# Using execution traces to debug multicore SoCs: An industrial experience

## MAD Workshop

October the 8th 2014

Miguel Santana
STMicroelectronics

life.augmented

# Outline



- Debugging multicore SoCs
- Execution traces in a multicore SoC
- Managing traces for multimedia products
- Real use cases
- Conclusion

# DEBUGGING MULTICORE SOC'S

# Multicore SoCs

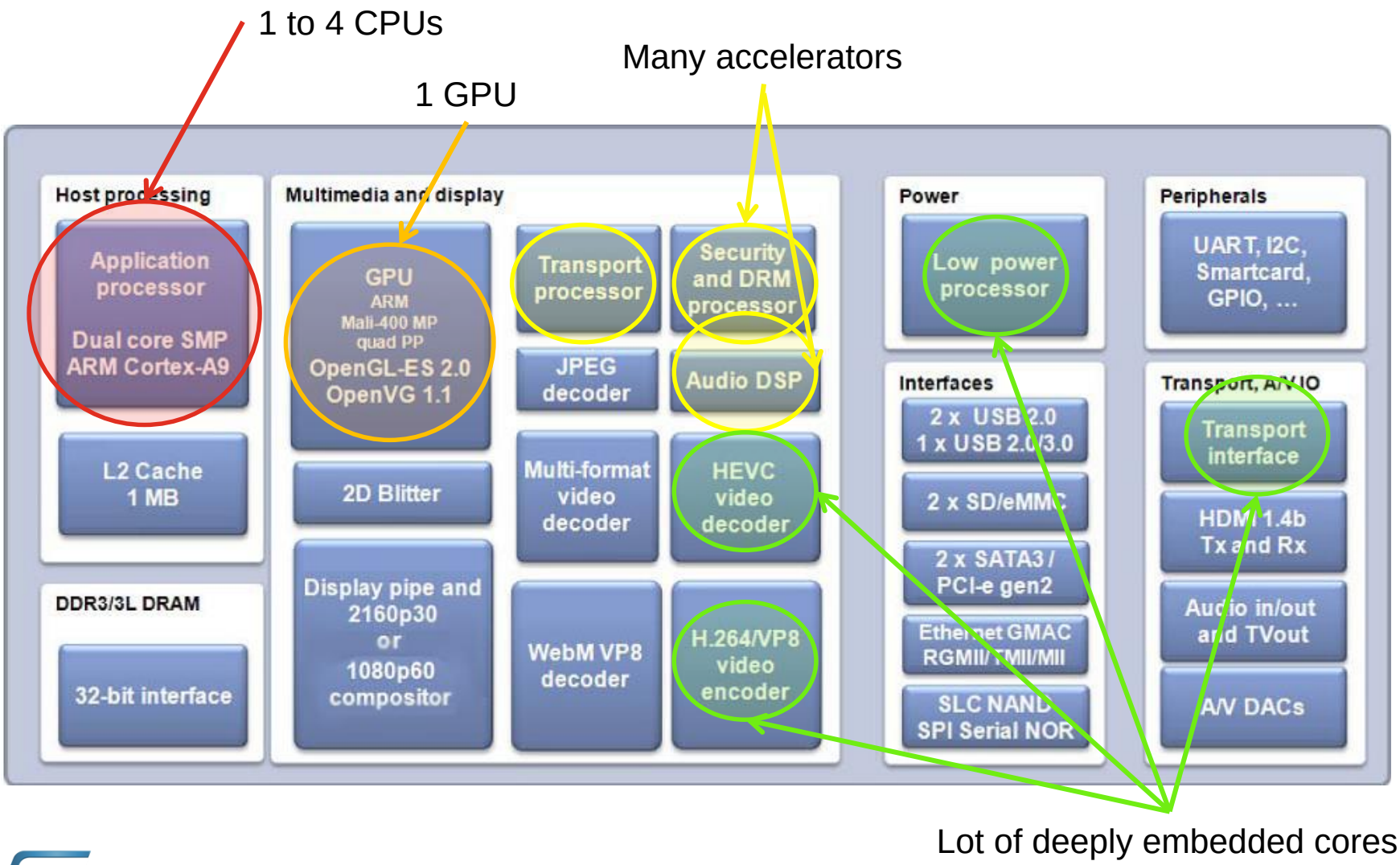


- Electronic devices are pervasive nowadays

- Many of them are based on multicore architectures
  - Mostly heterogeneous, having several categories of cores
  - Host or application processors are more and more based on SMP model
  - Clusters of processors are emerging
- Multimedia applications are a typical example of multicore arch.

# Cannes/Monaco

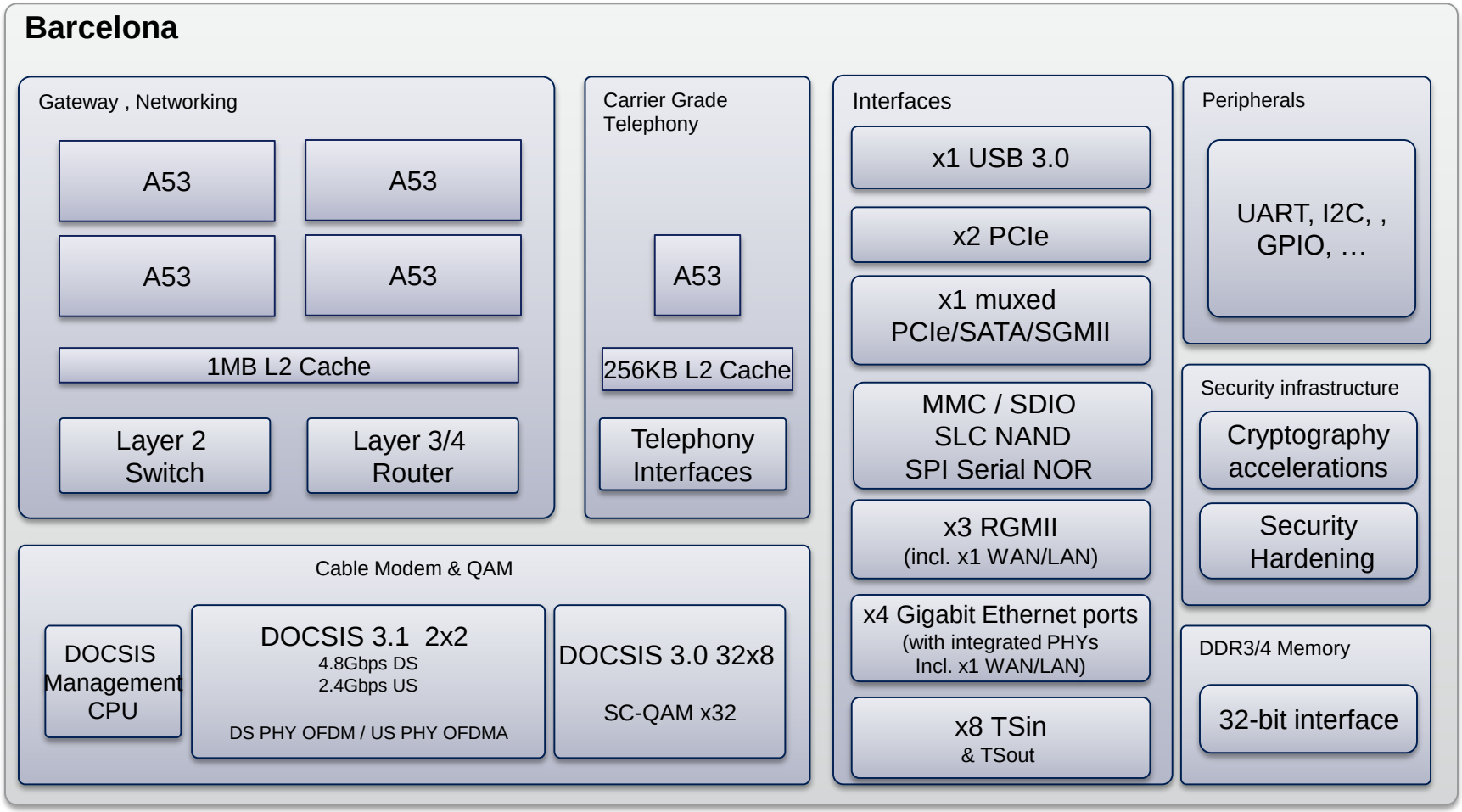





life.augmented

# Barcelona

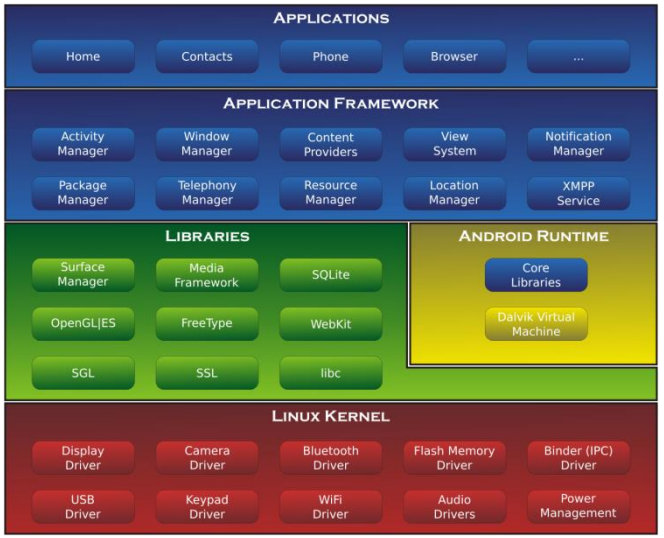



Barcelona
Gateway , Networking
A53
A53
A53
A53
1MB L2 Cache
Layer 2 Switch
Layer 3/4 Router
Carrier Grade Telephony
A53
256KB L2 Cache
Telephony Interfaces
Cable Modem & QAM
DOCSIS Management CPU
DOCSIS 3.1 2x2
4.8Gbps DS
2.4Gbps US
DS PHY OFDM / US PHY OFDMA
DOCSIS 3.0 32x8
SC-QAM x32
Interfaces
x1 USB 3.0
x2 PCIe
x1 muxed PCIe/SATA/SGMII
MMC / SDIO
SLC NAND
SPI Serial NOR
x3 RGMII
(incl. x1 WAN/LAN)
x4 Gigabit Ethernet ports
(with integrated PHYs
Incl. x1 WAN/LAN)
x8 TSin
& TSout
Peripherals
UART, I2C, , GPIO, …
Security infrastructure
Cryptography accelerations
Security Hardening
DDR3/4 Memory
32-bit interface

# Multimedia software stack



- Composed of many layers and components
  - Host side
    - Linux or Android stacks
    - Multimedia layers enriching kernel and user space
    - Application middleware (CDI, RDK, GoogleTV…)
    - Operator applications (Netflix, Orange, Sky…)
  - Accelerators side
    - RTOS
    - Multimedia layers
  - Deeply embedded software

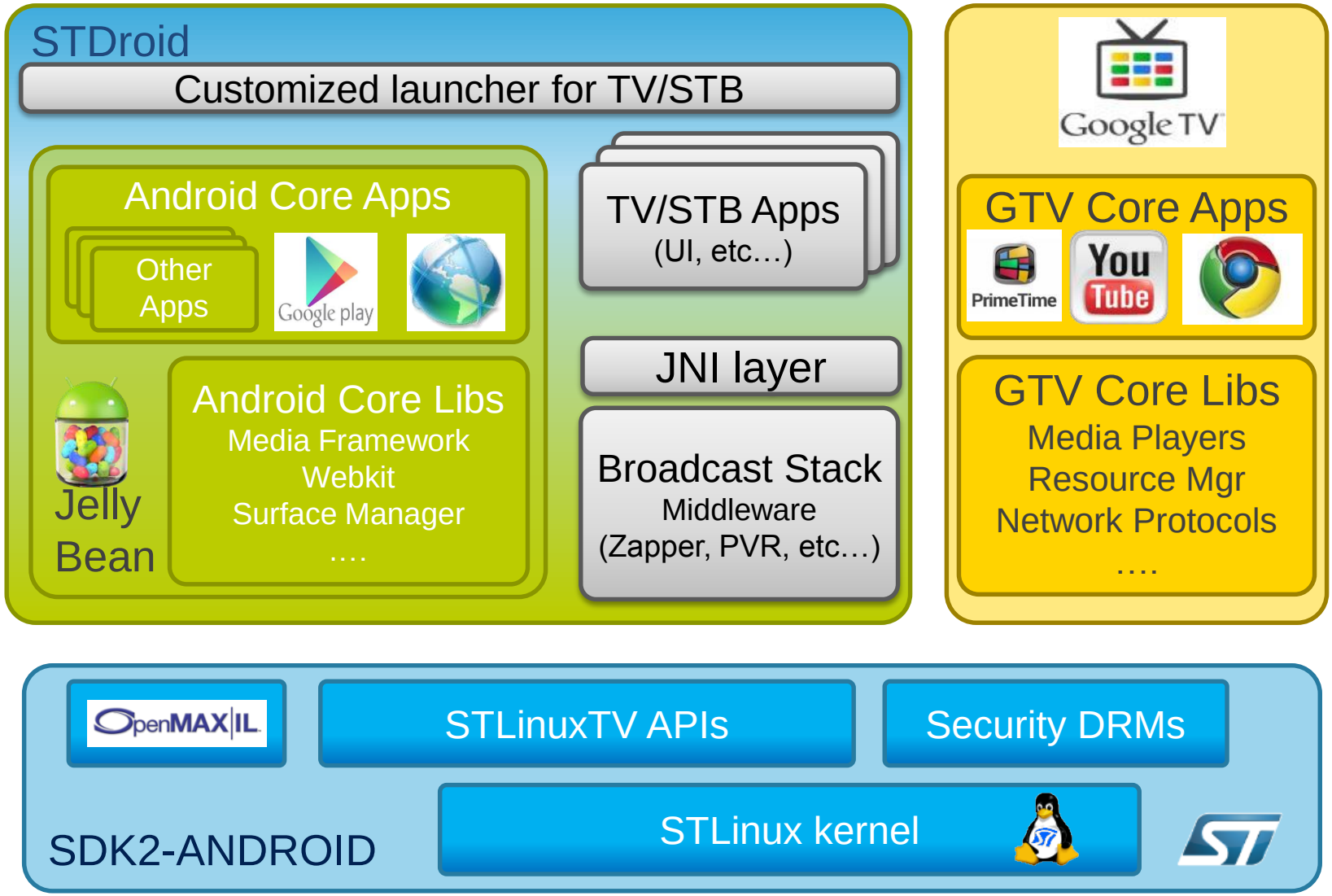



- Strong interactions between cores
  - Multithreading
  - Distributed systems using communication & synchronization

- So highly parallel environment…

# Google TV Stack





# Debugging Multimedia SoCs



- Dealing with a quite challenging environment
  - Heterogeneous multicore architecture
  - Real-time environment with strong QoS constraints
  - Complex software components interaction
  - Power management islands

  etc.

- Traditional debuggers are useful but…
  - Limited to functional aspects, preferably at early software validation stages
  - May help later on but can be quickly unusable because of their intrusiveness
  - They are anyway used as part of an enlarged debug toolset

- Back to execution traces to debug what remains!!
  - Linux mechanisms: printk, loggers, spies, etc.
  - Wider system mechanisms both HW and SW

# EXECUTION TRACES IN A MULTICORE SOC

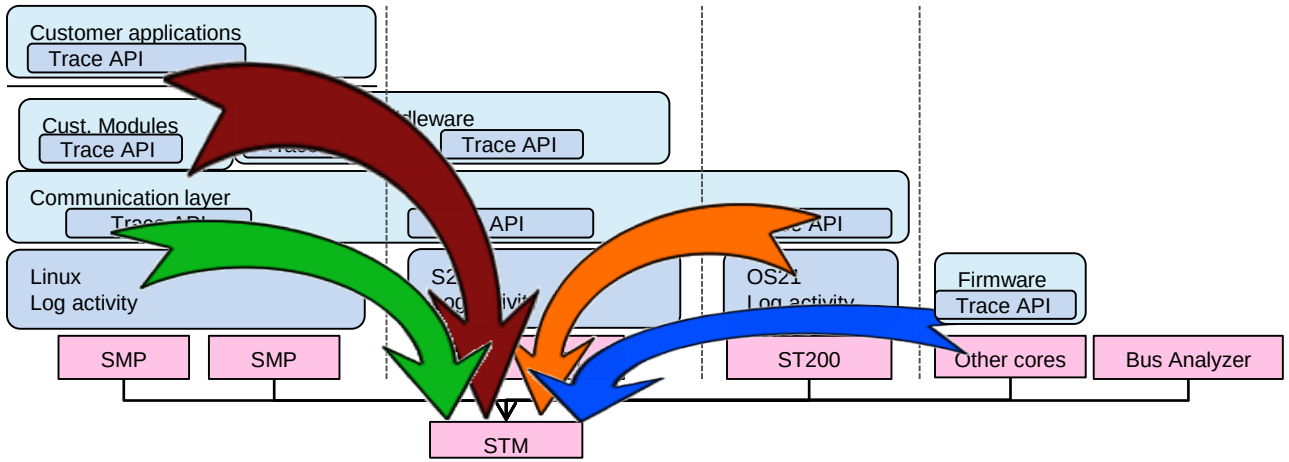
# Execution Traces



- Logging information about the execution of a software
  - Different levels of abstraction: application, kernel events, instructions, etc.
  - Using different mechanisms: HW IPs, kernel mechanisms, instrumentation, etc.
  - Recording on different medias: buffers, I/O ports, files, etc.
- System-on-Chip traces include traces from the whole system
  - Multithreading traces from the host core
  - Multicore traces from all the accelerators and deeply embedded cores
  - Hardware traces
- Need for a unified solution to log all these traces

# System Traces in ST

- Using a dedicated HW block to manage such traces
  - Industry standard defined by MIPI organization: STM (System Trace Module)
  - Collect, timestamp and transport traces coming from the whole system
  - Traces are evacuated through a specific I/O port
  - The lowest intrusiveness for software: message written to I/O registers
- Provide visibility at system level for embedded device behavior
  - Can be used separately to debug a part of the system, or jointly to correlate behaviors and debug a system-level issue
  - Traces are captured and analyzed at post-mortem time



# When System Traces Are Used?

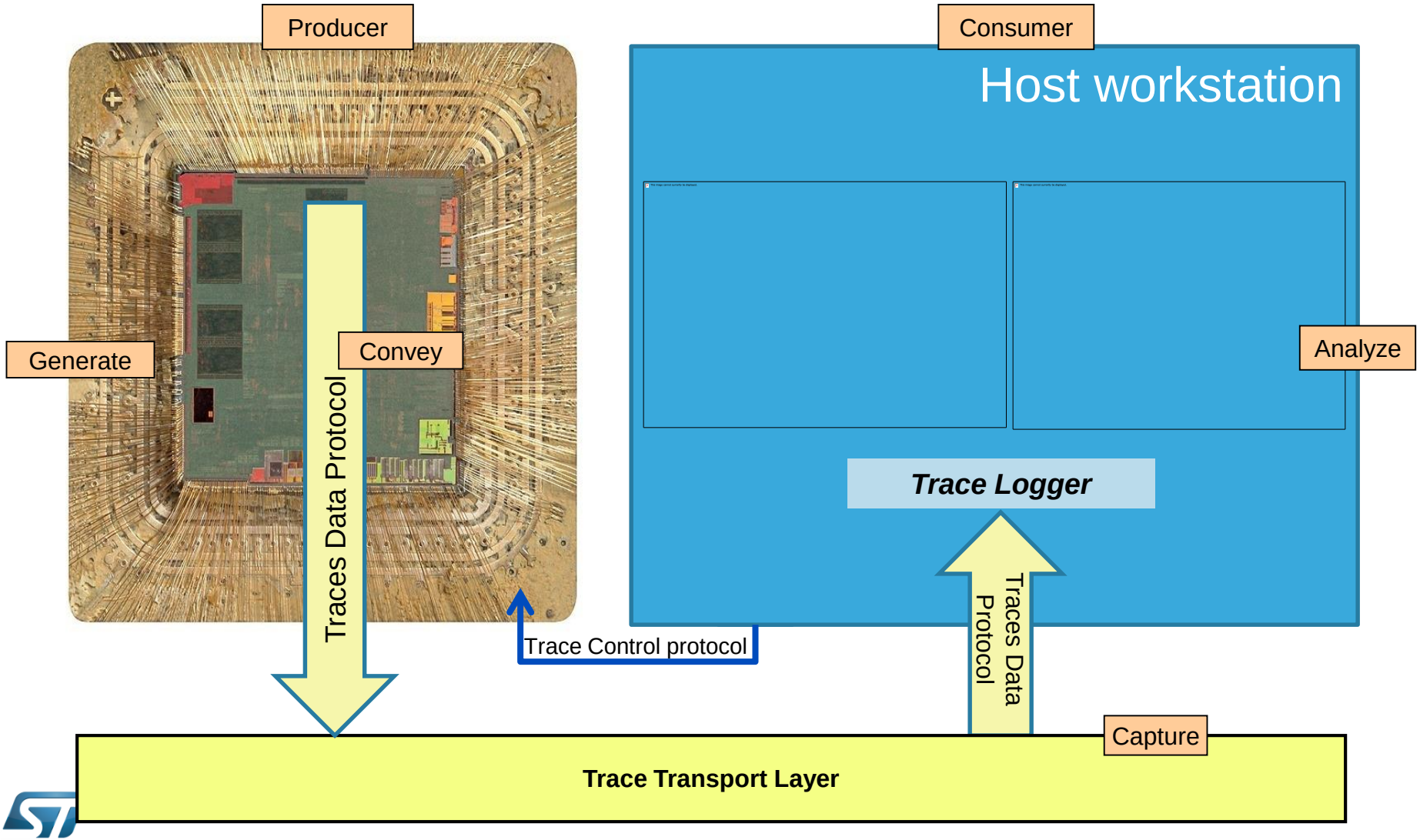


- Looking for anomalies or performance issues
  - Understanding real application behaviour
  - Performance validation and optimization
  - Debugging real-time issues
- Observation and debug of complex applications
  - Processes, interrupts
  - System scheduling, system calls
  - Applicative traces
- Support for regression tests during system test
  - Traces for comparisons between two system runs
  - Monitoring of state variables

# MANAGING TRACES FOR MULTIMEDIA PRODUCTS

# Trace Flow

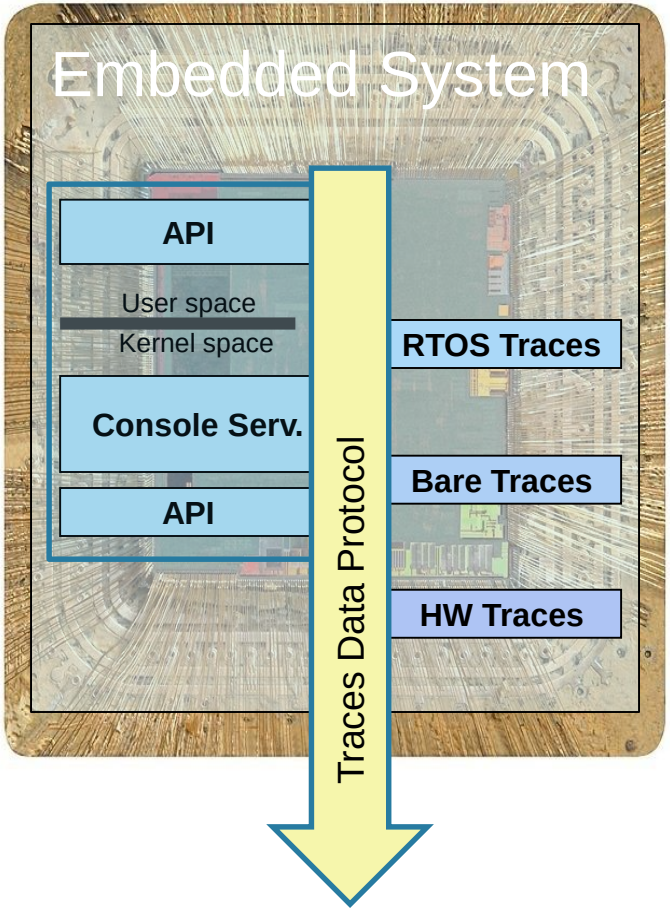



Producer
Consumer
Host workstation
Generate
Convey
Analyze
Traces Data Protocol
Trace Logger
Trace Control protocol
Traces Data Protocol
Capture
Trace Transport Layer

# Trace Providers



- Hardware traces
  - Monitoring IPs
  - Instruction traces
- Software traces
  - Linux/Android traces
    - Application instr.
    - Kernel monitoring
    - Kernel instr.
    - Console
  - RTOS traces
    - Application instr.
    - RTOS monitoring
  - Bare machine traces
    - Application instr.



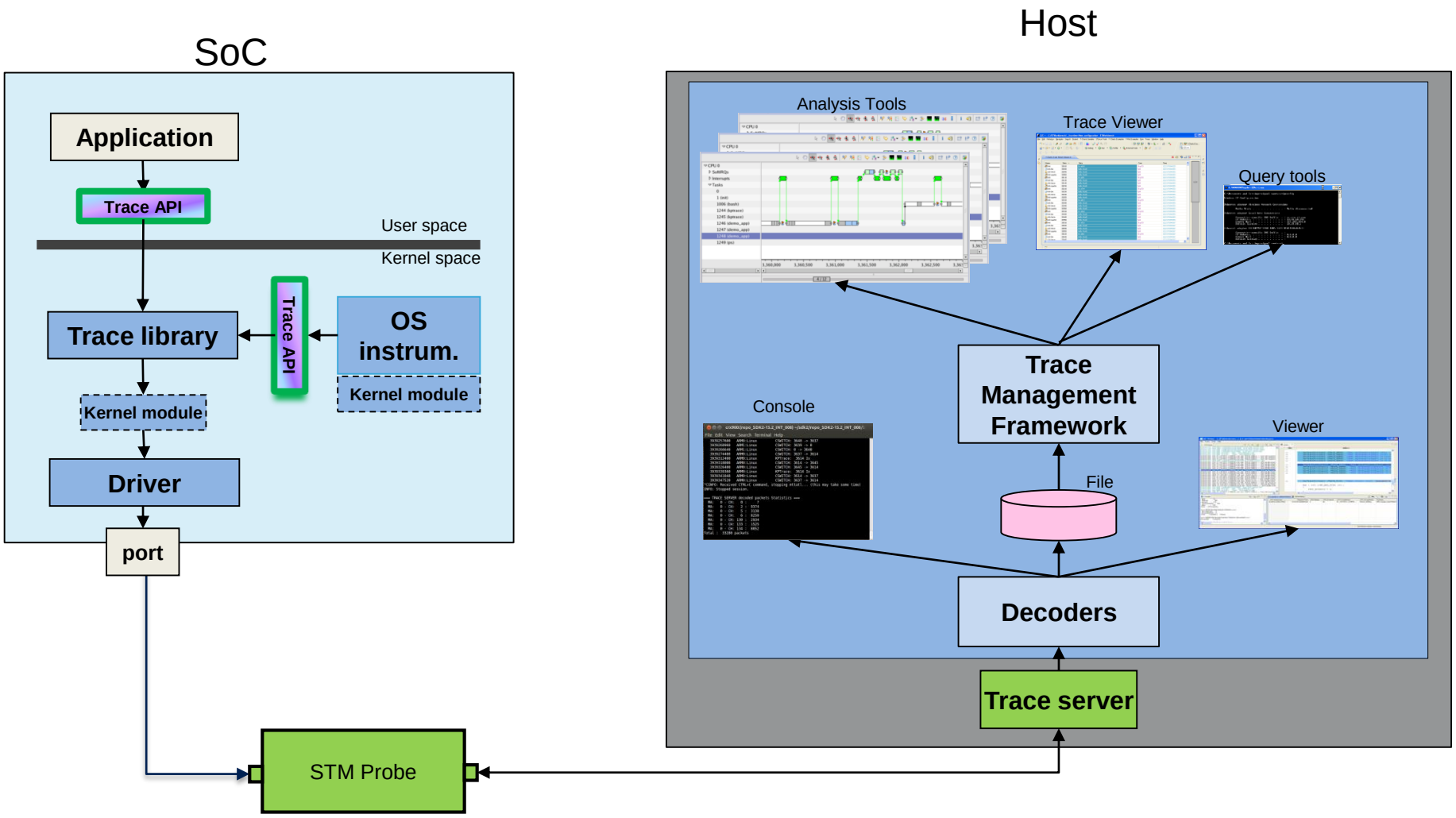
# System Trace Infrastructure



- Our unified solution to manage traces
  - Generation, capture, decoding and analysis of traces
  - Specific data protocol for trace messages
  - Logging features available across the whole system
  - Modular architecture to allow extensibility
- Standard tracing mechanisms diverted to infrastructure
  - printk traces
  - Linux and Android kernel traces
  - Trace generation tools as kptrace or ftrace
- Trace transport layer is transparent to the infrastructure
  - Standard output is STM ports but can use Ethernet, file-system or whatever
- Trace messages made available in human readable format at workstation level
  - Either through GUI or command line

# Instantiating the Flow for Linux

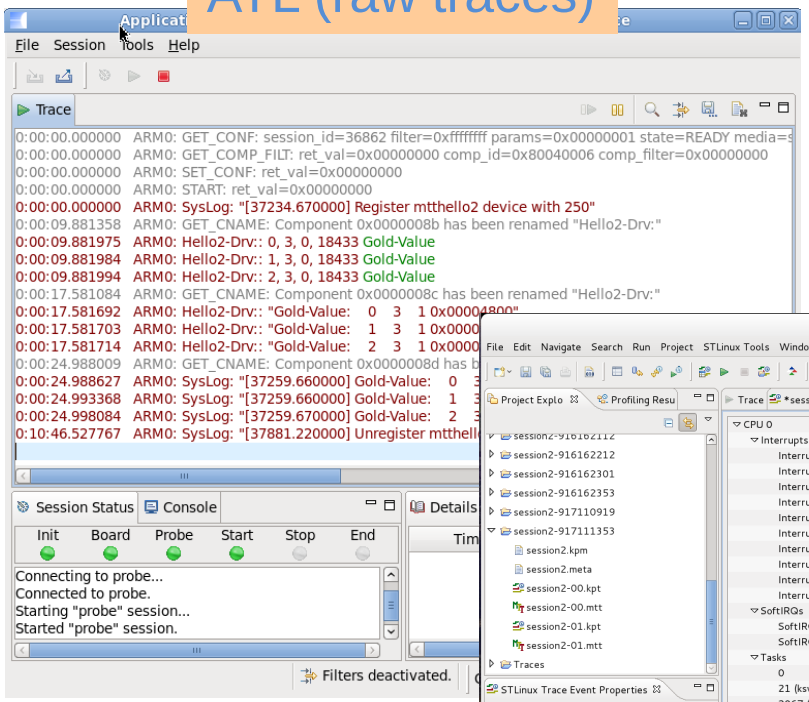




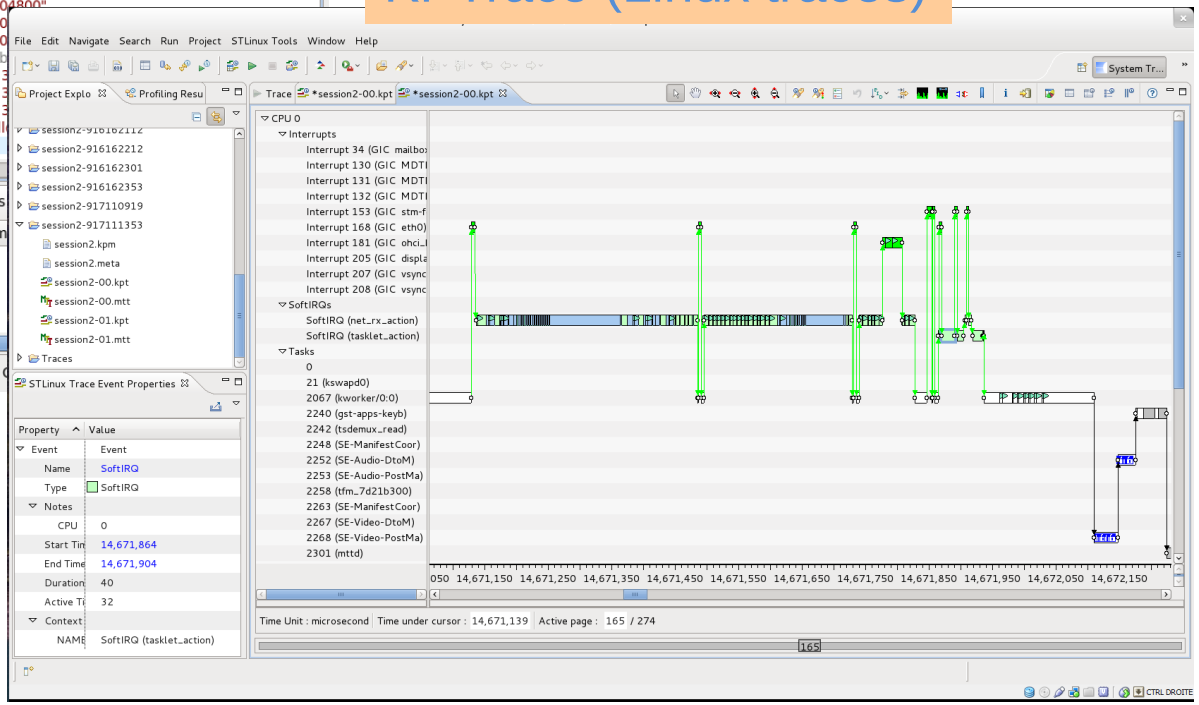
# Some Analysis Tools



ATL (raw traces)



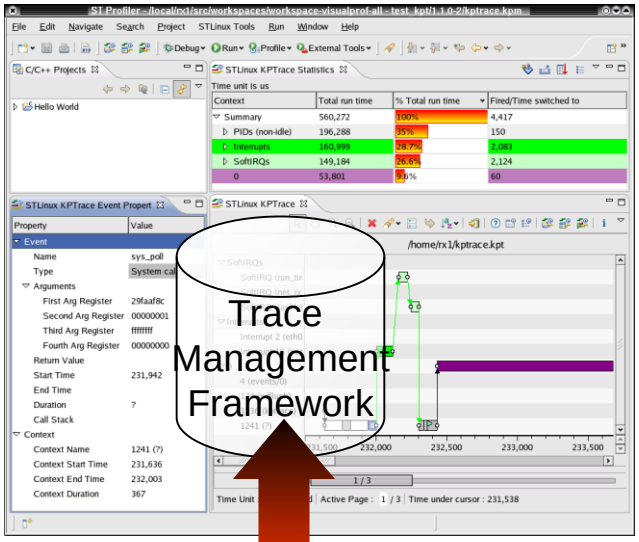

KPTrace (Linux traces)

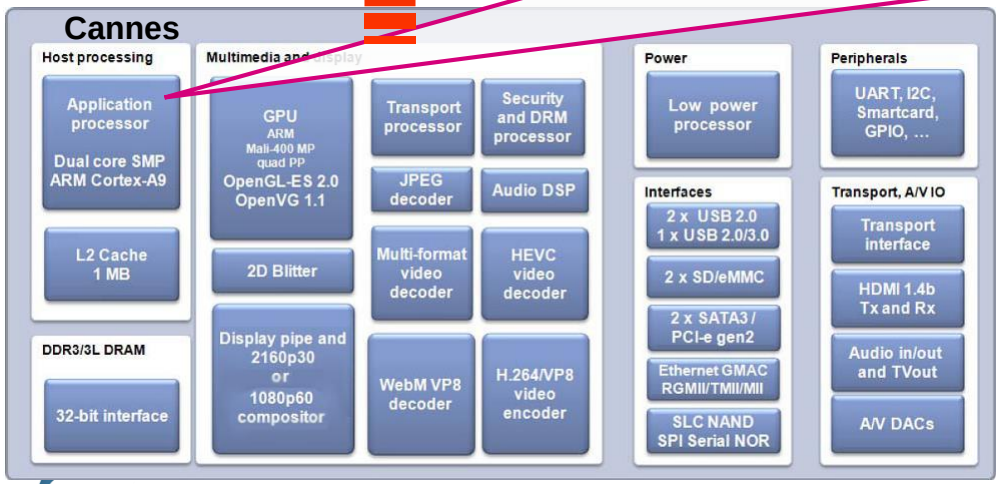


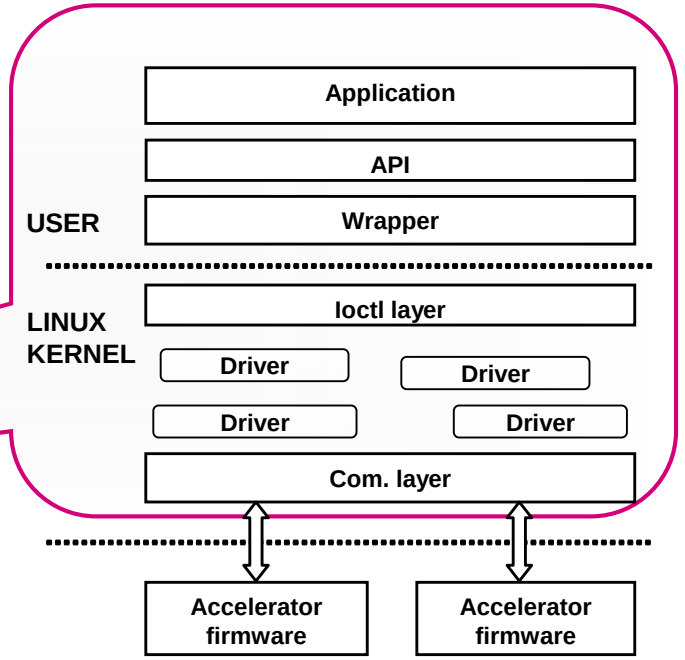

life.augmented

# REAL USE CASES

# Use Cases Environment

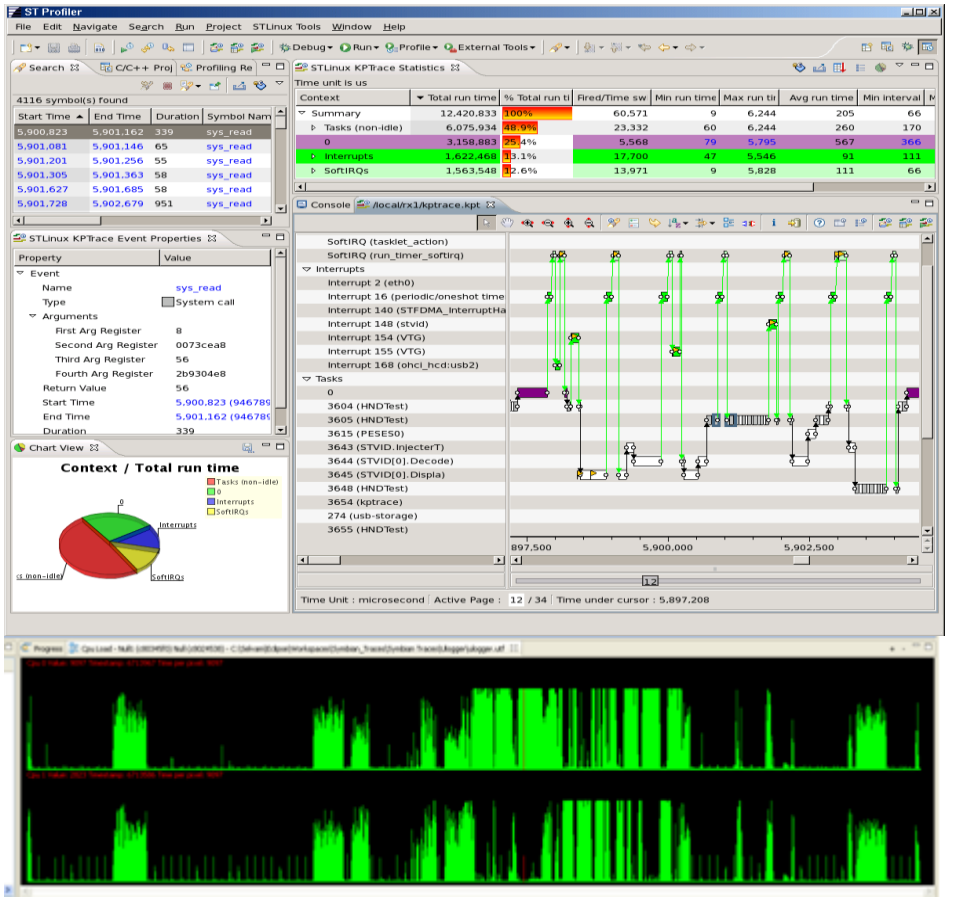


# Analysing the Traces

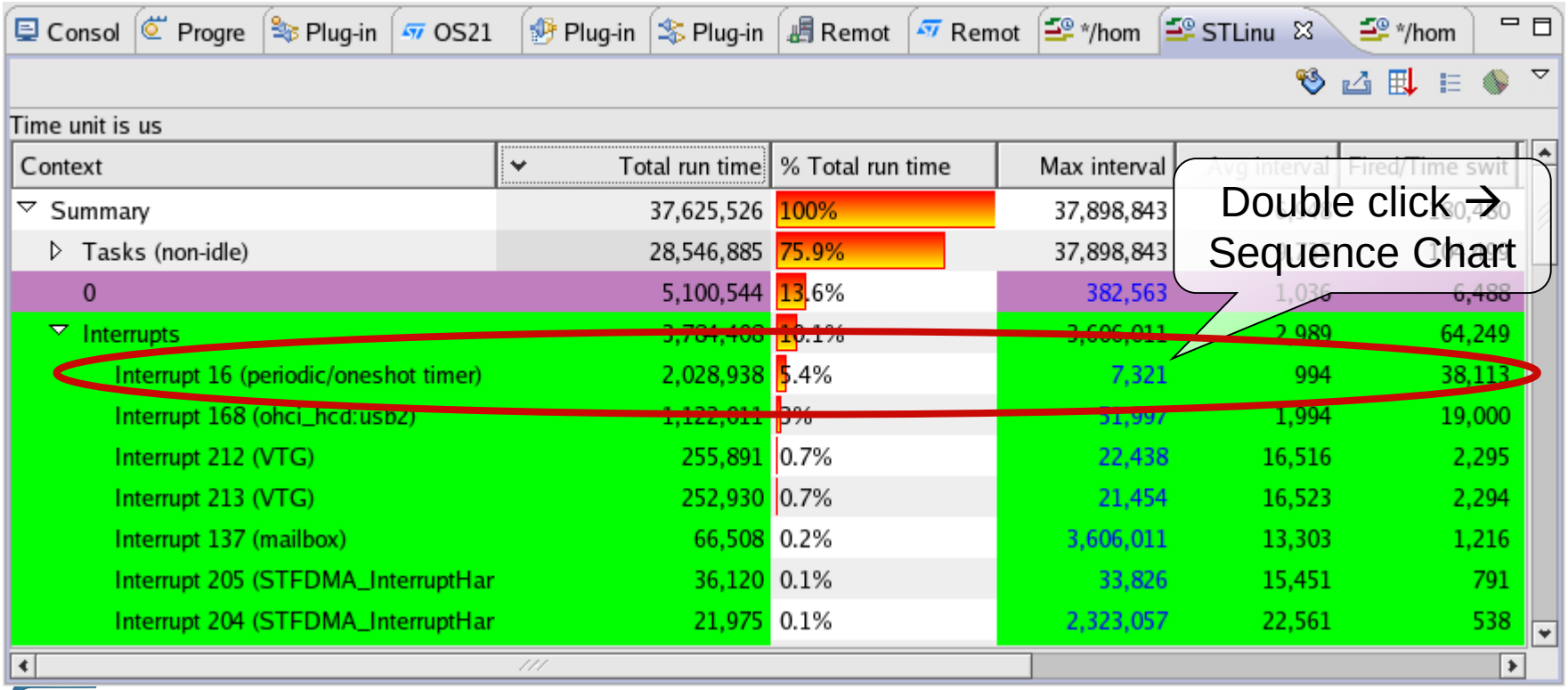
- Data collected using trace infra
  - Large amount of data available
  - Detailed execution info
- Focus on host side
  - Linux tasks and interrupts
  - Events related to them
- Post-mortem analysis
  - Graphical analysis with KPTrace
  - Different methods applied




life.augmented

# Use Case 1: Missing Frames

- Verifying execution statistics
  - *Interrupt 16* is a periodic signal (every 1ms)
  - Max interval recorded for *Interrupt 16*: 7.321ms => abnormal
  - Go to the observation view for further analysis

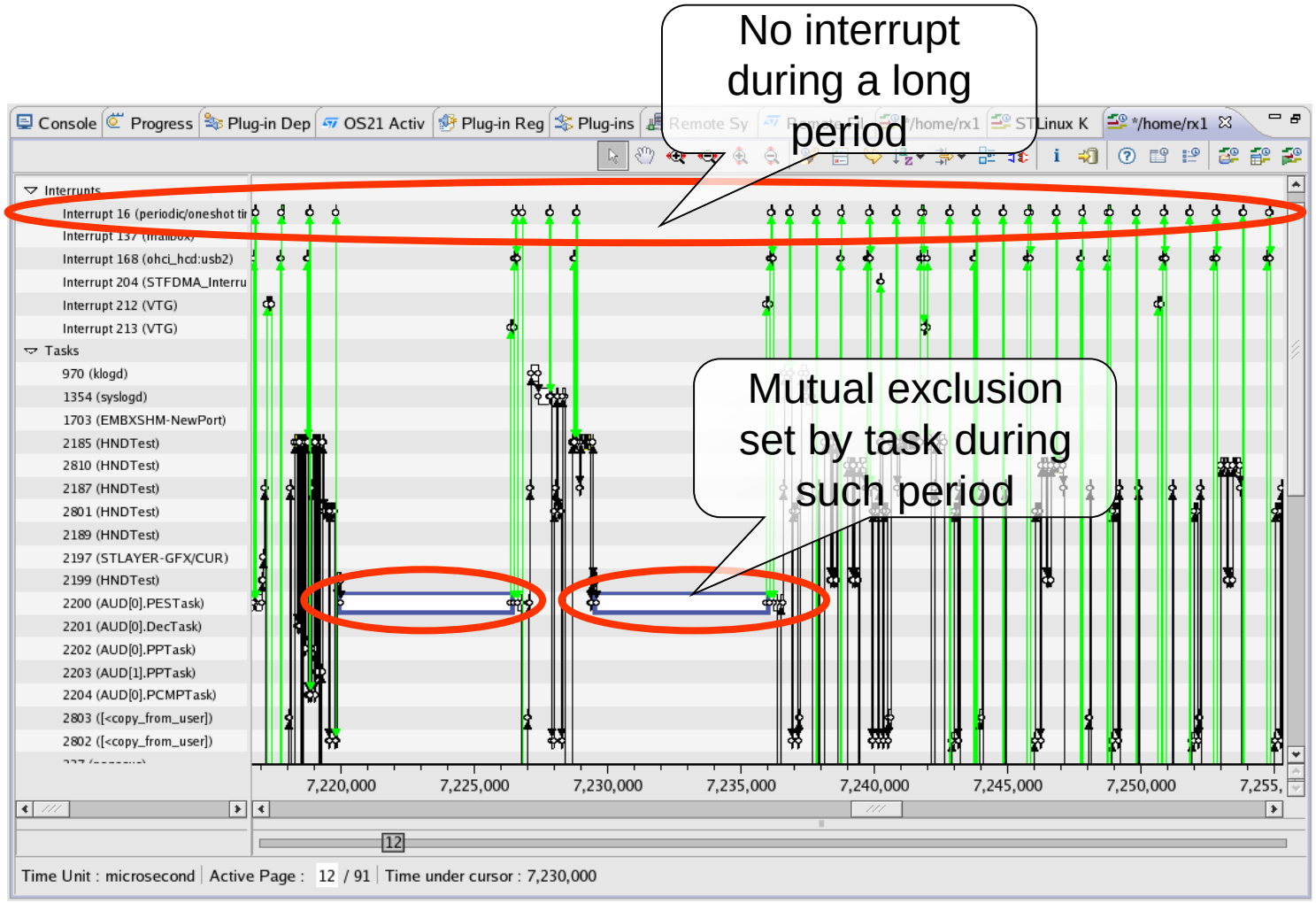



life.augmented

# Use Case 1: Missing Frames Digging into the Spotted Issue



- Looking for long delays for Interrupt 16…

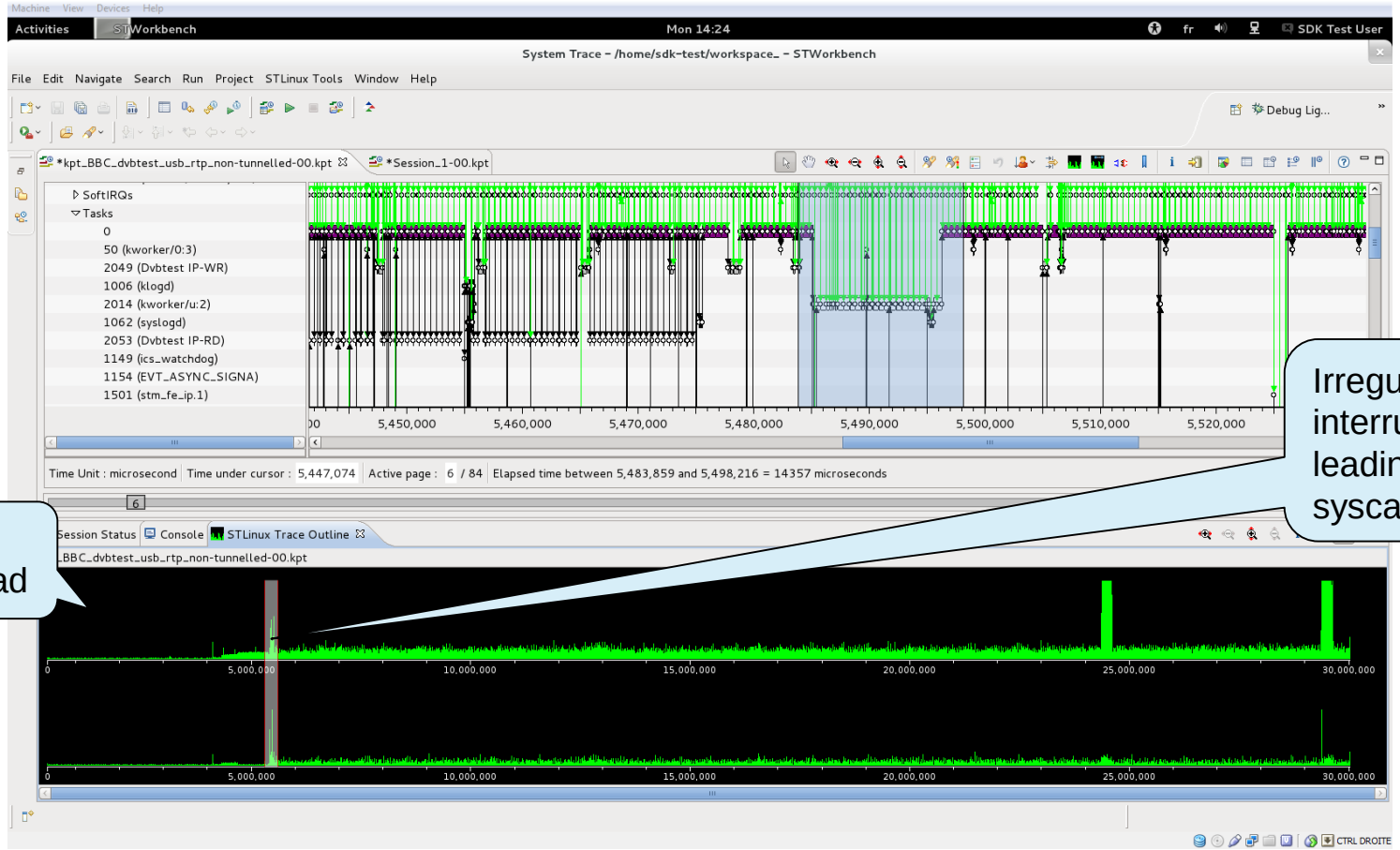


# Use Case 2: Incomplete Recording



# Use Case 3: Video Glitches

- This time we use an outline view to check irregular behaviours




life.augmented

# CONCLUSION

# Feedback from Experience



- System-level debugging involving multicore/multithreading computing
- Main focus are non-functional issues
- Very powerful debugging environment
- Large palette of tools
  - From simple ascii user traces to sophisticated graphical tools
  - Covering all software stack layers
  - Using many different mechanisms and tools
  - Orthogonal and even complementary with other debugging means
- Mastering the volume of traces is the main challenge
  - Data storage
  - Graphical visualization analysis techniques

# THANK YOU!