Taming the Legacy System Monster

ADAM HARLEY

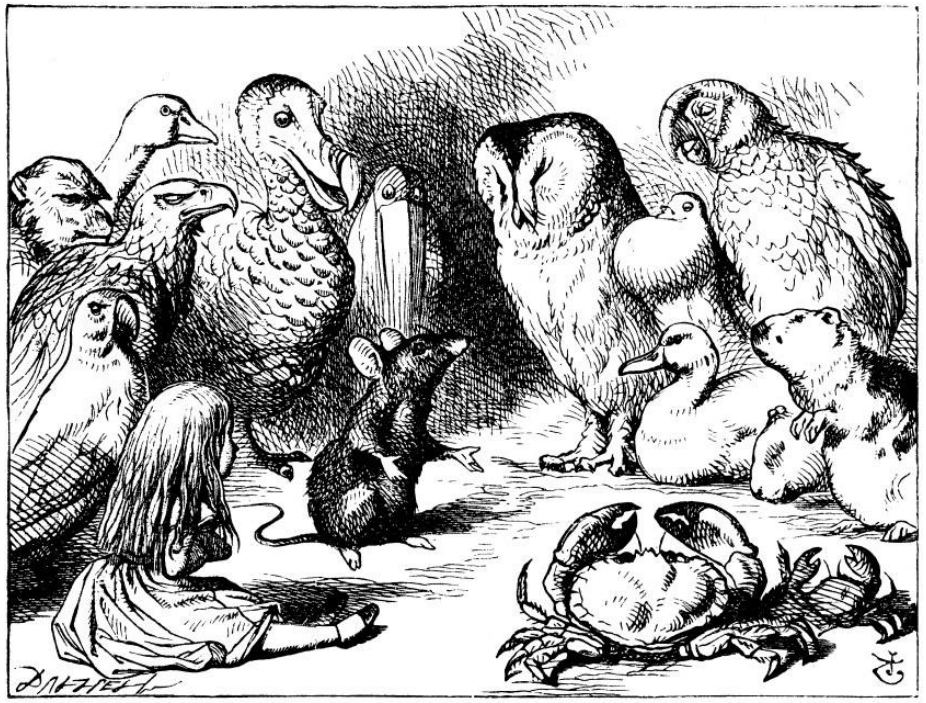
THE UNIVERSITY OF SHEFFIELD

Once Upon A Time...

In a land far, far away—deep in the uncharted North—intrepid developers and a foolhardy architect sought out the brave knight Sir Dave to learn how he alone tamed their kingdom's monster, knowing they had to master his craft before he retired and left the beast entirely to them.

They were soon forced to take up the reins themselves, charting a safe path toward the Land of Clouds while fending off the relentless, soul-crushing tides of “BAU”.

This is that story.



The Monster

- Combination of Bash scripts, Oracle PL/SQL packages, SFTP & SCP transfers, Cron jobs
- Underpins key university systems
- Never formally documented
- Strikes fear into developers and stakeholders

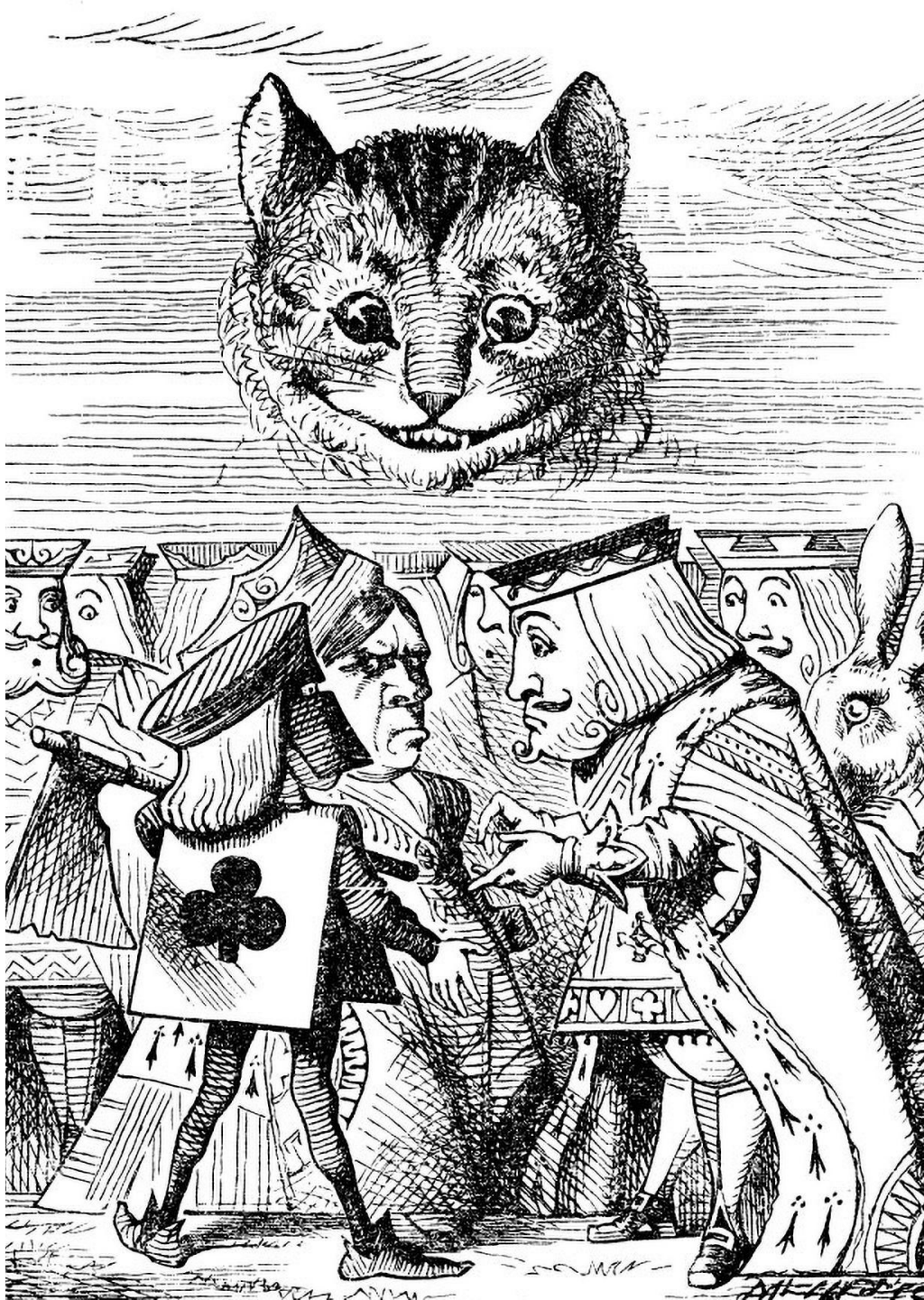
If we 'slay' the monster, we'd have to replace it, so let's just tame it instead

Sir Dave

- 70 years old
- Retired, then brought back to keep the system running
- Planning to retire for good this time, hopefully
- Knows exactly how to keep the legacy monster happy
- Living embodiment of the Single Point of Failure problem

Pesky Reality

- 3 months until Dave retires
- 2 junior developers
- 40 integrations across 10 systems
- Y2K era codebase
- 1970s version control system
- Limited documentation
- Panicking stakeholders
- 121-year-old organisation with £845 million annual income



Post-Fairytale: The Reality of “DaveOps”

Area	“DaveOps”	Target Model
Triage	Fix flaws in source data	Fix underlying system issue or document remedy
Stakeholder management	1:1 communication, no records	Group mailing lists with records of decisions and actions
Issue tracking	Only tracked in Dave’s head	Issue log recording project area, impact, priority & recorded in Jira if long-term work is necessary
Documentation	Sporadic and ad-hoc, from specific investigations	Formalised, in shared wiki resources using established templates for each subcomponent
Version control	SCCS, deltas, copy/paste	Git, branches/tags, CI/CD pipelines

Round-robin Documentation Building

Scoping

What do you have?

Bringing together disparate documentation sources

Analysing codebase at file level

Building foundational understanding of scale and timelines

Unpicking purpose

Why is it there?

Mapping out what the system is doing within the black box at a high level

Aiming to gain an understanding of why it was built as it was and what goes where

Identifying gaps in understanding for key systems

High level documentation

What is it doing?

Interrogating how subsystems interact

What dependencies are present

What formats are used

What is the nature of the data being handled

Low level documentation

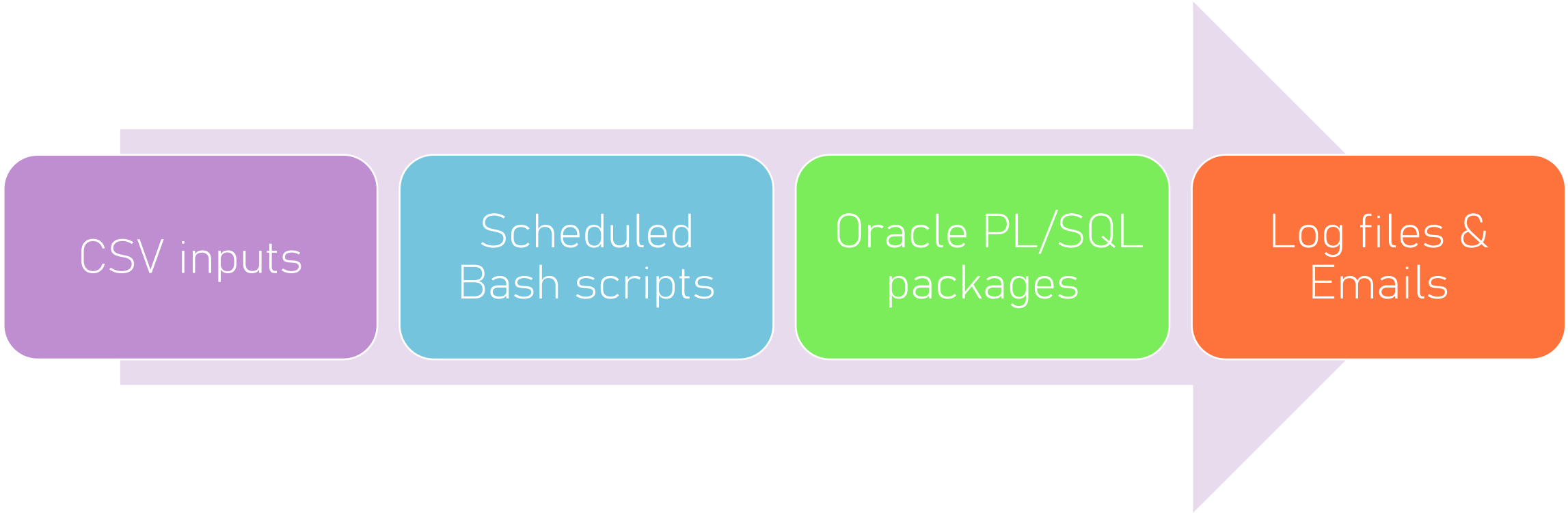
How does it work?

Unpicking code in depth

Identifying potential optimisations

Mapping logic to modern alternatives

The Legacy Stack



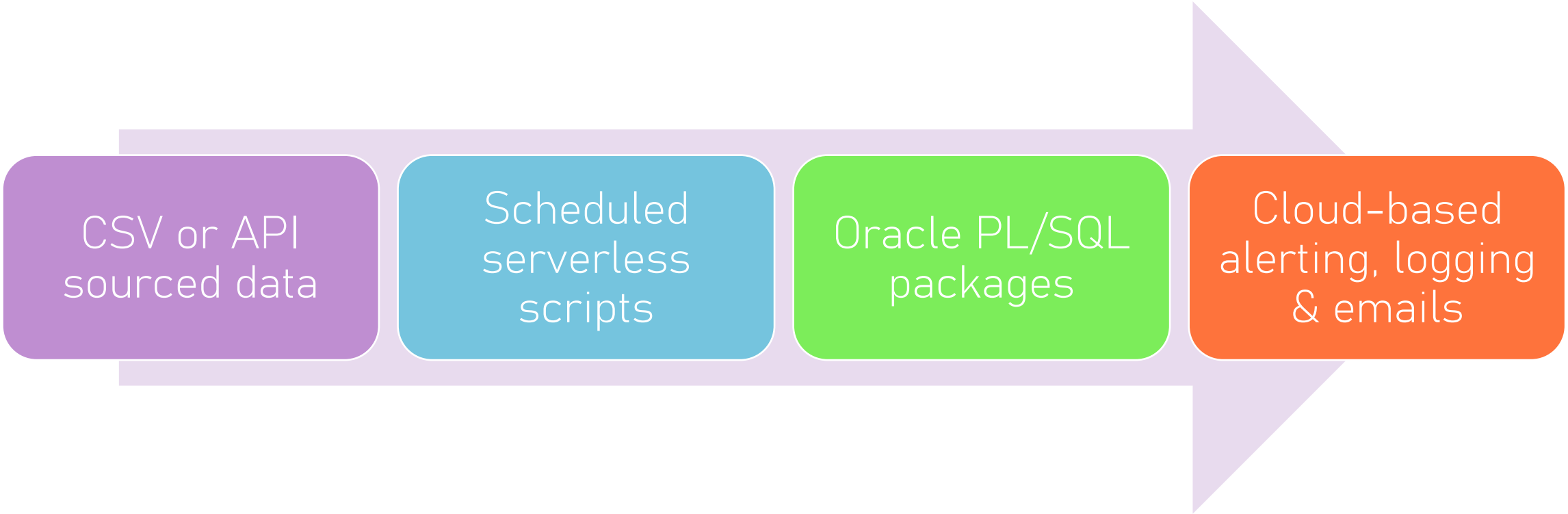
Breaking Down the Stack

- CSV inputs
  - Prone to poor formatting, ideally replace with APIs but for now, validate thoroughly
- Scheduled Bash scripts
  - Tied to single server making them unsuited to modern resilience standards
  - No validation
- Oracle PL/SQL packages
  - Reliable and complex business logic
- Log files & Emails
  - Logs are unmonitored and do not provide effective metrics or traceability

Growing the Strangler Fig

- “If it ain't broke, don't fix it“ – strangler fig architecture pattern lets new and old coexist
- Targeted replacement of outer layers with modern wrapper, retaining core logic in Oracle PL/SQL packages
- Enhanced resilience through improved monitoring of inputs & outputs and reporting
- Cloud-native façade using serverless infrastructure
- Allows piecemeal replacement of underlying code
- Begins to decouple legacy logic from legacy infrastructure

The New Stack



Selling It

- Provide stability
  - Formalised issue reporting frameworks provide certainty in uncertain times
  - Increased monitoring and alerting can help stakeholders feel empowered
- Go back to basics for the vision
  - Intended outcomes vs current implementation
- Prioritise according to need
  - Critical components failing regularly or taking up excessive maintenance time
- Be pragmatic
  - Easy wins to pain points, even if sideways moves rather than forward

Building It

- Document first, build second
- Focus on building *confidence* as much as code
- Identify a common framework and build a proof-of-concept
- Start small to keep costs and risks low
- Use tests to ensure consistency with legacy system
- Focus on like-for-like drop-in functionality as the baseline, note optimisations for later
- Keep legacy maintenance work alongside new build functionality to maintain necessary balance, allocate time on the assumption it's going to be needed (your issue log helps)

The Best-laid Schemes of Mice and Men Go Oft Awry

- Documentation is not a Single Point of Failure killer
- Resource contention is a real risk
- Stakeholder behaviours can regress, progress isn't necessarily linear
- Mixing legacy and modern in a single stack is never easy
- First attempts probably won't be the final solution
- Motivation and speed may not stay as you'd like

Key Deliverables & Milestones

- Legacy system fully documented: if Dave left tomorrow, you'd be okay
- Stakeholders put at ease about the idea of transition
- Common interim (wrapper) & target (full replacement) architectures designed
- Subsystem selected as candidate for proof-of-concept wrapper development
- Legacy version of subsystem replaced by newly developed wrapper version: change has a real foothold
- Remaining subsystems replaced with wrappers
- Subsystems rewritten from wrappers to end-to-end native code: legacy can be switched off, optimisations can be implemented

Happily Ever After?

- Migration is still ongoing, but that's okay: the monster is tamed
- Dave has been retired for over a year
- Nothing has failed spectacularly
- Main developer has now become a legacy system whisperer
- Systems will be replaced as and when they can, but it's a timeline on *our* terms, not a ticking timebomb

“I almost wish I hadn't gone down that rabbit-hole—and yet—and yet—it's rather curious, you know, this sort of life!”

— Lewis Carroll, *Alice's Adventures in Wonderland*