optiver

Benchmarking C++
From video games to algorithmic trading

Alexander Radchenko
Optiver

Quiz. How long it takes to run ?

- 3.5GHz Xeon at CentOS 7

```
std::string MakeFizzBuzz(uint64_t number)
{
    std::string output;
    if (number % 3 == 0)
    {
        output += "Fizz";
    }
    if (number % 5 == 0)
    {
        output += "Buzz";
    }
    if (output.empty())
    {
        output = std::to_string(number);
    }
    return output;
}
```

Outline

- Performance challenges in games
- How games tackle performance
- Performance challenges in trading
- How trading tackles performance
- Lightweight tracing use case

Background

- Game development for 15 years
- System / 3D graphics programming and optimisation
- Shipped 8 titles on various platforms
  - PS2, PS3, Xbox 360, Wii, iOS, Android, PC
- 3 years @ Optiver
  - Low latency trading systems
- Performance matters in both domains

Why performance matters ?

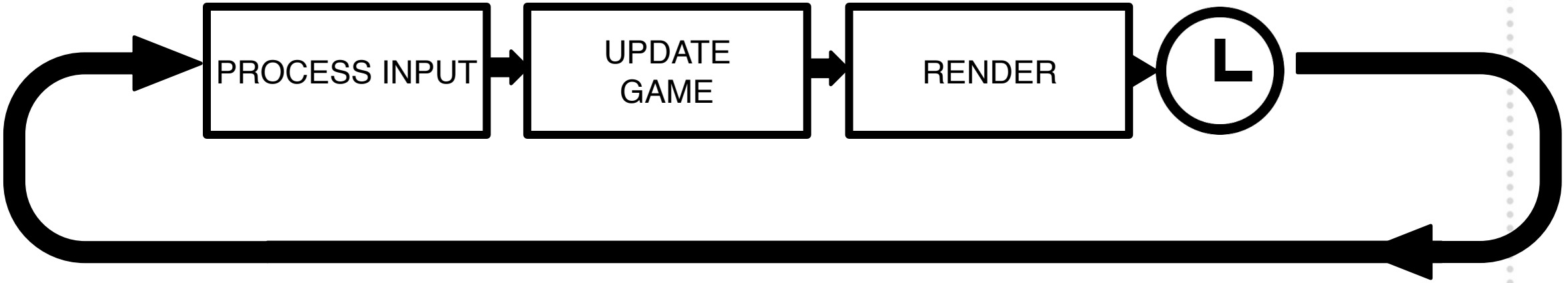
- Slow running game is no fun to play
  - Guess what's the second most common complaint about any PC game ?
- Slow trading system is not making money
  - In fact, it might lose your money

Games

- Soft real-time systems
- Performance is important to certain extent
- Normally run at 30 frames per second
- Consistent load
- Occasional spikes
- Throughput is the king

Game loop

- Needs to be fast enough on target HW
- Performance as a currency
  - Graphics
  - Animations
  - Physics



Performance challenges in games

- PC and Mobiles
  - Fragmented HW
- Game consoles
  - Fixed HW ☺️
  - They are cheap for a reason ☹️
  - Proprietary tools and devkits

Performance challenges in games

Branimir Karadžić
@bkaradzic

Follow

If you got pedal with your PS3 devkit you can use it for Vim:

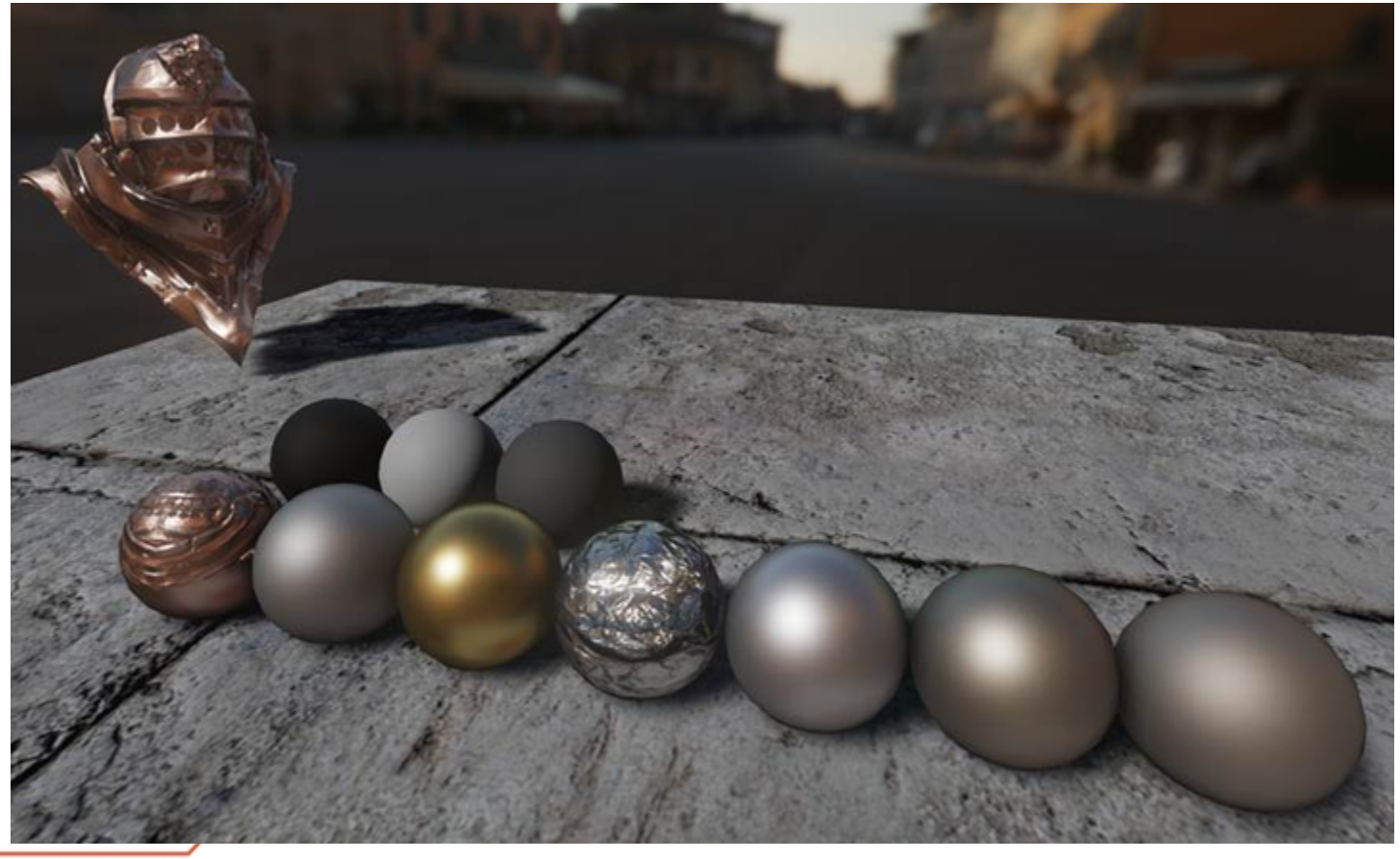
alevchuk/vim-clutch

A hardware pedal for improved text editing in Vim. Contribute to vim-clutch development by creating an account on GitHub.

github.com

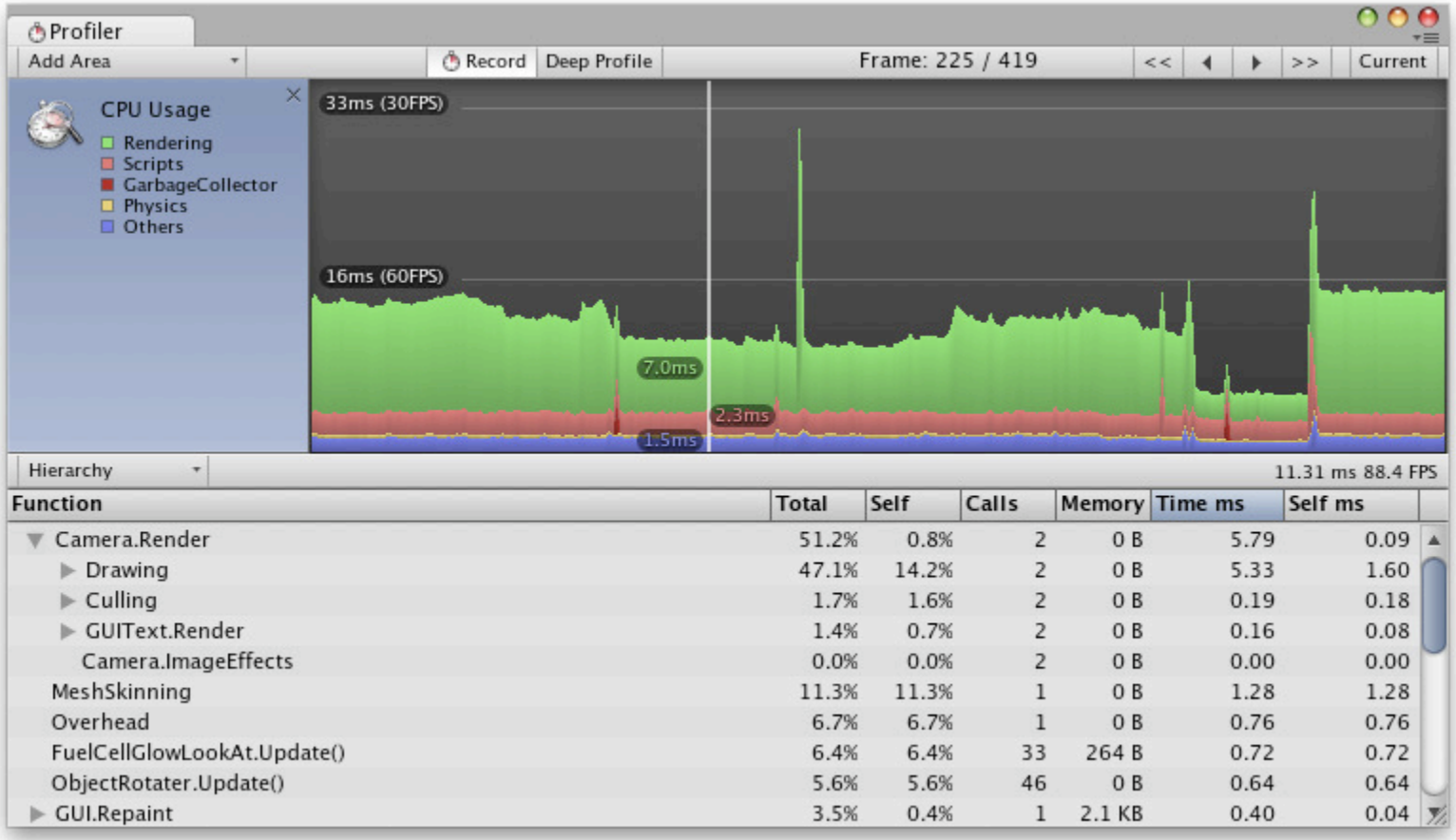
How games tackle performance

- Reference game levels



How games tackle performance

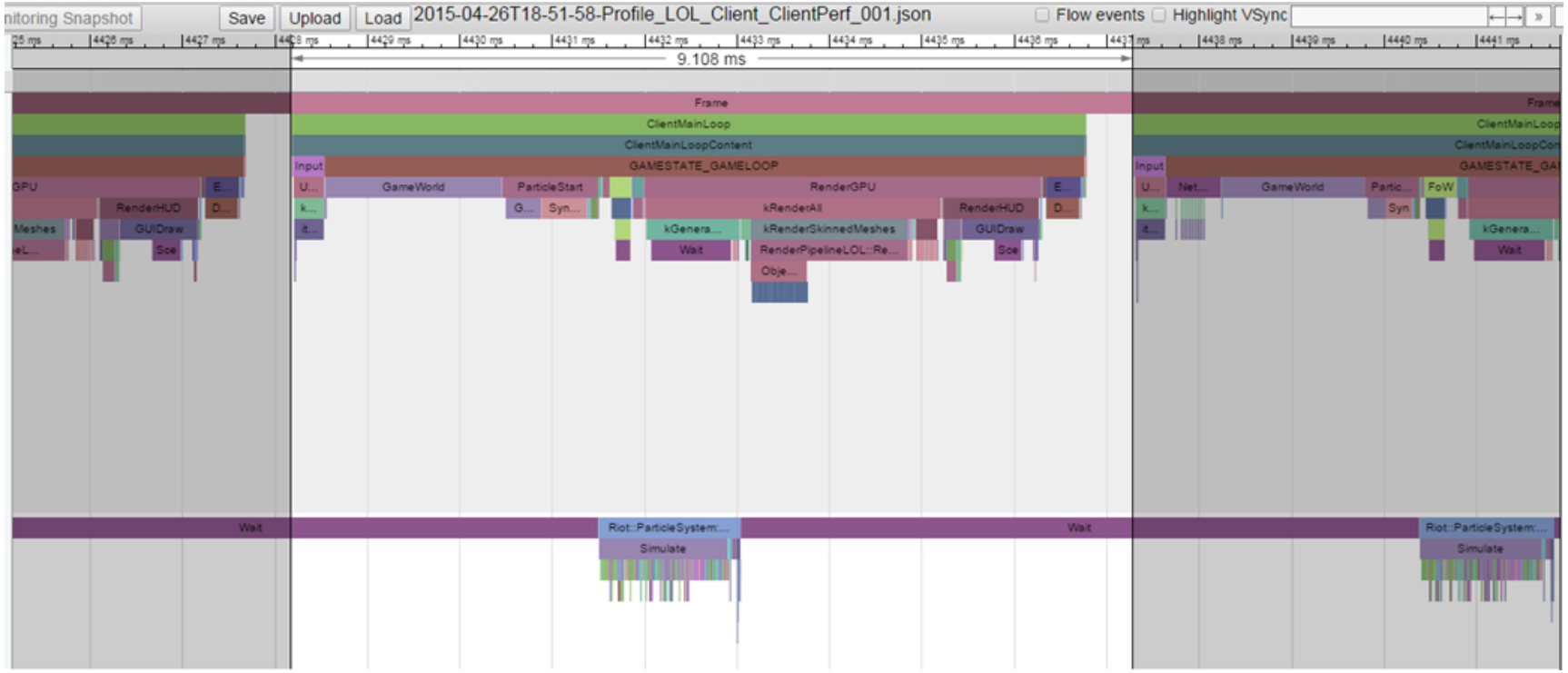
Custom profilers to analyze a whole game session



Function	Total	Self	Calls	Memory	Time ms	Self ms
Camera.Render	51.2%	0.8%	2	0 B	5.79	0.09
Drawing	47.1%	14.2%	2	0 B	5.33	1.60
Culling	1.7%	1.6%	2	0 B	0.19	0.18
GUIText.Render	1.4%	0.7%	2	0 B	0.16	0.08
Camera.ImageEffects	0.0%	0.0%	2	0 B	0.00	0.00
MeshSkinning	11.3%	11.3%	1	0 B	1.28	1.28
Overhead	6.7%	6.7%	1	0 B	0.76	0.76
FuelCellGlowLookAt.Update()	6.4%	6.4%	33	264 B	0.72	0.72
ObjectRotater.Update()	5.6%	5.6%	46	0 B	0.64	0.64
GUI.Repaint	3.5%	0.4%	1	2.1 KB	0.40	0.04

How games tackle performance

Custom profilers to analyze a single frame

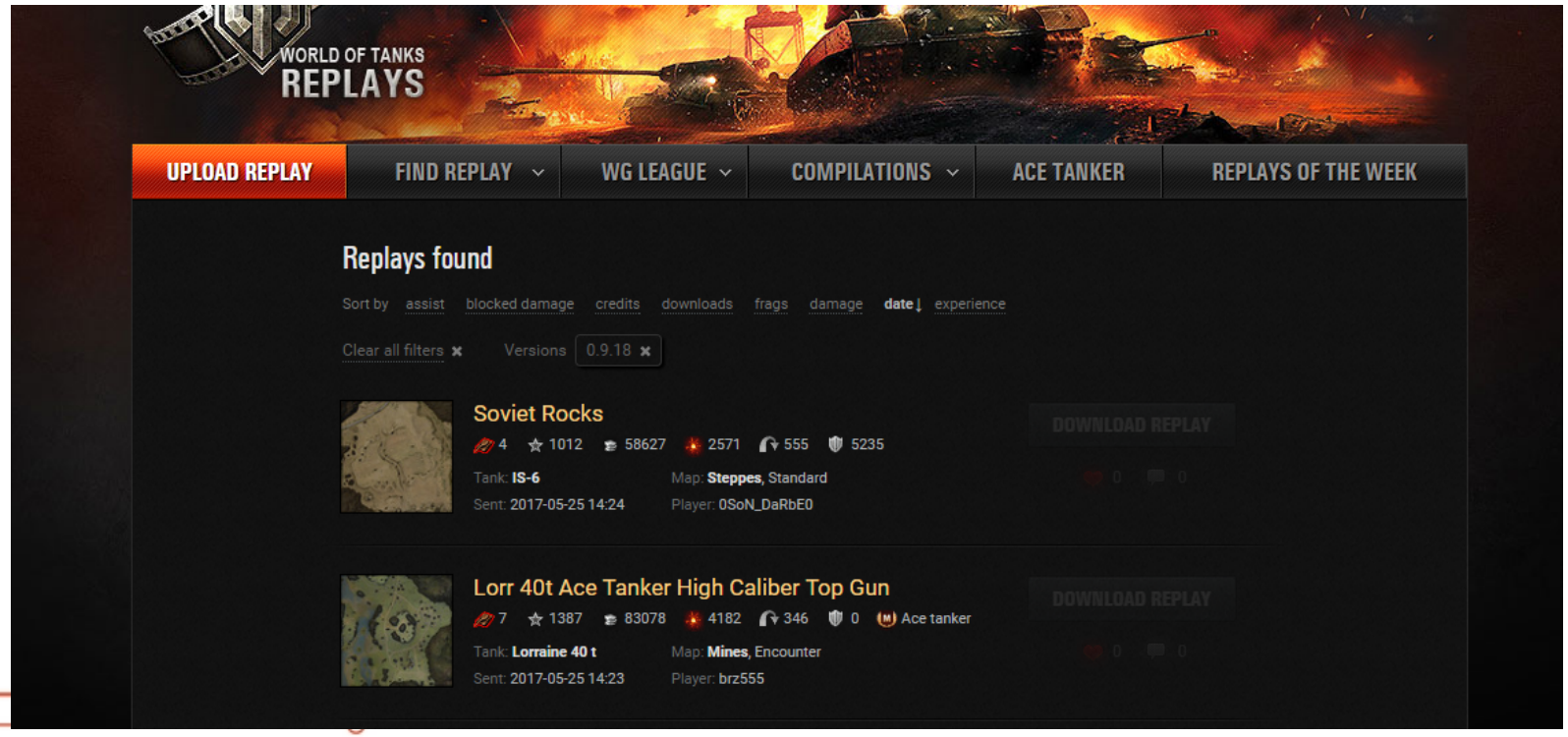


World of tanks

- Online MMO shooter
- Fragmented platform
- Wide range of HW
- Old laptops
- High-end desktops
- Everything in between

Replays

- Recording incoming network traffic
- Initially created to repro bugs
- At some point released to the public



Replays

- Useful tool for performance testing
- Extremely useful as a bug repro tool
- Compare different binaries or different content using the same replay
- In-house tools to drill down and identify problems

Replays: problems

- Protocol upgrades
- Game map changes may invalidate replay
- Security

Vulnerability Within Replay Files (self.WorldofTanks)
submitted 2 years ago * by KeeperOfTheFeels

A couple of months ago I was rooting around within the WoT replay files and their format. I discovered that they way they stored data within certain packets in the replays made it extremely easy to get code execution. After a couple of days working at reliable execution I came upon a reliable way to take any replay file and inject code to execute. This happens very quickly after opening the infected replay file with no way to prevent it once WoT begins reading from the replay.

Regression testing and replays

- Avoiding performance degradation
- Categorize HW: low, medium, high, ultra
- Run replays on a fixed set of HW
- 2s / 5s window averaged frame rate
- Do not allow release with a performance drop outside of fixed threshold

Trading

- Low latency request processing systems
- Performance ~= Money
  - Everyone will identify big opportunities
  - Race to the exchange
  - Winner takes all

optiver
CHALLENGES AWAIT

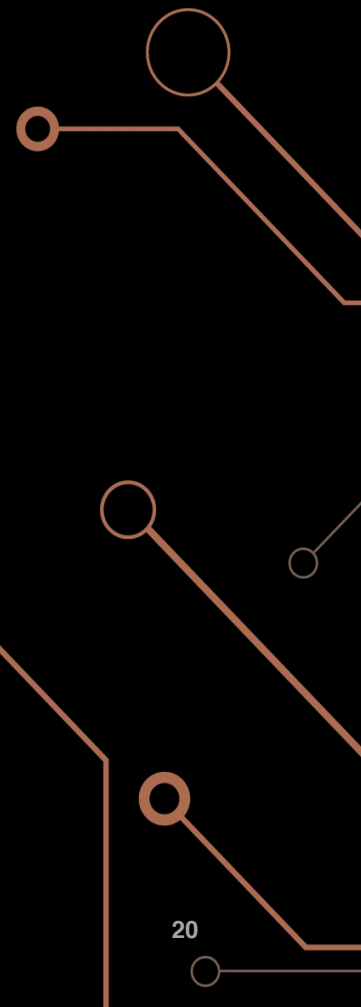
Trading

- Most of the time system is idle
- Bursts on big events
- Latency is the king
  - Speed to take profitable trades
  - Speed to adjust our own orders

optiver
CHALLENGES AWAIT

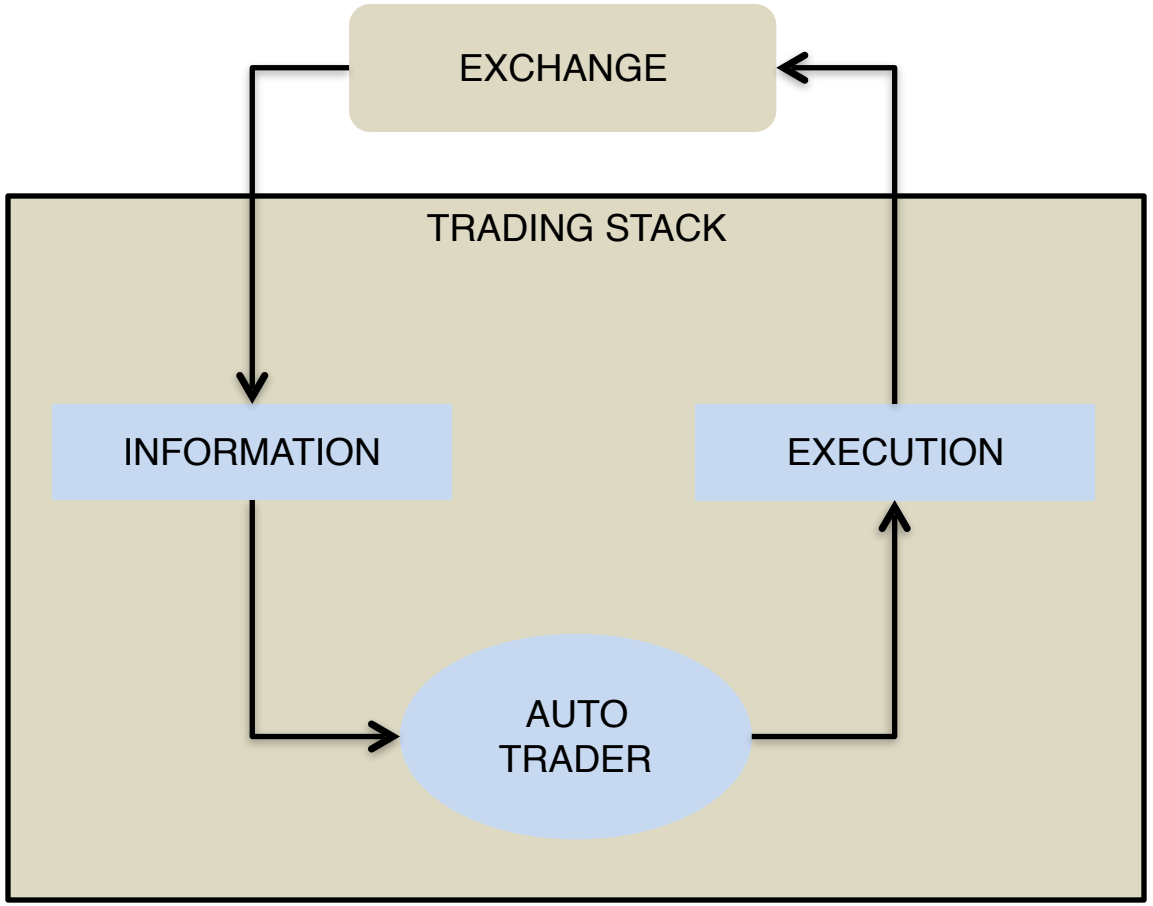
Trading

- Dedicated high end Linux HW
- Speedlab environment to test performance
- Lightweight tracing in speedlab and production
- Using time series DB to store captured data
  - Easy data retrieval for given time range
  - Historical data analysis



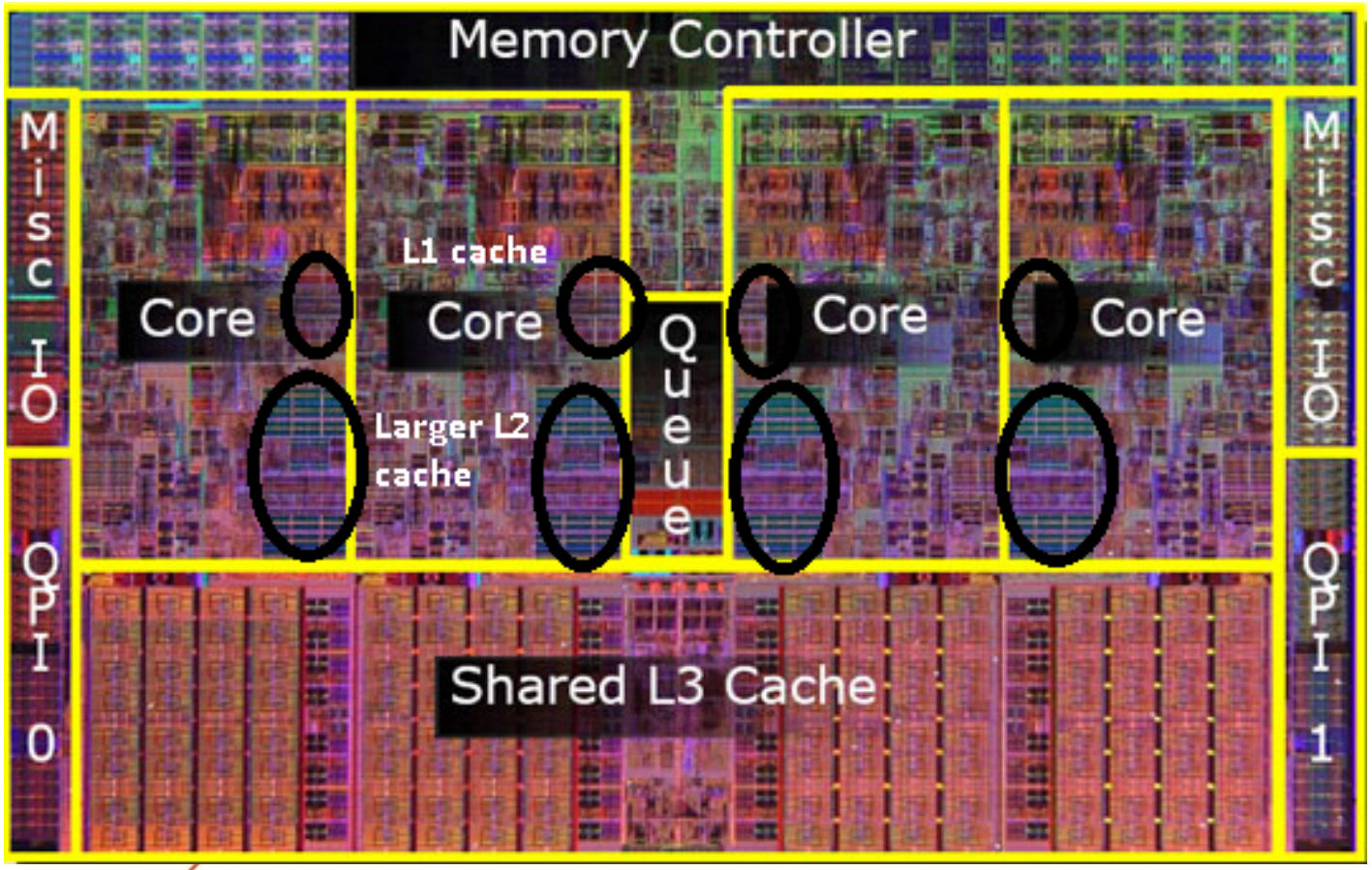
optiver
CHALLENGES AWAIT

Money loop


EXCHANGE
TRADING STACK
INFORMATION
EXECUTION
AUTO
TRADER

Performance challenges in trading

- Cache !



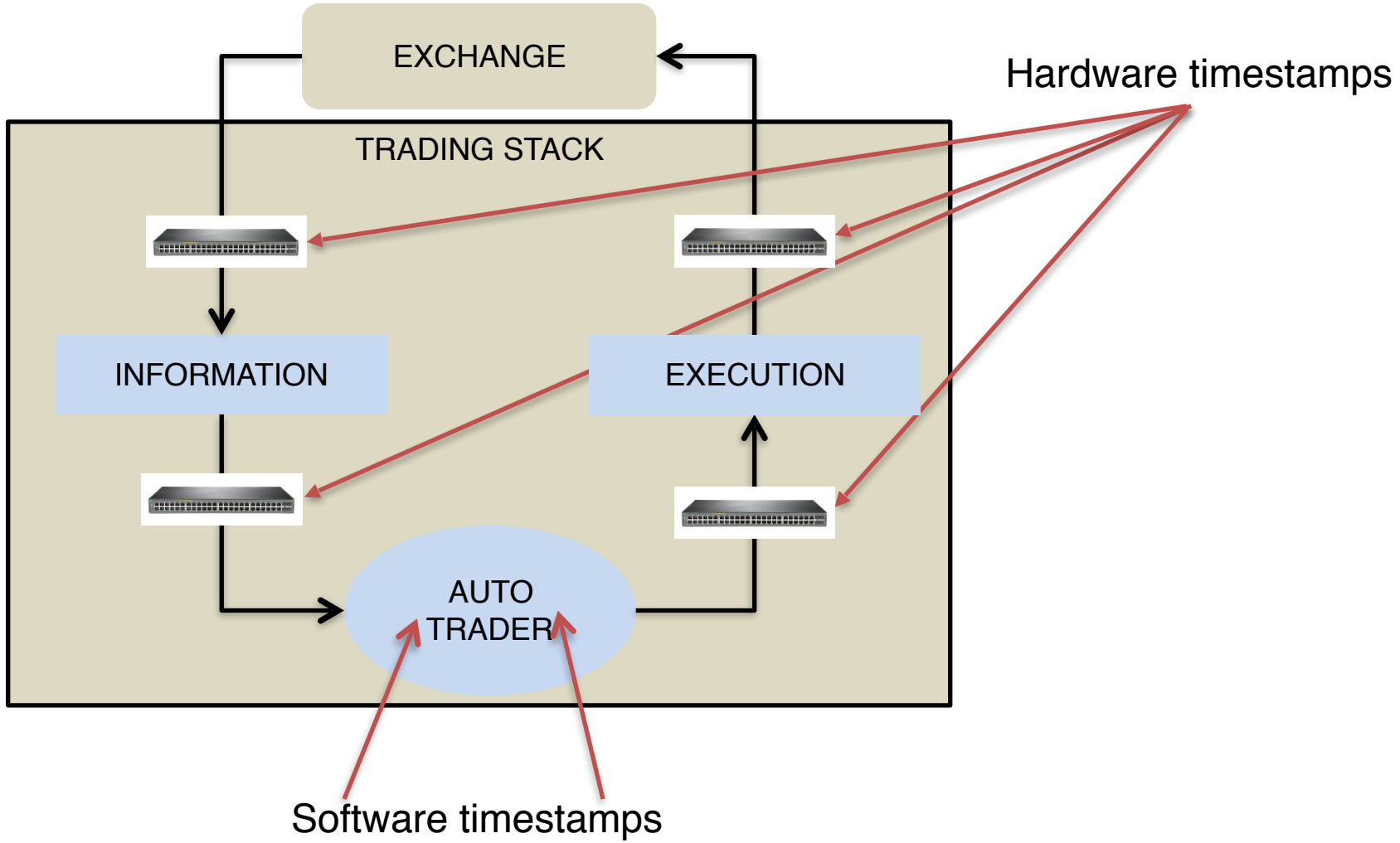
Performance challenges in trading. Cache

- Modern CPUs have multiple caches
  - L1 – fastest level 1 cache
  - L2 – intermediate speed level 2 cache
  - L3 – slow level 3 cache
  - Main memory is the slowest
- You want all your data to be in cache !

Performance challenges in trading

- Generally L3 is shared across all cores
- Pick your neighbors wisely
- HT threads share L1.
  - This is one of the reasons why we disable HT
- Cache warming techniques
  - Keep running
  - Keep touching memory

How trading tackles performance



How trading tackles performance

- Latency histograms
  - simulated environment
  - production
- Detecting outliers
- Drilling down specific events

Lightweight tracing

- How light it is ?
  - HW timestamp cost is a few nanoseconds
  - SW timestamp is higher, still very cheap
- Very useful for understanding performance profile
- Visualizing and recognizing patterns

Low Latency Fizzbuzz

- https://github.com/phejet/benchmarkingcpp_games_trading
- C++ server which reads input data
- Outputs Fizz, Buzz, FizzBuzz or just a number
- How to make it fast ?
- Measure first !!!

Fizzbuzz

- How long do you think it takes run this code ?
- 3.5GHz Xeon at CentOS 7

```
std::string MakeFizzBuzz(uint64_t number)
{
    std::string output;
    if (number % 3 == 0)
    {
        output += "Fizz";
    }
    if (number % 5 == 0)
    {
        output += "Buzz";
    }
    if (output.empty())
    {
        output = std::to_string(number);
    }
    return output;
}
```

Quiz results

```
Run on (24 X 3500 MHz CPU s)
-------------------------------
Benchmark                  Time
-------------------------------
BM_FizzBuzz               98 ns
```

Request processing

```
void Application::HandleRequest(const char* recvBuf, size_t recvSize, IConnection* conn)
{
    TimeCapture timeCapture(mTimingLog);        // <--- START PROCESSING TS
    uint64_t number = ReadInt<uint64_t>(recvBuf, recvSize);
    timeCapture.CaptureFinishParsingTSC();      // <--- FINISH PARSING TS

    std::string output = MakeFizzBuzz(number);
    timeCapture.CaptureFinishProcessingTSC();   // <--- FINISH PROCESSING TS

    conn->Send(output);
    timeCapture.CaptureFinishSendTSC();         // <--- FINISH SENDING TS

    timeCapture.SetRequest(number);
}
```

Timing

```
typedef std::chrono::time_point<std::chrono::high_resolution_clock> Timestamp;

struct TimeLogEntry
{
    uint64_t request = 0;
    Timestamp startTS;
    Timestamp finishParsingTS;
    Timestamp finishProcessingTS;
    Timestamp finishSendTS;
};

class TimeLog
{
public:
    TimeLog();

    void    DumpLog();

private:
    const size_t MAX_TIME_LOG_SIZE = 1000000;
    friend class TimeCapture;
    std::vector<TimeLogEntry> mEntries;
};
```

Timing

```
    void CaptureFinishProcessingTSC()
    {
        mEntry->finishProcessingTS = CaptureTS();
    }

    void CaptureFinishSendTSC()
    {
        mEntry->finishSendTS = CaptureTS();
    }

private:
    Timestamp CaptureTS()
    {
        return std::chrono::high_resolution_clock::now();
    }
```

Using Epoch

```
unsigned long long ToEpoch(Timestamp ts)
{
    return std::chrono::duration_cast<std::chrono::nanoseconds>(
                    ts.time_since_epoch()).count();
}
```

Timings output

```
Header: TimeLogEntry: request=uint64,startTS=timestamp,finishParsingTS=timestamp,fini
Data: TimeLogEntry: request=639788,startTS=1481021544033232856,finishParsingTS=148102
Data: TimeLogEntry: request=275754,startTS=1481021544033248315,finishParsingTS=148102
Data: TimeLogEntry: request=736735,startTS=1481021544033250390,finishParsingTS=148102
Data: TimeLogEntry: request=892288,startTS=1481021544033250703,finishParsingTS=148102
Data: TimeLogEntry: request=422500,startTS=1481021544033251379,finishParsingTS=148102
Data: TimeLogEntry: request=219419,startTS=1481021544033251600,finishParsingTS=148102
Data: TimeLogEntry: request=27509,startTS=1481021544033251945,finishParsingTS=1481021
Data: TimeLogEntry: request=650235,startTS=1481021544033252329,finishParsingTS=148102
Data: TimeLogEntry: request=221220,startTS=1481021544033252604,finishParsingTS=148102
Data: TimeLogEntry: request=809621,startTS=1481021544033252836,finishParsingTS=148102
Data: TimeLogEntry: request=806014,startTS=1481021544033253189,finishParsingTS=148102
Data: TimeLogEntry: request=340910,startTS=1481021544033253558,finishParsingTS=148102
Data: TimeLogEntry: request=957256,startTS=1481021544033253793,finishParsingTS=148102
```

Macro benchmark

```
def _run_benchmark(self):
    '''Run process under test on simulation data and print timings'''
    os.chdir(self.workspace_dir)
    cmd = '%s %s > output.txt' % (FULL_BINARY_PATH, SIMULATION_FILENAME)
    if IS_LINUX:
        cmd = 'taskset -c 1 ' + cmd
    os.system(cmd)
    print 'Parsing collected data'
    self._print_stats(self._parse_timings())

def test_bursts(self):
    # generate test data
    NUM_REQUESTS = 1000000

    file = ''
    for i in range(NUM_REQUESTS):
        file += '%d %d\n' % (random.randint(1000, 1000000), random.randint(0, 100))
    with open(os.path.join(self.workspace_dir, SIMULATION_FILENAME), 'w') as f:
        f.write(file)

    self._run_benchmark()
```

Quick feedback

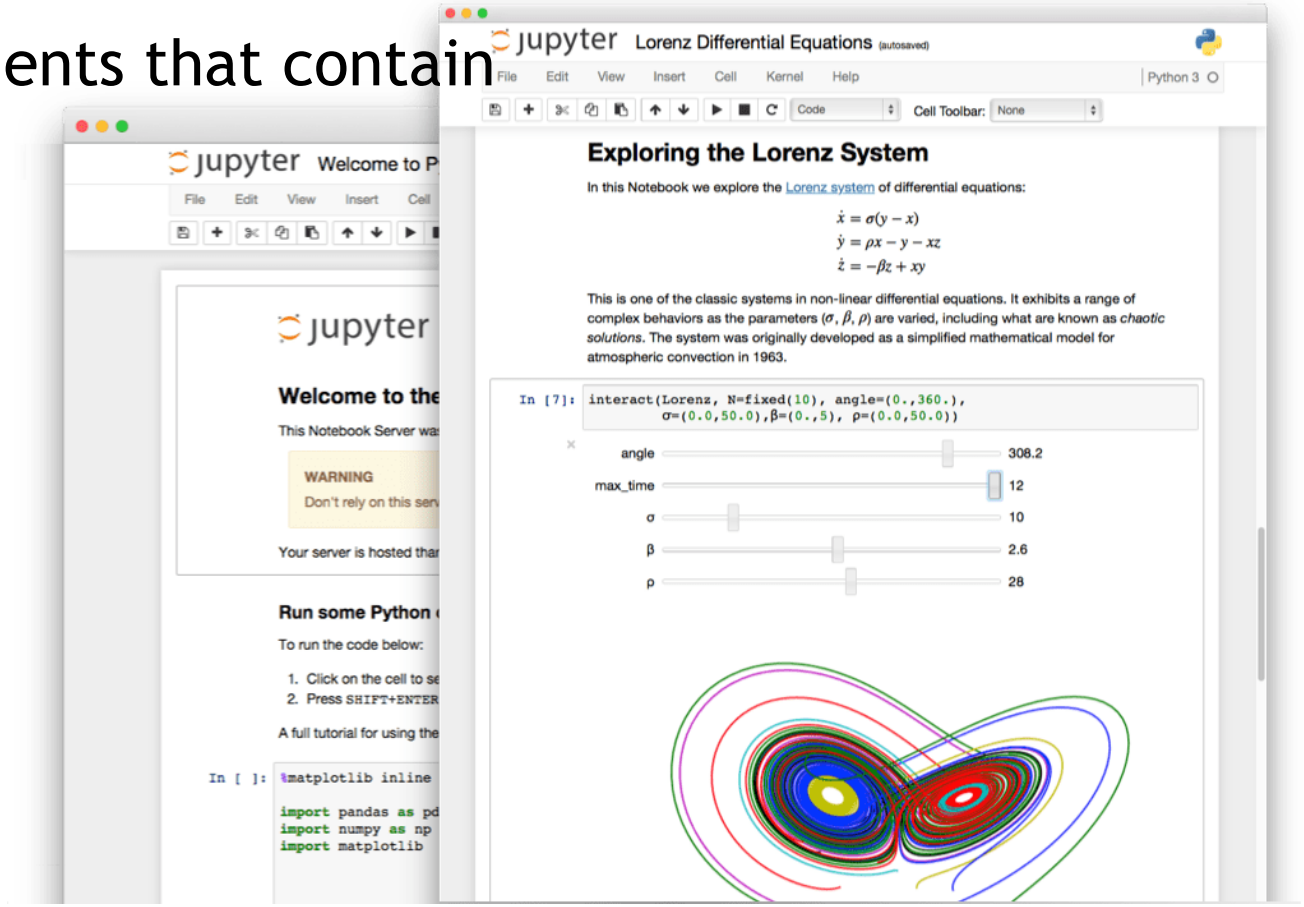
- Time in nanoseconds

test_bursts_large_numbers

Name	avg	25%	50%	75%
Parsing	38	37	38	39
Processing	97	47	133	138
Send	42	37	41	43
Total	178	127	209	216

Jupyter notebooks

- Open-source web application
- Create and share documents that contain
  - Live code
  - Equations
  - Visualizations
  - Narrative text



Jupyter notebook for in-depth analysis

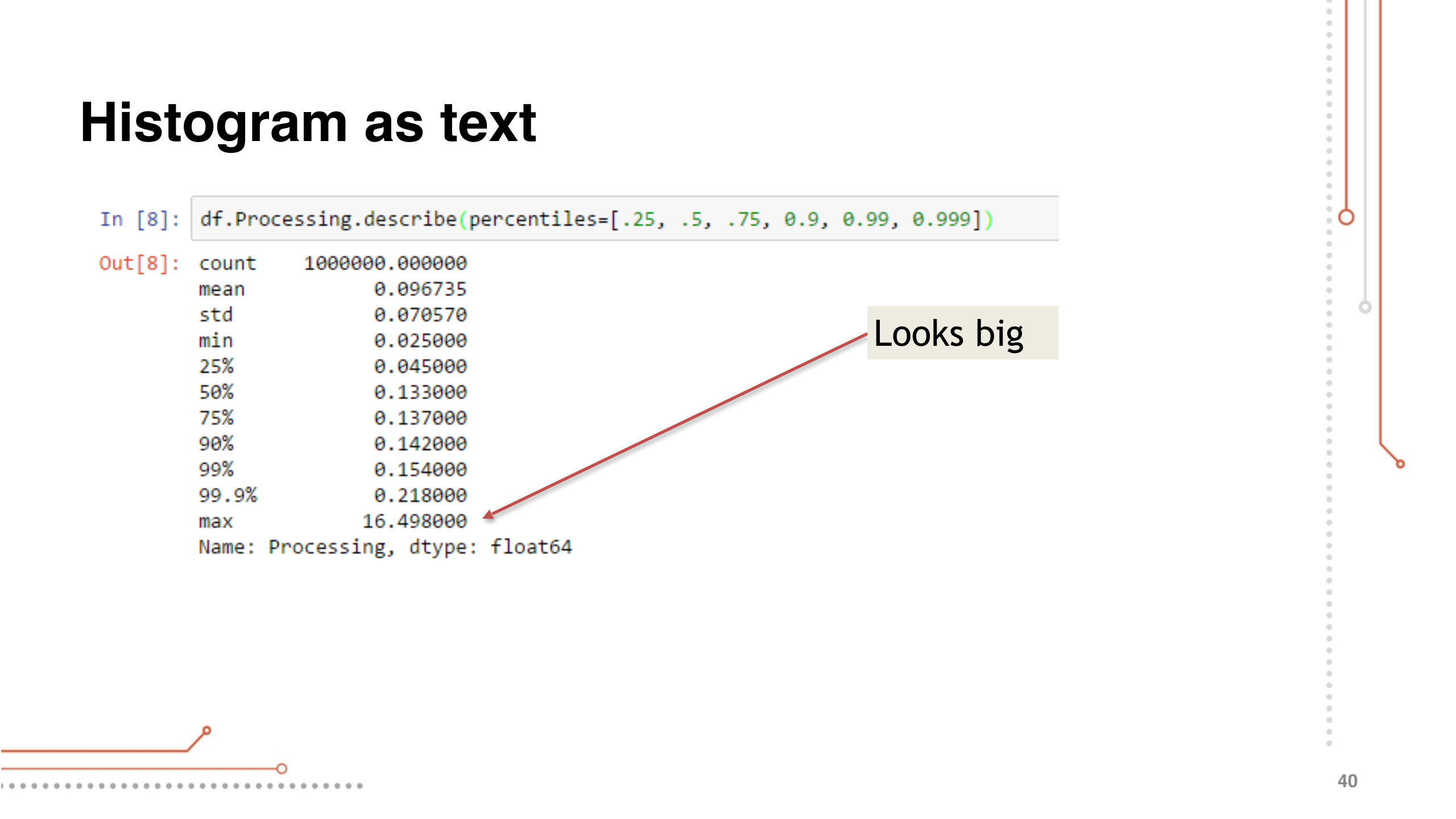
```
In [7]: df.head()
```

Out[7]:

	Parsing	Processing	Send	Total	finishParsingTS	finishProcessingTS	finishSendTS	request	startTS
0	0.177	11.333	3.711	15.221	2016-12-06 11:17:11.672531338	2016-12-06 11:17:11.672542671	2016-12-06 11:17:11.672546382	639788	2016-12-06 11:17:11.672531161
1	0.047	1.678	0.233	1.958	2016-12-06 11:17:11.672546696	2016-12-06 11:17:11.672548374	2016-12-06 11:17:11.672548607	25985	2016-12-06 11:17:11.672546649
2	0.050	0.084	0.048	0.182	2016-12-06 11:17:11.672548781	2016-12-06 11:17:11.672548865	2016-12-06 11:17:11.672548913	275754	2016-12-06 11:17:11.672548731
3	0.038	0.354	0.054	0.446	2016-12-06 11:17:11.672549057	2016-12-06 11:17:11.672549411	2016-12-06 11:17:11.672549465	223987	2016-12-06 11:17:11.672549019
4	0.038	0.057	0.045	0.140	2016-12-06 11:17:11.672549610	2016-12-06 11:17:11.672549667	2016-12-06 11:17:11.672549712	736735	2016-12-06 11:17:11.672549572

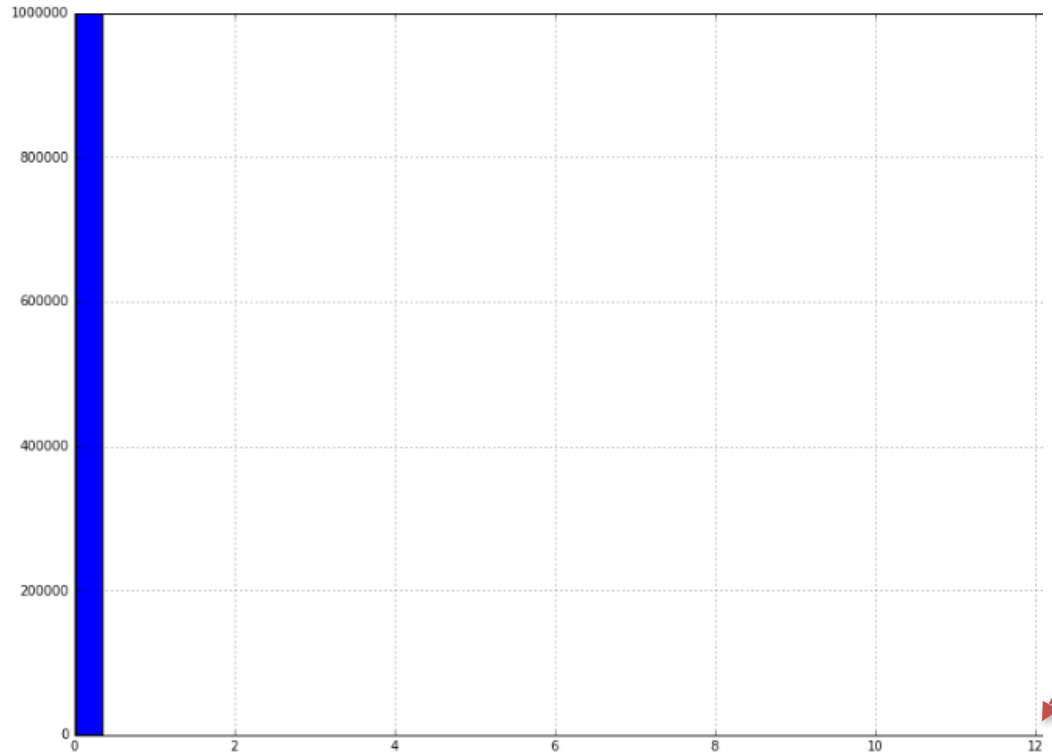
Histogram as text

```
In [8]: df.Processing.describe(percentiles=[.25, .5, .75, 0.9, 0.99, 0.999])

Out[8]: count    1000000.000000
        mean           0.096735
        std            0.070570
        min            0.025000
        25%            0.045000
        50%            0.133000
        75%            0.137000
        90%            0.142000
        99%            0.154000
        99.9%          0.218000
        max           16.498000
        Name: Processing, dtype: float64
```



Beware of outliers

```
In [17]: df.Processing.hist(grid=True, figsize=(20, 10), bins=50)
Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x1b8ba5c0>
```



Outlier

Discarding outliers

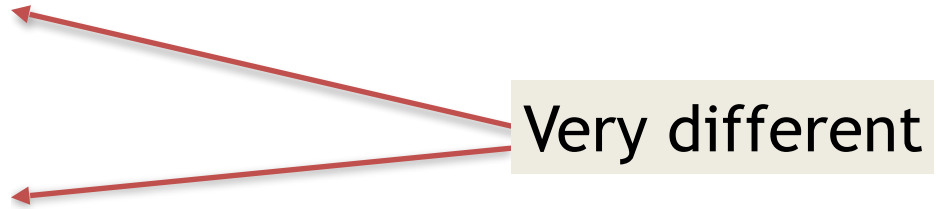
In [12]:
```
S = df.Processing

S = S[~((S - S.mean()).abs() > 3.5 * S.std())]
df.NORM = S
df.NORM.describe(percentiles=[.25, .5, .75, 0.9, 0.99, 0.999])
```

Out[12]:
```
count    999919.000000
mean          0.096387
std           0.045414
min           0.025000
25%           0.045000
50%           0.133000
75%           0.137000
90%           0.142000
99%           0.154000
99.9%         0.216000
max           0.338000
Name: Processing, dtype: float64
```

Max value more reasonable

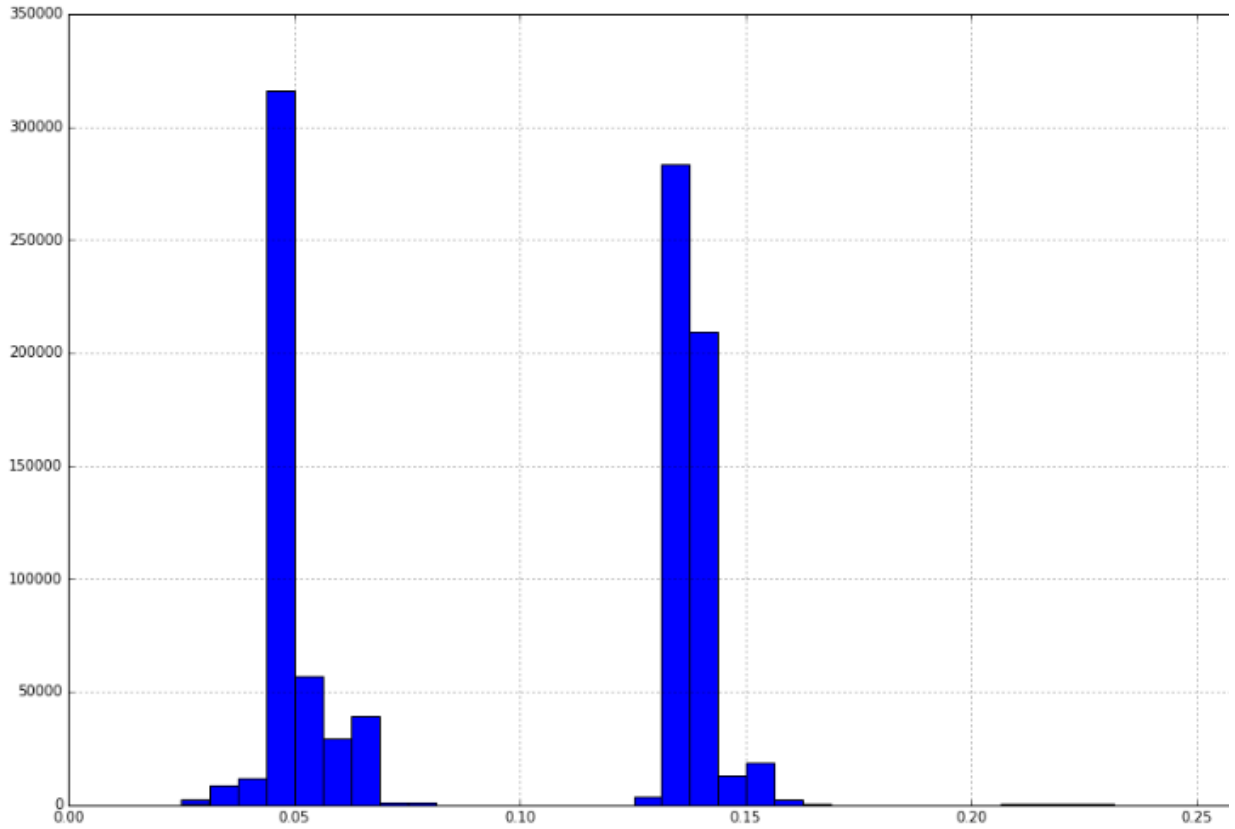
Distribution not normal ?

count	999919.000000
mean	0.096387
std	0.045414
min	0.025000
25%	0.045000
50%	0.133000
75%	0.137000
90%	0.142000
99%	0.154000
99.9%	0.216000
max	0.338000



Two events ?

```
In [18]: df.NORM.hist(grid=True, figsize=(20, 10), bins=50)
Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x1c4c6240>
```



Optiver profiler

- In-house tracing profiler
- Mark interesting parts of your code
  - Scope guards to capture entry/exit timestamps and function name
  - Single named events
- Nanosecond precision
- Multiple tools to view results
- Tarantula is the most interesting one

Tarantula

```
[path count = 2725883, steps = 9, total time = 660 ms (61.26%, 61.26% acc.)]
  mean: 242 ns, min: 214 ns, max: 5.6 us, iqr: 5 ns
  percentiles: 231 ns [235 ns, 238 ns, 240 ns] 251 ns

  accumulated                                  path                                    delta
              +-------------------------------------------------------------------+
   0%    0 ns |*                                                                  |    0 ns   0% enter HandleRequest
   5%   11 ns |  [*                                                                |   11 ns   5%   enter ReadInt
   8%   20 ns |    +*                                                             |    9 ns   4%     enter atoi
  23%   55 ns |        [[*]                                                       |   35 ns  15%     leave atoi
  26%   63 ns |           *                                                       |    8 ns   3%   leave ReadInt
  30%   73 ns |             [*                                                    |    9 ns   4%   enter MakeFizzBuzz
  35%   85 ns |                 *]                                                |   12 ns   5%     enter to_string
  86%  209 ns |                                                  [*]              |  124 ns  51%     leave to_string
  89%  217 ns |                                                    *              |    8 ns   3%   leave MakeFizzBuzz
 100%  242 ns |                                                       [*]         |   25 ns  11% leave HandleRequest
              +-------------------------------------------------------------------+
               0 ns                          (4 ns)                          242 ns
[path count = 2385144, steps = 7, total time = 379 ms (35.13%, 96.39% acc.)]
  mean: 159 ns, min: 126 ns, max: 13.6 us, iqr: 7 ns
  percentiles: 144 ns [149 ns, 152 ns, 156 ns] 182 ns

  accumulated                                  path                                    delta
              +-------------------------------------------------------------------+
   0%    0 ns |*                                                                  |    0 ns   0% enter HandleRequest
   7%   11 ns |    +*                                                             |   11 ns   7%   enter ReadInt
  13%   20 ns |       *]                                                          |    9 ns   6%     enter atoi
  36%   57 ns |           [[+*                                                    |   36 ns  23%     leave atoi
  41%   64 ns |              [*                                                   |    8 ns   5%   leave ReadInt
  46%   74 ns |                 [*                                                |    9 ns   6%   enter MakeFizzBuzz
  84%  133 ns |                                          [[+*.........]           |   59 ns  37%   leave MakeFizzBuzz
 100%  159 ns |                                                       [*]         |   26 ns  16% leave HandleRequest
              +-------------------------------------------------------------------+
               0 ns                          (2 ns)                          159 ns
```

Two codepaths !

```
std::string MakeFizzBuzz(uint64_t number)
{
    std::string output;
    if (number % 3 == 0)
    {
        output += "Fizz";
    }
    if (number % 5 == 0)
    {
        output += "Buzz";
    }
    if (output.empty())
    {
        output = std::to_string(number);
    }
    return output;
}
```

Optimisation

- FizzBuzz logic is the most expensive part of our request processing
- How can we make it faster ?

```
std::string MakeFizzBuzz(uint64_t number)
{
    std::string output;
    if (number % 3 == 0)
    {
        output += "Fizz";
    }
    if (number % 5 == 0)
    {
        output += "Buzz";
    }
    if (output.empty())
    {
        output = std::to_string(number);
    }
    return output;
}
```

Brute force approach

- Write custom function instead of using std::to_string
- Return result as const char* and use static buffer

Look at high level

```
void Application::HandleRequest(const char* recvBuf, size_t recvSize, IConnection* conn)
{
    TimeCapture timeCapture(mTimingLog);        // <--- START PROCESSING TS
    uint64_t number = ReadInt<uint64_t>(recvBuf, recvSize);
    timeCapture.CaptureFinishParsingTSC();      // <--- FINISH PARSING TS

    std::string output = MakeFizzBuzz(number);
    timeCapture.CaptureFinishProcessingTSC();   // <--- FINISH PROCESSING TS

    conn->Send(output);
    timeCapture.CaptureFinishSendTSC();         // <--- FINISH SENDING TS

    timeCapture.SetRequest(number);
}
```

Avoid int->string conversion

```
const char* MakeFizzBuzz(const char* input)
{
    int number = std::atoi(input);
    const bool isFizz = (number % 3) == 0;
    const bool isBuzz = (number % 5) == 0;
    if (isFizz && isBuzz)
    {
        return "FizzBuzz";
    }
    if (isFizz)
    {
        return "Fizz";
    }
    if (isBuzz)
    {
        return "Buzz";
    }
    return input;
}
```

Conclusion

- Having a simple and reproducible way to measure performance is very important
- Visualising performance data helps to understand it
- Understanding is a necessary first step before optimization
- When optimizing code, always look at high level picture