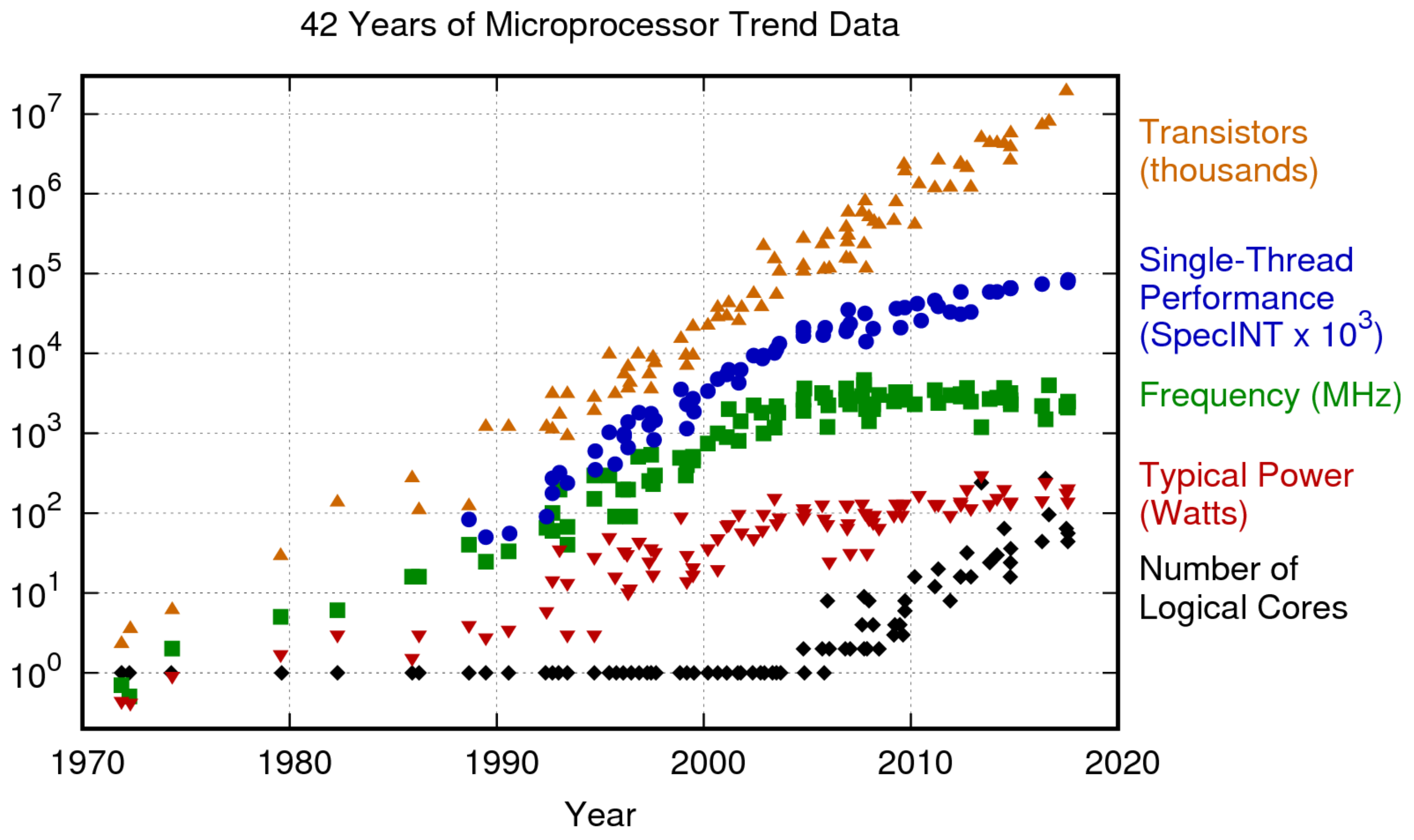
Golang Intro

Nikhil Ikhar

Stagnant Performance



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

Concurrency is afterthought in OLD language

	Language	Year
1	C	1972
2	C++	1983
3	Python	1991
4	Java	1995
5	JavaScript	1995
6	C#	2000

- If our basic tool, the language in which we design and code our programs, is also complicated, the language itself becomes part of the problem rather than part of its solution.” -**C.A.R. Hoare**

Philosophy

- Clear is better than clever
- We don’t read code, we decode it
  - But readability as a concept is subjective. It can be about braces, tab spaces, var names etc
- Clarity, is the property of the code
  - Code is read many more times than it is written.
  - Did you understand what you just read?
- Simplicity
  - Less is more or more is less
  - *The ability to simplify means to eliminate the unnecessary so that the necessary may speak*
- Productivity
  - Fast compilation
  - Enough tools to express ideas
  - Go toolchain

Clarity

```
• func comp(a, b int) int {
•         if a < b {
•                 return -1
•         } else if a > b {
•                 return 1
•         } else {
•                 return 0
•         }
• }
```

```
• func comp(a, b int) int {
•         switch {
•         case a < b:
•                 return -1
•         case a > b:
•                 return 1
•         default:
•                 return 0
•         }
• }
```

Pros

- Toolchain is awesome
- Capital-letter things are exported from packages.
- Support pointers but not pointer arithmetic
- First-class function, slice, map, and channel types.
- Fast compilation
- Create binaries for other architecture
- Closures
- concurrency primitives: goroutines, channels, select, mutexes
- No this keyword
- No inheritance. But use embedding
- Implicit Interfaces
- Excellent debugging with dlv
- Backward compatible

Cons

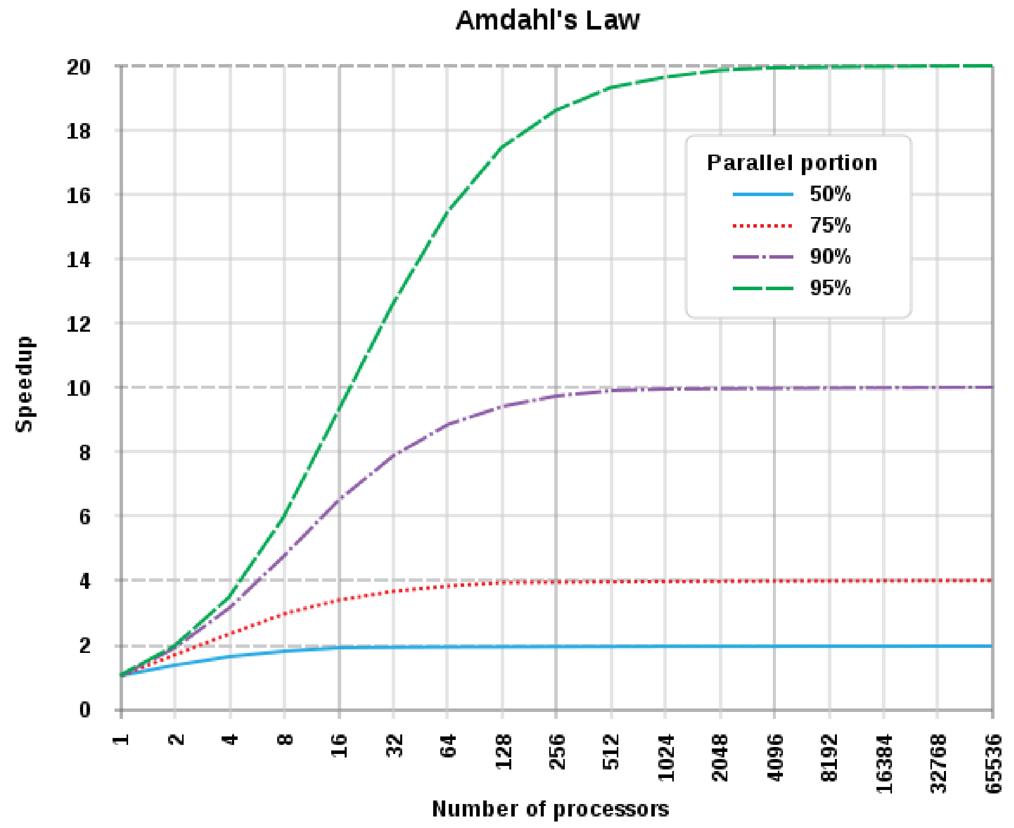
- ~~No Generics.~~
- Generics is slow
- Error handling is exhaustive for beginners
- Inheritance is different than what we see in Java, Python
- Lack of 'set' support
- Nil pointers can lead to errors
- Scientific computation support is limited.

Is Go blazingly fast

Amdahl's Law

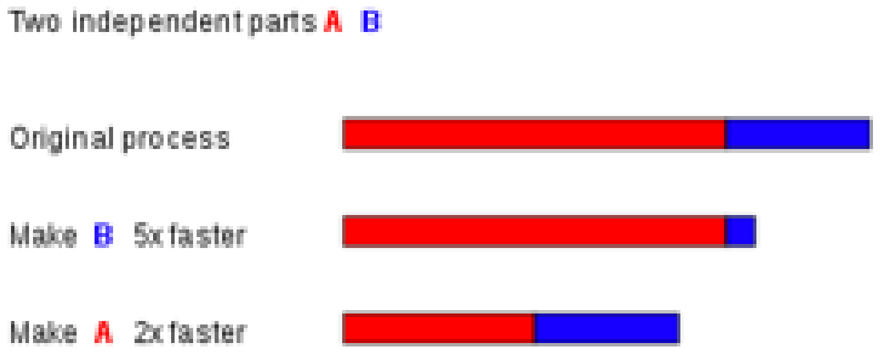
- "the overall performance improvement gained by optimizing a single part of a system is limited by the fraction of time that the improved part is actually used"
- The speedup is limited by the serial part of the program. For example, if 95% of the program can be parallelized, the theoretical maximum speedup using parallel computing would be 20 times.

Amdahl's Law



if one is running a fixed-size computation that will use all available processors to their capacity. Each new processor added to the system will add less usable power than the previous one. Each time one doubles the number of processors the speedup ratio will diminish, as the total throughput heads toward the limit of $1/(1 - p)$.

- Assume that a task has two independent parts, *A* and *B*. Part *B* takes roughly 25% of the time of the whole computation. By working very hard, one may be able to make this part 5 times faster, but this reduces the time of the whole computation only slightly. In contrast, one may need to perform less work to make part *A* perform twice as fast. This will make the computation much faster than by optimizing part *B*, even though part *B*'s speedup is greater in terms of the ratio, (5 times versus 2 times).



Getting Started

- Installation https://go.dev/doc/install next, next etc
- IDE Vscode, Goland
- Vscode Extension https://marketplace.visualstudio.com/items?itemName=golang.go

Data Type

- https://go.dev/ref/spec#Numeric_types
- String vs rune(int32)
- Array(fixed size) vs slice(can be modified in runtime)
- Map
- Interface

Tour

- https://gobyexample.com/
- https://go.dev/tour/list
- https://goplay.tools/
- https://go.dev/play/