JVM Performance and Tuning

Module Two

Performance Metrics

Module Topics

1. Defining Performance
2. Performance Metrics
3. Performance Measurements
4. Benchmarking and Profiling

Some Caveats

- There are no “go faster” options for a JVM
- There are no list of tips and tricks to optimize a JVM
- The performance of code depends on many factors including:
  - The hardware the JVM is running on
  - The version of the JVM
  - How the code was designed and written
- Java design choices
  - Program safety over performance
  - Programmer productivity over performance

Java Design Decisions

- Designed for interactive applications
  - Generally the human is the rate limiting step
  - As long as it is “fast enough” then performance can be sacrificed for productivity
    - *Main cost is software development is programmer time, both writing code and debugging and maintaining code*
  - Interactive applications tend to require more updating and maintenance as requirements evolve and change
- Relies on a number of managed subsystems
  - e.g. Memory allocation and garbage collection
  - These subsystems introduce complexity into the running of the JVM
- Even measuring performance affects performance

Performance Metrics

- Depends on what is important to you
- Business level metrics
  - Ease of use
  - Customer loyalty
  - Improves business operations
  - “Apdex” index
    - *Measure of how log a transaction should take – industry standard*
- Application level metrics
  - Throughput measures – how many simultaneous transactions
  - Reliability, durability, loading etc
  - Quality levels, risk and error levels

Performance Metrics

- Depends on what is important to you.. cont
- Execution Metrics (what we are interested in)
  - Efficient use of hardware
  - CPU usage, memory usage, I/O bandwidth
- Source Code Metrics
  - Readability – ease of modification and changes
  - Effective and efficient design
  - Algorithm choice and implementation

Performance Metrics

- Application level metrics are often design or topology related
- Code metrics are often related to code design issues
- But both application and code issues affect execution metrics
- We are concerned with how to improve execution metrics if we assume there are no issues at the application topology and source code levels

Module Topics

1. Defining Performance
2. Performance Metrics
3. Performance Measurements
4. Benchmarking and Profiling

Performance Metrics

- Standard metrics include:
  - Throughput
  - Latency
  - Capacity
  - Utilization
  - Efficiency
  - Scalability
  - Degradation

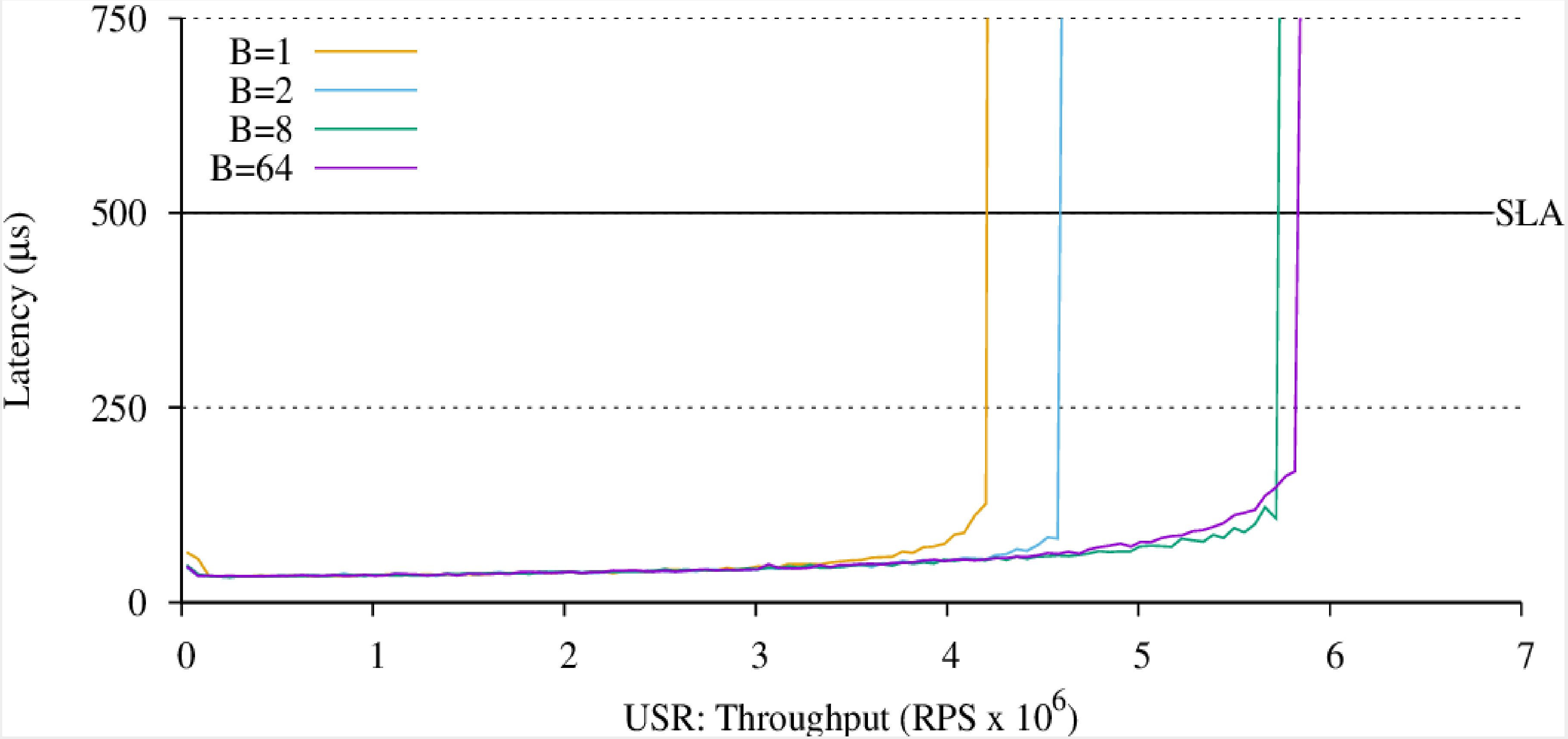
Throughput

- The rate of work a system can perform.
- Expressed as number of units of work in some time period
- Dependent on the platform where measured
- Dependent on how the unit of work is defined
- Often represented as number of complete transactions

Latency

- Time taken to process a single transaction
- Generally expressed as a start-to-end time.
- Dependent on workload
  - Expressed as a function of increasing workload
  - Increasing workload means less task access to resources
  - For example, browser latency increases the more tabs are open

Latency

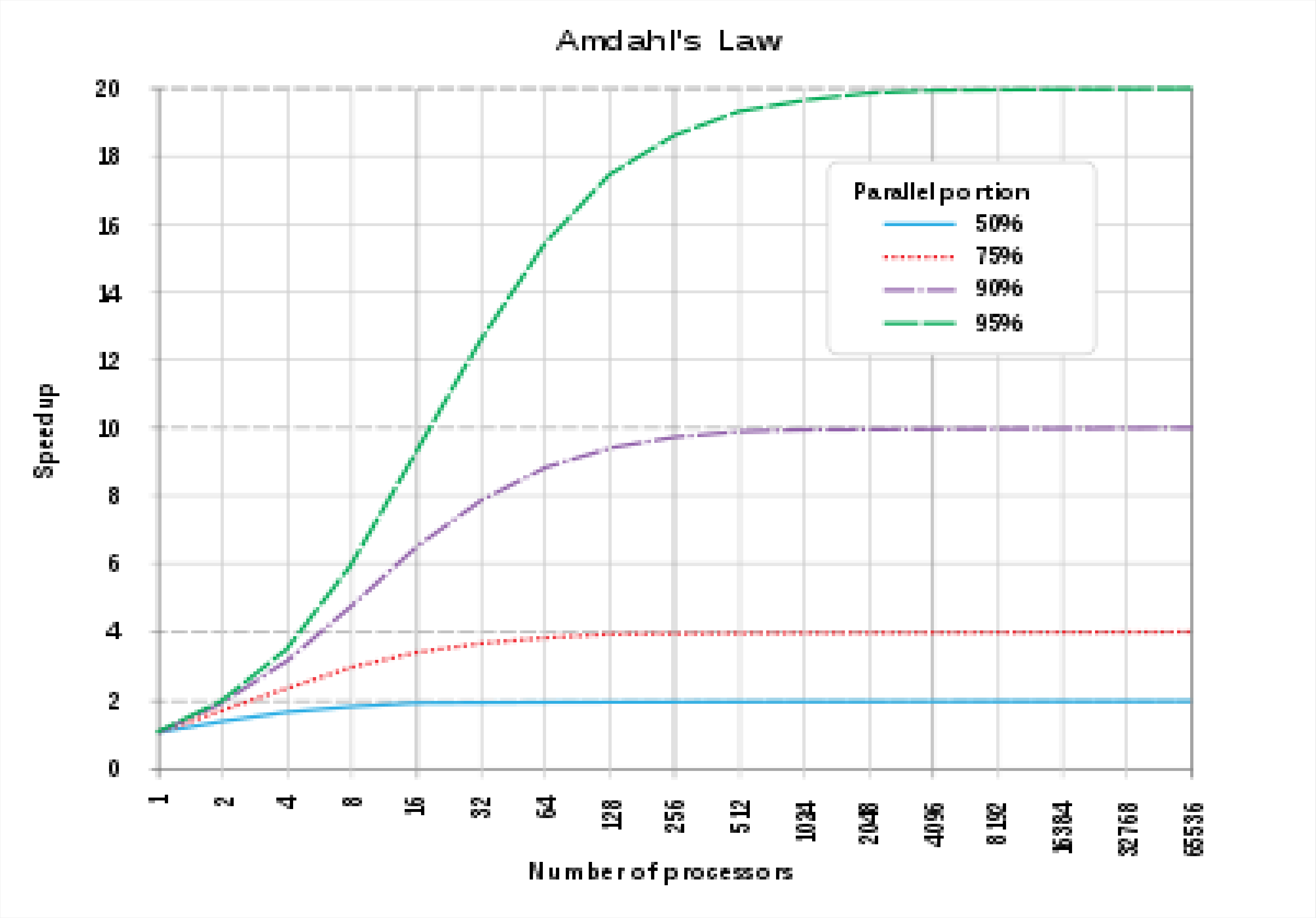


750
500
250
0
B=1
B=2
B=8
B=64
SLA
Latency (µs)
0
1
2
3
4
5
6
7
USR: Throughput (RPS x 10^6)

Capacity

- Amount of work parallelism a system possesses
- The number of units of work that can be simultaneously ongoing in the system
- Capacity is related to throughput
  - The more capacity available implies more throughout possible
- Restaurants put more staff on at meal times
  - Increases restaurant capacity
  - Improves throughput = number of meals prepared
  - Reduces latency = the amount of time a customer has to wait

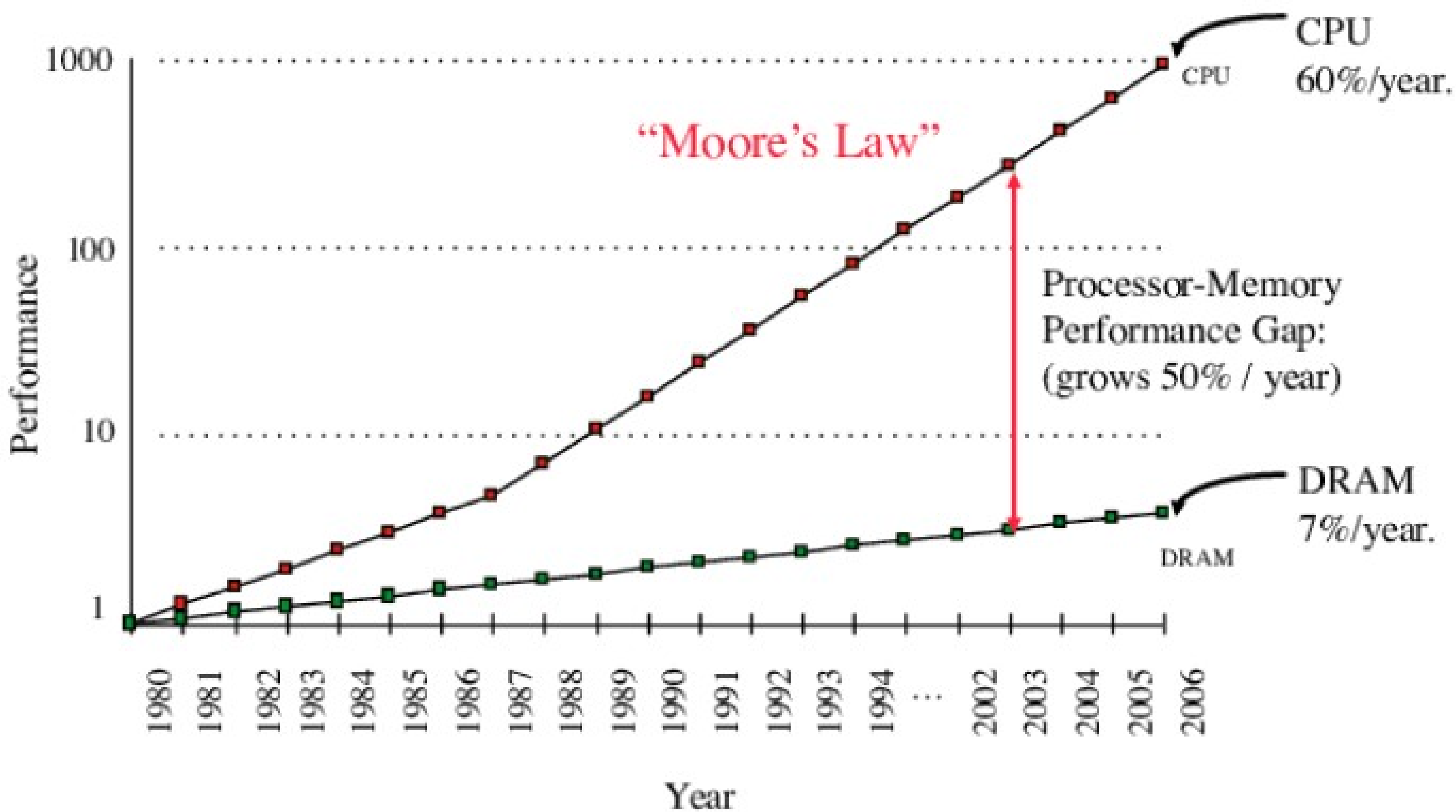
Capacity


Amdahl's Law
Parallel portion
50%
75%
90%
95%
Speedup
0
2
4
6
8
10
12
14
16
18
20
Number of processors
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536

Utilization

- Efficient use of a system's resources
- Computationally intensive tasks may have close to 100% CPU utilization but have little memory use
- A common problem today is:
  - CPU performance has increased dramatically
  - Memory performance has not
  - CPUs are idle waiting for memory to catch up moving data in and out of the CPU registers.

Utilization



Memory Access vs CPU Speed

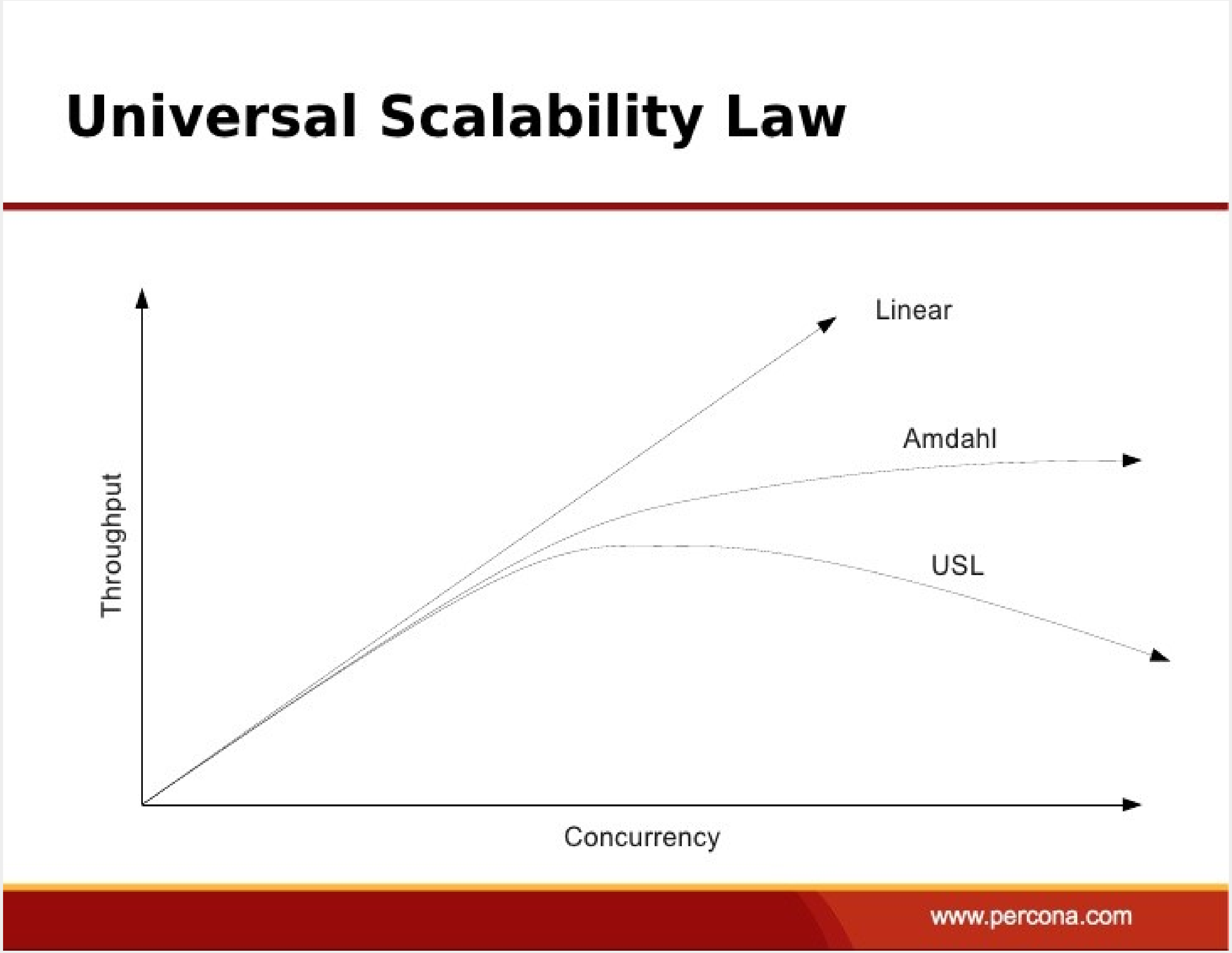
Efficiency

- Overall system efficiency is measured by
  - (throughput)/(utilized resources)
- Essentially making the most use of available resources
- Often used to compute costs of processing

Scalability

- Change in throughput as resources are added
- Ideally, throughput should increase with increase in capacity
- Scalability fails when increased capacity does not produce an increase in throughput

Scalability



Module Topics

1. Defining Performance
2. Performance Metrics
3. **Performance Measurements**
4. Benchmarking and Profiling

Performance Tests

- All tests, to be useful must be:
  - **Valid**: We have to actually test what we think we are testing
  - **Accurate**: The results are quantitatively accurate
  - **Reliable**: The results of the tests depend only on the tests and thing being tested
- These properties are true for all testing not just performance testing
  - Generally performance testing is statistical in nature
  - However it is difficult to identify all the contributing factors to the results of the tests
  - Good testing protocols need to be observed

Java Design

- Java performance philosophy
  - raw performance could be sacrificed if developer productivity benefited
  - provided performance was “good enough”
- Managed subsystems
  - low-level control exchanged for managed resources
  - eg. JVM memory management and garbage collection
  - Also provided a safer runtime “sandbox”

JVM Behaviour

- JVM runtime behaviour
  - Very complex – difficult to measure consistently
  - Observed measurements are often not normally distributed
  - Makes statistical analysis difficult – most statistical techniques assume underlying normal distributions
  - Critical measurements are easily missed via sampling
  - Outliers may be significant (“that one weird time that the app crashed”)
- Java performance measures can be misleading
  - We are trying to measure a highly complex set of subsystems
  - May even be non-linear in many aspects
    - *ie. small changes may produce large variations and vice versa*

The Experimenter Effect

- The actual measurement of performance affects performance
  - Measurement has overhead
  - Sampling measurements require resources
  - These resources are the same resources being measured
- This means that any sampling profile
  - Can only be regarded as approximate
  - And is not a true measure of performance.

A Basic Methodology

- Basic Steps
  - Define the desired outcome
  - Measure the existing system
  - Propose a modification that may produce the desired outcome
  - Apply the modification
  - Retest with the modification
  - Decide if the desired outcome has been achieved

Module Topics

1. Defining Performance
2. Performance Metrics
3. Performance Measurements
4. **Benchmarking and Profiling**

Benchmarking

- Compares two competing pieces of code
  - The code is executed a set number of times
  - Specific metrics are recorded each iteration
  - A final average results is computed
  - The process is repeated with the second piece of code
- Large numbers of repetitions are used
  - Evens out the variation we mentioned earlier
  - Not unsual to use hundreds of thousands of iterations

Profiling

- Looks for bottlenecks in a programs
  - Which methods are called
  - How long each method take to complete
  - Resource bottlenecks
    - *CPU bound – spends time using or waiting for CPU*
    - *IO bound – spends time waiting for or doing I/O*
    - *Memory bound – spends a lot of time paging memory*
  - Average values of data over multiple executions is often used

Module End