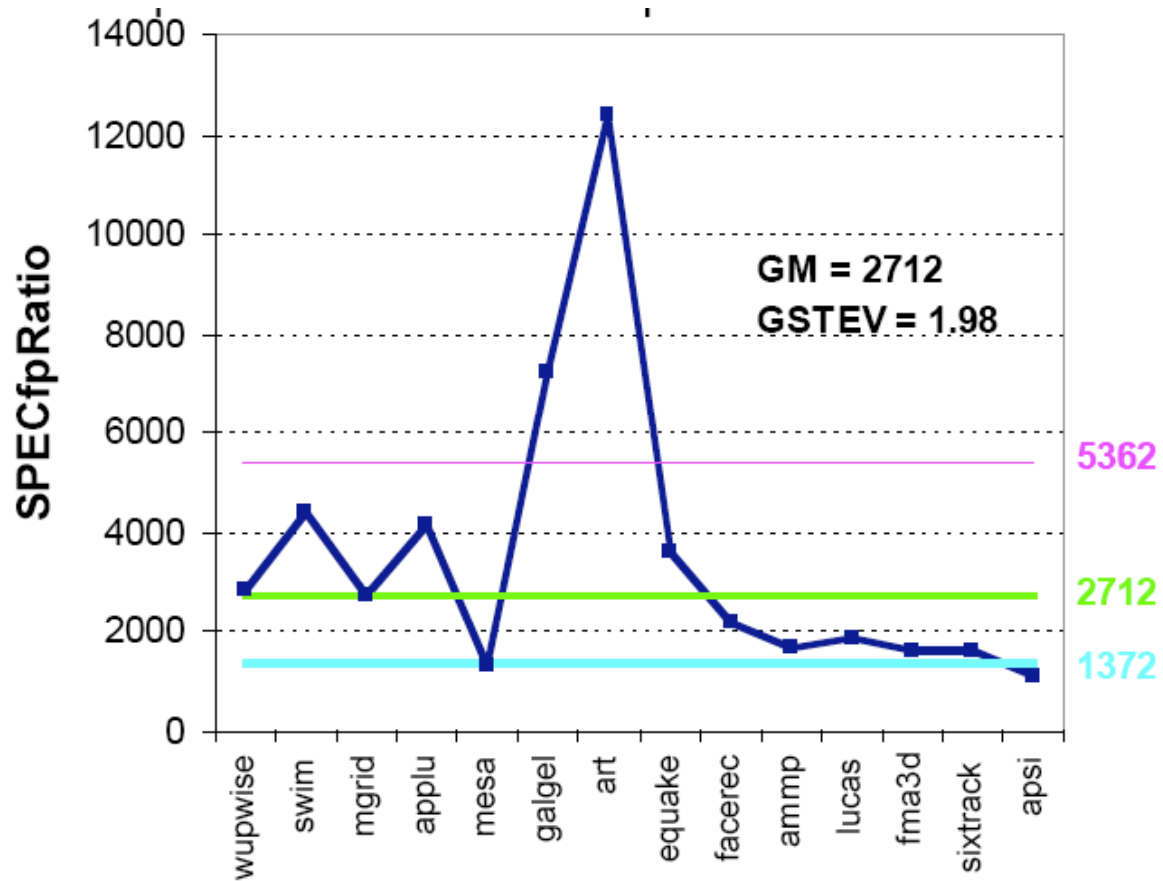
$$\text{AVERAGE} : AM = \mu = \frac{1}{N} \sum_{i=1}^N x_i = \text{Arithmetic Mean}$$

$$\text{VARP} : \sigma^2 = \frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2$$

$$\text{SDEVP} : \sigma = \sqrt{\sigma^2}$$

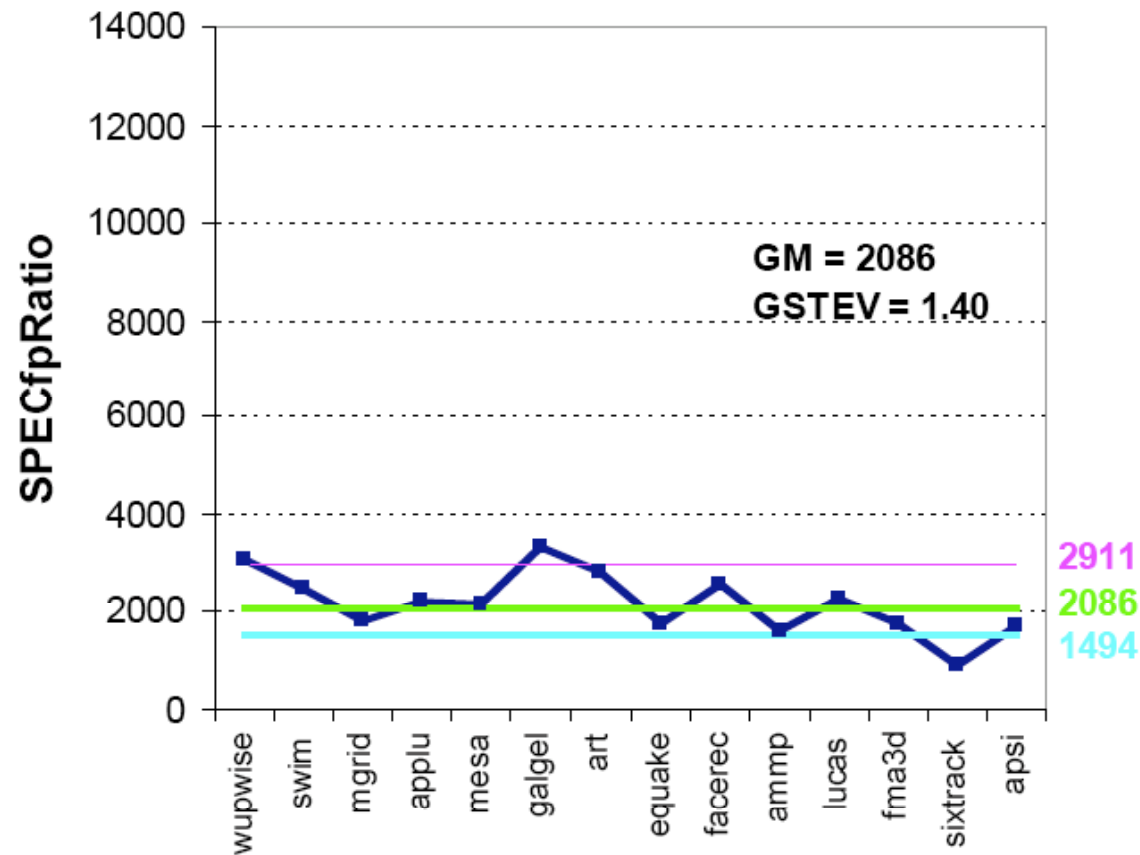
Example Standard Deviation (1/2)

- GM and multiplicative StDev of SPECfp2000 on Itanium 2



Example Standard Deviation (2/2)

- GM and multiplicative StDev of SPECfp2000 on Athlon

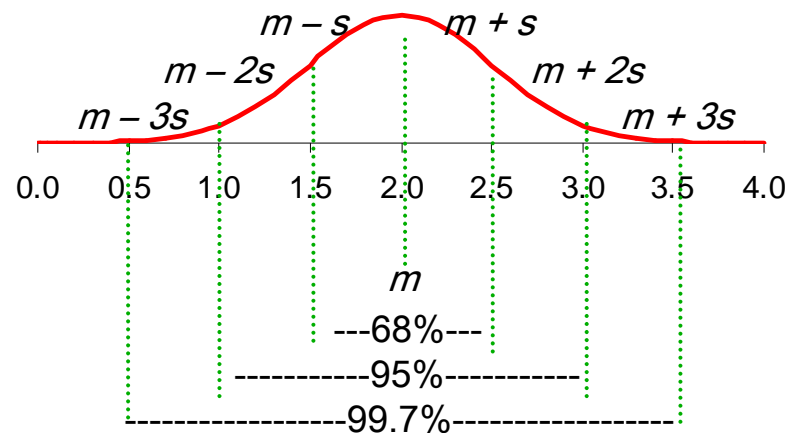


Comments on Example

- Standard deviation of 1.98 for Itanium 2 is much higher (1.40 for Athlon) so results will differ more widely from the mean, and therefore are likely less predictable
- Falling within one standard deviation:
 - 10 of 14 benchmarks (71%) for Itanium 2
 - 11 of 14 benchmarks (78%) for Athlon
- Thus, the results are quite compatible with a lognormal distribution (expect 68%)

The Familiar Normal (Gaussian) Distribution

- Arises from large number of small *additive* effects
- Completely specified by mean m and standard deviation σ
- Familiar, useful properties ... *never* automatically assume normal, but hope
- 68% within $m \pm \sigma$; 95% within $\pm 2\sigma$; 99.7% within $m \pm 3\sigma$
- Symmetric around the mean = intuitive measure of central tendency



Confidence Intervals of Population Mean

- *If* normal population, can compute *confidence intervals* for mean
- Example: 90% confidence interval
 - Assuming that the sample mean is normally distributed, what is K so that our sample mean is within K units of the real mean with probability 90%
- Interval improves (gets smaller) with smaller σ , larger sample n
- Quick approximation
 - Assuming $s = \sigma/\sqrt{n}$, where n is the size of the sample

Confidence Interval	Result
90%	+/- 1.65s
95%	+/- 1.96s
99%	+/- 2.58s

- Quick & dirty check
 - Multiply (s) by 2 to get an approximate 95% confidence interval

Handling Non-Normal Distributions

- Normal is so useful ... but nothing guarantees it, *so must check*
- If isn't normal, try to transform to one that could be
 - $X_i^* = f(x_i)$ transform; use whatever works, $f(x) = 1/x$, $f(x) = \ln(x)$, etc
 - Compute mean, standard deviation, other statistics from X_i^*
Check normality!
 - Back-transform mean (and other metrics that can be) via f^{-1}
 - *If X_i^* turns out to be normal, insight can be gained from understanding why that particular transform works*
 - *Widely-applied, standard statistical data analysis method*

Central Limit Theorem

- What if we know nothing about the initial distribution?
- Central Limit Theorem:
 - If t is the sample mean of a random variable X , with an *unknown* distribution with mean m and variance s^2 , then the distribution of t approaches a normal distribution $N(m, s/\sqrt{n})$ as n becomes large.
- What does this mean:
 - We can ignore the underlying distribution of X , as long as we have enough samples and they are independent and identically distributed. The distribution of the sample mean always approaches the normal distribution.

Summary: Quantitative Metrics

- Benchmarking
 - Your results will be as good as your benchmarks
- How to summarize performance
 - Arithmetic mean for times
 - Harmonic mean for throughput
 - Geometric mean for ratio
- Beyond averages
 - Look at statistics for the overall distribution
 - Calculate confidence intervals