The Performance Investigator's **Field Guide**

Sasha Goldshtein
CTO, Sela Group

goldshtn
goldshtn

DOTNEXT

The Plan

- I want to share with you my strategy, tactics, and vision for performance investigations in the field
- You'll learn:

❑ What a performance investigation looks like

❑ Methods and anti-patterns for performance investigations

❑ Tools for production profiling and monitoring .NET applications

❑ How to avoid getting lied to by tools, dashboards, and reports

Structure of a Performance Investigation

1. Obtain the problem description
2. Build a system diagram
3. Run a quick performance checklist
4. Understand which component is exhibiting the problem
5. Investigate thoroughly
6. Find the root cause
7. Resolve the issue
8. Verify resolution
9. Conduct and document post-mortem

From: Cust McCustomer <custmcc@acme.com>
Subject: URGENT PERFORMANCE PROBLEM!!!1
Priority: High

The application feels SLOW. Users can't really put their finger on it but it's bad. Mind taking a look?

- C

From: Cust McCustomer <custmcc@acme.com>
Subject: Quote for performance work

We have a 2-day budget for some performance work, can you take a look at our production environment and see what you can find?

- C

From: Monitor McMonitor <noreply@acme.com>
Subject: Latency increase in NYC-FE-003
Date: 30 Apr 2017 07:35 AM

Since 4:25am, 95th percentile latency to the frontend ASP.NET instance increased to 1400ms (from 60ms). This is consistent across geographies and hasn't gone down. Auto-scaling has kicked in but didn't help.

From: Boss McBoss <boss@acme.com>
Subject: URGENT Performance of CT screen
Priority: High

In the latest beta, clicking from the patient details to the CT screen takes 11 seconds in certain hospitals. This is unacceptable and needs to be under 2s. Please send quote.

- B

<me> What are your latency requirements?
<customer> We need an avg response of 20ms.
<me> What's the worst case requirement?
<customer> We don't have one.
<me> So some requests can take more than 5 minutes?
<customer> What?! No. Nothing worse than 100ms.
<me> Even if it's just 2 requests/day?
<customer> Fine. Nothing worse than 2 seconds.
<me> How often is a 500ms response acceptable? E.g. 96% 1ms, 4% 500ms works out to ~21ms mean.
*** customer has left the chat**

Rephrased from "How Not To Measure Latency", Azul Systems

Performance Metrics, Goals, And Monitoring

- Performance metric/goal:

```
 90% of full-text queries should complete under 200ms
 99% of full-text queries should complete under 600ms
100% of full-text queries should complete under 2000ms
```

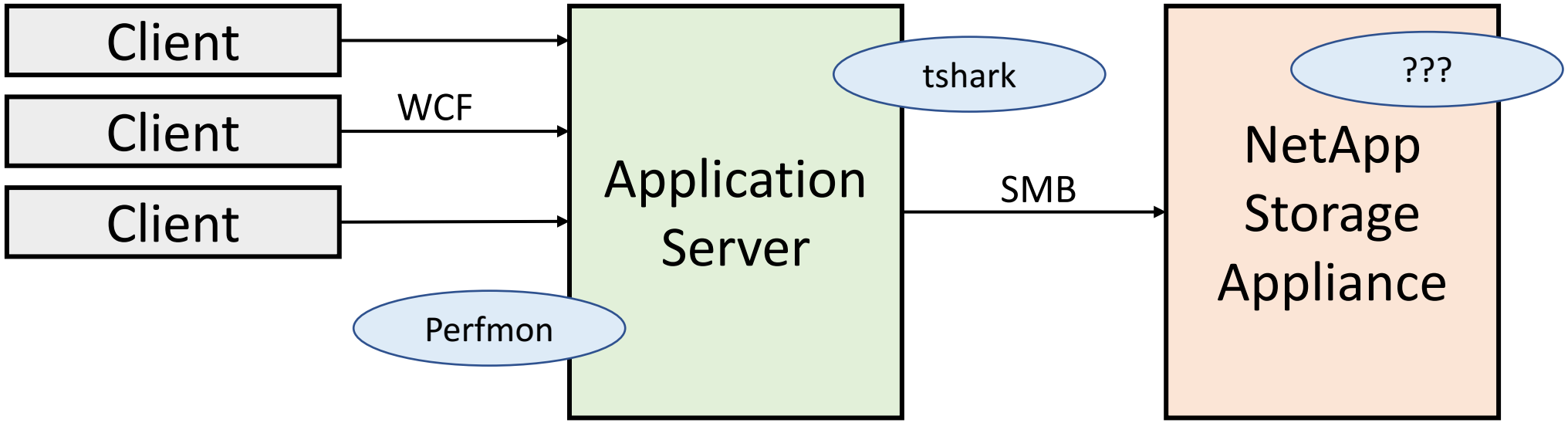
- **Performance metrics don’t live in a vacuum**
- Derive performance metrics from *business goals*
- Monitor these metrics in your APM solution, home-made dashboard or collection script, and get alerts

Investigation Anti-Methods

1. Make assumptions
2. Trust “instincts” and irrational beliefs
3. Look under the street light
4. Use random tools
5. Blame the tools

How I Failed to Diagnose a Slow Document Management System

- Customer reported load and save operations on certain documents caused a timeout
- Linked to network storage access (NetApp through SMB on LAN)
- I asked to investigate network latency and storage latency



From: Sasha Goldshtein
Subject: Network latency to DOC-FILER01

After processing 24 hours of network traces with tshark, the mean SMB latency to storage was 11ms, but peak values of up to 1200ms were observed! Need to look at network traces from the NetApp appliance, and at storage latency counters.

- Sasha

From: Network McNetwork
Subject: RE: Network latency to DOC-FILER01

> Need to look at network traces from the
> NetApp appliance.

PFA the network traces from the NetApp appliance. If I'm not mistaken, the peak _network_ latency was 10ms, with a mean <1ms.

From: Cust McCustomer
Subject: RE: FW: Re: Fwd: Perf investigation

The NetApp consultant who knows how to monitor NetApp latency will be in next week. Our local sysadmin says NetApp latency is under 5µs, so there's no way it's the storage.

- C

From: Cust McCustomer
Subject: RE: FW: Re: Fwd: Perf investigation

The NetApp consultant was here. He says it's 5ms, not 5µs. Almost the same thing. So, it's not the storage either?!

- C

From: Sasha Goldshtein
Subject: RE: FW: Re: Fwd: Perf investigation

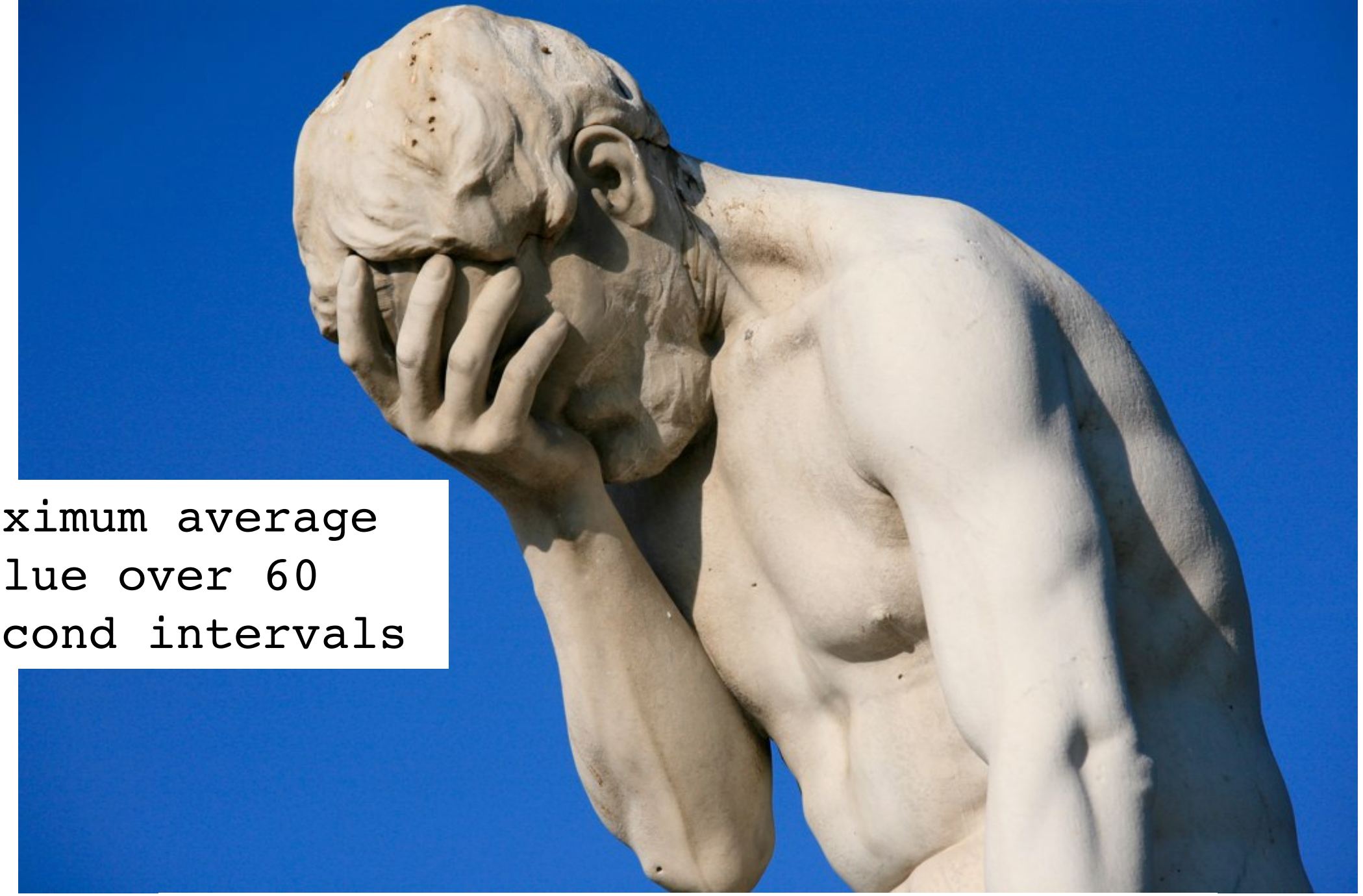
> The NetApp consultant was here. He says
> it's 5ms, not 5µs. Almost the same thing.

Is that an _average_ or the _peak_ value?

From: Cust McCustomer
Subject: RE: FW: Re: Fwd: Perf investigation

>> The NetApp consultant was here. He says
>> it's 5ms, not 5µs. Almost the same thing.
>
> Is that an _average_ or the _peak_ value?

He says it's the maximum average value over
60 second intervals.



From: Sasha Goldshtein
Subject: RE: FW: Re: Fwd: Perf investigation

OK, can we at least see the raw values from the following NetApp counters? It's still averages, but updated every second.

cifs object

Counter name	Description	Unit
cifs_latency	Average latency for CIFS operations	millisec
cifs_ops	Total number of CIFS operations	per sec
cifs_read_latency	Average latency, for CIFS read operations	millisec
cifs_write_latency	Average latency, for CIFS write operations	millisec
cifs_read_ops	Total number of CIFS read operations	per sec
cifs_write_ops	Total number of CIFS write operations	per sec

From NetApp documentation

From: Cust McCustomer
Subject: RE: FW: Re: Fwd: Perf investigation

> OK, can we at least see the raw values
> from the following NetApp counters?

We don't have the budget to bring the NetApp consultant again. I'll keep you posted.

From: Cust McCustomer
Subject: RE: FW: Re: Fwd: Perf investigation

So I talked to the NetApp guy and he says in our configuration, there is _no way_ the peak value is above 5ms. It is just not possible.

Investigation Anti-Methods

1. ~~Make assumptions~~
2. ~~Trust “instincts” and irrational beliefs~~
3. Look under the street light
4. Use random tools
5. Blame the tools

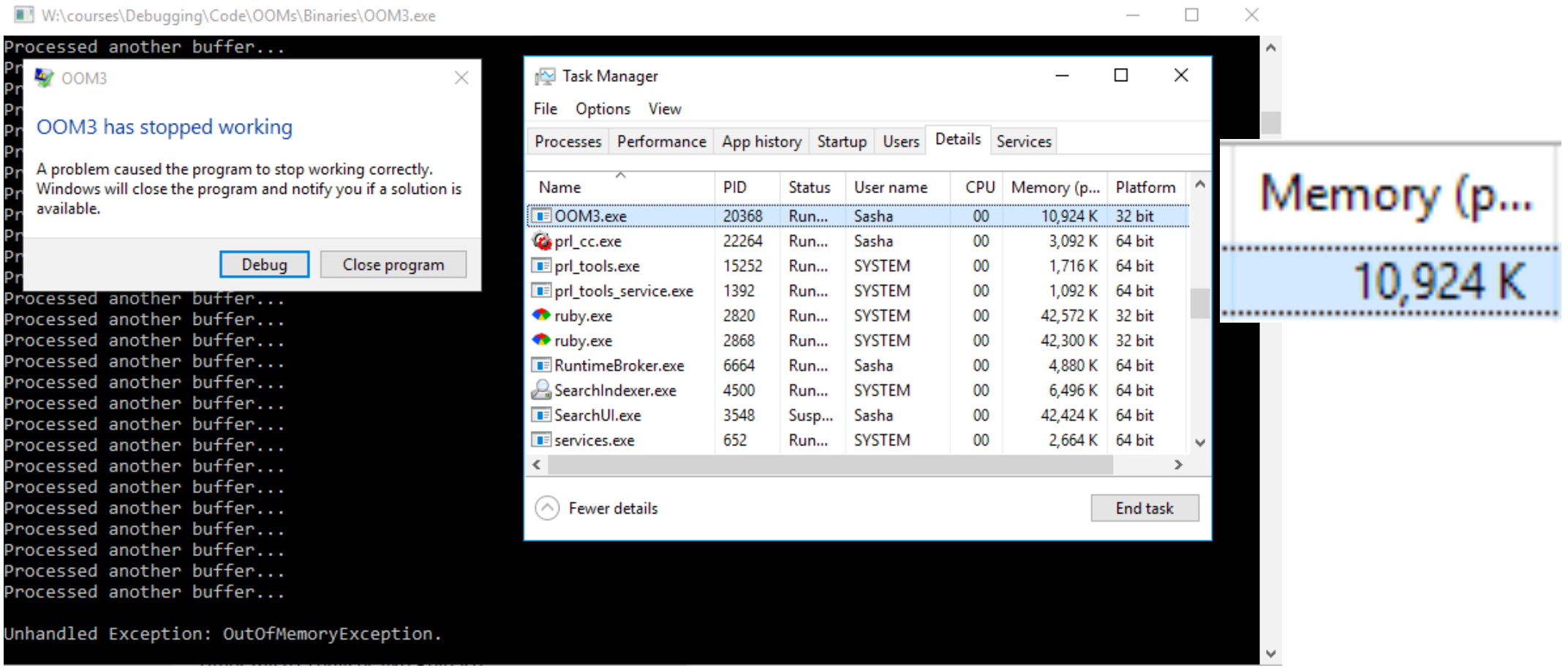
Let’s Run a CPU Profiler Because We Just Bought an Expensive License

- Profiling a web crawler, we get the following results. Clearly, `Console.WriteLine` is slow and has a bug.

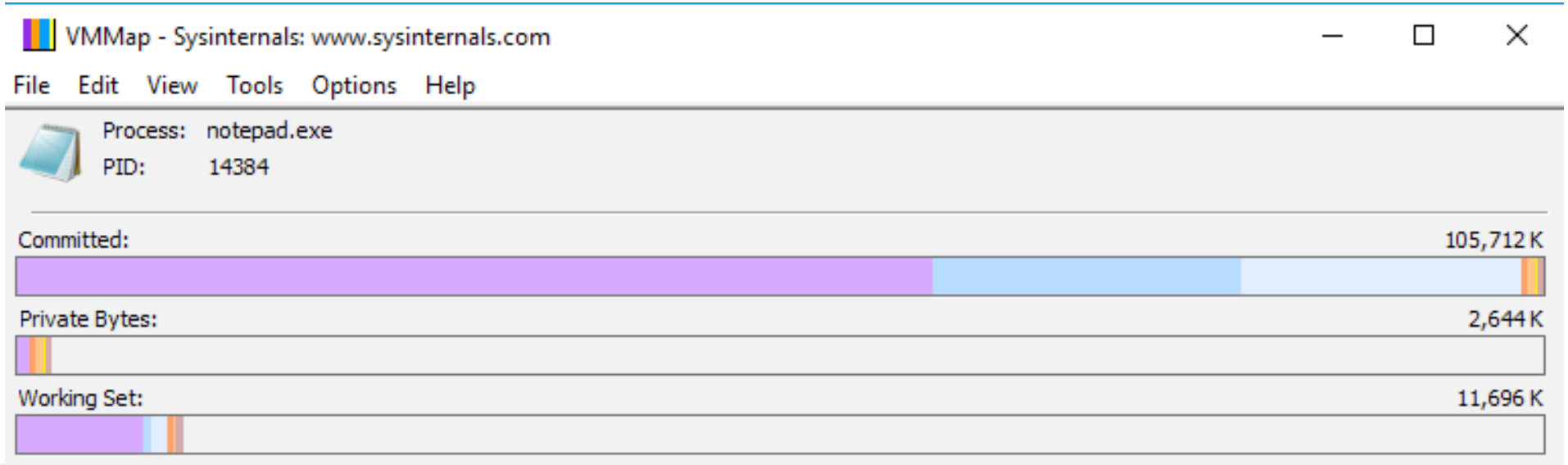
Hot Path

Function Name	Inclusive Samples %	Exclusive Samples %
[kernel32.dll]	100.00	0.00
Thread::intermediateThreadProc	81.93	0.42
WebCrawler.Program+<>c.<Main>b__1_0	81.51	0.00
WebCrawler.Program.CrawlWebsite	81.51	0.00
System.Console.WriteLine	**66.39**	**66.39**

This Application Doesn’t Have a Memory Leak



This Tool Must Be Broken



Type	Size
Total	2,147,605,280 K
Image	63,396 K
Mapped File	21,292 K
Shareable	2,147,509,284 K
Heap	1,528 K

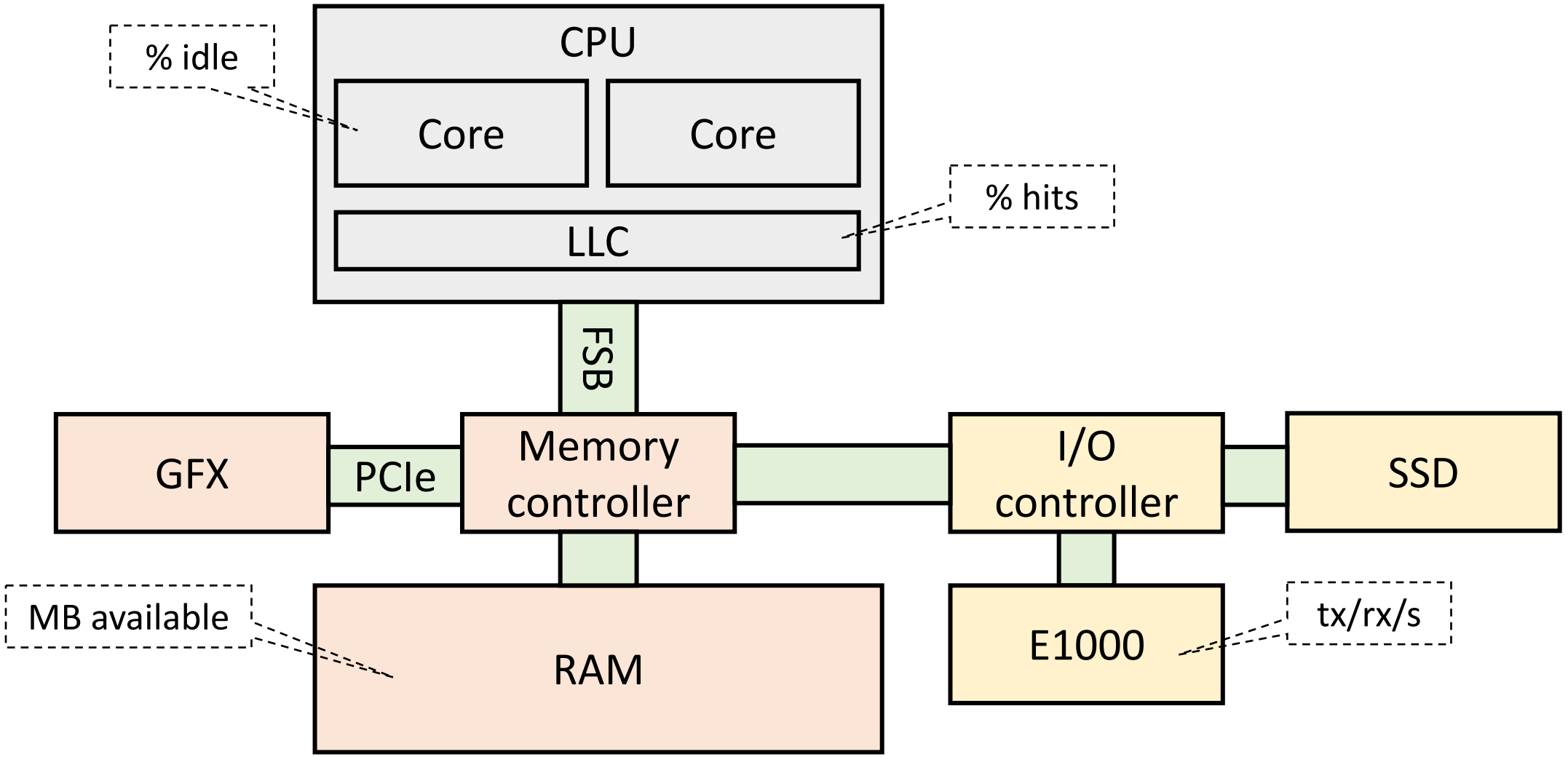
Committed	Private	Total WS	Private WS	Shareable WS	Sh
105,712 K	2,644 K	11,696 K	1,980 K	9,716 K	
63,396 K	1,072 K	8,856 K	908 K	7,948 K	
21,292 K		616 K		616 K	
19,388 K		1,144 K		1,144 K	
512 K	448 K	444 K	440 K	4 K	

The USE Method

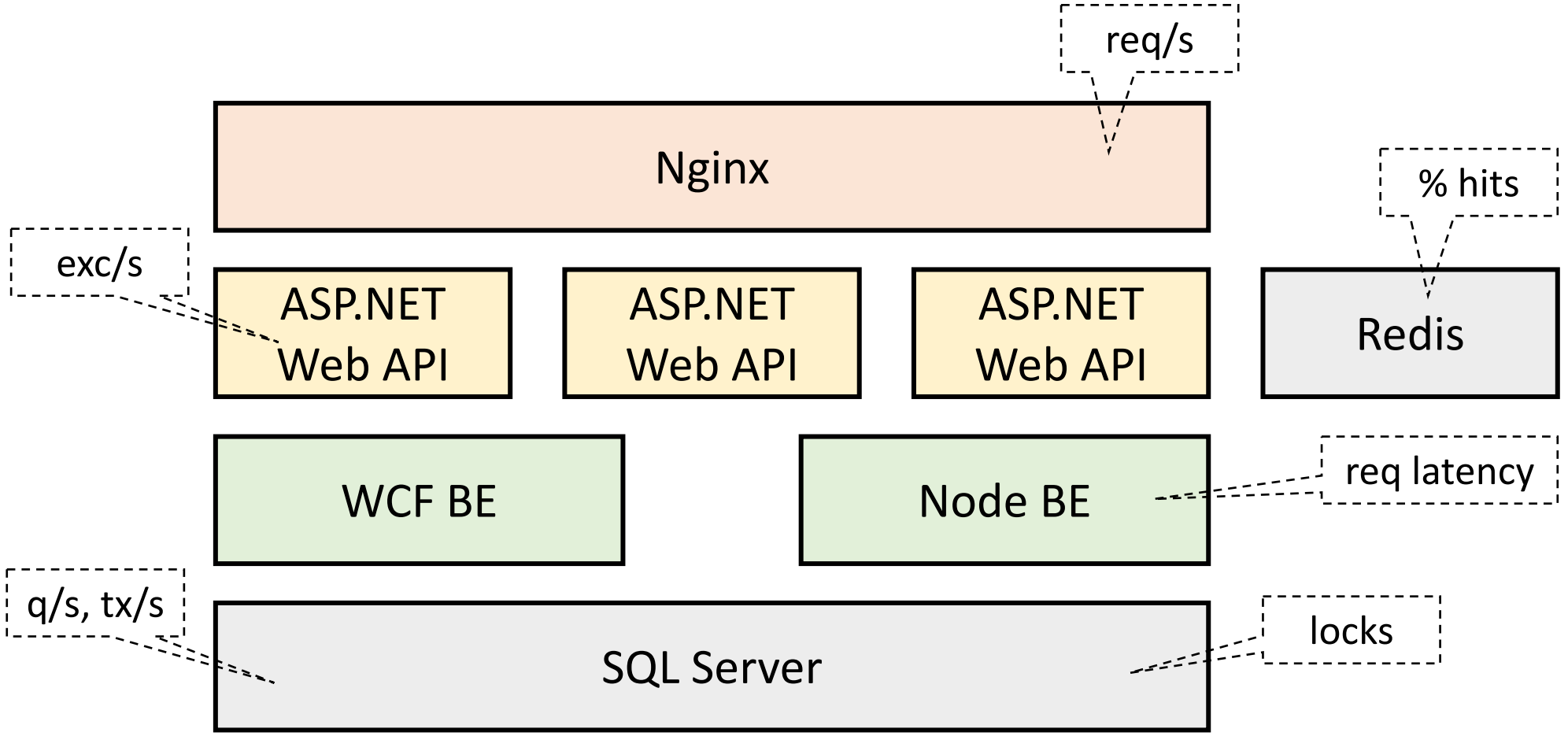
- USE: **U**tilization, **S**aturation, **E**rrors

1. Build a functional diagram of the system, including hardware/software resources
2. For each resource, identify utilization, saturation, and errors
3. Understand, resolve, and verify errors, excessive saturation/utilization, under-utilization

USE For Hardware Resources



USE For Software Resources



USE Checklist For Windows Systems (1/2)

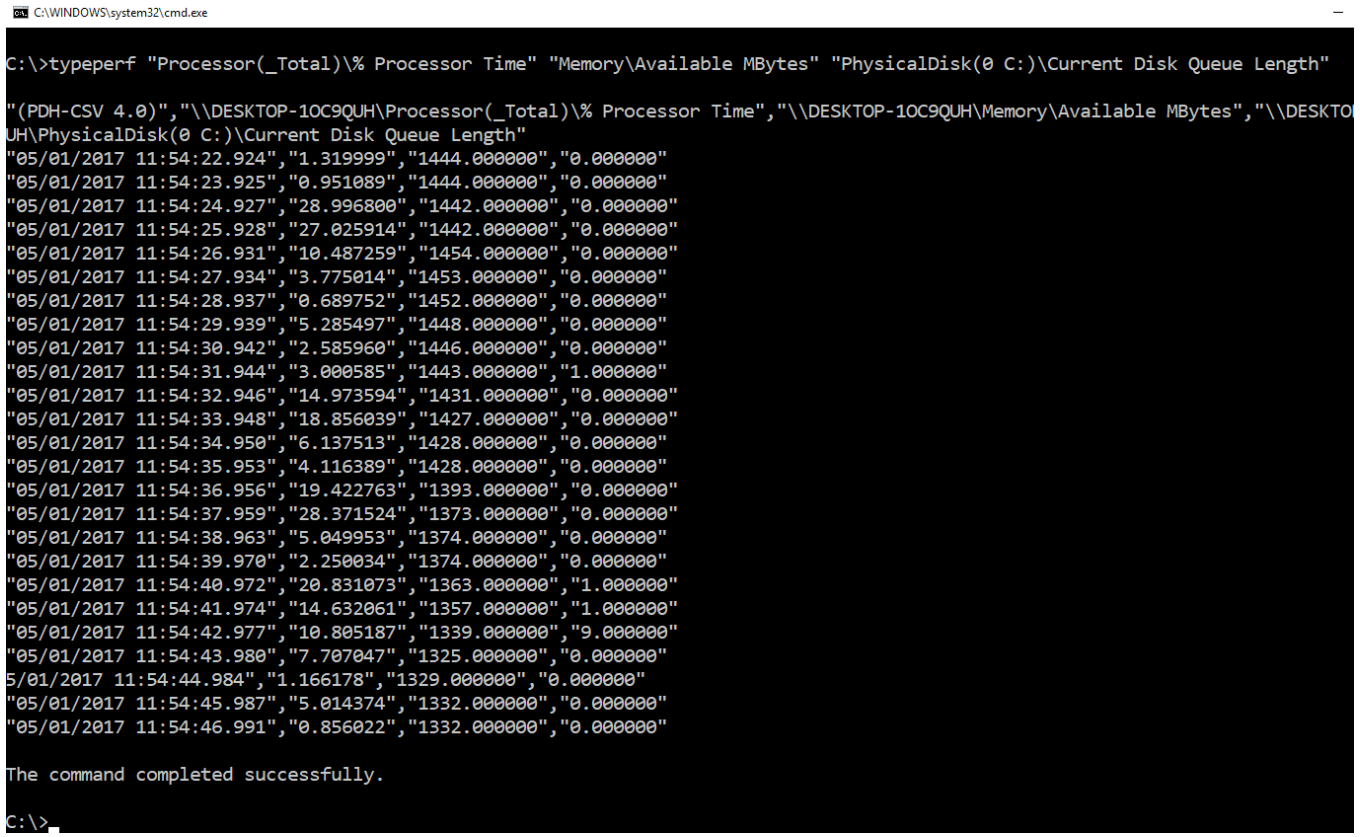
Component	Type	Performance Counter or Other Tool
CPU	Utilization	Processor(_Total)\% Processor Time, % User Time Process(MyApp)\% Processor Time
CPU	Saturation	System\Processor Queue Length
CPU	Errors	Intel Processor Diagnostic Tool (and others)
Memory	Utilization	Memory\Available Mbytes Process\Virtual Size, Private Bytes, Working Set .NET CLR Memory\# Bytes in all Heaps VMMap, RAMMap
Memory	Saturation	Memory\Pages/sec Memory\Committed Bytes vs. Commit Limit
Memory	Errors	Windows Memory Diagnostic Utility (and others)

USE Checklist For Windows Systems (2/2)

Component	Type	Performance Counter/Tool
Network	Utilization	Network Interface\Bytes Received/sec, Bytes Sent/sec
Network	Saturation	Network Interface\Output Queue Length, Packets Outbound Discarded, Packets Received Discarded
Network	Errors	Network Interface\Packets Outbound Errors, Packets Received Errors
Disk	Utilization	Physical Disk\% Disk Time, % Idle Time, Disk Reads/sec, Disk Writes/sec
Disk	Saturation	Physical Disk\Current Disk Queue Length
Disk	Errors	Chkdisk (and other tools)
Application	Errors	.NET CLR Exceptions\# of Exceps Thrown/sec ASP.NET\Error Events Raised ...and many others

Automating The Checklist

- **Typeperf** can generate a simple CSV file with performance counter values (easily scriptable)
- **Perfmon** can collect performance counter logs continuously, or when triggered by an alert
- Plenty of third-party monitoring agents exist



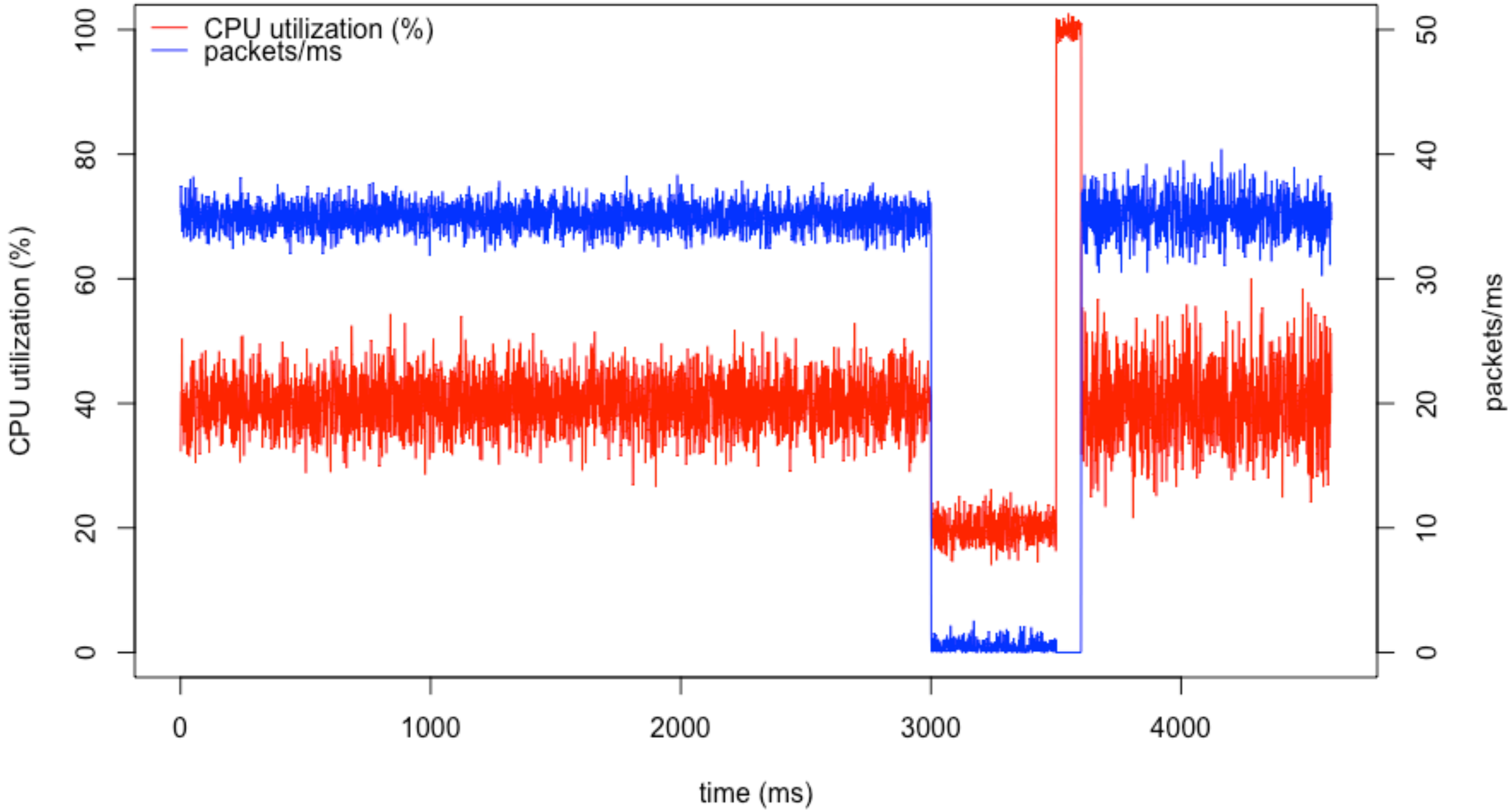
How I Found a Bug in CLR 2.0's GC

- I was asked to help with a soft real-time packet processing system that would occasionally miss thousands of packets
- Logs indicated this only happened when processing a very particular, fairly exotic protocol

```
14:03:44.057 DEBUG Start parsing TCP header seq #1778112
14:03:44.057 DEBUG End parsing TCP header seq #1778112, protocol 0x448
14:03:44.057 DEBUG Start parsing packet protocol 0x448 size 0x90
14:03:44.559 DEBUG Finished parsing packet protocol 0x448
14:03:44.557 DEBUG Start parsing TCP header seq #177811
14:03:44.558 WARN  Receive buffer was full, dropped 11891 packets
…
```

From: Eng McEngineer <engmce@acme.com>
Subject: CPU performance counter logs

We've run the checklist for 24 hours and looked for correlations with the log data on missed packets. It seems that there is a considerable CPU spike towards _the end_ of these windows, take a look at one example:


CPU utilization (%)
packets/ms
CPU utilization (%)
packets/ms
time (ms)
0
20
40
60
80
100
0
10
20
30
40
50
0
1000
2000
3000
4000

From: Eng McEngineer <engmce@acme.com>
Subject: CPU performance counter logs

And here's another funny thing. These delay windows are _always_ close to a multiple of 250ms. I've seen 252ms, 503ms, 751ms, 1005ms, and even 1252ms.

From: Sasha Goldshtein
Subject: Re: CPU performance counter logs

Oh, that looks suspicious! Can you share the GC counters around these times? This looks like a full server GC on 4 cores with full application suspension.

From: Eng McEngineer <engmce@acme.com>
Subject: Re: CPU performance counter logs

Yeah it does correlate with GC, but we have such a tiny heap, it doesn't make sense!
Also, what do you make of the 250ms thing?

From: Sasha Goldshtein
Subject: Re: CPU performance counter logs

> such a tiny heap, it doesn't make sense!

OK, let's grab a stack dump around these windows to see what the GC threads and the application threads are doing.

From: Eng McEngineer <engmce@acme.com>
Subject: Re: CPU performance counter logs

Here goes a stack dump from the start of a delay window:

ParseMultiple 448Packets	SuspendEE	WaitFor GCEvent	WaitFor GCEvent	WaitFor GCEvent	WaitFor GCEvent
ParseProto	GCForAlloc	GCThreadProc	GCThreadProc	GCThreadProc	GCThreadProc
ProcessPkt	JIT_NewArr				
ProcessLoop	ProcessPkt				
Main	ProcessLoop				
	ThreadPool				

From: Sasha Goldshtein
Subject: Re: CPU performance counter logs

Right! So we have one thread in a hot loop parsing packets, and another thread trying to suspend it for GC. What is it doing there in SuspendEE? Is it waiting for something?

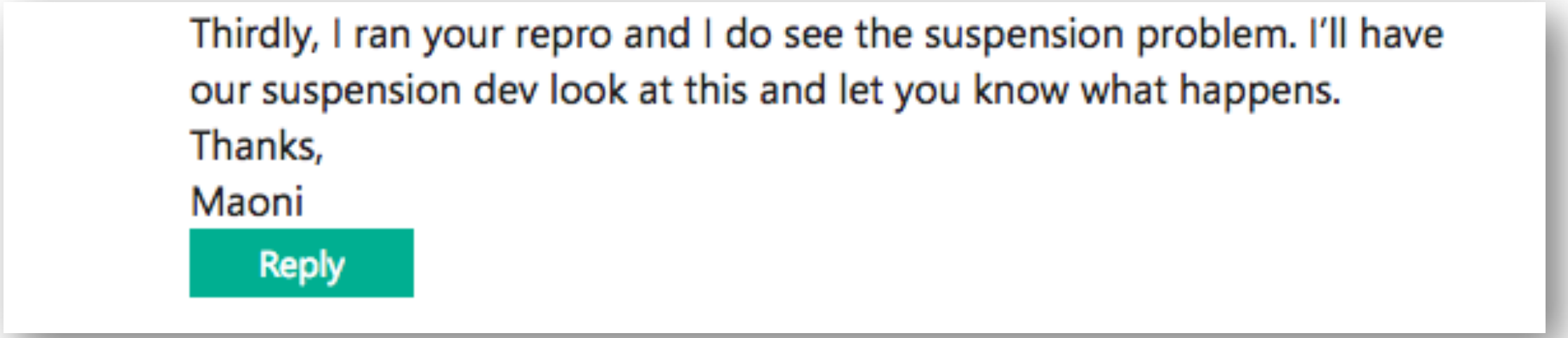
From: Eng McEngineer <engmce@acme.com>
Subject: Re: CPU performance counter logs

Yes! Looks like something like this:

```
while (!GCSuspensionComplete && !gaveUp) {
  WaitForSingleObject(GCSuspensionEvent, 250);
}
if (!GCSuspensionComplete && gaveUp) {
  SuspendThread(targetThread);
}
```

Turned Out, It's a Bug

- CLR 2.0 had a bug around codegen for tight loops with no safepoints, and suspension logic waiting with 250ms intervals



Full story: http://blogs.microsoft.co.il/sasha/2009/07/31/garbage-collection-thread-suspension-delay-250ms-multiples/

Qualities of Good Performance Tools

- Three kinds of tools:
  - How often is this happening? (Counting)
  - How much time is this taking? (Latency)
  - What is causing this thing to happen? (Stacks)

1. Low overhead
2. Accurate
3. Quick turnaround
4. Production-ready (non-invasive)
5. Focusable

⚠ Mind The Overhead

- Any observation can change the state of the system, but some observations are worse than others
  - Check the docs
  - Try on a test system first
  - Measure the degradation introduced by the tool

OVERHEAD
The tool's overhead mostly depends on the number of events traced. For example, CPU sampling with default settings across the system runs at virtually 0% CPU overhead during collection. System call collection on an idle system was observed at 1-2% CPU usage, spiking to 5-10% when heavy I/O processes (issuing approximately 100K system calls per second) were running. […] Additionally, processing symbols to display call stacks can lead to spikes of CPU, disk, and network activity as symbols are downloaded, loaded from disk, parsed in memory, and then cached. During heavy symbol processing, LiveStacks will routinely exhibit 100% CPU utilization for short time periods. To address this, you can limit the number of stacks displayed with the -T switch.

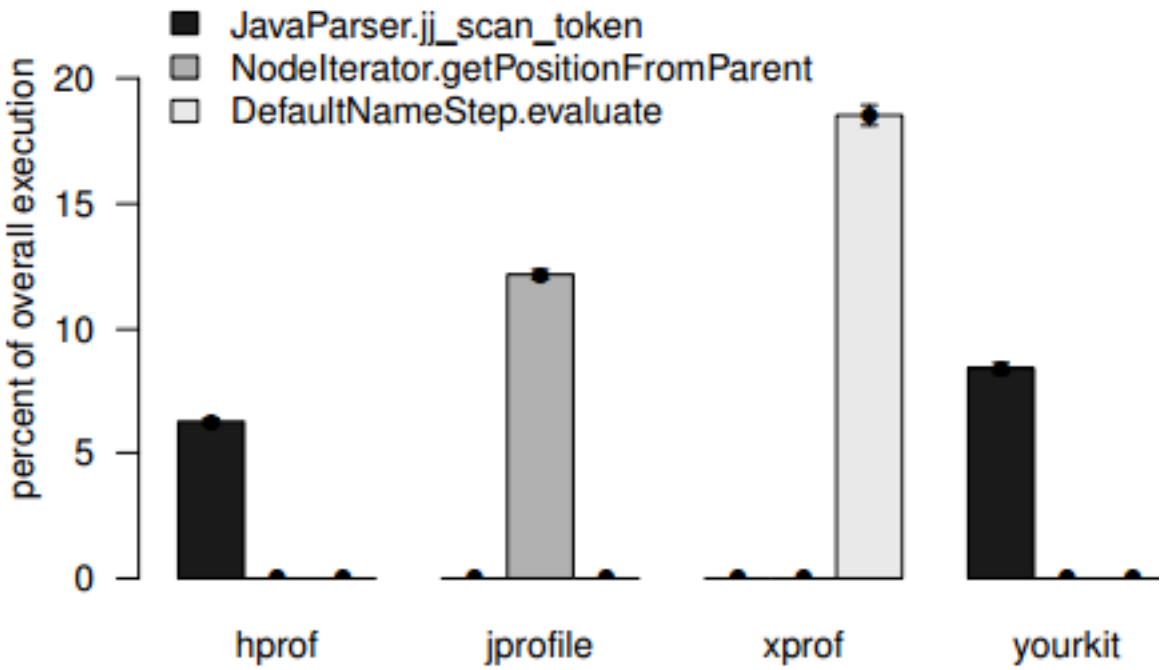
— *From LiveStacks README.md*

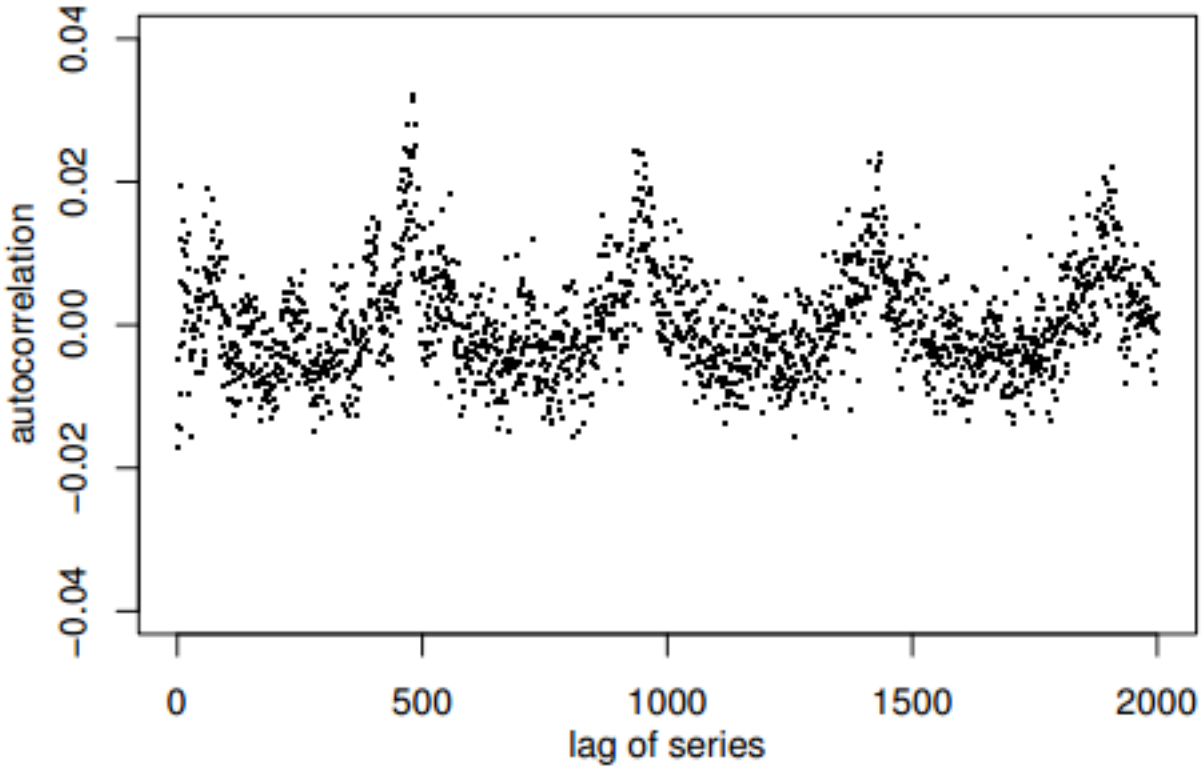
Accuracy: The Safepoints Story

- Most Java profilers lie (VisualVM, jstack, YourKit, JProfiler, …)
- Most Java profilers use a documented JVMTI API called `GetAllStackTraces`, which gets a stack sample of all the threads
- But: the stack sample is captured *only at a safepoint*
- Result: you get totally bogus stacks with a strong bias towards safepoints, and no idea about the rest of the code
- The same problem applies to .NET Core profilers on Linux that use signals and the CLR stack walking API

Evaluating The Accuracy of Java Profilers

Mytkowicz, Diwan, Hauswirth, Sweeney





Appendix A: Sanity Test Your Profiler

1. Build an artificial benchmark, 50% CPU bound, 50% sleep or I/O, with deep stacks
2. Run CPU profiler and verify that the results make sense
3. While you are at it, look at the overhead

Quick Turnaround

```
# ***** HOW TO USE THIS SCRIPT *****
## This script can be used to collect and view performance data collected with perf_event on Linux.
# Its job is to make it simple to collect performance traces.
## How to collect a performance trace:
# 1. Prior to starting the .NET process, set the environment variable COMPlus_PerfMapEnabled=1.
#    This tells the runtime to emit information that enables perf_event to resolve JIT-compiled code symbols.
# 2. Setup your system to reproduce the performance issue you'd like to capture.  Data collection can be
#    started on already running processes.
# 2. Run this script: sudo ./perfcollect collect samplePerfTrace
#    This will start data collection.
# 3. Let the repro run as long as you need to capture the performance problem.
# 4. Hit CTRL+C to stop collection.
#    When collection is stopped, the script will create a trace.zip file matching the name specified on the
#    command line.  This file will contain the trace, JIT-compiled symbol information, and all debugging
#    symbols for binaries referenced by this trace that were available on the machine at collection time.
```

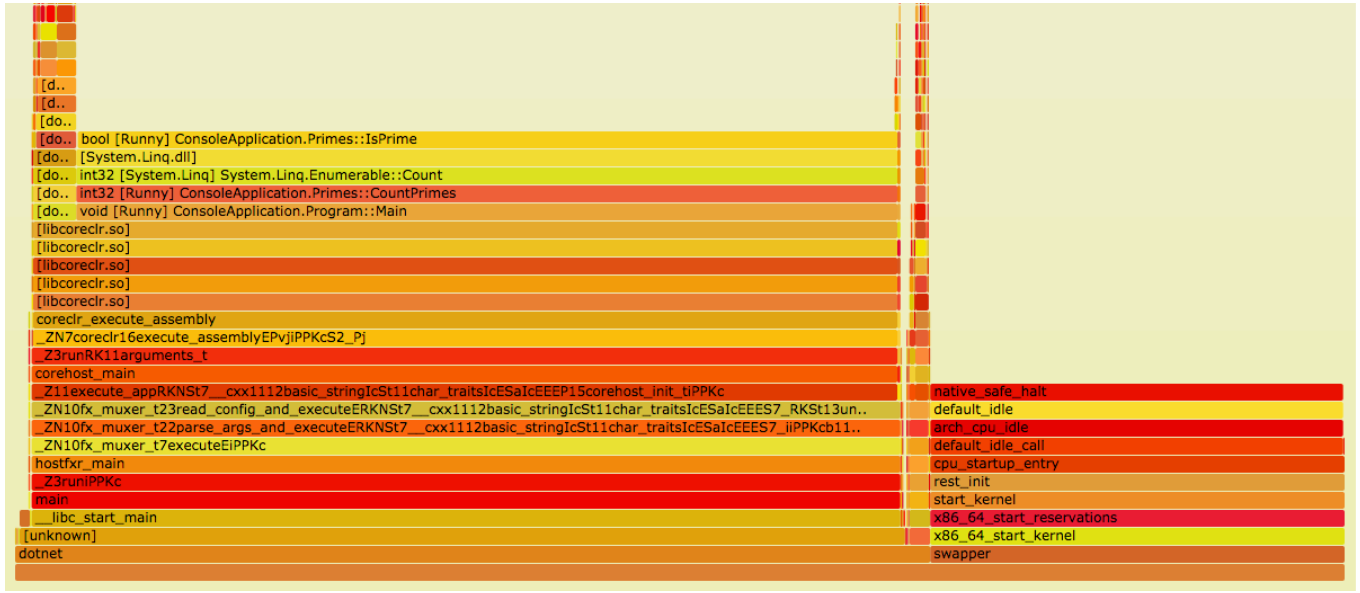
Open the Trace File

1. **Copy the trace.zip file from Linux to a Windows machine.**
2. Download PerfView from http://aka.ms/perfview.
3. Run PerfView.exe

https://github.com/dotnet/corefx-tools/blob/master/src/performance/perfcollect/perfcollect
https://github.com/dotnet/coreclr/blob/master/Documentation/project-docs/linux-performance-tracing.md

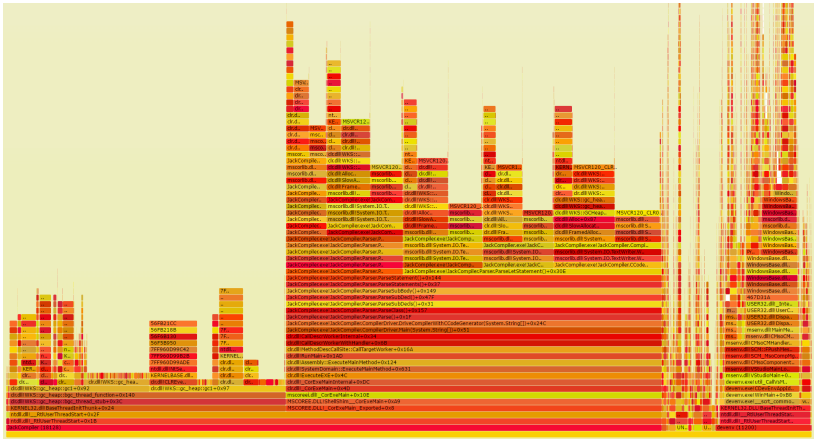
Quick Turnaround

```
$ ./dotnet-mapgen.py generate 4118
$ ./dotnet-mapgen.py merge 4118
# perf record -p 4118 -F 97 -g
# perf script | ./stackcollapse-perf.pl > stacks
$ ./flamegraph.pl stacks > stacks.svg
```



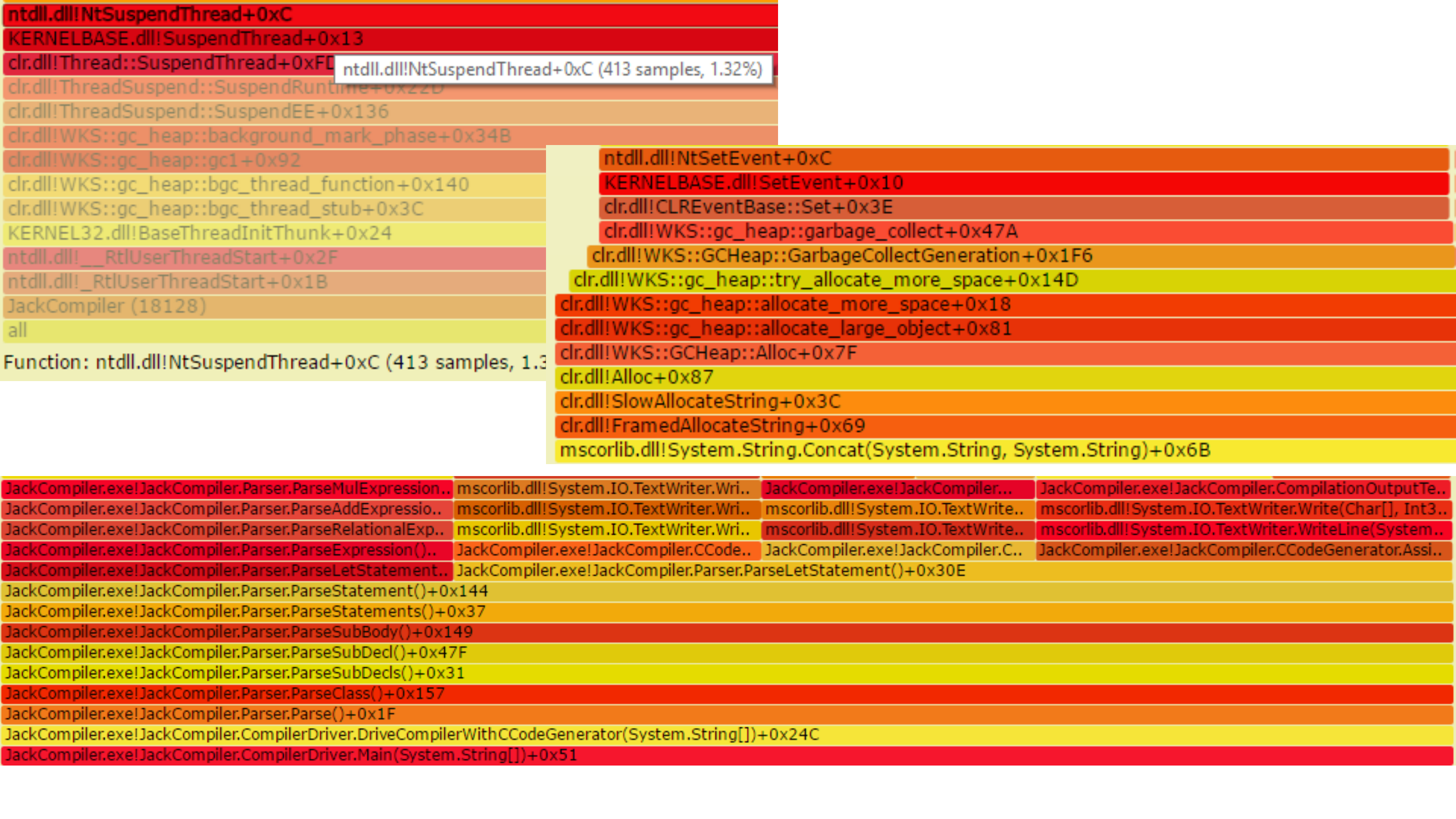
Flame Graphs

- A visualization method (adjacency graph), very useful for stack traces, invented by Brendan Gregg
  - http://www.brendangregg.com/flamegraphs.html
- Turns thousands of stack trace pages into a single interactive graph
- Example scenarios:
  - Identify CPU hotspots on the system/application
  - Show stacks that perform heavy disk accesses
  - Find threads that block for a long time and the stack where they do it



Reading a Flame Graph

- Each rectangle is a function
- Y-axis: stack depth
- X-axis: sorted stacks (not time)
- Wider frames are more common
- Supports zoom, find
- Filter with `grep` 😎

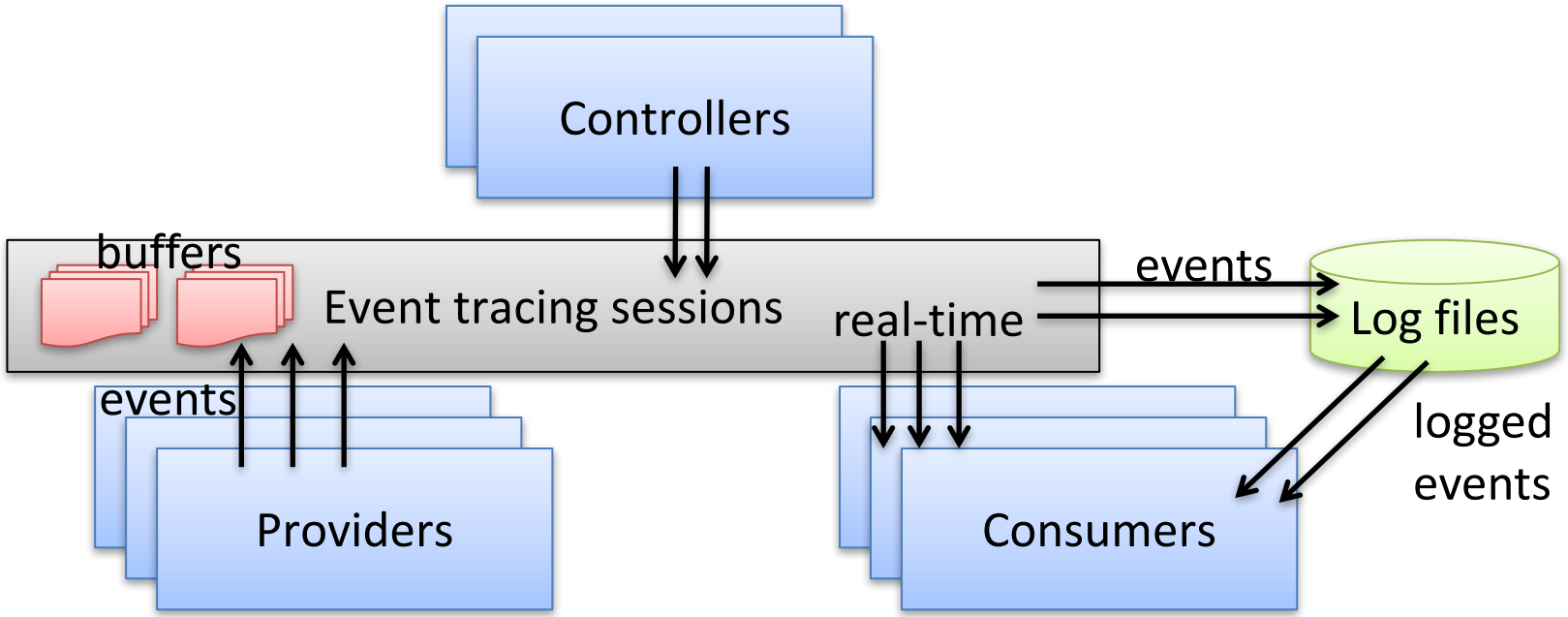

ntdll.dll!NtSuspendThread+0xC
KERNELBASE.dll!SuspendThread+0x13
clr.dll!Thread::SuspendThread+0xFD
ntdll.dll!NtSuspendThread+0xC (413 samples, 1.32%)
clr.dll!ThreadSuspend::SuspendRuntime+0x22D
clr.dll!ThreadSuspend::SuspendEE+0x136
clr.dll!WKS::gc_heap::background_mark_phase+0x34B
clr.dll!WKS::gc_heap::gc1+0x92
clr.dll!WKS::gc_heap::bgc_thread_function+0x140
clr.dll!WKS::gc_heap::bgc_thread_stub+0x3C
KERNEL32.dll!BaseThreadInitThunk+0x24
ntdll.dll!__RtlUserThreadStart+0x2F
ntdll.dll!_RtlUserThreadStart+0x1B
JackCompiler (18128)
all
Function: ntdll.dll!NtSuspendThread+0xC (413 samples, 1.3
ntdll.dll!NtSetEvent+0xC
KERNELBASE.dll!SetEvent+0x10
clr.dll!CLREventBase::Set+0x3E
clr.dll!WKS::gc_heap::garbage_collect+0x47A
clr.dll!WKS::GCHeap::GarbageCollectGeneration+0x1F6
clr.dll!WKS::gc_heap::try_allocate_more_space+0x14D
clr.dll!WKS::gc_heap::allocate_more_space+0x18
clr.dll!WKS::gc_heap::allocate_large_object+0x81
clr.dll!WKS::GCHeap::Alloc+0x7F
clr.dll!Alloc+0x87
clr.dll!SlowAllocateString+0x3C
clr.dll!FramedAllocateString+0x69
mscorlib.dll!System.String.Concat(System.String, System.String)+0x6B
JackCompiler.exe!JackCompiler.Parser.ParseMulExpression..
mscorlib.dll!System.IO.TextWriter.Wri..
JackCompiler.exe!JackCompiler...
JackCompiler.exe!JackCompiler.CompilationOutputTe..
JackCompiler.exe!JackCompiler.Parser.ParseAddExpressio..
mscorlib.dll!System.IO.TextWriter.Wri..
mscorlib.dll!System.IO.TextWrite..
mscorlib.dll!System.IO.TextWriter.Write(Char[], Int3..
JackCompiler.exe!JackCompiler.Parser.ParseRelationalExp..
mscorlib.dll!System.IO.TextWriter.Wri..
mscorlib.dll!System.IO.TextWrite..
mscorlib.dll!System.IO.TextWriter.WriteLine(System..
JackCompiler.exe!JackCompiler.Parser.ParseExpression()..
JackCompiler.exe!JackCompiler.CCode..
JackCompiler.exe!JackCompiler.C..
JackCompiler.exe!JackCompiler.CCodeGenerator.Assi..
JackCompiler.exe!JackCompiler.Parser.ParseLetStatement..
JackCompiler.exe!JackCompiler.Parser.ParseLetStatement()+0x30E
JackCompiler.exe!JackCompiler.Parser.ParseStatement()+0x144
JackCompiler.exe!JackCompiler.Parser.ParseStatements()+0x37
JackCompiler.exe!JackCompiler.Parser.ParseSubBody()+0x149
JackCompiler.exe!JackCompiler.Parser.ParseSubDecl()+0x47F
JackCompiler.exe!JackCompiler.Parser.ParseSubDecls()+0x31
JackCompiler.exe!JackCompiler.Parser.ParseClass()+0x157
JackCompiler.exe!JackCompiler.Parser.Parse()+0x1F
JackCompiler.exe!JackCompiler.CompilerDriver.DriveCompilerWithCCodeGenerator(System.String[])+0x24C
JackCompiler.exe!JackCompiler.CompilerDriver.Main(System.String[])+0x51

Invasiveness

- Invasive tools are bad for performance, reliability, responsiveness
- Visual Studio instrumenting profiler and IntelliTrace require an application restart and attach code to every method enter/exit
- The CLR Profiling API is based on injecting a DLL into the profiled application
- Extreme example: Linux SystemTap, LTTng, SysDig tracing frameworks requires loading a *custom kernel module*

Event Tracing For Windows

- High-performance facility for emitting 100K+ log events per second with rich payloads and stack trace support
- File accesses, image loads, GC events, JIT, interop, allocs, threads, …



Quick ETW Collection Turnaround With PerfView (From Log)

```
[Starting collection at 5/1/2017 12:36:15 PM]
[Collecting   10 sec: Size= 15.6 MB.]
[Manually Stopped (Gui)]
Stopping tracing for sessions 'NT Kernel Logger' and 'PerfViewSession'.
[Sending rundown command to CLR providers...]
...
CLR Rundown took 12.744 sec.
Heap events were active for this trace.
Insuring .NET Allocation profiler not installed.
Stop Completed at 5/1/2017 12:36:43 PM
[Conversion complete 91,988 events.  Conversion took 4 sec.]
...
Completed: Opening PerfViewData.etl (unmerged)   (Elapsed Time: 4.311 sec)
Started: Opening PerfViewData.etl (unmerged)
Completed: Opening PerfViewData.etl (unmerged)   (Elapsed Time: 0.209 sec)
```

Announcing: **etrace**

- **etrace**: a real-time, command-line frontend for ETW events
  https://github.com/goldshtn/etrace

```
> etrace --help

...
Examples:
  etrace --clr GC --event GC/AllocationTick
  etrace --kernel Process,Thread,FileIO,FileIOInit --event File/Create
  etrace --file trace.etl --stats
  etrace --clr GC --event GC/Start --field PID,TID,Reason[12],Type
  etrace --kernel Process --event Process/Start --where ImageFileName=myapp
  etrace --clr GC --event GC/Start --duration 60
  etrace --other Microsoft-Windows-Win32k --event QueuePostMessage
  etrace --list CLR,Kernel
```

Administrator: C:\WINDOWS\SYSTEM32\cmd.exe

```
C:\Dev\etrace\etrace\bin\Release>etrace --clr GC --event GC/Start --field PID,TID,Reason[12],Type
Processing start time: 5/1/2017 12:46:37 PM
PID   TID   Reason       Type
-----------------------------------------------------
11200 12860 AllocSmall   NonConcurrentGC
11200 12860 AllocSmall   NonConcurrentGC
10056 11580 AllocSmall   NonConcurrentGC
11200 12860 AllocSmall   NonConcurrentGC

Processing end time:           5/1/2017 12:46:43 PM
Processing duration:           00:00:05.4203371
Processed events:              2174
Displayed events:              4
Events lost:                   0
^C
C:\Dev\etrace\etrace\bin\Release>etrace --clr GC --event GC/AllocationTick --field PID,AllocationAmount,TypeName
Processing start time: 5/1/2017 12:46:48 PM
PID   AllocationAmount              TypeName
-----------------------------------------------------------------
31292 108936                       Microsoft.Diagnostics.Tracing.Parsers.Clr.AuthenticodeVeri...
11200 106888                       System.Threading.ExecutionContext
11200 106928                       System.Boolean[]
11200 106836                       MS.Internal.Text.LineProperties
11200 107012                       HashtableEnumerator
11200 106212                       MS.Internal.Span[]
11200 106812                       System.Windows.Documents.TextPointer
11200 106824                       System.Windows.Documents.RangeContentEnumerator
11200 107456                       ContingentProperties
11200 107164                       MS.Internal.Text.TextInterface.Generics.NativeIUnknownWrap...
11200 106884                       MS.Utility.ArrayItemList`1[MS.Internal.Span]
11200 107012                       System.Byte[]
```

Announcing: **LiveStacks**

- **LiveStacks**: a real-time, command-line stack collector and resolver
  https://github.com/goldshtn/LiveStacks

```
> LiveStacks --help

...
Examples:
  LiveStacks
  LiveStacks -p 7408
  LiveStacks -e clr:gc:gc/triggered
  LiveStacks -e kernel:imageload -i 1 -T 5
  LiveStacks -c 1 -f
```

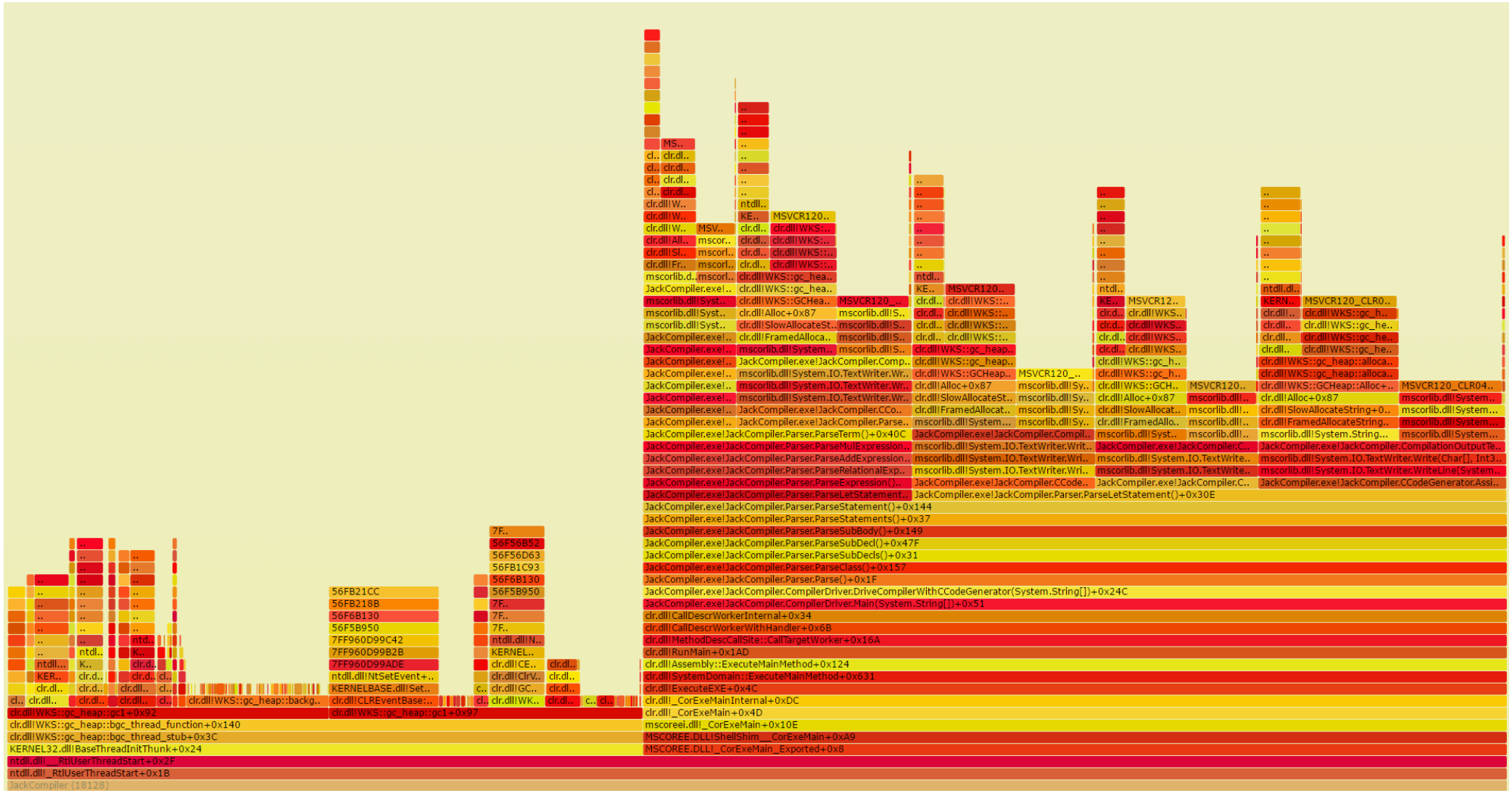
Administrator: C:\WINDOWS\SYSTEM32\cmd.exe

```
C:\Dev\LiveStacks\LiveStacks\bin\x86\Release>livestacks -P devenv -T 1
Ctrl+C pressed, stopping...
12:55:36 PM
         34 [devenv 11200]
           56FB21CC
           56FB218B
           56F6B130
           56F5B950
       7FF960DB2B55
       7FF960DEA1FC
       7FF960D99B2B
       7FF960D99ADE
    ntdll.dll!NtSetEvent+0xC
    KERNELBASE.dll!SetEvent+0x10
    wpfgfx_v0400.dll!CPartitionManager::ScheduleBatchProcessing+0x5F
    wpfgfx_v0400.dll!CCrossThreadComposition::SubmitBatch+0x34
    wpfgfx_v0400.dll!CMilServerChannel::SubmitBatch+0x27
    wpfgfx_v0400.dll!CConnectionContext::SendBatchToChannel+0x4A
    wpfgfx_v0400.dll!CMilConnection::SubmitBatch+0x13
    wpfgfx_v0400.dll!CMilChannel::Commit+0x40
    wpfgfx_v0400.dll!MilChannel_CommitChannel+0x1C
    PresentationCore.dll!System.Windows.Media.Composition.DUCE+Channel.Commit()+0x38
    PresentationCore.dll!System.Windows.Media.MediaContext.CommitChannel()+0x175
    PresentationCore.dll!System.Windows.Media.MediaContext.EstimatedNextVSyncTimeExpired(System.Object, System.EventArgs)+0xF3
    WindowsBase.dll!System.Windows.Threading.DispatcherTimer.FireTick(System.Object)+0x34
    WindowsBase.dll!System.Windows.Threading.ExceptionWrapper.InternalRealCall(System.Delegate, System.Object, Int32)+0x52
    WindowsBase.dll!System.Windows.Threading.ExceptionWrapper.TryCatchWhen(System.Object, System.Delegate, System.Object, Int32, System.D
elegate)+0x34
    WindowsBase.dll!System.Windows.Threading.DispatcherOperation.InvokeImpl()+0xD2
    WindowsBase.dll!System.Windows.Threading.DispatcherOperation.InvokeInSecurityContext(System.Object)+0x3C
    mscorlib.dll!System.Threading.ExecutionContext.Run(System.Threading.ExecutionContext, System.Threading.ContextCallback, System.Object
, Boolean)+0x16
    mscorlib.dll!System.Threading.ExecutionContext.Run(System.Threading.ExecutionContext, System.Threading.ContextCallback, System.Object
```

Administrator: C:\WINDOWS\SYSTEM32\cmd.exe

```
C:\Dev\LiveStacks\LiveStacks\bin\x86\Release>livestacks -T 1 -e clr:gc:gc/triggered
Ctrl+C pressed, stopping...
12:57:20 PM
          1 [Microsoft.Alm.Shared.Remoting.RemoteContainer.dll 10608]
    clr.dll!EtwCallout+0x12E
    clr.dll!CoTemplate_qh+0x6F
    clr.dll!WKS::GCHeap::GarbageCollectGeneration+0x1E5
    clr.dll!WKS::gc_heap::try_allocate_more_space+0x14D
    clr.dll!WKS::gc_heap::allocate_more_space+0x18
    clr.dll!WKS::GCHeap::Alloc+0x5C
    clr.dll!Alloc+0x87
    clr.dll!AllocateObject+0x99
    clr.dll!JIT_New+0x6B
    Microsoft.CodeAnalysis.Workspaces.dll!Microsoft.CodeAnalysis.Solution.GetProject(Microsoft.CodeAnalysis.IAssemblySymbol, System.Threa
ding.CancellationToken)+0x41
    Microsoft.CodeAnalysis.Workspaces.dll!Microsoft.CodeAnalysis.FindSymbols.FindReferencesSearchEngine+<>c__DisplayClass25_0+<<Determine
AllSymbolsCoreAsync>b__0>d.MoveNext()+0x17D
    mscorlib.dll!System.Runtime.CompilerServices.AsyncTaskMethodBuilder.Start[[Microsoft.CodeAnalysis.FindSymbols.FindReferencesSearchEng
ine+<>c__DisplayClass25_0+<<DetermineAllSymbolsCoreAsync>b__0>d, Microsoft.CodeAnalysis.Workspaces]](<<DetermineAllSymbolsCoreAsync>b__0>
d ByRef)+0x43
    Microsoft.CodeAnalysis.Workspaces.dll!Microsoft.CodeAnalysis.FindSymbols.FindReferencesSearchEngine+<>c__DisplayClass25_0.<DetermineA
llSymbolsCoreAsync>b__0()+0x6A
    mscorlib.dll!System.Threading.Tasks.Task`1[[System.__Canon, mscorlib]].InnerInvoke()+0x7A
    mscorlib.dll!System.Threading.Tasks.Task.Execute()+0x30
    mscorlib.dll!System.Threading.Tasks.Task.ExecutionContextCallback(System.Object)+0x1A
    mscorlib.dll!System.Threading.ExecutionContext.Run(System.Threading.ExecutionContext, System.Threading.ContextCallback, System.Object
, Boolean)+0x16
    mscorlib.dll!System.Threading.Tasks.Task.ExecuteWithThreadLocal(System.Threading.Tasks.Task ByRef)+0xD8
    mscorlib.dll!System.Threading.Tasks.Task.ExecuteEntry(Boolean)+0x5F
    mscorlib.dll!System.Threading.Tasks.Task.System.Threading.IThreadPoolWorkItem.ExecuteWorkItem()+0xC
    mscorlib.dll!System.Threading._ThreadPoolWaitCallback.PerformWaitCallback()+0xA
    clr.dll!CallDescrWorkerWithHandler+0x6B
    clr.dll!MethodDescCallSite::CallTargetWorker+0x16A
```

```
> LiveStacks -P JackCompiler -f > stacks.txt
^C
> perl flamegraph.pl stacks.txt > stacks.svg
```

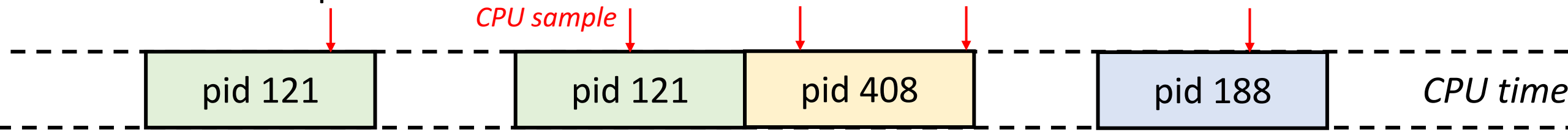


Designing Systems For Instrumentation

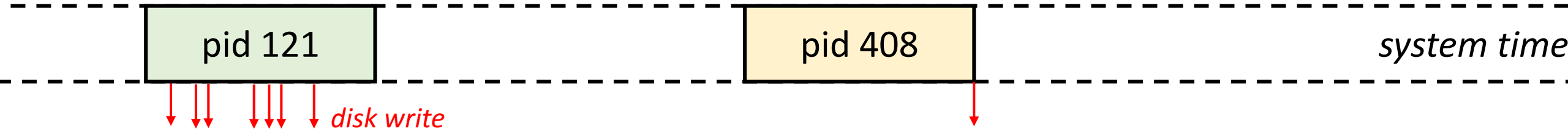
1. Make it easy to get call stacks for samples and events
2. Embed static instrumentation (tracepoints)
3. Ensure important events are easy to turn on and have a low overhead
4. Account for dynamic instrumentation (probes)
5. Have examples and documentation on file

Sampling vs. Tracing

- **Sampling** works by getting a snapshot or a call stack every N occurrences of an interesting event
  - For most events, implemented in the PMU using overflow counters and interrupts

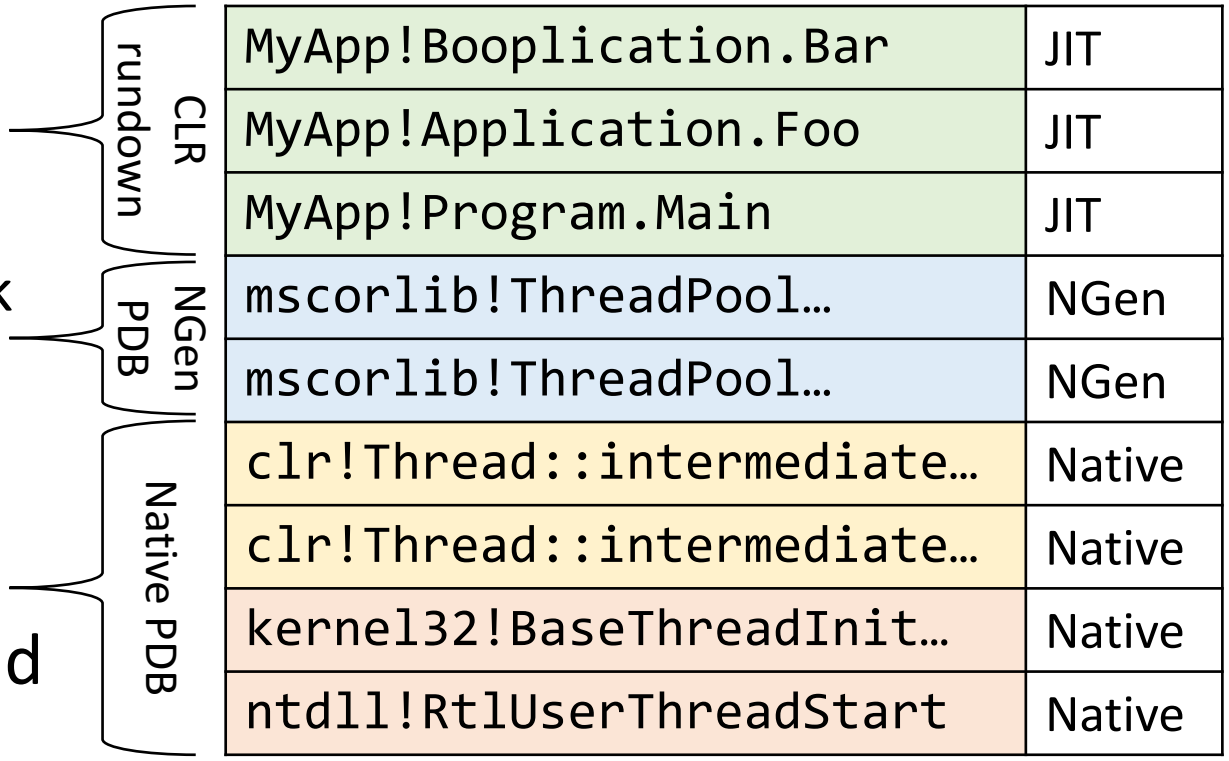


- **Tracing** works by getting a message or a call stack at every occurrence of an interesting event



Well-Thought-Out: Full .NET on Windows

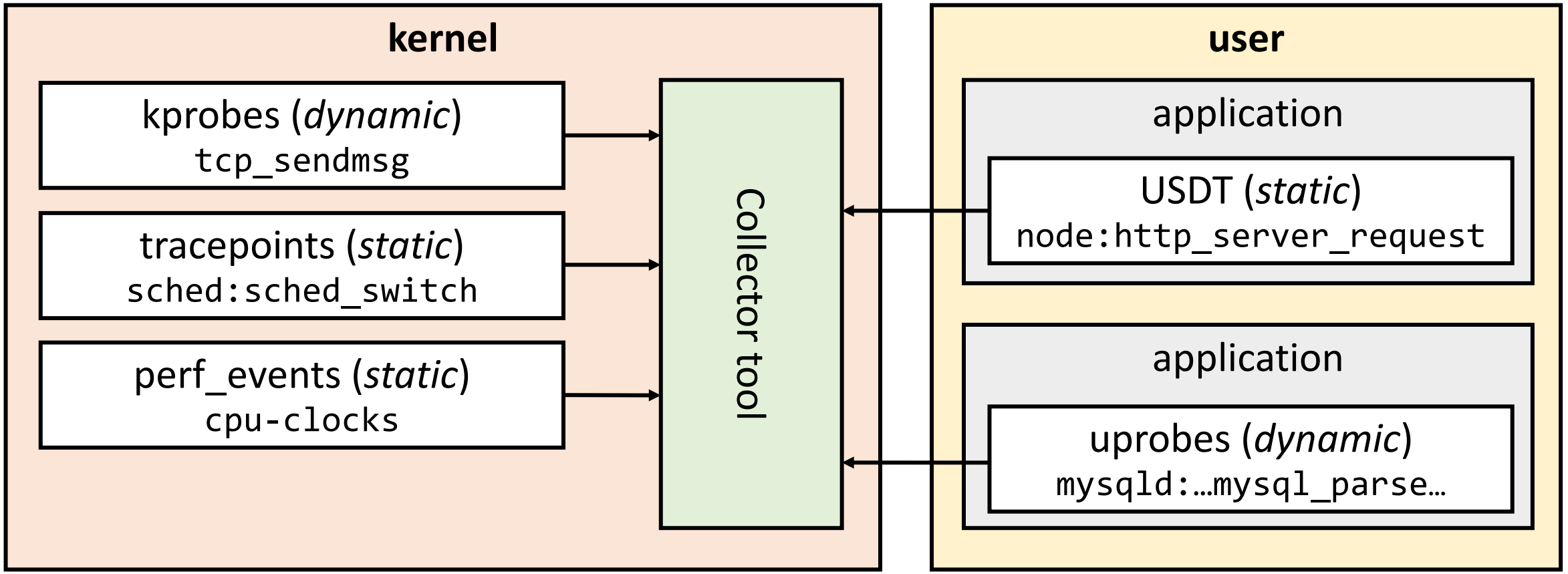
- The CLR has dozens of ETW events: GC, JIT, exceptions, interop, assembly loading, threads, …
  - Events are associated with a call stack
- Arbitrary CPU sampling is possible with ETW (kernel mechanism)
- Integrated experience in PerfView, WPA, VS Concurrency Visualizer, and other tools
- **On by default**



Not So Good: .NET Core 1.0 on Linux

- CoreCLR produces events through LTTng
  - Off by default, need environment variable to turn on
  - No stack support
- Arbitrary CPU sampling possible with perf_events (kernel mechanism)
- To resolve symbols:
  - Maps for JITted code are emitted if started with an environment variable
  - Maps for NGen code are emitted using **crossgen** tool (not in-box), and can't be parsed by any standard Linux performance tools
  - CoreCLR native symbols not always available if not building from source

Even Better: Linux Tracepoints And Probes

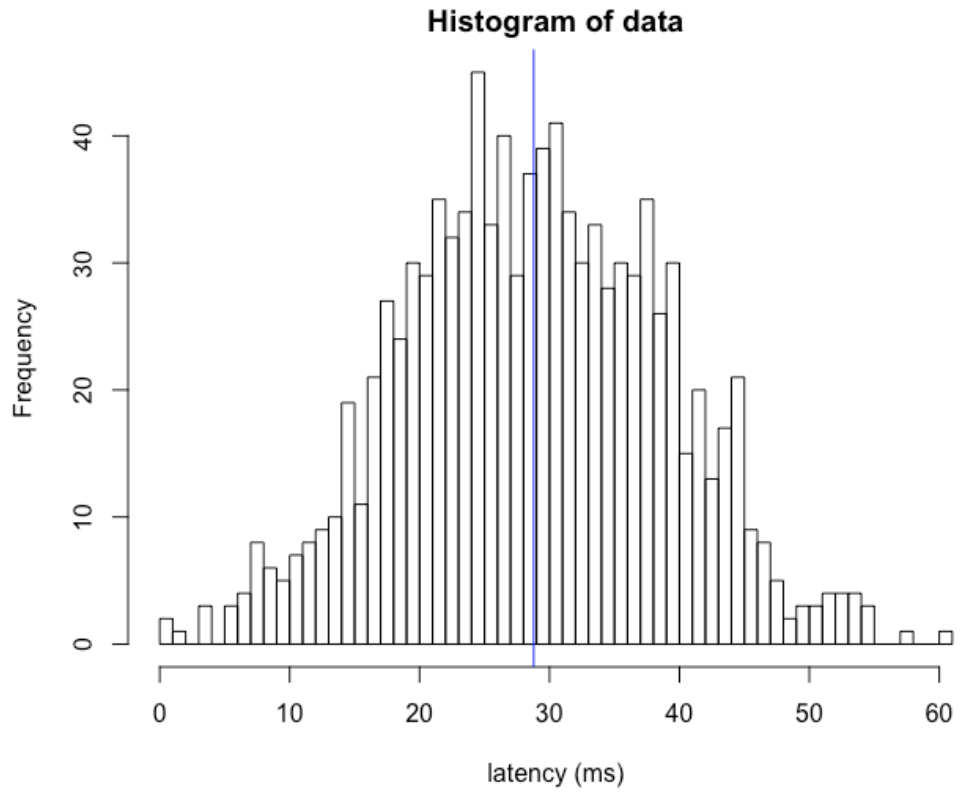


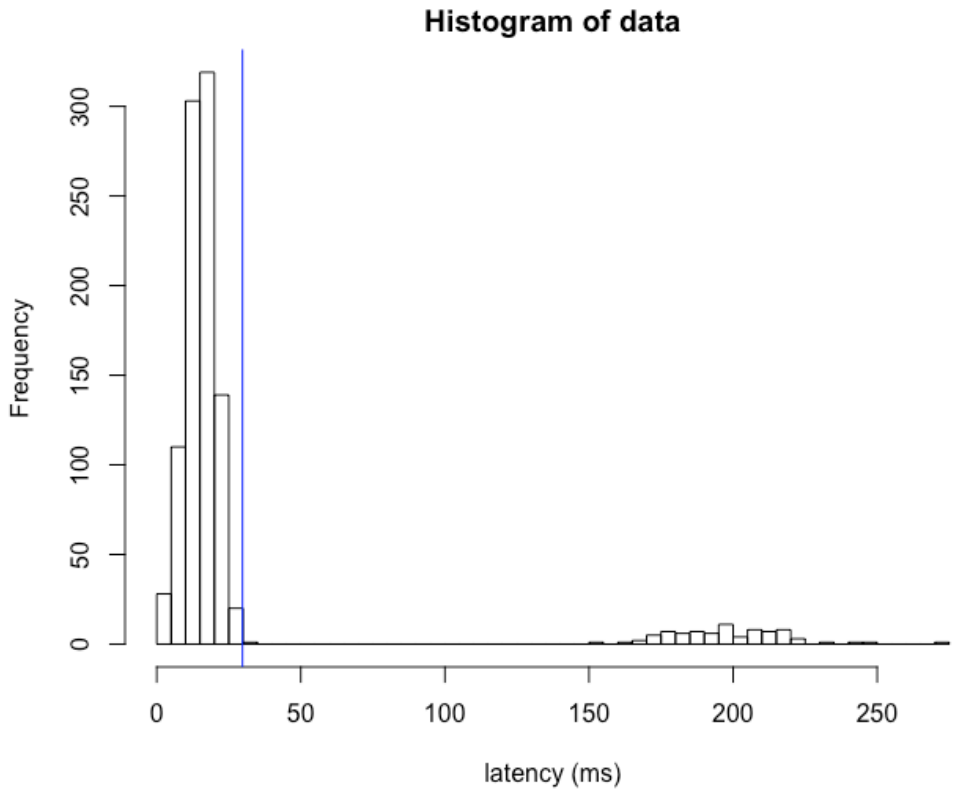
Be Careful With Statistics

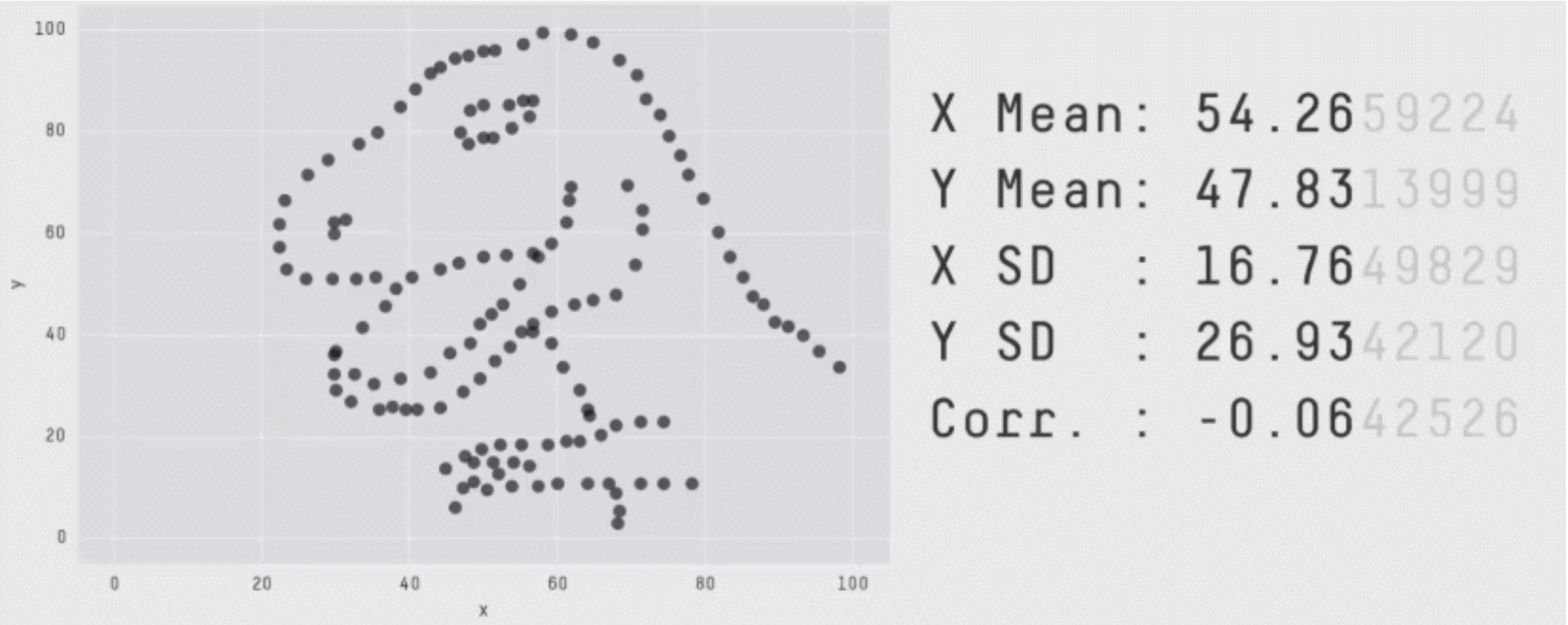
- Averages are meaningless
- Medians are almost meaningless
- Percentiles are OK if you know what you're doing
- Find good *visualizations* for your performance data
- Beware coordinated omission

Statistics Lie

- “The average response time is 29ms”
- Is this good? Does it mean anything?







Same Stats, Different Graphs: Generating Datasets with Varied Appearance and Identical Statistics through Simulated Annealing, Matejka and Fitzmaurice

C:\WINDOWS\system32\cmd.exe

```
// * Detailed results *
MyAPI.Implementation1: DefaultJob
Runtime = Clr 4.0.30319.42000, 32bit LegacyJIT-v4.7.2046.0; GC = Concurrent Workstation
Mean = 294.9818 ns, StdErr = 0.6124 ns (0.21%); N = 14, StdDev = 2.2915 ns
Min = 290.5997 ns, Q1 = 293.9090 ns, Median = 294.6716 ns, Q3 = 297.3666 ns, Max = 298.3551 ns
IQR = 3.4577 ns, LowerFence = 288.7224 ns, UpperFence = 302.5532 ns
ConfidenceInterval = [292.3968 ns; 297.5667 ns] (CI 99.9%), Margin = 2.5849 ns (0.88% of Mean)
Skewness = -0.12, Kurtosis = 2.04


MyAPI.Implementation2: DefaultJob
Runtime = Clr 4.0.30319.42000, 32bit LegacyJIT-v4.7.2046.0; GC = Concurrent Workstation
Mean = 1.7450 ms, StdErr = 0.0082 ms (0.47%); N = 15, StdDev = 0.0318 ms
Min = 1.6984 ms, Q1 = 1.7149 ms, Median = 1.7462 ms, Q3 = 1.7698 ms, Max = 1.7962 ms
IQR = 0.0549 ms, LowerFence = 1.6326 ms, UpperFence = 1.8521 ms
ConfidenceInterval = [1.7111 ms; 1.7790 ms] (CI 99.9%), Margin = 0.0340 ms (1.95% of Mean)
Skewness = -0.03, Kurtosis = 1.53


Total time: 00:00:40 (40.97 sec)

// * Summary *

BenchmarkDotNet=v0.10.5, OS=Windows 10.0.15063
Processor=Intel Core i7-4980HQ CPU 2.80GHz (Haswell), ProcessorCount=4
Frequency=2728060 Hz, Resolution=366.5609 ns, Timer=TSC
  [Host]     : Clr 4.0.30319.42000, 32bit LegacyJIT-v4.7.2046.0
  DefaultJob : Clr 4.0.30319.42000, 32bit LegacyJIT-v4.7.2046.0


         Method |           Mean |         Error |        StdDev |
--------------- |---------------:|--------------:|--------------:|
 Implementation1 |       295.0 ns |      2.585 ns |      2.291 ns |
 Implementation2 | 1,745,023.6 ns | 33,951.808 ns | 31,758.539 ns |
```

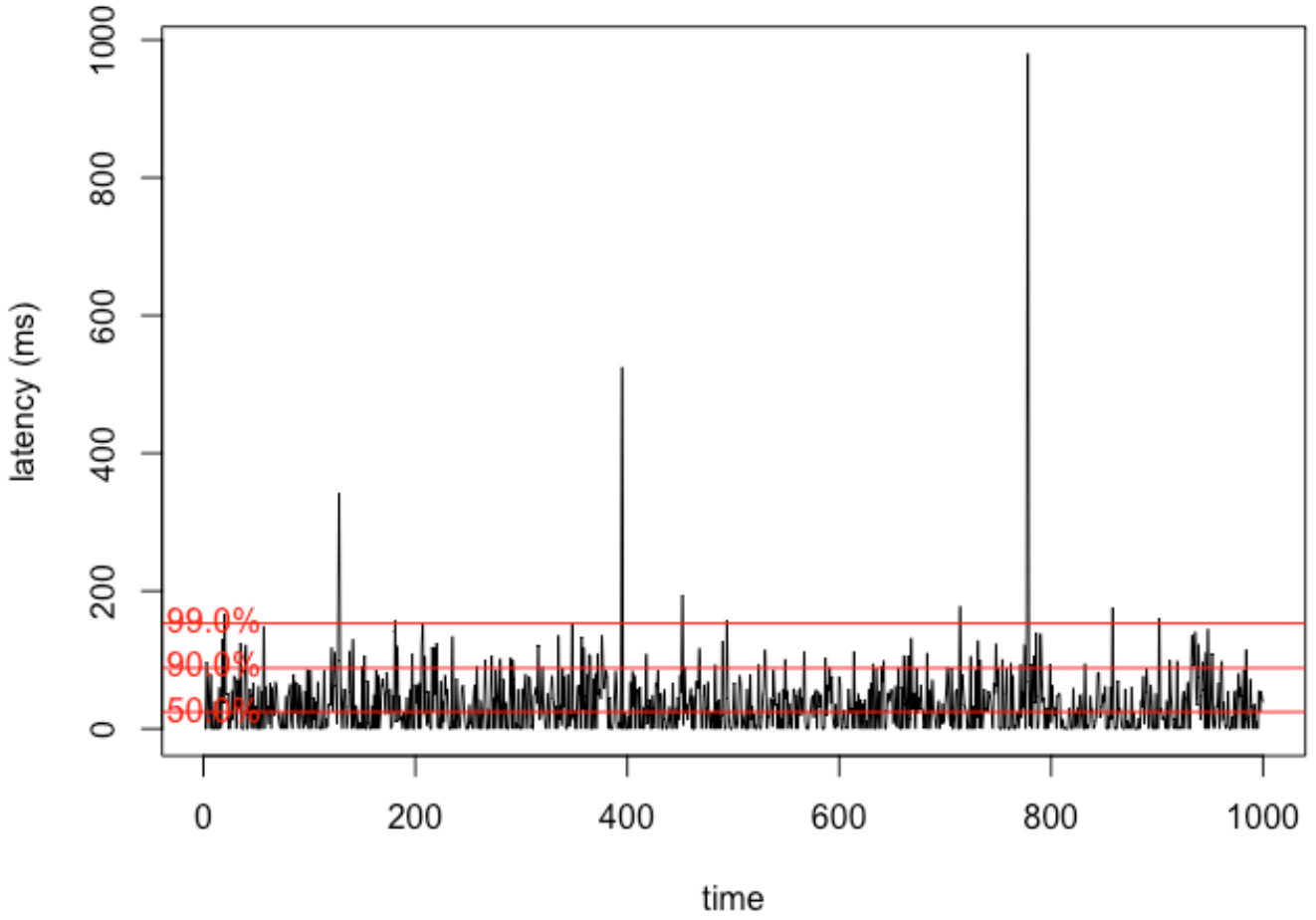
1. This is awesome
2. But you can't really *see* it

Yes, I know about RPlotExporter

C:\WINDOWS\system32\cmd.exe

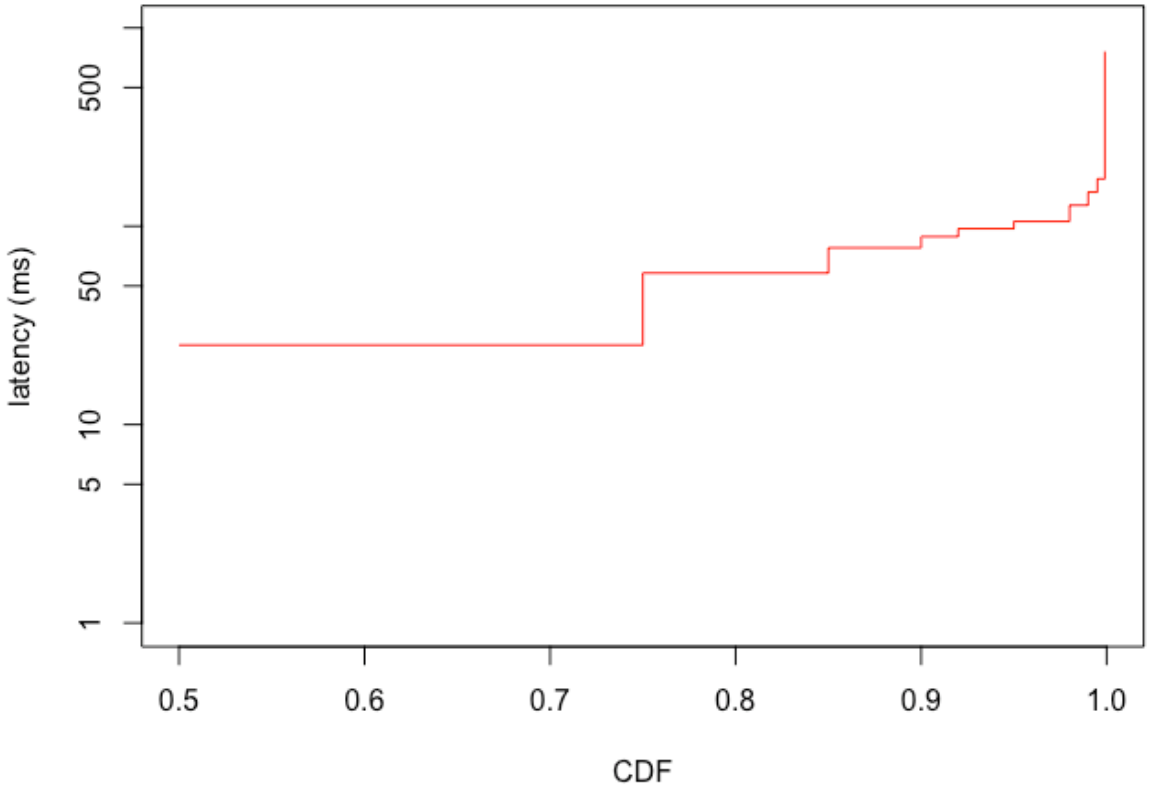
```
// * Detailed results *
MyAPI.Implementation1: DefaultJob
Runtime = Clr 4.0.30319.42000, 32bit LegacyJIT-v4.7.2046.0; GC = Concurrent Workstation
Mean = 5.4432 us, StdErr = 0.0320 us (0.59%); N = 52, StdDev = 0.2309 us
Min = 5.1388 us, Q1 = 5.2880 us, Median = 5.3850 us, Q3 = 5.5639 us, Max = 6.3504 us
IQR = 0.2759 us, LowerFence = 4.8742 us, UpperFence = 5.9778 us
ConfidenceInterval = [5.3314 us; 5.5550 us] (CI 99.9%), Margin = 0.1118 us (2.05% of Mean)
Skewness = 1.48, Kurtosis = 5.94


Total time: 00:00:56 (56.91 sec)

// * Summary *

BenchmarkDotNet=v0.10.5, OS=Windows 10.0.15063
Processor=Intel Core i7-4980HQ CPU 2.80GHz (Haswell), ProcessorCount=4
Frequency=2728060 Hz, Resolution=366.5609 ns, Timer=TSC
  [Host]     : Clr 4.0.30319.42000, 32bit LegacyJIT-v4.7.2046.0
  DefaultJob : Clr 4.0.30319.42000, 32bit LegacyJIT-v4.7.2046.0


         Method |     Mean |     Error |    StdDev |
--------------- |---------:|----------:|----------:|
 Implementation1 | 5.443 us | 0.1118 us | 0.2309 us |

// * Hints *
Outliers
  MyAPI.Implementation1: Default -> 11 outliers were removed

// * Legends *
  Mean   : Arithmetic mean of all measurements
  Error  : Half of 99.9% confidence interval
  StdDev : Standard deviation of all measurements
```

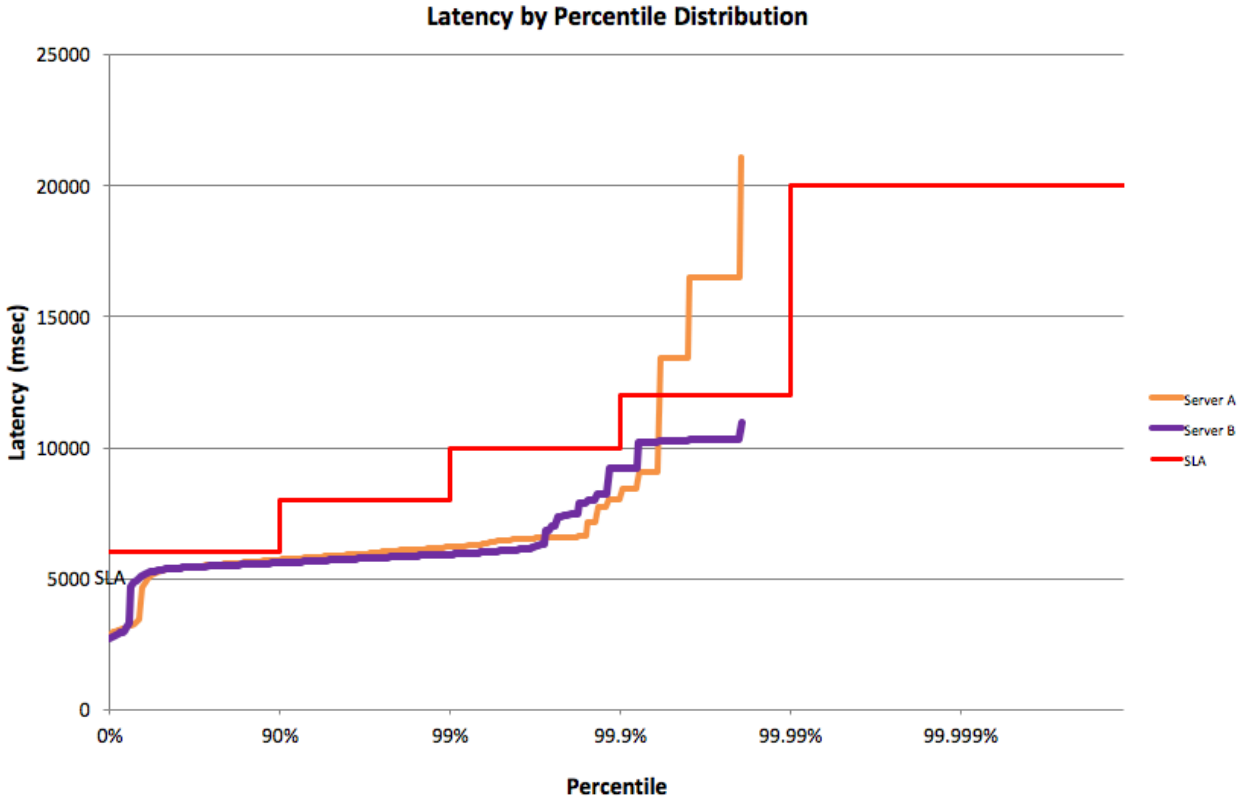
Hiccups



- Hiccups are important: these are the odd conditions you're often investigating
- μ = 35.75ms
- σ = 50.95ms
- μ + 3σ = 188.60ms
- 99th percentile still looks fine!

Plotting Percentiles (Cumulative Distribution)





To create your own percentile charts, try [HdrHistogram Plotter]

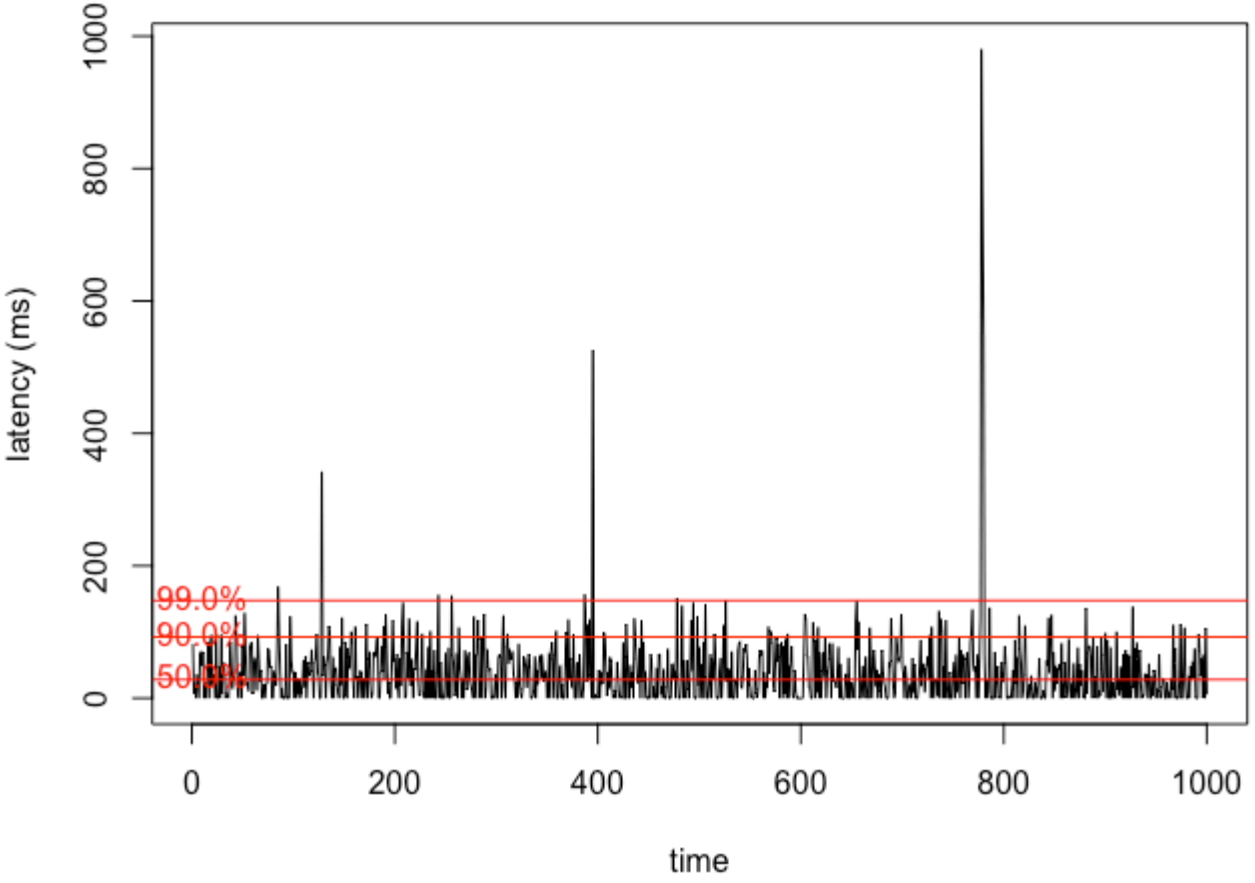
From: Cust McCustomer <custmcc@acme.com>
Subject: Latest latency measurements

I'm happy to report that over the last 24 hours, the average 90th percentile latency across all instances was 57ms, well within the SLA.

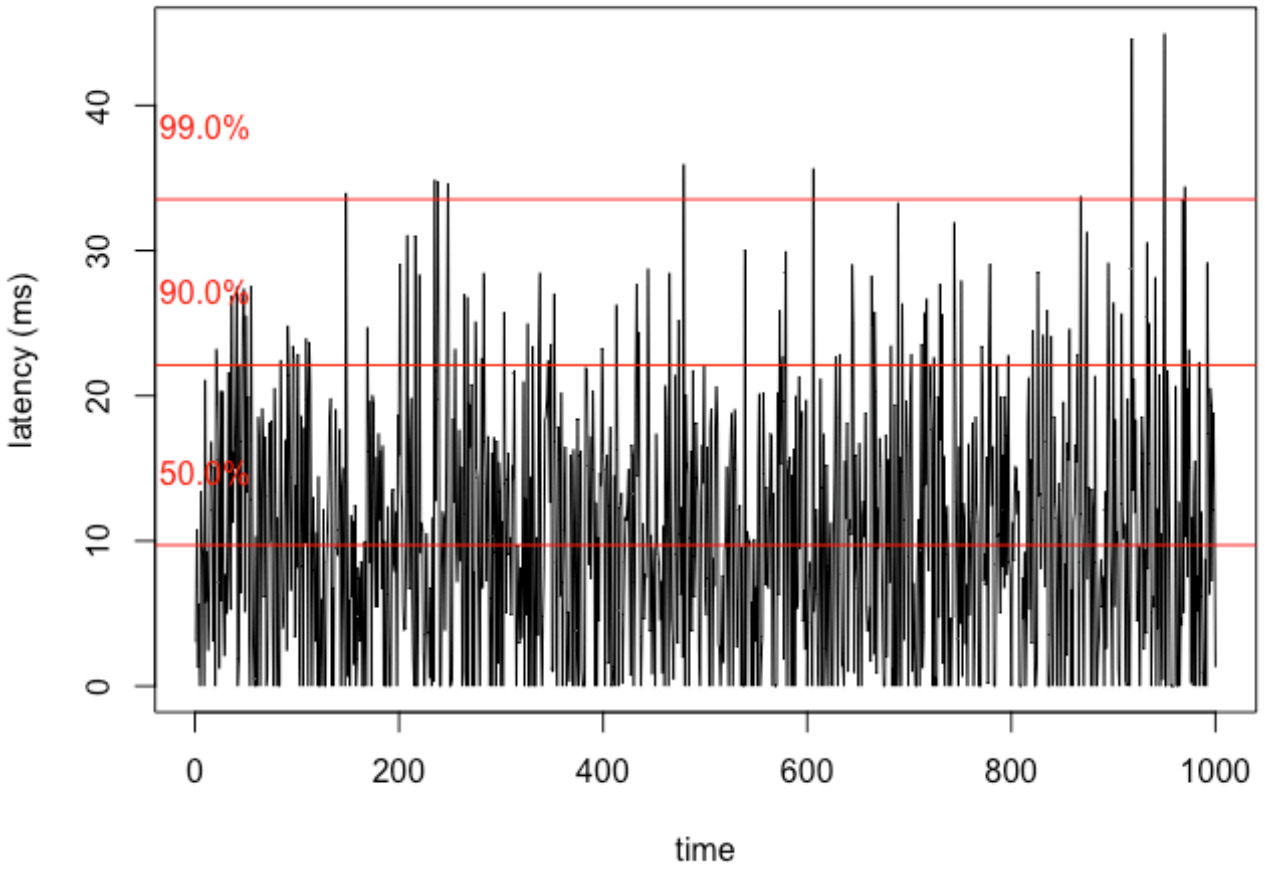
Congrats!

- C

Server A: 90% percentile is 92ms



Server B: 90% percentile is 22ms



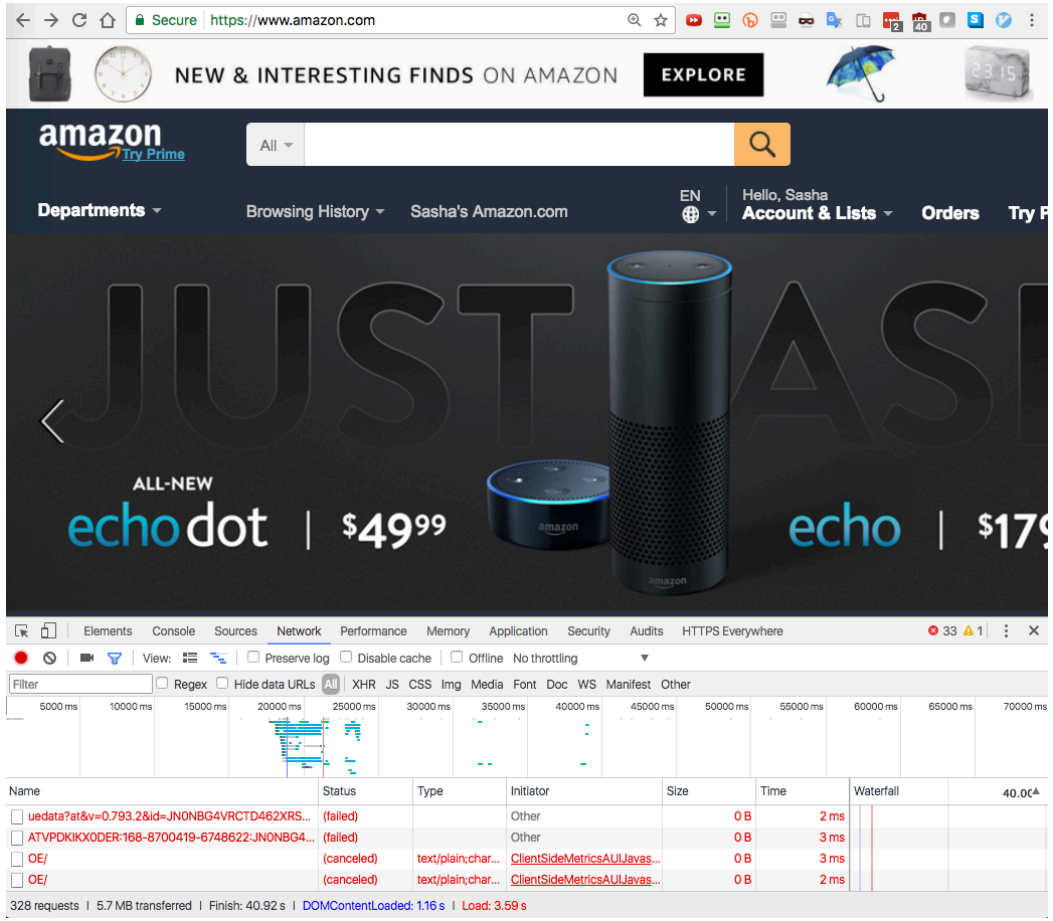
It is not true that 90% of requests are getting a response in under 57ms

90% percentile of all requests is in fact 68ms

Who Cares About The 99th Percentile?

- Amazon.com made 328 requests
- Probability for at least one 99th percentile request, assuming all requests are independent:

$$P = 1 - 0.99^{328} \approx 0.96$$



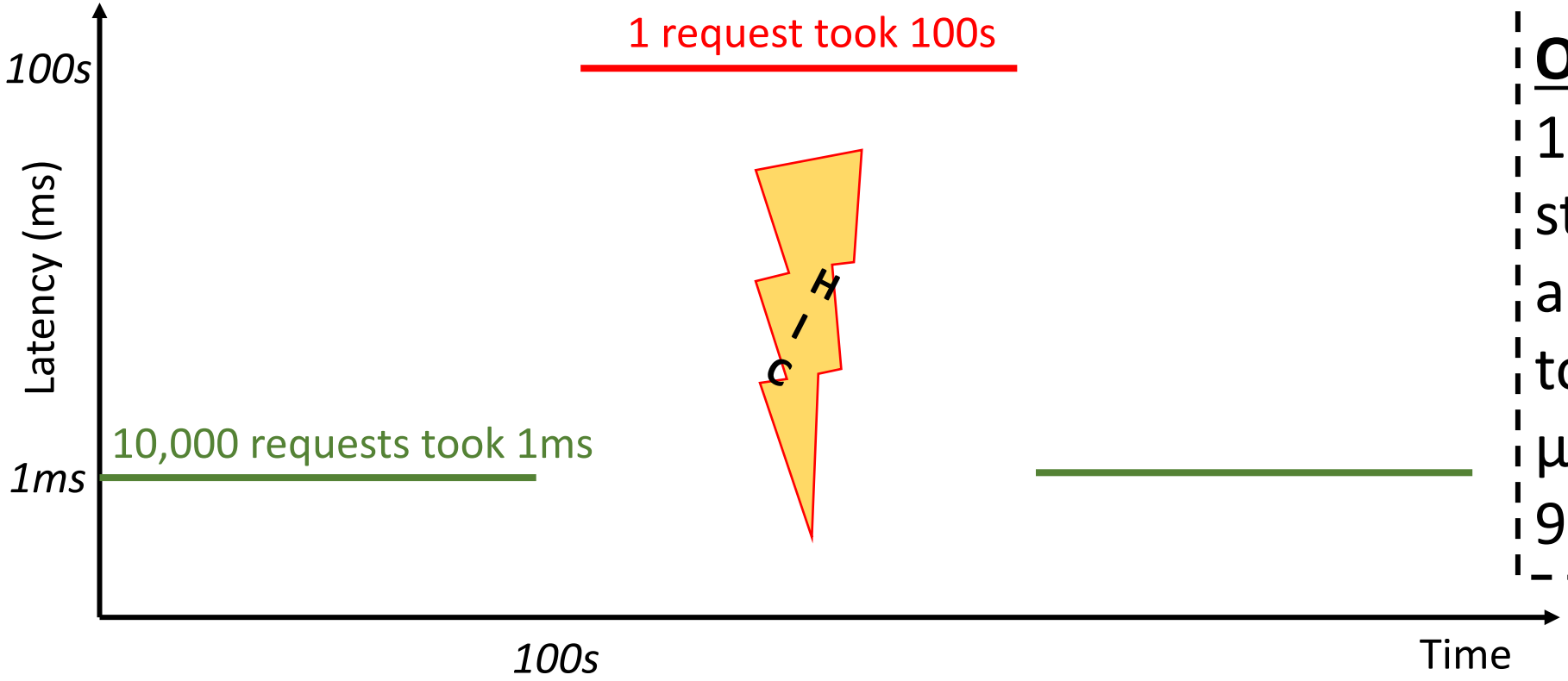
Coordinated Omission

- Basically: hiccups make your latency data too optimistic!
- Request latency measurements often do not include queueing time

```
var sw = Stopwatch.StartNew();
DoShowCart();
ReportTime("ShowCart", sw.Elapsed.Ticks);
```

- Benchmarking tools often wait for a slow request to complete before issuing the next one (rather than *N* requests/sec no matter what)

Coordinated Omission, Illustrated
Arrival Rate: 100 Requests/Sec



Option 1 😀

μ = 11ms

99.99% = 1ms

Option 2 😟

10,000 requests were stalled during hiccup and took 1ms to 100s to complete!

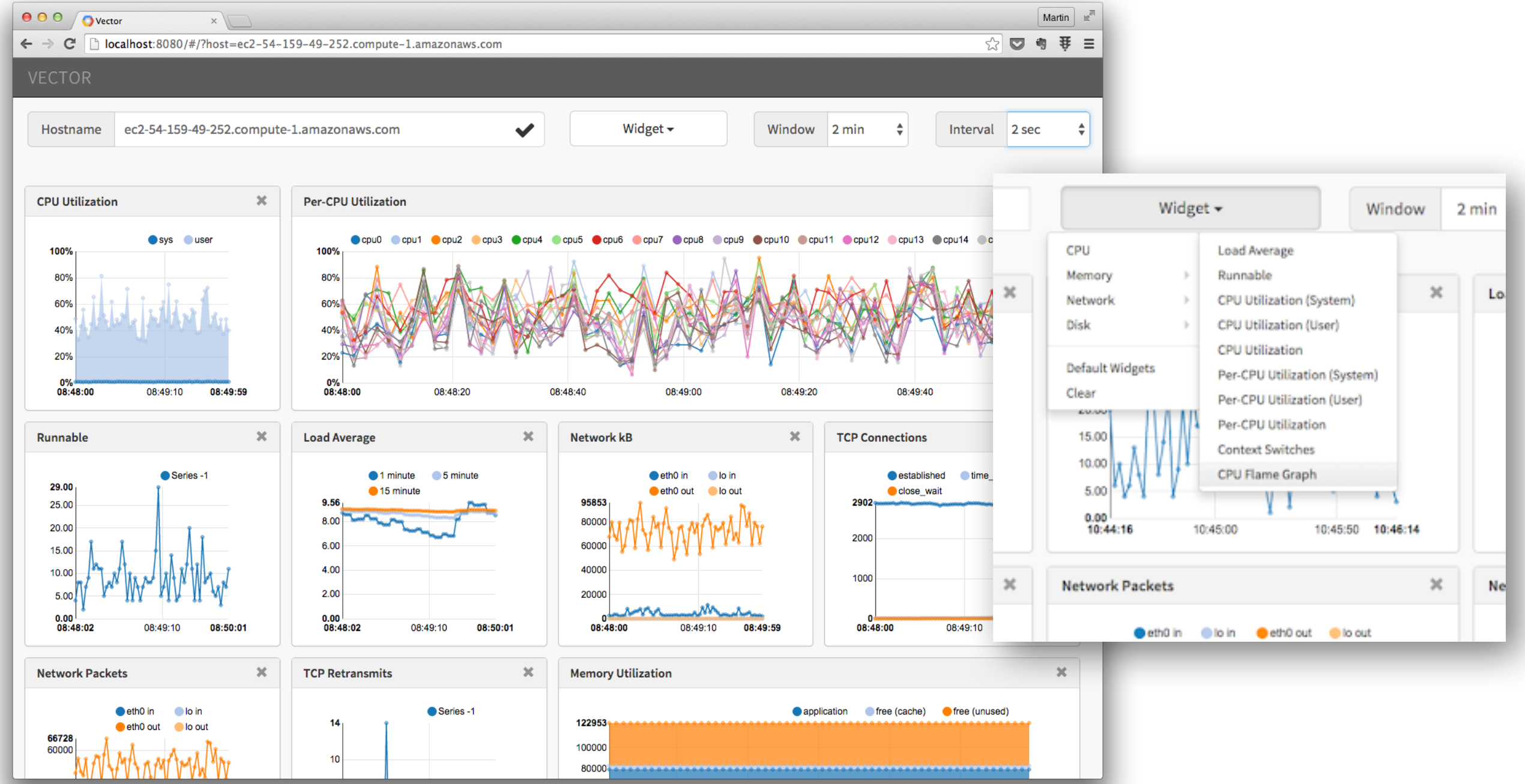
$\mu_{stalled}$ = 50s

99.99% = 100s

Adapted from [“How Not To Measure Latency”](), Azul Systems

Large Systems Need Large Tools

- Collect performance metrics from a large fleet
- Associate user activities with a session or transaction id
- Visualize using a dashboard
- Allow drilling-in and running single-machine performance tools with a single click


Vector
Martin
localhost:8080/#/?host=ec2-54-159-49-252.compute-1.amazonaws.com
VECTOR
Hostname
ec2-54-159-49-252.compute-1.amazonaws.com
Widget
Window
2 min
Interval
2 sec
CPU Utilization
sys
user
Per-CPU Utilization
cpu0 cpu1 cpu2 cpu3 cpu4 cpu5 cpu6 cpu7 cpu8 cpu9 cpu10 cpu11 cpu12 cpu13 cpu14
Runnable
Series -1
Load Average
1 minute
5 minute
15 minute
Network kB
eth0 in
lo in
eth0 out
lo out
TCP Connections
established
close_wait
Network Packets
TCP Retransmits
Series -1
Memory Utilization
application
free (cache)
free (unused)
Widget
Window
2 min
CPU
Memory
Network
Disk
Default Widgets
Clear
Load Average
Runnable
CPU Utilization (System)
CPU Utilization (User)
CPU Utilization
Per-CPU Utilization (System)
Per-CPU Utilization (User)
Per-CPU Utilization
Context Switches
CPU Flame Graph
Network Packets
eth0 in
lo in
eth0 out
lo out

App server | Browser

Graph view | Table view

Most time consuming

Alerts::EventsController#recent_events	**10.7%**
CurrentStatusController#status_bar	8.31%
Api::V1::DataController#multi_app_data	7.7%
ChartData::MetricChartsController#app_breakdown	6.79%
ApplicationsController#index	5.07%
Api::V1::DataController#data	4.92%
ChartData::BaseChartsController#combined_apdex	4.39%
ApplicationsController#load_app_server_transactior	3.85%
ApplicationsController#show	3.82%
ChartData::MetricChartsController#metric_regex	3.78%
PublicAccess::ChartsController#data	3.49%
Api::V1::DataController#chart_data	2.45%
ApplicationsController#hosts	2.37%
Api::V1::ThresholdValuesController#index	2.32%
AccountFeeds::EventFeedsController#index	2.22%
ChartData::NrqlController#prime_cache	1.97%
Infrastructure::HostsController#show	1.62%
ChartData::MetricChartsController#enduser_map	1.58%
ChartData::MetricChartsController#single_metric	1.58%
Api::V1::ApplicationsController#index	1.46%

Alerts::EventsController#recent_events

Key Transaction

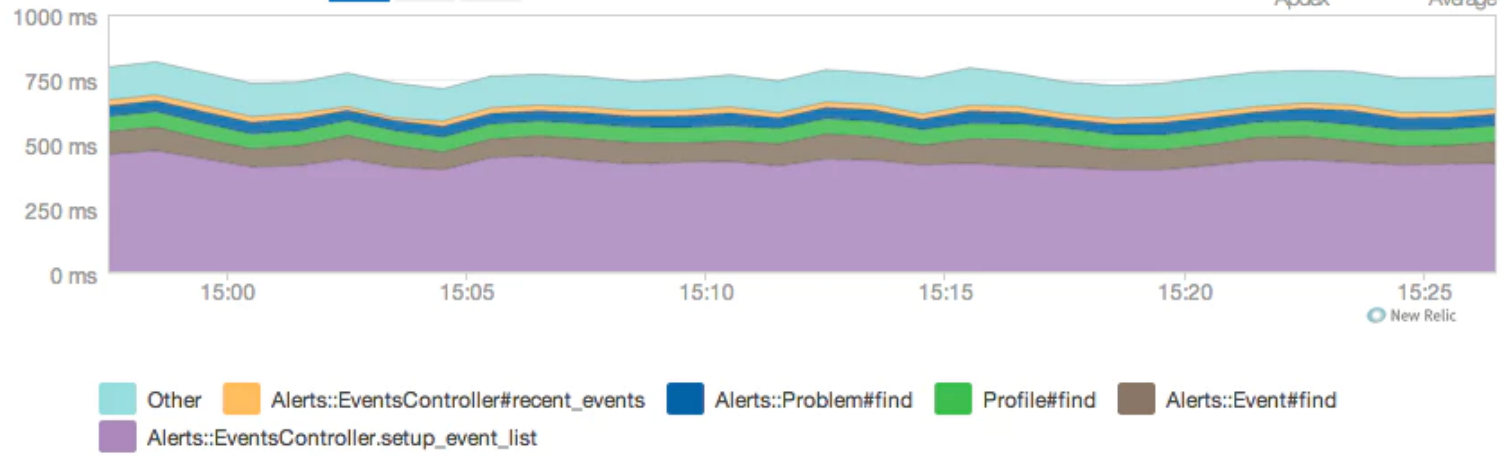
App performance | Browser performance | Historical performance

App server breakdown

%

0.69 Apdex

760 ms Average



Other | Alerts::EventsController#recent_events | Alerts::Problem#find | Profile#find | Alerts::Event#find | Alerts::EventsController.setup_event_list

Throughput

740 rpm Average

Breakdown table

App server transaction traces

Sample performance details	App server	Browser
15:25 – 3 minutes ago	2,021 ms	
15:27 – 1 minute ago	1,818 ms	

Conduct a Postmortem

1. Document the steps taken to identify, diagnose, resolve, and verify the problem
2. Which tools did you use? Can they be improved?
3. Where were the bottlenecks in your investigation?
4. Can you add monitoring for sysadmins/ops?
5. Can you add instrumentation for investigators?
6. How do we triage this problem *automatically* next time it happens?

Summary

- We have learned:

✓What a performance investigation looks like

✓Methods and anti-patterns for performance investigations

✓Tools for production profiling and monitoring .NET applications

✓How to avoid getting lied to by tools, dashboards, and reports

References And Further Reading

- Methodology
  - Systems Performance: Enterprise and The Cloud
  - The USE method
- Event Tracing for Windows
  - My PerfView talk
  - Dina Goldshtein's ETW talk
- Visualization
  - Flame graphs
  - Histograms and percentiles
- Tools
  - etrace
  - LiveStacks
- Statistics
  - Statistics Tutorial for IT Operations Engineers
  - How Not To Measure Latency
- System internals
  - Windows Internals, 6th edition
  - Windows Memory Usage, Demystified
  - Pro .NET Performance
  - Understanding the Linux Kernel

Thank You!

https://s.sashag.net/dotnext17

Sasha Goldshtein
CTO, Sela Group


goldshtn


goldshtn