ZealCore Embedded Solutions

WE MAKE IT VISIBLE

ZealCore
Embedded Solutions

Accelerated Troubleshooting

Record Play

Henrik Thane, Ph.D.,
Founder
SVP Market Development

ZealCore Embedded Solutions

ZealCore Focus

Monitoring,
Debugging,
Testing of:

- *Embedded*
- *Multi-tasking*
- *Real-Time*
- *Multi-Core*
- *Distributed Systems…*

- *Used in*
  - *The Field / Deployed*
  - *The Lab / Pre-deployment*

About ZealCore

- Spin-off from the pioneering research work performed at the Mälardalen Real-Time Research Centre in Västerås, and the Royal Institute of Technology in Stockholm, Sweden
- Recognized by Gartner as Cool Vendor:

Gartner COOL VENDORS 2005 Emerging Trends and Technologies

- Award by MIS, 2005
  - "One of top 10 risings stars in the world".

Our Research and Technology

Tackles the difficulty of understanding the behavior of complex embedded systems

We provide solutions for

- Observability
- Reproducibility

Questions we try to answer

- What information is of interest?
- How to extract the information?
- How to analyze the information?
- How to manage the information?

Fact: The Inadequacy of Verification

Incorrect specifications cause over 50% of all failures

[N. G. Leveson. Safeware System, Safety and Computers. Addison Wesley 1995.]

'The radar system of HMS Sheffield identified an incoming Exocet missile as non-Soviet and thus friendly. No alarm sounded. The Ship sunk with substantial losses of life.'

[ACM Software Engineering Notes , Vol.8, No. 3.]

- All wrong despite V&V

We need means to see what happens out there…

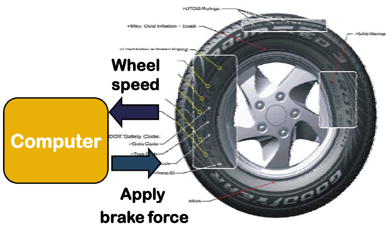
Fact: Debugging

Difficult to understand the behavior of complex embedded systems

- Difficult to observe behavior
- Difficult to reproduce behavior

ZealCore Embedded Solutions

Fact: Debugging



Software controlled ABS

Fault Identification

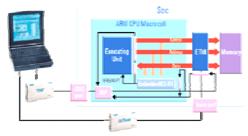
(Debugging)

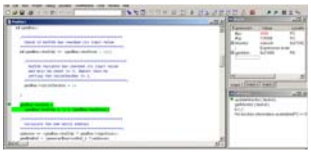
- Non Stop
  - System keeps running even if software stops
- Indeterminism
  - Rerunning a program gives different results
  - Hard to reproduce failures
- Probe-effect (Heisenbugs)
  - Instrumentation to record behavior changes behavior
  - Bugs hide when we are looking for them

ZealCore Embedded Solutions

Common Solutions

ZealCore Embedded Solutions — Observation & Reproducibility Techniques



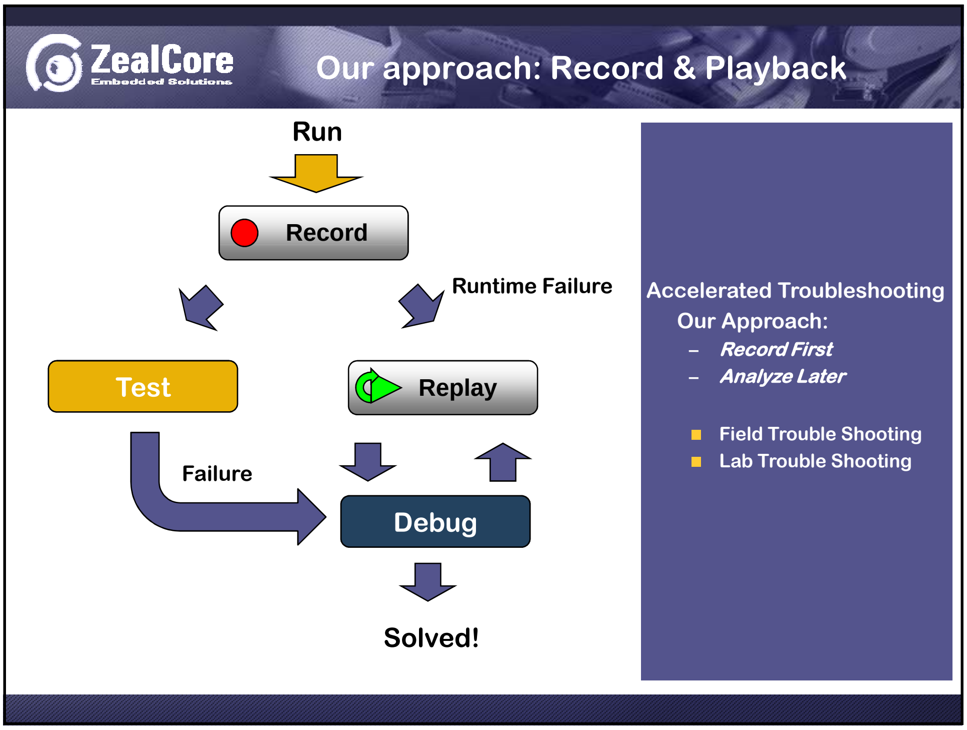


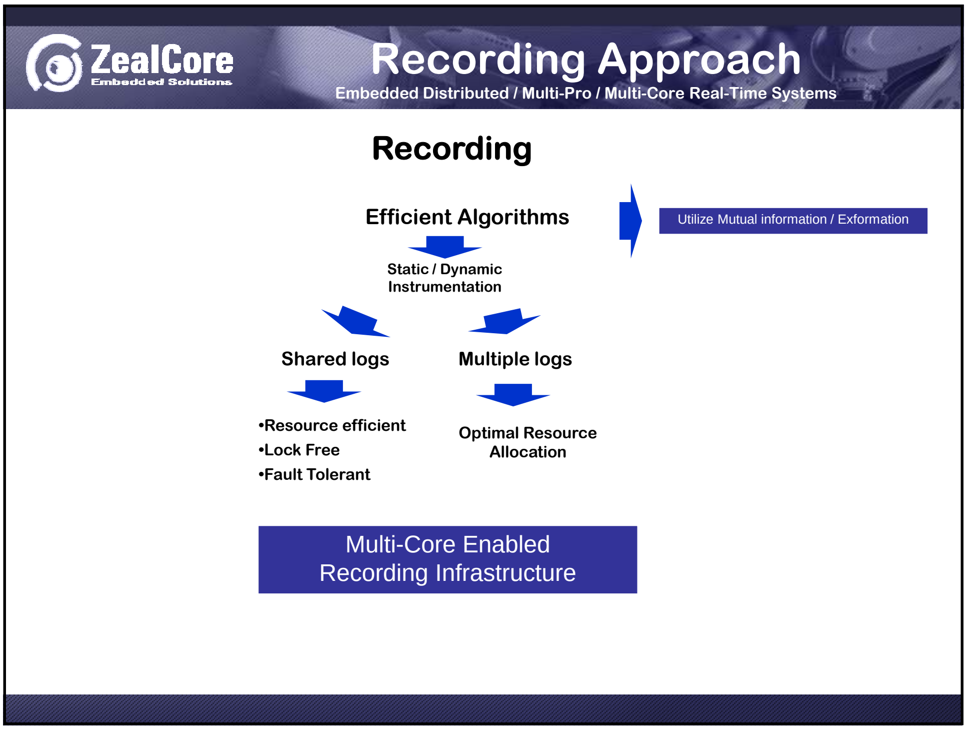
TRACE_1("ROUTE", "gps_ROUTE_R4_Link")

- Hardware features
  - In-circuit emulators, BDM, ETM, JTAG ports, logic analyzers
  - Usually do not scale to deployed systems. Expensive to "Insure" target with hardware recorders.
- Using a symbolic source code debugger
  - Watches
  - Problem: Non stop debugging
  - Problem: Indeterminism
    - How to reproduce inputs from the field
  - Problem: Probe-effects
- Virtualization/simulators
  - Non intrusive
  - Problem: typically only virtualization of computer HW, not environment, e.g., Dynamics of a Robot
  - No input from deployed systems.
  - In some cases lock step simulation – but 50% of failures come from incorrect specifications.
- Software instrumentation
  - The most common means by which the programmer can increase the observability and reproducibility
  - auxiliary outputs, typically in terms of classic "printf()" statements and storage in logs.

ZealCore Embedded Solutions — Software Instrumentation Techniques

- For Languages with Pre Processors (C/C++) or Using Aspects
  - Textual substitution by parser
  - Textual substitution with macros of the C-preprocessor
- Inclusion/replacement additional library
- Trace functions/methods
  - Log4J (logging package for Java)
- Binary wrapping of object code
- Dynamic instrumentation of executable/executing code
  - ATOM,EEL Etch (Need restart)
  - DynInst (no restart, no real-time considerations)
  - Dynamic Patch/Relink (no restart, real-time considerations)
  - WindRiver Sensorpoints

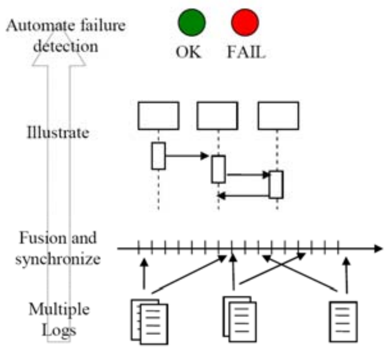

ZealCore
Embedded Solutions
Our approach: Record & Playback
Run
Record
Runtime Failure
Test
Replay
Failure
Debug
Solved!
Accelerated Troubleshooting
Our Approach:
– Record First
– Analyze Later
Field Trouble Shooting
Lab Trouble Shooting


ZealCore
Embedded Solutions
Recording Approach
Embedded Distributed / Multi-Pro / Multi-Core Real-Time Systems
Recording
Efficient Algorithms
Utilize Mutual information / Exformation
Static / Dynamic Instrumentation
Shared logs
Multiple logs
•Resource efficient
•Lock Free
•Fault Tolerant
Optimal Resource Allocation
Multi-Core Enabled Recording Infrastructure

System Debugging

ZealCore Embedded Solutions

System Debugging =
Software Monitoring + System Event Processing



A necessary evolution Of traditional debugging methods
- To find and understand complex bugs
- Need to fuse & synchronize logs
- Need to automate the process
- Need to raise the abstraction level

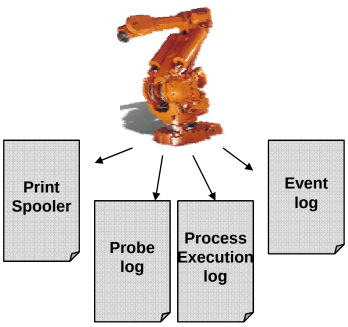
Computer aided bug analysis
- Automatic import of several logs and synchronization
- Use graphics/abstractions
- Use reverse engineering
- Automatic fault identification

The Problem of System Debugging

ZealCore Embedded Solutions

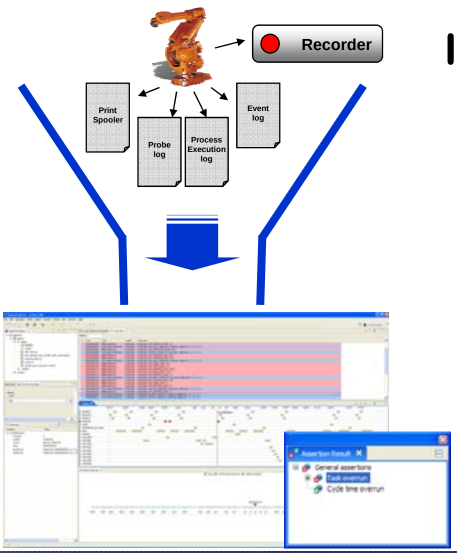
Current Industrial Situation

Poor Trouble Shooting



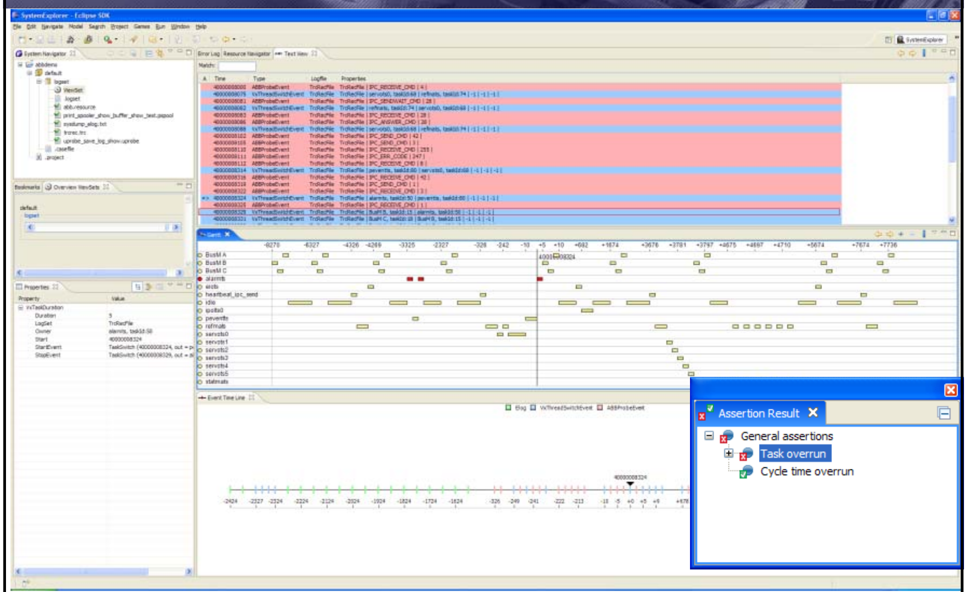
- General
  - Non Stop Behavior
  - Non Determinism
- Many logs/recordings
  - Different formats
  - Difficult to read and understand
  - No synchronization

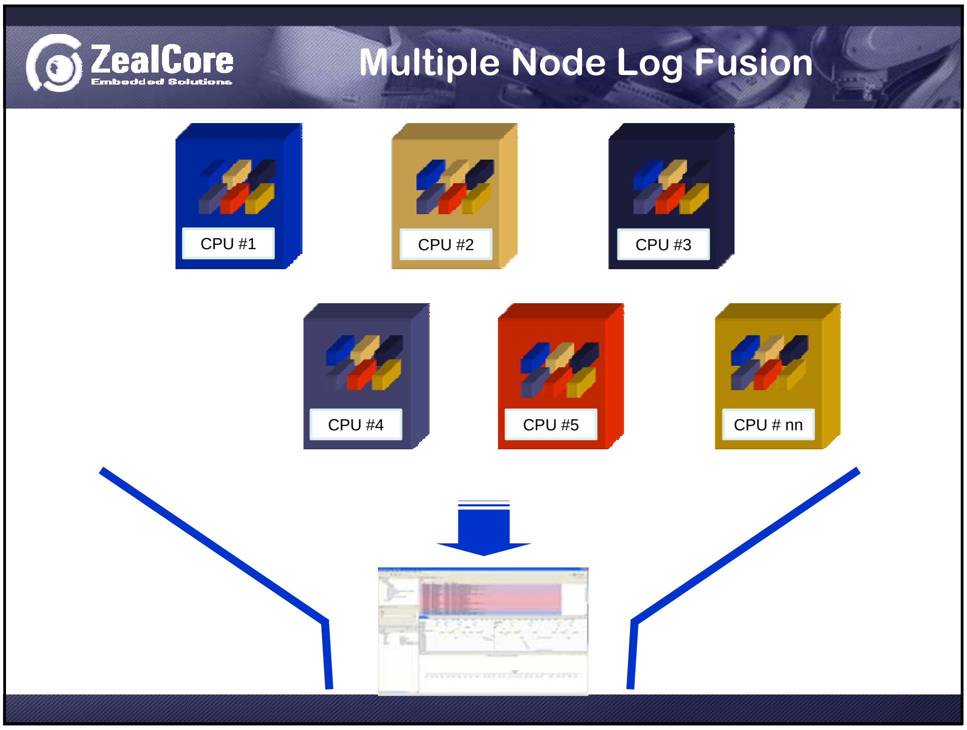
Our System Debugger Approach

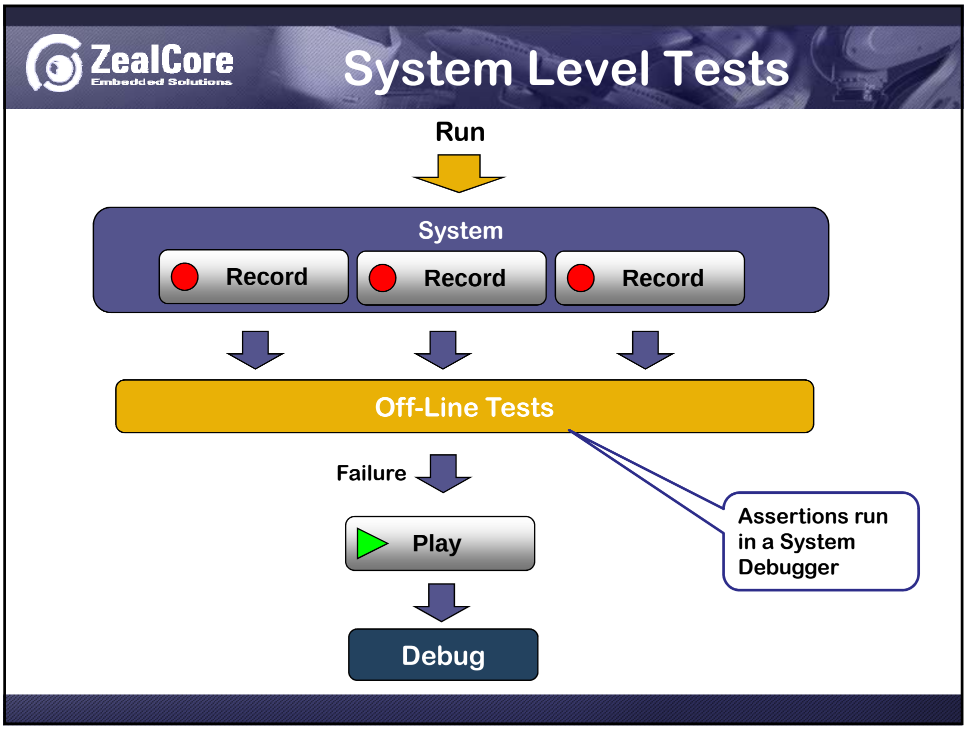


Information Funnel

- System Level Debugger
  - Import and synchronization of Logs
  - Correlation of Data
- Automatic Fault Identification
- Automatic Trend Analysis
- Visualization & Reanimation
- Works on-top of existing tools
- Eclipse Based




ZealCore
Embedded Solutions
Multiple Node Log Fusion
CPU #1
CPU #2
CPU #3
CPU #4
CPU #5
CPU # nn


ZealCore
Embedded Solutions
System Level Tests
Run
System
Record
Record
Record
Off-Line Tests
Failure
Play
Debug
Assertions run in a System Debugger

ZealCore
Embedded Solutions

Application Example

ZealCore
Embedded Solutions

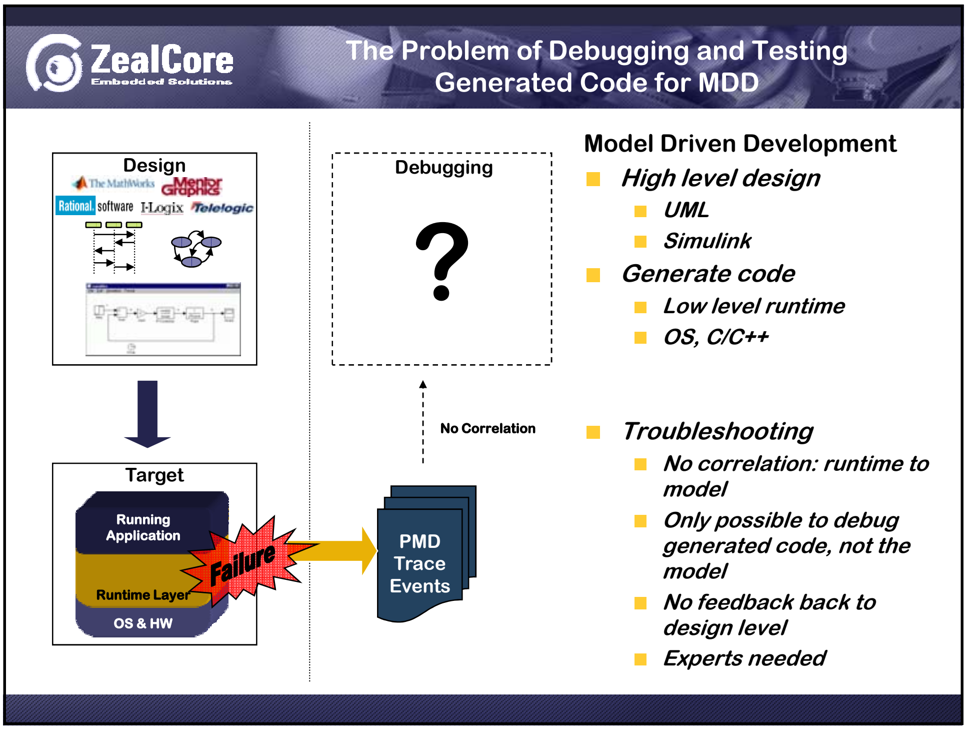
WE MAKE
IT VISIBLE

Approach # 1

Replay Debugging and Testing for Automatic Code Generation

Record

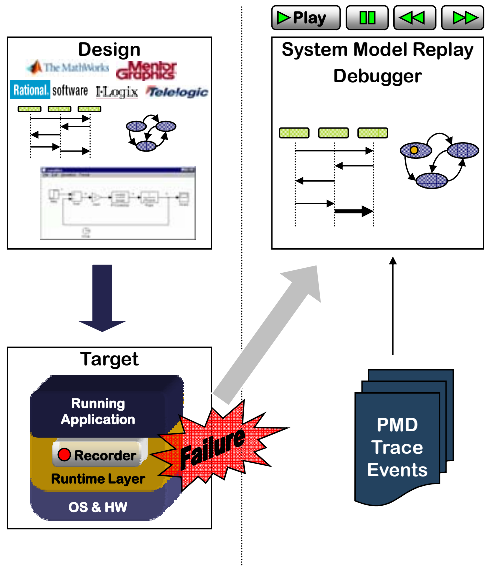
Play


ZealCore
Embedded Solutions
The Problem of Debugging and Testing Generated Code for MDD
Design
Rational. software
I-Logix
Telelogic
Target
Running Application
Runtime Layer
OS & HW
Failure
Debugging
?
No Correlation
PMD Trace Events
Model Driven Development
High level design
UML
Simulink
Generate code
Low level runtime
OS, C/C++
Troubleshooting
No correlation: runtime to model
Only possible to debug generated code, not the model
No feedback back to design level
Experts needed


ZealCore
Embedded Solutions
WE MAKE IT VISIBLE
The Solution
Record
Play

ZealCore Embedded Solutions

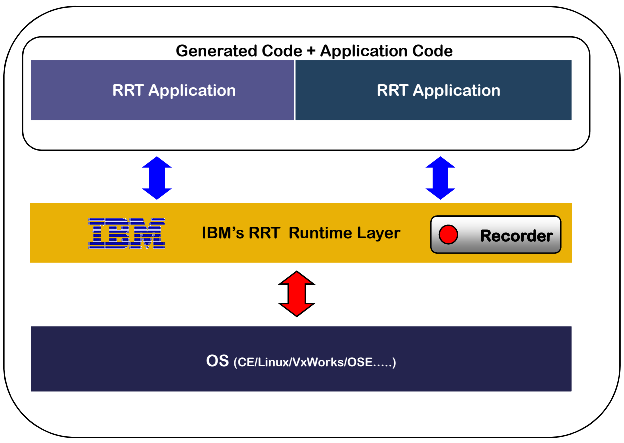
ZealCore's Replay Debugger



- ***Use software recorder***
- ***Fuse data***
- ***"Do reverse engineering"***
- ***Replay***
  - *State machines, signaling and states*
  - *Animate sequence- and state diagrams*
  - ***Exact representation of runtime behavior***
- ***Correlate***
  - *Model + OS-events*
  - *Program modules*
  - *Processors*
  - *Native logs*
- ***Automatic fault identification***

ZealCore Embedded Solutions

Rational Rose Real-Time (RRT) & ZealCore System Recorder Integration



Recorder

- **Only Added to the Rose Real-Time Runtime Library**
- **No Modification of the Application**
- **Footprint is so Small that recorder can be Enabled even in Deployed Systems**
  - PowerPC 705 (700MHz)
  - Heavy application 80-100% CPU Load
  - ~ 550 k Events/s (40Byte)
  - Recording cost: ~ +1 % CPU Load
  - Code footprint: ~ 2K lines C code

ZealCore Embedded Solutions

Other Integrations

Mentor Graphics

I-Logix Telelogic

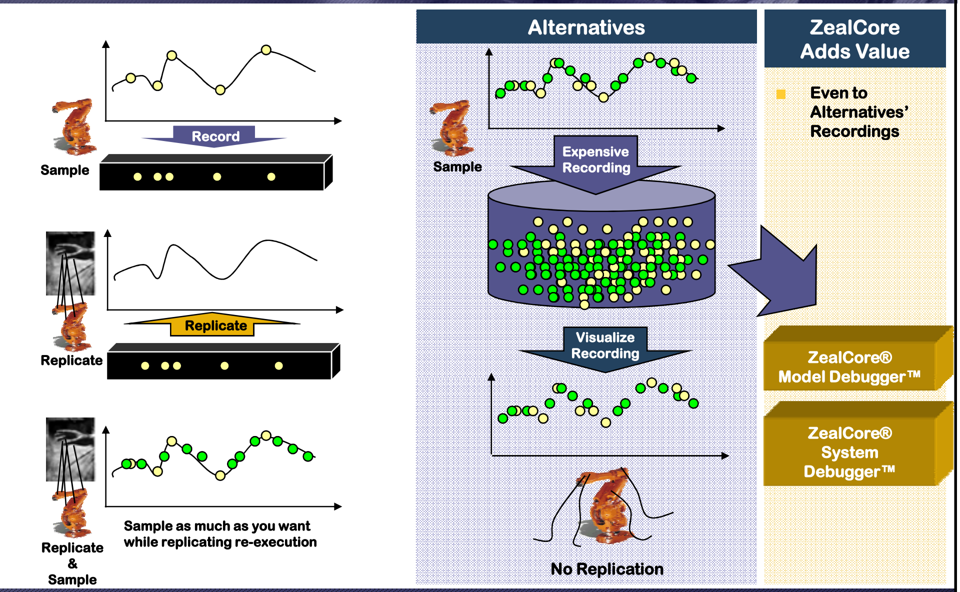
The MathWorks dSPACE

ENEA WIND RIVER

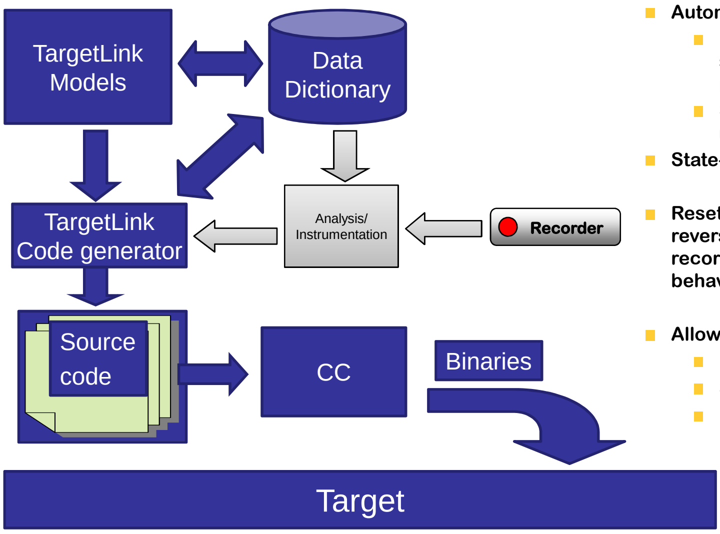
Ready for IBM. Rational. software

ZealCore Embedded Solutions

A Step Further: Replication



Simulink/TargetLink ZealCore System Recorder Integration



- Automatic minimal set
  - Identification of stimuli and persistent state
  - Source code modification
- State-of-the-art recorder
- Resetting state and reversing stimuli from recording replays system behavior
- Allows replay on
  - Binary level
  - Source code level
  - Model level

Summary

- There is an increasing need for debugging methods that can handle system level faults
  - The ever increasing complexity of software
    - Abstraction & computer aided analysis
  - Need to handle
    - Observability
    - Reproducibility
  - Equally important
    - Field debugging
    - Lab debugging

Challenges

The ever increasing complexity of software in terms of size but also concurrency and timeliness:

- How to achieve confirmation (regression) testing on the system level (i.e., taking concurrency and real-time aspects into account)
- How to monitor software using technologies that scale to deployed systems, i.e., large populations of systems with minimal and predictable probe-effects (new systems as well as legacy)
- How to increase the use of the collected/monitored information such that it can be understood by people or automatically analyzed by computers for debugging, or testing.
- How to manage the collected information over time
  - How to make trend or regression analysis based on historical data.
- How to achieve reproducible debugging environments (problem related to confirmation testing)

- Equally important
  - Field use
  - Lab use

THANK YOU!

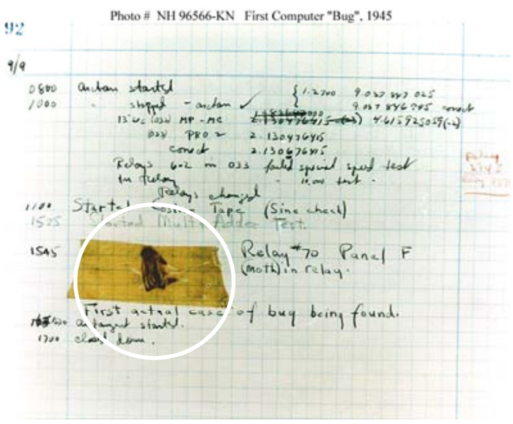


Figure 1. Allegedly the first computer bug - found by Grace Hopper's Team in 1945. Exhibited at the Museum History of American Technology/Smithsonian

henrik.thane@zealcore.com

www.zealcore.com