ISA Aging: A x86 case study

WIVOSCA 2013

Bruno Lopes, Rafael Auler, Rodolfo Azevedo, Edson Borin
University of Campinas, Brasil
Institute of Computing
{blopes,rauler,rodolfo,edson}@ic.unicamp.br

Introduction

- Despite large amount of available **main memory**
- **Code compression** importance is still high
- Instruction **cache misses** impact performance
- IA-32 CISC (**x86**) used to be compact

Introduction

ISA Evolution

- CISC IA-32 code compaction does **not scale** with the addition of new features
- Addition of new instructions: instruction length **harms code compaction**
- Mode exchange (e.g. IA-32e and ARM/Thumb2): **increases the hardware complexity**

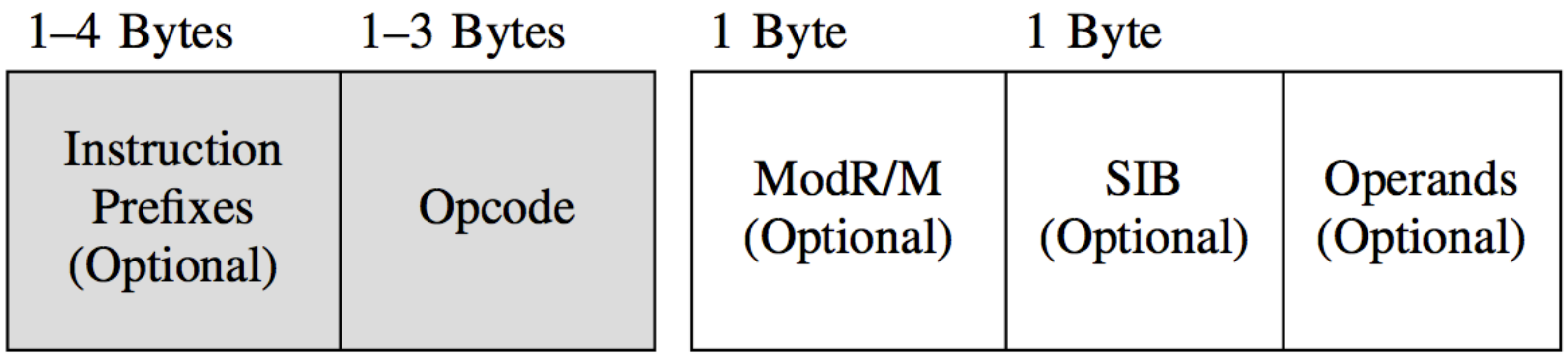
Introduction

Historical Analysis

- Study the IA-32 evolution over time
  - Opcode **usage and deprecation**
- Raise discussion about opcode utilization and **backward compatibility**

The x86 instruction set

- Intel 8086 family, **variable-length** format
- **Operation code**: opcode + other bits to uniquely identify an instruction



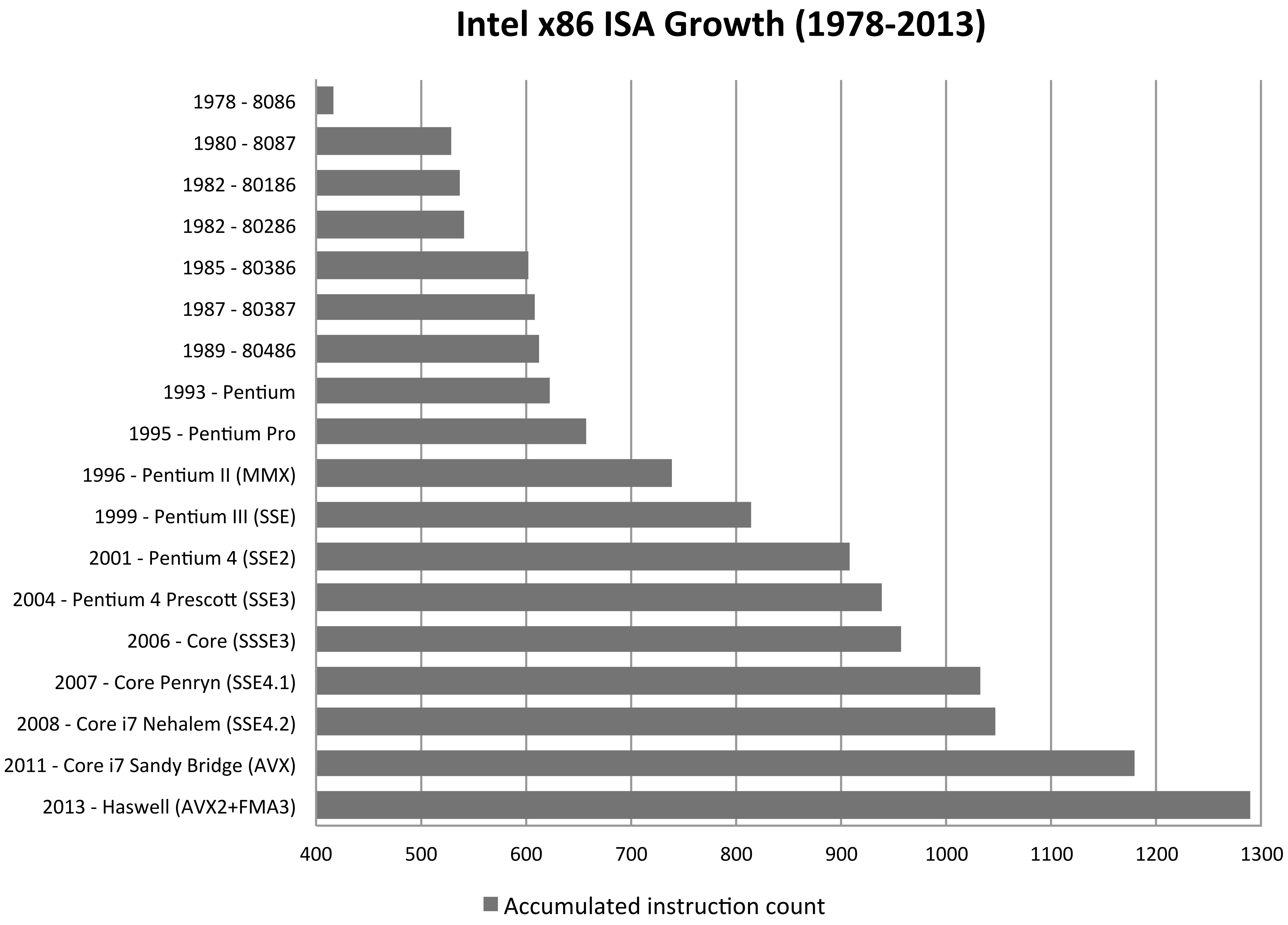
1–4 Bytes	1–3 Bytes	1 Byte	1 Byte	
Instruction Prefixes (Optional)	Opcode	ModR/M (Optional)	SIB (Optional)	Operands (Optional)

Total size cannot exceed 15 bytes

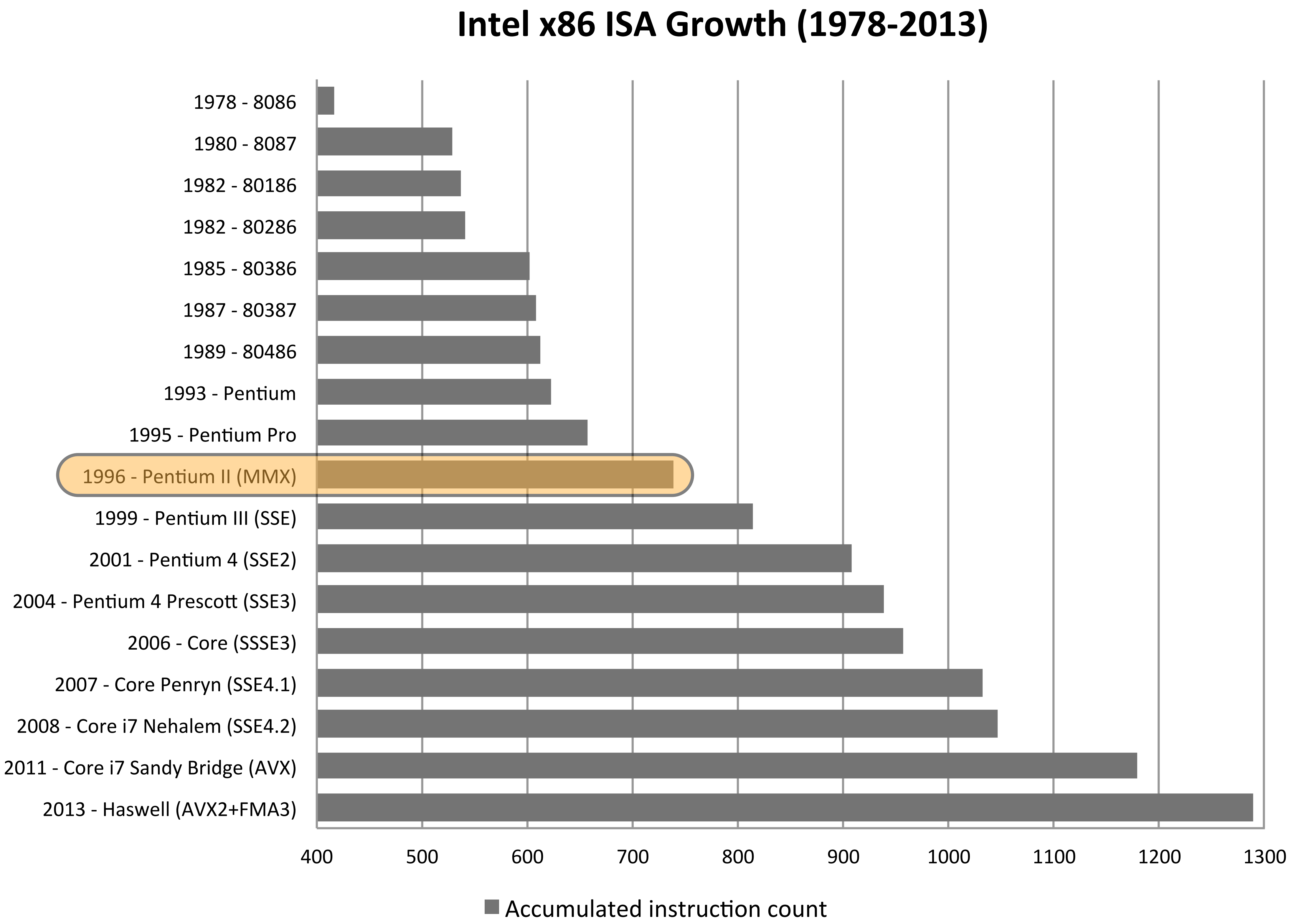
The x86 instruction set

- 16-bit 8086 (1978): **~400** instructions
- Haswell (2013): **~1300** instructions
- Multimedia instructions has the complexity and size of an entire new ISA

The x86 instruction set



The x86 instruction set

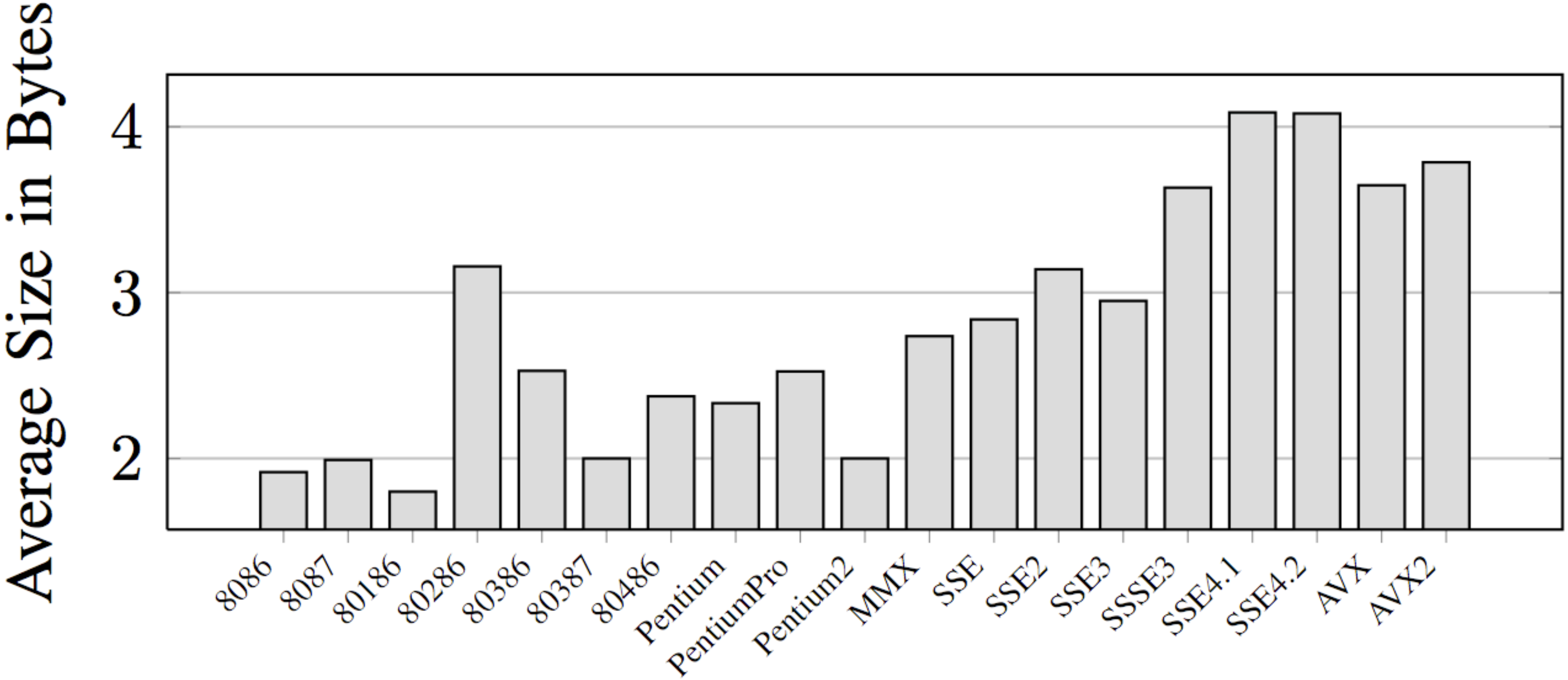


The x86 instruction set

- Avoids backward **compatibility breakage**, with new operation codes to hold new functionality
- More bytes per instruction: average number of operation code bytes changed from **2.7 to 4 bytes**

The x86 instruction set

Average instruction operation code size for each x86 feature



Methodology

- Selected **32-bit** x86 software environment for each selected year: **Home and office** SW to improve coverage

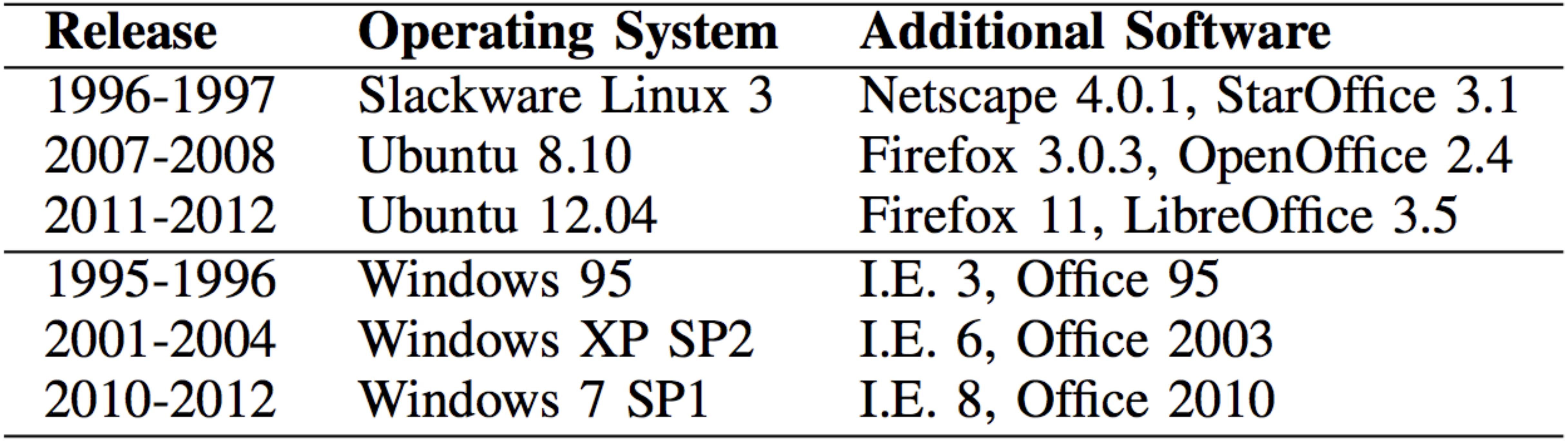
- Static frequencies of x86 instructions of different types and their evolution in time both in **Windows and Linux desktops**

Methodology

- Static analysis uses a crawler, analyzing entire virtual machines disks for executable files.
- Found x86 instructions are catalogued using disassemblers libraries:
  - Agner's **object file converter**
  - **Bochs** disassembler library

Analyzed SW

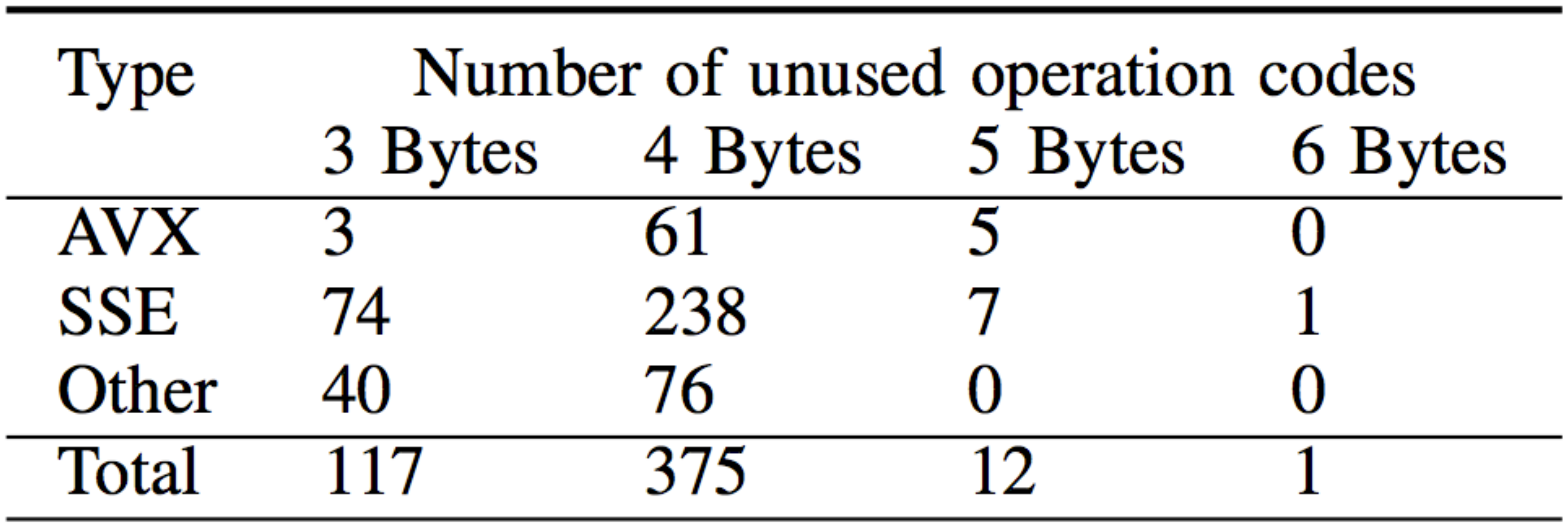
Software systems analyzed, each with its own virtual machine



Release	Operating System	Additional Software
1996-1997	Slackware Linux 3	Netscape 4.0.1, StarOffice 3.1
2007-2008	Ubuntu 8.10	Firefox 3.0.3, OpenOffice 2.4
2011-2012	Ubuntu 12.04	Firefox 11, LibreOffice 3.5
1995-1996	Windows 95	I.E. 3, Office 95
2001-2004	Windows XP SP2	I.E. 6, Office 2003
2010-2012	Windows 7 SP1	I.E. 8, Office 2010

Unused Instructions

- 505 unused operation codes in all disks (**30%** of all 32-bit operation codes)

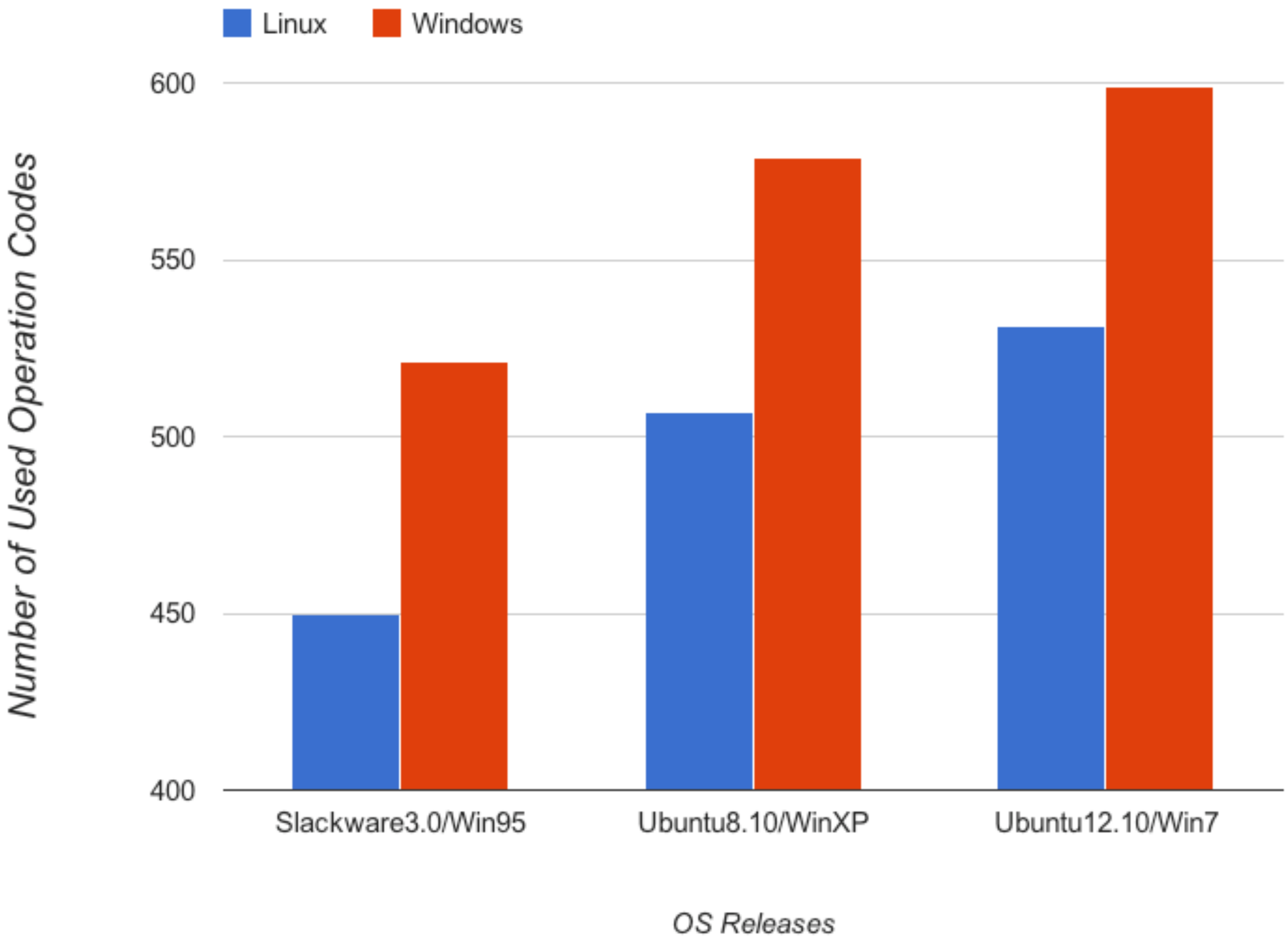


Type	Number of unused operation codes			
	3 Bytes	4 Bytes	5 Bytes	6 Bytes
AVX	3	61	5	0
SSE	74	238	7	1
Other	40	76	0	0
Total	117	375	12	1

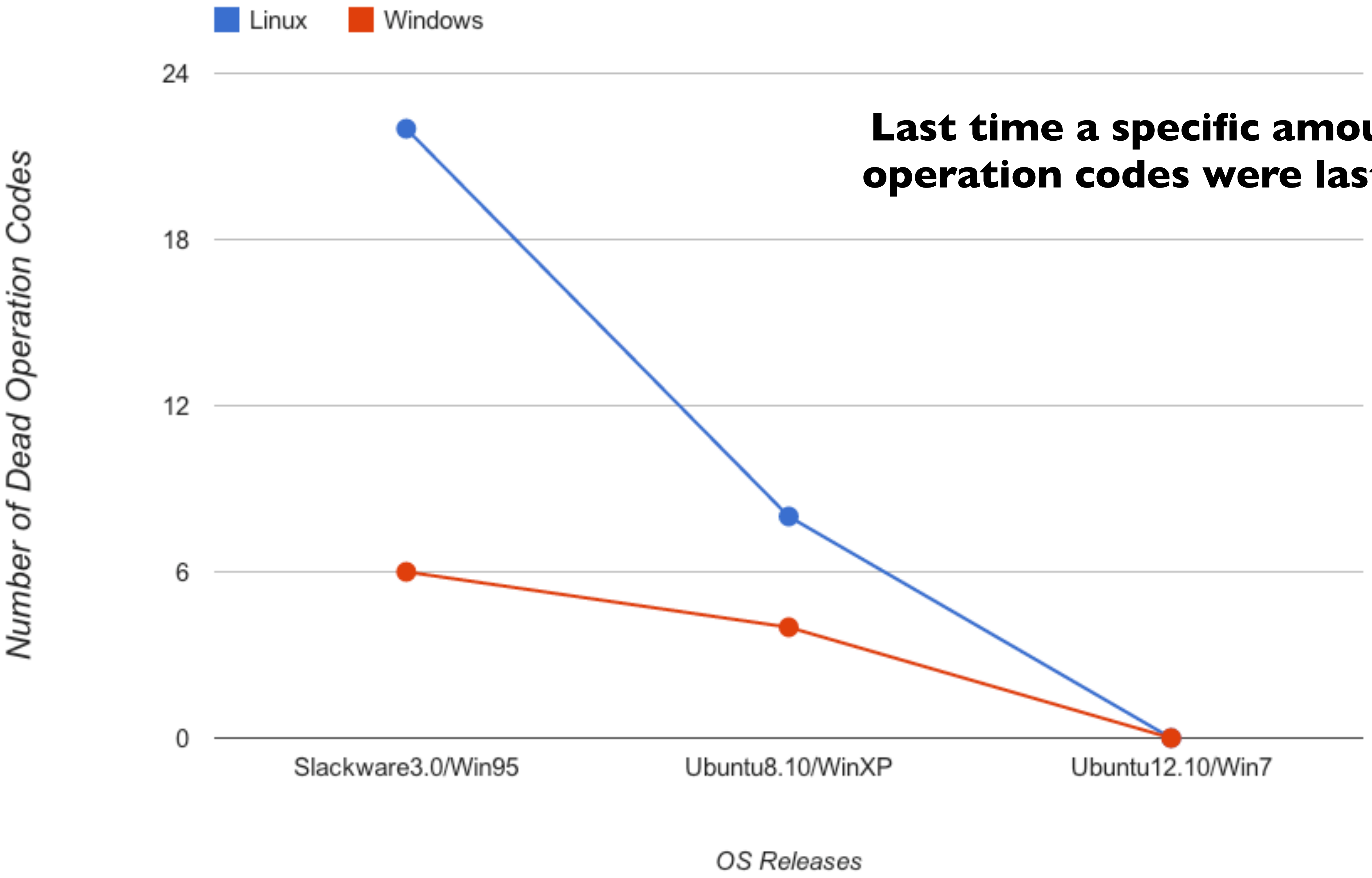
Number of unused operation codes by size.

There were no unused 1 and 2 bytes operation codes

Used Instructions

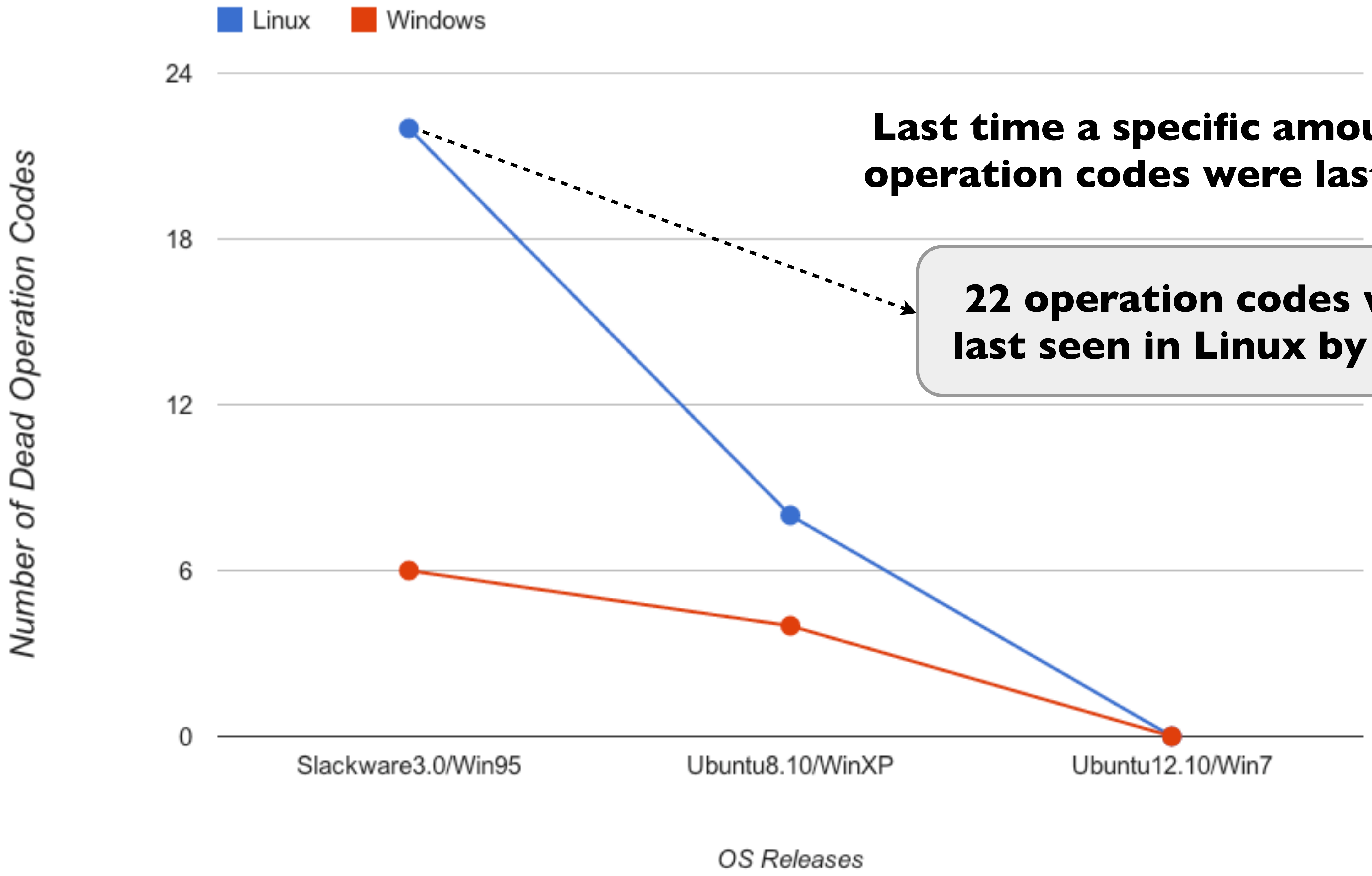


Dead Instructions



Last time a specific amount of operation codes were last seen

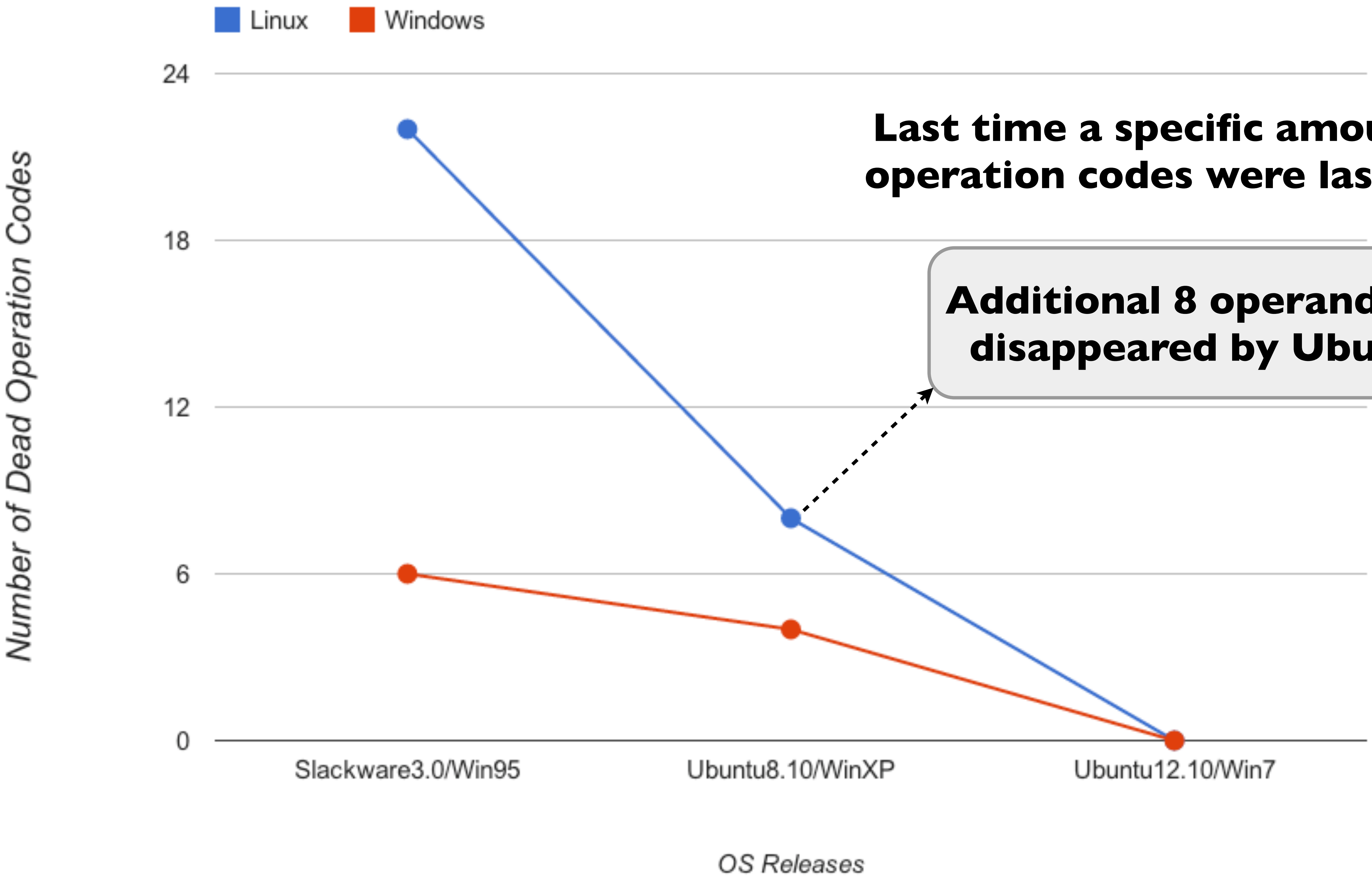
Dead Instructions



Last time a specific amount of operation codes were last seen

22 operation codes were last seen in Linux by 1996

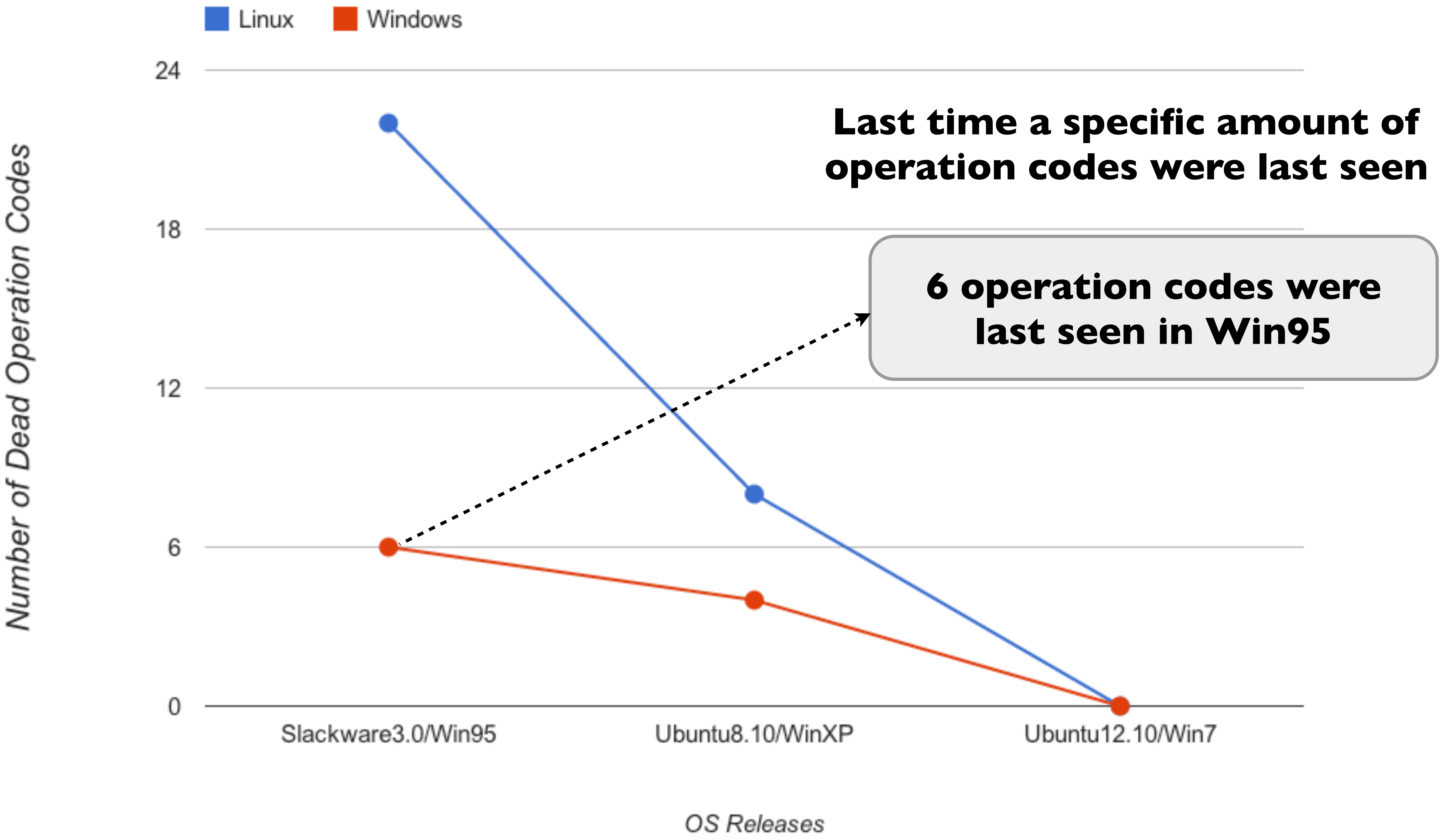
Dead Instructions



Last time a specific amount of operation codes were last seen

Additional 8 operand codes disappeared by Ubuntu 8

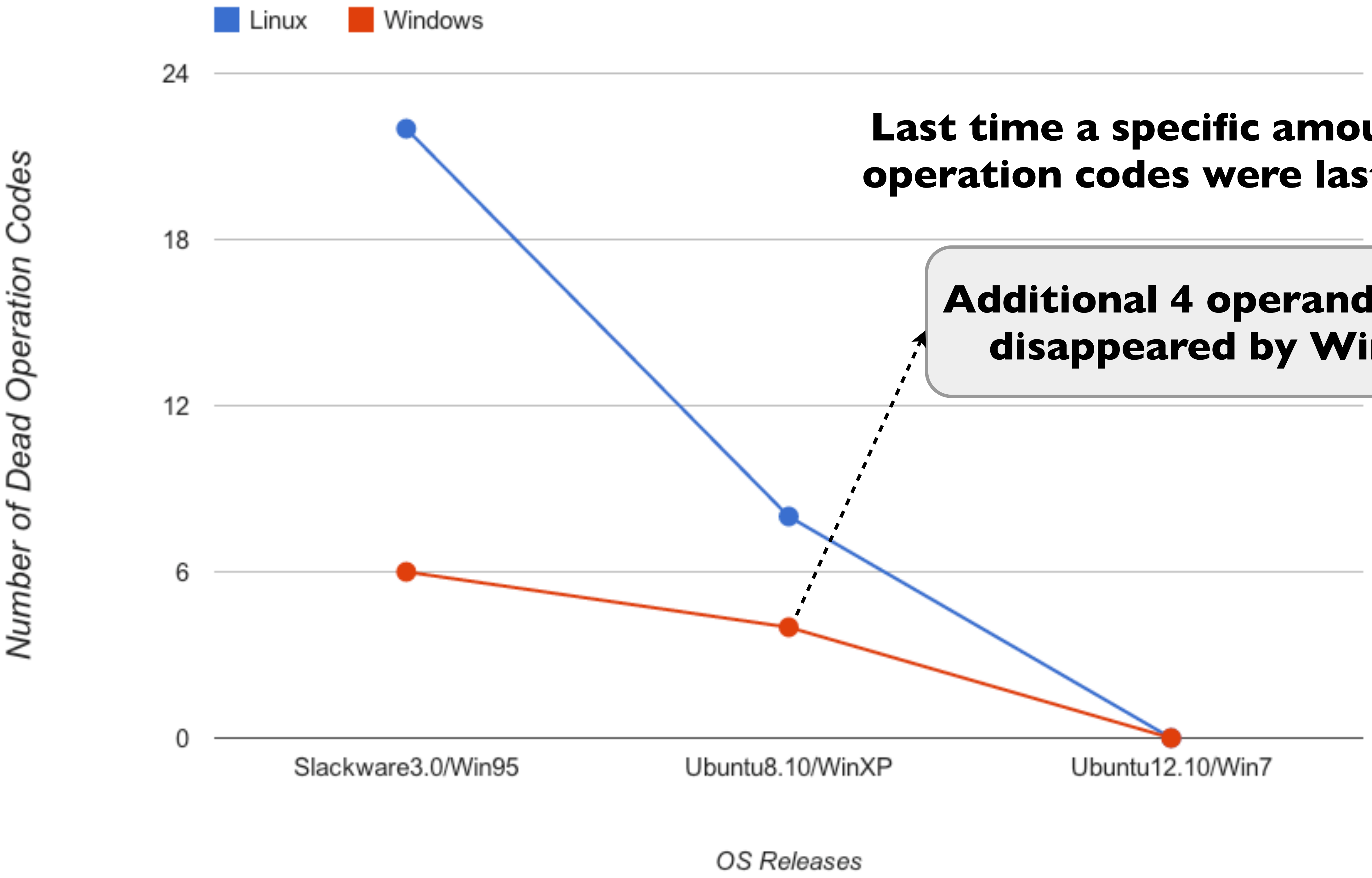
Dead Instructions



Last time a specific amount of operation codes were last seen

6 operation codes were last seen in Win95

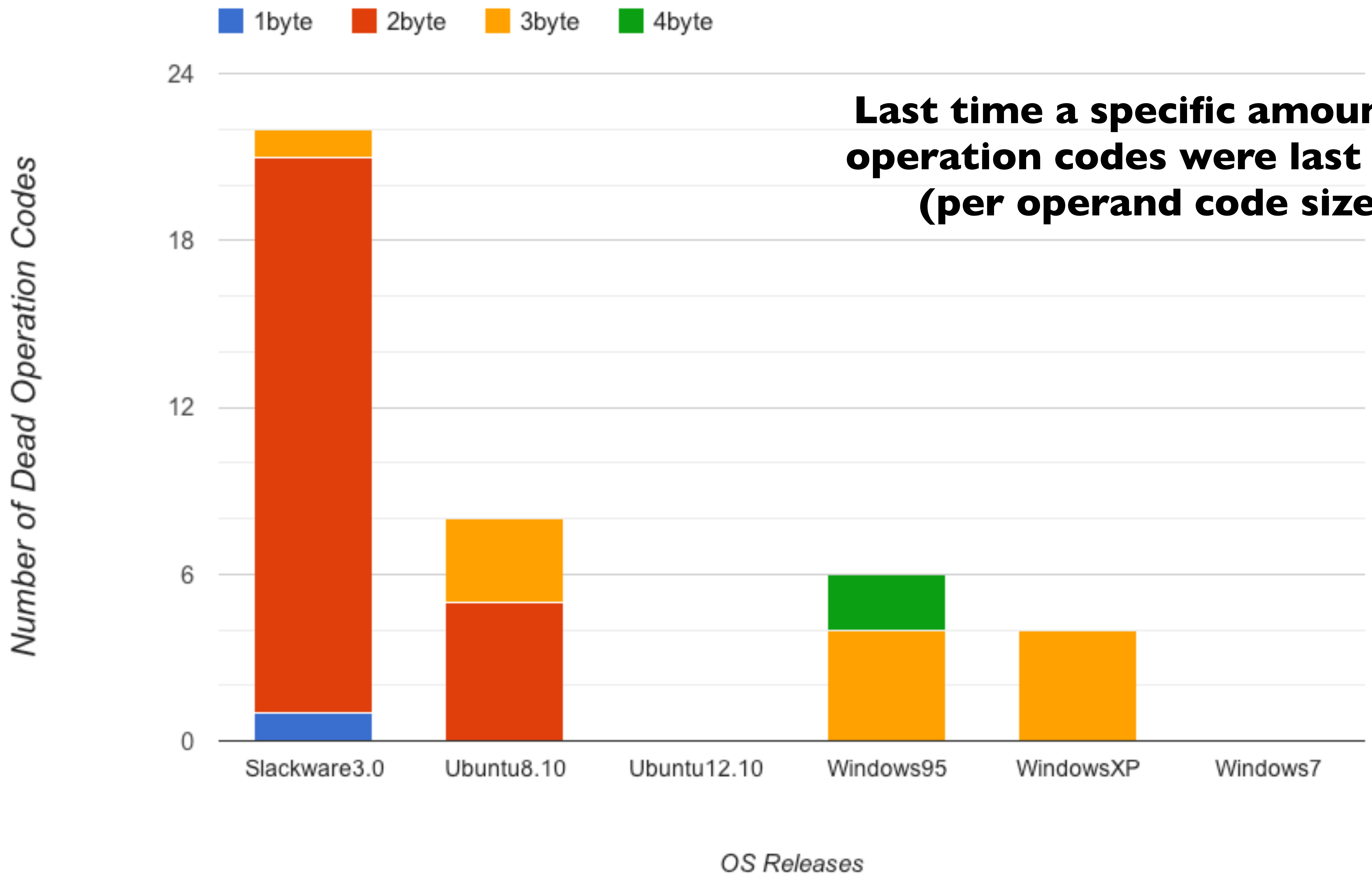
Dead Instructions



Last time a specific amount of operation codes were last seen

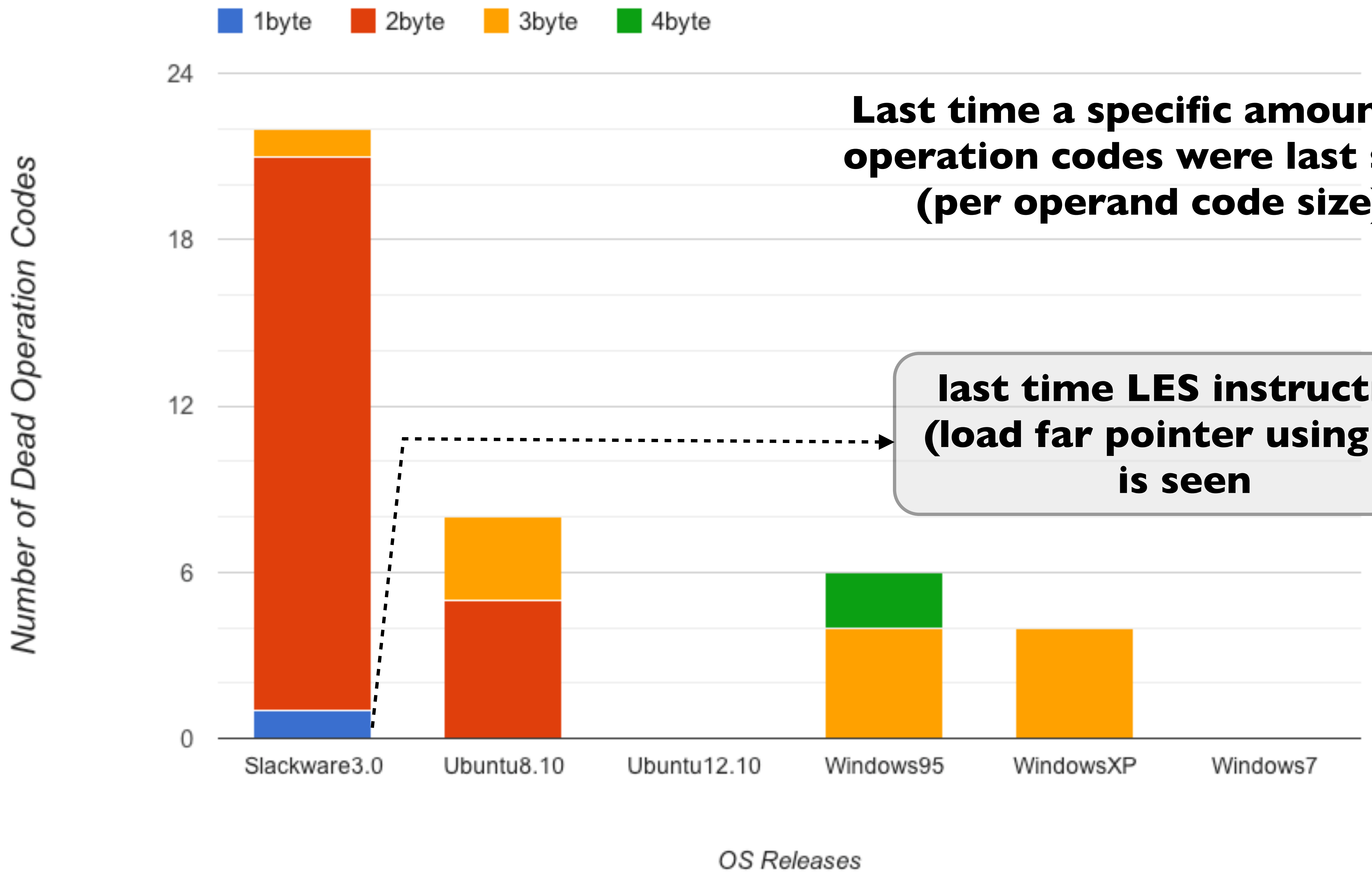
Additional 4 operand codes disappeared by WinXP

Dead Instructions



Last time a specific amount of operation codes were last seen (per operand code size)

Dead Instructions



Last time a specific amount of operation codes were last seen (per operand code size)

last time LES instruction (load far pointer using ES) is seen

Free operation codes

- In a scenario where the operation code space could be reused, it is specially important to pick a **1-byte instruction**
  - Escape code to encode **256 new 2-byte** instructions.

Vector Extensions

- First extensions to address floating-point calculations were 8087 and 80387
  - Stack based, old and **inconvenient** method for modern compilers
- Now **Superseded** by MMX, SSE, AVX, ...
  - Multiple fp calculations on the same cycle
  - Regular register operands (easier to the compiler to handle)

Vector Extensions

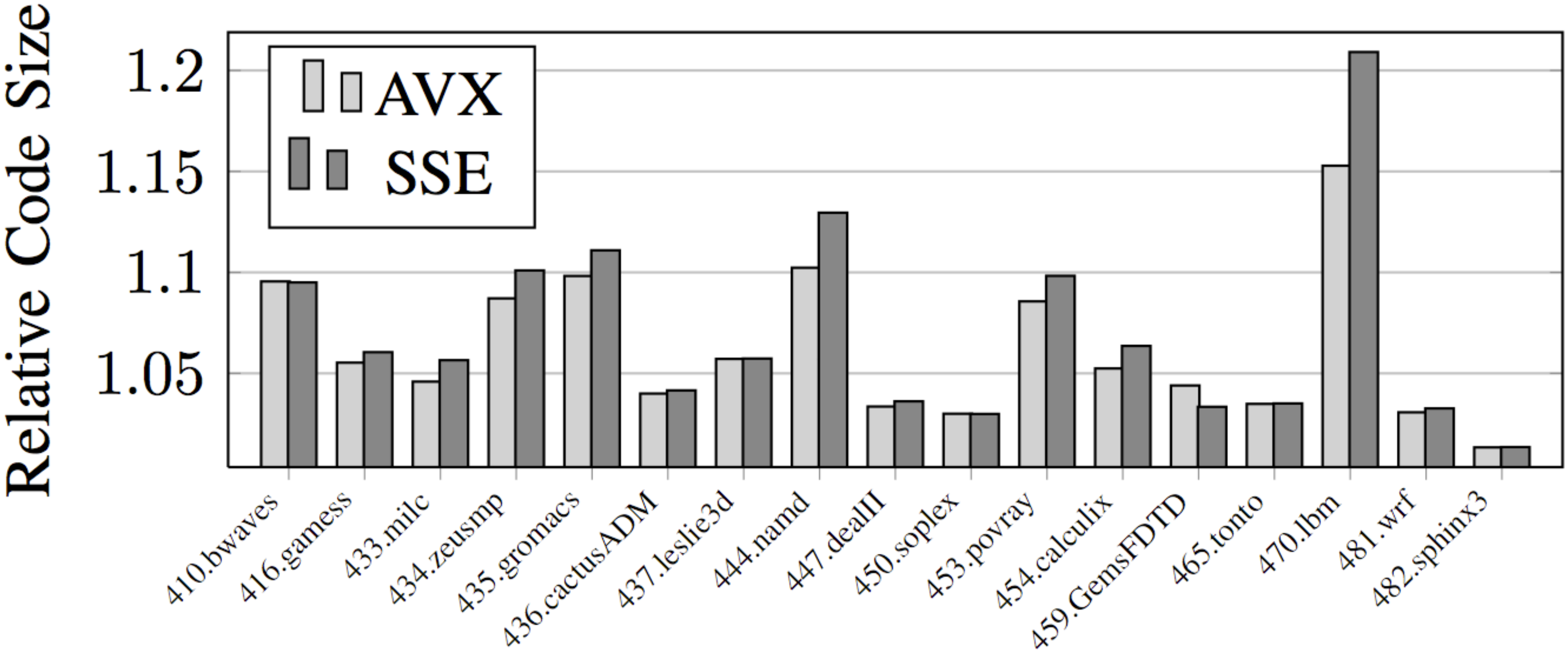
- Older IA-x87 floating-point extensions are still **widely present** in modern software
  - Default floating point option for several production compilers such as **GCC**

Vector Extensions

- ISA is forced to be redundant
  - E.g. It is possible to add two floating point data using either IA-x87 or vector extensions, a **suboptimal** operation code organization
  - **Larger binaries** using multimedia extensions than using IA-x87

Vector Extensions

SPEC2006FP Code Size: **SSE, AVX** and **IA-x87**



ISA Aging

Summary

- Compilers may explore old encodings because they have **better compaction rates**
- Many other operation codes are being deprecated, leaving, in terms of compaction rate, **valuable encodings unused**
- Removing unused instructions reduces hardware **complexity**

Conclusion

- Would **backward compatibility disruption** in favor of ISA evolution negatively impact systems?
- In practice, our Windows and Linux-based benchmarks show that many instructions were **definitely retired** by the software industry

Questions?