WHAT SHOULD I ASK FOR THIS CHRISTMAS?

GRAPHCORE

Rich Porter
Verification Futures 2022

Introductions

- Great to be back!
- Rich Porter
  - Director of Silicon Verification @ Graphcore
  - Since beginning
- Graphcore
  - ML silicon start up
  - Head office & design centre Bristol, UK
    - Beijing, China
    - Cambridge, UK
    - Oslo, Norway
- Why should you listen to me?
  - Hopefully, I'll say something interesting or controversial or both

Christmas list!?

Only 200 days to Christmas!

Don't ask – don't get

Hard vs Soft Verification

Hard Verification

- Specifications
- Simulation
- Testbenches
- Verification Planning
- Constrained Pseudo Random
- Checkers
- Functional Coverage
- UVM
- Static Formal

Nuts and bolts verification

"Conventional" verification

Soft Verification

- Process
- CI/CD
- Checklists & Certificates
- Infrastructure
- Documentation
- Team organisation & interaction
- Project planning
- Tools & practices adopted from software engineering

Engineering the best project outcome

Best practice verification at the project level

1. I want more licenses and faster tools

The way we use EDA licenses is changing

The Problem

Contemporary verification teams are changing the way they use and interact with their tooling

The problem

- Adoption of software practices like CI/CD
- Shifting left of integration
- Greater integration of commodity hardware – more CPU cores available

What is Continuous Integration?

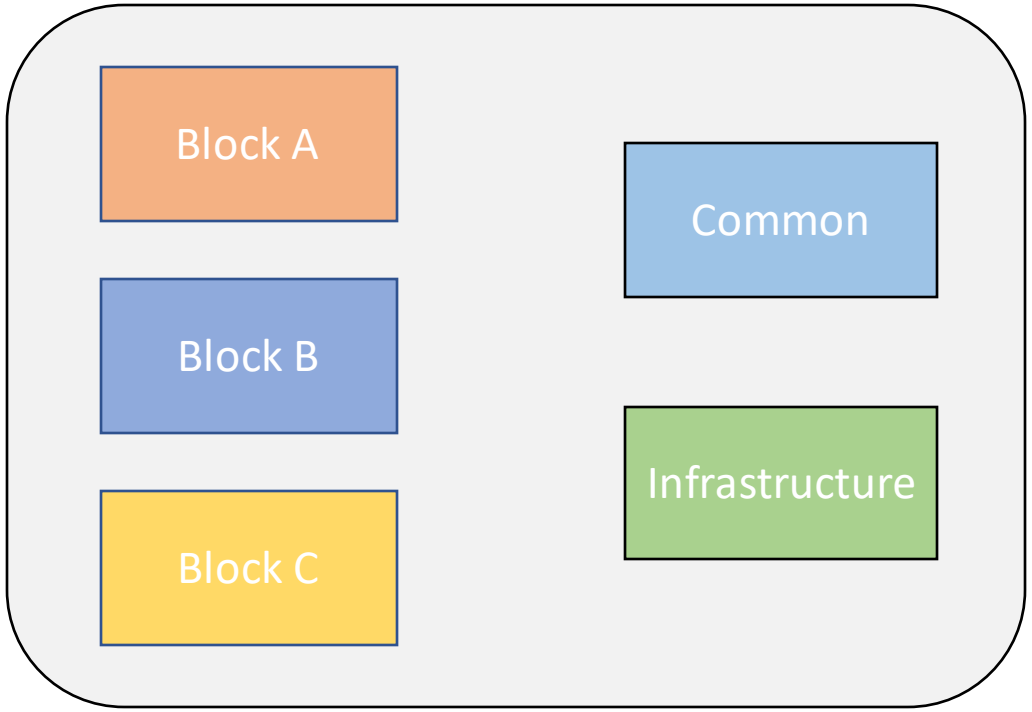
- Stolen from software/Dev ops
- Encourage all contributors to make 'small' changes
  - Automate a suite of non-regression tests
  - Potentially gate merging on successful testing
- Continually improving quality of the code in the repository
  - Adding tests as functionality is added
- Requires 'always working' repository model
- Run testset on changes, hourly, nightly, weekly

Mono repo versus repo-of-repos

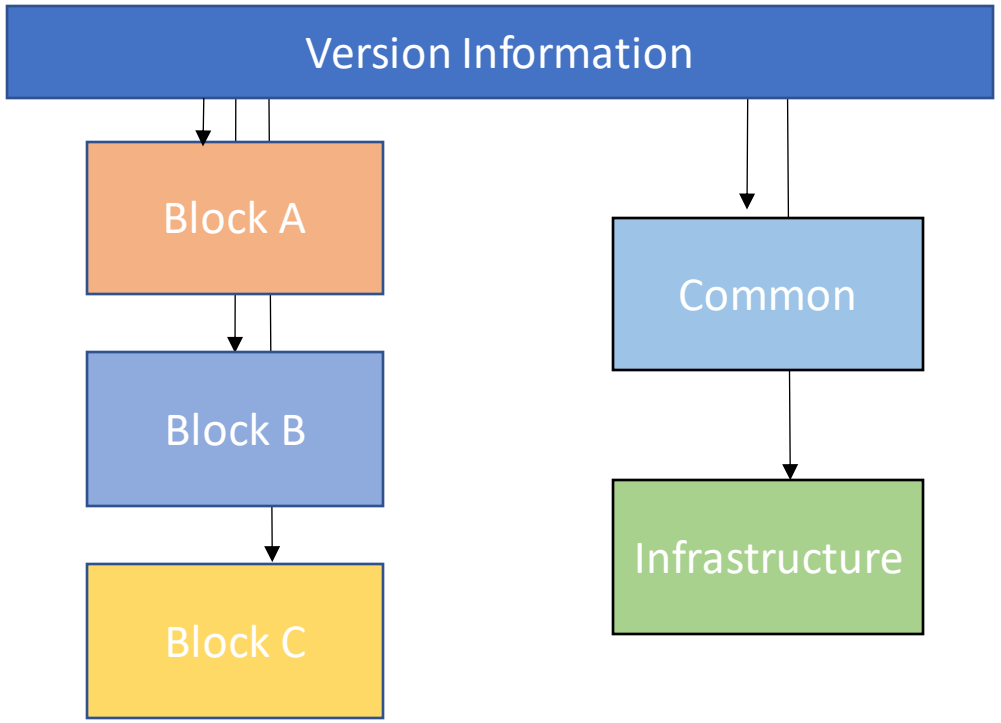
- Repo = source code repository, e.g. git
- Mono-repo contains all projects in a single repository
  - For a chip or chips
  - For a chipset
- Repo-of-repos has separate repositories for each block
  - Finer granularity
  - Another repository to contain versioning information
    - What versions of the other repos combine to make a chip

Mono repo vs repo-of-repos

Mono repo



Repo of repos



Mono repo vs repo-of-repos

Mono repo

+ Shifting left integration. Coupled with CI can easily check that everything integrates at the chip level at every version

- Increased CI workload

- Engineers need to know about everything

Repo of repos

+ Individuals can work very efficiently in the smaller repos, smaller CI at block level

- Keeping the chip level working a is much harder

- Engineers siloed

Experience

- Graphcore thinks the advantages warrant usage of mono repo
- Mono repo with CI generates high compute burden
  - Mostly non novel work
  - Difficult to compute test subset based on changes
- Quality of repository head well known
- Shift left of integration
- Keep previous generations of designs alive
  - With updated infrastructure

Workload is changing

- Expansion of CI beyond functional verification
- Not just simulation
  - Static formal
  - Synthesis
  - Logical equivalence
  - CDC/RDC
  - DFT
  - Lint
  - Tests of test generation
  - Tests of test infrastructure
- I need more licenses for everything
  - Non novel work, just checking it still works

Trends in commodity processors

- End of Moore's law and Denard scaling
  - Can no longer upgrade to get more performance
- Increasing core counts
  - Have more cores by default
- MCM, CoWoS, CoW, WoW
- Long lead times, increasing cost
  - For little per thread performance boost
- Cattle versus pets
  - Don't retire them, hammer them until they die
- RISC servers vs x86 servers
  - Higher integration, lower power per unit of simulation

Optimizations

- I need lowest latency possible for regression run
- I can optimize jobs for speed or power
  - Target quick jobs to older or lower power
  - Target 'long pole' jobs to fastest
    - Optionally using thread level parallelism
- Should I also have to optimize for licenses?
  - If I just had more licenses I wouldn't need to
- Cost of energy is increasing
  - When will the CFO turn off my compute farm?

Fast simulation

- I still want single threaded simulation to be as fast as possible
- Get to the point of failure as quickly as possible when debugging

Summary

- CI workload requires
  - Lots of compute
  - Corresponding license counts
  - Most jobs are non novel
- Using lower power processors saves energy but uses more licenses
- I don't want to have to optimize for license usage as well as power and latency
- When should I dispose of my old compute?

2. I want a common logging API

Preserving log message semantics

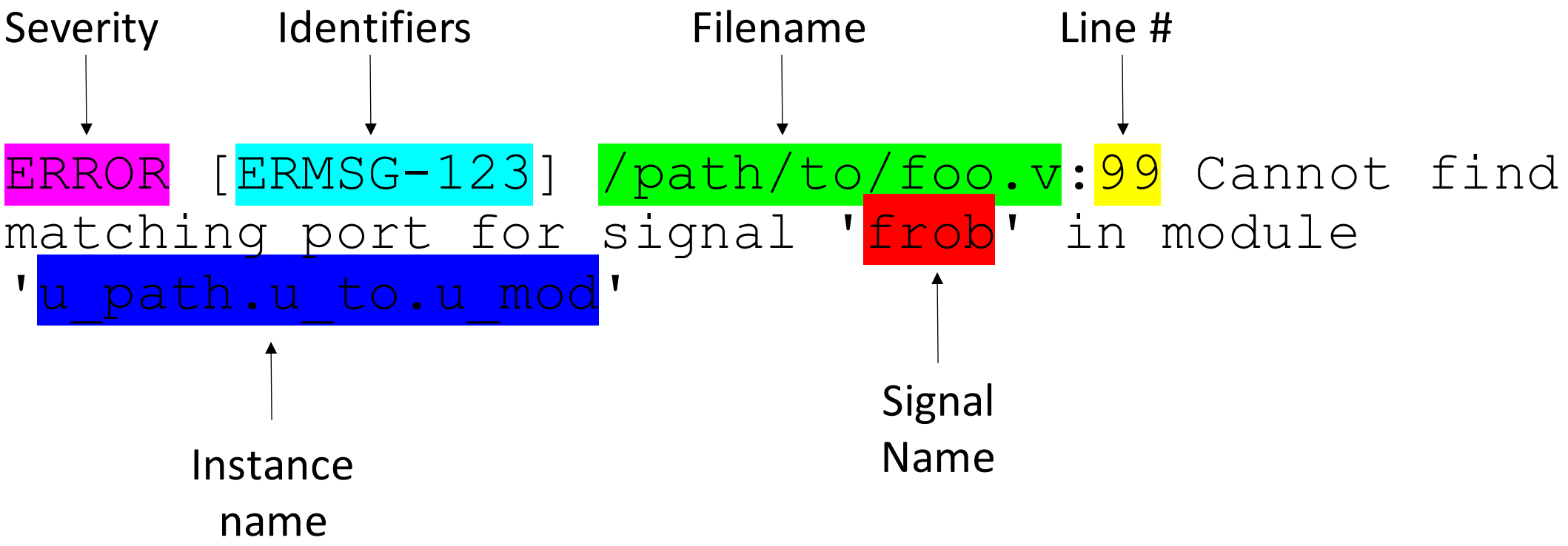
The Problem

EDA tools flatten message semantics when writing text log files

Example

```
ERROR [ERMSG-123] /path/to/foo.v:99 Cannot find
matching port for signal 'frob' in module
'u_path.u_to.u_mod'
```

Example


Severity
Identifiers
Filename
Line #
ERROR [ERMSG-123] /path/to/foo.v:99 Cannot find matching port for signal 'frob' in module 'u_path.u_to.u_mod'
Instance name
Signal Name

Hang on …

- But now I've got to use some error prone method (probably involving regexps) to get the interesting data back out
  - Inconsistent use of formatting
  - Variable line breaks
  - Can't `grep -i error` in designs with entities that have **`error`** in their names
  - Excessively chatty messages use lots of processor time to parse, potentially slowing tool execution
  - I wish I'd not used regexps and done it with a parser
- Also messages generated by my infrastructure are mixed in with tool output
  - And not in the right order!

Arrrrgggghhh!!!!!

- Infuriatingly the tool had **all** this information and flattened it *into text* for a human to read!
- But I'm processing the information with a machine!
- And I have *millions* of logs!

WHY?!

What do I want?

Avoid missing real errors

Over eager signoff regexps

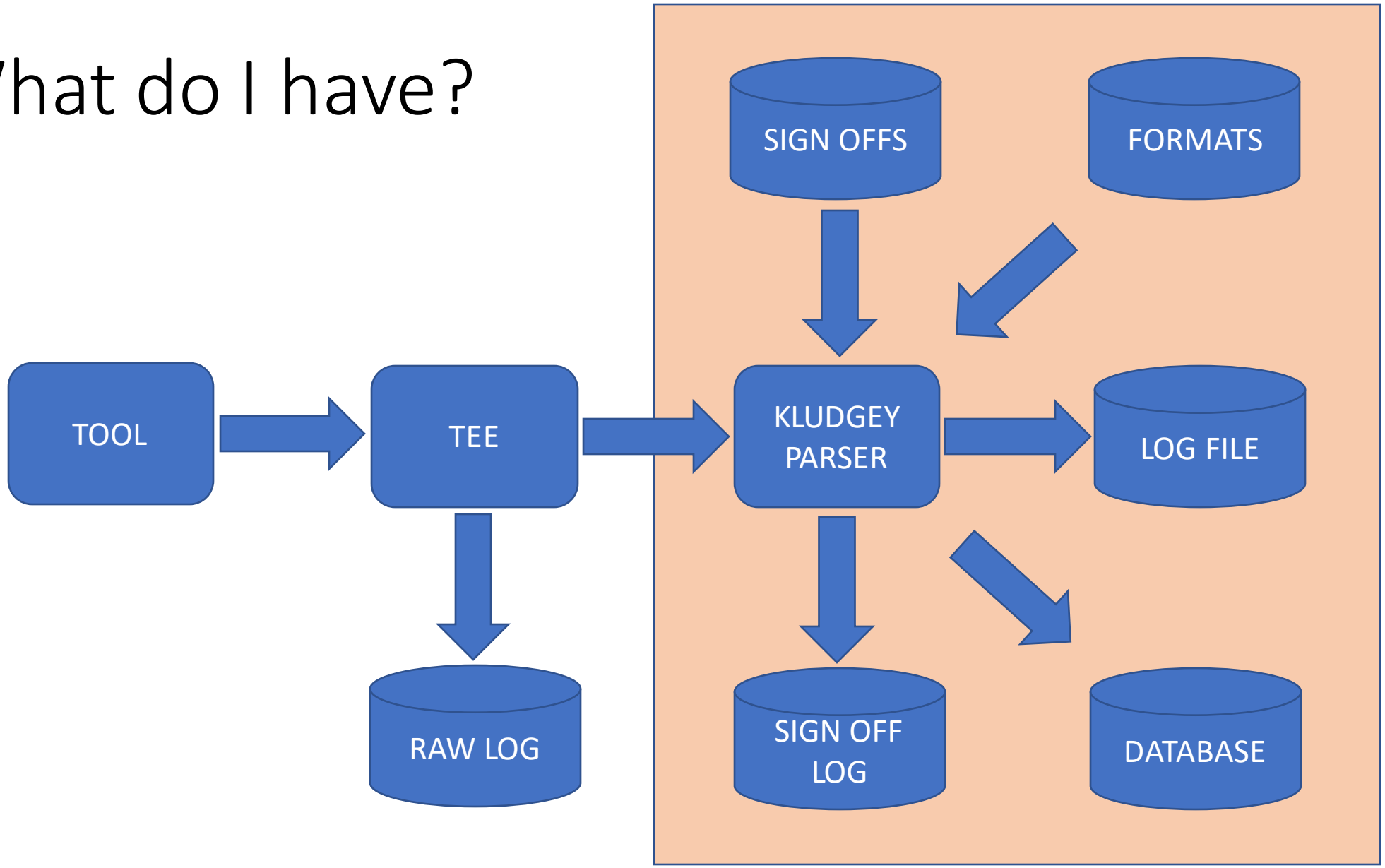
Make signoffs more targeted

And easier to describe and review and test

Automated triage

Group by severity, identifier, filename, ...

What do I have?



What would I like?

An API to interact with EDA tool messages

A callback when a message is generated

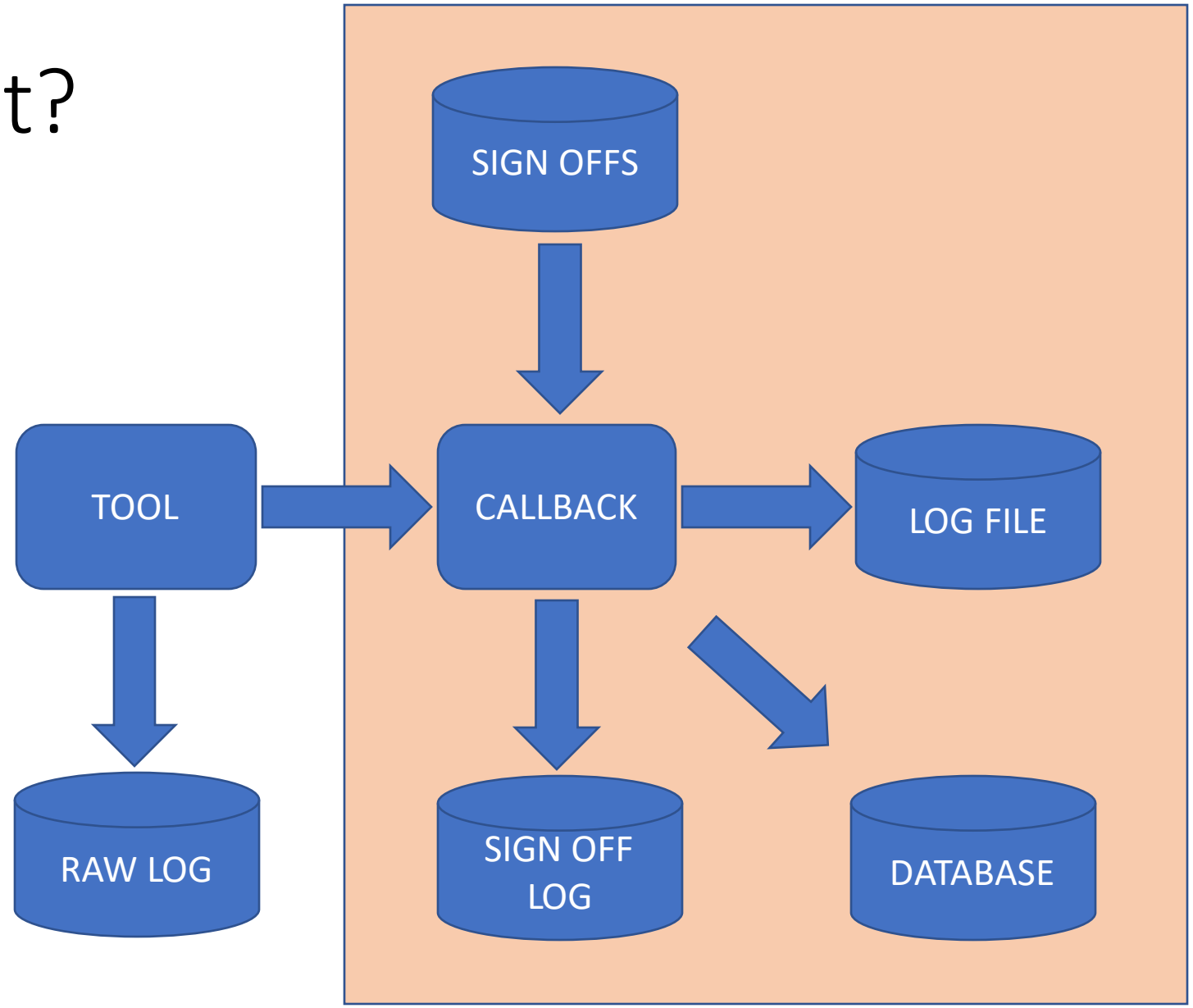
Access to message attributes

No bug-ridden reverse engineering of **sprintf** required

Active signoff management

Terminate tool execution

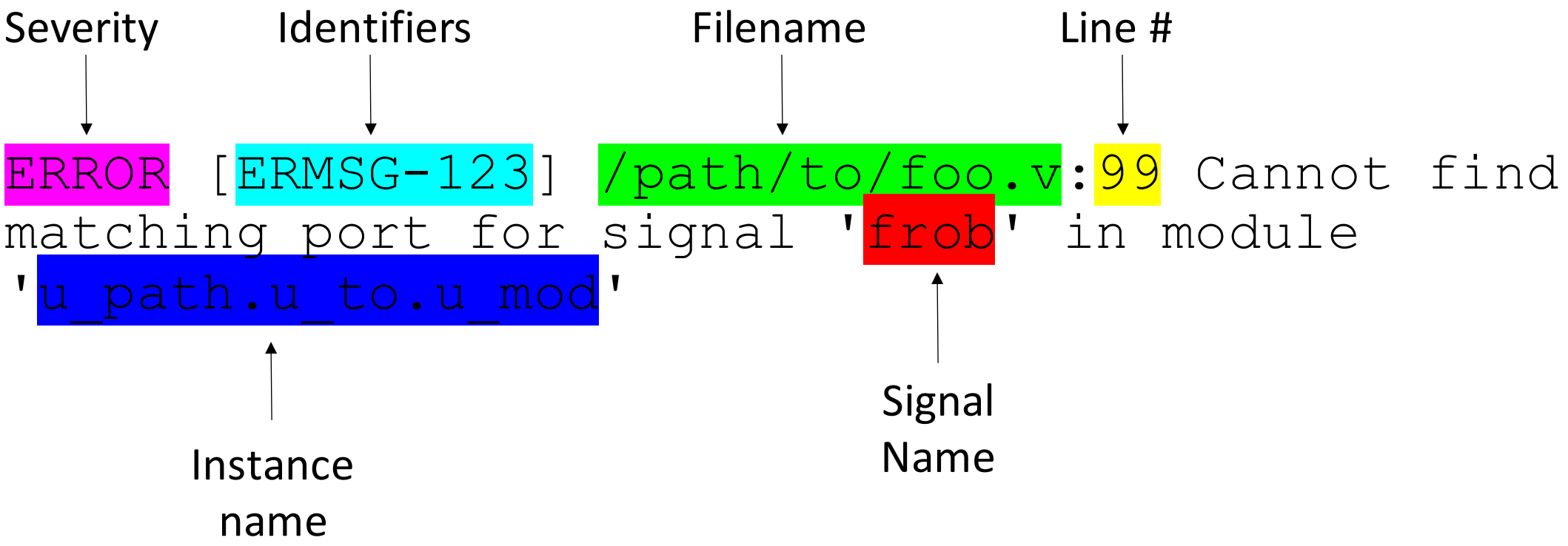
What do I get?


SIGN OFFS
TOOL
CALLBACK
LOG FILE
RAW LOG
SIGN OFF LOG
DATABASE

Choices

- C API
- JSON
- XML
- …

Example


Severity
Identifiers
Filename
Line #
ERROR [ERMSG-123] /path/to/foo.v:99 Cannot find matching port for signal 'frob' in module 'u_path.u_to.u_mod'
Instance name
Signal Name

JSON example

```
{
  "message" : "ERROR [ERMSG-123] /path/to/foo.v:99 Cannot find matching port for signal
'frob' in module 'u_path.u_to.u_mod'",
  "timestamp" : 1653512270,
  "severity" : "ERROR",
  "identifiers" : [ "ERMSG", 123 ],
  "file" : {
    "name" : "/path/to/foo.v",
    "line" : 99
  }
  "attrs" : {
    "instance name" : "frob",
    "signal name" : "u_path.u_to.u_mod"
  }
}
```

The next Problem

Why do I still have to use TCL to program EDA tools?

The Problem more...

- I still have to use TCL to program EDA tools
- But my infrastructure is written in another language
- I cannot re-use code from my infrastructure in the EDA tool
- TCL is not well suited to writing my infrastructure
  - Yes, I know it was developed by Ousterhout specifically for IC design
  - Newer, faster, more fashionable alternatives

- Will TCL last forever?

A solution?

- Why does the tool interface have to be TCL?
- I could ask for another language OR
- I could ask for a C API
  - Because C linkage ***is*** going to last forever?
- Better still a tool could have a C API ***and*** distributions of select scripting languages using this API
  - I could build my own or use a prebuilt version

There is a precedent

- Verilog has the VPI
- It is possible to emulate much of the TCL interface of a simulator through the VPI
- Graphcore does this with Python
  - This is not unique and has been done many times before

I would like …

- To be able to fit my preferred application/tool integration language to ***any*** EDA tool
  - To promote code reuse
- To have a C API to the tool instead of a TCL interpreter
  - Shims for popular languages a bonus
  - Include TCL for backwards compatibility

Our hack

- Connecting a Python interpreter via TCL C API
- Allows passing of TCL commands and objects to and from Python
- Proved reuse of our Python infrastructure with a static formal tool

- Inelegant … but it works

Summary

- Tools serialize structured data to text
- Preserving the data semantics would
  - Make interpreting messages less error prone
  - Simplify signoffs
  - Use less processing power
  - Improve accuracy of automated triage
- Presenting a tool API with C linkage would allow me to use any scripting language I wanted

3. I want more alternatives to UVM and SV functional coverage

Or at least non HDL based frameworks

The Problem

UVM

VS

The *real* Problem

System Verilog is not a general purpose programming language

System Verilog & Frameworks

- System Verilog is a parallel DSL
- SV has always felt like several languages cobbled together
  - Verilog 2005 at the core
  - Class based SV
  - SVA & temporal expressions
  - Functional coverage
- UVM is a framework built on top
- There are many other frameworks built on top of VHDL, System Verilog
  - There already alternatives to UVM
  - None of them as widely used as UVM

Verification frameworks in other languages

- Advocating creating verification frameworks with general purpose languages
  - Facilitating mixing verification & general purpose code
  - Library support of other languages
  - Leveraging performance
  - Larger online community support
- Keep simulation core to logic simulation
  - And BFMs
- Pseudo random test generation with functional coverage is best practice architecture
  - Unwise to change this

Example

- Work being done with pyuvm on top of cocotb
  - IEEE 1800.2 based
- Doesn't need to be UVM based
- Could be more than one framework
  - Targeting different types of verification, CPU vs ASIC
  - Better support for offline test generation

Functional coverage

- Separate coverage library
- Allow introspection of coverage
- Apply functional coverage to other parts of verification
- Coverage analysis tools

Summary

- I am not calling for the abolition of UVM
- Non HDL based verification frameworks will enable mixing of verification and other general purpose code
- Removing functional coverage from the simulator will widen its use into non simulation environments

Key Takeaways

- I need more EDA licenses to run CI on modern computers
- I need a better way of determining if a test/job has passed or why it has failed
- I need more choice in verification frameworks

Thanks Santa!

THANK YOU

Rich Porter
Richard.Porter@graphcore.ai