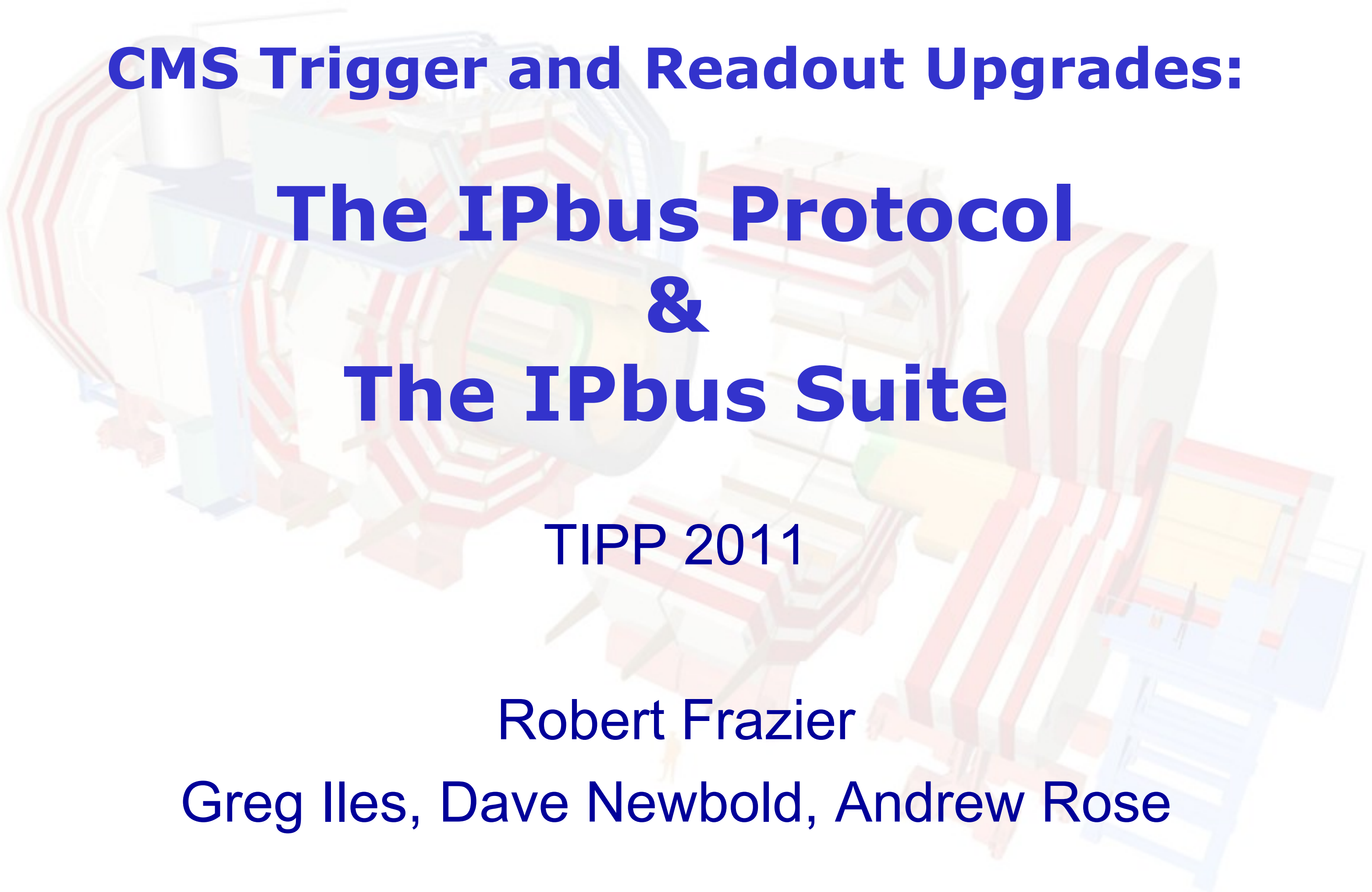


CMS Trigger and Readout Upgrades:

The IPbus Protocol
&
The IPbus Suite

TIPP 2011

Robert Frazier

Greg Iles, Dave Newbold, Andrew Rose

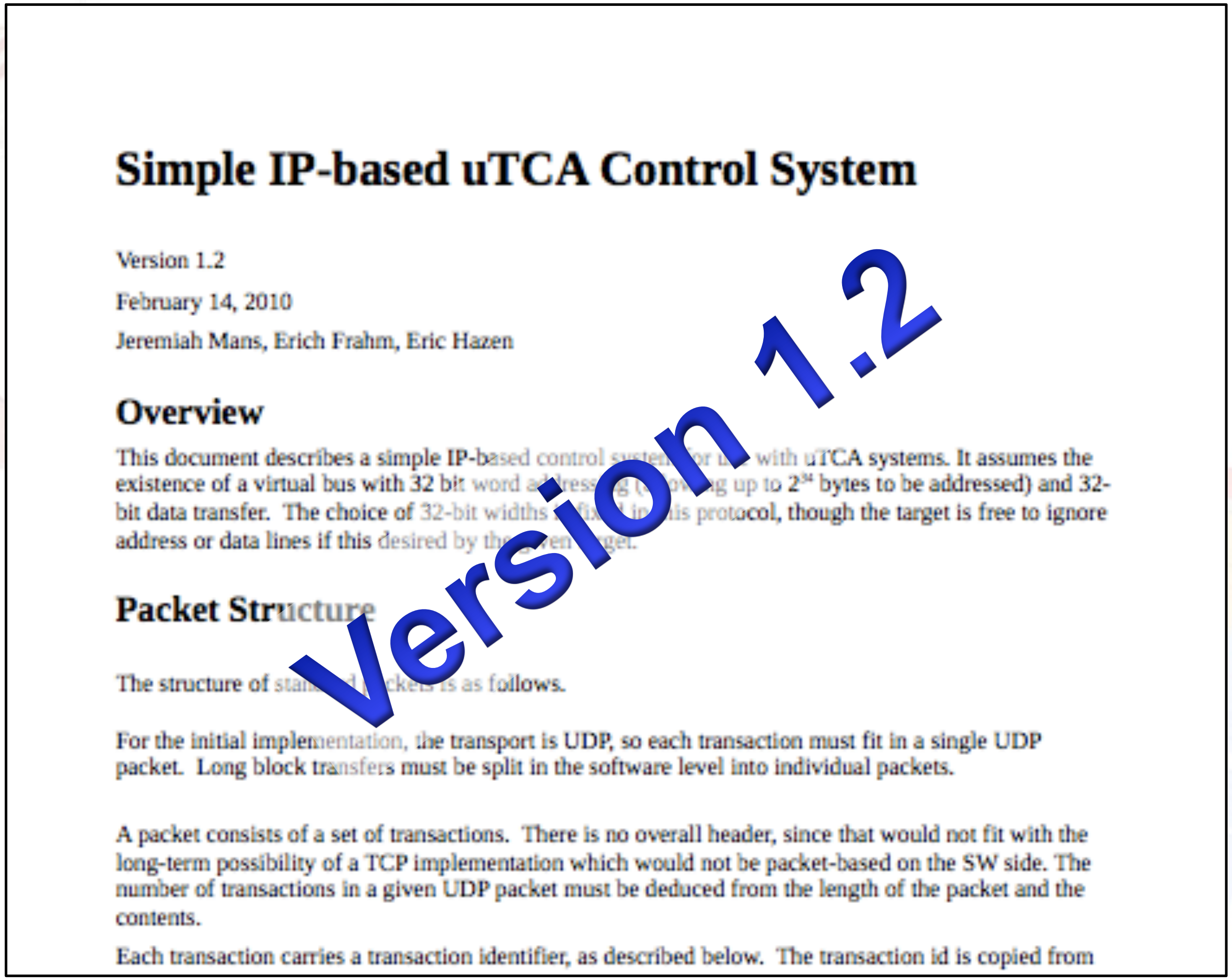
Summary

- What is the IPbus Protocol?
  - And what do we need from it?
- Introduction to the IPbus suite
  - Firmware
  - Software: Redwood, Control Hub, PyChips
- IPbus Testing
  - Reliability
  - Throughput
  - Scaling
- What’s Next?
- Conclusions

What is IPbus?

- IPbus is a simple, IP-based control protocol
  - Originally created by Jeremy Mans, et al in 2009/2010
  - Now **all** s/w and f/w development is being done by a UK collaboration
    - University of Bristol and Imperial College London
- Designed for controlling future CMS trigger and readout h/w
  - Control “standard” for µTCA or TCA-based hardware over Gigabit Ethernet
- Protocol describes basic transactions needed to control h/w
  - Read/write, non-incrementing read/write, etc, etc.
- UDP is the recommended transport implementation
  - Easiest to implement in firmware
  - Uses relatively few FPGA resources

What is IPbus?



Simple IP-based uTCA Control System

Version 1.2

February 14, 2010

Jeremiah Mans, Erich Frahm, Eric Hazen

Overview

This document describes a simple IP-based control system for use with uTCA systems. It assumes the existence of a virtual bus with 32 bit word addressing (allowing up to 2^{34} bytes to be addressed) and 32-bit data transfer. The choice of 32-bit widths is fixed in this protocol, though the target is free to ignore address or data lines if this desired by the given target.

Packet Structure

The structure of standard packets is as follows.

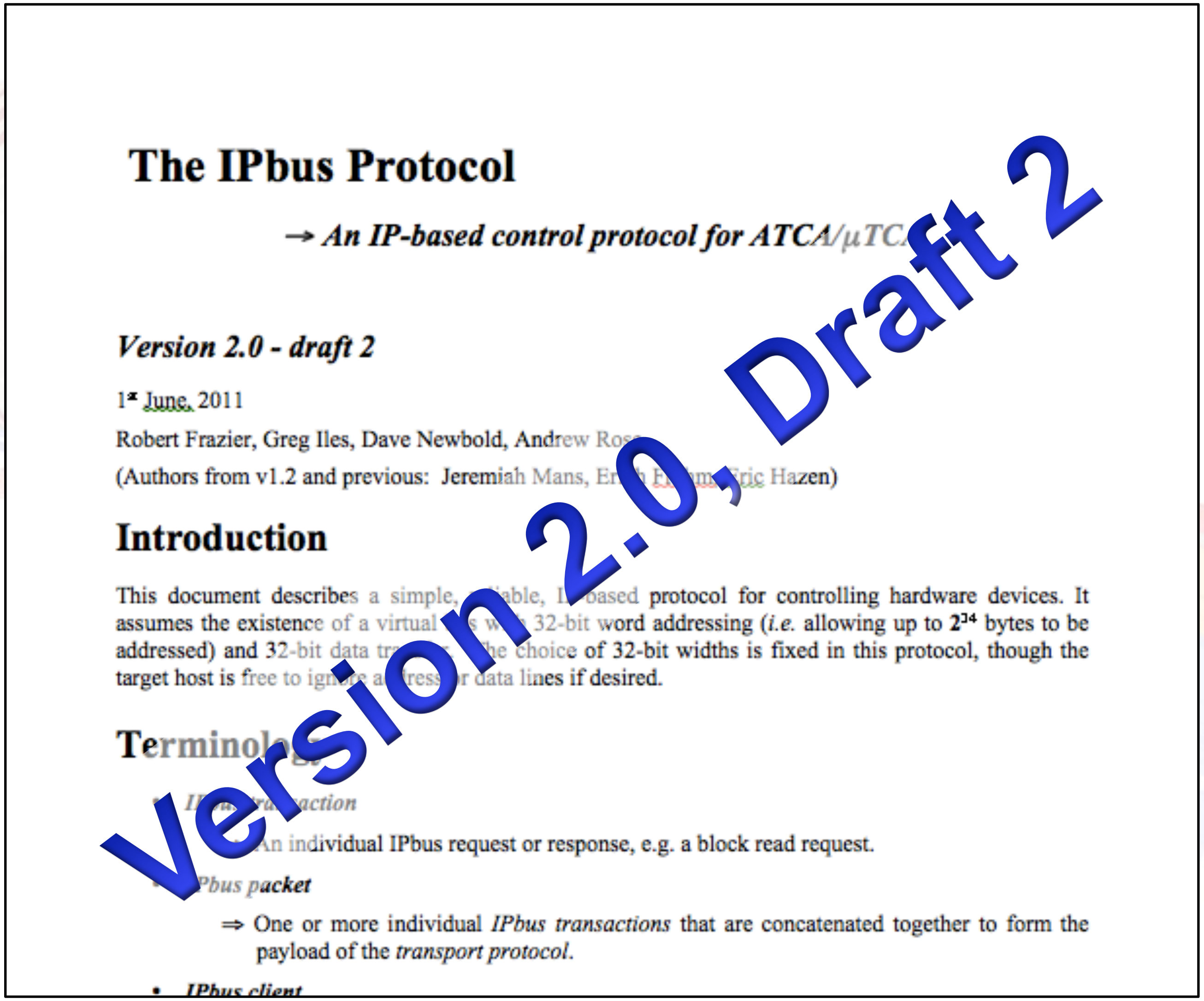
For the initial implementation, the transport is UDP, so each transaction must fit in a single UDP packet. Long block transfers must be split in the software level into individual packets.

A packet consists of a set of transactions. There is no overall header, since that would not fit with the long-term possibility of a TCP implementation which would not be packet-based on the SW side. The number of transactions in a given UDP packet must be deduced from the length of the packet and the contents.

Each transaction carries a transaction identifier, as described below. The transaction id is copied from

Version 1.2

What is IPbus?



The IPbus Protocol

→ An IP-based control protocol for ATCA/μTCA

Version 2.0 - draft 2

1st June, 2011

Robert Frazier, Greg Iles, Dave Newbold, Andrew Rose

(Authors from v1.2 and previous: Jeremiah Mans, Eric Hazen)

Introduction

This document describes a simple, reliable, IP-based protocol for controlling hardware devices. It assumes the existence of a virtual bus with 32-bit word addressing (*i.e.* allowing up to 2^{34} bytes to be addressed) and 32-bit data transfer. The choice of 32-bit widths is fixed in this protocol, though the target host is free to ignore address or data lines if desired.

Terminology

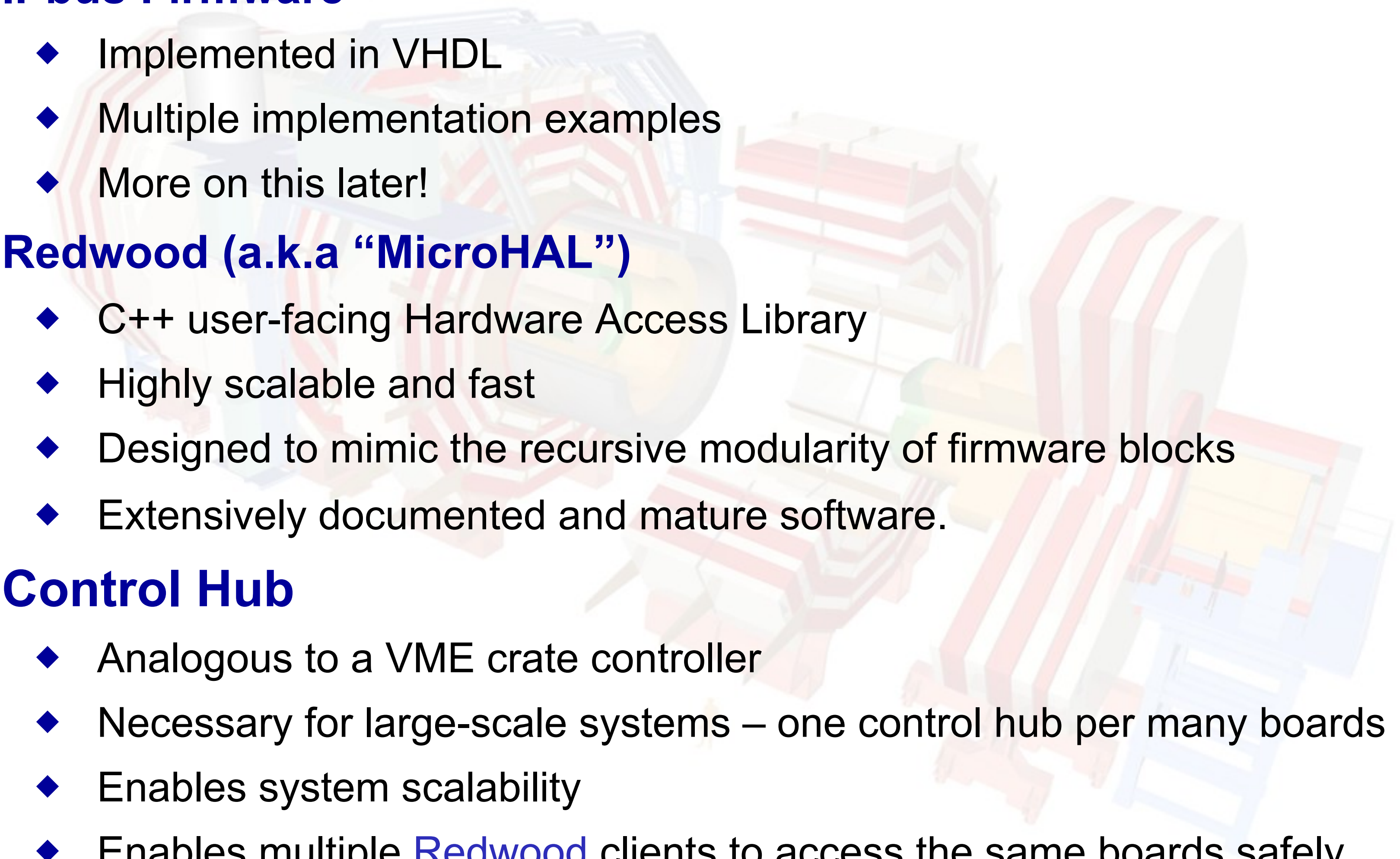
- ***IPbus transaction***
  ⇒ An individual IPbus request or response, e.g. a block read request.
- ***IPbus packet***
  ⇒ One or more individual *IPbus transactions* that are concatenated together to form the payload of the *transport protocol*.
- ***IPbus client***

Version 2.0, Draft 2

What do we need from IPbus?

- Reliability
  - Nothing worse than a flaky system you don’t trust
- Scalability
  - The current L1 alone is ~4000 boards
- It needs to be fast
  - Local DAQ? A tedious process with current VME-based systems
- It needs to be usable and well documented
  - Drop-in firmware modules already exist
    - Examples for a variety of Xilinx demo-boards are available!
  - Two software suites (C++ or Python) already available
    - Both of these are already very mature
- It needs to have strong future support & development
  - UK is committed to this project
  - Current team already has extensive experience in this area.

The IPbus Suite Overview 1



IPbus Firmware

- Implemented in VHDL
- Multiple implementation examples
- More on this later!

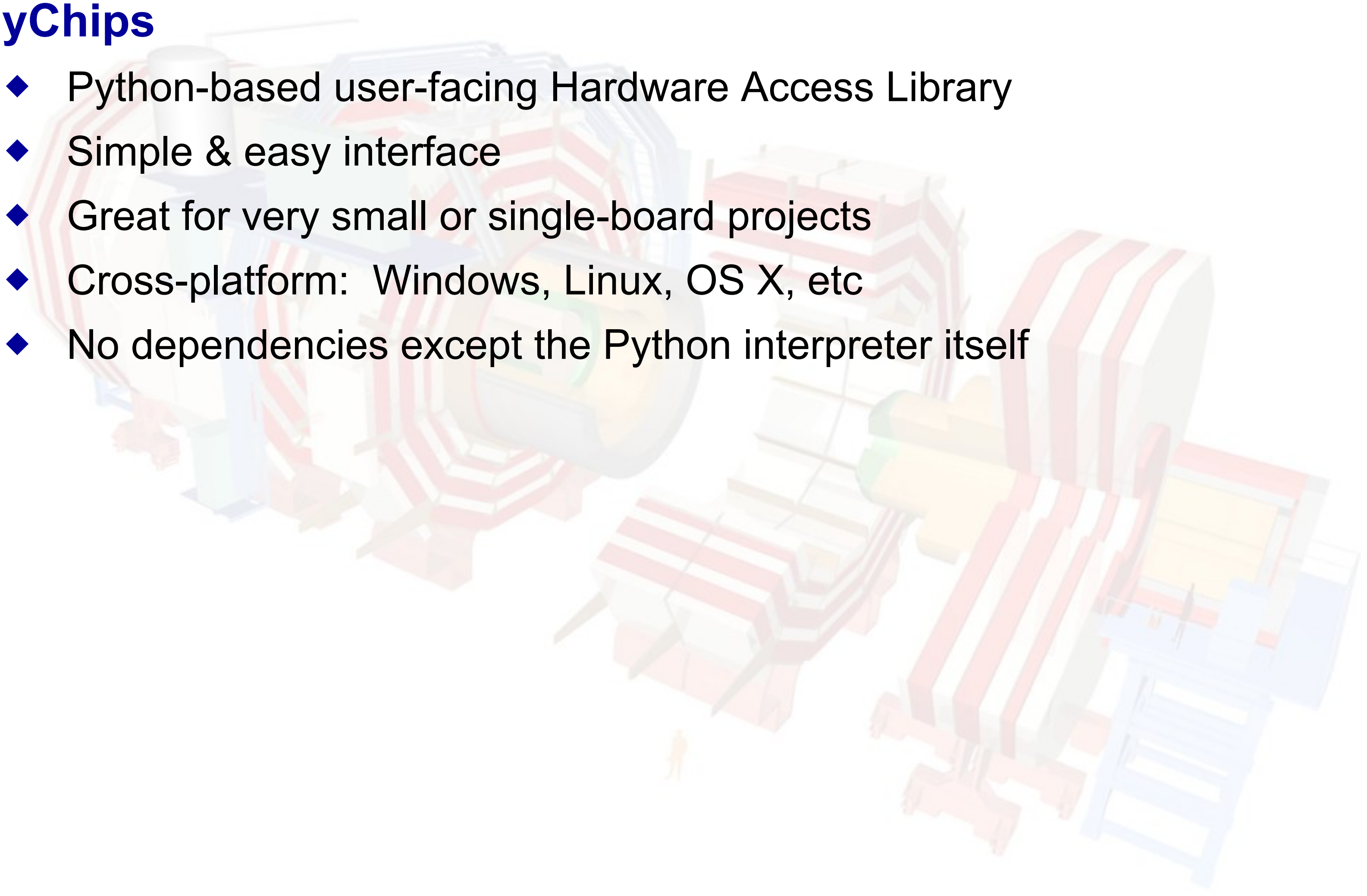
Redwood (a.k.a "MicroHAL")

- C++ user-facing Hardware Access Library
- Highly scalable and fast
- Designed to mimic the recursive modularity of firmware blocks
- Extensively documented and mature software.

Control Hub

- Analogous to a VME crate controller
- Necessary for large-scale systems – one control hub per many boards
- Enables system scalability
- Enables multiple Redwood clients to access the same boards safely

IPbus Suite Overview 2



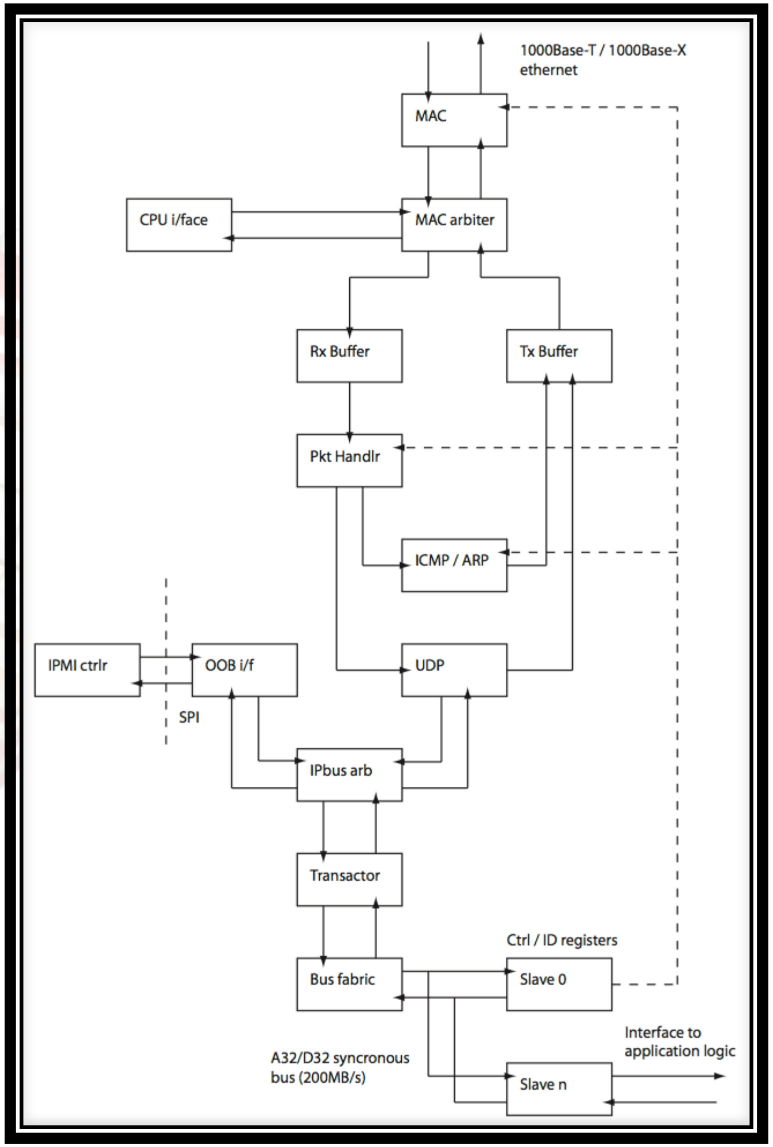
- **PyChips**
  - Python-based user-facing Hardware Access Library
  - Simple & easy interface
  - Great for very small or single-board projects
  - Cross-platform: Windows, Linux, OS X, etc
  - No dependencies except the Python interpreter itself

IPbus Firmware 1

- Original firmware by Jeremy Mans, et al
- Extensively re-worked by Dave Newbold and Andrew Rose
- Implemented in VHDL
- Includes fully working simulation test-bench
  - Simulation responds to IPbus transaction packets over UDP
  - Software tests can be run against the firmware simulation!
    - Ensures complete software/firmware compatibility
- Working implementation examples exist for:
  - Xilinx SP601 (Spartan 6) demo board
  - Xilinx SP605 (Spartan 6) demo board
  - Xilinx ML605 (Virtex 6) demo board
  - Avnet AES-V5FXT-EV30 (Virtex 5) demo board

IPbus Firmware 2

- Firmware overview →
  - Well modularised
  - Dave Newbold can provide more details…
- Can be tailored to many different solutions depending on…
  - Available block RAM
  - Performance requirements, etc
- Matter of ~hours to port to new platform



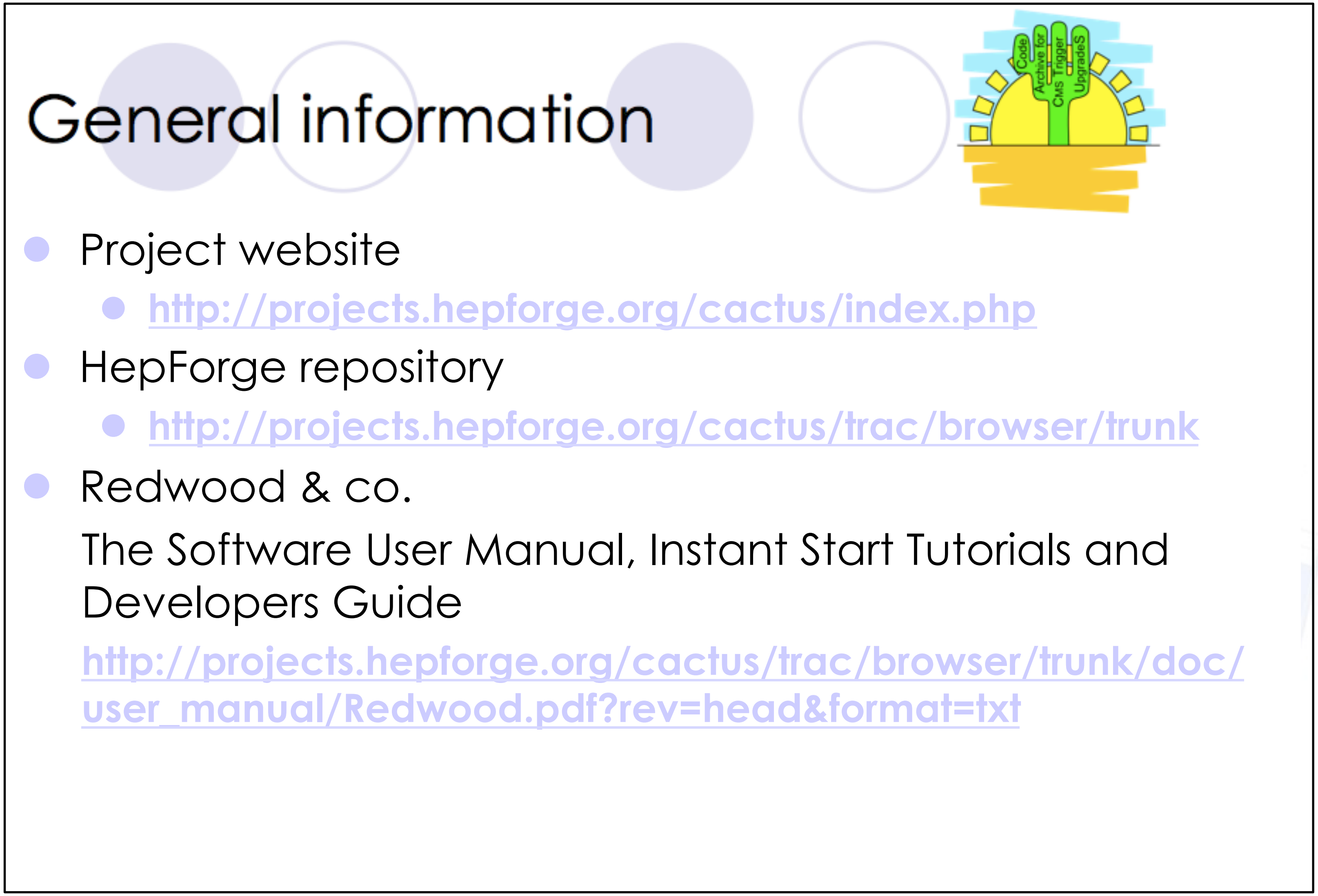
IPbus Firmware 3

- IPbus firmware resource usage:
  - Baseline system is the Xilinx SP601 Demo board
    - Costs £200/$350
    - One of the smallest Spartan 6 FPGAs (XC6LX16-CS324)
    - Uses 7% of registers, 18% of LUTs and 25% BRAM
  - Block RAM usage may increase slightly for v2.0 protocol
- Additional features:
  - Firmware also includes interface to Wisconsin IPMI controller
    - Allow setup/spy via IPbus
  - Can also share Ethernet or IPbus with a soft/hard CPU core

Redwood/MicroHAL 1

- C++ Hardware Access Library for the IPbus protocol
- Designed to reflect the structure of your firmware
  - Firmware is intrinsically hierarchical
  - Redwood allows to write software to mirror this structure
  - Strongly promotes code reuse and modularity
- Fast and scalable in conjunction with Control Hubs
- Can be used in a standalone manner…
  - Redwood Application → Device(s)
- …Or with Control Hubs
  - Redwood Application(s) → Control Hub(s) → Devices

Redwood/MicroHAL 2



General information

- Project website
  - http://projects.hepforge.org/cactus/index.php
- HepForge repository
  - http://projects.hepforge.org/cactus/trac/browser/trunk
- Redwood & co.

  The Software User Manual, Instant Start Tutorials and Developers Guide

  http://projects.hepforge.org/cactus/trac/browser/trunk/doc/user_manual/Redwood.pdf?rev=head&format=txt

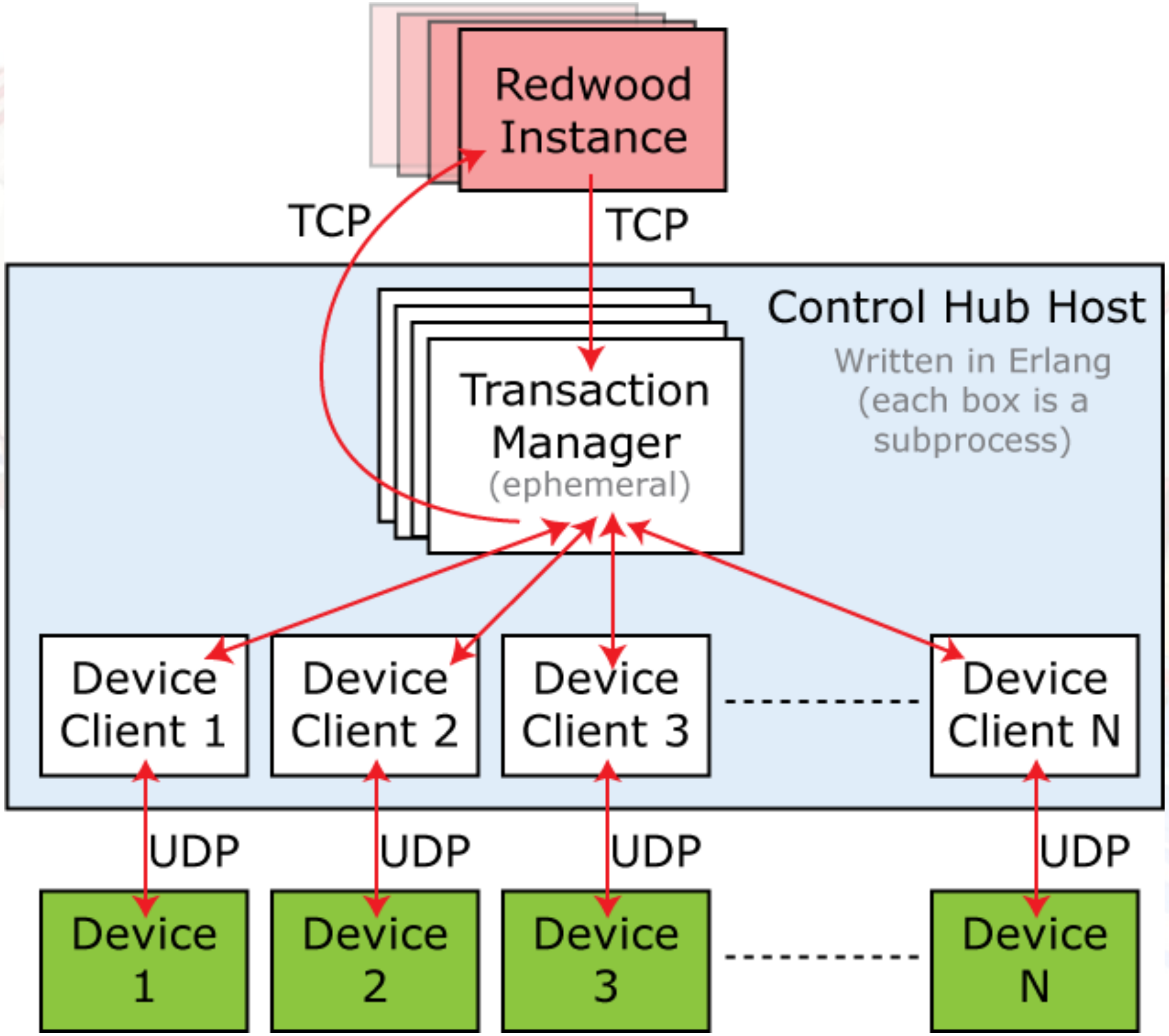
Code
Archive for
CMS Trigger
UpgradeS

Andrew Rose

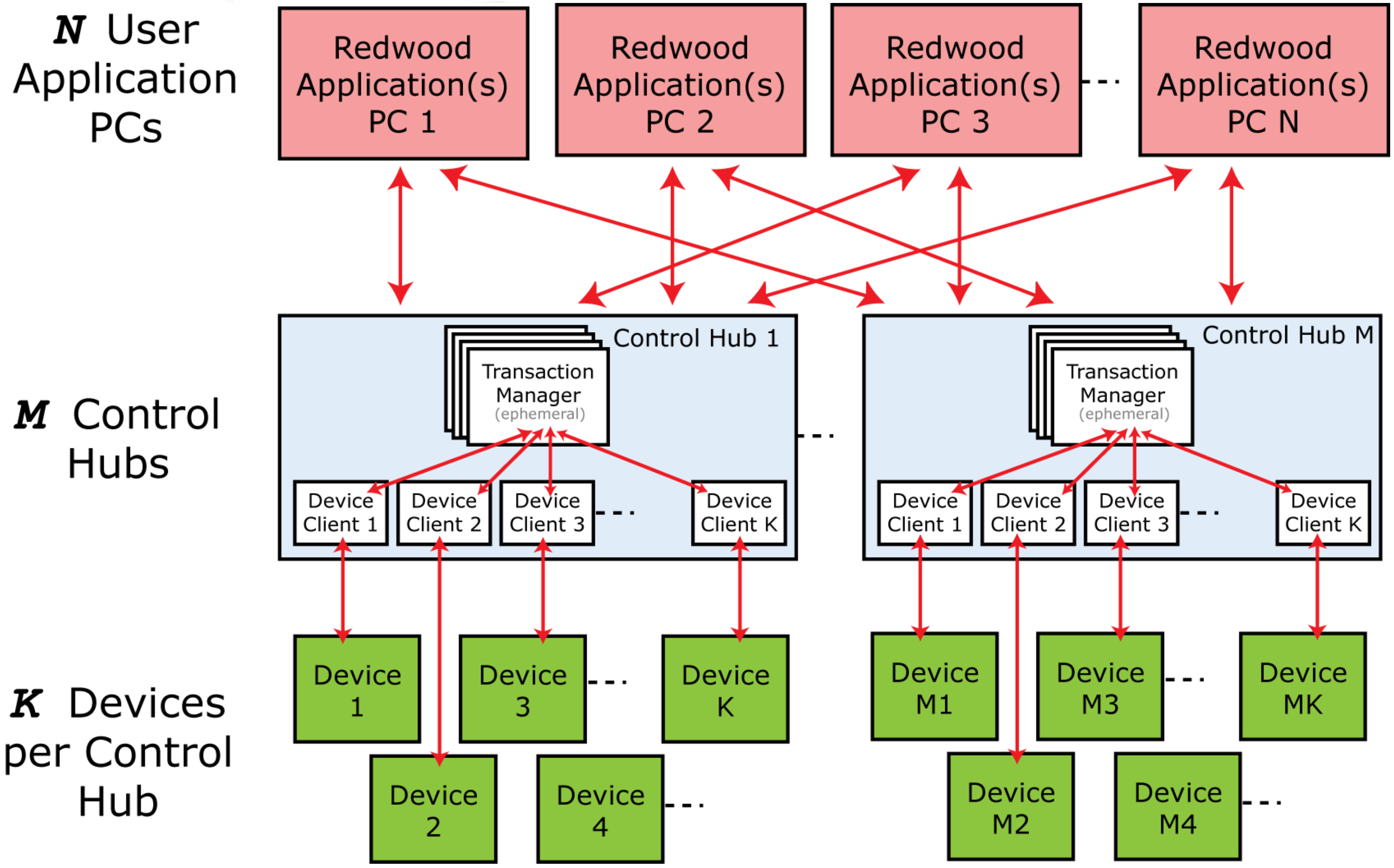
Control Hub 1

- Largely analogous to a VME crate controller
  - Except can control more than a single crate if desired
  - In short: the crate controller is now a rack PC + software
- Single point of contact with hardware
  - Allows multiple applications/clients to access a single board
- Reliability and scalability are crucial!
- Solution: Erlang
  - Concurrent programming language developed for the telecoms industry
    - Joe Armstrong, et al, at Ericsson
  - Designed for robustness, concurrency, scalability and reliability
  - Scales transparently across multiple CPU cores
  - Ericsson have achieved Erlang systems with 99.9999999 percent reliability
    - 31 milliseconds of downtime in a year!

Control Hub 2



Scalability with Redwood and the Control Hub



PyChips

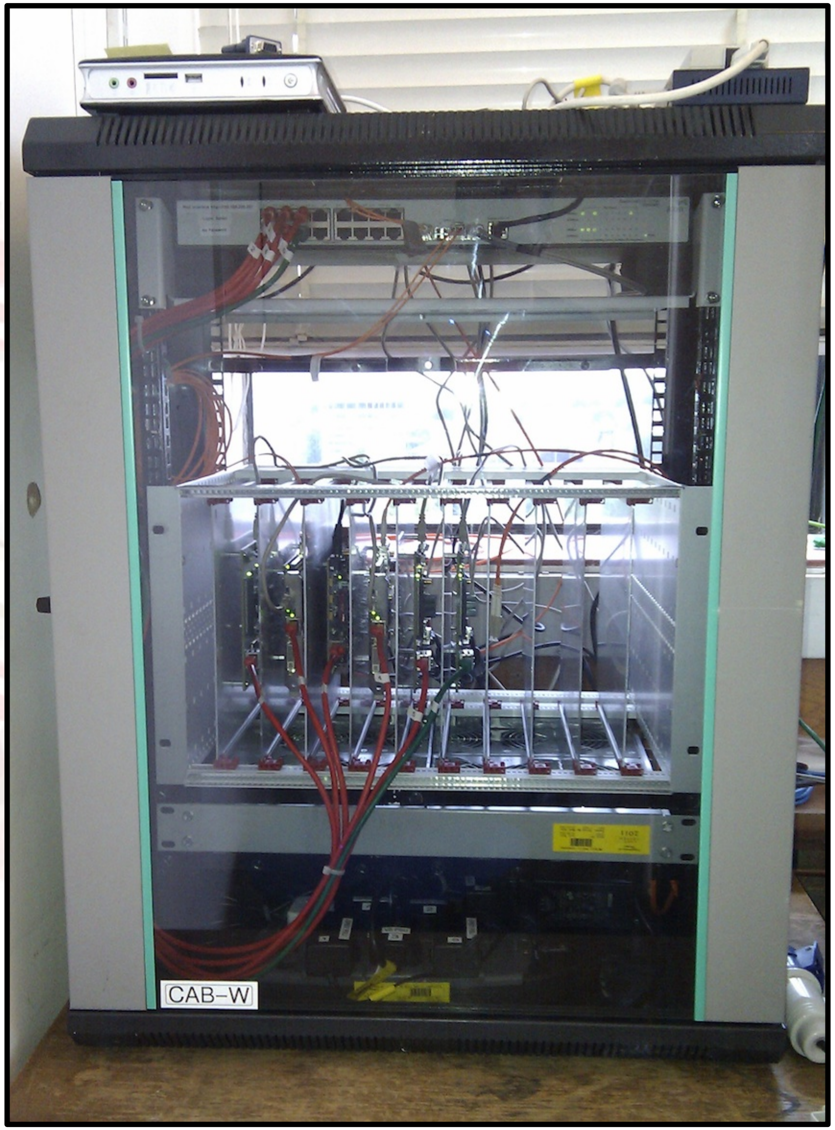
- Simple Python API for creating IPbus applications
  - Use to create short control scripts…
  - … or something more complex!
- Absolutely perfect for most single-board projects
  - Particularly if that single board is an inexpensive Xilinx demo board!
  - We already have several such projects running at Bristol
    - E.g. CMS Binary Chip Test Platform project.
- Cross platform
  - Anywhere you can install a Python interpreter!
- Shortcomings:
  - Not at all scalable
  - Certainly not fast for DAQ purposes (max ~1 MB second read)

Testing the IPbus Suite

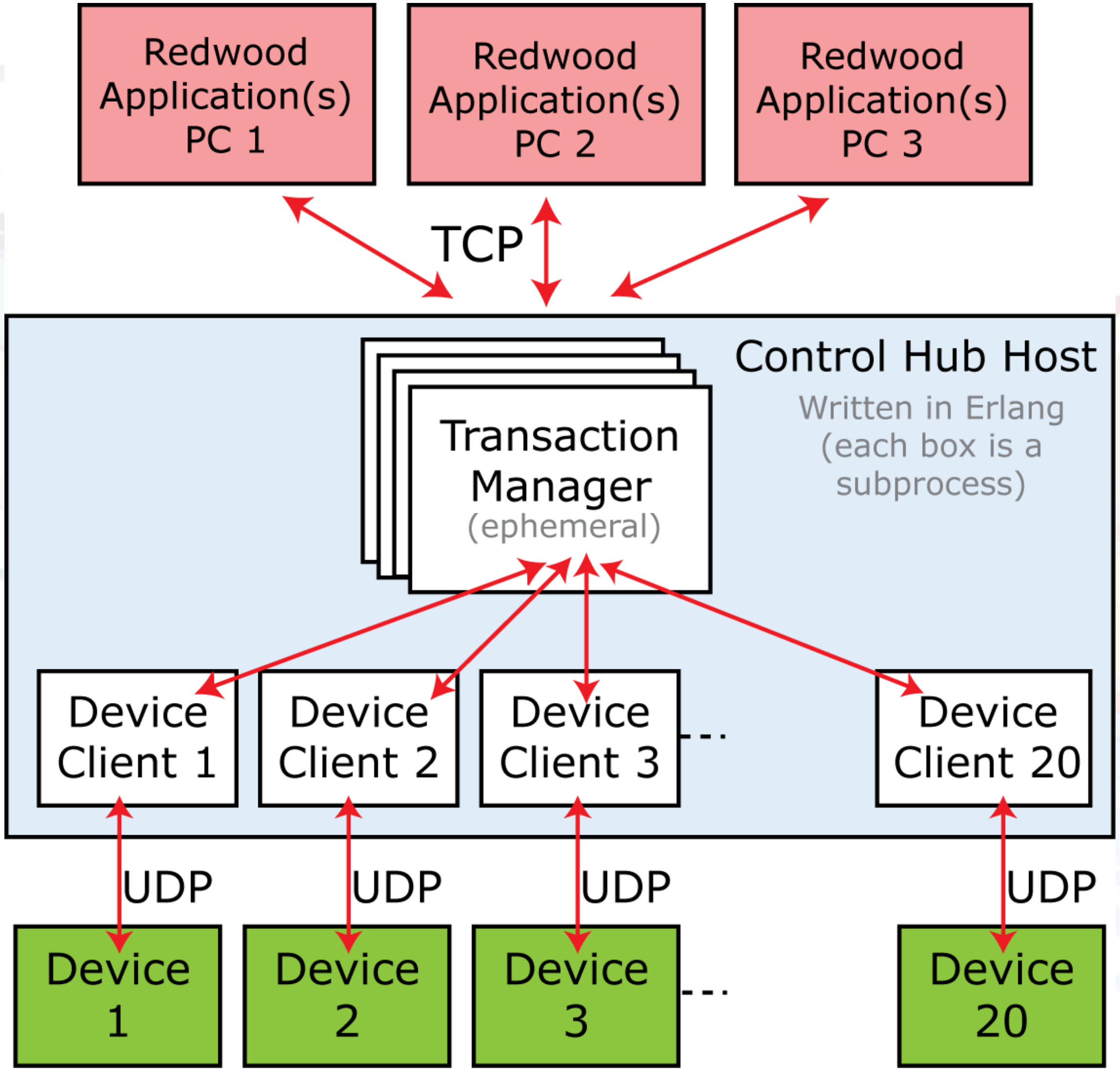
- We wanted a fully representative test system
- Wanted to test many things:
  - Throughput
    - Single board throughput
    - Multiple board throughput
    - Full chain throughput: Redwood➔ Control Hub ➔ Board(s)
  - Reliability
    - Find protocol problems
    - Find interface problems
    - Long soak tests
  - Scalability
    - CPU usage of Control Hub
    - How many boards can the Control Hub serve?
    - Number of possible Redwood clients, etc.

IPbus Test System 1

- **3** user-level (Redwood) rack PCs
- **1** Control Hub rack PC
  - Connecting to H/W via two switches (using fibre)
  - Final fan-out to boards using Cat 5e
- **20** IPbus clients
  - Running on 6 development boards:
  - 3 x Xilinx SP601 (Spartan 6)
  - 2 x Xilinx SP605 (Spartan 6)
  - 1 x Avnet AES-V5FXT-EV30 (Virtex 5)



IPbus Test System 2


Redwood Application(s) PC 1
Redwood Application(s) PC 2
Redwood Application(s) PC 3
TCP
Control Hub Host
Written in Erlang (each box is a subprocess)
Transaction Manager (ephemeral)
Device Client 1
Device Client 2
Device Client 3
...
Device Client 20
UDP
UDP
UDP
UDP
Device 1
Device 2
Device 3
...
Device 20

IPbus Testing – Reliability 1

- Clearly UDP is not a reliable transport protocol
  - Current v1.2 protocol does not provide an error/retry mechanism
  - Version 2.0 protocol remedies this
    - Software/Firmware suite undergoing transition to Version 2.0
- Many potential forms of error:
  1. Outbound packet loss
  2. Return packet loss
  3. Multi-transaction packets that fail part-way through
  4. Packet duplication
  5. Out of order packets
- Why not use a reliable transport protocol, such as TCP?
  - Very complex to implement at firmware level
  - Slow when using embedded processors with TCP stack
  - Not excluded by the protocol, but doesn’t solve everything

IPbus Testing – Reliability 2

- How reliable is the IPbus v1.2 protocol currently?
  - I.e. without the error/retry mechanism v2.0 protocol will bring
- Answer: actually pretty good
  - On a private network just for hardware, with…
  - Simple network topology
  - Good cables/fibre
  - All unnecessary network protocols switched off (spanning tree, etc)
- Testing involved sending 5 billion block read requests
  - 10 billion packets total, 53 went missing.
  - 350 * 32-bit block read
  - 7 Terabytes IPbus payload data received
  - 19 IPbus clients used in test
- Packet loss averages at **1 in 189 million** UDP packets

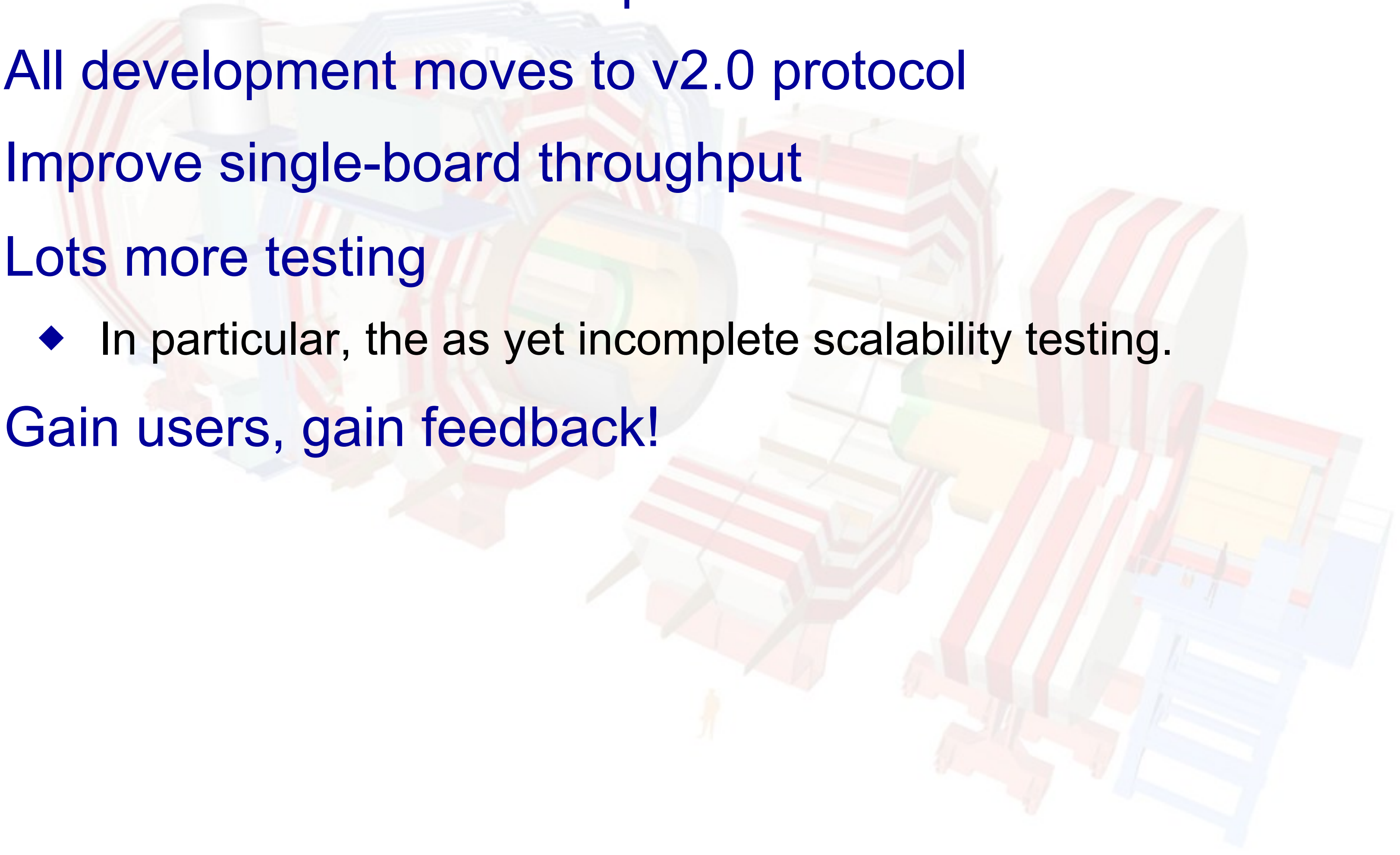
Throughput Tests

- Single-board throughput actually limited by firmware
  - Currently 60 Mbit/s Tx or Rx for a single board
  - Limitation caused by moving IPbus data around internally
    - 5 copy stages currently
    - Being reduced to 3
    - Other performance tweaks also being done
    - Aim to improve to >100 Mbit/s
- Multi-board throughput
  - 600 Mbit/s receive achieved to 19 IPbus instances
  - Not clear why this is ~half the single-board throughput yet
    - Possibly to do with hosting multiple IPbus instances on single board
- In summary – more than good enough for now
  - Plenty of low-hanging fruit for improvement.

Scalability Tests

- Still more work to do on this…
  - 600 Mbit/s to 19 boards uses less than 3 logical cores
    - Twin-socket, 2.4 GHz Nehalem server (8 physical/16 logical cores)
    - Lack of CPU resources not an issue currently
  - Doesn’t yet included data being received from Redwood clients and being repackaged + routed.
  - Didn’t quite finish my tests in time for this talk ☹

What's Next?



- Final release of v1.2 compatible software/firmware
- All development moves to v2.0 protocol
- Improve single-board throughput
- Lots more testing
  - In particular, the as yet incomplete scalability testing.
- Gain users, gain feedback!

Summary/Conclusions

- The IPbus Protocol Suite aims to provide a standard control interface for Ethernet-attached hardware (xTCA, etc).
  - For small- and large-scale projects
- Mature software and firmware already available
  - Although many improvements still to come with v2.0 protocol
  - Large system scalability testing still needs to be completed

- Firmware queries, contact Dave Newbold: dave.newbold@cern.ch
- Software queries, contact me: robert.frazier@cern.ch
- Project home: https://projects.hepforge.org/cactus/index.php