Developing OS-agnostic visualization tools
using System Viewer and LTTng as an example

Andrew McDermott

Wind River, UK

29-30th January 2008

WIND RIVER

Overview

- What is System Viewer?
- The challenges in extending the tool to LTTng
- Some solutions to those challenges
- Conclusions
- Questions?

- Demo

What is System Viewer?

- Runtime instrumentation
  - Static compile-time instrumentation for core OS features/facilities
    - semaphores, message queues, signals, tasks, timers, user events, …

- Runtime Configuration/Data Collection
  - Configure: buffer sizes, buffer mode, timestamp mode, …
  - Collection: socket based, file system based (NFS)

- Runtime Triggering
  - Allows programmatic capture of events

- Visualization tool
  - Displays **events** over time
  - Displays the interaction between tasks, interrupts and system objects

Types of problems it helps to solve?

- Deadlocks
- Race conditions
- Problems related to reentrancy
- Problems related with priority settings
  - And priority inversion
- Timing problems
  - Why is my task not running?
- Load and CPU/Core utilization
- System exploration
  - Using user defined events

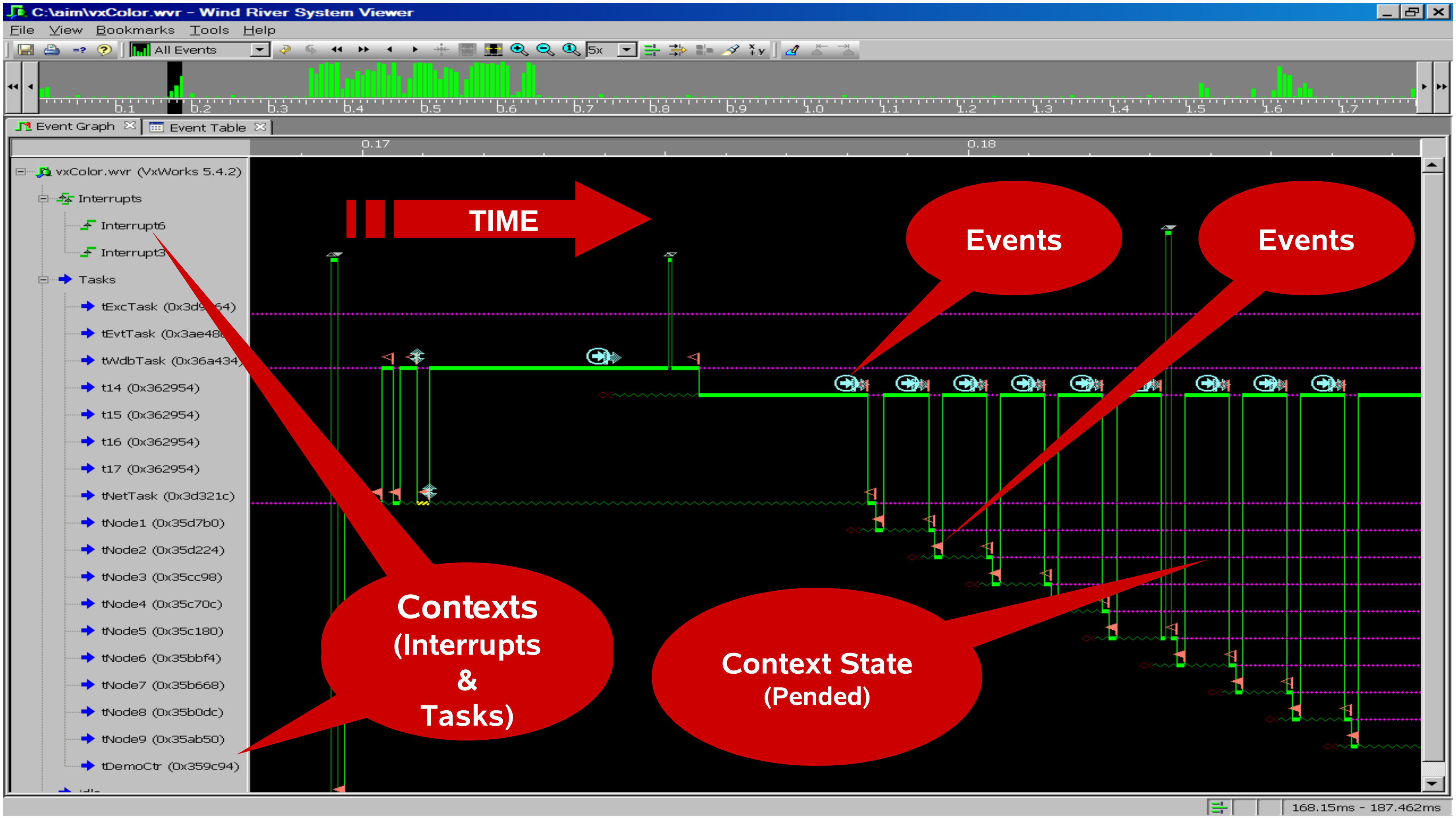
WIND RIVER

System Viewer Workflow (VxWorks)

- **Configure the system with instrumentation support**
  - **By default just context switch is instrumented**
  - **Optionally select "libraries" of additional instrumentation:**
    - VxWorks:
      - Tasks, Semaphores, Message Queues, Watchdogs, …
- **Configure the target**
  - **Buffer size**
  - **Buffer wrapping mode**
- **Start logging, stop logging and upload the log**
  - **The uploaded log is called the EVENT LOG**
  - **Historically this is has a ".WVR" file extension**

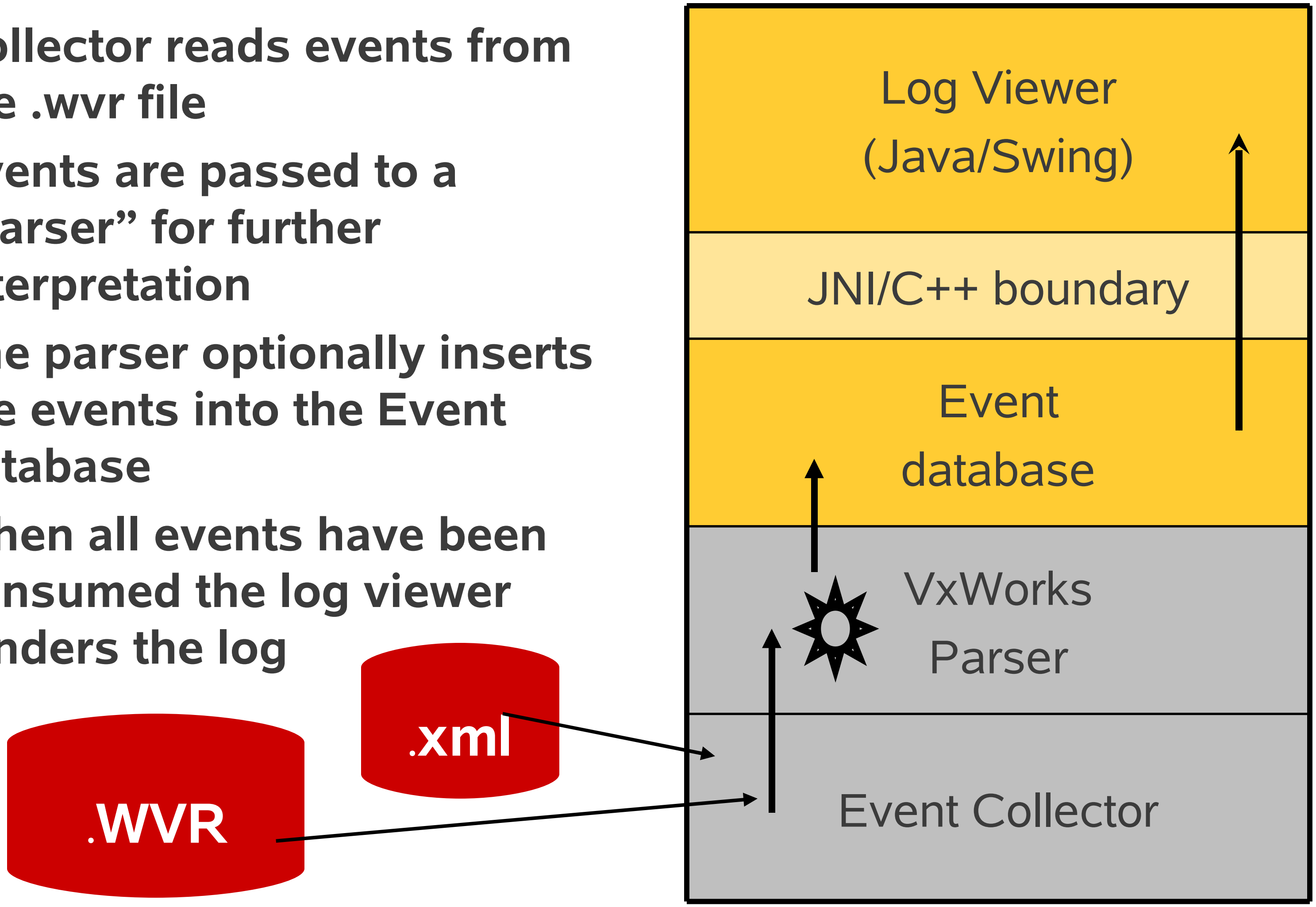
- **The visualization tools present a view of the .WVR file**

System Viewer's Log Viewer



Architectural View

- Collector reads events from the .wvr file
- Events are passed to a “parser” for further interpretation
- The parser optionally inserts the events into the Event database
- When all events have been consumed the log viewer renders the log



WIND RIVER

What is in the Event database?

- **It is a model that maintains a set of entities and relationships:**
  - **CONTEXTS**
    - thread, task, interrupt, process, etc
  - **EVENTS**
    - These are "things" that happened and are attributed to a context (e.g., semTake, intEnter, userEvent, etc)
  - **STATES**
    - These represent the state of the "context" when the event occurred (e.g., Running, Pended, Interruptible, etc)
  - **PARAMETERS**
    - Attributes of an event. e.g., recurseCount, fd, address, PC, …
  - **(There are 15 entities in total)**
- **The model strives to be OS-agnostic**
  - We have had success rendering VxWorks, Linux, and … event logs
- **Often referred to as the "eventbase"**

The scope of the Event database

- System Viewer's Log Viewer and/or the Eventbase is NOT a general purpose viewer

- The primary focus and scope of the Eventbase schema is to enable and support time-based trace systems

- It is a bespoke tool

- But If you conform to the Eventbase model you'll get visualization for "free"

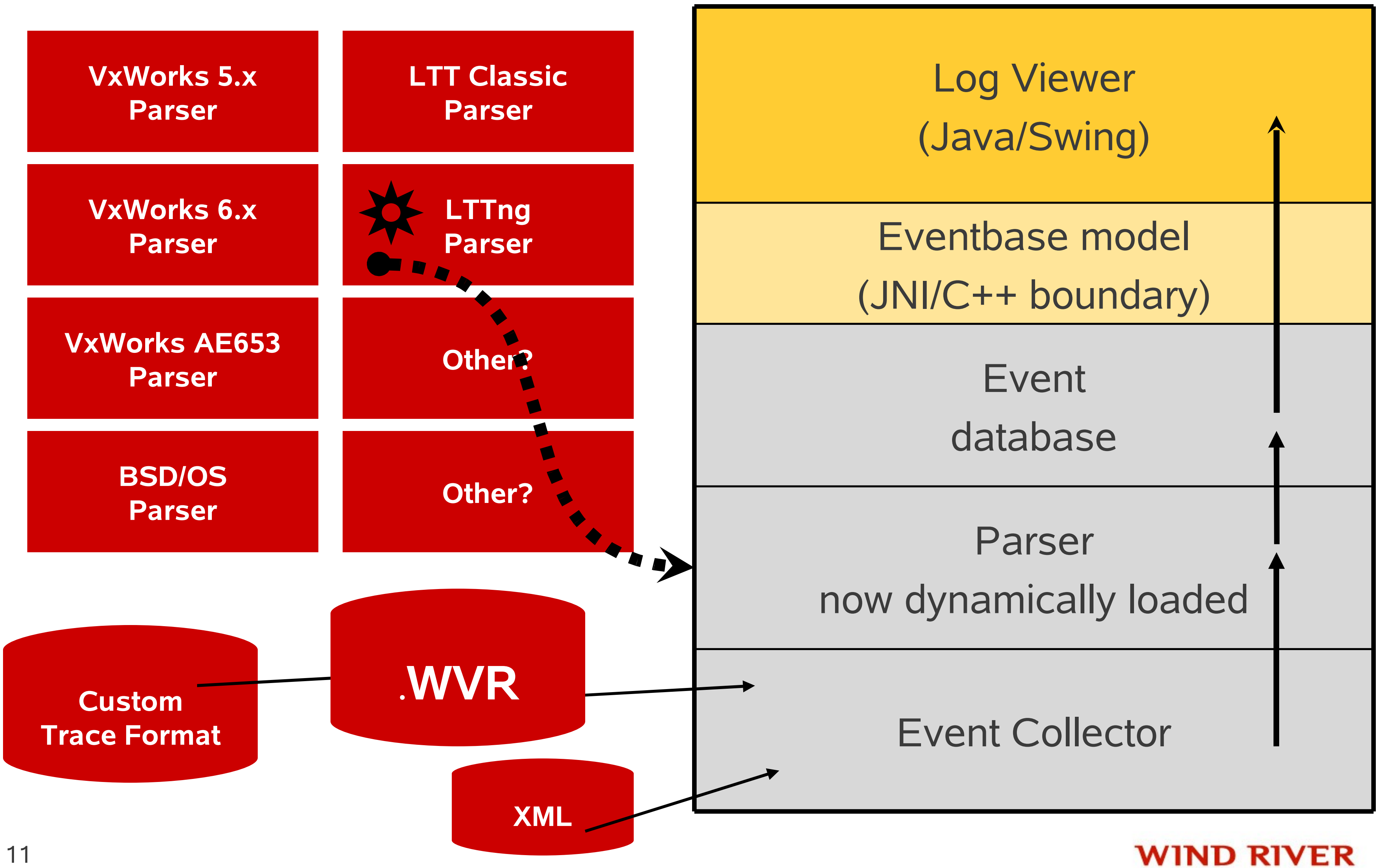
How do we extend SV to LTTng?

- **So we have this great visualization tool but how do we extend it to other trace formats?**

- **It turns out that it is really EASY!**
  - **We convert the LTT trace format to .WVR format**
  - **We write a new parser which inserts events, states, etc into the event database**
    - There is some additional XML files & icons to be provided
  - **We get visualization for free!**

- **What about other trace systems?**
  - **Using this model we have successfully done:**
    - VxWorks 5.x, VxWorks 6.x, VxWorks AE653, BSD/OS, LTT classic, C++

Supporting other trace formats



Additional Challenges supporting LTTng

- **Specific LTTng challenges:**
  - **LTTng has its own means of configuration and data collection**
    - We have modified the lttctl and lttd programs
  - **LTTng only targets and compiles on Linux**
    - Note: The log decoding API (libltt.so) only compiles on Linux
  - **LTTng changes rapidly**
    - We don't or can't change the tool for every (minor) release
    - We don't want to chase the bleeding edge either
  - **LTTng can generate huge data sets**
    - On a modern x86 desktop it is possible to collect gigabytes of data very quickly

Architectural Growing Pains

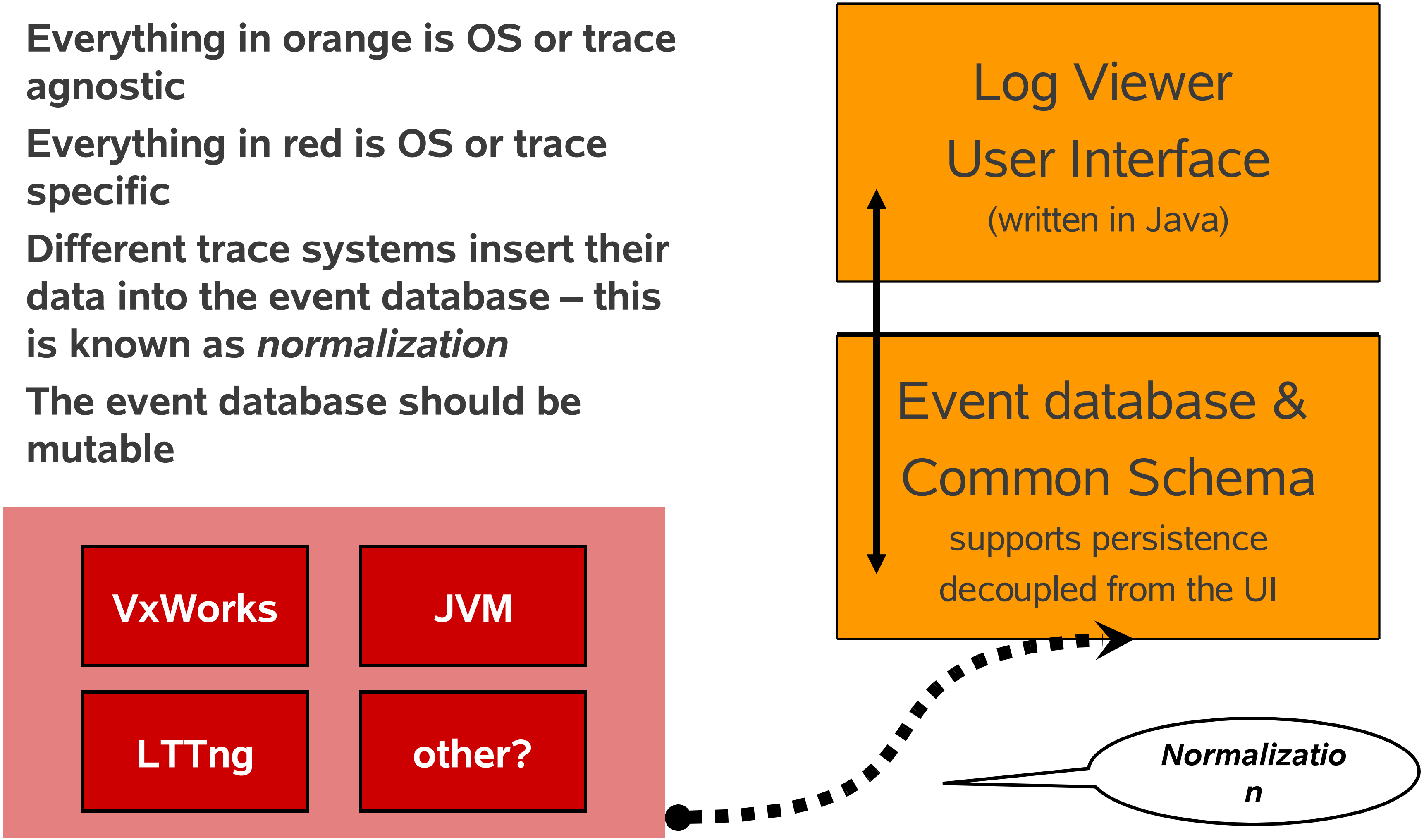
- Using the .WVR format works but there are limitations to the current model and the underlying architecture
- Monolithic application
  - The UI and the data is one large program
- The event database is not a persistence model
  - Everything is in memory
- It is not scalable to large data sets
  - LTT logs get bigger much quicker than VxWorks logs
- No clear separation between the UI and the data model
  - There is a restricted Java API for programmatic access to the .WVR
- The event database is not mutable once data has been entered
- The .WVR file format is beginning to show its vintage
  - Timestamps are 32-bit only, 64-bit values were strings…

Architectural Goals (1/2)

- We want to visualize new operating or tracing systems
  - But we don't want to rewrite it for each new operating systems
- We want to change as little as possible so that:
  - Faster time to market for new systems
  - The core product becomes extremely stable over time
  - There is consistency for end users
  - We don't have to retest the UI layers over and over
- We need to handle large data sets
  - SMP systems are larger still
- We need to have a persistent and common data format
- We want the event database to be mutable
- We need a language-independent means of making ad-hoc queries
- We want 3rd parties and other internal groups to be able to build on the work we have done

Architectural Goals (2/2)

- Everything in orange is OS or trace agnostic
- Everything in red is OS or trace specific
- Different trace systems insert their data into the event database – this is known as *normalization*
- The event database should be mutable



The New Relational Event Database

- **The Eventbase and Log Viewer has since been reworked to use SQLite as its database engine**
  - **The SQLite website has a long list of features but these are the most important to us**

  - Self-contained: no external dependencies
  - Sources are in the public domain. Use for any purpose.
  - In process – it is not a client/server database
  - Zero-configuration - no setup or administration needed
  - Faster than popular client/server database engines for most common operations
  - A complete database is stored in a single disk file
  - Database files can be freely shared between machines with different byte orders.

Eventbase (SQLite) Performance (1/2)

- **SQLite is “fast”, but how “fast” ?**
  - **There are a number of performance metrics to consider**
    - INSERT performance
    - QUERY performance
    - INDEX generation
- **INSERT performance**
  - **To convert a 8.5MB VxWorks .wvr file takes ~44s**
  - **The converted database has ~4.5 million rows**
  - **Which is ~100,000 rows per second**
  - **To get these numbers we modified SQLite**
    - To not use the journal file
    - To increase the default page size
      - Without these changes the conversion takes ~60s.
- **QUERY performance**
  - **We found query performance generally excellent and on a par with our home grown database**

WIND RIVER

Eventbase (SQLite) Performance (2/2)

- INDEX performance is not great
  - To add indices to some of the tables, notably the events, states and parameters can double log conversion time
  - This appears to be the general case; if it takes 100s to convert without using indices, it takes another 50s to create the indices
  - This is much complained about on the SQLite mailing list

- To mitigate the INDEX creation time we reworked the Log Viewer and the schema to not require indices
  - Today all queries run without the need for indices
  - It's possible to add indices to a converted database at a later date

The advantages of SQLite

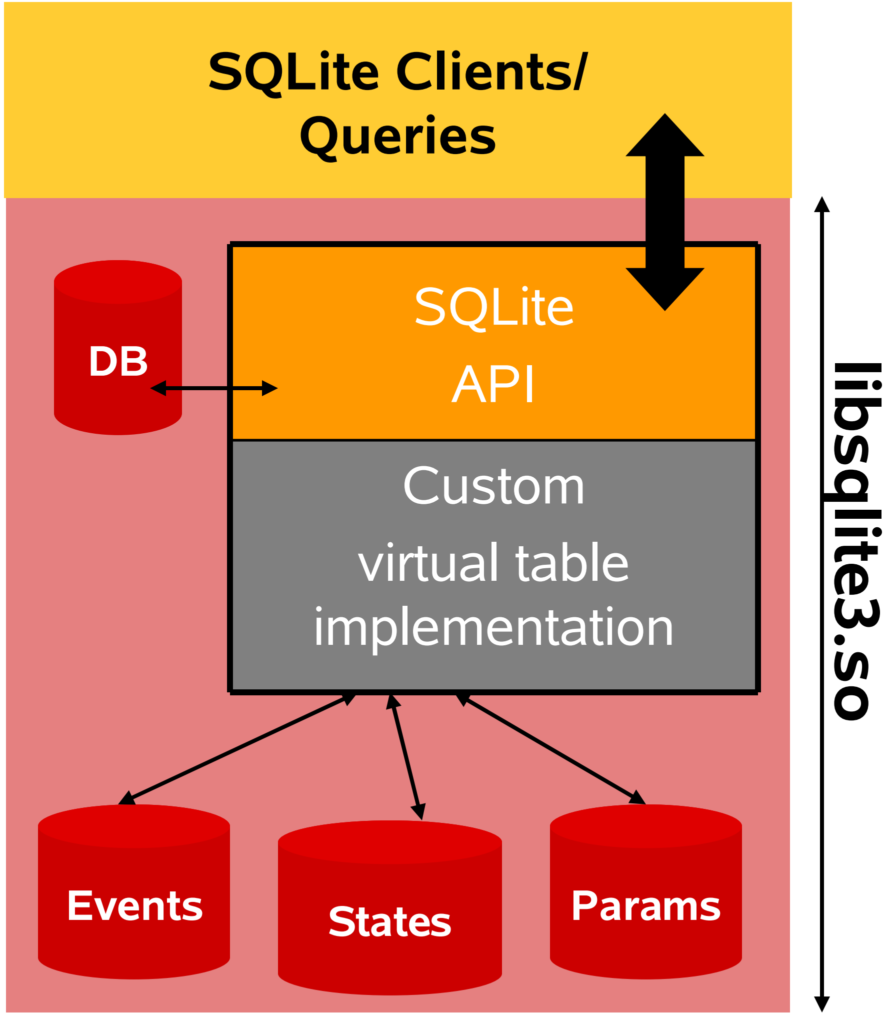
- It provides a clear separation between presentation and data
- We've stopped writing our own database engine
- We no longer have to provide a programmable API to access the eventbase
  - SQLite has bindings for many languages:
    - Perl, Python, PHP, C, C++, Java, Lua, Lisp
- There are other UI orientated tools for managing/using a SQLite database over and above the sqlite command line interface
- It is both extensible and very malleable
  - By using SQL statements existing data can be added or removed
- Customers can write their own queries, run their own data-mining operations all without having to wait for Wind River to support such features
- SQL is a generalized and well understood language
- Can be used to prototype new analyses outside of the analysis developmental team

Where SQLite doesn't scale

- **The size of a SQLite database is cause for concern**
  - **Disk may be cheap, but sending a large database to support@windriver.com has a cost**
  - **Converting a 8.5MB VxWorks log produces a 115MB SQLite database**
  - **We have not looked at optimizing the file size**
    - Our customers still send us .WVR files
- **Creating INDICES take too long**
  - **BUT appropriate indices make QUERIES run extremely quickly!**
- **Concurrency**
  - **Thought this is not a concern for the Log Viewer**
- **INSERT performance needs to be much quicker**
  - **Converting *large* event logs still takes way too long**

SQLite's Virtual Tables aid performance

- The virtual table mechanism allows us to extend SQLite without changing the client
- From the perspective of a SQL statement, the virtual table object looks like any other table.
  - Behind the scenes, queries to a virtual table invoke custom methods instead of reading and writing to the database file.
- We now populate the database by writing directly to the file system, bypassing SQLite
  - INSERT speed is now bounded by file system write



WIND RIVER

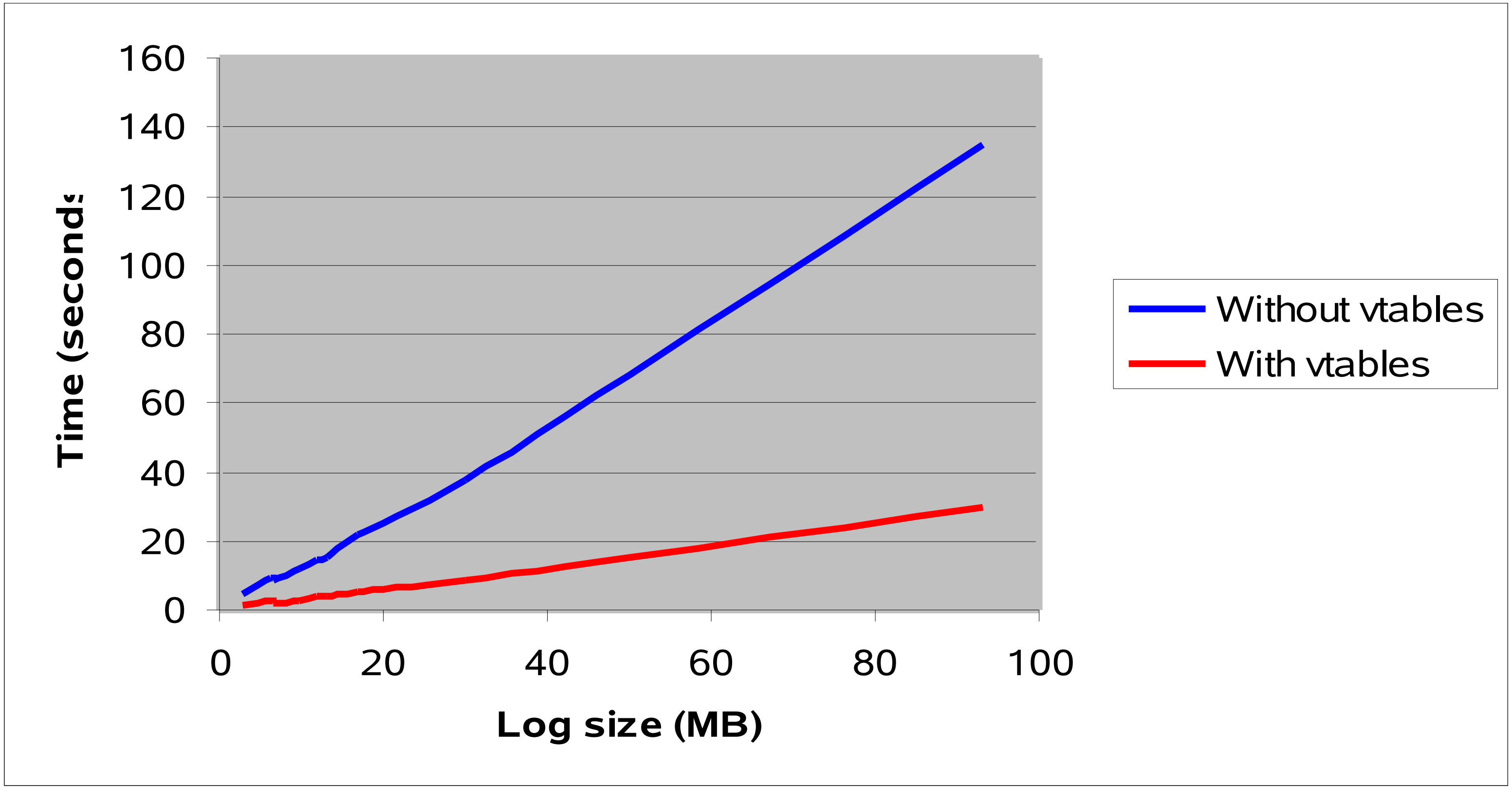
Virtual Table Performance (1/2)

- **Log conversion times are much improved**

Log Size (MB)	Without vtables (seconds)	With vtables (seconds)	Quicker
2.7	4.60	1.28	72%
6.1	9.45	2.49	74%
6.7	8.68	2.21	75%
9.7	12.20	2.91	76%
12	14.66	3.96	73%
13	15.44	3.83	75%
17	21.77	5.35	75%
30	37.91	8.62	77%
93	134.60	29.77	78%

Virtual Table Performance (2/2)

- How does it scale?
  - The trend looks good!



WIND RIVER

Conclusions

- **The key is splitting data from the presentation layers**
  - **We convert from an arbitrary data format to a common format**
  - **The UI then need only understand one model**
  - **The UI makes little or no interpretation of the data**
    - It is a thin client
  - **The split now allows us to port the Log Viewer to Eclipse**

- **By using SQLite**
  - **We now have persistence which aids scalability**
  - **We now have a standardized and commoditized data engine**
  - **SQL is well understood**
  - **Access to the data is either via SQL or a language binding**
  - **Virtual tables makes SQLite viable for large data sets**

- **Extensible by 3rd parties**
  - **We want 3rd parties or customers to use the data in a way we never imagined or conceived**

Future

- We are extending the database approach to other Wind River analysis tools
  - We have already done Memscope
- The future is exploiting the database to provide “New Analysis” views – this is where the real value now lies
  - CPU utilization
  - System Load
  - Memory Usage
  - Better search capabilities
- Intangibles
  - Ease of development
  - Regression testing

Questions?

Log Viewer Demo