# Constraint-Based Design:

## *An Unexpected Gateway to Understanding Modern LLM Architecture*

**Edward Barnard**

# Constraint-Based Design: An Unexpected Gateway to Understanding Modern LLM Architecture

The HPC Tradecraft Master Practitioner, Volume 3

Edward W. Barnard

This book is available at https://leanpub.com/constraint-design

This version was published on 2026-07-03




This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2026 Edward W. Barnard

## Also By Edward W. Barnard

The AI Coverup: Are We Really Hallucinating?
Ethics in High-Stakes Environments: Personal Survival Within a Power Imbalance
Francis Barnard Part A
Unexpected Histories: Spotting Patterns and Making Connections That Others Miss
Large Language Model Architecture Patterns in PHP: No Mathematics Required
The Impossible Challenge Manual for Age 14 and Up, Even For Adults: How to accomplish what everyone says you can't
The Wizard's Lens: Learn to Think Like AI
Nobody but Us: A History of Cray Research's Software and the Building of the World's Fastest Supercomputer

# Contents

**Chapter 1. Book Development Plan** . . . . . **1**
Source Code . . . . . 1
The Book Design . . . . . 2
On Paper . . . . . 2
Whole-System Viewpoint . . . . . 3
Domain-Specific Language . . . . . 3
Executable Code . . . . . 4
LLM Correspondence . . . . . 4

**Chapter 2. User Experience** . . . . . **6**
Problem of Credibility . . . . . 6
Gateway to AI . . . . . 6
Show Me The Code . . . . . 6
Survival Requirements . . . . . 7

**Chapter 3. Developer Experience** . . . . . **8**
The Case Method . . . . . 8
Bare Metal Case . . . . . 8
Which Part is Relevant . . . . . 8
Work in Progress . . . . . 8

**Road Map** . . . . . **9**

**Appendix A. HPC Tradecraft Road Map** . . . . . **10**
The 1995 Barrier . . . . . 10
Two Series . . . . . 11
Book 1. The AI Coverup: Are We Really Hallucinating? . . . . . 11
Book 2. Nobody but Us: A History of Cray Research's Software and the Building of the World's Fastest Supercomputer . . . . . 12
Book 3. The Wizard's Lens: Learn to Think Like AI . . . . . 12

Book 4. Ethics in High-Stakes Environments: Personal Survival Within a Power Imbalance . . . . . 13
Book 5. Unexpected Histories: Spotting Patterns and Making Connections That Others Miss . . . . . 13
Book 6. Constraint-Based Design: An Unexpected Gateway to Understanding Modern LLM Architecture . . . . . 14

**Index** . . . . . **17**

# Chapter 1. Book Development Plan

This book is nothing like a traditional IT book that includes code listings. This book is a case study in top-to-bottom systems thinking as practiced at Cray Research during the 1980s and 1990s.

The code itself is irrelevant. The design think matters. This case study is an **apprenticeship** with strong relevance to modern AI. Did I say the code is irrelevant? I take that back. The code itself stands as proof of this case study's direct relevance to AI.

Here is the code and a free (no information collected) website that allows you to explore the actual code as the player:

- The live website is here: Swiss Adventure
- The assembler and PHP source code are here: SwissAdventure

This will be a large book because I have several layers of experience to share with you. Before I show the book design, I need to explain the above code (the archive and the live game website).

## Source Code

The 1986 source makes no reference to AI, and was not designed with AI in mind. That is also true of the port to PHP that I did in April and May of 2019. I am leaving those two artifacts untouched in the public GitHub repository, because touching anything would update the latest timestamp. GitHub is a respected third-party confirmation of the timestamp. This matters as proof that I did not "retrofit" any AI concepts onto old code.

In 2025, I took the PHP source code and instrumented it by creating a separate "LLMTrace" module, independent of the 2019 engine. I then inserted trace calls into the 2019 code (a separate copy of the code in a separate private repository) without changing any execution paths. In other words, I layered the tracing on top of the 1986/2019 functionality as commentary showing the flow from request to response.

I then added a Laravel (PHP framework) wrapper that simply routes web requests into the Swiss Adventure PHP engine. I added a React.js front end that emulates a 1980s operator console, with three side consoles showing the events passing through. I used Claude to write the React.js code. It is all stateless and collects no user information, and does not require user registration.

The trace facility shows both code flow and the flow of LLM patterns from request to response. The "LLMTrace" instrumentation and display is AI-specific, but everything else has the 1986 origin and therefore makes no reference to modern AI.

## The Book Design

My general plan for this book is to reproduce my actual design thinking at the time (40 years ago), which is as follows.

First, I did not set out to design a game program. I set out to solve a problem. I used the computer (a CRAY-1 supercomputer, which is pretty cool) to assist in solving the problem: establish credibility as a subject matter expert without bragging.

## On Paper

I then designed the adventurer (user) game experience, using Colossal Cave Adventure and Zork as inspiration. They established the pattern of creating formation-like experiences, forcing the adventurer to form judgment under adversarial conditions. Checking the authors' work environments, that formation aspect makes sense, but to me at the time it was simply the normal way of doing things. The focus was on teaching ways of thinking rather than providing entertainment.

On paper, I worked out the navigation, the location descriptions, the maze variations. We will work through these steps because they reflect the systems thinker's design process. The value will not be obvious until later steps, but the value exists. I created the gate conditions: checkpoint encountered, trap door, hall of testing, final exam.

Since the basic hardware unit was 16 bits, I used that as my base number of "things". Sixteen checkpoints, sixteen maze locations, sixteen test stations,

sixteen final exam questions. Since I was doing this for fun and to prove a point, I was totally free to go "hard core". I carried the entire game state, top to bottom, first to last, in a single 16-bit register.

Note that I did not have 65536 states, as a 16-bit number would imply. I had sixteen independent yes/no states, packed as a single register. The modern term might be tensor or binary feature vector.

Another way to put this in modern terms is that I designed 16 independent attention heads, and then built the feed-forward network that integrates the attention heads in code, producing the response. I implemented the navigation as weighted locations. But the design, to this point, is all on paper. As of 1986, nobody had yet changed the centuries-old terminology of "vector" or "matrix" to cool-sounding names like tensor.

We call numbers "Arabic" numerals to distinguish them from "Roman" numerals. Therefore, naturally, I numbered the checkpoints waahid, ithneyn, thalatha... rather than one, two, three... because "one, two, three" is definitely not Arabic. I figure that on bare metal, accuracy is important.

## Whole-System Viewpoint

In this part of the book, you will play the online game side by side with listings and screen shots. We can observe the full game sequence and gate conditions. By understanding and experiencing how everything works, you then have a way to place every detail in context. We can see how data are represented and then displayed.

**This allows the key skill to be taught and experienced.** The key skill that is rarely taught is the ability to understand and keep track of the system as a whole while focusing on specific details, and to freely ride up and down layers of the system. We used layers and abstractions, but we never allowed the abstractions to hide things. The layer underneath the abstraction remained surpassingly important, as did understanding how systems interacted with each other.

## Domain-Specific Language

I was extremely comfortable with the mainframe assembler CAL (Cray Assembly Language), and knew the I/O Subsystem assembler APML ("A" Processor

Macro Language) worked similarly. I had experience modifying the mainframe assembler itself. This had to do with compiler construction, but the "Dragon" book did not yet exist. (It came out in 1986 but I was unaware of it. I had learned from Gries, *Compiler Construction for Digital Computers*.) That depth of knowledge of the assembler itself meant that I could *use* it effectively.

I was therefore comfortable in creating what we now call a Domain-Specific Language encoding the on-paper game design. I used the assembler's macros and micros and built in directives such as ECHO. In modern terms I was taking advantage of my build toolchain's features.

Part of this level of design process was understanding what I was designing *for*. I was designing a data store that would reside in a high-speed external device like SSD, with individual items pulled into local memory as needed.

I therefore designed packed data structures on paper and decided that *all* game aspects should be encoded in the same minimalist data structure. It was my software design, and I gave myself permission to keep it simple. In assembly language you can get away with making everything look the same, without the descriptions being the same format. Macros and micros did the translation and implementation.

This also meant I could expand or change game locations or relationships without changing any executable code. Since I was the developer, I decided that being developer-friendly would be a good thing. And more fun.

At this point we have learned two layers, seen how it works in actual game play, and walked through the Domain-Specific Language implementation. That implementation includes building all game data structures at compile time.

## Executable Code

I designed the executable code based on what functions had to be co-resident. I designed the code around hardware constraints. I made the call stack as short as I could with no cross dependencies, keeping each overlay as small as possible, and thus the smallest memory footprint I could manage.

Remember the problem to be solved: demonstrate subject-matter expertise. I was showing that I was comfortable with various standard operating system techniques and in particular, constraint-based design. That latter aspect is what makes this case study valuable in our modern-day context.

## LLM Correspondence

The adventurer experience is designed as a formation experience. That is the subject of the next chapter, “user experience”. The following chapter shifts to the designer/developer experience.

Finally, once all levels of design choices are understood, we can take on the LLM correspondence. The correspondence exists because modern LLMs face similar types of constraints, and similar constraints tend to shape similar solutions, even when the scale is several orders of magnitude different.

# Chapter 2. User Experience

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/constraint-design.

## Problem of Credibility

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/constraint-design.

## Gateway to AI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/constraint-design.

## Show Me The Code

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/constraint-design.

### Layered Disorientation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/constraint-design.

### Maze as Adversarial Crisis

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/constraint-design.

### Checkpoint System as Distributed Knowledge

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/constraint-design.

### Hall of Testing as Gated Progression

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/constraint-design.

### Scrap Heap as Consequence

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/constraint-design.

## Survival Requirements

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/constraint-design.

# Chapter 3. Developer Experience

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/constraint-design.

## The Case Method

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/constraint-design.

## Bare Metal Case

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/constraint-design.

## Which Part is Relevant

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/constraint-design.

## Work in Progress

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/constraint-design.

# Road Map

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/constraint-design.

# Appendix A. HPC Tradecraft Road Map

What is "HPC tradecraft" as I know it? This is high-performance computing (HPC) arising from signals intelligence (code breaking) needs. My particular lineage begins with the "Tunny" code breaking project at Bletchley Park during World War II. That produced such an odd-looking contraption that the "Wrens" operating it named it the Heath Robinson. W. Heath Robinson was a cartoonist known for portraying fantastically complex machines designed to accomplish simple tasks.

When U.S. Navy veterans built something similar for a similar purpose, they named it GOLDBERG in honor of cartoonist Rube Goldberg's fantastically complex machines, explicitly acknowledging the Heath Robinson tradecraft origins. One of the founders of Engineering Research Associates (ERA) was Bill Norris. ERA built GOLDBERG, and also hired Seymour Cray. Norris later founded Control Data Corporation (CDC) and Cray soon followed as their youngest employee. Cray later founded Cray Research, which I joined early 1980.

Thus the high-performance tradecraft I learned within Cray Research already had a 40-year history. Perhaps because so much was classified Top Secret, we never wrote down *how* we did things. Institutional knowledge was passed from person to person as mentorship, collaboration, and informal apprenticeship.

## The 1995 Barrier

Circa 1995, something shifted. We hid the messy details behind abstractions, allowing for far more complex software development. We no longer needed to think of systems as a whole, and how things performed on bare metal.

For high-performance computing, this presents a problem. HPC tradecraft remains constraint-based design, but we can no longer see the constraints informing the design. This became a niche topic rarely covered in school (or the workplace).

Meanwhile, the pre-1995 tacit knowledge is retiring out of the workforce. My role as Custodian (the Russian term is more specific, хранитель) is to share this knowledge with you as a living practice.

## Two Series

I created two book series to teach HPC tradecraft.

- "HPC Tradecraft Apprenticeship", three books, is the institutional tradecraft as I learned and practiced, and still practice. Tradecraft is useless unless replicable.
  - *The AI Coverup: Are We Really Hallucinating?* teaches my replication method.
  - *Nobody but Us: A History of Cray Research's Software and the Building of the World's Fastest Supercomputer* teaches *how* we accomplished what we did.
  - *The Wizard's Lens: Learn to Think Like AI* teaches systems thinking, with close observation and characterization of HPC systems under load, using modern AI as the example.
  - The above three books are transmission protocol; HPC design; HPC operations.
- "The HPC Tradecraft Master Practitioner", three books, is the author letting loose and having fun, expressing his mastery of HPC tradecraft.
  - *Ethics in High-Stakes Environments: Personal Survival Within a Power Imbalance* prepares for the insurmountable power imbalance of a new graduate in the amoral corporate environment.
  - *Unexpected Histories: Spotting Patterns and Making Connections That Others Miss* demonstrates debugging skills applied outside computer science, showing their value and timeless nature.
  - *Constraint-Based Design: An Unexpected Gateway to Understanding Modern LLM Architecture* demonstrates 1986 tradecraft directly applies to modern AI transformer design and usage.

## Book 1. The AI Coverup: Are We Really Hallucinating?

I iteratively created book 1 as I circled what I actually have to share, why it is important, and how to make it accessible to the right persons.

- Show my teaching method of formation text that performs what it teaches, which I frame as the manner of replication.
- Act as filter and qualification, so that readers have the necessary information before buying/committing to the two large flagship books.
- State *Nobody but Us* as prerequisite to *Wizard's Lens*, which readers might perceive as problematic if they came for AI. Without *Nobody but Us*, readers lack the systems approach we stopped teaching around 1995.

The sequence itself is formation. This was the teaching order within Cray Research.

## Book 2. Nobody but Us: A History of Cray Research's Software and the Building of the World's Fastest Supercomputer

This book is by far the largest with the greatest depth. However, persona walk-throughs consistently report the quality as "uneven." This perception is because the book attempts to overcome the Edwin Abbott *Flatland* paradox. "Look here" in the midst of formation text appears to be uneven quality requiring an editor, and a human might see it the same way, but "look here" is pointing to something most readers do not have the vocabulary to see on their own.

The book serves a crucial purpose of credentialing the author (via demonstration not assertion). Why? Because the next book makes apparently-outrageous claims which are not proven until reading through to the final chapter. The claims are based in "Nobody but Us" tradecraft.

What Cray Research accomplished is well documented. This book teaches *how* we did it. But there is another aspect, which is the next book.

## Book 3. The Wizard's Lens: Learn to Think Like AI

Think Jay Forrester and Donella Meadows and Eli Goldratt and Robert Gagné and Anders Ericsson, all of whom are referenced in the book. This is the craft of closely observing and characterizing HPC systems, using AI as the working example. This is the Cray Research attitude of treating barriers as opportunities. Barriers identify the point of maximum leverage. Meadows lists points of leverage. For my purposes, I collapse her list to Liebig's Law of the Minimum (plant growth is bounded by the relatively scarcest nutrient). This principle matches Goldratt's *The Goal*.

Rather than the skills in the prior book, this teaches shifting perspective, systems as a whole, interactions between systems at the boundary, and the attitude of not being distracted by the fact that it has never been done before.

This ends the primary series, the "HPC Tradecraft Apprenticeship."

## Book 4. Ethics in High-Stakes Environments: Personal Survival Within a Power Imbalance

The book is a work in progress. My premise is that we do not give computer scientists, particularly recent college graduates, the skills to survive the corporate power imbalance. I wrote two essays regarding that power imbalance. They are the raw material from which I plan to write the book, but the essays themselves have urgent importance, thus publishing as a work in progress.

## Book 5. Unexpected Histories: Spotting Patterns and Making Connections That Others Miss

This book consists of 15 essays/chapters, 49,000 words, demonstrating tradecraft in primary source analysis and constructing interesting narratives by "connecting the dots" that others miss. While nothing in here is technically difficult, some seem to have this aptitude and some do not. Book 2, *Nobody but Us*, has a chapter on what William Friedman has to say about developing intuition. This book is practice in developing that premise.

Tradecraft demonstrated across the centuries and outside computing demonstrates invariants, skills non-specific to a given technology, while demonstrating Dr. Piotr Galperin's final stage of formation.

## Book 6. Constraint-Based Design: An Unexpected Gateway to Understanding Modern LLM Architecture

In 1986, I wrote a bare metal device driver inside the Cray I/O Subsystem as a joke and to demonstrate the author's capability. It is a text adventure game, Swiss Adventure, inspired by Colossal Cave Adventure. What is useful is its set of characteristics.

The game itself forces the user (adventurer) into an operational apprenticeship, working with incomplete information in an unfamiliar context. I created a PHP/Laravel/React.js version that is live on my website, stateless, requiring no login or registration or data collection. The live version is the game itself presented as typing at the operator console, with side displays (essentially event traces) showing flow through the assembly language overlays, and flow through the LLM attention mechanism patterns.

Thus it is an executable companion to the book. The 1986 source code and the PHP engine are in a public GitHub repository, with the most recent timestamp in the repository as May 2019, which is before GPT and Claude became well known publicly.

- The live website is here: Swiss Adventure
- The assembler and PHP source code are here: SwissAdventure

There are several aspects of interest.

### UPDATE

The source code is in UPDATE format. Secondary sources (wikipedia) indicate CDC UPDATE is one of the earliest known source code management systems, from late 1950s or early 1960s, punch card based. I know from personal (as primary source) knowledge that the first Cray Research software person was hired directly from CDC. I know from internal statements that CDC's UPDATE

was ported to Cray Research. While onsite at Boeing Computer Services 1980-1982, the front end systems were CDC Kronos. I used CDC UPDATE and Cray Research Update side by side. Thus I am clear on the provenance.

Relatively few extant examples of live code in UPDATE format exist. This one is publicly available on GitHub.

## APML

When the CRAY-1/S was announced with a new I/O Subsystem (IOS) in addition to the mainframe CPU, the IOS had been developed as the "A" Processor. Extant documentation as of 1980 called it that. The cpu assembler was CAL, Cray Assembly Language. The IOS assembler was called APML, "A" Processor Macro Language.

The 1986 source code is a live, complete, package, and likely the only substantial example of IOS programming written by an actual Cray Research IOS programmer. It was written (as a student) during a class *teaching* IOS programming, but I was later part of the IOS group for 10 years, so can act as primary source asserting it is authentic and representative.

"Live" might be misleading in that no system still exists to run it. But the precise port to PHP *is* live and online.

## Adventurer Experience

The adventurer experience has characteristics similar to many realms of operational training. I intentionally wrote it inspired by Colossal Cave Adventure and Zork. It might be worth noting that the authors of those two programs worked in operational environments. Thus I was not inventing the formation aspect. That was already a characteristic of text adventure games.

## Constraint-Based Design

The source code has many levels and aspects of constraint-based design, and it is possible to teach the design without dwelling on assembler syntax. I implemented the training data (all navigation and text content) as a domain-specific language using the assembler macro and micro facility. I had a deep enough understanding (including experience with making mods to the CAL

assembler source) to do this. Adding game features did not require executable code changes.

Here is the odd part: The source code architecture is isomorphic to 2017 "attention is all you need" architecture function. Similar constraints shape similar solutions (Genrich Altshuller's TRIZ principle). My operating environment was quite similar to a modern hot LLM token context under load. Attention heads, weighting, pulling from external source then deallocating… quite a few features are present.

A working bare-metal example of constraint-based design can be useful. But a 1980s demonstration that HPC methods are relevant to modern AI has interesting implications, which is why I decided to include this as a book.

# Index


Bletchley Park, 10

CDC
    Kronos, 15
    UPDATE, 14
Colossal Cave Adventure, 14
constraint-based design, 10
*Constraint-Based Design: An Unexpected Gateway to Understanding Modern LLM Architecture*, 14
Control Data Corporation, 10
Cray Assembly Language, 16
Cray I/O Subsystem, 14
Cray Research, 10
Cray, Seymour, 10
CRAY-1/S, 15

domain-specific language, 15

Engineering Research Associates, 10
*Ethics in High-Stakes Environments: Personal Survival Within a Power Imbalance*, 13

Galperin, Piotr, 14
GOLDBERG (computer), 10
Goldberg, Rube, 10

Heath Robinson (computer), 10
HPC tradecraft, 10
HPC Tradecraft Apprenticeship (series), 11

*Nobody but Us: A History of Cray Research's Software and the Building of the World's Fastest Supercomputer*, 12
Norris, William C., 10

Robinson, W. Heath (cartoonist), 10

Swiss Adventure, 14

*The AI Coverup: Are We Really Hallucinating?*, 12
The HPC Tradecraft Master Practitioner, 11
*The Wizard's Lens: Learn to Think Like AI*, 13

*Unexpected Histories: Spotting Patterns and Making Connections That Others Miss*, 13
UPDATE format, 15

Zork, 15