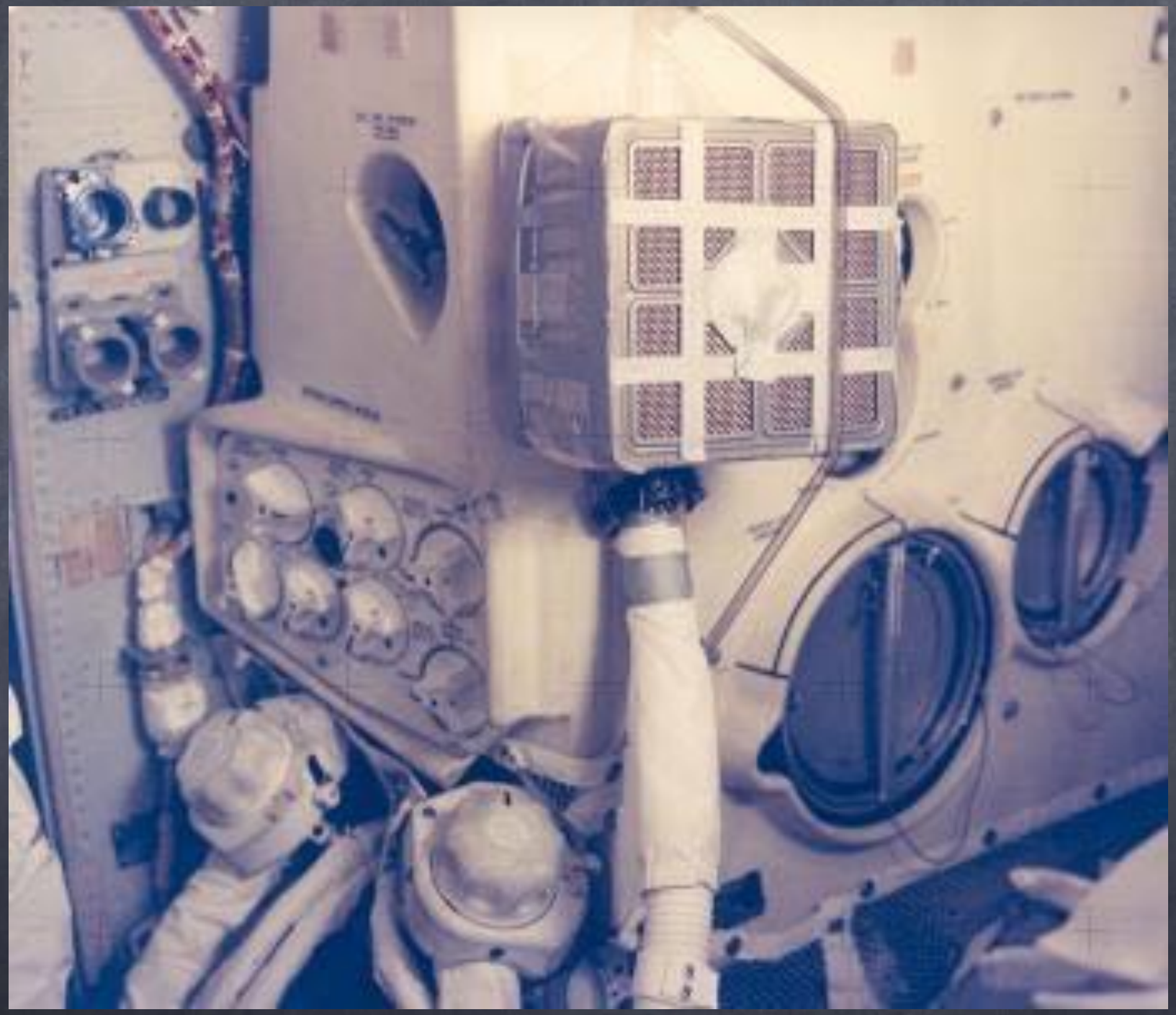


AZUL SYSTEMS

Java at Speed

getting the most out of modern hardware

Gil Tene, CTO & co-Founder, Azul Systems

AZUL SYSTEMS

High level agenda

- Intro & Motivation
- Some hardware trends and new features
- Some compiler stuff
- A microbenchmark detour
- Some more compiler stuff
- Warmup, and what we can do about it
- Putting it all together (and maybe some bragging)

About me: Gil Tene

- co-founder, CTO @Azul Systems
- Have been working on "think different" GC and runtime approaches since 2002
- A Long history building Virtual & Physical Machines, Operating Systems, Enterprise apps, etc…
- At Azul we make JVMs that dramatically improve response time and latency behaviors
- I also depress people by demonstrating how terribly wrong their latency measurements are…

* working on real-world trash compaction issues, circa 2004

AZUL SYSTEMS

Speed

What is it good for?

?

?

Are you fast?

?

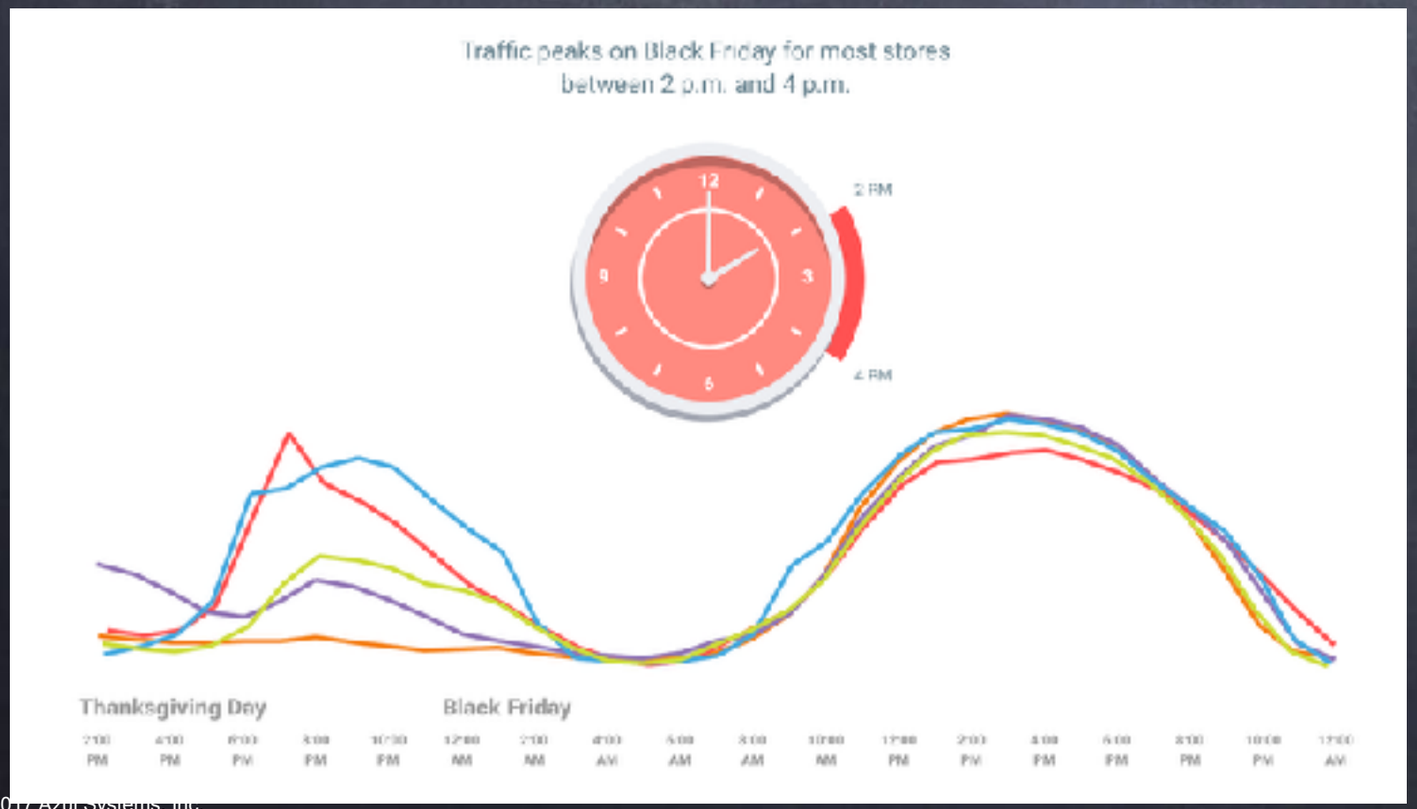
?

Are you fast when new code rolls out?

?

?

Are you fast when it matters?



?

?

Are you fast at Market Open?

?

?

Are you fast when you actually trade?

?

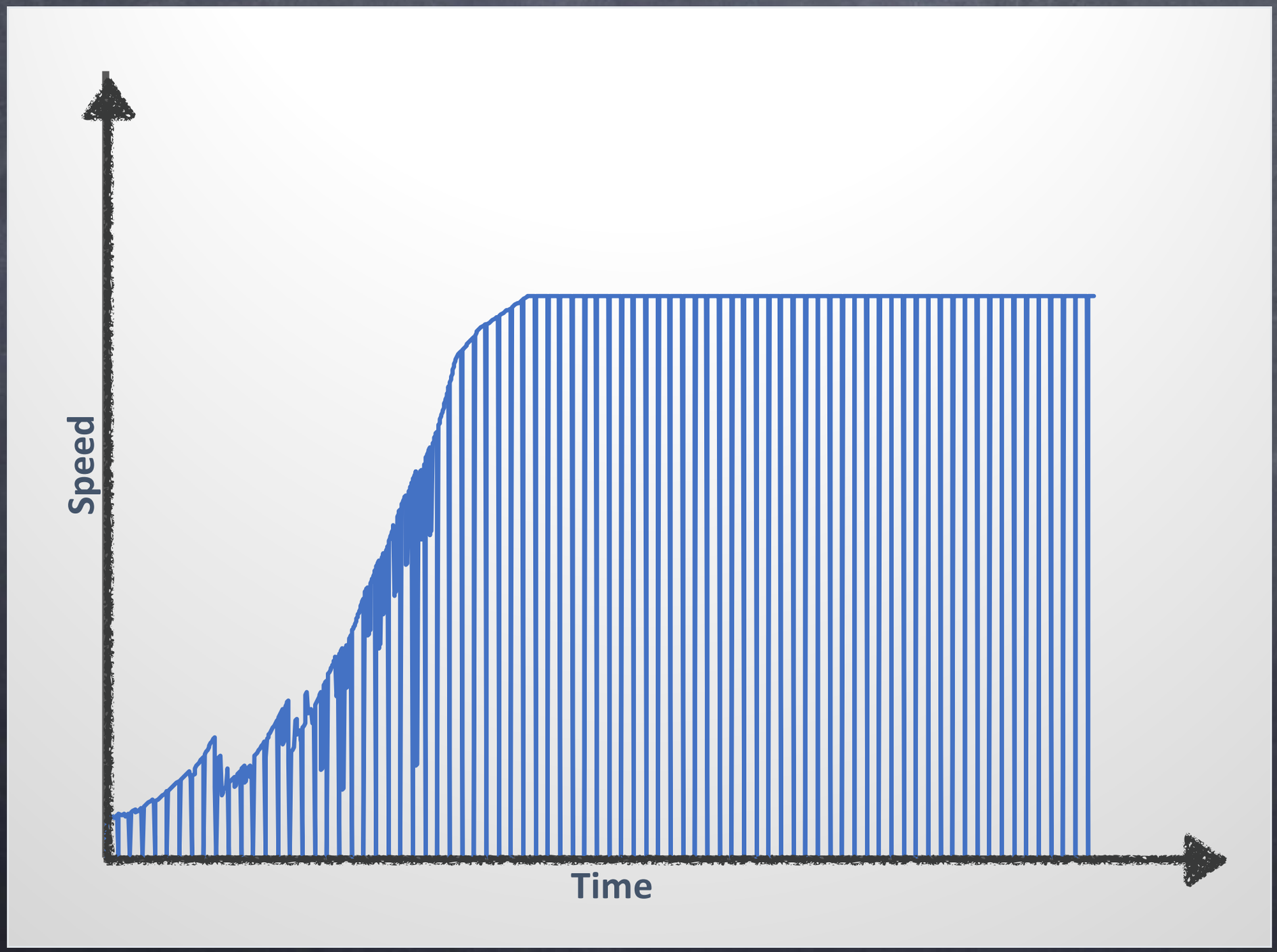
?

Are you reliably fast?

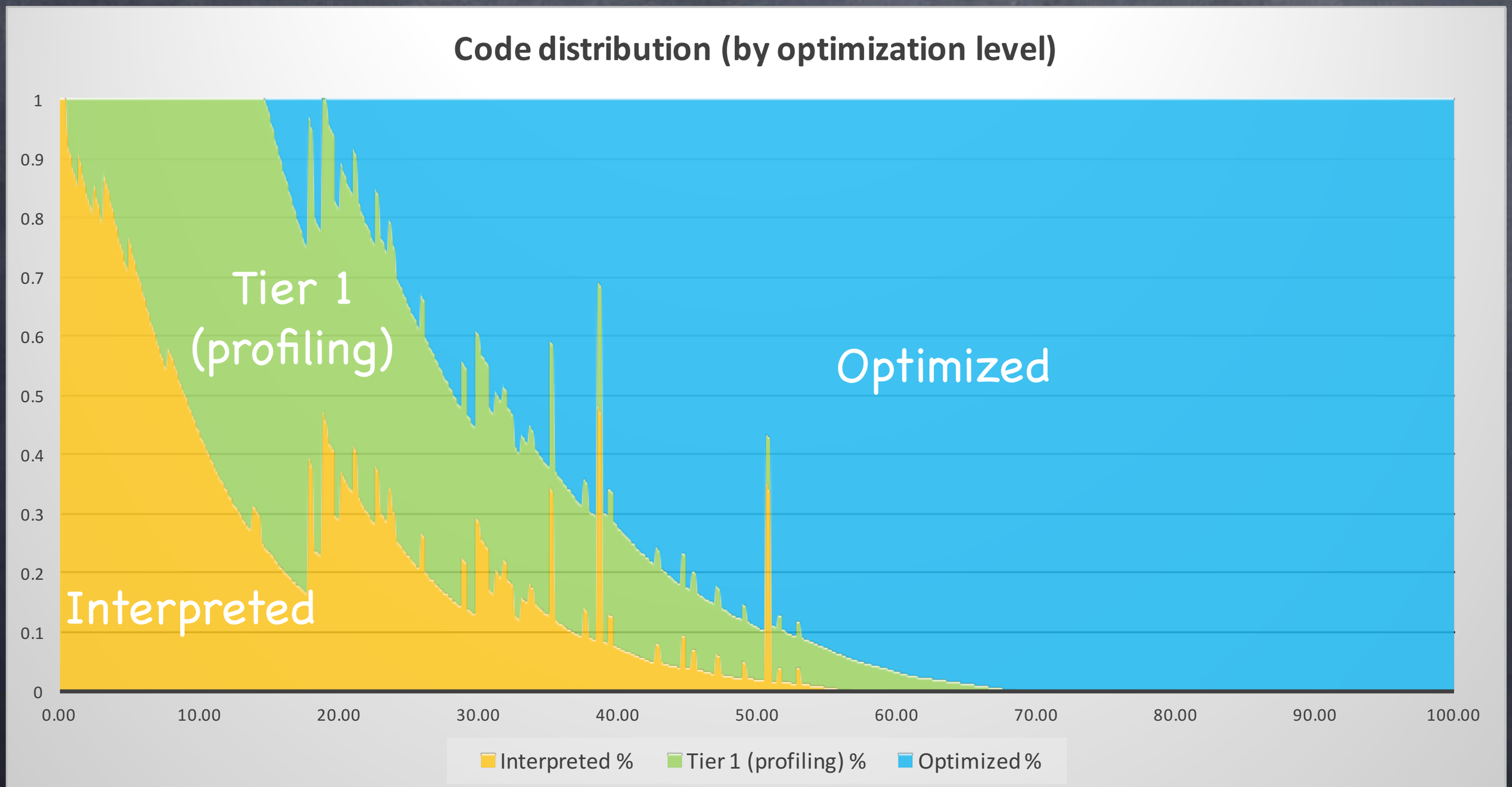
What do you mean by “fast”?

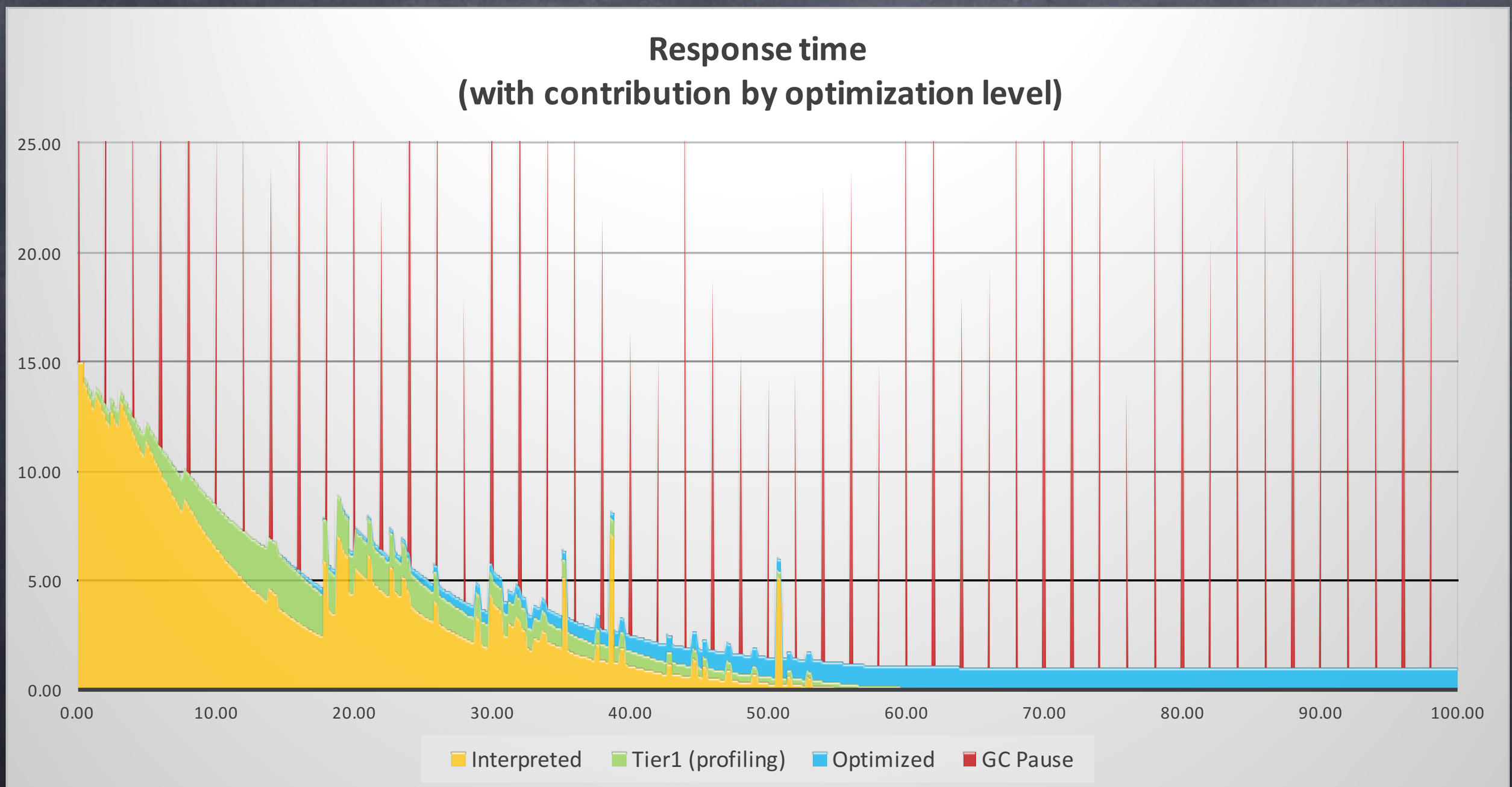
??

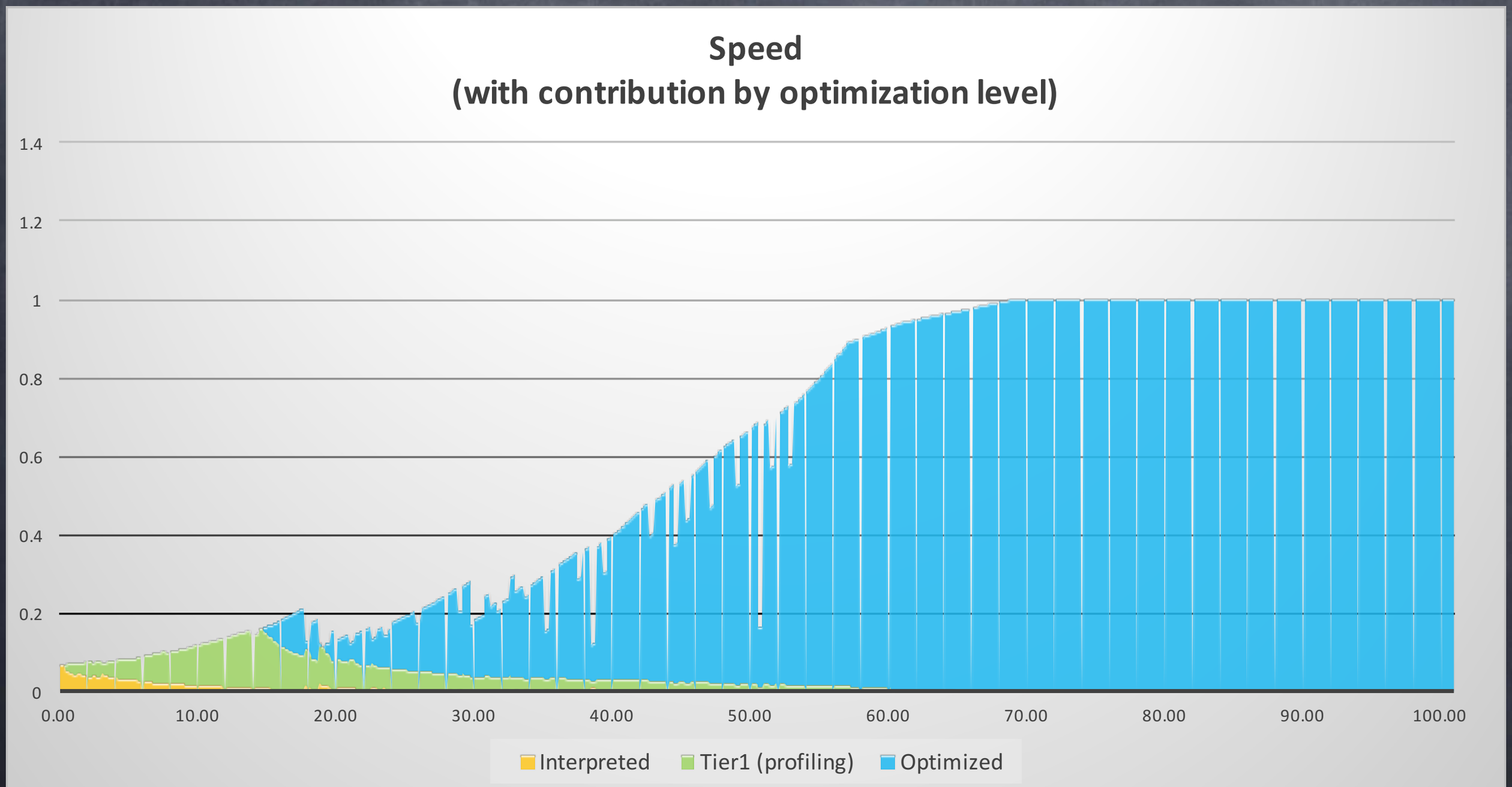
What do you mean by “fast”?



Speed in the Java world…







Some notes on modern servers

AZUL SYSTEMS

Code name	Model	Intro Date		cores/chip
Nehalem EP	Xeon 5500	March 2009		4
Westemere EP	Xeon 5600	June 2010		6
Sandy Bridge EP	E5-2600	March 2012	AVX	8
Ivy Bridge EP	E5-2600 V2	Sep. 2013		12
Haswell EP	E5-2600 V3	Sep. 2014	AVX2, BMI, BMI2	18
Broadwell EP	E5-2600 V4	March 2016	TSX, HLE	22
Skylake SP	Silver/Gold/…	July 2017	AVX512	32

Instruction Window Keeps Increasing

	Sandy Bridge	Haswell	SkyLake
Out-of-order Window	168	192	224 ↑
In-flight Loads	64	72	72
In-flight Stores	36	42	56 ↑
Scheduler Entries	54	60	97 ↑
Integer Register File	160	168	180 ↑
FP Register File	144	168	168
Allocation Queue	28/thread	56	64/thread ↑

Extract more parallelism in every generation

Intel Next Generation Microarchitecture Code Name Skylake

IDF15

AZUL SYSTEMS

Some machine code zoom-in

AZUL SYSTEMS — Azul ZVision

User: gil Host: cperf2 / 47781 Version: 1.8.0-zing_99.99.99.99.dev-b1686
Uptime: 00:21:51

Overview | Azul Support | Threads | CPU | Memory | Compilers | Applications

Tick Profile | Metaticks | Event Tracker

Pause The Tick Collection | Reset Tick Profile | Stop Saving Ticks To Disk

Timer Tick Profile

Cutoff: .1 Threads (comma separated list): None JVM All Submit

Note: Functions that could not be resolved to a name string are displayed as an address followed by "<-" and the first calling function (in their stack) that can.

Percent	Ticks	Source
41.8%	11,459	bench.CodeGenExampleBench.trueIfMaskMatched (codeblob)
18.9%	5,169	bench.CodeGenExampleBench.addArraysIfEven (codeblob)
11.2%	3,063	bench.CodeGenExampleBench.addXtoArray (codeblob)
8.3%	2,267	bench.CodeGenExampleBench.sumShifted (codeblob)
8.0%	2,190	bench.CodeGenExampleBench.sumIfEvenLoop (codeblob)
5.1%	1,405	bench.CodeGenExampleBench.sumLoop (codeblob)
0.5%	128	ElfSymbolTable::lookup(unsigned char*, int*, int*, int*, unsigned long*) (PC ref)
0.3%	93	_dl_addr (PC ref)
0.2%	64	_int_free (PC ref)
0.2%	59	libLLVM-4.0svn.so (PC ref)
0.2%	58	_int_malloc (PC ref)
0.2%	57	__libc_malloc (PC ref)
0.1%	35	syscall (PC ref)
0.1%	32	llvm::PMDataManager::findAnalysisPass(void const*, bool) (PC ref)
0.1%	16	computeKnownBits(llvm::Value const*, llvm::APInt&, llvm::APInt&, unsigned int, (anonymous namespace)::Query const&) (PC ref)
0.1%	15	llvm::PMTopLevelManager::setLastUser(llvm::ArrayRef, llvm::Pass*) (PC ref)
0.0%	12	bench.CodeGenExampleBench.doAddArraysIfEven (codeblob)
0.0%	12	ioctl (PC ref)
0.0%	12	malloc_consolidate (PC ref)
0.0%	12	llvm::InstCombiner::run() (PC ref)
0.0%	11	llvm::PMTopLevelManager::findAnalysisPass(void const*) (PC ref)
0.0%	11	llvm::Use::getImpliedUser() const (PC ref)
0.0%	10	llvm::ValueHandleBase::AddToUseList() (PC ref)
0.0%	10	mprotect (PC ref)
0.0%	10	malloc (PC ref)
0.0%	9	sysmalloc (PC ref)
0.0%	8	bench.CodeGenExampleBench.doSumShiftedLoop (codeblob)
0.0%	8	llvm::Value::getValueName() const (PC ref)

AZUL SYSTEMS

bench.CodeGenExampleBench.sumLoop([I)I

ID: 0x3000f600

Reset Tick Profile

Assembly | Callee | Caller

Percent	Ticks	Address	Code	Opcode
		0x3000f650	pushq %rax	0xfff0
		0x3000f652	cmpl $0, %gs:104	0x65833c2568000000000
		0x3000f65b	jne 127 ; ABS: 0x3000f6dc	0x757f
		0x3000f65d	movl 8(%rsi), %ecx // NPE-> 0x3000f6f5	0x8b4e08
		0x3000f660	testq %rcx, %rcx	0x4885c9
		0x3000f663	je 115 ; ABS: 0x3000f6d8	0x7473
		0x3000f665	cmpl $7, %ecx	0x83f907
		0x3000f668	ja 20 ; ABS: 0x3000f67e	0x7714
		0x3000f66a	xorl %edx, %edx	0x31d2
		0x3000f66c	xorl %eax, %eax	0x31c0
		0x3000f66e	nop	0x6690
		0x3000f670	addl 12(%rsi,%rdx,4), %eax	0x0344960c
		0x3000f674	incq %rdx	0x48ffc2
		0x3000f677	cmpq %rcx, %rdx	0x4839ca
		0x3000f67a	jl -12 ; ABS: 0x3000f670	0x7cf4
		0x3000f67c	popq %rcx	0x59
0.05%	1	0x3000f67d	retq	0xc3
		0x3000f67e	movl %ecx, %r8d	0x4189c8
		0x3000f681	andl $7, %r8d	0x4183e007
		0x3000f685	movq %rcx, %rdx	0x4889ca
		0x3000f688	subq %r8, %rdx	0x4c29c2
0.05%	1	0x3000f68b	je -35 ; ABS: 0x3000f66a	0x74dd
		0x3000f68d	leaq 28(%rsi), %rax	0x488d461c
		0x3000f691	pxor %xmm0, %xmm0	0x660fefc0
		0x3000f695	movq %rdx, %rdi	0x4889d7
		0x3000f698	pxor %xmm1, %xmm1	0x660fefc9
		0x3000f69c	nopl (%rax)	0x0f1f4000
21.90%	418	0x3000f6a0	movdqu -16(%rax), %xmm2	0xf30f6f50f0
5.08%	97	0x3000f6a5	movdqu (%rax), %xmm3	0xf30f6f18
40.86%	780	0x3000f6a9	paddd %xmm2, %xmm0	0x660ffec2
2.72%	52	0x3000f6ad	paddd %xmm3, %xmm1	0x660ffecb
29.23%	558	0x3000f6b1	addq $32, %rax	0x4883c020
		0x3000f6b5	addq $-8, %rdi	0x4883c7f8
		0x3000f6b9	jne -27 ; ABS: 0x3000f6a0	0x75e5
		0x3000f6bb	paddd %xmm0, %xmm1	0x660ffec8
		0x3000f6bf	pshufd $78, %xmm1, %xmm0	0x660f70c14e
		0x3000f6c4	paddd %xmm1, %xmm0	0x660ffec1
		0x3000f6c8	phaddd %xmm0, %xmm0	0x660f3802c0
0.05%	1	0x3000f6cd	movd %xmm0, %eax	0x660f7ec0
0.05%	1	0x3000f6d1	testl %r8d, %r8d	0x4585c0
		0x3000f6d4	jne -102 ; ABS: 0x3000f670	0x759a
		0x3000f6d6	jmp -92 ; ABS: 0x3000f67c	0xeba4
		0x3000f6d8	xorl %eax, %eax	0x31c0
		0x3000f6da	popq %rcx	0x59
		0x3000f6db	retq	0xc3

A simple array summing loop

```
private int sumLoop(int[] a) {
    int sum = 0;
    for (int i = 0; i < a.length; i++) {
        sum += a[i];
    }
    return sum;
}
```

```java
private int sumLoop(int[] a) {
    int sum = 0;
    for (int i = 0; i < a.length; i++) {
        sum += a[i];
    }
    return sum;
}
```

Address	Code	Opcode
0x3000fe50	pushq %rax	0xfff0
0x3000fe52	cmpl $0, %gs:104	0x65833c256800000000
0x3000fe5b	jne 127 ; ABS: 0x3000fedc	0x757f
0x3000fe5d	movl 8(%rsi), %ecx // NPE-> 0x3000fef5	0x8b4e08
0x3000fe60	testq %rcx, %rcx	0x4885c9
0x3000fe63	je 115 ; ABS: 0x3000fed8	0x7473
0x3000fe65	cmpl $7, %ecx	0x83f907
0x3000fe68	ja 20 ; ABS: 0x3000fe7e	0x7714
0x3000fe6a	xorl %edx, %edx	0x31d2
0x3000fe6c	xorl %eax, %eax	0x31c0
0x3000fe6e	nop	0x6690
0x3000fe70	addl 12(%rsi,%rdx,4), %eax	0x0344960c
0x3000fe74	incq %rdx	0x48ffc2
0x3000fe77	cmpq %rcx, %rdx	0x4839ca
0x3000fe7a	jl -12 ; ABS: 0x3000fe70	0x7cf4
0x3000fe7c	popq %rcx	0x59
0x3000fe7d	retq	0xc3
0x3000fe7e	movl %ecx, %r8d	0x4189c8
0x3000fe81	andl $7, %r8d	0x4183e007
0x3000fe85	movq %rcx, %rdx	0x4889ca
0x3000fe88	subq %r8, %rdx	0x4c29c2
0x3000fe8b	je -35 ; ABS: 0x3000fe6a	0x74dd
0x3000fe8d	leaq 28(%rsi), %rax	0x488d461c
0x3000fe91	pxor %xmm0, %xmm0	0x660fefc0
0x3000fe95	movq %rdx, %rdi	0x4889d7
0x3000fe98	pxor %xmm1, %xmm1	0x660fefc9
0x3000fe9c	nopl (%rax)	0x0f1f4000
0x3000fea0	movdqu -16(%rax), %xmm2	0xf30f6f50f0
0x3000fea5	movdqu (%rax), %xmm3	0xf30f6f18
0x3000fea9	paddd %xmm2, %xmm0	0x660ffec2
0x3000fead	paddd %xmm3, %xmm1	0x660ffecb
0x3000feb1	addq $32, %rax	0x4883c020
0x3000feb5	addq $-8, %rdi	0x4883c7f8
0x3000feb9	jne -27 ; ABS: 0x3000fea0	0x75e5
0x3000febb	paddd %xmm0, %xmm1	0x660ffec8
0x3000febf	pshufd $78, %xmm1, %xmm0	0x660f70c14e
0x3000fec4	paddd %xmm1, %xmm0	0x660ffec1
0x3000fec8	phaddd %xmm0, %xmm0	0x660f3802c0
0x3000fecd	movd %xmm0, %eax	0x660f7ec0
0x3000fed1	testl %r8d, %r8d	0x4585c0
0x3000fed4	jne -102 ; ABS: 0x3000fe70	0x759a
0x3000fed6	jmp -92 ; ABS: 0x3000fe7c	0xeba4
0x3000fed8	xorl %eax, %eax	0x31c0
0x3000feda	popq %rcx	0x59
0x3000fedb	retq	0xc3
0x3000fedc	movq %rsi, (%rsp)	0x48893424
0x3000fee0	movabsq $805334400, %rax	0x48b8806d003000000000
0x3000feea	callq *%rax	0xffd0
0x3000feec	movq (%rsp), %rsi	0x488b3424
0x3000fef0	jmp -152 ; ABS: 0x3000fe5d	0xe968ffffff
0x3000fef5	movabsq $805319872, %rax	0x48b8c034003000000000
0x3000feff	movl $7, %edi	0xbf07000000
0x3000ff04	callq *%rax	0xffd0
0x3000ff06	addq $-8, %rsp	0x4883c4f8
0x3000ff0a	jmp -50575 ; ABS: 0x30003980 = StubRoutines::deoptimize	0xe9713affff
0x3000ff0f	int3	0xcc

AZUL SYSTEMS

```java
private int sumLoop(int[] a) {
    int sum = 0;
    for (int i = 0; i < a.length; i++) {
        sum += a[i];
    }
    return sum;
}
```

Percent	Ticks	Address	Code	Opcode
		0x3000fe50	pushq %rax	0xfff0
		0x3000fe52	cmpl $0, %gs:104	0x65833c256800000000
		0x3000fe5b	jne 127 ; ABS: 0x3000fedc	0x757f
		0x3000fe5d	movl 8(%rsi), %ecx // NPE-> 0x3000fef5	0x8b4e08
0.06%	1	0x3000fe60	testq %rcx, %rcx	0x4885c9
		0x3000fe63	je 115 ; ABS: 0x3000fed8	0x7473
		0x3000fe65	cmpl $7, %ecx	0x83f907
		0x3000fe68	ja 20 ; ABS: 0x3000fe7e	0x7714
		0x3000fe6a	xorl %edx, %edx	0x31d2
		0x3000fe6c	xorl %eax, %eax	0x31c0
		0x3000fe6e	nop	0x6690
		0x3000fe70	addl 12(%rsi,%rdx,4), %eax	0x0344960c
		0x3000fe74	incq %rdx	0x48ffc2
		0x3000fe77	cmpq %rcx, %rdx	0x4839ca
		0x3000fe7a	jl -12 ; ABS: 0x3000fe70	0x7cf4
		0x3000fe7c	popq %rcx	0x59
		0x3000fe7d	retq	0xc3
0.06%	1	0x3000fe7e	movl %ecx, %r8d	0x4189c8
		0x3000fe81	andl $7, %r8d	0x4183e007
		0x3000fe85	movq %rcx, %rdx	0x4889ca
		0x3000fe88	subq %r8, %rdx	0x4c29c2
		0x3000fe8b	je -35 ; ABS: 0x3000fe6a	0x74dd
		0x3000fe8d	leaq 28(%rsi), %rax	0x488d461c
		0x3000fe91	pxor %xmm0, %xmm0	0x660fefc0
		0x3000fe95	movq %rdx, %rdi	0x4889d7
		0x3000fe98	pxor %xmm1, %xmm1	0x660fefc9
		0x3000fe9c	nopl (%rax)	0x0f1f4000
20.64%	327	0x3000fea0	movdqu -16(%rax), %xmm2	0xf30f6f50f0
35.42%	561	0x3000fea5	movdqu (%rax), %xmm3	0xf30f6f18
11.30%	179	0x3000fea9	paddd %xmm2, %xmm0	0x660ffec2
13.70%	217	0x3000fead	paddd %xmm3, %xmm1	0x660ffecb
18.69%	296	0x3000feb1	addq $32, %rax	0x4883c020
		0x3000feb5	addq $-8, %rdi	0x4883c7f8
		0x3000feb9	jne -27 ; ABS: 0x3000fea0	0x75e5
		0x3000febb	paddd %xmm0, %xmm1	0x660ffec8
		0x3000febf	pshufd $78, %xmm1, %xmm0	0x660f70c14e
		0x3000fec4	paddd %xmm1, %xmm0	0x660ffec1
		0x3000fec8	phaddd %xmm0, %xmm0	0x660f3802c0
0.13%	2	0x3000fecd	movd %xmm0, %eax	0x660f7ec0
		0x3000fed1	testl %r8d, %r8d	0x4585c0
		0x3000fed4	jne -102 ; ABS: 0x3000fe70	0x759a
		0x3000fed6	jmp -92 ; ABS: 0x3000fe7c	0xeba4
		0x3000fed8	xorl %eax, %eax	0x31c0
		0x3000feda	popq %rcx	0x59
		0x3000fedb	retq	0xc3
		0x3000fedc	movq %rsi, (%rsp)	0x48893424
		0x3000fee0	movabsq $805334400, %rax	0x48b8806d003000000000
		0x3000feea	callq *%rax	0xffd0
		0x3000feec	movq (%rsp), %rsi	0x488b3424
		0x3000fef0	jmp -152 ; ABS: 0x3000fe5d	0xe968ffffff
		0x3000fef5	movabsq $805319872, %rax	0x48b8c034003000000000
		0x3000feff	movl $7, %edi	0xbf07000000
		0x3000ff04	callq *%rax	0xffd0
		0x3000ff06	addq $-8, %rsp	0x4883c4f8
		0x3000ff0a	jmp -50575 ; ABS: 0x30003980 = StubRoutines::deoptimize	0xe9713affff
		0x3000ff0f	int3	0xcc

AZUL SYSTEMS

This is on X5690 (Westmere)

Uses SSE (128bit)

```
private int sumLoop(int[] a) {
    int sum = 0;
    for (int i = 0; i < a.length; i++) {
        sum += a[i];
    }
    return sum;
}
```

		0x3000fe81	andl $7, %r8d
		0x3000fe85	movq %rcx, %rdx
		0x3000fe88	subq %r8, %rdx
		0x3000fe8b	je -35 ; ABS: 0x3000fe6a
		0x3000fe8d	leaq 28(%rsi), %rax
		0x3000fe91	pxor %xmm0, %xmm0
		0x3000fe95	movq %rdx, %rdi
		0x3000fe98	pxor %xmm1, %xmm1
		0x3000fe9c	nopl (%rax)
20.64%	327	0x3000fea0	movdqu -16(%rax), %xmm2
35.42%	561	0x3000fea5	movdqu (%rax), %xmm3
11.30%	179	0x3000fea9	paddd %xmm2, %xmm0
13.70%	217	0x3000fead	paddd %xmm3, %xmm1
18.69%	296	0x3000feb1	addq $32, %rax
		0x3000feb5	addq $-8, %rdi
		0x3000feb9	jne -27 ; ABS: 0x3000fea0
		0x3000febb	paddd %xmm0, %xmm1
		0x3000febf	pshufd $78, %xmm1, %xmm0
		0x3000fec4	paddd %xmm1, %xmm0
		0x3000fec8	phaddd %xmm0, %xmm0
0.13%	2	0x3000fecd	movd %xmm0, %eax
		0x3000fed1	testl %r8d, %r8d
		0x3000fed4	jne -102 ; ABS: 0x3000fe70
		0x3000fed6	jmp -92 ; ABS: 0x3000fe7c
		0x3000fed8	xorl %eax, %eax
		0x3000feda	popq %rcx
		0x3000fedb	retq
		0x3000fedc	movq %rsi, (%rsp)
		0x3000fee0	movabsq $805334400, %rax
		0x3000feea	callq *%rax
		0x3000feec	movq (%rsp), %rsi

```
private int sumLoop(int[] a) {
    int sum = 0;
    for (int i = 0; i < a.length; i++) {
        sum += a[i];
    }
    return sum;
}
```

This is on E5-2690 v4 (Broadwell)

Uses AVX2 (256bit)

		0x3001269c	subq %r8, %rdx
		0x3001269f	je -47 ; ABS: 0x30012672
		0x300126a1	leaq 108(%rsi), %rax
		0x300126a5	vpxor %ymm0, %ymm0, %ymm0
		0x300126a9	movq %rdx, %rdi
		0x300126ac	vpxor %ymm1, %ymm1, %ymm1
		0x300126b0	vpxor %ymm2, %ymm2, %ymm2
		0x300126b4	vpxor %ymm3, %ymm3, %ymm3
		0x300126b8	nopl (%rax,%rax)
0.06%	1	0x300126c0	vpaddd -96(%rax), %ymm0, %ymm0
37.08%	603	0x300126c5	vpaddd -64(%rax), %ymm1, %ymm1
0.92%	15	0x300126ca	vpaddd -32(%rax), %ymm2, %ymm2
54.80%	891	0x300126cf	vpaddd (%rax), %ymm3, %ymm3
0.55%	9	0x300126d3	subq $-128, %rax
5.84%	95	0x300126d7	addq $-32, %rdi
		0x300126db	jne -29 ; ABS: 0x300126c0
		0x300126dd	vpaddd %ymm0, %ymm1, %ymm0
0.12%	2	0x300126e1	vpaddd %ymm2, %ymm0, %ymm0
		0x300126e5	vpaddd %ymm3, %ymm0, %ymm0
		0x300126e9	vextracti128 $1, %ymm0, %xmm1
0.25%	4	0x300126ef	vpaddd %ymm0, %ymm1, %ymm0
		0x300126f3	vpshufd $78, %xmm0, %xmm1
0.06%	1	0x300126f8	vpaddd %ymm0, %ymm1, %ymm0
0.12%	2	0x300126fc	vphaddd %ymm0, %ymm0, %ymm0
		0x30012701	vmovd %xmm0, %eax
		0x30012705	testl %r8d, %r8d
		0x30012708	jne -142 ; ABS: 0x30012680
		0x3001270e	jmp -134 ; ABS: 0x3001268d
		0x30012713	xorl %eax, %eax
		0x30012715	popq %rcx
		0x30012716	retq
		0x30012717	movq %rsi, (%rsp)
		0x3001271b	movabsq $805344640, %rax
		0x30012725	callq *%rax
		0x30012727	movq (%rsp), %rsi
		0x3001272b	jmp -207 ; ABS: 0x30012661

A conditional array cell addition loop

```
private void addArraysIfEven(int a[], int b[]) {
    if (a.length != b.length) {
        throw new RuntimeException("length mismatch");
    }
    for (int i = 0; i < a.length; i++) {
        if ((b[i] & 0x1) == 0) {
            a[i] += b[i];
        }
    }
}
```

AZUL SYSTEMS

```java
private void addArraysIfEven(int a[], int b[]) {
    if (a.length != b.length) {
        throw new RuntimeException("length mismatch");
    }
    for (int i = 0; i < a.length; i++) {
        if ((b[i] & 0x1) == 0) {
            a[i] += b[i];
        }
    }
}
```

Percent	Ticks	Address	Code	Opcode
		0x3(010650	subq $24, %rsp	0x4883ec18
		0x3(010654	cmpl $0, %gs:104	0x65833c256800000000
		0x3(01065d	jne 106 ; ABS: 0x300106c9	0x756a
0.01%	1	0x3(01065f	movl 8(%rsi), %eax // NPE-> 0x300106ee	0x8b4608
		0x3(010662	cmpl 8(%rdx), %eax // NPE-> 0x300106ff	0x3b4208
		0x3(010665	jne 165 ; ABS: 0x30010710	0x0f85a5000000
		0x3(01066b	testl %eax, %eax	0x85c0
		0x3(01066d	je 85 ; ABS: 0x300106c4	0x7455
		0x3(01066f	testb $1, %al	0xa801
		0x3(010671	jne 4 ; ABS: 0x30010677	0x7504
		0x3(010673	xorl %edi, %edi	0x31ff
		0x3(010675	jmp 18 ; ABS: 0x30010687	0xeb10
		0x3(010677	movl 12(%rdx), %ecx	0x8b4a0c
		0x3(01067a	testb $1, %cl	0xf6c101
		0x3(01067d	jne 3 ; ABS: 0x30010682	0x7503
		0x3(01067f	addl %ecx, 12(%rsi)	0x014e0c
		0x3(010682	movl $1, %edi	0xbf01000000
		0x3(010687	cmpl $1, %eax	0x83f801
		0x3(01068a	je 56 ; ABS: 0x300106c4	0x7438
		0x3(01068c	subq %rdi, %rax	0x4829f8
		0x3(01068f	leaq 16(%rdx,%rdi,4), %rcx	0x488d4cba10
		0x3(010694	leaq 16(%rsi,%rdi,4), %rdx	0x488d54be10
		0x3(010699	nopl (%rax)	0x0f1f8000000000
16.84%	1,286	0x3(0106a0	movl -4(%rcx), %esi	0x8b71fc
6.10%	466	0x3(0106a3	testb $1, %sil	0x40f6c601
		0x3(0106a7	jne 3 ; ABS: 0x300106ac	0x7503
7.61%	581	0x3(0106a9	addl %esi, -4(%rdx)	0x0172fc
29.41%	2,246	0x3(0106ac	movl (%rcx), %esi	0x8b31
2.25%	172	0x3(0106ae	testb $1, %sil	0x40f6c601
		0x3(0106b2	jne 2 ; ABS: 0x300106b6	0x7502
8.00%	611	0x3(0106b4	addl %esi, (%rdx)	0x0132
29.73%	2,271	0x3(0106b6	addq $8, %rcx	0x4883c108
		0x3(0106ba	addq $8, %rdx	0x4883c208
		0x3(0106be	addq $-2, %rax	0x4883c0fe
		0x3(0106c2	jne -38 ; ABS: 0x300106a0	0x75dc
0.03%	2	0x3(0106c4	addq $24, %rsp	0x4883c418
0.03%	2	0x3(0106c8	retq	0xc3
		0x3(0106c9	movq %rsi, 16(%rsp)	0x4889742410
		0x3(0106ce	movq %rdx, 8(%rsp)	0x4889542408
		0x3(0106d3	movabsq $805334400, %rax	0x48b8806d003000000000
		0x3(0106dd	callq *%rax	0xffd0
		0x3(0106df	movq 8(%rsp), %rdx	0x488b542408
		0x3(0106e4	movq 16(%rsp), %rsi	0x488b742410
		0x3(0106e9	jmp -143 ; ABS: 0x3001065f	0xe971ffffff
		0x3(0106ee	movabsq $805319872, %rax	0x48b8c034003000000000
		0x3(0106f8	movl $7, %edi	0xbf07000000
		0x3(0106fd	callq *%rax	0xffd0
		0x3(0106ff	movabsq $805319872, %rax	0x48b8c034003000000000
		0x3(010709	movl $7, %edi	0xbf07000000
		0x3(01070e	callq *%rax	0xffd0
		0x3(010710	movabsq $805319872, %rax	0x48b8c034003000000000
		0x3(01071a	movl $4294967288, %edi	0xbff8ffffff
		0x3(01071f	callq *%rax	0xffd0
		0x3(010721	addq $-8, %rsp	0x4883c4f8
		0x3(010725	jmp -52650 ; ABS: 0x30003980 = StubRoutines::deoptimize	0xe95632ffff
		0x3(01072a	int3	0xcc

AZUL SYSTEMS

Traditional JVM JITs

per-element jumps,
2 elements per iteration

```
private void addArraysIfEven(int a[], int b[]) {
    if (a.length != b.length) {
        throw new RuntimeException("length mismatch");
    }
    for (int i = 0; i < a.length; i++) {
        if ((b[i] & 0x1) == 0) {
            a[i] += b[i];
        }
    }
}
```

		0x3001067a	testb $1, %cl
		0x3001067d	jne 3 ; ABS: 0x30010682
		0x3001067f	addl %ecx, 12(%rsi)
		0x30010682	movl $1, %edi
		0x30010687	cmpl $1, %eax
		0x3001068a	je 56 ; ABS: 0x300106c4
		0x3001068c	subq %rdi, %rax
		0x3001068f	leaq 16(%rdx,%rdi,4), %rcx
		0x30010694	leaq 16(%rsi,%rdi,4), %rdx
		0x30010699	nopl (%rax)
16.84%	1,286	0x300106a0	movl -4(%rcx), %esi
6.10%	466	0x300106a3	testb $1, %sil
		0x300106a7	jne 3 ; ABS: 0x300106ac
7.61%	581	0x300106a9	addl %esi, -4(%rdx)
29.41%	2,246	0x300106ac	movl (%rcx), %esi
2.25%	172	0x300106ae	testb $1, %sil
		0x300106b2	jne 2 ; ABS: 0x300106b6
8.00%	611	0x300106b4	addl %esi, (%rdx)
29.73%	2,271	0x300106b6	addq $8, %rcx
		0x300106ba	addq $8, %rdx
		0x300106be	addq $-2, %rax
		0x300106c2	jne -36 ; ABS: 0x300106a0
0.03%	2	0x300106c4	addq $24, %rsp
0.03%	2	0x300106c8	retq
		0x300106c9	movq %rsi, 16(%rsp)
		0x300106ce	movq %rdx, 8(%rsp)
		0x300106d3	movabsq $805334400, %rax
		0x300106dd	callq *%rax
		0x300106df	movq 8(%rsp), %rdx
		0x300106e4	movq 16(%rsp), %rsi
		0x300106e9	jmp -143 ; ABS: 0x3001065f
		0x300106ee	movabsq $805319872, %rax
		0x300106f8	movl $7, %edi

```
private void addArraysIfEven(int a[], int b[]) {
    if (a.length != b.length) {
        throw new RuntimeException("length mismatch");
    }
    for (int i = 0; i < a.length; i++) {
        if ((b[i] & 0x1) == 0) {
            a[i] += b[i];
        }
    }
}
```

This is on E5-2690 v4 (Broadwell)

Vectorized with AVX2
32 elements per iteration

		0x30014548	movabsq $805389904, %rbx
		0x30014552	vpbroadcastd (%rbx), %ymm0
		0x30014557	vpxor %ymm1, %ymm1, %ymm1
		0x3001455b	movq %rdi, %rbx
		0x3001455e	nop
0.15%	4	0x30014560	vmovdqu -96(%r11), %ymm2
12.31%	320	0x30014566	vmovdqu -64(%r11), %ymm3
0.50%	13	0x3001456c	vmovdqu -32(%r11), %ymm4
2.04%	53	0x30014572	vmovdqu (%r11), %ymm5
0.31%	8	0x30014577	vpand %ymm0, %ymm2, %ymm6
4.54%	118	0x3001457b	vpand %ymm0, %ymm3, %ymm7
0.69%	18	0x3001457f	vpand %ymm0, %ymm4, %ymm8
1.35%	35	0x30014583	vpand %ymm0, %ymm5, %ymm9
0.42%	11	0x30014587	vpcmpeqd %ymm1, %ymm6, %ymm8
2.58%	67	0x3001458b	vpmaskmovd -96(%rcx), %ymm6, %ymm10
3.58%	93	0x30014591	vpcmpeqd %ymm1, %ymm7, %ymm7
2.12%	55	0x30014595	vpmaskmovd -64(%rcx), %ymm7, %ymm11
12.12%	315	0x3001459b	vpcmpeqd %ymm1, %ymm8, %ymm8
1.50%	39	0x3001459f	vpmaskmovd -32(%rcx), %ymm8, %ymm12
3.69%	96	0x300145a5	vpcmpeqd %ymm1, %ymm9, %ymm9
1.81%	47	0x300145a9	vpmaskmovd (%rcx), %ymm9, %ymm13
12.27%	319	0x300145ae	vpaddd %ymm2, %ymm10, %ymm2
0.58%	15	0x300145b2	vpaddd %ymm3, %ymm11, %ymm3
0.19%	5	0x300145b6	vpaddd %ymm4, %ymm12, %ymm4
0.58%	15	0x300145ba	vpaddd %ymm5, %ymm13, %ymm5
3.27%	85	0x300145be	vpmaskmovd %ymm2, %ymm6, -96(%rcx)
7.15%	186	0x300145c4	vpmaskmovd %ymm3, %ymm7, -64(%rcx)
13.65%	355	0x300145ca	vpmaskmovd %ymm4, %ymm8, -32(%rcx)
4.58%	119	0x300145d0	vpmaskmovd %ymm5, %ymm9, (%rcx)
6.81%	177	0x300145d5	subq $-128, %r11
0.69%	18	0x300145d9	subq $-128, %rcx
0.31%	8	0x300145dd	addq $-32, %rbx
		0x300145e1	jne -135 ; ABS: 0x30014560
		0x300145e7	testl %r9d, %r9d
		0x300145ea	jne -356 ; ABS: 0x3001448c
		0x300145f0	jmp -233 ; ABS: 0x3001450c

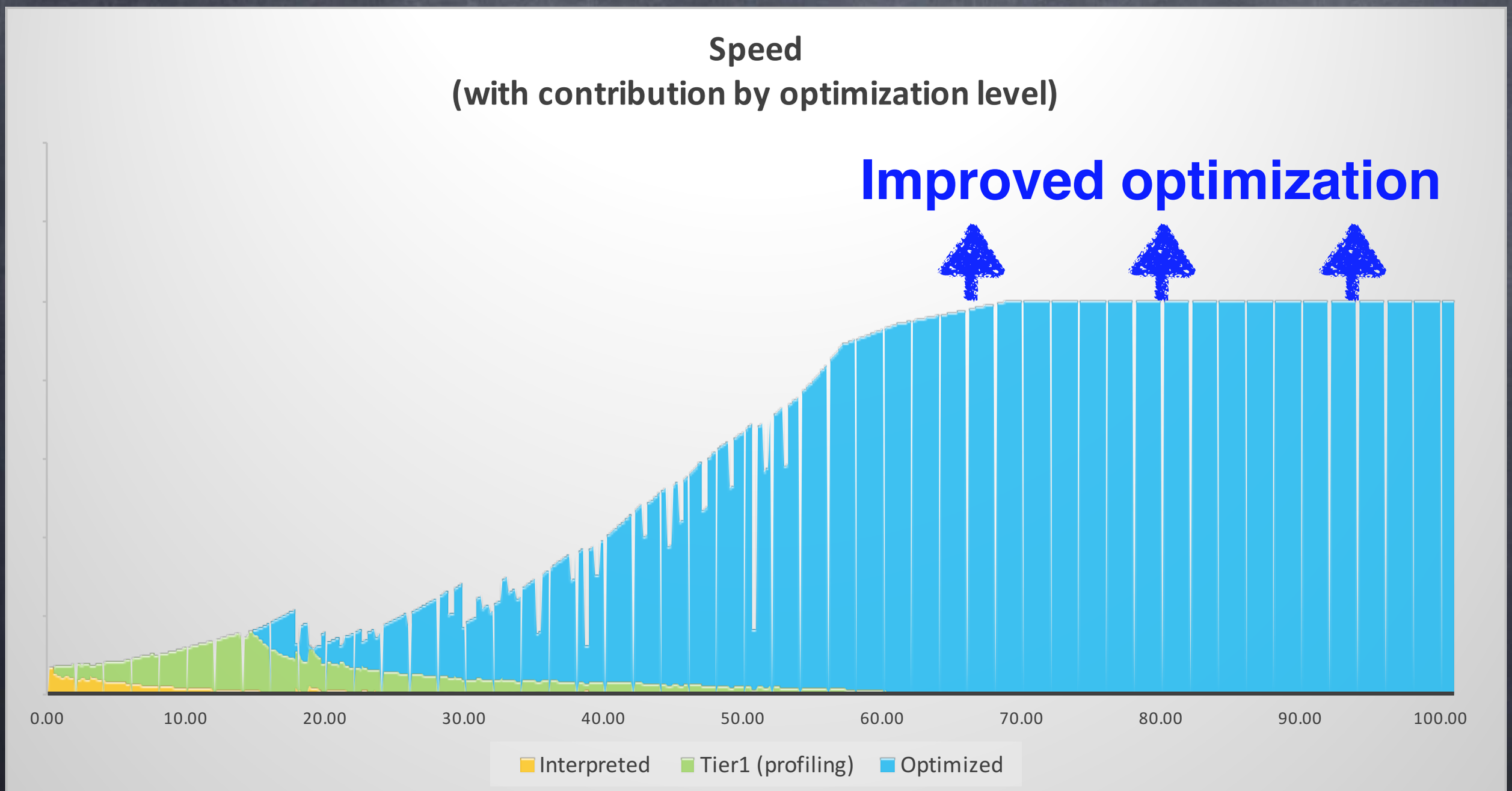
```
private void addArraysIfEven(int a[], int b[]) {
    if (a.length != b.length) {
        throw new RuntimeException("length mismatch");
    }
    for (int i = 0; i < a.length; i++) {
        if ((b[i] & 0x1) == 0) {
            a[i] += b[i];
        }
    }
}
```

This is on Skylake SP

Vectorized with AVX512
64 elements per iteration

		0x3001ab4a	jb -206 ; ABS: 0x3001aa82
		0x3001ab50	movl %eax, %ecx
		0x3001ab52	andl $2147483584, %ecx
0.02%	1	0x3001ab58	leaq 204(%rdx), %r8
		0x3001ab5f	leaq 204(%rsi), %rdi
		0x3001ab66	movabsq $805416064, %rbx
		0x3001ab70	vpbroadcastd (%rbx), %zmm0
0.04%	2	0x3001ab76	movq %rcx, %r9
		0x3001ab79	nopl (%rax)
0.46%	24	0x3001ab80	vmovdqu32 -192(%r8), %zmm1
3.46%	179	0x3001ab87	vmovdqu32 -128(%r8), %zmm2
3.52%	182	0x3001ab8e	vmovdqu32 -64(%r8), %zmm3
4.95%	256	0x3001ab95	vmovdqu32 (%r8), %zmm4
12.73%	658	0x3001ab9b	vptestnmd %zmm0, %zmm1, %k1
1.99%	103	0x3001aba1	vptestnmd %zmm0, %zmm2, %k2
1.53%	79	0x3001aba7	vptestnmd %zmm0, %zmm3, %k3
2.69%	139	0x3001abad	vmovdqu32 -192(%rdi), %zmm5 {%k1} {z}
8.38%	433	0x3001abb4	vmovdqu32 -128(%rdi), %zmm6 {%k2} {z}
11.13%	575	0x3001abbb	vmovdqu32 -64(%rdi), %zmm7 {%k3} {z}
10.51%	543	0x3001abc2	vptestnmd %zmm0, %zmm4, %k4
1.76%	91	0x3001abc8	vmovdqu32 (%rdi), %zmm8 {%k4} {z}
18.25%	943	0x3001abce	vpaddd %zmm1, %zmm5, %zmm1
0.97%	50	0x3001abd4	vpaddd %zmm2, %zmm6, %zmm2
0.91%	47	0x3001abda	vpaddd %zmm3, %zmm7, %zmm3
0.46%	24	0x3001abe0	vmovdqu32 %zmm1, -192(%rdi) {%k1}
2.26%	117	0x3001abe7	vmovdqu32 %zmm2, -128(%rdi) {%k2}
0.66%	34	0x3001abee	vmovdqu32 %zmm3, -64(%rdi) {%k3}
1.10%	57	0x3001abf5	vpaddd %zmm4, %zmm8, %zmm1
0.64%	33	0x3001abfb	vmovdqu32 %zmm1, (%rdi) {%k4}
9.85%	514	0x3001ac01	addq $256, %r8
0.64%	33	0x3001ac08	addq $256, %rdi
0.85%	44	0x3001ac0f	addq $-64, %r9
		0x3001ac13	jne -153 ; ABS: 0x3001ab80
		0x3001ac19	cmpq %rax, %rcx
		0x3001ac1c	jne -414 ; ABS: 0x3001aa84

Better JIT'ing is basically about speed



Compiler Stuff

Some simple compiler tricks

Code can be reordered…

```
int doMath(int x, int y, int z) {
  int a = x + y;
  int b = x - y;
  int c = z + x;
  return a + b;
}
```

Can be reordered to:

```
int doMath(int x, int y, int z) {
  int c = z + x;
  int b = x - y;
  int a = x + y;
  return a + b;
}
```


AZUL SYSTEMS

Dead code can be removed

```
int doMath(int x, int y, int z) {
  int a = x + y;
  int b = x - y;
  int c = z + x;
  return a + b;
}
```

Can be reduced to:

```
int doMath(int x, int y, int z) {
  int a = x + y;
  int b = x - y;
  return a + b;
}
```


AZUL SYSTEMS®

Values can be propagated

```
int doMath(int x, int y, int z) {
  int a = x + y;
  int b = x - y;
  int c = z + x;
  return a + b;
}
```

Can be reduced to:

```
int doMath(int x, int y, int z) {
  return x + y + x - y;
}
```


AZUL SYSTEMS®

Math can be simplified

```
int doMath(int x, int y, int z) {
  int a = x + y;
  int b = x - y;
  int c = z + x;
  return a + b;
}
```

Can be reduced to:

```
int doMath(int x, int y, int z) {
  return x + x;
}
```


AZUL SYSTEMS®

Some more compiler tricks

propagation can affect flow

constants can be propagated to pre-compute results:

```
int computeBias() {
  int bias, val = 5;
  if (val > 10) {
      bias = computeComplicatedBias(val);
  else {
      bias = 1;
  }
  return bias;
}
```

Can be reduced to:

```
int computeBias() {
  return 1;
}
```


AZUL SYSTEMS

Reads can be cached

```
class Point { int x, y; }

int distanceRatio(Point a) {
   int distanceTo = a.x - start;
   int distanceAfter = end - a.x;
   return distanceTo/distanceAfter;
}
```

Is (semantically) the same as

```
int distanceRatio(Point a) {
   int x = a.x;
   int distanceTo = x - start;
   int distanceAfter = end - x;
   return distanceTo/distanceAfter;
}
```

Reads can be cached

```
class Trigger { boolean flag; }

void loopUntilFlagSet(Tigger a) {
  while (!a.flag) {
    loopcount++;
  }
}
```

Is the same as:

```
void loopUntilFlagSet(Object a) {
  boolean flagIsSet = a.flag;
  while (!flagIsSet) {
    loopcount++;
  }
}
```

That's what volatile is for...

AZUL SYSTEMS

Writes can be eliminated

Intermediate values might never be visible

```
void updateDistance(Point a) {
  int distance = 100;
  a.x = distance;
  a.x = distance * 2;
  a.x = distance * 3;
}
```

Is the same as

```
void updateDistance(Point a) {
  a.x = 300;
}
```


AZUL SYSTEMS

Writes can be eliminated

Intermediate values might never be visible

```
void updateDistance(SomeObject a) {
  a.visibleValue = 0;
  for (int i = 0; i < 1000000; i++) {
    a.internalValue = i;
  }
  a.visibleValue = a.internalValue;
}
```

Is the same as

```
void updateDistance(SomeObject a) {
  a.internalValue = 1000000;
  a.visibleValue = 1000000;
}
```


AZUL SYSTEMS

Inlining…

```
public class Thing {
   private int x;
   public final int getX() { return x };
}
…
myX = thing.getX();
```

Is the same as

```
Class Thing {
   int x;
}
…
myX = thing.x;
```


AZUL SYSTEMS

Inlining is very powerful

inlining exposes other optimizations

```
int computeBias(int val) {
  int bias;
  if (val > 10) {
      bias = computeComplicatedBias(val);
  else {
      bias = 1;
  }
  return bias;
}
...
myBias = computeBias(5);
```

Can be reduced to:

```
myBias = 1;
```


AZUL SYSTEMS

A uBenchmark sidetrack

A simple loop uBenchmark (0)

```
public void loopUbench0(int count) {
    long sum = 0;
    for (int i = 0; i < count; i++) {
        sum++;
    }
}
```

Turns out this is "really fast"

As in: when count = 1,000,000 we complete
~500,000,000 calls per second
(for 5,000,000,000,000,000 iterations/sec)

AZUL SYSTEMS

A simple loop uBenchmark (1)

```
public void loopUbench1(int count) {
    long sum = 0;
    for (int i = 0; i < count; i++) {
        sum += i;
    }
}
```

Still "impossibly fast"

It's all "provably dead code".
Compiler translates the method to a no-op

A simple loop uBenchmark (2)

```
public long loopUbench2(int count) {
    long sum = 0;
    for (int i = 0; i < count; i++) {
        sum++;
    }
    return sum;
}
```

Better?

No. Still "impossibly fast".

Compiler returns count. No loop.

A simple loop uBenchmark (3)

```
public long loopUbench3(int count) {
    long sum = 0;
    for (int i = 0; i < count; i++) {
        sum += i;
    }
    return sum;
}
```

Better?

Depends. On HotSpot and Zing C2, yes.

But Zing's new Falcon compiler is smart enough to recognize arithmetic series

A simple loop uBenchmark (4)

```
public long loopUbench4(int count) {
    long sum = 0;
    for (int i = 0; i < count; i++) {
        sum *= i;
    }
    return sum;
}
```

How about this?

Zing's Falcon will even figure out this one.
(it returns zero)

AZUL SYSTEMS

A simple loop uBenchmark (5)

```
public long loopUbench5(int count) {
    long sum = 0;
    for (int i = 0; i < count; i++) {
        sum += i + ((i - 3) & 0x7);
    }
    return sum;
}
```

Seems to be complicated enough to defeat *current* compilers...

uBenchmarking Takeaways

- uBenchmarking is "hard". As in "very tricky"
- You may not be measuring what you think you are
- "Trickiness" can change over time, between versions
- Sanity check EVERYTHING
- Use jmh
- Use jmh
- Use jmh
- And even then, suspect everything

Back to compiler stuff

Speculative compiler tricks

JIT compilers can do things that static compilers can have a hard time with…

AZUL SYSTEMS

Untaken path example

"Never taken" paths can be optimized away with benefits:

```
int computeMagnitude(int val) {
  if (val > 10) {
      bias = computeBias(val);
  else {
      bias = 1;
  }
  return Math.log10(bias + 99);
}
```

When all values so far were <= 10 , could be compiled to:

```
int computeMagnitude(int val) {
  if (val > 10) uncommonTrap();
  return 2;
}
```

Implicit Null Check example

All field and array access in Java is null checked

```
x = foo.x;
```

is (in equivalent required machine code):

```
if (foo == null)
    throw new NullPointerException();
x = foo.x;
```

But compiler can "hope" for non-nulls, and handle SEGV

```
<Point where later SEGV will appear to throw>
x = foo.x;
```

This is faster *IF* no nulls are encountered…

Class Hierarchy Analysis (CHA)

- Can perform global analysis on currently loaded code
- Deduce stuff about inheritance, method overrides, etc.
- Can make optimization decisions based on assumptions
- Re-evaluate assumptions when loading new classes
- Throw away code that conflicts with assumptions before class loading makes them invalid

Inlining works without "final"

```
public class Animal {
   private int color;
   public int getColor() { return color };
}
…
myColor = animal.getColor();
```

Is the same as

```
Class Animal {
   int color;
}
…
myColor = animal.color;
```

THIS (CHA) is why Java field accessors are free & clean

As long as only one implementer of getColor() exists

AZUL SYSTEMS

Inlining monomorphic sites

```
public class Animal {
   private int color;
   public int getColor() { return color };
}
…
myColor = animal.getColor();
```

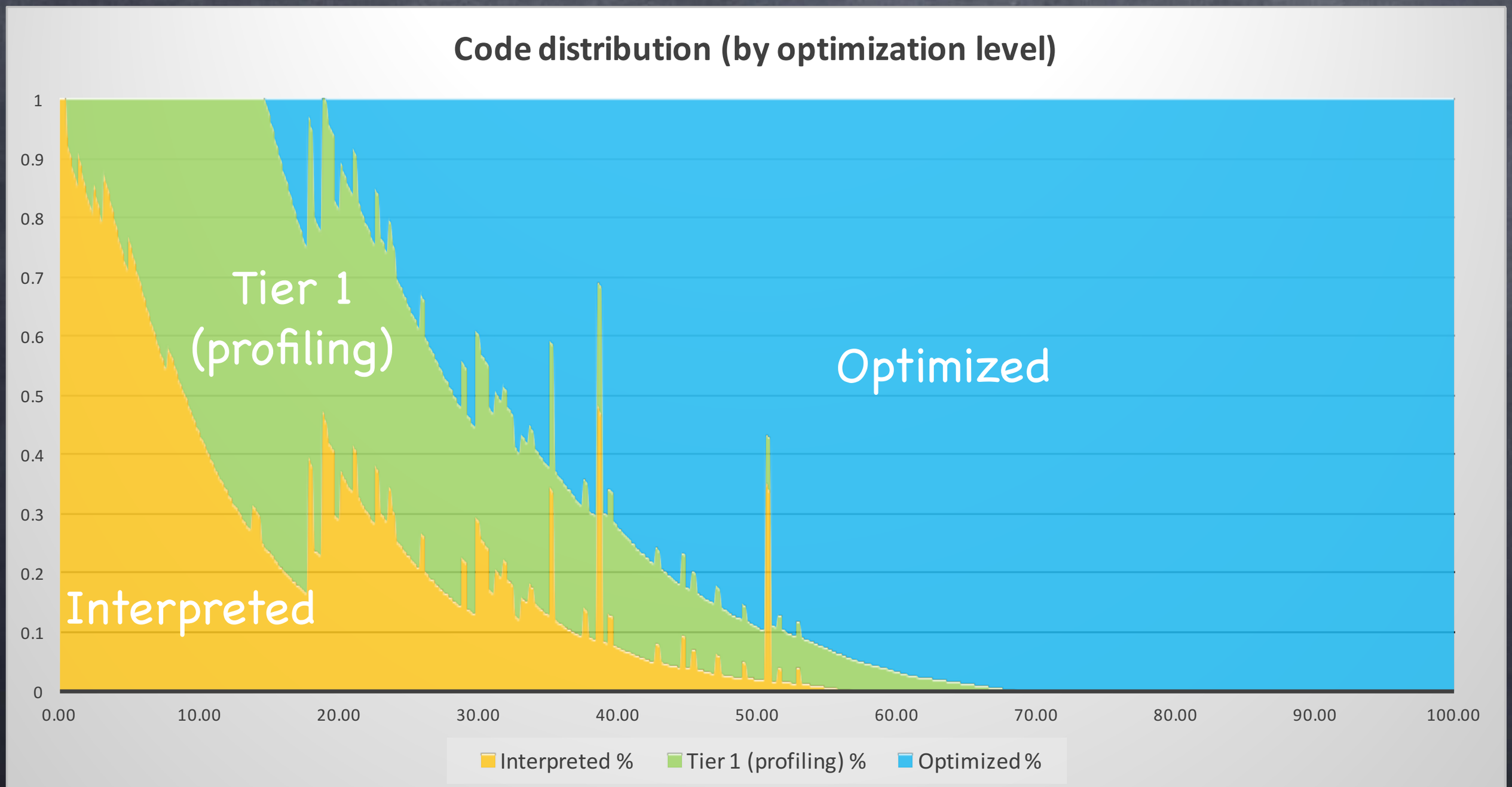
Can be converted to:

```
…
if (animal.type != Dog) uncommonTrap();
myColor = animal.color;
```

Even if we have multiple conflicting implementors…

AZUL SYSTEMS

Deoptimization



Deoptimization: Adaptive compilation is... adaptive

- Micro-benchmarking is a black art
- So is the art of the Warmup
  - Running code long enough to compile is just the start...
- Deoptimizations can occur at any time
  - often occur after you *think* the code is warmed up.
- Many potential causes

AZUL SYSTEMS

Warmup often doesn't cut it...

- Common Example:
  - Trading system wants to have the first trade be fast
  - So run 20,000 "fake" messages through the system to warm up
  - let JIT compilers optimize, learn, and deopt before actual trades
- But...
  - Code is written to do different things "if this is a fake message"
  - e.g. "Don't send to the exchange if this is a fake message"
- What really happens
  - JITs optimize for fake path, including speculatively assuming "fake"
  - First real message through causes a deopt...

Market Open

Java at Market Open

Java's "Just In Time" Reality

- Starts slow, learns fast
- Lazy loading & initialization
- Aggressively optimized for the common case
- (temporarily) Reverts to slower execution to adapt

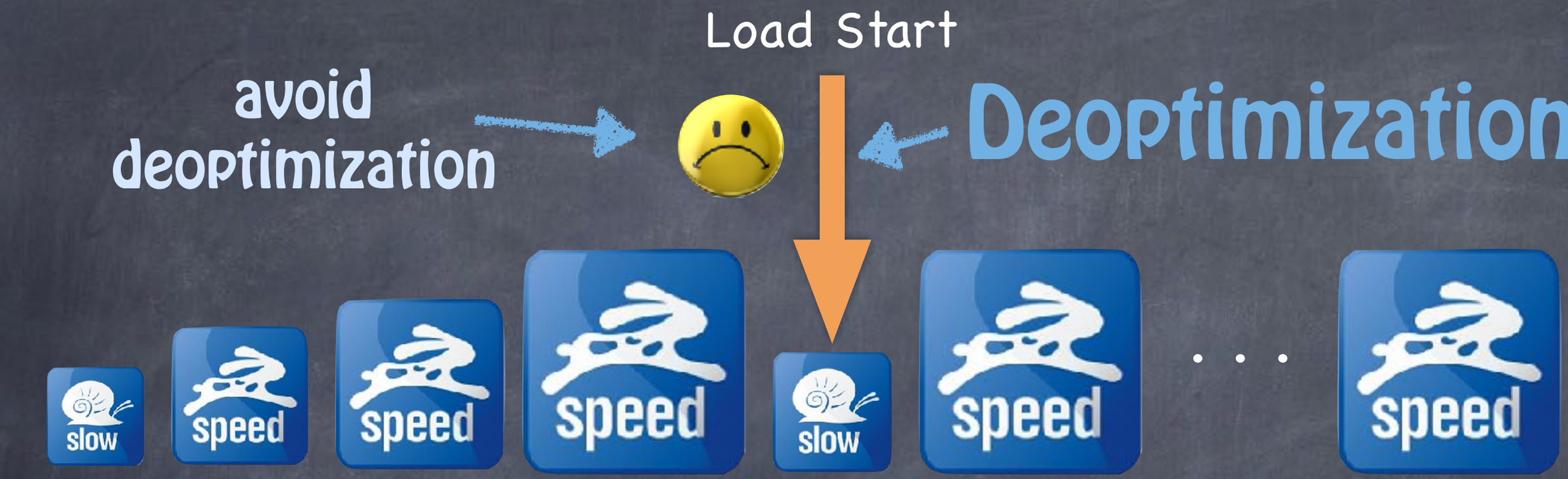
Warmup

Deoptimization

AZUL SYSTEMS

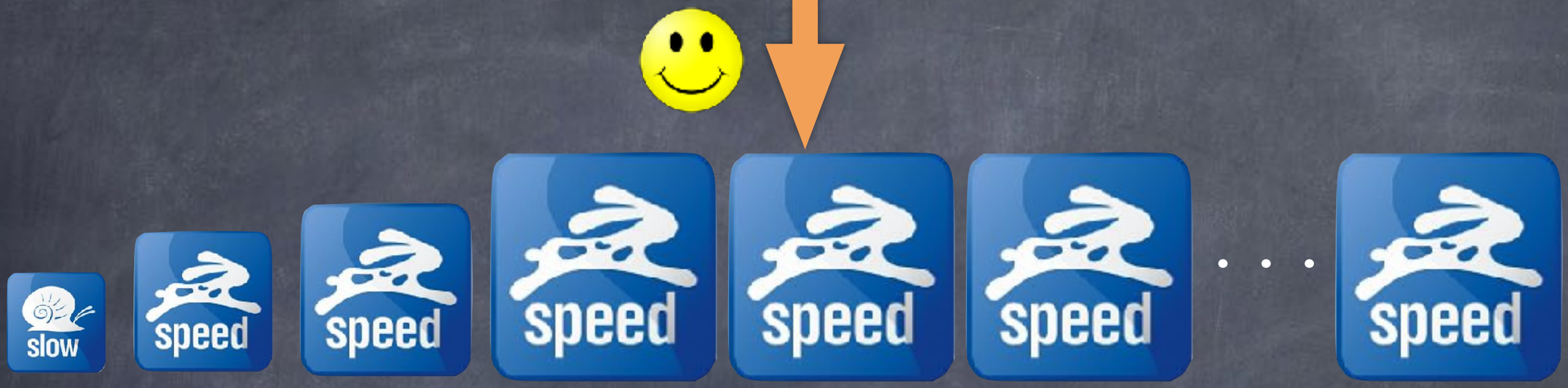
Logging and "replaying" optimizations

- Log optimization information
  - Record ongoing optimization decisions and stats
  - Record optimization dependencies
  - Establish "stable optimization state" at end of previous run
- Read prior logs at startup
  - "Prime" JVM with knowledge of prior stable optimizations
  - Apply optimizations as their dependencies get resolved
- Build workflow to promote confidence
  - Let you know if/when all optimizations have been applied
  - If some optimization haven't been applied, let you know why...



Java at "Load Start"

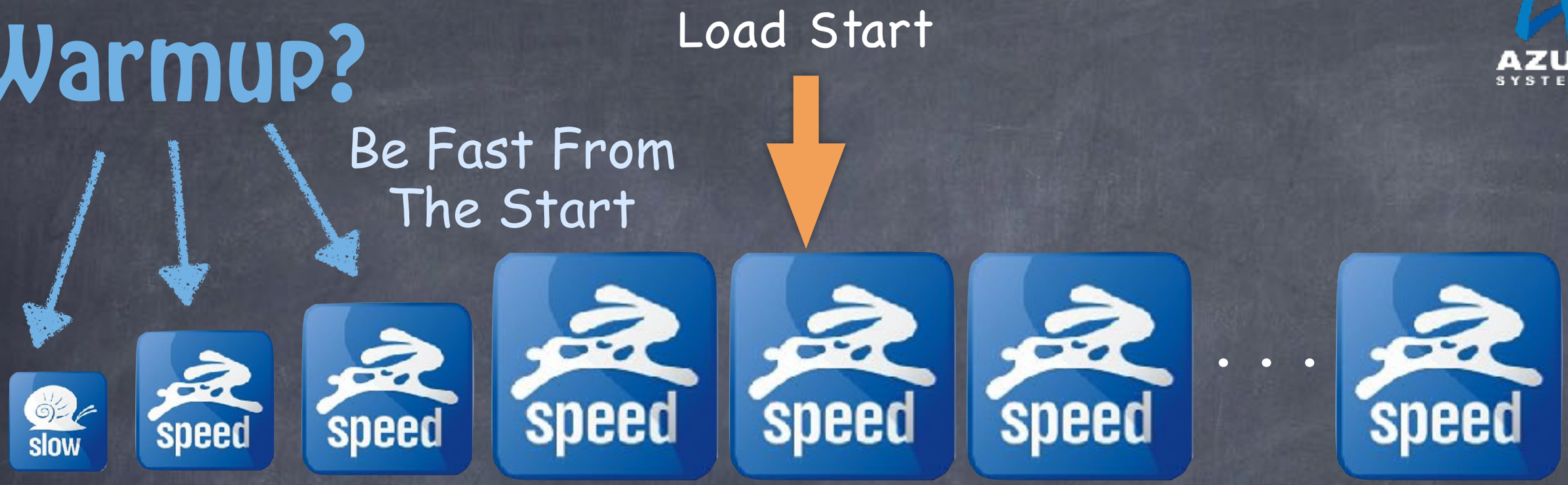
Load Start



Java at "Load Start"

With de-optimization avoided

Warmup?



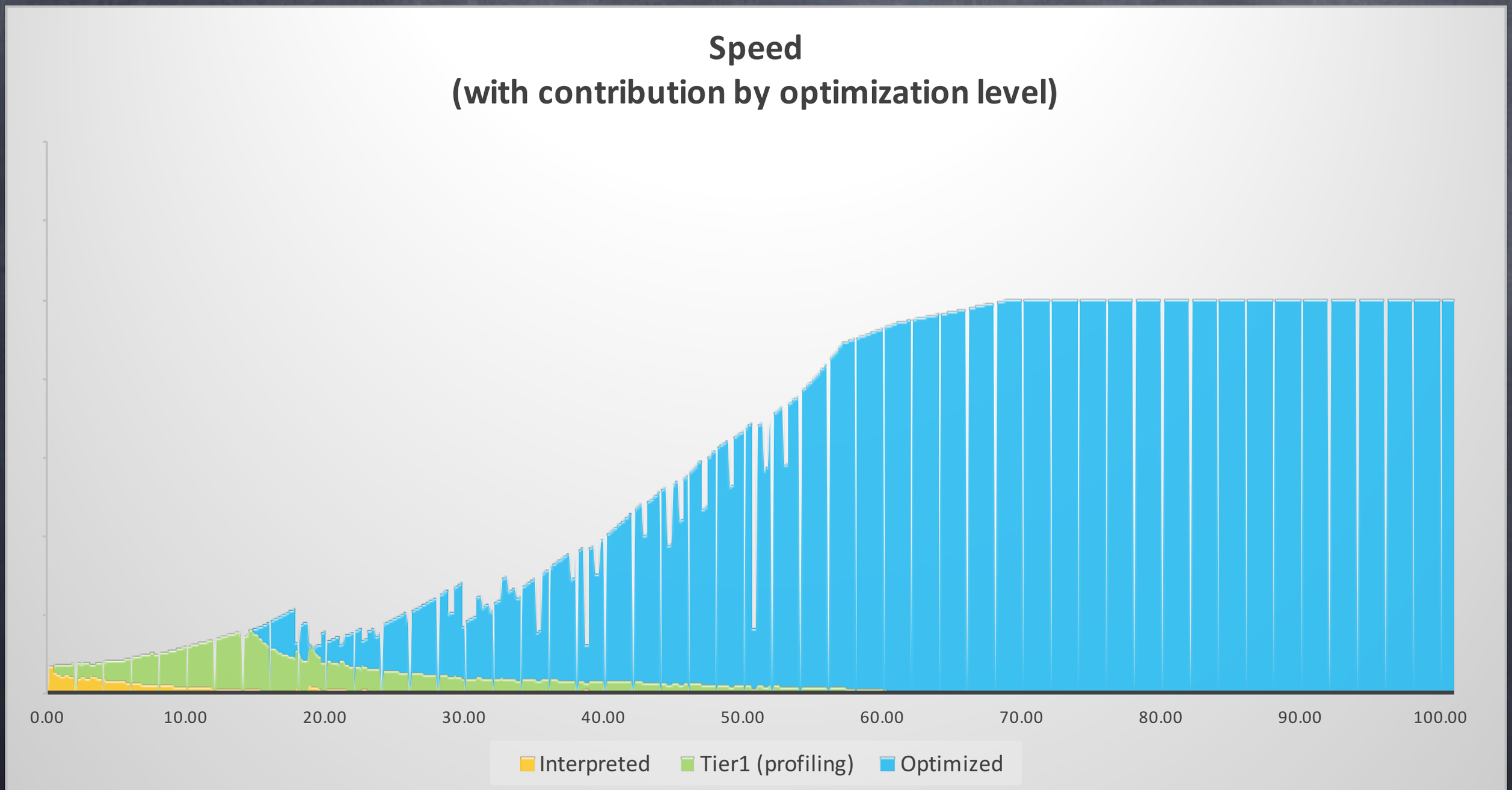
Java at "Load Start"

Load Start

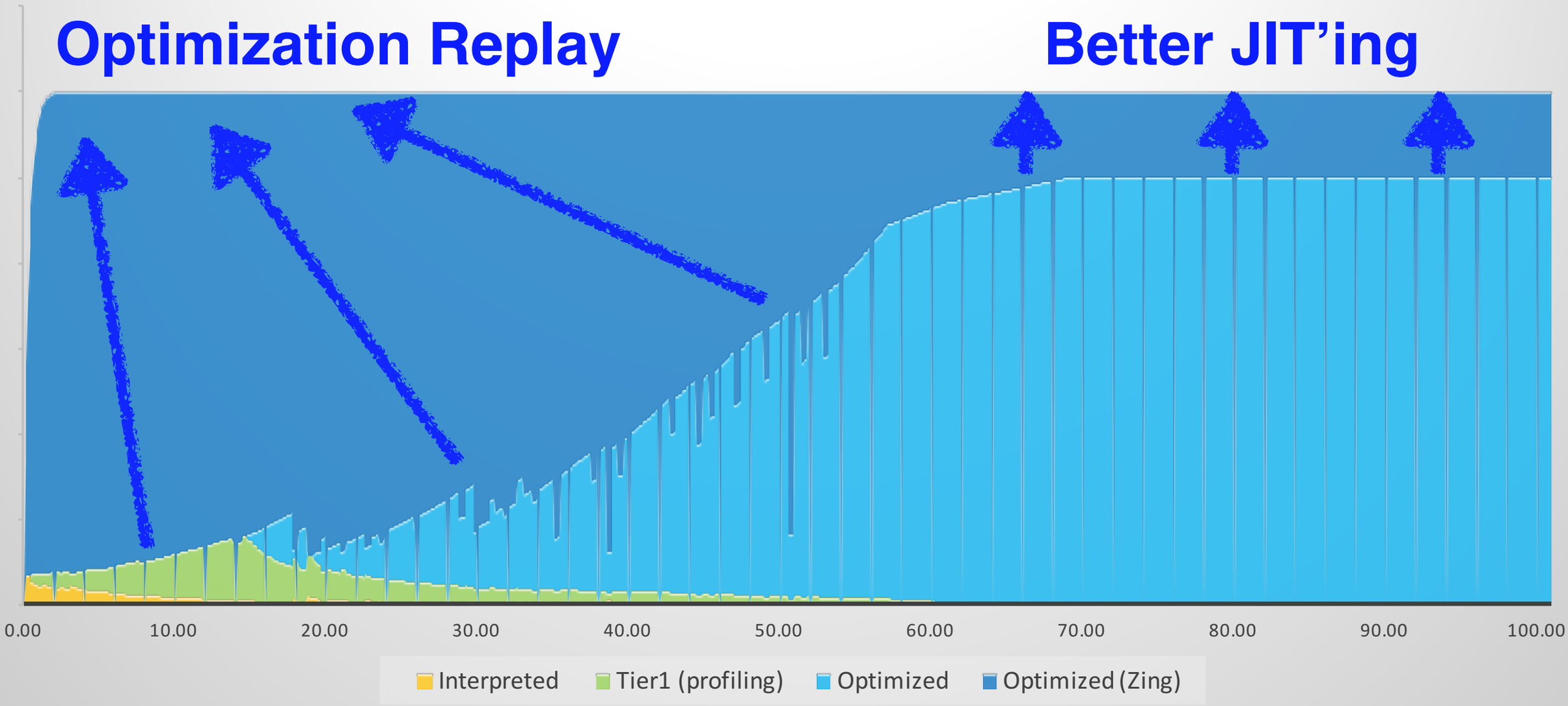
Java at "Load Start"

With pre-loading of prior optimizations

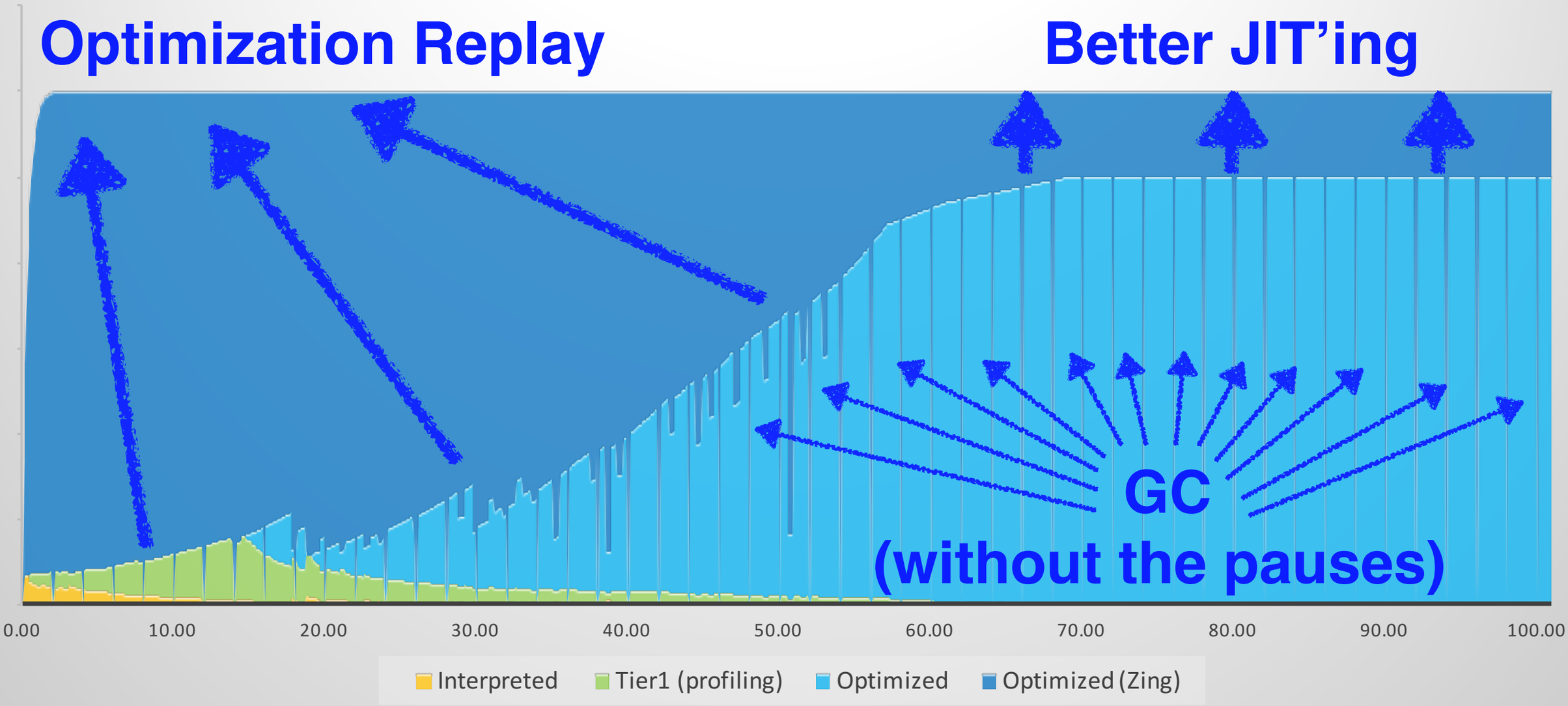
Speed improvements



Speed
(with contribution by optimization level)



Speed
(with contribution by optimization level)

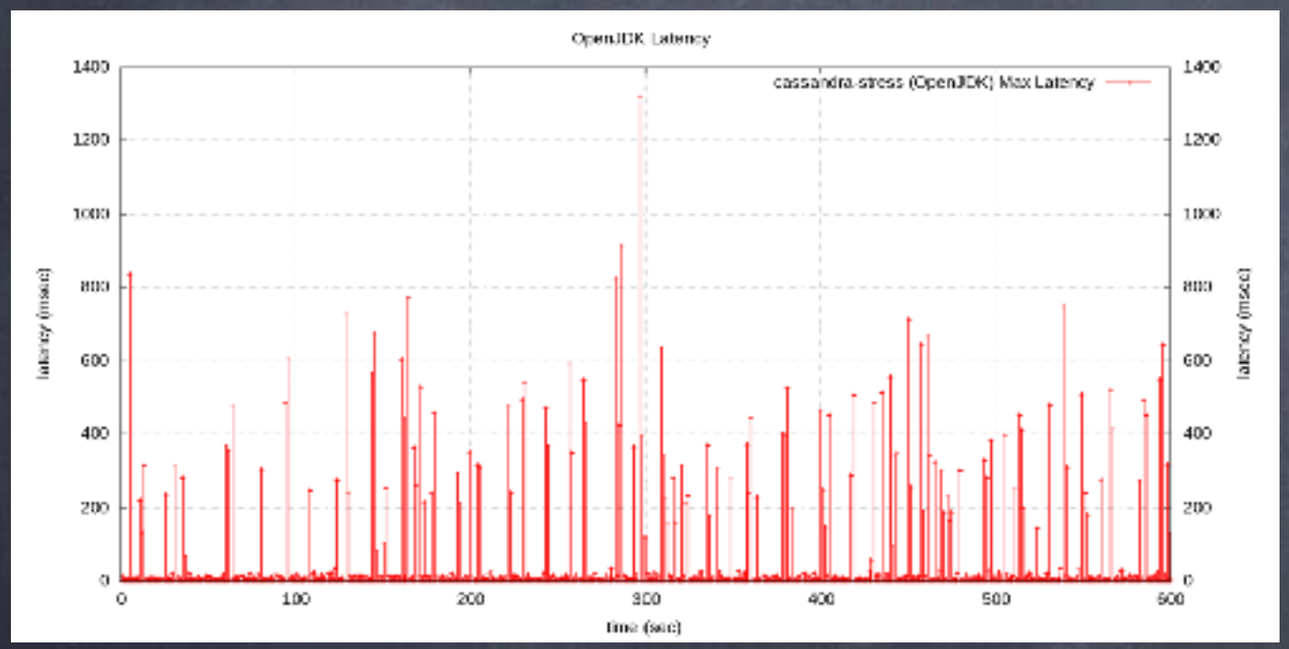


C4 Garbage Collector

ELIMINATES Garbage Collection as a concern for enterprise applications

A simple visual summary

This is <Your App> on HotSpot



This is <Your App> on Zing

Any Questions?

AZUL SYSTEMS

GC Tuning

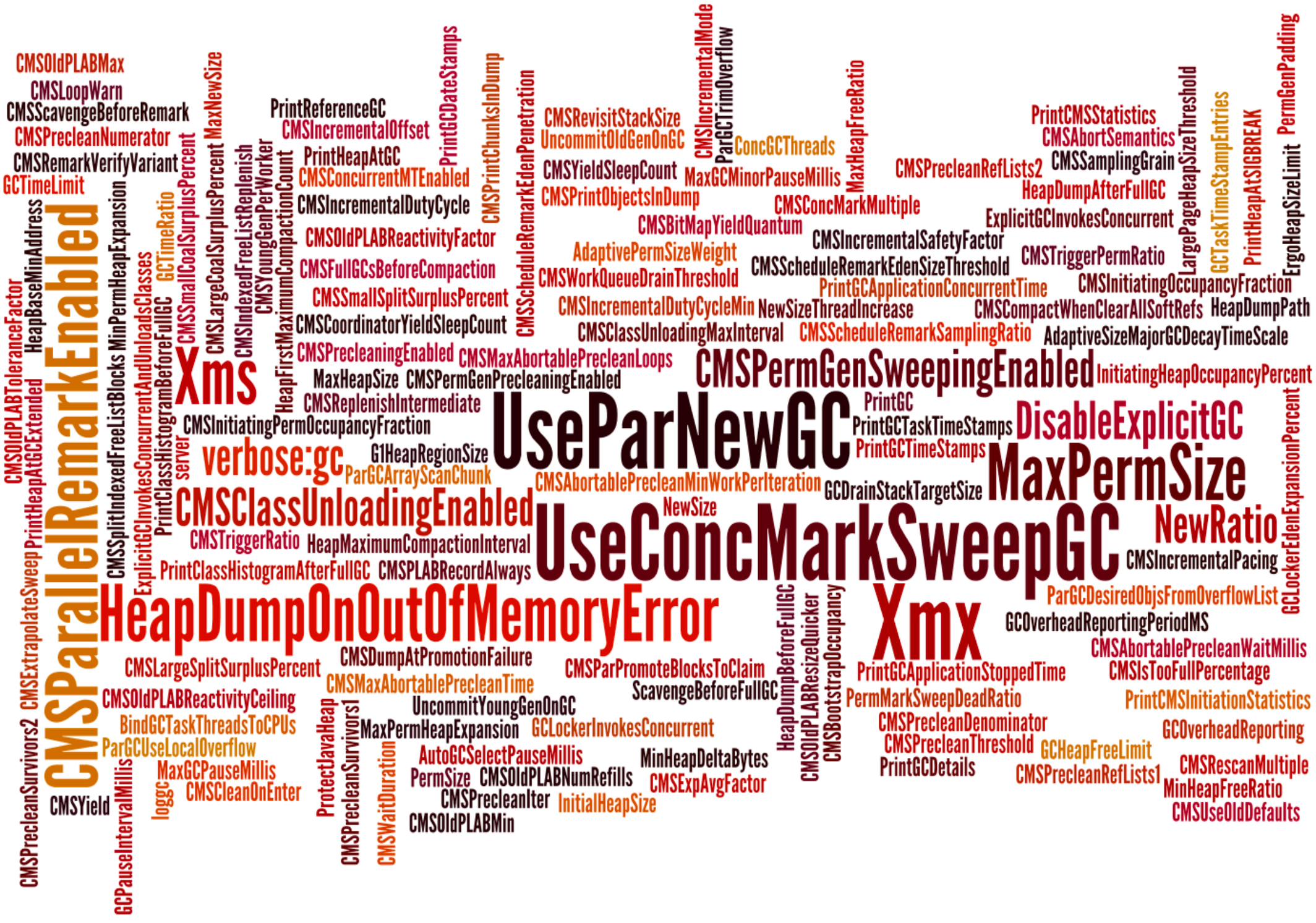
Java GC tuning is "hard"...

Examples of actual command line GC tuning parameters:

```
Java -Xmx12g -XX:MaxPermSize=64M -XX:PermSize=32M -XX:MaxNewSize=2g
        -XX:NewSize=1g -XX:SurvivorRatio=128 -XX:+UseParNewGC
        -XX:+UseConcMarkSweepGC -XX:MaxTenuringThreshold=0
        -XX:CMSInitiatingOccupancyFraction=60 -XX:+CMSParallelRemarkEnabled
        -XX:+UseCMSInitiatingOccupancyOnly -XX:ParallelGCThreads=12
        -XX:LargePageSizeInBytes=256m ...

Java -Xms8g -Xmx8g -Xmn2g -XX:PermSize=64M -XX:MaxPermSize=256M
        -XX:-OmitStackTraceInFastThrow -XX:SurvivorRatio=2 -XX:-UseAdaptiveSizePolicy
        -XX:+UseConcMarkSweepGC -XX:+CMSConcurrentMTEnabled
        -XX:+CMSParallelRemarkEnabled -XX:+CMSParallelSurvivorRemarkEnabled
        -XX:CMSMaxAbortablePrecleanTime=10000 -XX:+UseCMSInitiatingOccupancyOnly
        -XX:CMSInitiatingOccupancyFraction=63 -XX:+UseParNewGC -Xnoclassgc ...
```

A few more GC tuning flags



Source: Word Cloud created by Frank Pavageau in his Devoxx FR 2012 presentation titled “Death by Pauses”

The complete guide to modern GC tuning

java -Xmx40g

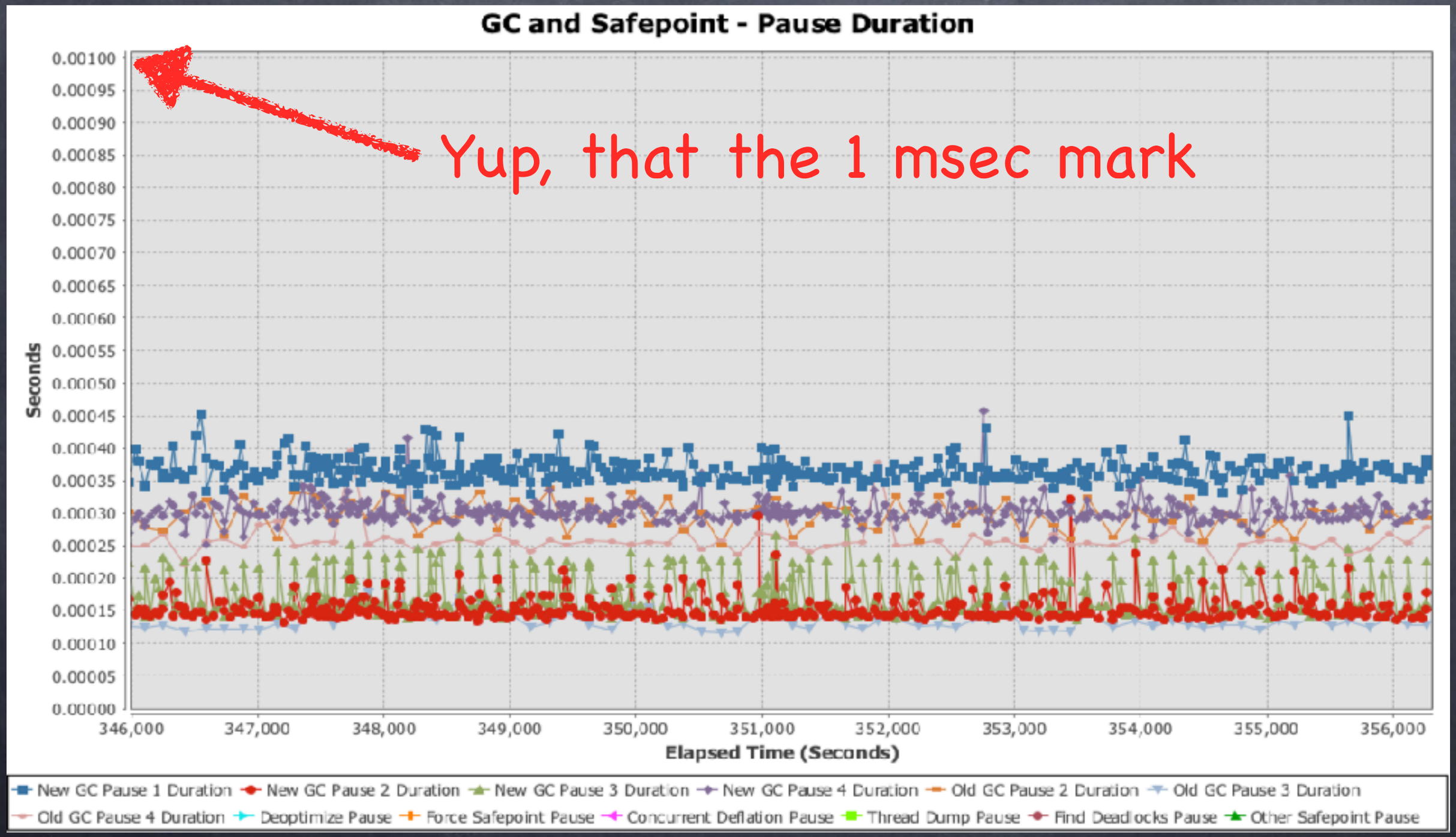
java -Xmx20g

java -Xmx10g

java -Xmx5g

AZUL SYSTEMS

Cassandra under heavy load, Intel E5-2690 v4 server

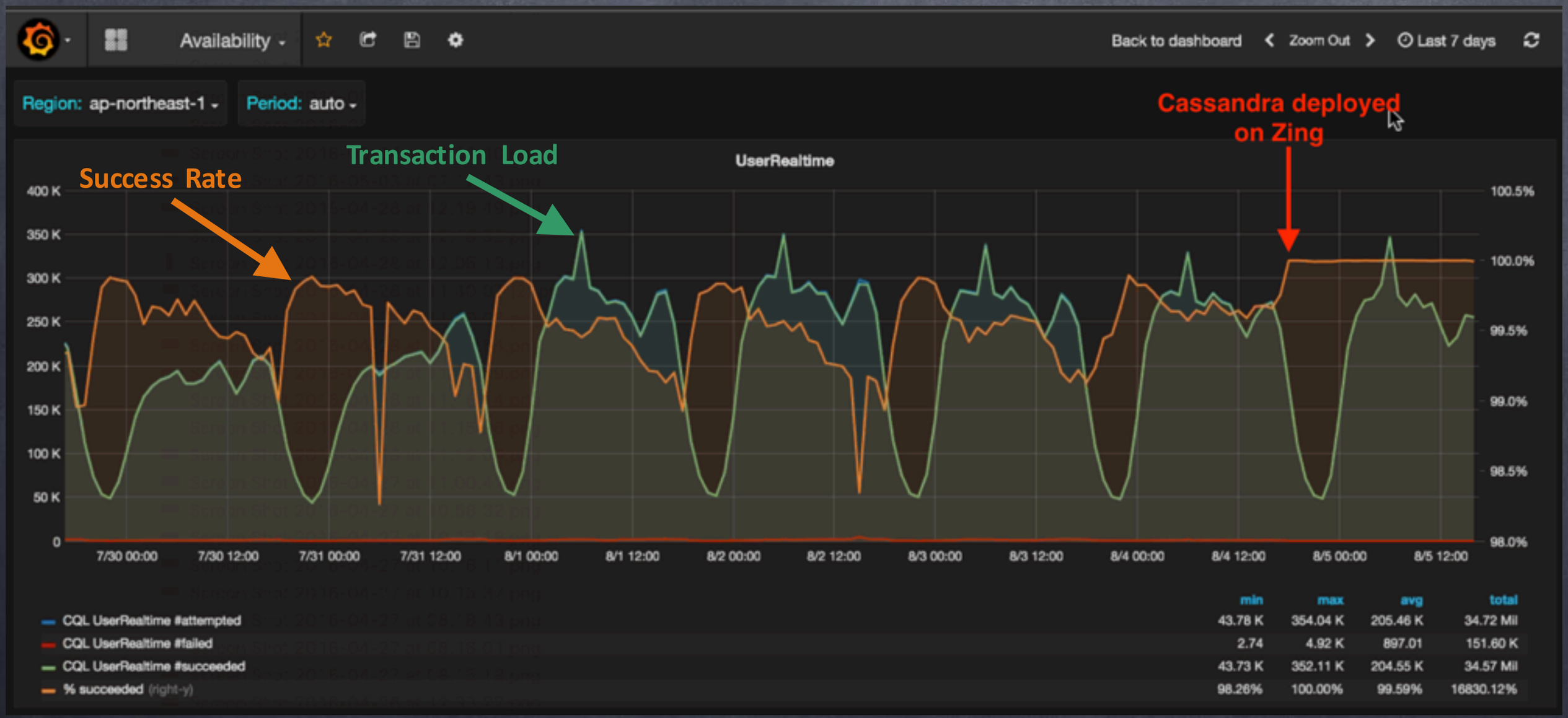


AZUL SYSTEMS

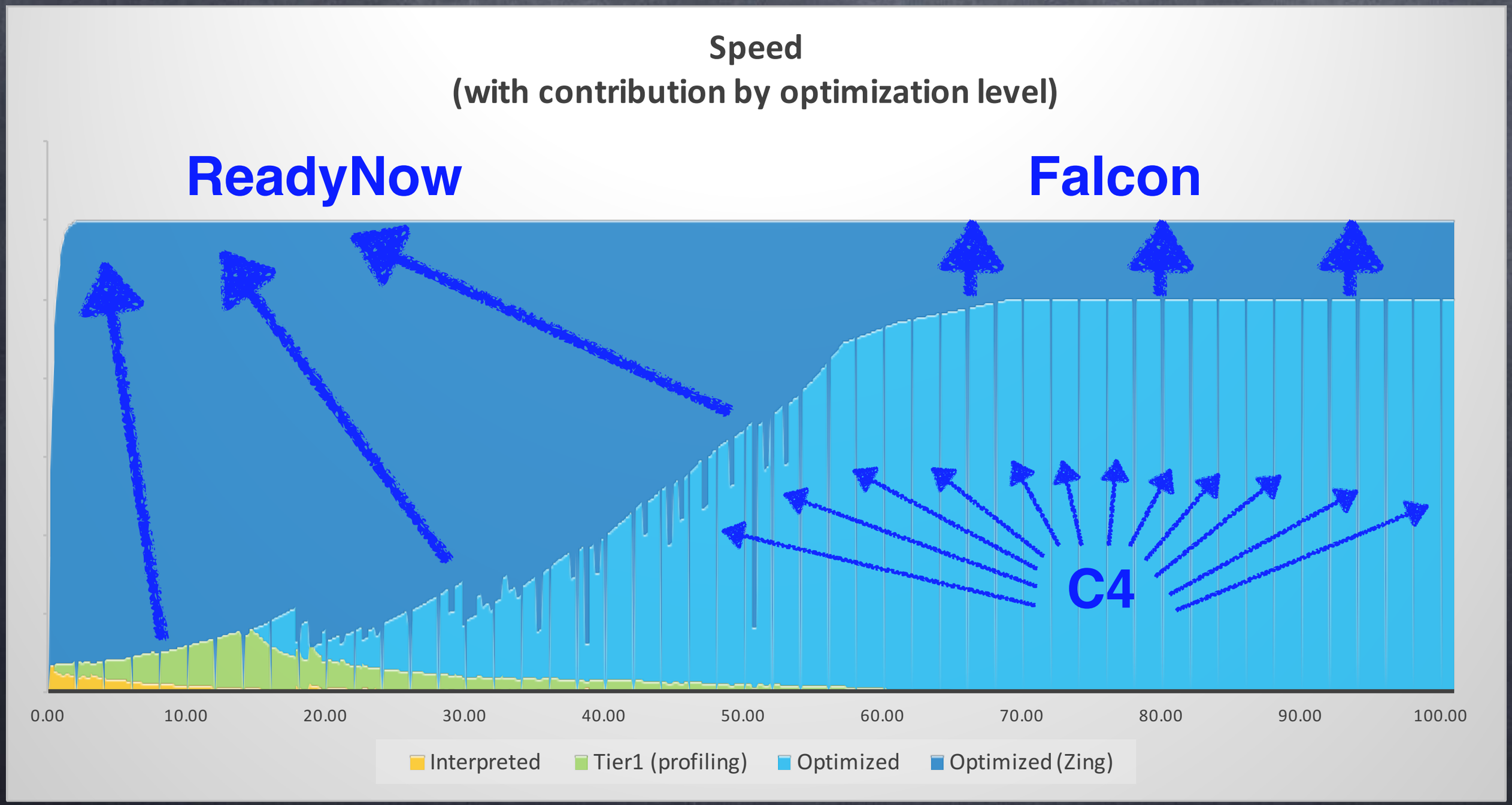
Warning: results may be too good

A practical real-world example: Improve Digital (Video Advertising)

- Cassandra cluster running on 6x AWS i3.2xlarge
  - Approx. 80/20 write/read split
  - Data read and written with quorum consistency
  - 6 client machines sending requests collocated in the same AZs
- SLA requirements for read operations:
  - 20ms at 99.9%
  - 50ms at 99.99%
  - 100ms at 99.998% (not a typo, last 9 hard to maintain on AWS)
- HotSpot w/G1: can maintain ~4K TPS before SLA breach
- Zing: can maintain ~21K TPS before SLA breach
- QED...



Start Fast, Go Fast, Stay Fast



Q & A

@giltene http://www.azul.com

http://stuff-gil-says.blogspot.com

http://latencytipoftheday.blogspot.com