whogloo

ABL:Still got it, just needs a wingman

Eight ways to extend the ABL
with useful friends

Julian Lyndon-Smith

Eight routes, eight separate chapters

01 **Native DLLs**
Call a C function

02 **Modern .NET**
Use a managed class or control

03 **Pipes**
Talk to a local worker

04 **OS-COMMAND**
Run an external program

05 **Starting ABL**
Let another program invoke ABL

06 **Sockets**
Exchange bytes over TCP

07 **HTTP and PASOE**
Call services in either direction

08 **Message buses**
Publish work and receive callbacks

01

Native DLLs and .so libraries

ABL calls an exported C function inside its own process.

C does the heavy lifting. ABL gets the applause.

A native call from ABL



Out of ABL Customer code, file path, number or buffer

Back into ABL A return code, a number or bytes in a buffer

Best fit: an existing local C API. A native crash can also bring down the AVM.

A real SQLite entry point

ABL declares the C function, then calls it

```
DEFINE VARIABLE version AS INTEGER NO-UNDO.
PROCEDURE sqlite3_libversion_number
  EXTERNAL "sqlite3.dll" CDECL:
  DEFINE RETURN PARAMETER result AS LONG.
END PROCEDURE.
RUN sqlite3_libversion_number (OUTPUT version).
```

Back into ABL The deployed SQLite version as an integer

Windows: sqlite3.dll. Linux: use the actual shared-library name, often libsqlite3.so.0.

A useful C bridge to a SQLite cache

Custom C facade: customer_terms

```
SET-SIZE(reply) = 256.
RUN customer_terms ("ACME", reply, 256, OUTPUT rc).
IF rc = 0 THEN terms = GET-STRING(reply, 1).
SET-SIZE(reply) = 0.
```

Inside C: parameterized SQLite query

```
SELECT terms FROM customer_cache WHERE code = ?1
```

Example result: NET30. ABL applies its own payment policy. The facade is your code.

Native libraries worth borrowing

Library	Useful external work	Back into ABL
libcurl	HTTP or file transfer	Response / status
libarchive	Read or create an archive	Files / archive path
libsodium	Encrypt or verify data	Bytes / validity result
MagickWand	Resize catalogue images	Thumbnail file
Zint	Generate a product barcode	Barcode image

A small C facade makes awkward APIs consumable. Match AVM bitness and memory ownership.

02

.NET Core

ABL creates managed objects and calls their methods.

Yes, Linux too. The .dll extension does not get a vote.

A .NET class inside the ABL application

Native DLL

PROCEDURE … EXTERNAL
RUN calls a C entry point
Values and buffers cross the ABI

.NET assembly

USING … FROM ASSEMBLY
NEW creates a managed object
ABL calls methods and properties

The filename can end in .dll in both cases. The calling mechanism is different.

A real .NET method call from ABL

A product-code check using System.Text.RegularExpressions

```
USING System.Text.RegularExpressions.Regex
  FROM ASSEMBLY.
DEFINE VARIABLE valid AS LOGICAL NO-UNDO.
valid = Regex:IsMatch("SKU-104", "^[A-Z]+-[0-9]{3}$").
```

Back into ABL TRUE. ABL can accept or reject the product code.

One class call buys a tested regular-expression engine. Your business rules stay in ABL.

An Excel export with ClosedXML

ABL passes rows and receives a workbook path

```
USING Acme.Reporting.ReportBridge FROM ASSEMBLY.
DATASET dsOrders:WRITE-JSON("FILE", "orders.json").
filePath = ReportBridge:CreateExcel(
  "orders.json", "orders.xlsx").
```

Inside your C# facade: actual ClosedXML calls

```
worksheet.Cell("A2").Value = 1042;
workbook.SaveAs("orders.xlsx");
```

The user receives orders.xlsx. ReportBridge is your facade, not a built-in ABL API.

.NET libraries for documents and data

Library	Useful work	ABL receives
ClosedXML	Create or read XLSX	Workbook / imported rows
PDFsharp / MigraDoc	Create an invoice PDF	Finished document
CsvHelper	Map a supplier CSV	Rows to validate
Npgsql	Query PostgreSQL	Rows / lookup result
Microsoft.Data.Sqlite	Read a local SQLite file	Cached values

Use suitable APIs directly or put a small C# facade around the package.

.NET libraries for integration

Library	A feature to add	ABL input / result
MailKit / MimeKit	Send or collect email	Invoice / mail result
ZXing.Net	Read or generate a barcode	Product code / image
SSH.NET	Exchange supplier files	Remote path / local file

Image bindings, async APIs and callbacks may need extra facade work.

A .NET library or a separate .NET service

Inside the AVM

ClosedXML creates a workbook
Npgsql performs a local lookup

Separate process

ASP.NET Core shares a credit service
Blazor presents a browser UI
The service calls PASOE for ABL rules

Linux support does not magically port Windows controls, fonts or native dependencies.

03

Pipes

ABL exchanges messages with another local process.

Two processes. One conversation. Very little ceremony.

Process streams and named pipes

Process streams

ABL starts a child worker
Writes to the worker's stdin
Reads the worker's stdout
INPUT-OUTPUT THROUGH on UNIX

Named pipes

Processes start independently
UNIX FIFO paths carry bytes
Usually one pipe each direction
Windows can use a .NET facade

A pipe carries bytes. Your worker and ABL agree on the message format.

A Go worker through UNIX streams

ABL writes one request and reads one response

```
INPUT-OUTPUT THROUGH lookup-worker UNBUFFERED.
PUT UNFORMATTED requestJson SKIP.
IMPORT UNFORMATTED replyJson.
INPUT-OUTPUT CLOSE.
```

Out `{"id":"42","code":"ACME"}`

In `{"id":"42","terms":"NET30"}`

The worker uses pgx. ABL decides how NET30 affects the order.

A named FIFO on Linux

ABL sends a request to an independently started worker

```
OUTPUT TO "/run/abl/request.pipe" APPEND.
PUT UNFORMATTED '{"id":"42","code":"ACME"}' SKIP.
OUTPUT CLOSE.
INPUT FROM "/run/abl/reply-42.pipe" NO-ECHO.
IMPORT UNFORMATTED replyJson.
INPUT CLOSE.
```

Back into ABL A correlated reply for request 42

Create the FIFOs first. Opening either end can block until its peer is ready.

```
DEFINE STREAM sReq.
DEFINE STREAM sResp.
DEFINE VARIABLE cKey    AS CHARACTER NO-UNDO.
DEFINE VARIABLE cAnswer AS CHARACTER NO-UNDO.
DEFINE VARIABLE i       AS INTEGER   NO-UNDO.

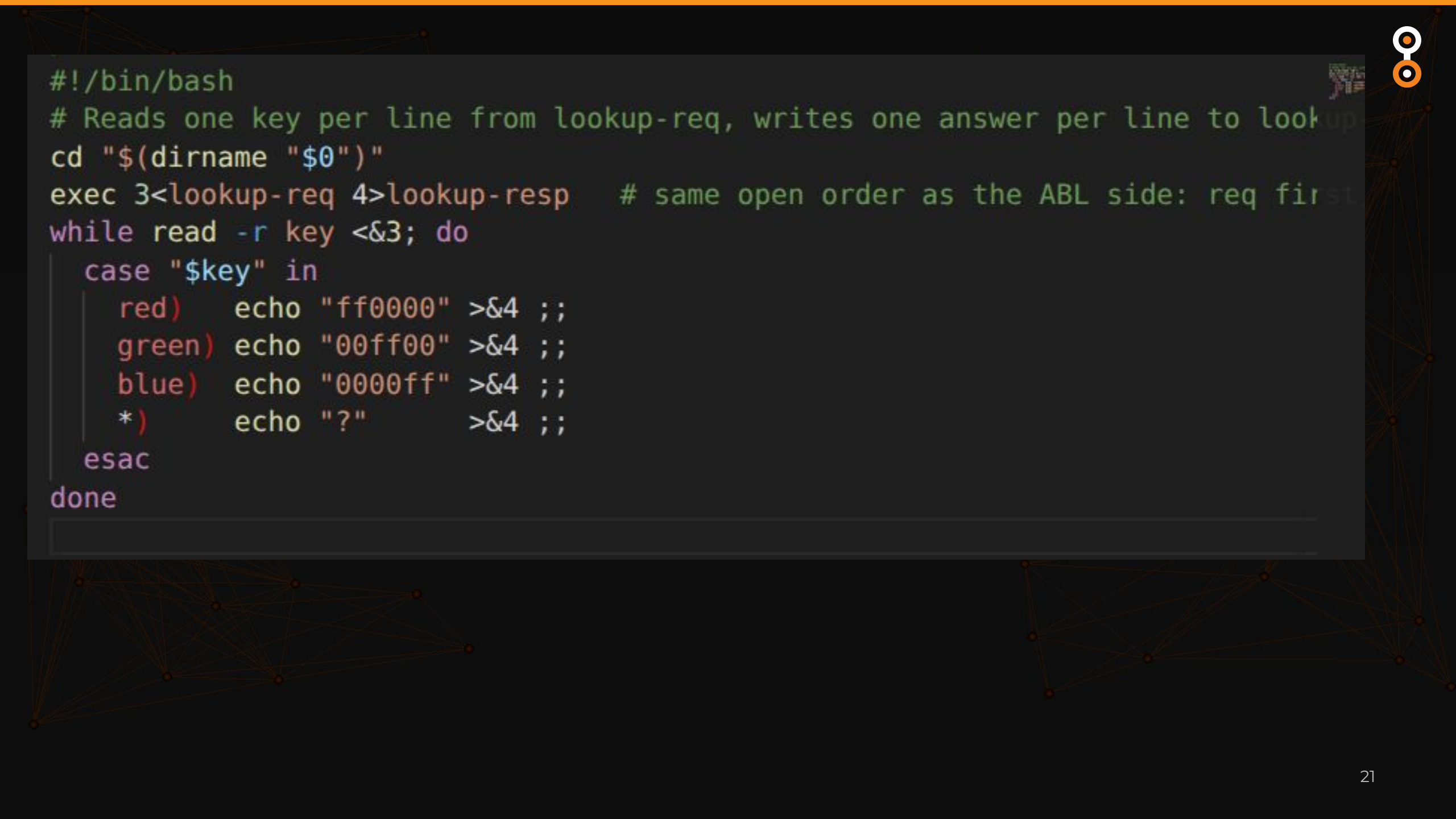
OUTPUT TO "/dev/stdout" UNBUFFERED.   /* batch mode on a terminal has no default

/* Open order must match the worker: req (write) first, then resp (read). */
OUTPUT STREAM sReq  TO   ./lookup-req UNBUFFERED.
INPUT  STREAM sResp FROM ./lookup-resp.

DO i = 1 TO 4:
  cKey = ENTRY(i, "red,green,blue,pink").
  PUT STREAM sReq UNFORMATTED cKey SKIP.
  IMPORT STREAM sResp UNFORMATTED cAnswer.
  PUT UNFORMATTED cKey " -> " cAnswer SKIP.
END.

OUTPUT STREAM sReq CLOSE.   /* worker sees EOF and exits */
INPUT  STREAM sResp CLOSE.
QUIT.
```

```
#!/bin/bash
# Reads one key per line from lookup-req, writes one answer per line to lookup-resp
cd "$(dirname "$0")"
exec 3<lookup-req 4>lookup-resp   # same open order as the ABL side: req first
while read -r key <&3; do
  case "$key" in
    red)   echo "ff0000" >&4 ;;
    green) echo "00ff00" >&4 ;;
    blue)  echo "0000ff" >&4 ;;
    *)     echo "?"      >&4 ;;
  esac
done
```



Useful libraries behind a pipe worker

Worker uses	ABL sends	ABL receives
Go + pgx	Customer code	PostgreSQL lookup JSON
Node + ExcelJS	Rows and output path	Workbook path
.NET + ClosedXML	Report job JSON	XLSX path / status
Node + sharp	Image path and dimensions	Resized image path

Each worker needs a small protocol adapter. These packages are not built-in pipe services.

A pipe conversation needs a few rules

Successful lookup

Request 42 asks for ACME
Worker flushes a complete JSON line
ABL matches reply 42
ABL applies the returned terms

Worker vanished

ABL detects EOF or a deadline
Marks the lookup unavailable
Keeps the order on hold
Restarts or reports the worker

Shared writers need routing and framing. Otherwise the replies become a raffle.

04

OS-COMMAND

ABL starts an external program to do a job.

The operating system already has staff. Give them work.

An external program called by ABL

ABL prepares	Writes job-42.json with input and output paths
The program works	A Go report-worker generates report-42.xlsx
ABL continues	Reads result-42.json and uses the returned file

```
OS-COMMAND SILENT
  VALUE("report-worker --job job-42.json").
```

report-worker is your executable. ABL uses its result file.

A PDF pack with qpdf

ABL asks qpdf to combine a quotation and its terms

```
OS-COMMAND SILENT VALUE(
   "qpdf --empty --pages quote.pdf terms.pdf -- pack.pdf").
```



The user gets one document pack. ABL records the outcome after checking job status.

Useful command-line tools for ABL

Tool	Job started by ABL	Useful output
Tesseract	Read a scanned delivery note	OCR text to validate
ImageMagick	Resize product photographs	Thumbnails
LibreOffice headless	Convert a document to PDF	PDF file
7-Zip	Bundle an export directory	Archive file
rclone	Copy documents to storage	Transfer status

Best fit: an occasional job with clear files in and out. Existing tooling does the heavy work.

A command result ABL can actually use

Your wrapper captures the exit code and writes the result file.

Success: result-42.json

```
{
  "ok": true,
  "path": "pack.pdf",
  "exitCode": 0
}
```

Failure: result-42.json

```
{
  "ok": false,
  "error": "INPUT_MISSING",
  "exitCode": 2
}
```

ABL attaches the file

ABL reports the failed job

OS-ERROR is not the child exit code. SILENT does not make a failed command successful.

05

Starting an ABL session

Go, Node or .NET starts OpenEdge and asks ABL to run a job.

Sometimes the new tech needs to call the grown-up.

A caller starts a fresh AVM




```
_progres -b -pf app.pf -p connection-probe.p -param probe-42.txt
```

Into ABL SESSION:PARAMETER contains probe-42.txt

Back to the caller A result file plus the process outcome

Go starts the ABL probe

The arguments are explicit. The job has a deadline.

```
cmd := exec.CommandContext(ctx, "/usr/dlc/bin/_progres",
  "-b", "-pf", "app.pf",
  "-p", "connection-probe.p", "-param", "probe-42.txt")
err := cmd.Run()
```

On success Go reads probe-42.txt and forwards the metric.

This starts ABL. The OpenEdge procedure performs the database work.

Other callers can use the same ABL job

Caller / facility	A useful ABL job	Result returned
Go os/exec	OpenEdge VST probe	Metric / probe file
Node child_process.spawn	Supplier CSV import	Accepted / rejected rows
.NET System.Diagnostics.Process	Batch reconciliation	Result file / exit status
cron / Task Scheduler	Nightly ABL report	Report path / job log

The caller chooses the job. ABL owns validation, database access and business rules.

A new session or a reused ABL service

Fresh ABL session

Scheduled probes
Occasional imports
Batch tools with a file contract
Caller observes process completion

PASOE operation

Repeated order requests
Interactive web applications
Shared business services
Managed ABL session reuse

Process startup has a cost. Your laptop is a suspiciously optimistic test lab.

06

Sockets

ABL exchanges bytes with a program or device over TCP.

Congratulations. You now own the protocol.

A TCP conversation with ABL



Out of ABL A printer command or a framed JSON request

Back into ABL Device status or a correlated service reply

A TCP read can contain part of a message. Keep reading until the complete frame arrives.

ABL sends a label to a network printer

Socket excerpt: a raw ZPL printer listener

```
CREATE SOCKET hSock.
ok = hSock:CONNECT("-H 192.0.2.10 -S 9100").
zpl = "^XA^FO50,50^ADN,36,20^FDOrder 1042^FS^XZ".
SET-SIZE(wire) = LENGTH(zpl, "RAW") + 1.
PUT-STRING(wire, 1) = zpl.
IF ok THEN ok = hSock:WRITE(wire, 1, LENGTH(zpl, "RAW")).
```

Back into ABL Write status. Check the printer for job completion.

192.0.2.10 is an example address. Cleanup and error handling are omitted from this excerpt.

A Go socket service can reply to ABL

ABL sends one complete line

```
{"id":"42","operation":"credit","code":"ACME"}
```

Go replies with one complete line

```
{"id":"42","availableMinor":2500000,"currency":"GBP"}
```

ABL buffers reads, matches id 42 and applies its credit rule.

A newline terminates each JSON frame. READ-RESPONSE tells ABL that bytes are available.

Useful socket counterparts

Counterpart	A useful connection	ABL input / result
Go net + pgx	Internal credit worker	Customer / credit JSON
Node net + sharp	Image worker	Path / thumbnail path
.NET TCP service	Existing .NET calculation	Values / calculation result
Network label printer	ZPL on its configured port	Label bytes / device status

Best fit: a device or existing stream protocol. A new HTTP service usually needs less plumbing.

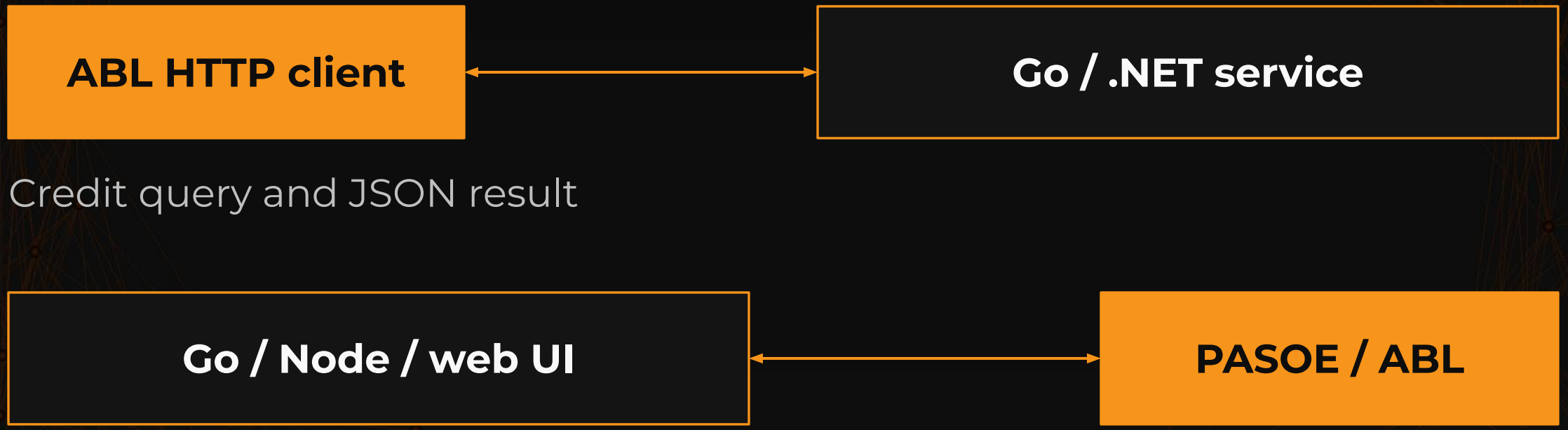
07

HTTP and PASOE

ABL calls external services. External services call ABL.

An API does not require a personality transplant.

HTTP works in both directions



URLs name operations. ABL continues to own its permissions and database transaction.

ABL calls an external credit service

OpenEdge.Net.HTTP request excerpt

```
USING OpenEdge.Net.HTTP.*.
request = RequestBuilder:Get(
  "https://credit.example/credit/ACME"):Request.
response = ClientBuilder:Build():Client:Execute(request).
```

Response body

```
{"availableMinor":2500000,"currency":"GBP"}
```

ABL checks the HTTP outcome, then compares the returned credit with its order policy.

PostgreSQL in Go, business rules in ABL

Inside the Go service: a parameterized pgxpool query

```
err := pool.QueryRow(ctx,
  "SELECT available_minor, currency FROM credit WHERE code=$1",
  "ACME").Scan(&availableMinor, &currency)
```

Success ABL receives a credit amount and currency

Lookup failed ABL keeps the order on hold and reports the error

External services with useful ABL results

Service	A feature for an ABL app	ABL receives
Amazon S3	Store invoice documents	Object key / file link
Microsoft Graph	Send an invoice email	Acceptance / error
Keycloak	Authenticate a user	Identity / claims
Stripe	Request and track payment	Payment state / event
Shippo	Buy a shipping label	Label / tracking

Use HTTP directly or let a Go/.NET facade handle the provider SDK and credentials.

A service calls an ABL operation in PASOE

Example application endpoint: POST /api/orders/1042/release

```
{"requestId":"release-42"}
```

Inside ABL Checks permission, stock and the order state

Back to caller {"orderId":1042,"status":"ready-to-ship"}

The service requests a business operation. ABL decides whether it can succeed.

A .NET web UI with ABL behind it



Web UI ASP.NET Core runs on Windows or Linux and calls PASOE

Order screen GET /api/orders/1042 returns the order details from ABL

The web app draws the screen. ABL checks access and returns the permitted data.

HTTP or .NET Open Client

HTTP / JSON

Go, Node, .NET or browser gateways
Deliberate business URLs
Ordinary JSON and HTTP status
Useful for shared external APIs

.NET Open Client

A .NET caller invokes ABL
Generated proxy API
Supported ABL parameter types
Useful for existing typed integrations

Choose the contract your callers need. Both routes can put ABL business logic behind PASOE.

08

Message buses

ABL publishes work. A subscriber can later call back into ABL.

Eventually consistent. Immediately opinionated.

Work queues and published events

Work queue

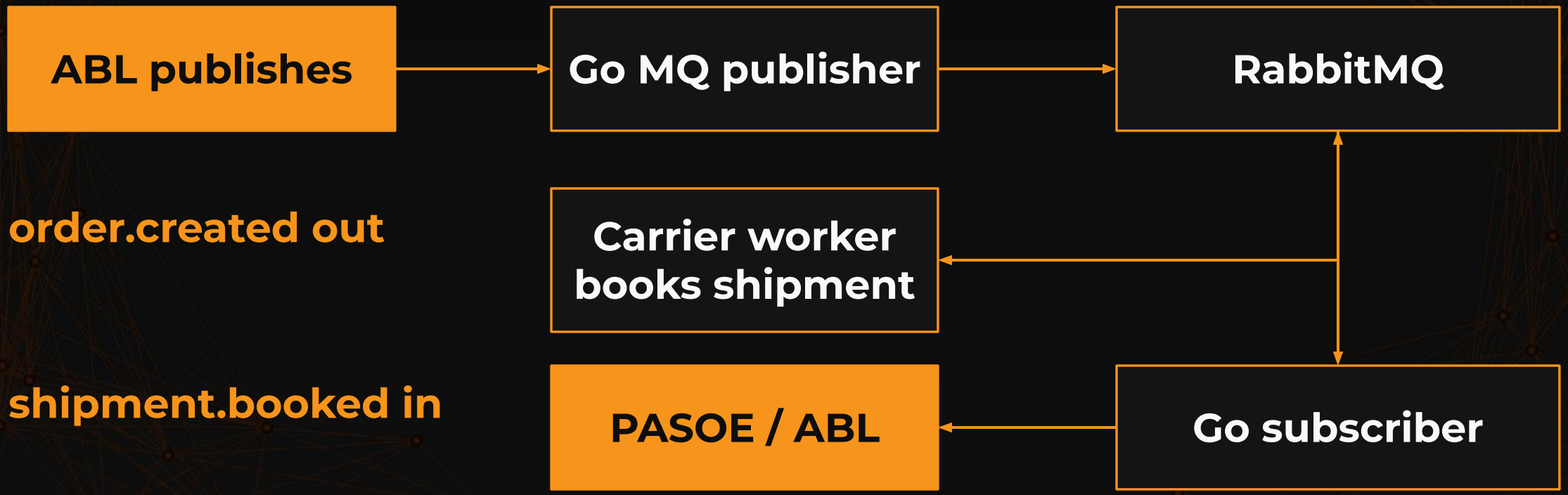
Import supplier file 42
One available worker handles the job
ABL validates and imports the rows

Published event

Order 1042 has been committed
Shipping and invoicing subscribe
Each performs its own work

An event needs a delivery and failure policy. The broker does not invent one for you.

A Go MQ bridge connects ABL out and in



Go owns the subscription. Each received shipment event invokes an ABL operation.

The payloads on that round trip

ABL publishes order.created

```
{
  "eventId": "order-42",
  "orderId": 1042,
  "customer": "ACME"
}
```

Go calls POST /api/shipments

```
{
  "eventId": "ship-42",
  "orderId": 1042,
  "tracking": "TRACK-1042"
}
```

ABL records the shipment once and returns its business result to the subscriber.

Broker clients that can serve an ABL app

Broker	Useful client option	ABL participates through
Kafka	OpenEdge.Messaging.Kafka	Supported ABL API
RabbitMQ	amqp091-go / RabbitMQ.Client	Go bridge / .NET facade
NATS / JetStream	nats.go	Go publish / subscription
Kafka	franz-go	Go service / PASOE call
MQTT	Paho mqtt.golang	Device events forwarded to ABL

A bridge is an option. Check the ABL messaging APIs already available in your version.

Retries without duplicate business actions

Publishing failed

ABL has committed the order
A pending event remains in its database
The publisher retries the event
The order does not disappear

Callback arrived twice

ABL recognizes ship-42
Returns the earlier result
Keeps one shipment record
The subscriber can finish safely

A retry should repair delivery, not create order 1042’s secret twin.

Which mechanism fits the feature?

Mechanism	A useful first fit
Native DLL / .so	A local C library or device API
Modern .NET	A library method or Windows control
Pipes	A local worker with request / reply
OS-COMMAND	An occasional file-processing job
Starting ABL	A probe or controlled batch job
Sockets	An existing TCP device / protocol
HTTP / PASOE	A shared business service in either direction
Message bus	Queued work or a later business callback

Telemetry gives the slow bit an alibi

Illustrative trace	Duration	Meaning
ABL release-order	124 ms	Whole business operation
Go credit lookup	88 ms	Nested external call
PostgreSQL query	75 ms	Nested inside Go lookup

ABL / PASOE tracing + Go/.NET instrumentation

Spirit-style probes add OpenEdge facts. A shared trace shows where the request waited.

Ok, so how slow is this ?

#	What was timed	Library / command	Runs	Result
1	OS-COMMAND (bare cost of starting a shell)	true	20	**1.60 ms/call**
2	OS-COMMAND, system service query	service dbus status	20	**13.85 ms/call**
3	OS-COMMAND, system service query	systemctl is-active dbus	20	2.60 ms/call (not a real measurement, see note)
4	External .so, first call (loads the library)	libc.so.6	1	under 1 ms (too fast for ETIME)
5	External .so call	libc.so.6 getpid	100,000	**0.98 µs/call**
6	External .so, first call (loads the library and starts the Go runtime)	OTel bridge	1	**3–6 ms** (3 runs)
7	External .so call that does almost no work	OTel bridge SpiritGetLastError	100,000	**2.5–2.75 µs/call**
8	First span (bridge connects to the wraith on localhost:4317)	OTel bridge	1	**2–3 ms**
9	Same span pair, 100,000 back-to-back	OTel bridge	100,000	**2.77 µs per span**

Ok, so how slow is this ?

```
Timing for _progres -b -p quit.p, where quit.p contains only QU
```

Measure	Time
min	9.8 ms
median	11.0 ms
p90	12.8 ms
max	16.6 ms
100 sessions back to back	701 ms total, about 7 ms each



ABL still runs the business

Which wingman will you hire first?

Julian Lyndon-Smith