Introduction to Performance Analysis

Yury Oleynik
oleynik@in.tum.de

Outline

- **Motivation for performance tuning**
- Performance optimization cycle
- Basics of performance analysis
- Performance analysis techniques

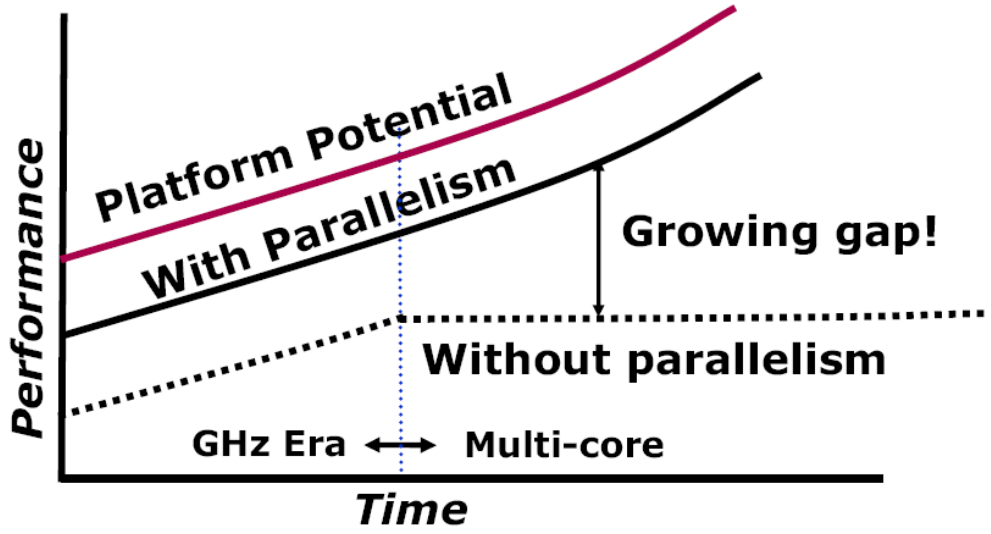
Motivation

- Goal of performance optimization
  - Reduce resource consumption to the acceptable limit

  or
  - Produce more results under a given consumption limit
- Resources:
  - Execution time
  - Memory consumption
  - Power consumption
  - …
- Derive non-functional requirements

Motivation – Parallelism

Need for Parallelism



Parallelism is crucial for optimal performance

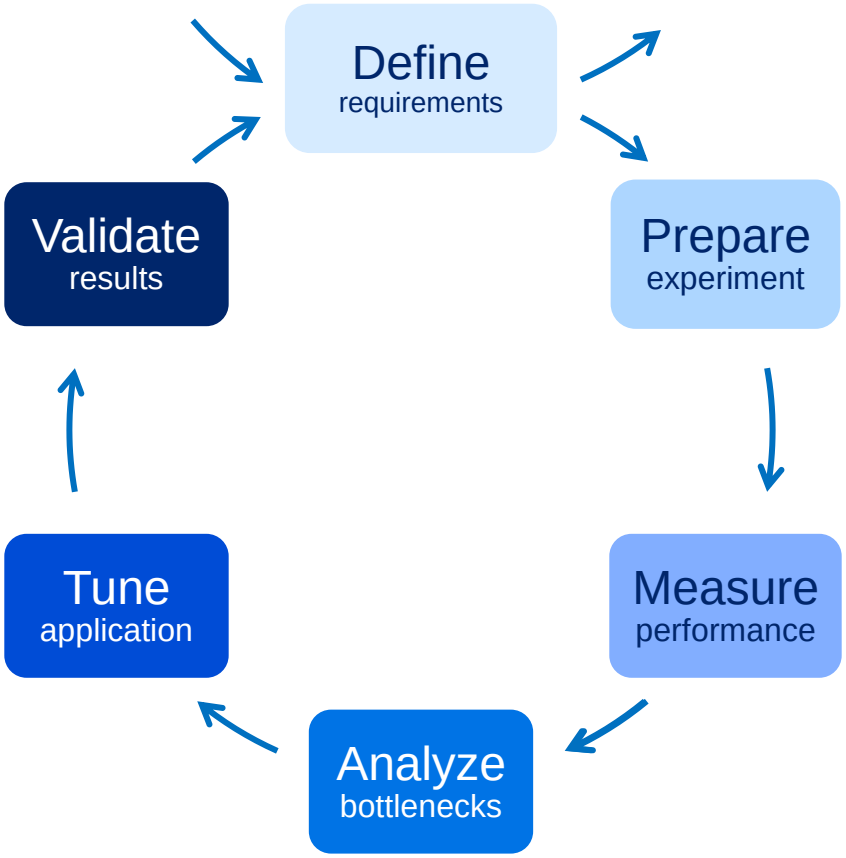
Performance optimization challenges

- High complexity in parallel and distributed systems
  - Application
    - Algorithm, data structures
  - Parallel programming interface
    - Compiler, parallel libraries, communication, synchronization
  - Operating system
    - Process and memory management, IO
  - Hardware
    - CPU, memory, network

Outline

- Motivation for performance tuning
- **Performance optimization cycle**
- Basics of performance analysis
- Performance analysis techniques

Performance optimization cycle



Defining requirements

Define
requirements

- Identify performance optimization objective
  - Execution time
  - Memory footprint
  - …
- Identify the target execution context
  - Data set
  - Environment configuration
  - Algorithms
- Define the acceptance criteria
  - Execution time below 4 hours
- Refine requirements if needed
  - Consider abortion of the tuning to avoid wasting resources

Preparing application

- Prepare the test configuration reflecting the target execution context

Prepare
experiment

- Design performance experiment
  - Execution aspect to be analyzed
  - Performance analysis tool to be used
  - …
- Instrument application
  - Insertion of the probe functions
  - Consider granularity vs overhead
  - Manual or automatic

Measuring performance

- Prepare execution environment
  - Batch scripts
  - Environment settings
- Start instrumented application using the selected performance analysis tool
- Raw performance data is produced

Measure
performance

Analyzing performance

Analyze
bottlenecks

- Process raw performance data
  - Visualize
  - Compute derived metrics
- Identify application hotspots
  - Against the target optimization criteria
- Relate hotspots back to the source code
- Compute severity of the hotspots
- Rank the hotspots, identify bottleneck
- Select hotspots for optimization

Tuning performance

- Identify possible bottleneck optimization recipes
  - Compiler optimization
  - Loop transformation
  - Hardware specific pitfalls
  - Choose better algorithms
  - …

Tune
application

- Is done manually by application developer…

Validating achieved performance

- Check the acceptance criteria to be satisfied
- Check the correctness of the produced results
- Record the achieved performance and the applied tuning actions

Outline

- Motivation for performance tuning
- Performance optimization cycle
- **Basics of performance analysis**
- Performance analysis techniques

Performance analysis, prediction and benchmarking

- **Performance analysis** determines the performance on a given machine.
- **Performance prediction** allows to evaluate programs for a hypthetical machine. It is based on:
  - runtime data of an actual execution
  - machine model of the target machine
  - analytical techniques
  - simulation techniques
- **Benchmarking** determines the performance of a computer system on the basis of a set of typical applications.

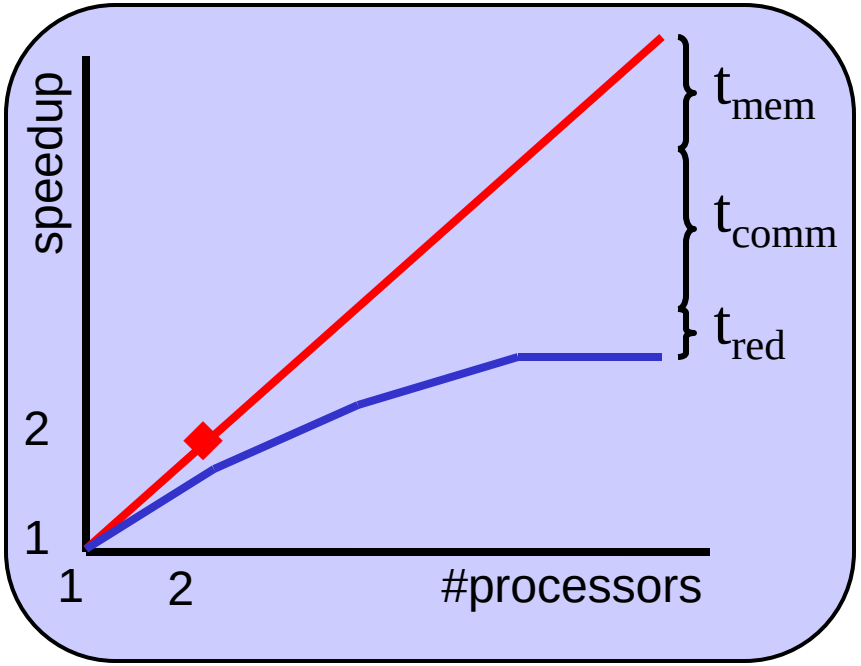
Sequential vs parallel performance

- Factors which influence performance of parallel programs
  - “Sequential” factors
    - Computation
    - Cache and memory
    - Input / output
  - “Parallel” factors
    - Communication (Message passing)
    - Threading
    - Synchronization

- How to decide whether a code performs well:
  - Comparison of measured MFLOPS with peak performance
  - Comparison with a sequential version

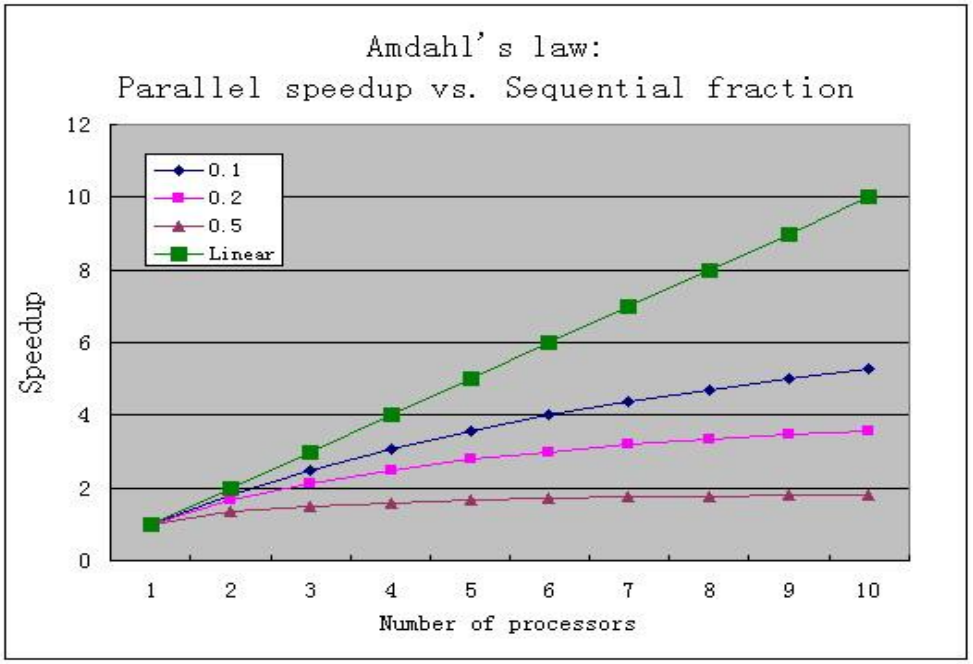
$$speedup(p) = \frac{t_s}{t_p}$$

  - Estimate distance to ideal time via overhead classes
    - t_{mem}
    - t_{comm}
    - t_{sync}
    - t_{red}
    - ...



Amdahl's Law

- The speedup of a program using multiple processors in parallel computing is limited by the time needed for the sequential fraction of the program.



Source: Wikipedia

- Scaled speedup
  - Problem size grows with machine size

$$\text{scaled\_speedup}(p) = \frac{t_s(n_p)}{t_p(n_p)}$$

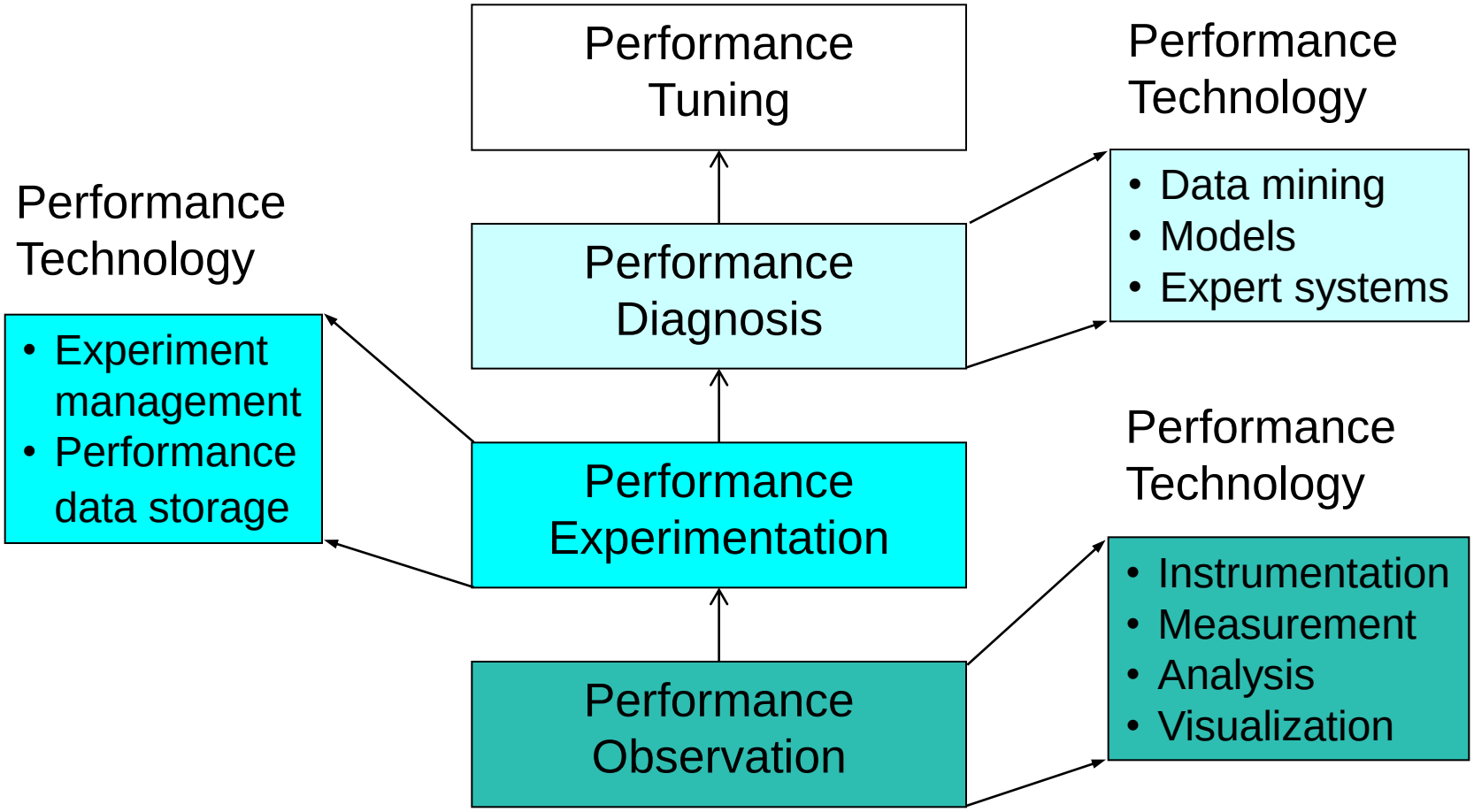
- Parallel efficiency: Percentage of ideal speedup

$$\text{efficiency}(p) = \text{speedup}(p)/p = \frac{t_s}{t_p * p}$$

Outline

- Motivation for performance tuning
- Performance optimization cycle
- Basics of performance analysis
- **Performance analysis techniques**

- What the tools can offer?



Performance Observation

- Understanding performance requires observation of performance properties.
- Performance tools and methodologies are primarily distinguished by what observations are made and how.
  - How application program is instrumented
  - What performance data are obtained
- Tools and methods cover broad range.

- Execution *actions* exposed as *events*
  - In general, actions reflect some execution state
    - presence at a code location or change in data
    - occurrence in parallelism context (thread of execution)
  - Events encode actions for observation
- Observation is *direct*
  - Direct instrumentation of program code (probes)
  - Instrumentation invokes performance monitoring
  - Measurement = event + performance data + context

Direct Observation: Instrumentation

- Static instrumentation
  - Program instrumented prior to execution
- Dynamic instrumentation
  - Program instrumented at runtime
- Manual and automatic mechanisms
- Tools required for automatic support
  - Source time: preprocessor, translator, compiler
  - Link time: wrapper library
  - Execution time: binary rewrite, dynamic
- Advantages / disadvantages

Indirect Performance Observation

- Events are actions external to program code
- Program code instrumentation is not used
- Performance is observed indirectly
  - Execution is interrupted
    - can be triggered by different events
  - Execution state is queried (sampled)
    - different performance data measured
- Performance attribution is inferred
  - Determined by execution context (state)
  - Observation resolution determined by interrupt period
  - Performance data associated with context for period

- Direct performance observation
  - ☺ Measures performance data exactly
  - ☺ Links performance data with application events
  - 😐 Requires instrumentation of code
  - ☹ Measurement overhead can cause execution intrusion and possibly performance perturbation
- Indirect performance observation
  - ☺ Argued to have less overhead and intrusion
  - ☺ Can observe finer granularity
  - ☺ No code modification required (may need symbols)
  - ☹ Inexact measurement and attribution without hardware support

- Accuracy
  - Timing and counting accuracy depends on resolution
  - Any performance measurement generates overhead
    - Execution on performance measurement code
  - Measurement overhead can lead to intrusion
  - Intrusion can cause perturbation
    - alters program behavior
- Granularity
  - How many measurements are made
  - How much overhead per measurement
- Tradeoff (general wisdom)
  - Accuracy is inversely correlated with granularity

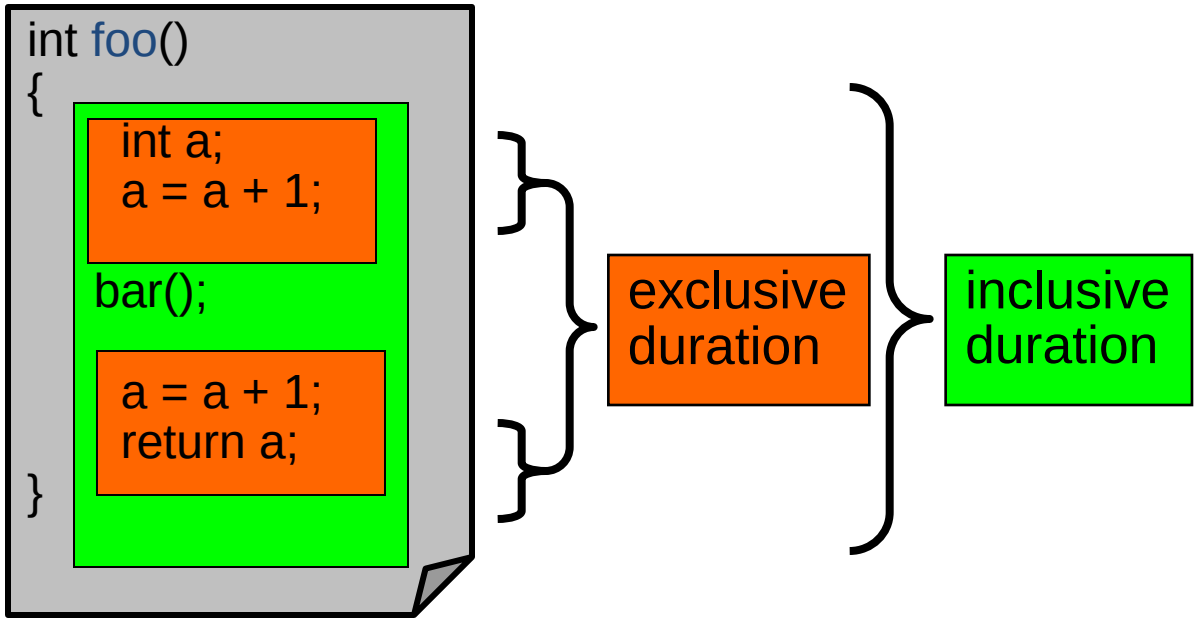
- How are measurements made?
  - Profiling
    - summarizes performance data during execution
    - per process / thread and organized with respect to context
  - Tracing
    - trace record with performance data and timestamp
    - per process / thread

- Recording of aggregated information
  - Counts, time, …
- … about program and system entities
  - Functions, loops, basic blocks, …
  - Processes, threads
- Methods
  - Event-based sampling (indirect, statistical)
  - Direct measurement (deterministic)

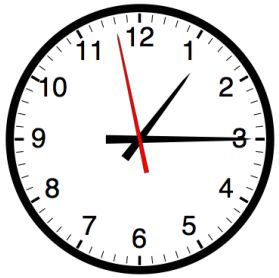
- Flat profile
  - Performance metrics for when event is active
  - Exclusive and inclusive
- Callpath profile
  - Performance metrics for calling path (event chain)
  - Differentiate performance with respect to program execution state
  - Exclusive and inclusive

Inclusive and Exclusive Profiles

- Performance with respect to code regions
- Exclusive measurements for region only
- Inclusive measurements includes child regions



Trace File Generation



1	master
2	worker
3	...

Process A:

```
void master {
  trace(ENTER, 1);
  ...
  trace(SEND, B);
  send(B, tag, buf);
  ...
  trace(EXIT, 1);
}
```

Process B:

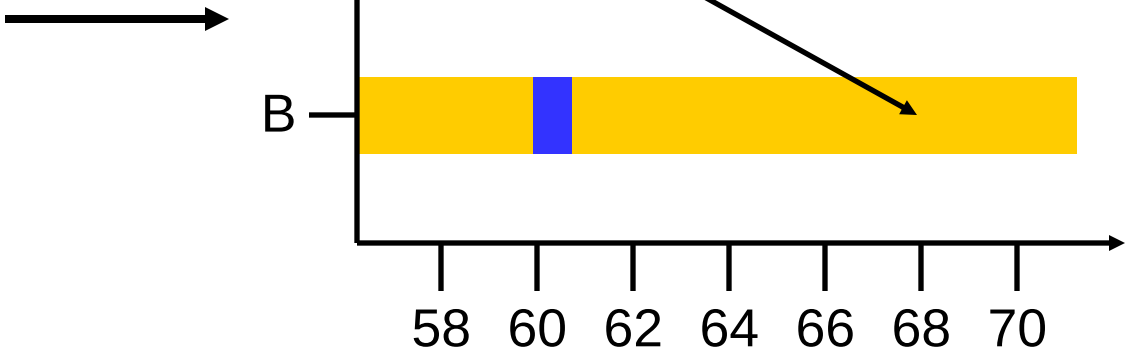
```
void worker {
  trace(ENTER, 2);
  ...
  recv(A, tag, buf);
  trace(RECV, A);
  ...
  trace(EXIT, 2);
}
```

MONITOR

...			
58	A	ENTER	1
60	B	ENTER	2
62	A	SEND	B
64	A	EXIT	1
68	B	RECV	A
69	B	EXIT	2
...			

1	master
2	worker
3	...

...			
58	A	ENTER	1
60	B	ENTER	2
62	A	SEND	B
64	A	EXIT	1
68	B	RECV	A
69	B	EXIT	2
...			



- Profiling
  - ☺ Finite, bounded performance data size
  - ☺ Applicable to both direct and indirect methods
  - 😐 Loses time dimension (not entirely)
  - ☹ Lacks ability to fully describe process interaction
- Tracing
  - ☺ Temporal and spatial dimension to performance data
  - ☺ Capture parallel dynamics and process interaction
  - 😐 Some inconsistencies with indirect methods
  - ☹ Unbounded performance data size (large)
  - ☹ Complex event buffering and clock synchronization

Questions?