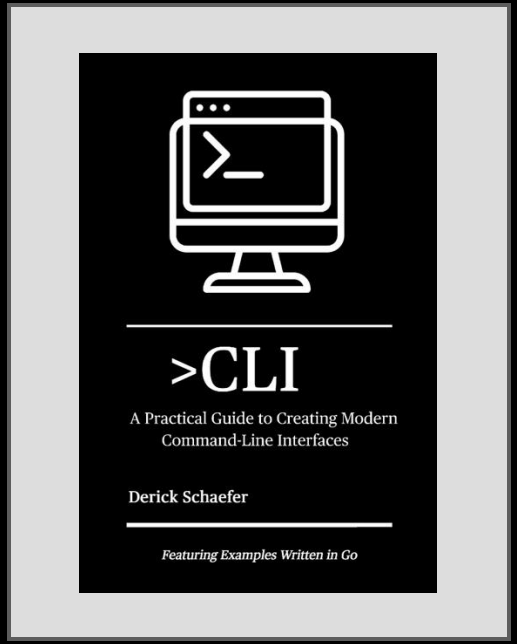
Systems Programming & Command-Line Interfaces

NOVEMBER 4, 2025
SOUTHWESTERN UNIVERSITY

DERICK SCHAEFER



Four Lightning Talks

- System Programming
- Command-Line Interfaces
- The Rise of Complete Language Ecosystems
- Structured AI Output

Who Am I?

- A technology business leader, author, consultant, and software developer
- 35 years in the enterprise software business
- Microsoft, Web Synthesis (WP Engine), Digital Insight (Intuit -> NCR), and Trintech alumnus.
- Father of an SU student (Sebastian Schaefer ‘26)

System Programming

A Definition (for today’s talk)

System Programming is the mindset of writing software with a deep awareness of how computation, memory, storage, and networks behave — and taking responsibility for how your code uses those resources.

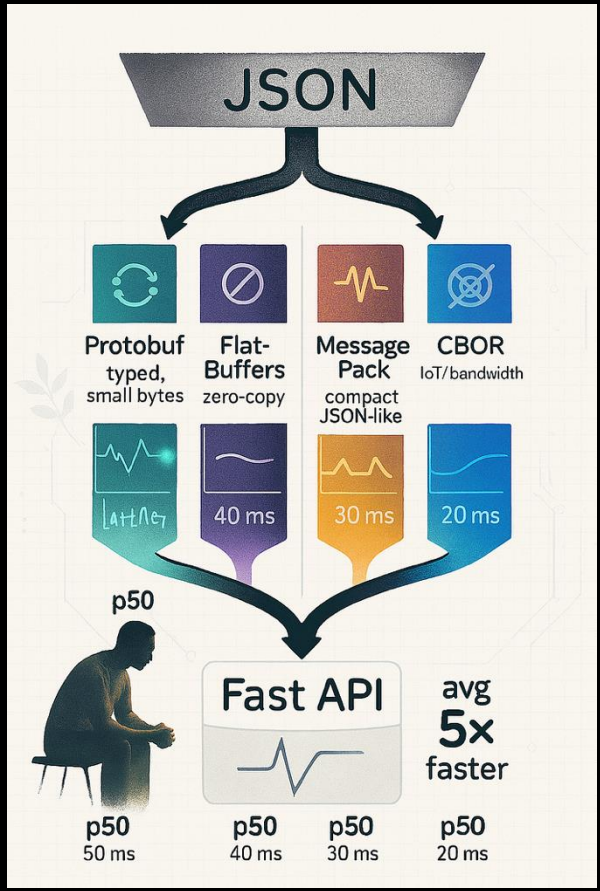
Areas of Optimization

CPU & GPU Usage	Network & File I/O	Memory Usage
Battery Usage	Concurrency & Parallelism	Automation (composability)

What is an API?

An API (Application Programming Interface) is the structured contract through which systems communicate — most commonly over HTTP today, often exchanging data in JSON.

JSON Alternatives



https://medium.com/@Krishnajlathi/forget-json-these-4-data-formats-made-my-apis-5-faster-9b14c1102113

Example 1: API Payloads

JSON

```
{
  "id": 12345,
  "first_name": "Alice",
  "last_name": "Nguyen",
  "is_premium_member": true
}
```

Protocol Buffers

```
syntax = "proto3";

message Customer {
  int32 id = 1;
  string first_name = 2;
. . .
 }
08 B9 60 12 05 41 6C 69 63 65 1A
06 4E 67 75 79 65 6E 20 01
```

Map Types – Mailbox Example

Traditional Map Type

- Letters are hashed and placed in box organized by their hash
- Searches for data require traversing the boxes
- Historical Context: Memory footprints were small and Moore's Law empowered CPU

Swiss Table Implementation (Google Patent)

- Place an 8-bit descriptor on each box to organize data
- Bills in one box, flyers in another, Personal letters, etc
- Historical Context: Memory footprints huge and CPU improvement slowing

Example 2: Map Type Implementations

Classic Map Type

```
Bucket 1:
  Used: [true true  false ..]
  Keys: [AAPL MSFT ...]
  Vals: [189  402 ...]
```

Swiss Table

```
Bucket 1:
  Ctrl: E4 F3 00 00 00 00 …
  Keys: [AAPL MSFT ...]
  Vals: [189  402   ...]
```

Built into Go 1.24 and available in C++

Take Aways

1) Ask Why? ALWAYS Ask Why!!!
2) Use Resources to Explore – Library, Professors, Wiki, LLM's, Developer/Debug/Profiling tools
3) Explore Open Source – Go Language, WordPress, Linux, Nginx (list in Appendix)
4) Be Better Than AI Co-Pilots

Use A System Programmer Mindset!

Command-Line Interfaces

A 1960’s Interface Back On Center Stage!

>CLI (the book)

- ➔ Published 2025
- ➔ Focuses on history, design, and best practices for “modern cli” development
- ➔ http://moderncli.dev
- ➔ Pickup a copy after the talk

The Unix Philosophy

- ➔ Write programs that do one thing and do it well.
- ➔ Write programs to work together.
- ➔ Write programs to handle text streams, because that is a universal interface.
- ➔ Favor simplicity over complexity.
- ➔ Build from small, composable pieces.

Command Models

Traditional Command Model

```
crm_add “ACME Co”

or

crm –add-customer=“ACME Co”
crm –list-customer
crm –delete-customer=“ACME*”
```

Object Command Model

```
cli <command> <sub-command>
cli <noun> <verb>

crm customer add “ACME. .”
crm customer list –format=

cli <verb> <noun>
crm add customer
```

Demo – WordPress and the WP-CLI

- ➔ Released in 2011
- ➔ Complete CLI to GUI Parity
- ➔ Noun – Verb Object Command Model
- ➔ “Favors options over decisions”
- ➔ Works well with others:

sort - **1971**
grep - **1973**
sed - **1974**
⚠ - 2000’s

The Rise of Complete Language Ecosystems

Programming Language Comparison

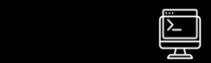
Aspect	C# (2000)	Go (2009)	Rust (2010)
Strength	Great for business apps and teams using Microsoft tools	Fast, simple, and easy to build cloud software	Extremely fast and safe for system-level code
Performance	High (but runs on a managed runtime)	Very high (compiled to native code)	Near C/C++ performance with safety built in
Ease of Use	Easiest to learn — familiar and well-documented	Very simple and consistent	Powerful but harder to learn at first
Memory / Safety	Automatic memory cleanup (garbage collector)	Automatic memory cleanup	No garbage collector — compiler enforces safety
Designed For:	Enterprise apps, desktop, web, and games	Cloud services, APIs, DevOps tools	High-performance systems, security-critical code

Go

- ➔ Funded by Google
- ➔ Built from the Plan 9 toolchain -> distributed scale by design (Ken Thompson, Rob Pike, Robert Griesemer)
- ➔ Simple – 25 keywords
- ➔ Compatibility - “code that compiles today will compile in future Go releases.”
- ➔ Natively compiles cross platform
- ➔ Think Python (simple) meets C++ (industrial strength) with Java/C# management (garbage collection)
- ➔ 37% Go devs work on ML data pipelines; 41% build AI specific API endpoints. GitHub Top 10 language.

Two Demos

- ➔ Cross patform compiled Hello World
- ➔ Goroutines and Waitgroups (concurrency on steroids)



LLM Structured Outputs

Use Case

Large language models (LLMs) can be given reasonable amounts of data and be asked to provided a structured response to a question or task:

Return a %, Boolean value (true/false), a text summary or recommendation

Flow

Data Input	**Stock Analysis** (symbol, RSI, WAP) **News Feed** (Headline, sentiment) **Zoom Transcript** (person, dialogue)
Context	“**You are a senior analyst** who specializes in. . .give me a recommendation/summary/action”
LLM	**Model:** "gpt-4.1-nano-2025-04-14"
Structured Output	**Answer** = True **Stock Symbols:** MSFT, KMX, NET **Themes:** Profits, Growth, Risk **Hedge:** 10%

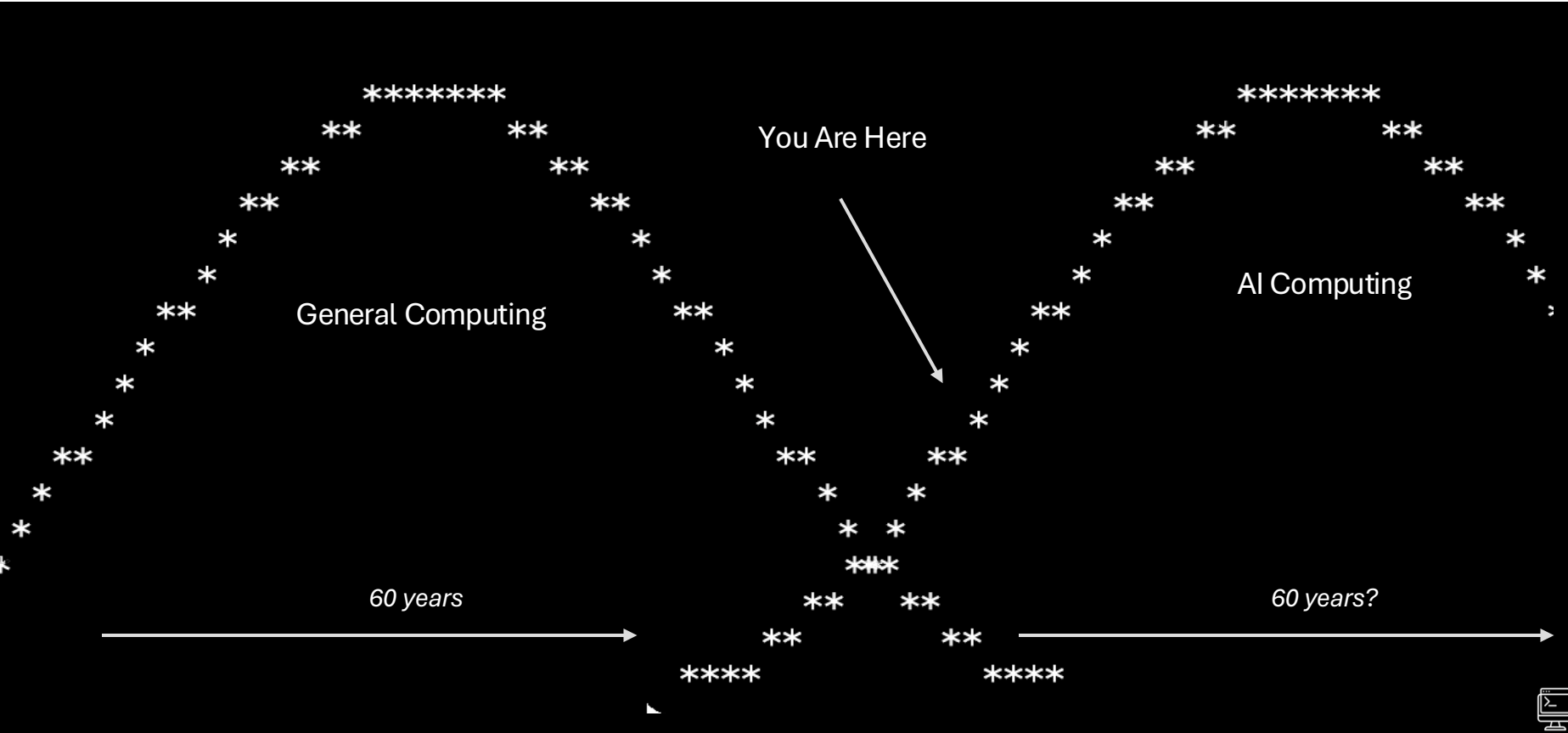
Demo

➔ *Momentum Trading Analysis Report*

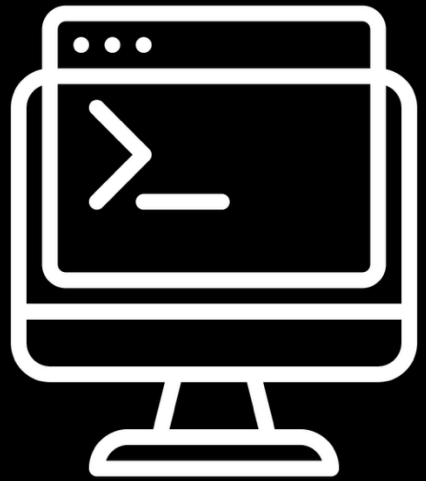
Takeaways

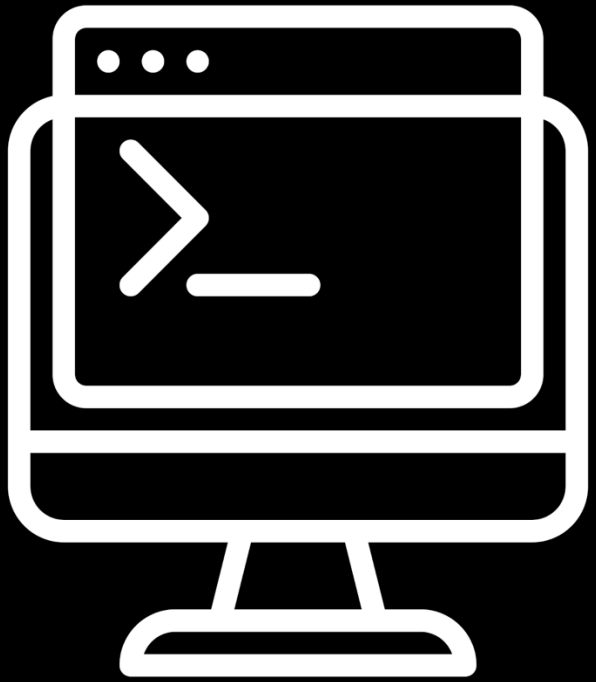
- ➔ Career developers should be proficient in 3 languages; one should be there go to and something they enjoy.
- ➔ Think of patterns when solving a problem and understand the high-level pro's and con's of at least 10 patterns. (e.g. CLI, micro-service, Single Page Architecture)
- ➔ Find your "irreplaceable home" in this brave new AI enabled world. We are producing way more code than we can review.
- ➔ People, Process, and Technology - We have way too much tech right now. Be a top 10% professional (people) and challenge yourself to learn about governance, quality, and team coding approaches (process)
- ➔ CLI's are 50-60 years old. . .there is a reason they are still a major force in computing.

Disruption Cycles (60 years)



Questions?





Appendix

Contact

- schaeferauthor@gmail.com
- https://github.com/derickschaefer
- https://www.linkedin.com/in/derickschaefer/
- https://moderncli.dev

- https://patents.justia.com/patent/12295446 (Sebastian's Patent)

Major Outages (improper use of resources)

1. Healthcare.gov (2013)

The disastrous launch of the Affordable Care Act's online insurance marketplace is one of the most famous government website failures.

- **Cause:** A combination of inadequate capacity planning, software coding errors, and a failure to conduct sufficient testing before launch. The site received five times more traffic than expected on its first day, overwhelming unprepared back-end systems.
- **Impact:** The site was almost unusable for weeks due to crashes, long delays, and software bugs, making it impossible for many citizens to sign up for health insurance.

2. CrowdStrike Outage (2024)

This widespread global IT disruption was caused by an update to a single configuration file in the cybersecurity software, CrowdStrike Falcon Sensor.

- **Cause:** A logic error in a software update pushed by CrowdStrike caused a system crash and Blue Screen of Death (BSOD) on Windows devices across many organizations.
- **Impact:** Hospitals and health systems, including those using EMRs, were severely affected, leading to delayed treatments and canceled surgeries. It also crippled airline booking systems, financial services, and other industries worldwide.

3. Amazon Web Services (AWS) Outage (2017)

A human error at the world's largest cloud provider caused a cascading failure that took many popular websites offline.

Performance Pattern Implementations

⚙️ Major Event-Driven, Non-Blocking Software Systems

Category	Example Software / Frameworks	Notes
Web Servers & APIs	**Node.js, Nginx, Netty (Java), Vert.x, FastAPI (Python)**	Built on asynchronous I/O and event loops — thousands of concurrent connections on minimal threads.
Message Brokers & Event Buses	**Kafka, RabbitMQ, NATS, Redis Streams, Pulsar**	Core event-driven systems enabling streaming, pub/sub, and async message passing.
Microservice Frameworks	**Spring WebFlux (Java), Akka (Scala), Quarkus, Micronaut, Go Kit**	Use reactive or actor-based models to handle non-blocking requests efficiently.
Frontend / UX Systems	**React, Vue.js, Angular**	Entirely event-driven: UI reacts to user events, async data changes, and state updates.
Cloud & Edge Platforms	**AWS Lambda, Azure Functions, Google Cloud Functions**	*Serverless event-driven computing* — triggers from queues, HTTP, or file events.
Databases & Streaming Systems	**Cassandra, Elasticsearch, ClickHouse, Flink, Storm**	Event streams, async writes, and reactive ingestion pipelines.
Operating Systems / Kernels	**Linux epoll, Windows IOCP, libuv, Tokio (Rust)**	Provide the underlying async I/O mechanisms that make non-blocking apps possible.

Generic vs. Swiss Table Map

Feature	Non-Swiss (Traditional)	Swiss Table
Control bytes	None	Compact control array storing occupancy + hash fragments
Memory locality	Poor — probes jump all over	Excellent — control bytes & data packed together
Probing	Linear or quadratic	SIMD-accelerated probing of 8 or 16 control bytes at once
Empty slots	Explicit empty marker per bucket	Bitmask encoded in control bytes
Insertion speed	Slower — multiple cache misses	Faster — fewer cache misses, SIMD matches

Systems Programming Topics (1 of 3)

Data Representation & Efficiency

Binary serialization formats — Protobuf, FlatBuffers, Cap'n Proto, MessagePack

Memory layout & alignment — struct padding, cache line effects, false sharing

Endianness — how byte order affects portability and performance

Copy-on-write & zero-copy designs — minimizing unnecessary data movement

Memory & Storage Behavior

Stack vs Heap — allocation cost, lifetime, and locality

Memory allocators — jemalloc, tcmalloc, mimalloc, and arena allocators

Paging and virtual memory — what happens when you exceed RAM

Filesystem semantics — fsync, journaling, write amplification, and durability tradeoffs

Systems Programming Topics (2 of 3)

Data Structures Evolving

Hash map designs — Swiss Table, Hopscotch hashing, Robin Hood hashing

Lock-free and concurrent data structures — atomic primitives, RCU, hazard pointers

Bloom filters and probabilistic data structures — trading accuracy for speed

B-Trees, LSM Trees, and log-structured storage — foundations of modern databases

Concurrency & Execution

Threads vs async I/O vs coroutines — concurrency models compared

Futures, promises, and event loops — control flow for concurrent systems

Synchronization primitives — mutexes, spinlocks, atomics, memory fences

Context switching & scheduling — kernel vs user-space threads

Systems Programming Topics (3 of 3)

Network & I/O Awareness

Blocking vs non-blocking I/O — epoll, kqueue, io_uring

Serialization over the wire — binary framing, compression, checksums

Latency vs throughput — measuring and tuning tradeoffs

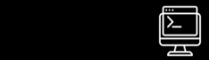
Performance & Profiling

CPU cache hierarchy — cache misses, prefetching, branch prediction

Vectorization & SIMD — using hardware parallelism

Memory bandwidth vs compute limits — identifying bottlenecks

Instrumentation & tracing — perf, eBPF, flamegraphs



CLI Use Cases

Headless Server Management	Software Development (e.g. Claude Code)	Proto-typing
Power Tools	DevOps	CLI Framework Extensions (WP-CLI)

Command-Line Fundamentals (1 of 2)

Shell Fundamentals

Shell grammar: quoting, variable expansion, command substitution ($(), backticks)

Pipes and redirection: |, >, >>, <, 2>&1, /dev/null

Exit codes & control flow: $?, &&, ||, if, for, while

Environment vs shell variables: export, set, env, unset

Command substitution & subshells: $(...), (...), { ...; }

Command-line editing: readline shortcuts, history (!!, !$, Ctrl+R)

Core Text & Stream Tools

grep — pattern matching, regex filters, -v, -r, -A/-B context

sed — stream editing and substitution, s/pattern/repl/g

awk — field processing, numeric and text transforms

cut, sort, uniq, tr, paste, join — composable data manipulation

tee — duplicating output streams in pipelines

Command-Line Fundamentals (2 of 2)

Files, Processes & System Awareness

File inspection: ls, stat, file, find, du, df

Permissions: chmod, chown, umask

Process control: ps, top, kill, jobs, bg, fg

Signals & exit traps: trap, kill -l

Subprocess management: backgrounding (&), xargs, parallel

Everyday Power Tools

vim or nano — text editing fluency

ssh, scp, rsync — remote access and file transfer

curl, wget, jq — network and API interaction

tar, gzip, zip — compression and archiving basics

type, which, command — locating and understanding commands

git — version control as a CLI tool

WHODUNIT (Written in Go)

Great blog post on navigating LLMs with a WHODUNIT Murder Mystery game written in Go

https://blog.apartment304.com/whodunit-llm-murder-mysteries/

https://whodunit.rip/