The art of Measurement

1. Overview of Evaluation Techniques

1.1. Analytical Modeling

- Based on a rigorous mathematical models
- For example: Markov Chain, Queue Theory

Positive

- Provides the best insight into the effects of different parameters and their interaction
- Can be done before the system is built and takes a short time

Negative

- these models are rarely accurate
    - Depend on the quality and correctness of assumptions that made to simplify the model

1.2. Simulation

simulate the system operation

Positive

- **full control** of simulation model, parameters, level of detail (high flexibility
- Can be done **before the system is built**

Negative

It may still include simplifying **assumptions**

1.3. Measurement

Implement the system in full and measure its performance directly

positive

The most **convincing**

Negative

has high costs

2. Model-based Evaluation

2.1. How to design a model-based evaluation?

What is the goal

Can you achieve what you want to achieve by measurement?

What to measure

- Metrics: Does your metric really measure what you want to measure
- Workload: What type of input workload should you consider

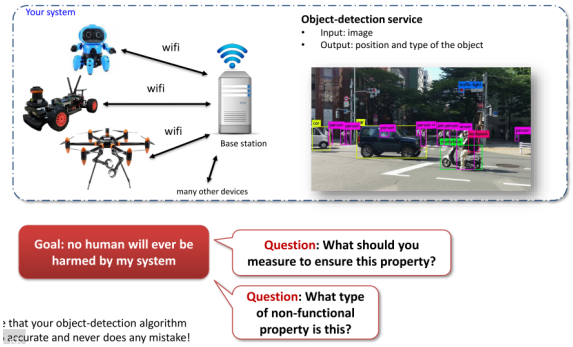
How to measure

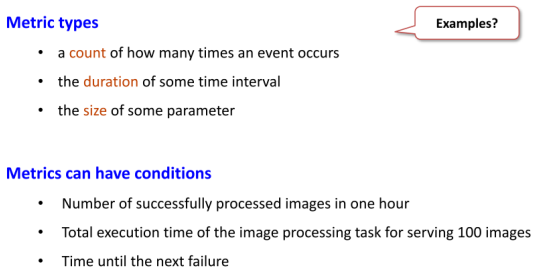
- What **tools** should/can you use for measuring a metrics?
- **Considerations** while doing measurement

How to summarize and present the output?

- How to **summarize** your thousands of data points
- What are the **statistical methods/concepts** that can help you

2.2. An example





2.3. System Parameters (configuration)

System parameters are configurations and parameters that impact a system's performance **independent of the system's input workload**

Example:

Metric is the round trip time of the object tracking

Sends its position to the base station

Navigates the device through the campus

Question: what are the system parameters in our example?

System parameters for our example:

- Speed of: base station's CPU, robot's microcontroller, and network.
- Scheduling policy on both ends.
- Network protocol.
- Operating system overhead for interfacing with the network.
- Reliability of the network affecting the number of retransmissions required.
- ...

2.4. Workload model

- Workload model defines the model of input workload to the system
- It is usually a representation of the **actual usage of the system** after deployment

Question: what are the workload parameters?

Sends its position to the base station

Navigates the device through the campus

Workload parameters for this example:

- Number of IoT devices (user)
- Frequency of sending requests to the base station
  - and distribution (periodic, Poisson, ...)
- Complexity of the navigation task at different locations
  - Example: around cross roads is harder than straight roads
  - Proximity (presence) of other devices when it reaches to a cross road
- Other load on the robot's microcontroller and base station CPUs
- Other load on the network

2.5. Factors

- **Factors**: Parameters that are varied during the evaluation
- **Levels**: Values of a factor

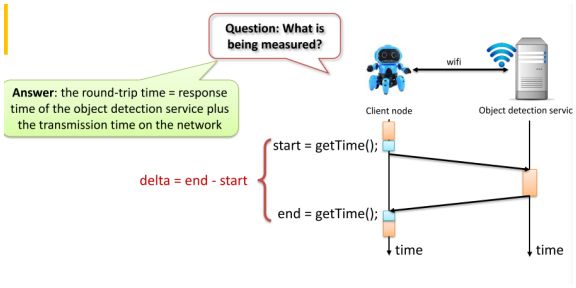
Example:

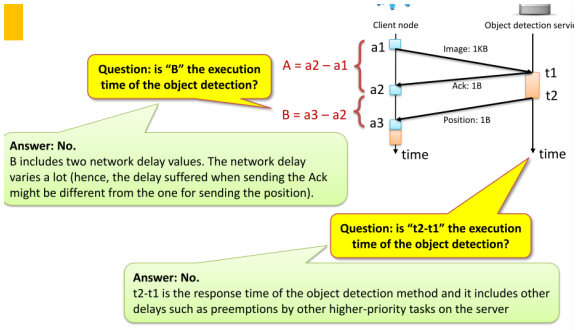
- Image size: {10KB, 100KB, 1MB}
- Frequency of calls to service:
  - Periodic: 1 every {1, 10, 100, 1000} milliseconds
  - Exponential distribution with rate: {0.1, 0.2, 0.3, 0.4, 0.5}

Assumption:

- Fixed: type of CPU and operating system.
- Measure under no other load on the hosts and the network.

3. Metrics





4. Example of practical considerations when measuring time

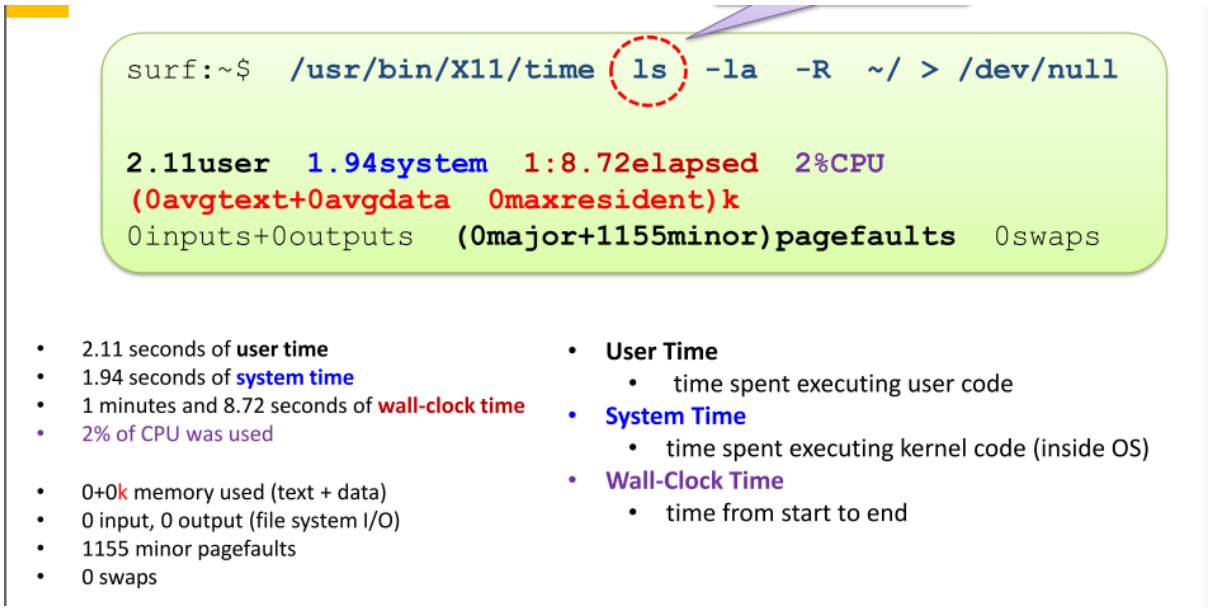
4.1. Attention

Don't measure by hand!

4.2. Measuring a process's execution time

We use **UNIX's "time" command**

Example



Times:

User time:

time spent executing **user code**

System time:

- time spent executing **kernel code** (inside OS)
- Larger system time may be a sign of higher calls to system APIs such as fopen(), fread(), fwrite(), etc

Wall-Clock time:

time **from start to end**

Wall-Clock time

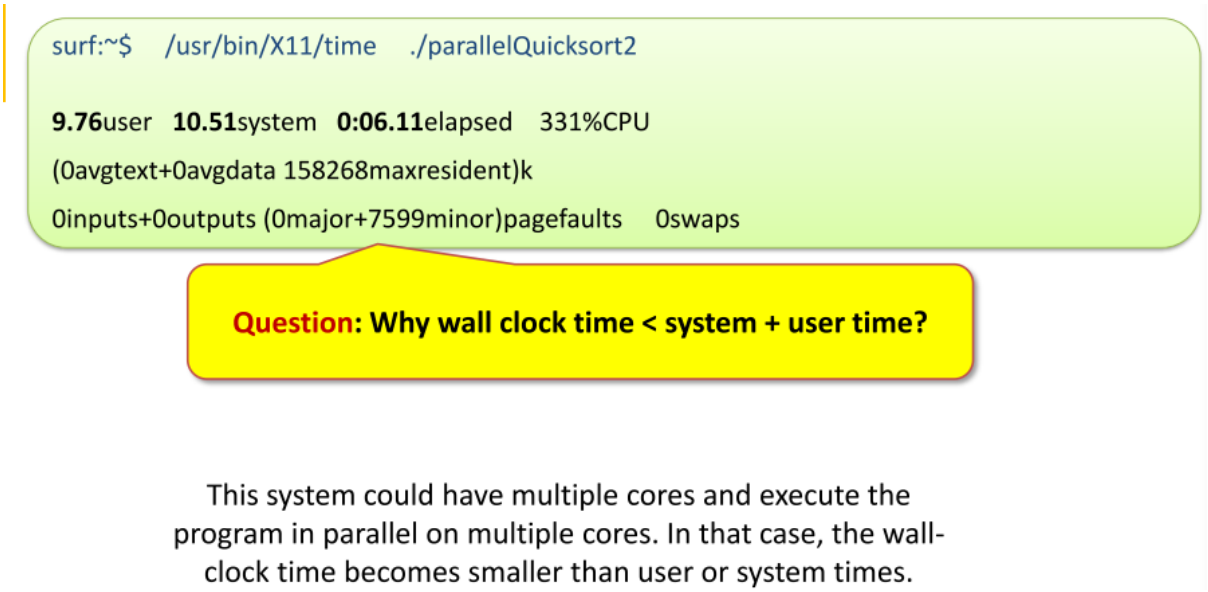
Wall clock – system – user = I/O + preemption + suspension + other interferences

- **I/O**: time waiting for performing an I/O interaction (including reading from or writing to files, ports, etc.)
- **Preemption**: time waiting for other tasks to execute

- **Suspension**: time being blocked (e.g., when a task wants to access a data that is protected by a lock) or being suspended due to an I/O operation
- **Other interferences**: time spent on accessing memory bus of a memory bank, etc.

One way to control the impact of these delays is try to run your processs in isolation (when no other process is running)

Multi-Core Situation



Drawbacks

- The time command has poor resolution
  - “Only" milliseconds
- “time” times the **whole code**
  - Often ES code is not terminating (i.e. runs forever)
  - Sometimes we're only interested in timing some part of the code, for instance the one that we are trying to optimize

4.3. Measuring time inside a program

C library: Gettimeofday & clock_gettime()

- Measures the number of microseconds since midnight, Jan 1st 1970, expressed in seconds and microseconds
- Compute the time elapsed in microseconds: (end.tv sec*1000000.0* + *end.tv usec* *start.tv sec*1000000.0 - start.tv usec) / 1000000.0)

Using timers

Assumption: There is no other process or an operating system in this example. The goal is to measure the runtime of the middle instruction that runs on an Arduino Mega.

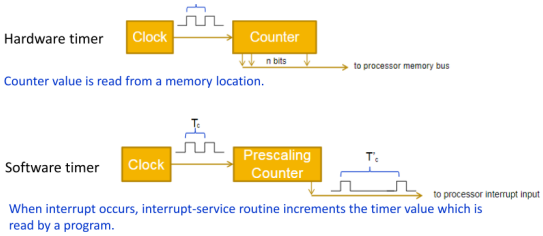
```
long x = 30, start, end;
start = micros();
x = 42 * x;
end = micros();
printf(end - start);
```

micros() returns values that are integer multiples of 4 microseconds

What can invalidate this measurement?

Each instruction is about 2~3 microsecond

- Timers are not infinitely accurate
  - All clocks have a granularity
  - The error in a time measurement, even if everything is perfect, may be the size of this granularity (sometimes called a clock tick)
- Always know your clock granularity
- Ensure that your measurement is for a long enough duration (say 100x the "tick")



	T_c'	32-bits	64-bits
• high resolution • early rollover	10 ns	42 s	5058 years
	1 μs	1.2 hour	0.5 million years
• low resolution • late rollover	1 ms	49 days	0.5×10^9

Time Rollover

- Occurs when an n-bit counter undergoes a transition from its maximum value 2^n-1 to zero.
- here is a **trade-off** between rollover time and granularity of the timer

4.4. Measuring time from outside of the system

Monitor

A Monitor is a tool to observe the activities on a system. In general, it

1. makes **measurements** on the system (Observation)

2. collects performance **statistics** (Collection),

3. **analyze**s the data (Analysis),

4. **displays** results (Presentation).

5. The Art of Data Gathering

5.1. Repeat Measurements

Errors

Error is the difference between **the measured value** and **the 'true value' of the thing being measured**

- Error can be a caused **by a low timer resolution**

Uncertainties

Uncertainty is a **quantification** of the doubt about the measurement result

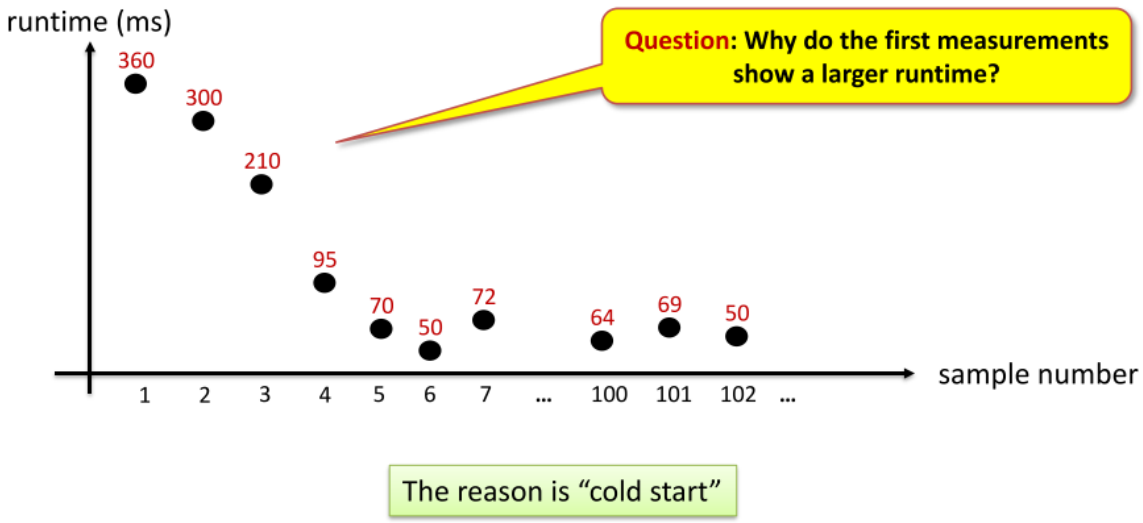
- Network delay and signal wifi strength can be two sources of uncertainty

5.2. Cold start

- Code may still be on disk, and not even loaded into memory.
- Data may be in slow memory rather than fast, e.g., cache

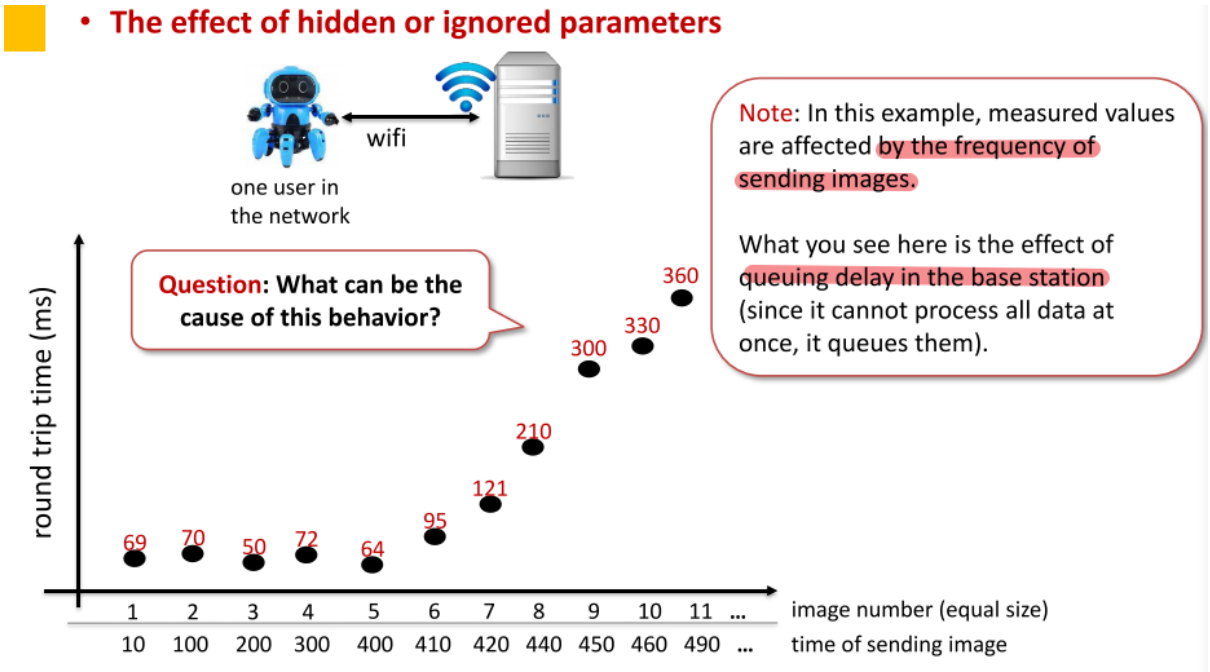
So, we should let the program run for a while and then start measuring

- usually you need to put effort to ensure that the data in the intended part of the memory hierarchy.



5.3. Other Considerations of Gathering Data

The effect of hidden or ignored parameters



Measurement-induced perturbations

The system resources consumed by **the measurement tool itself** as it collects data may strongly affect the system’s performance.

On embedded systems, writing outputs on the terminal will heavily impact the system's behavior

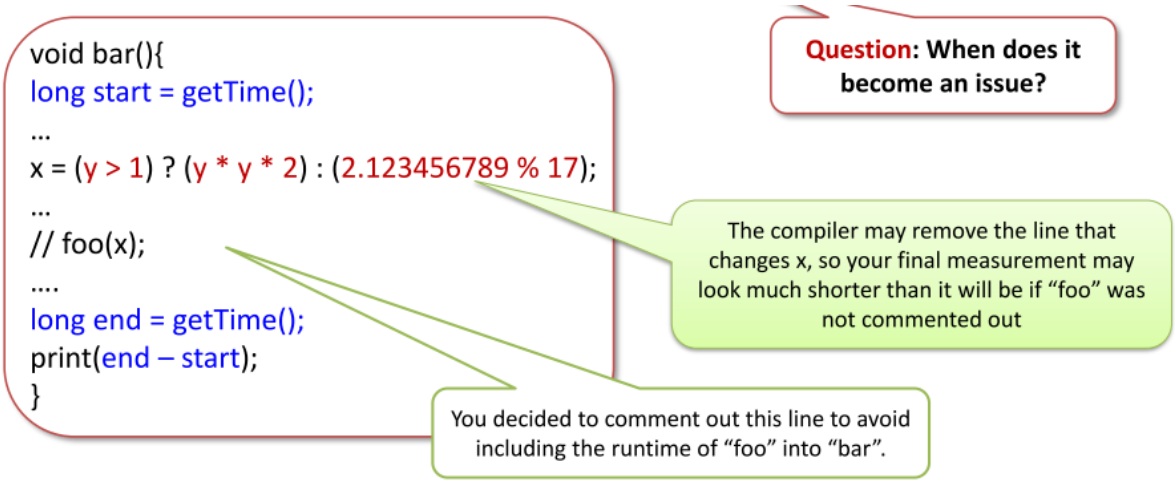
In Arduino Mega, storing time in an array takes about 2~3 microseconds, while writing it using printf() is about 70~80 microseconds

- Do not write the measured data on the output while gathering samples
  - **Solution:** store the data in an array and then output the array after gathering samples
- Do not use dynamic memory allocation for storing samples
  - It **messes up the memory alignment** of the program and changes its behavior
  - You may even run into **"out of memory"** problem due to fragmentation of memory
  - **Solution**: Use an array with predefined size (static memory allocation)

Smart Compiler

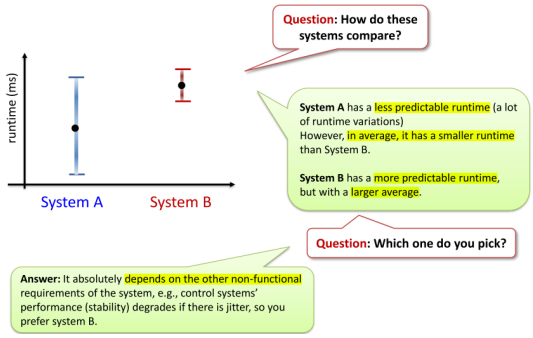
If the result of the computation is not used, the compiler may eliminate the code.

- So sometimes, your measurement will be less than the actual time the program need

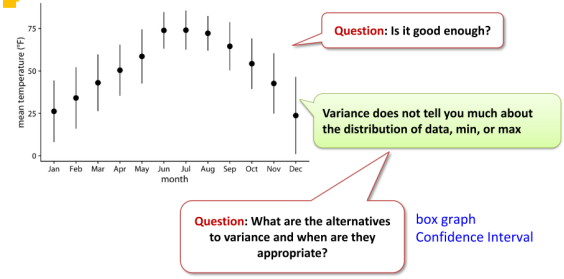


6. The Art of Data Summarization

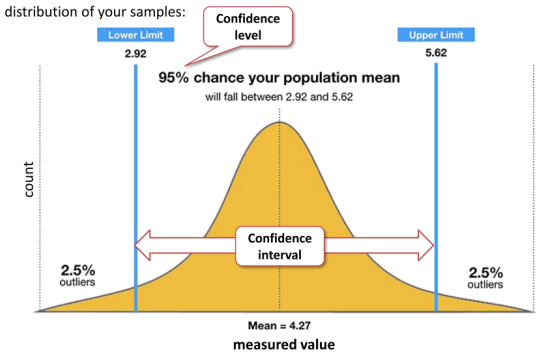
Interpreting results



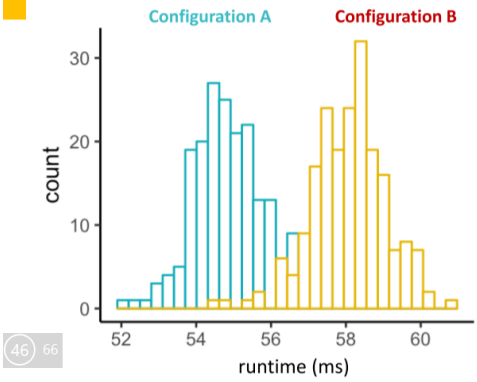
- You can show both mean and variance

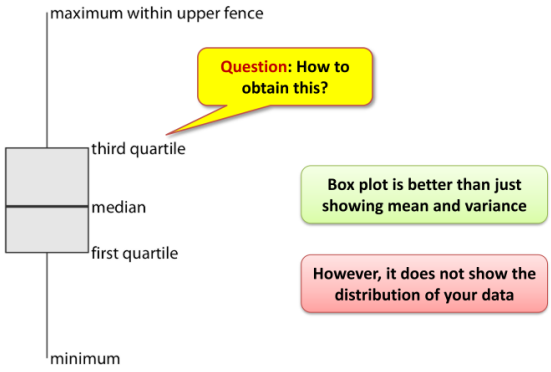


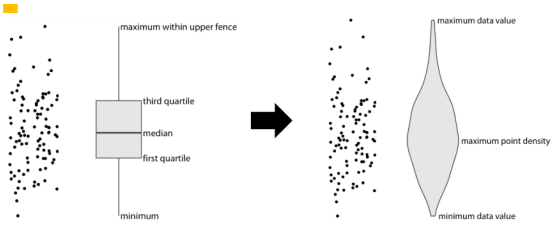
Confidence interval



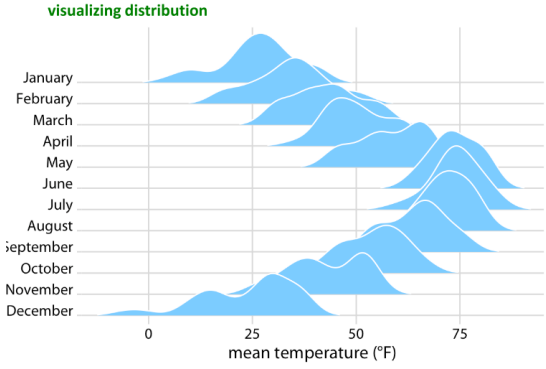
Histogram

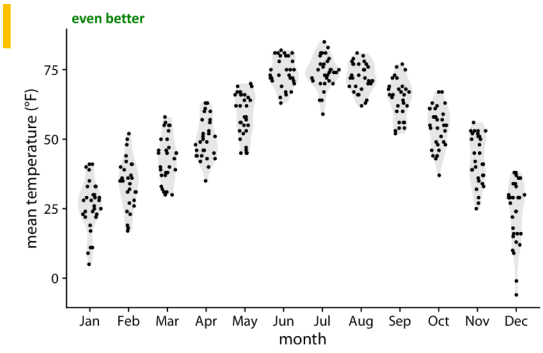




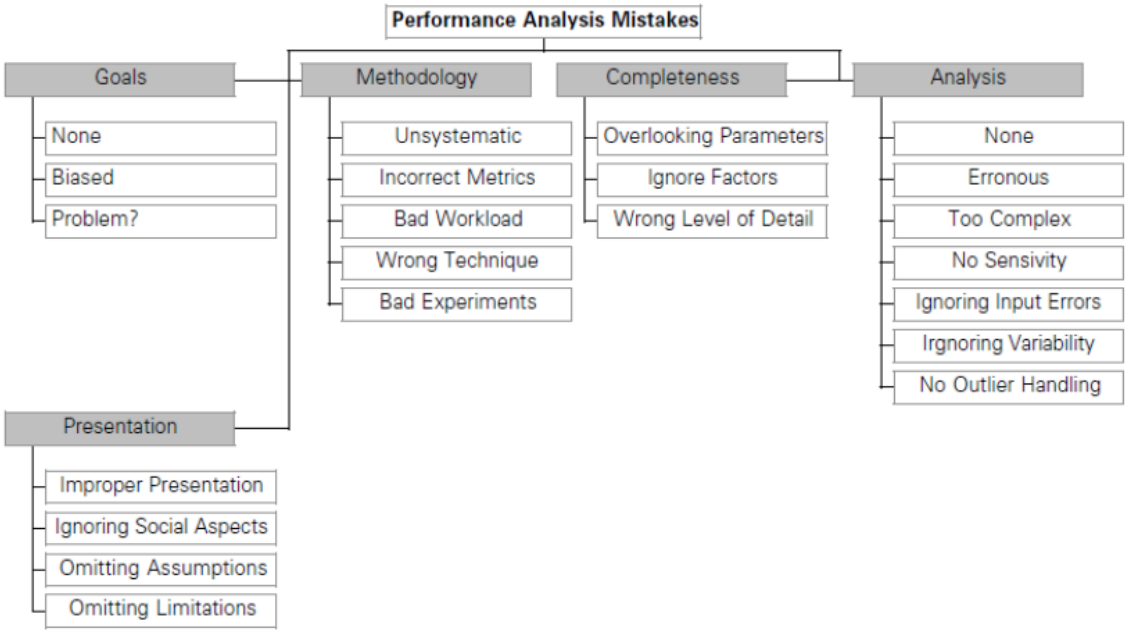


visualizing distribution





7. Common mistakes when design measurements



Biased Goals

Example:

- To show that OUR system is better than THEIRS
- Mistake: Finding the metrics and workload such that OUR system turns out better rather than finding the right metrics and workloads

Unsystematic Approach

- Selecting system parameters , factors, metrics, and workload arbitrarily

Incorrect Performance Metrics

Example: Comparing Two CPUs based on the throughput (MIPS)

CISC

RISC

Unrepresentative Workload

- Example: if packet in the network are generally **mixture of long and short**, workload should consist of **short and long packet sizes**, not just long ones.

Not performing a sensitivity analysis on the factor

- Sensitivity analysis answers the question, **"if the factors deviate from the expectations, what will the effect be on the system being analyzed"**.
- Putting too much emphasis on the results of the analysis, **presenting it as fact rather than evidence.**

Ignoring variability of results when reporting the outcome

- Only reporting mean value

Omitting Assumptions and limitations

- Assumption and limitations of the analysis are often omitted in the final report.
- This may lead the user to apply the analysis to another context where assumption will not be valid anymore.