PERFORMANCE-ORIENTED COMPUTING

Experimentation

GOALS

- In order to optimize performance, we first need to be able to **accurately measure and characterize** it
- Define the **metrics** we are interested in
- Come up with an **experimental setup**
  - Focus on **reproducibility**
- Monitor our metrics **over time**
  → performance integration testing

METRICS

PERFORMANCE METRICS

- 3 categories of metrics:

	General Metrics		
Characteristics	• Applicable to a variety of programs • Generally relatively easy to measure in a platform-independent way		
Examples	• Execution time • Peak memory usage • CPU utilization		

PERFORMANCE METRICS

- 3 categories of metrics:

	General Metrics	Domain-specific Metrics	
Characteristics	• Applicable to a variety of programs • Generally relatively easy to measure in a platform-independent way	• Represent quantities of the application domain • Not portable between domains	
Examples	• Execution time • Peak memory usage • CPU utilization	• Maximum concurrent requests serviced • Simulation time steps computed per second	

PERFORMANCE METRICS

- 3 categories of metrics:

	General Metrics	Domain-specific Metrics	Low-level Metrics
Characteristics	• Applicable to a variety of programs • Generally relatively easy to measure in a platform-independent way	• Represent quantities of the application domain • Not portable between domains	• Measure aspects of how the underlying hardware is utilized • May not be portable across hardware architectures
Examples	• Execution time • Peak memory usage • CPU utilization	• Maximum concurrent requests serviced • Simulation time steps computed per second	• L2 cache misses • Branch mispredictions • Vector operations retired

THROUGHPUT VS. LATENCY

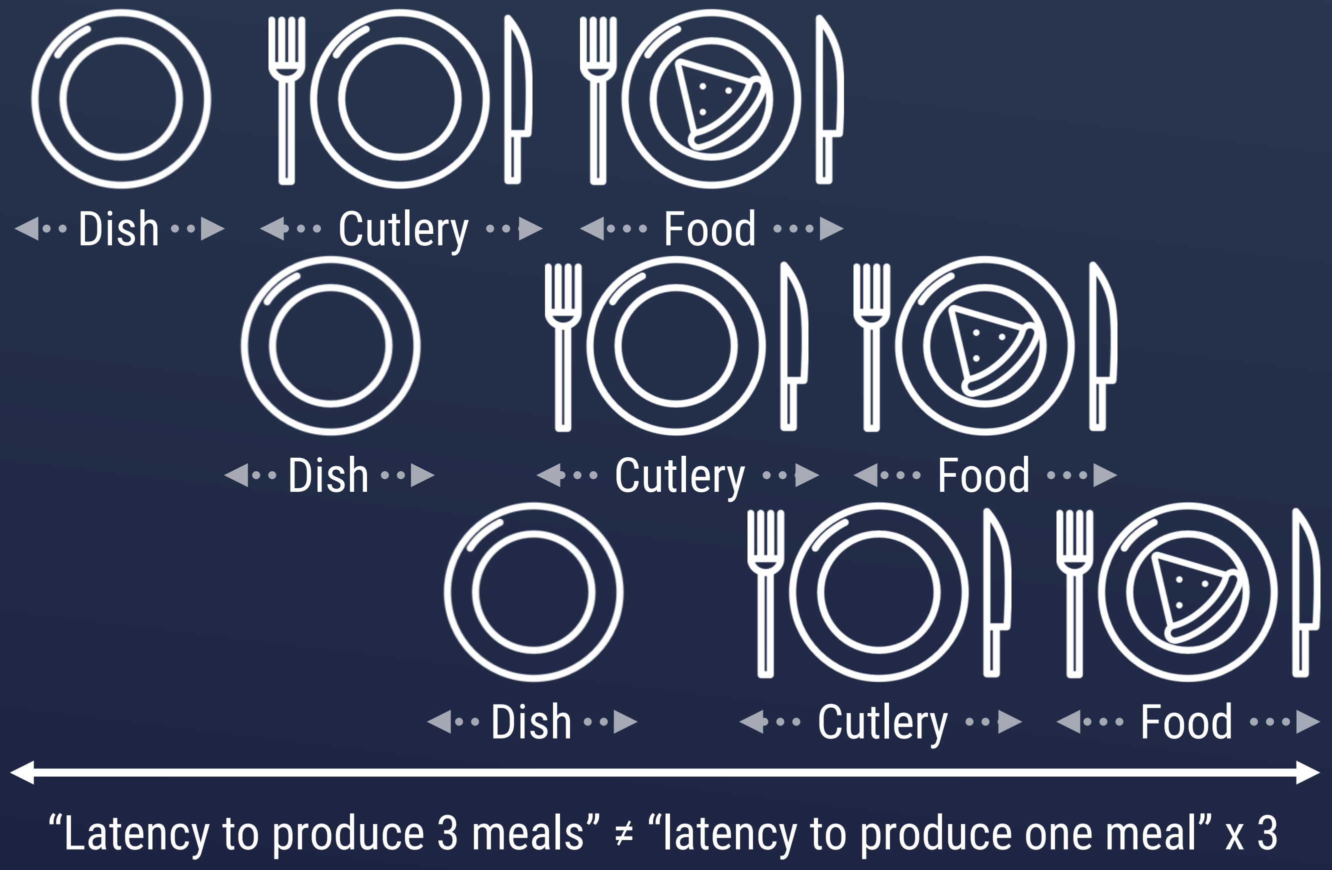
- Another important axis to characterize metrics on
- ***Latency-focused*** metrics are concerned with the time from the start to end of a single operation
- ***Throughput-focused*** metrics are concerned with the total number of operations which can be completed in a given timeframe

THROUGHPUT VS. LATENCY

- In a very simplistic model – with no asynchronicity, parallelism or pipelining – throughput and latency metrics can be interchangeable

THROUGHPUT VS. LATENCY

- In most realistic models of computation, every larger result is computed out of parts which are parallelized or pipelined at some level



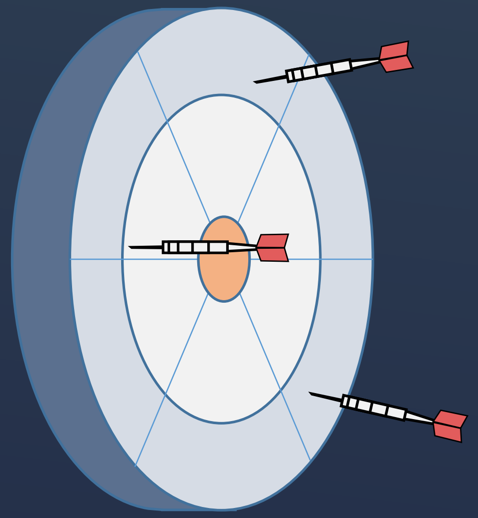
MEASUREMENTS

MEASUREMENT CHALLENGES

1. **Consistency** / Reproducibility
   - More challenging with smaller quantities (i.e. time scale)
   - Essential to overcome!

2. Access / Portability
   - Can be difficult particularly for low-level-metrics

POTENTIAL MEASUREMENT ISSUES



- **Imprecise** measurements
- Impact of **external load**
- Measuring in **non-representative scenarios**
- Properties to be measured **vary** with **input data**

→ Need to distinguish between *random* and *systematic* errors

Can be mitigated by statistical evaluation of many runs

Can **not** be mitigated by repetitions

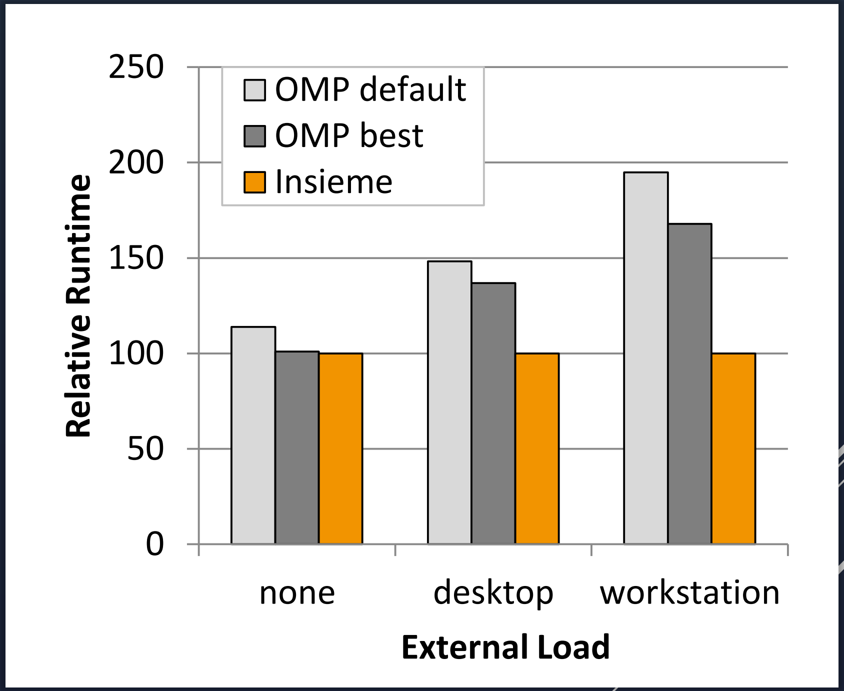
MEASUREMENT PRECISION

- The precision which can be achieved depends on multiple factors:
  - Type of quantity being measured
    (i.e. CPU time can be measured more precisely than power consumption)
  - Overall extent (e.g. in time) of the event to be measured
  - HW/SW employed for the measurement

- How to improve precision?
  - Use best available method (i.e. high performance timers, CPU counters)
  - Make the events to be measured as large as possible
    - One option: repetition → but be careful, might be non-representative!

IMPACT OF EXTERNAL LOAD

- ***External Load***: system resources are used by processes other than the program we intend to measure
- Difficult to completely avoid on consumer hardware with standard OS configurations
- Impact can be reduced by
  - Benchmarking on **dedicated** systems
  - **Core pinning** / affinity masking



NON-REPRESENTATIVE SCENARIOS

- Problems occur when different scenarios are used for experiments and actual execution – examples:
  - Measuring **debug builds**
  - Measuring **different HW** architectures
  - Measuring of **individual components** (where behaviour changes when integrated)
  - The **act of measuring** itself might modify the behaviour

- Try to **reproduce production scenario** as precisely as possible
  - If required, statistically capture different HW
  - Integrate benchmarking features in release builds

VARIANCE WITH INPUT DATA

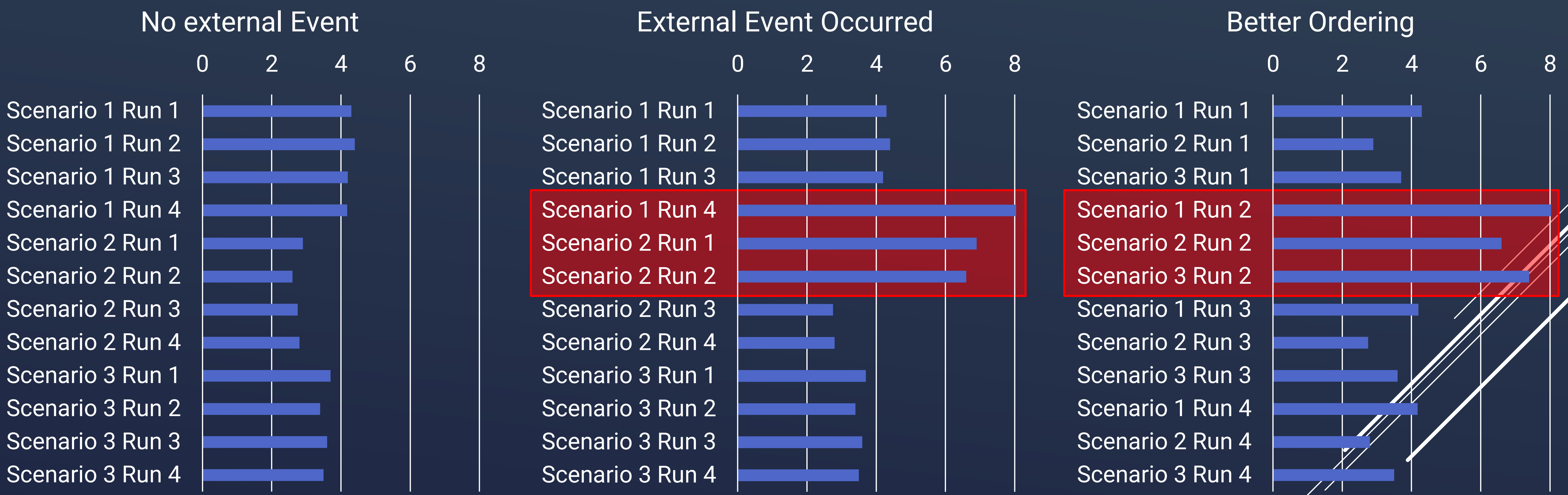
- Depending on the algorithm, **performance can vary with input data**
- Might lead to wrong optimization decisions, when they are based on a single data set

- If algorithm performance varies with input data, need **several data sets** for **statistical evaluation**
  - *Challenge*:
    Represent all relevant characteristics with as few runs as possible

STATISTICAL EVALUATION

- Basic idea: repeat runs, aggregate the results – *different goals*:
  - Reproduce **different scenarios** (i.e. HW/SW stack)
  - Use different **input data**
  - Reduce **non-systemic** errors
- Important factors to consider:
  - **Order** of executions
  - Statistical methods / **aggregation** functions applied

ORDER OF EXECUTION



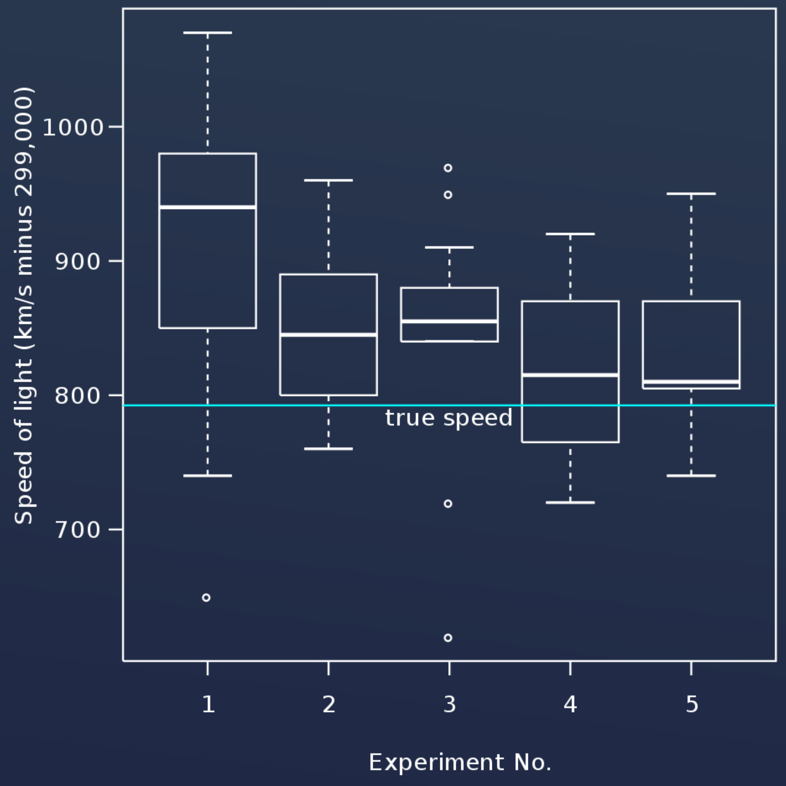
→ Repetitions with the purpose of reducing random errors should be **spaced out as far as possible**

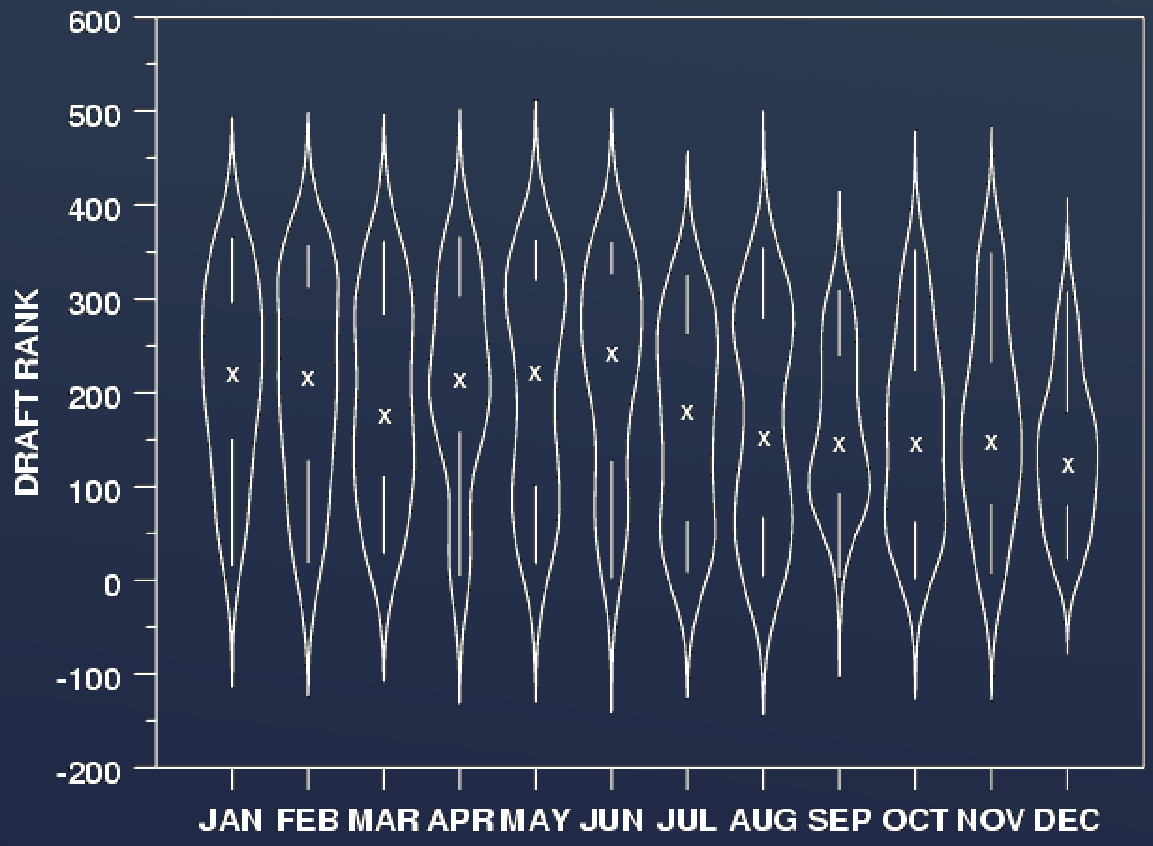
AGGREGATION FUNCTIONS

Best choice depends on *purpose* and *context* of aggregation – examples:

- Want to measure **maximum throughput** (e.g. of HW or algorithms)
  - Use **minimum** of times (effectively minimizes impact of external load)
- Represent realistic "average" performance
  - Use **median** if a single number is required; Ideal: analyse **distribution**
- Examining response times in a soft-realtime context
  - **Maximum** of 99th percentile

DATA REPRESENTATION





Representations such as box and violin plots allow for a better understanding of comparative performance at a glance.

Public domain plots sources: itl.nist.gov, wikipedia

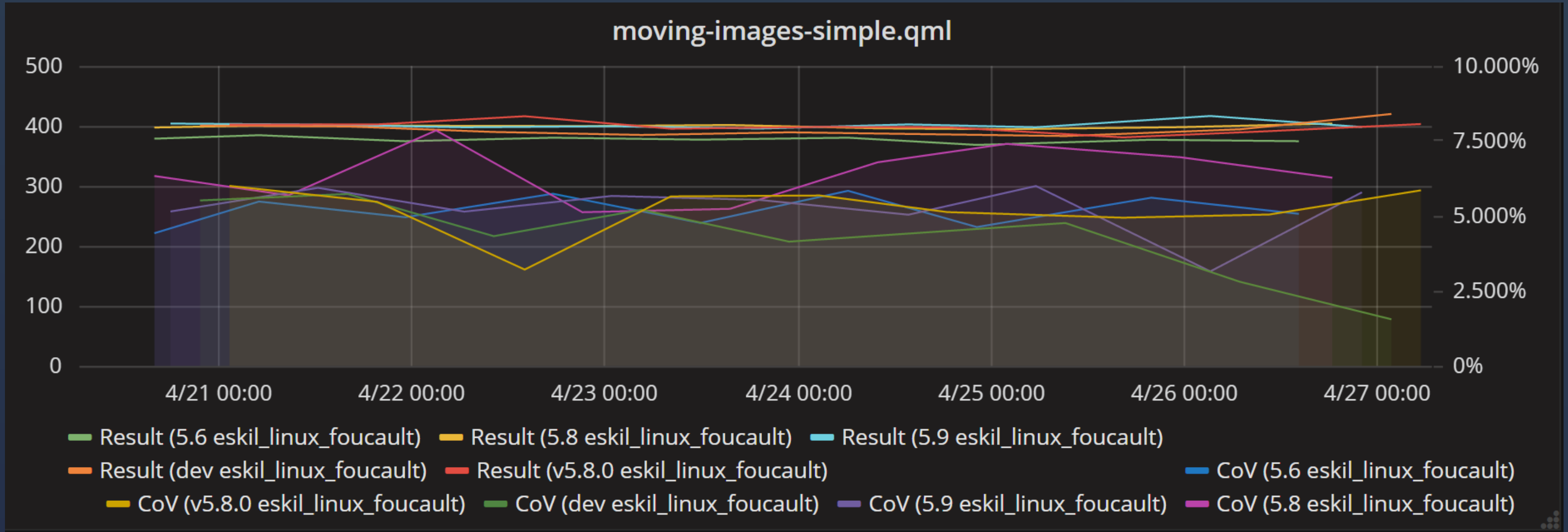
PERFORMANCE INTEGRATION TESTING

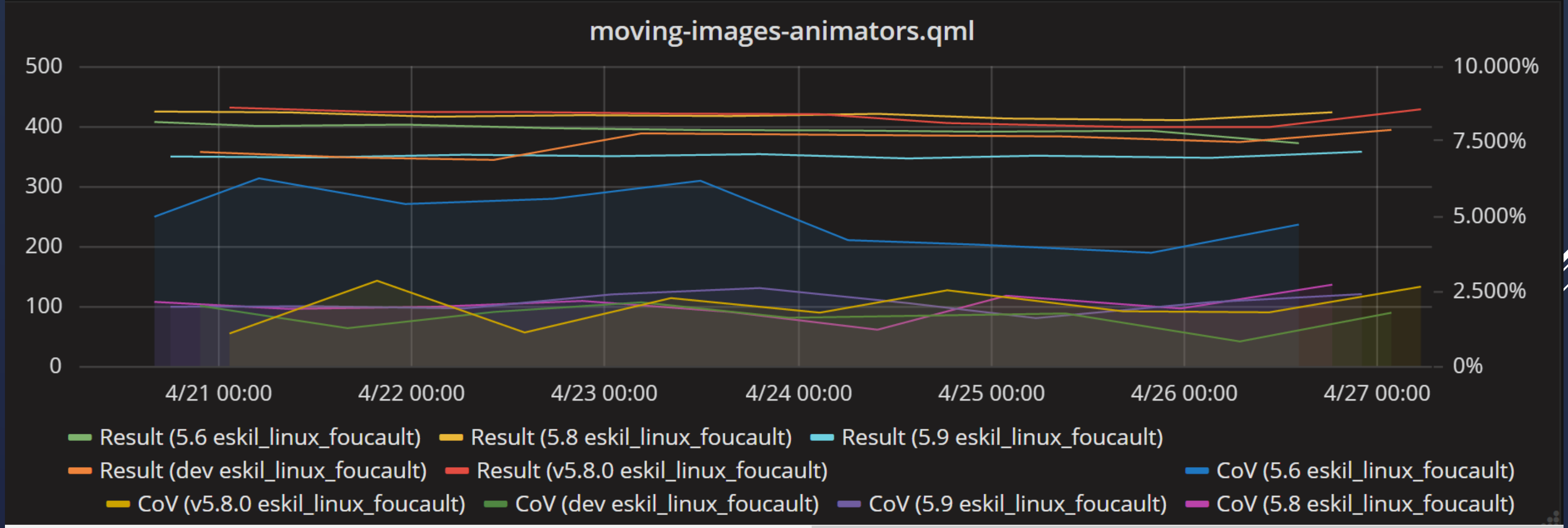
OVERVIEW

- Some different names in use for the same fundamental goal:
  - “Performance integration testing”
  - “Continuous performance testing“
  - “Performance regression testing”

- *Idea:* monitor performance **continuously** during development
  - Particularly, take not of **unintentional/unknown performance regressions**
    - e.g. ”feature X was introduced, suddenly unrelated feature Y is 5 times slower”

EXAMPLE





Source: *"Performance regression testing of Qt Quick"* – Qt

IMPORTANT FACTORS

- Clearly define **target metrics** and experiments to measure them
  - Follow all good practices for experiments we just discussed
  - Track “benchmark coverage” similarly to test coverage
- Implement **as early as possible**
  - Better overview of performance over the full duration of development
  - Design issues are generally easier and cheaper to fix early!
- Generate **reports / summaries / charts**
  - Having the data is a good first step, but if no one looks at it since it’s just a bunch of CSV files that doesn’t help much

PERF INTEGRATION TESTING CHALLENGES

- More **complex** and **costly** to set up than normal testing
  - Needs **dedicated hardware** to ensure reproducibility
  - Potentially needs **specific features** (i.e. core count, SIMD, GPU, …)
  - Might require **multiple instances** with differen HW/SW setup to cover target configurations

- Should be **fully integrated** with continuous build/test/delivery
  - In order to investigate the performance impact of changes later
  - **Long-term storage** and efficient browsing/comparison of results also needs to be set up

CONCLUSION

SUMMARY

- Understand different types of **metrics**
  - General / Domain-specific / Low-level
  - Throughput or latency-focused
- Know how to perform **accurate and repeatable measurements**
  - Systemic vs. random errors
  - Achieving precision and reproducing production scenarios
  - Experiment design
- Be aware of the potential of **performance integration testing**
  - And its challenges/requirements

QUESTIONS ?