whogloo

ABL Has 500 Statements. Go Has 25. So Who Wins?

ABL, Go, TypeScript, Rust and
knowing where each one belongs

Julian Lyndon-Smith



A safe space, mostly

Hands up if someone has told you to rewrite it in Go

Keep your hand up if that person has never posted a journal to the general ledger.

Welcome. You are among survivors.

Two philosophies walk into a codebase

ABL: language is the platform

Data, transactions and logic in one compiler.

The decisions are already made.

Thursday's job ships on Thursday.

Go, TS, Rust: language is the kernel

Small core, enormous ecosystem.

The decisions are yours.

So is the three-week ORM argument.

One sold you a house. The other sold you excellent tools and a field.

The cast, one hire later

Brenda and Dave have a new colleague.

Brenda

Builds it in ABL

“Give me until Thursday.”

Ships the business rule.
Never sized a connection pool.

Dave

Knows how it works

“Don’t touch that. I’ll explain later.”

Still explaining it.
Now in a second language.

Zoe

New hire, Go and TypeScript

“Where’s the connection?”

Asks the right question.
Assumed to be rude.

Barry wants it by Friday. Barry does not care which compiler.

Ground rules

What this is not

A rewrite pitch

A benchmark

A eulogy

What this is

Same job, three languages

Numbers measured or sourced

Rudeness, evenly distributed

If a number flatters Go, ask how I got it. Zoe will do the same for ABL.

ABL brought a dictionary

1,982

alphabetic keyword spellings

OpenEdge 12.8.8 compiler table, extracted by Julian

Some of you have seen this number. It hasn’t got any smaller.

Go brought a Post-it note

25

reserved keywords, same count in 2012 as today

```
break case chan const continue default defer else fallthrough for func go goto
if import interface map package range return select struct switch type var
```

You could fit Go's reserved words on one slide. I just did.

1,982 versus 25 is a rigged fight

ABL 12.8.8

1,982 alphabetic spellings in the compiler table.

Most of them are not reserved.

Go, TypeScript and Rust

Go: 25 reserved keywords, 44 predeclared names.

TypeScript: 36 reserved, 9 more in strict mode, 84 in its scanner.

Rust: 39 strict, 14 reserved for later.

Different rulers. Quote 1,982 against 25 and you’re selling something.

Go didn’t lose the words. It moved them.

5,872

exported names in Go’s standard library

Across 168 packages, counting top-level names only

14,567 with methods and fields. Nobody calls that a small language either.

Same job, different shelf

In ABL, the compiler has

FOR EACH, FIND
TRANSACTION, UNDO
TEMP-TABLE
EXPORT, IMPORT
BREAK BY, ACCUMULATE
FORM, DISPLAY

In Go, you reach for

database/sql, plus a driver
BeginTx, Commit, Rollback
Structs, slices and maps
encoding/csv
slices.SortFunc and a loop
html/template, or a team

Everything on the left is sold as a language. Everything on the right, as a choice.

The whole keyword argument, in one line

A keyword is a library you can’t uninstall

A library is a keyword you have to choose, vet, pin and patch.

Pick your poison. Both come with a maintenance plan.

1981: applications without programmers

The bet

Business people can build business systems.

So: English-like syntax, with the database inside the language.

The world

No package manager.

No public repositories.

Small machines, fewer programmers.

You couldn’t npm install in 1984. You could barely dial a modem.

The architectural reason

The compiler was the only distribution channel

A keyword reached every customer, in the box, supported, documented and compatible.

No stdlib. No registry. Just the next release.

Every era left words behind

The problem

1980s: character terminals

1993: Windows and GUI

Late 1990s: web, remote calls

2000s: structured data

2006: object orientation

Words it left

FORM, FRAME, DISPLAY, PROMPT-FOR

WINDOW, BROWSE, WIDGET-HANDLE

WEB-CONTEXT, ASYNCHRONOUS

DATASET, DATA-SOURCE, XML

CLASS, METHOD, INHERITS, INTERFACE

Reading the keyword table is archaeology. You can date the layers.

The API lives in the keyword table

Attributes and methods are language vocabulary too

```
hBuffer:buffer-field("creditLimit"):buffer-value = 0.

hQuery:query-prepare("for each customer no-lock").
hQuery:query-open().
hQuery:get-first().
```

In Go, that's a package you can ignore. In ABL, it's part of the language.

Words only go in

Why they can’t leave

Remove a word and you break somebody’s 1994.

Compatibility is the product.

Why most aren’t reserved

Reserve a new word and you break a field name.

So: unreserved words, and the -k forget list.

Somewhere, a 1992 field is named after a keyword from 2026.

So, is it bloat?

1,982 is an audit log

Every word is a customer who asked,
and a vendor with no other way to answer.

Forty years of “can it also do this?” Yes. It could. In the compiler.

2007: a language born waiting for a build

45 minutes

to build one large Google binary

Go was conceived in late 2007, reportedly during one of those builds

ABL came from a warehouse. Go came from a very long coffee break.

Two languages, two scarcities

ABL, 1981: for Brenda

Scarce: programmers and memory.

Small team, one business, one database.

Optimised for writing.

Go, 2007: for Google

Scarce: build time and shared understanding.

Huge teams, many services, multicore.

Optimised for reading.

ABL assumes one person knows everything. Go assumes nobody does.

Why so few words is architecture

Small is a build-time and tooling decision.

The parser

Parses with no symbol table.

So tools are easy to write.

gofmt, gopls, vet.

The network

Packages travel by URL.

No compiler release needed.

Go left the words out because code could finally travel.

The reader

One loop keyword. One format.

Unused import? Compile error.

That rule attacks the 45-minute build directly.

ABL’s compiler arrived with everything. Go’s arrived with an internet connection.

The bill for a small language

Proposed: check (2018), try (2019), ? (2024). Accepted: none of them.

Everywhere you look

```
if err != nil {
    return err
}
```

Go team blog, June 2025

“For the foreseeable future, the Go team will stop pursuing syntactic language changes for error handling.”

ABL developers, meanwhile, have had UNDO, RETRY since before Rob Pike was bored.

Same disease, different hospital

ABL's treatment

New words stay unreserved.
Collisions go on the -k forget list.

```
class Box<T as Object>:
```

Generics, OpenEdge 13.0

Go's treatment

New words are predeclared names.
You may shadow them. It compiles.

```
len := 3; min := "ok"
```

any (1.18), min and max (1.21)

Both languages are terrified of your variable names.

TypeScript: the third answer

It didn’t start a new language. It made an unavoidable one bearable.

The limit

JavaScript runs in every browser.

You can’t replace it.

So don’t try.

The move

Types layered on top.

Checked at build, erased at run.

A type is a claim, not a door check.

The result

The ecosystem is the language.

Plus 84 keywords, thanks for asking.

It owns the browser.

Compatibility again. TypeScript is 30 years of JavaScript with a good lawyer.

Rust: the fourth answer

What if the compiler simply said no?

The limit

C and C++ are fast and unsafe.

A garbage collector doesn't always fit.

Graydon Hoare, 2006.

The move

Every value has one owner.

Checked at compile, with no GC.

Rust 1.0, May 2015.

The result

Whole bug classes don't compile.

Speed and safety, paid in learning.

Google on Android, 2025: 1000x lower memory bug density.

ABL came from a warehouse, Go from a long build. Rust reportedly came from a broken lift.

Rust found the exit: editions

Each crate picks a year

2015, 2018, 2021 or 2024, set in Cargo.toml.

Crates of different editions still link together.

So keywords can arrive

async in 2018. gen reserved in 2024.

Someone else’s old name? Write r#async.

ABL’s words only go in. Rust’s come with a calendar.

Barry has a job

Give every UK customer a 5% credit limit uplift

Skip anyone on credit hold. Audit each change.
Commit per customer. Stop on error. Keep what's done.

A rule per row, the kind of job ABL was built for.

Brenda’s version

Nine lines. Written before her coffee went cold.

```
for each customer
    where customer.country = "UK"
    exclusive-lock on error undo, throw:
  if customer.onHold then next.
  customer.creditLimit = customer.creditLimit * 1.05.
  create auditLog.
  assign auditLog.custNum = customer.custNum
         auditLog.note    = "5% uplift".
end.
```

Connection, types, lock, transaction, undo. All implied.

Zoe’s version does the same job

ABL

9 lines

Compiled against the schema

Go

60 lines

With 8 separate err checks

Same behaviour, tested on PostgreSQL 16. Zoe’s coffee went cold.

One customer, in Go

```
tx, err := db.BeginTx(ctx, nil)
if err != nil {
  return err
}
defer tx.Rollback()

var onHold bool
err = tx.QueryRowContext(ctx,
  `SELECT on_hold FROM customer
  WHERE cust_num = $1
  FOR UPDATE`, n).Scan(&onHold)
if err != nil {
  return err
}
if onHold {
  return nil
}
```

```
if _, err := tx.ExecContext(ctx,
  `UPDATE customer
  SET credit_limit = credit_limit * 1.05
  WHERE cust_num = $1`, n); err != nil {
  return err
}
if _, err := tx.ExecContext(ctx,
  `INSERT INTO audit_log (cust_num, note)
  VALUES ($1, '5% uplift')`,
  n); err != nil {
  return err
}
return tx.Commit()
```

Everything ABL implied, written down. Dave would hate it. Dave's successor won't.

Who did the work?

ABL

Connection: the session has it
Field types: schema, at compile
Row lock: EXCLUSIVE-LOCK
Transaction: the block
Rollback: UNDO, by default

Go

Connection: sql.Open, pool, driver
Field types: Scan into variables
Row lock: FOR UPDATE, in a string
Transaction: BeginTx and Commit
Rollback: defer tx.Rollback()

ABL gives you the answers. Go makes you show your working.

Spell a column wrong

ABL

The compiler refuses. The schema is part of the language.

```
customer.creditLimt = 0.
```

Fails on your desk.

Go

go build: fine. go vet: fine. It’s only a string.

```
SET credit_limt = 0
```

Fails at run time.

pq: column "credit_limt" of relation "customer" does not exist. Real output.

Rust gets ABL’s best trick back

sqlx

The SQL is checked against the database while compiling.

```
SET credit_limt = ...
```

Fails on your desk again.

The cost

A macro, plus a live database or saved metadata at build.

```
37 lines, not 60
```

Same job, same result.

error: column "credit_limt" of relation "customer" does not exist. From cargo build.

And here’s the bill for implicit

Nobody typed TRANSACTION. So where does it end?

```
find first customer exclusive-lock.
customer.creditLimit = 0.

run send-statement.p.   /* still locked */
run wait-for-gateway.p. /* still locked */
```

Go makes you type the boundary. ABL makes you know it. Ask Dave.

Where ABL still excels

In the language

Locks, scope and undo are grammar, in the block where the rule lives.

Table and field references are checked at compile, not in production.

In the box

Temp-tables, ProDataSets and BREAK BY, with no extra dependency.

Your code still compiles. Few stacks can say that about last year.

Forty years of month-ends beats most start-ups' test suites.

A database in your pocket

An indexed in-memory table, then control-break totals

```
define temp-table ttSale no-undo
  field region as character
  field amount as decimal
  index idxRegion region.

for each ttSale break by ttSale.region:
  accumulate ttSale.amount (total by ttSale.region).
  if last-of(ttSale.region) then
    display ttSale.region
      (accum total by ttSale.region ttSale.amount).
end.
```

In Go this is a map, a sort, and a bug in the last group.

Now Zoe’s home ground: waiting

One keyword: go

```
var wg sync.WaitGroup
for i := range 100000 {
  wg.Add(1)
  go func() {
    defer wg.Done()
    callSlowService(i)
  }()
}
wg.Wait()
```

Measured

100,000

concurrent 100ms waits

0.36 seconds, about 145MB, in a small cloud sandbox

ABL’s answer is more sessions, more agents and a capacity plan.

Copy one file. That’s the deployment.

Go: 8MB

One static HTTP server binary, no runtime to install.

Built for Linux, Windows, macOS, ARM and AIX from one machine.

ABL: a site

Runtime, licence, r-code, PROPATH, .pf files.

Rock solid. Nobody’s dropping it on a customer’s box for a quick job.

scp is a deployment strategy again. Don’t tell the DevOps team.

Go’s opinions moved into the toolchain

One command, in the box

```
go test -race
go fmt
go vet
go mod tidy
go tool pprof
```

What that buys

Tests, format and profiling need no meeting.

Every project starts the same way.

gofmt ended the indentation war. We’re still SHOUTING OUR KEYWORDS.

TypeScript’s home game: JSON and the glass

ABL

```
oTags = new JsonArray().
oTags:Add("uk").
oBody = new JsonObject().
oBody:Add("name", cName).
oBody:Add("limit", dLimit).
oBody:Add("tags", oTags).
```

TypeScript

```
const body = {
  name,
  limit,
  tags: ["uk"],
};
```

JSON is TypeScript’s mother tongue. In ABL, it’s a phrasebook.

Packages needed for a web hello

Measured this week

65

TypeScript

Express 5 on npm

Including everything it pulls in

0

Go

net/http

Already in the standard library

0

ABL

PASOE web handler

Though you'll need a licence

ABL and Go agree: hello should not need a supply chain. One still needs a purchase order.

The scorecard, with no winner

Productivity, then maintainability, then flexibility.

ABL

Fastest for data-heavy business rules

Stable, but needs Dave

Narrow, deep

Go

Fastest for services and tools

Explicit, dull, readable

Wide, thin in business data

TypeScript

Fastest for UI and JSON

Strong types, restless dependencies

Everywhere, including places it shouldn’t be

“Which language is best?” is a question from someone who owns only one hammer.



Overqualified for the wrong jobs

We keep asking the CFO to fix the printer

HTTP gateways. Log shippers. File watchers.
JSON proxies. Command-line tools.

She can do it. It’s a shocking use of the salary.

Give each one the right job

The job	Give it to
Order posting, pricing, stock	ABL: it knows the rules
Reports on business records	ABL: BREAK BY exists
Gateways, webhooks, fan-out	Go: it’s mostly waiting
Agents, exporters, CLI tools	Go: one small file
Browser UI, JSON shaping	TypeScript: it owns the glass
Hot paths, no garbage collector	Rust: fast, safe, no GC
Bulk changes, analytics	SQL: think in sets

SQL snuck on. It was always invited. It just didn’t want to come to the fight.

One system, each language in its seat

Browser → API edge → PASOE → OpenEdge DB

TypeScript

Browser UI

It shows.

Replaceable when fashions change.

Go

API edge, auth, fan-out, agents

It waits.

Agents don’t own business rules.

ABL

PASOE: rules, transactions

It decides.

The only thing that changes business records.

Rules at the centre. Everything else is a guest and uses the front door.

Four ways across the border

Use it for

PASOE REST: anything that writes

Message queue: events

SQL over ODBC: read-mostly reports

Spirit Medium: live queries in Postgres

Watch out for

Sessions held while waiting

Retries, ordering, duplicates

Skips ABL triggers

For querying, not your write path

Pick the seam first. The language argument can wait until the pub.

The trap in the seam

“SQL clients and servers do not execute ABL-based triggers.”

Go writing straight to OpenEdge over SQL
walks past every trigger you own.

Writes go through ABL. Dave put those triggers there for a reason. Probably.

The boundary is the architecture

What makes this hard

```
for each customer:
  /* rule */
  /* database */
  /* UI */
  /* integration */
end.
```

What makes this easy

Small ports. Rules that don’t know who called them.

ABL classes can do this today.

If your FOR EACH knows about HTTP, no language will save you.

Three questions before moving anything

Apply these to one real program on Monday.

1

Does it make business decisions?

Yes: it stays, behind a boundary.

Knowledge is the asset.

2

Does it spend its life waiting?

Yes: it’s a candidate to move.

Usually cheaper to move.

3

Can two people explain it?

No: moving it only relocates Dave.

Get the explanation first.

If you can’t explain it in ABL, you can’t rewrite it in anything.

About that rewrite in Go

Forty years of business rules, retyped by someone who wasn’t there

Edges first. Seams second. Core last, if ever.

The new system will be finished right after the old one stops changing.

whoGloo's own example

Specialist tools beside OpenEdge, not instead of it.

Wraith

One agent per host, beside OpenEdge.

Observes and executes.

Buffers on disk when the link drops.

Nexus

Telemetry in TimescaleDB.

Stores and correlates.

In OpenTelemetry form.

Spirit Medium

OpenEdge tables from PostgreSQL.

Real-time queries.

A PostgreSQL extension.

Wraith and Nexus: no ABL rewrite, and no change to OpenEdge itself.



ABL isn’t outdated.
It’s overqualified for the wrong jobs.

The future is knowing where each tool belongs, and using it there.

Julian Lyndon-Smith

Sources and method

ABL and OpenEdge

ABL Keyword Index
Keyword Forget List (-k)
Control the size of a transaction
OpenEdge SQL: work with triggers
PASOE: ABL sessions
User-defined generics in ABL
FINPUG: A History of Progress

Go and history

Rob Pike, Go at Google (2012)
Rob Pike, Less is exponentially more
Go blog: error handling syntax (2025)
Go FAQ and language spec
Rust Reference and Edition Guide
Google: Rust in Android (2025)
Fourth-generation language, Wikipedia

Measured Nov 2026: Go 1.24, TypeScript 5.9, Rust 1.97, PostgreSQL 16. ABL count: Julian, 12.8.8.