Wind River Device Management

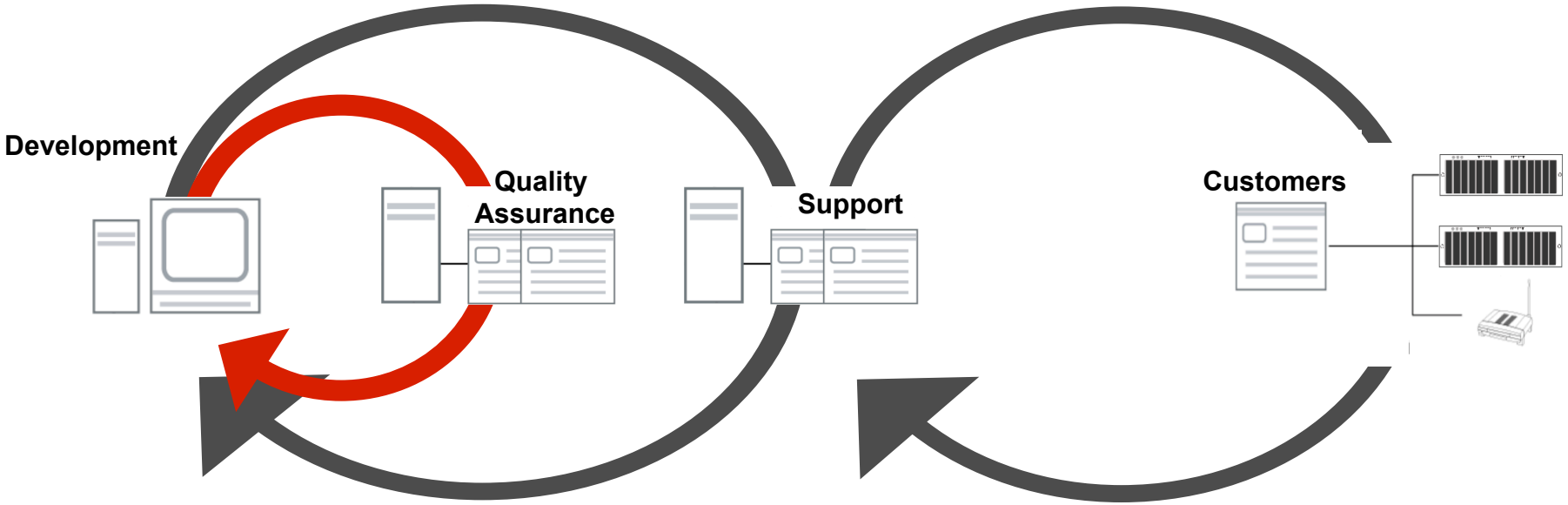
Quality, TTM and Uptime Challenges

Development	SW Quality Assurance			Deployment		
Implementation	**Software Integration Testing**	**Verification Testing**	**Validation Testing**	**Field Trial**	**Support**	**Maintenance**

Wind River Device Management

Lab Diagnostics

Field Diagnostics



Streamlines development and QA processes to deliver higher quality devices to market faster

Streamlines the support process to increase device uptime and to increase device-user satisfaction

WIND RIVER

Wind River Lab Diagnostics

Software QA Best Practices
for Device Software

Engineering Challenges

Development

SQA

Engineering Challenges

- Development engineering must write more software
- Must increase performance
- Must increase system quality
- Must deploy products faster
- Has fixed resources

Software QA Challenges

- SQA engineering must reduce MTTR
- Must increase code coverage, SW performance and SW stress test capabilities
- Must test more software
- Has fixed resources

Quality and TTM Challenges

Development	Software Quality Assurance		
Implementation	**Software Integration Testing**	**Software Verification Testing**	**Product Validation Testing**

Quality Assurance Phase: Testing and problem resolution consume 40-50% of a typical development schedule

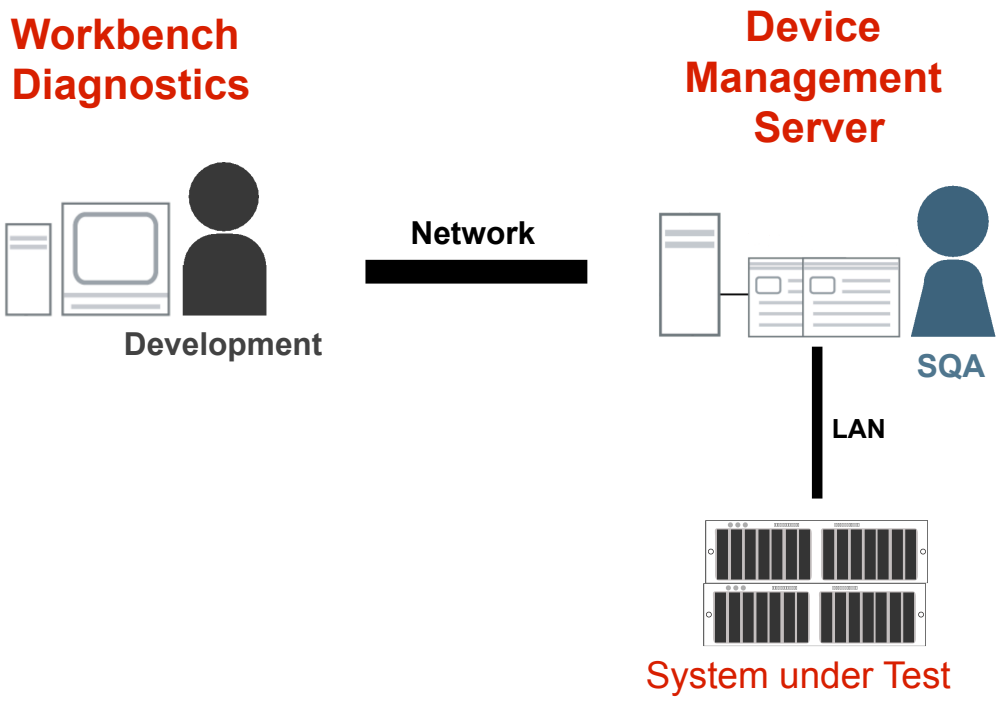
Software Integration: Prolonged software integration delays the schedule

Software Verification Testing: Incomplete white-box testing causes downtime in deployed products

Product Validation Testing: Software bugs encountered late in the project delay product release

Wind River Lab Diagnostics

A scalable, distributed software diagnostics system that enables development and test engineers to perform real-time tests and resolve bugs during software integration, software verification and product validation



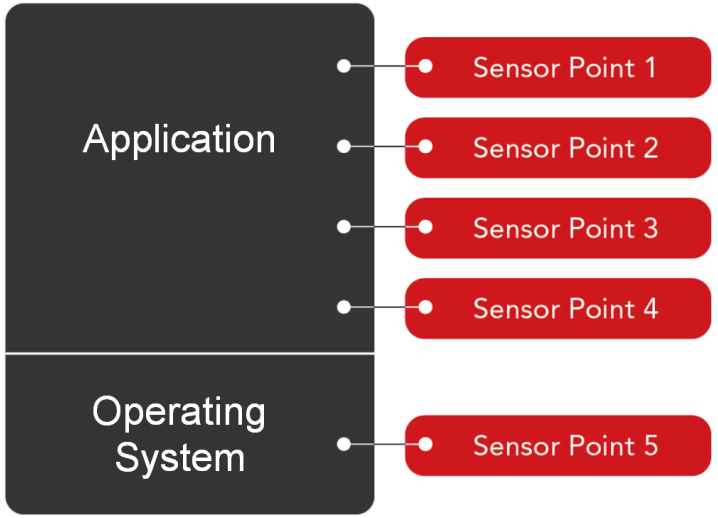
Streamlines Development and QA processes

- Enables testable device software
- Enables real-time testing of running software on live systems
  - Software API verification
  - Performance testing
  - Execution path coverage
  - Fault injection to characterize the response of running applications
- Facilitates collaboration between development and test engineers
- Shortens time to fix bugs

Sensorpoint Technology

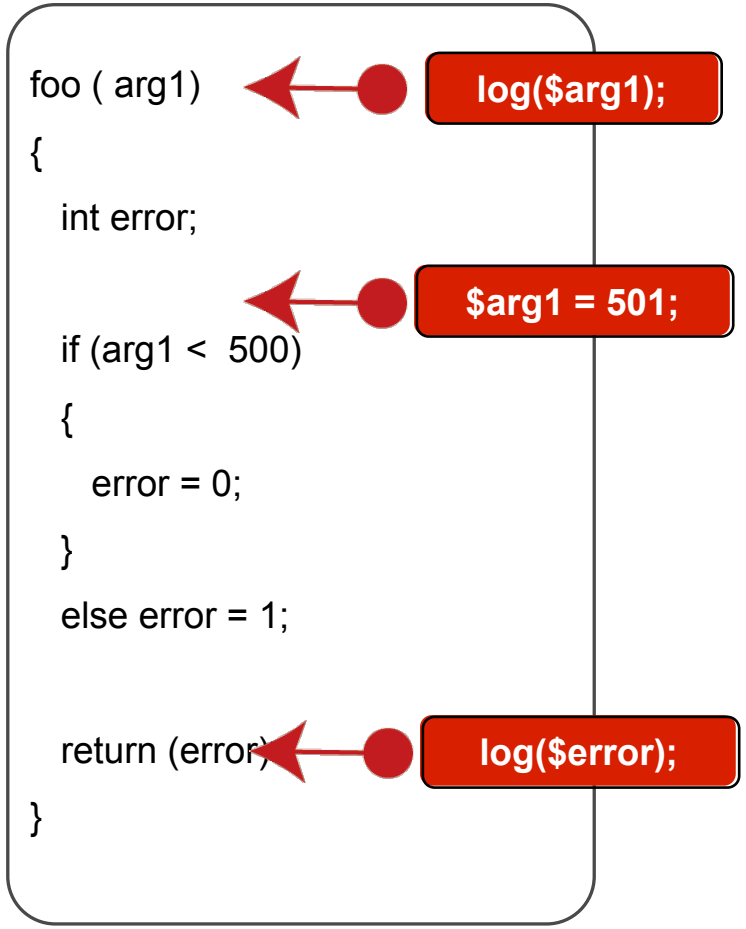
- **Dynamic instrumentation of functions or methods of running applications**
  - **No application modification, recompile, reloading or rebooting needed**
  - **Minimally intrusive**
  - **Run-time agent has a small footprint**
- **Enables comprehensive white-box testing of running software**

```
sensorpoint thread
{
  sensorpoint “foo.c”:”foo1()”
  {
    on_entry()
    {
      log($arg1);
    }
  }func_foo1;
}
```



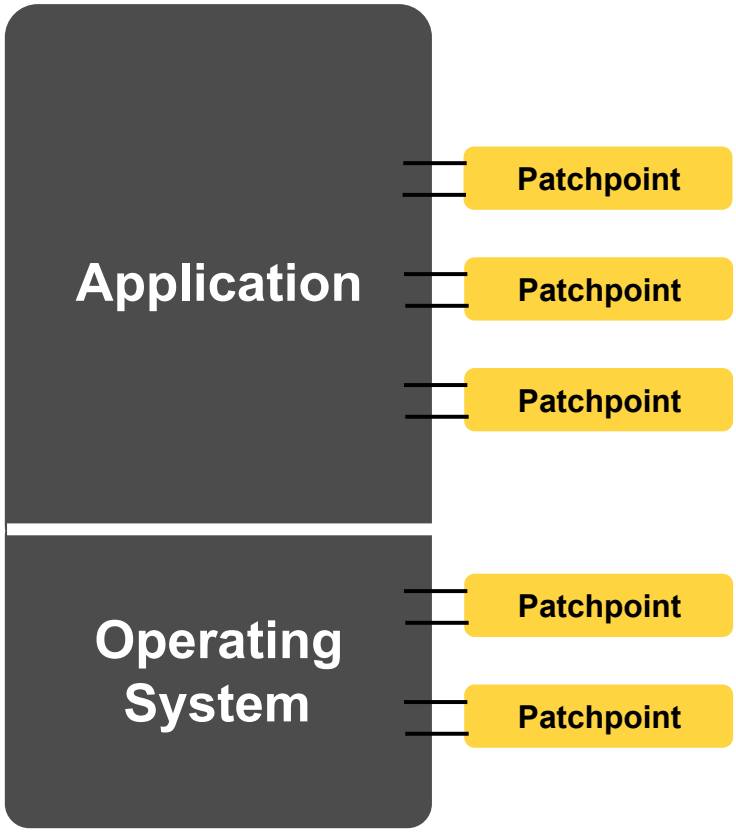
Function-level Instrumentation using Sensorpoints

- Dynamically patch a function or method:
  - on_entry
  - on_line
  - on_offset
  - on_exit
- Access variables within the scope of functions
  - Log and change variable value

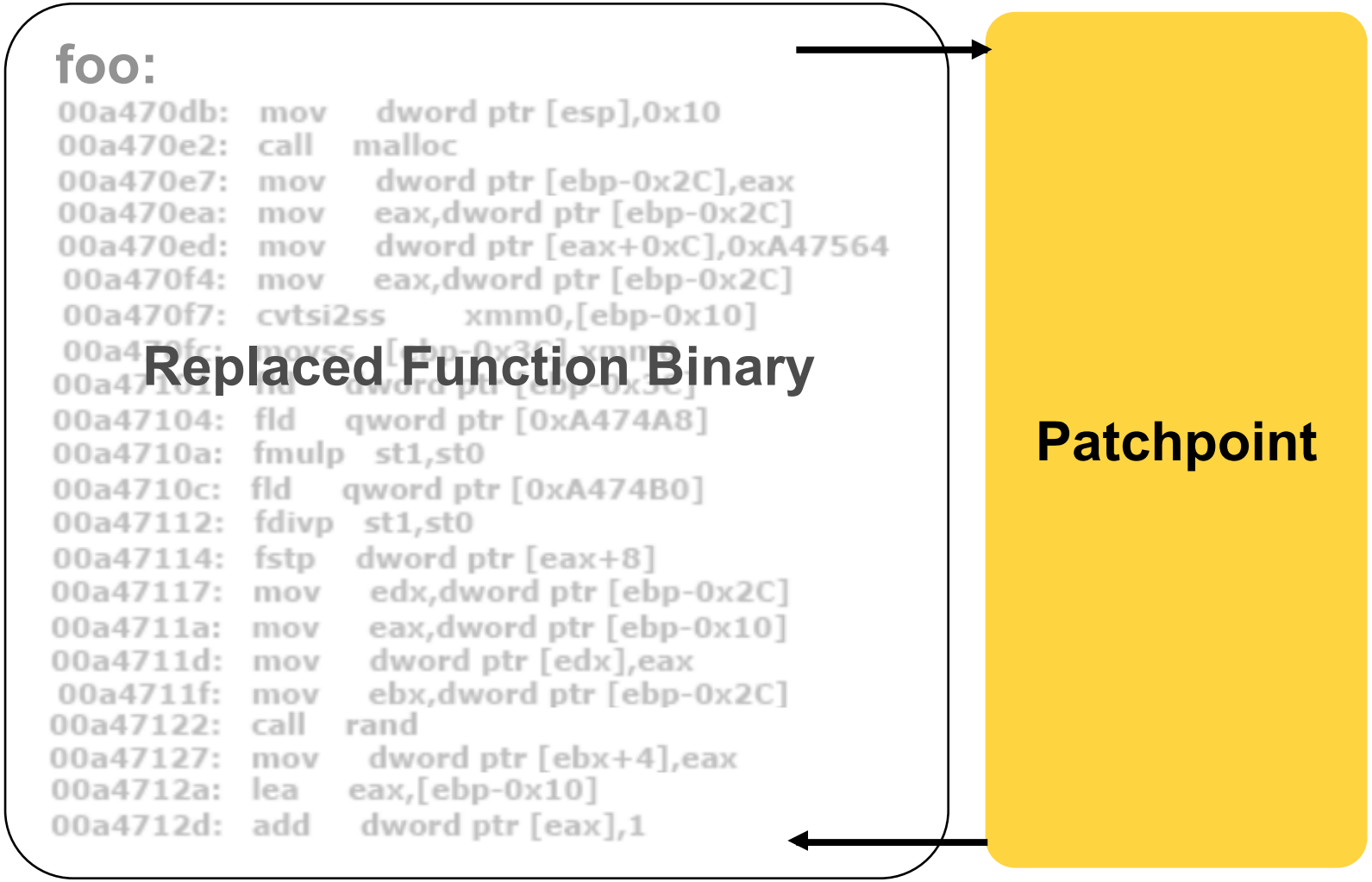


Patchpoint Technology

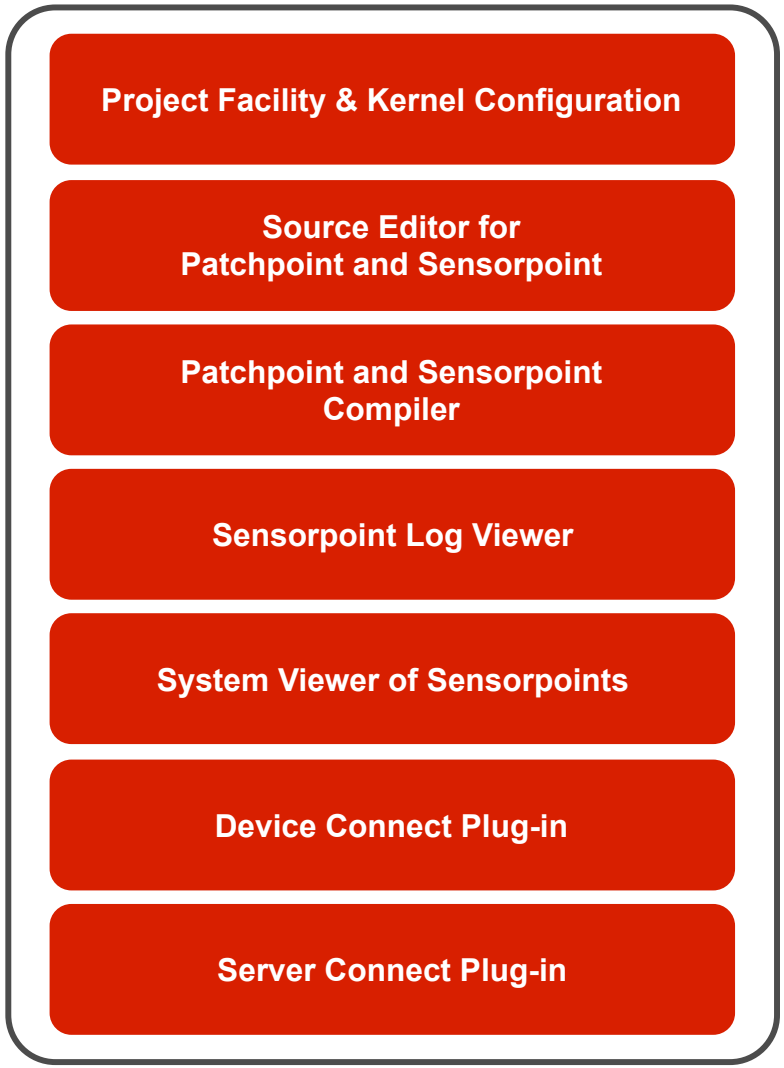
- Dynamic replacement of functions or methods in running applications
  - Guarantees image integrity
  - Apply corrective code without application re-start or device re-boot
  - Minimally intrusive
- Complementary to firmware updates and dynamically downloadable binaries



Function Replacement with Patchpoint

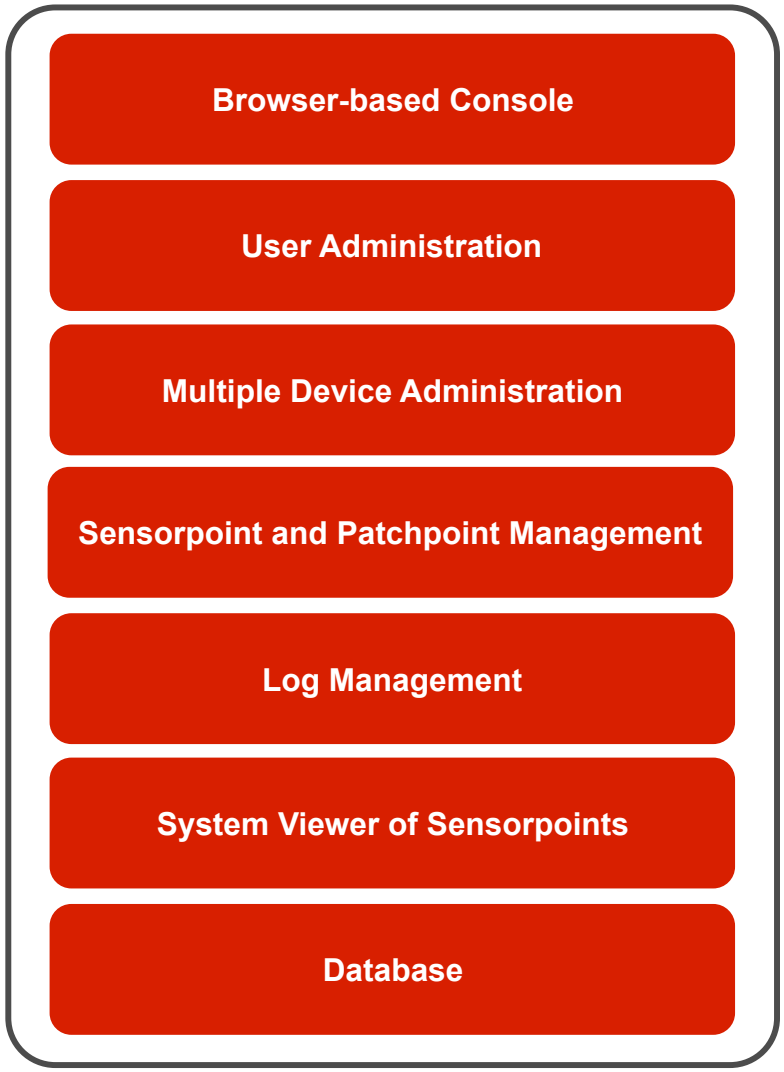


Workbench Diagnostics



- Workbench IDE plug-ins
- Development engineers can:
  - Design-in testability
  - Create software test harnesses with Sensorpoints
  - Analyze test and fault data
  - Resolve bugs with Patchpoints

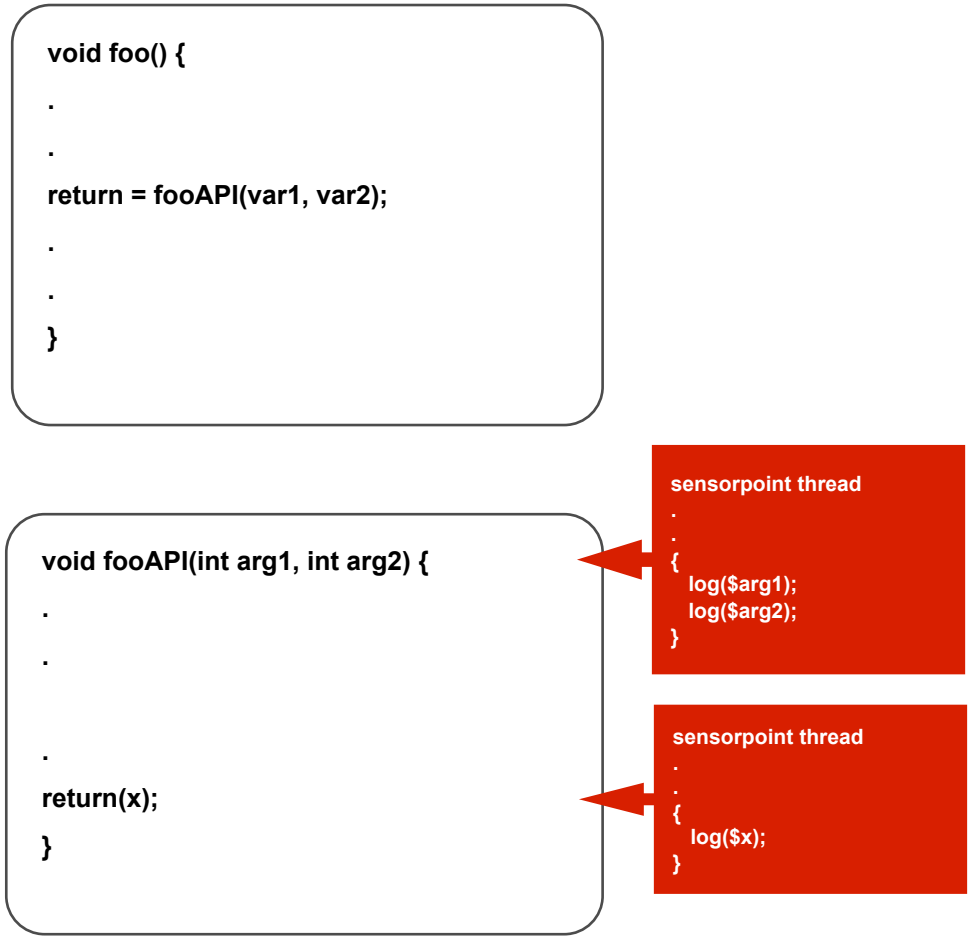
Device Management Server



- J2EE server application for real-time software testing
  - Browser-based, multi-user console
- Engineers can:
  - Manage multiple devices under tests
  - Deploy Sensorpoints to execute white-box tests
  - Collect test and fault data
  - Analyze test and fault data
  - Deploy Patchpoint to resolve bugs

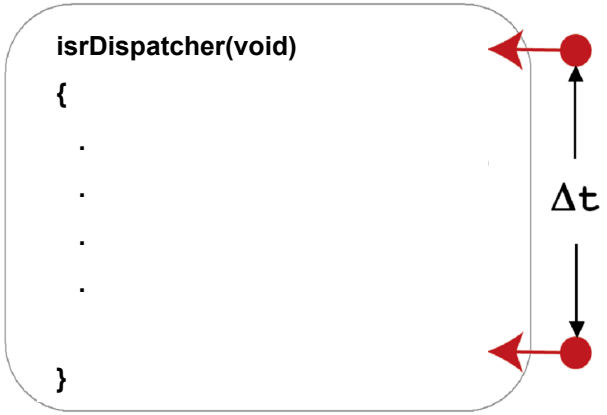
Use Cases

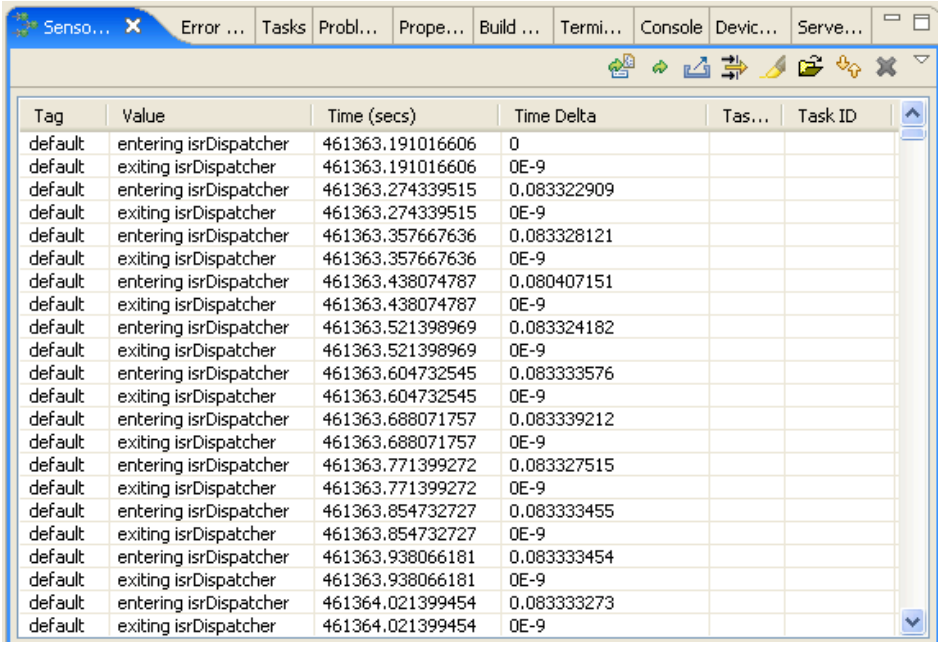
Use Case: Interface Verification



- Trace APIs on running applications with Sensorpoints
- Collect API arguments and return values per iteration
- Resolve interface issues

Use Case: Performance Test





Senso... | Error ... | Tasks | Probl... | Prope... | Build ... | Termi... | Console | Devic... | Serve...

Tag	Value	Time (secs)	Time Delta	Tas...	Task ID
default	entering isrDispatcher	461363.191016606	0		
default	exiting isrDispatcher	461363.191016606	0E-9		
default	entering isrDispatcher	461363.274339515	0.083322909		
default	exiting isrDispatcher	461363.274339515	0E-9		
default	entering isrDispatcher	461363.357667636	0.083328121		
default	exiting isrDispatcher	461363.357667636	0E-9		
default	entering isrDispatcher	461363.438074787	0.080407151		
default	exiting isrDispatcher	461363.438074787	0E-9		
default	entering isrDispatcher	461363.521398969	0.083324182		
default	exiting isrDispatcher	461363.521398969	0E-9		
default	entering isrDispatcher	461363.604732545	0.083333576		
default	exiting isrDispatcher	461363.604732545	0E-9		
default	entering isrDispatcher	461363.688071757	0.083339212		
default	exiting isrDispatcher	461363.688071757	0E-9		
default	entering isrDispatcher	461363.771399272	0.083327515		
default	exiting isrDispatcher	461363.771399272	0E-9		
default	entering isrDispatcher	461363.854732727	0.083333455		
default	exiting isrDispatcher	461363.854732727	0E-9		
default	entering isrDispatcher	461363.938066181	0.083333454		
default	exiting isrDispatcher	461363.938066181	0E-9		
default	entering isrDispatcher	461364.021399454	0.083333273		
default	exiting isrDispatcher	461364.021399454	0E-9		

- Real-time performance testing on “live” systems with Sensorpoints
- Measure execution time at
  - Function level
  - Sub-system level
- Isolate performance bottlenecks
- Tune software for performance

Use Case: Fault Injection & Execution Path Coverage

```
foo()
{
  .
  .
  return = messageSend(*message);
  switch(return)
  {
    case RESEND: messageResend(*message);
    case ERROR1: alarm(ERROR1);
    case ERROR2: alert(ERROR2);
    case ERROR3: linkReconnect();
    default: alarm();
  }
  .
  .
}
```

```
messageSend()
{
  .
  .
  .
  return(rValue);
}
```

```
sensorpoint thread
.
.
{
  int i=0;

  switch(i)
  {
    case 1: $rValue=ERROR1;
    case 2: $rValue=ERROR2;
    case 3: $rValue=ERROR3;
    case 4: $rValue=UNEXPECT;
    default: alarm();
  }
}
```

- Inject faults and control execution of code with Sensorpoints
- Test more running code
  - Execute uncommon execution paths or branches
  - Test error handlers
  - Test robustness and stability of products
- Increase product quality by
  - Fully characterizing fault response of applications
  - Increasing test coverage of executing binaries

Results with Lab Diagnostics

- **Faster to Market**
  - Software Integration: shorten time to stable, integrated software
  - Software Verification: shorten time to test coverage
  - Product Validation: shorten time to problem resolution
- **Higher Quality**
  - Functional: trace execution of device software
  - Reliable: fault inject and execution path coverage
  - High Performance: comprehensive timing information

Device Management Breakout

- 13:00 - 13:50: Workbench Diagnostics
- 15:45 – 16:35: Mapping Wind River Field Diagnostics to your customer Support Process

WIND RIVER