EASEly extending Eclipse Trace Compass

Bernd Hufmann, Ericsson AB
Geneviève Bastien, Polytechnique Montréal

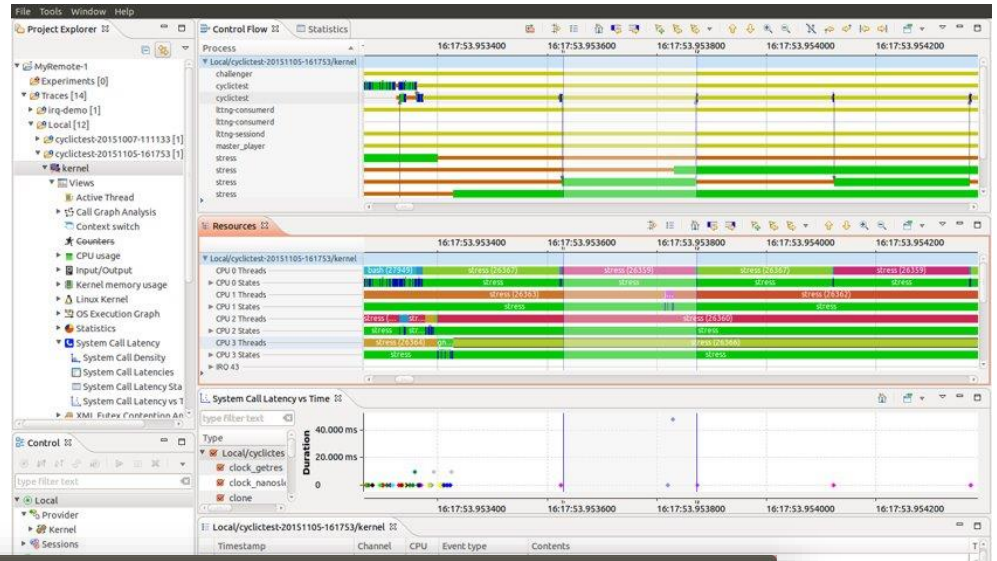
2019-10-23

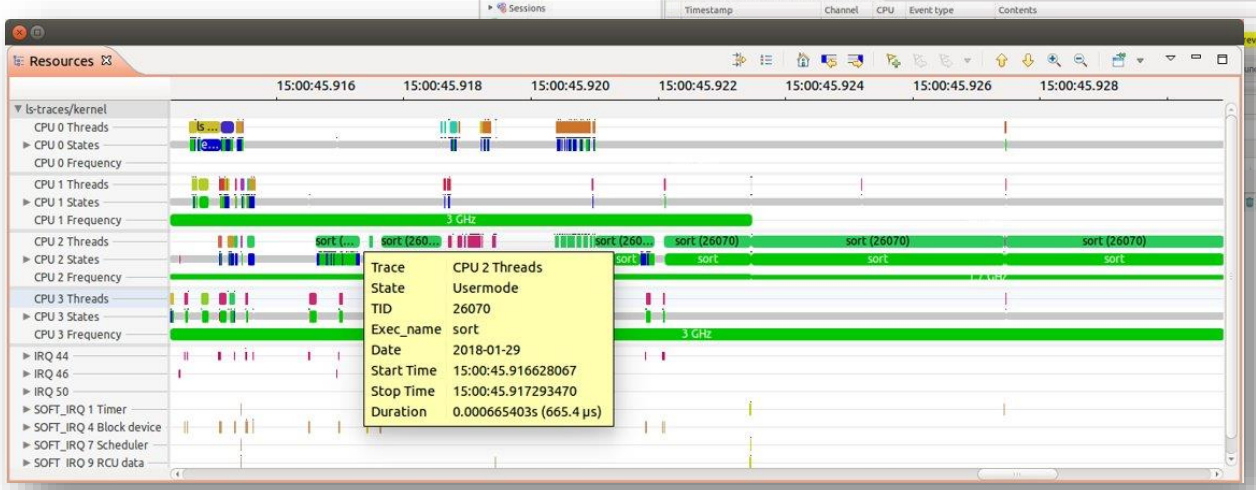
Agenda

- Background
- Trace Compass and its extensibility
- Trace Compass and EASE
- Demo
- Q&A

What is tracing?

- Trace
  - Series of events over time
  - Event collected at tracepoints during program execution
  - Each event has a type and payload
- Use the events as input for analysis
- Create visualization graphs with these analysis
- Tracing use cases
  - Profile application
  - Find long executions
  - Investigate real-time deadlines
  - Find memory or load issues
  - Investigate concurrency problems



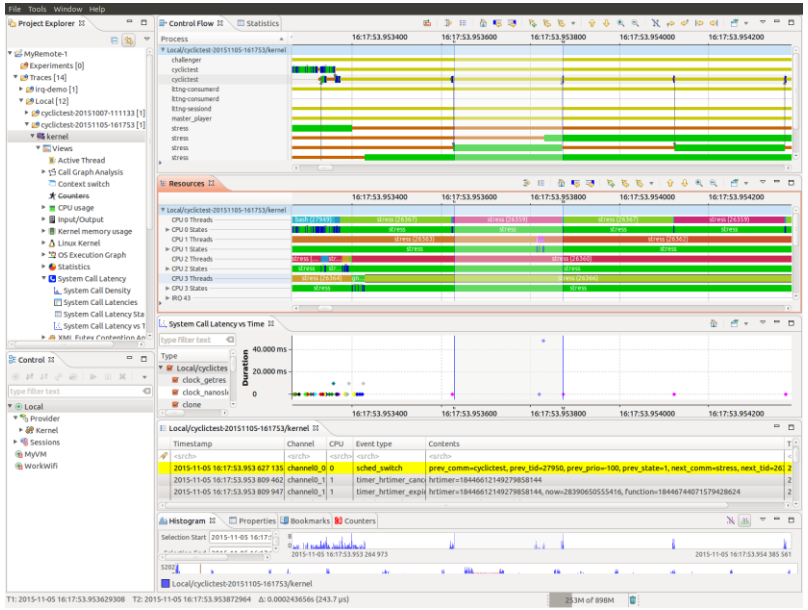


Eclipse Trace Compass is an open source application to solve performance and reliability issues by reading and analyzing traces and logs of a system.

Its goal is to provide views, graphs, metrics, and more to help extract useful information from traces, in a way that is more user-friendly and informative than huge text dumps.

Key characteristics

- Handles trace larger than available memory
- Correlates traces from heterogenous system



Why is extensibility important for us?

Why is extensibility important for us?

"One of these things is not like the others, one of these things is not the same" (Sesame Street)

Application *One*	≠	Application *Two*
Trace format *A*	≠	Trace format *B*
Analysis *Foo*	≠	Analysis *Bar*
Graph Baz	≠	Graph *Qux*
Problem *X*	≠	Problem *Y*

Why is extensibility important for us?

"One of these things is not like the others, one of these things is not the same" (Sesame Street)

Application *One*	≠	Application *Two*
Trace format *A*	≠	Trace format *B*
Analysis *Foo*	≠	Analysis *Bar*
Graph Baz	≠	Graph *Qux*
Problem *X*	≠	Problem *Y*

But many re-usable concepts!

Why is extensibility important for us?

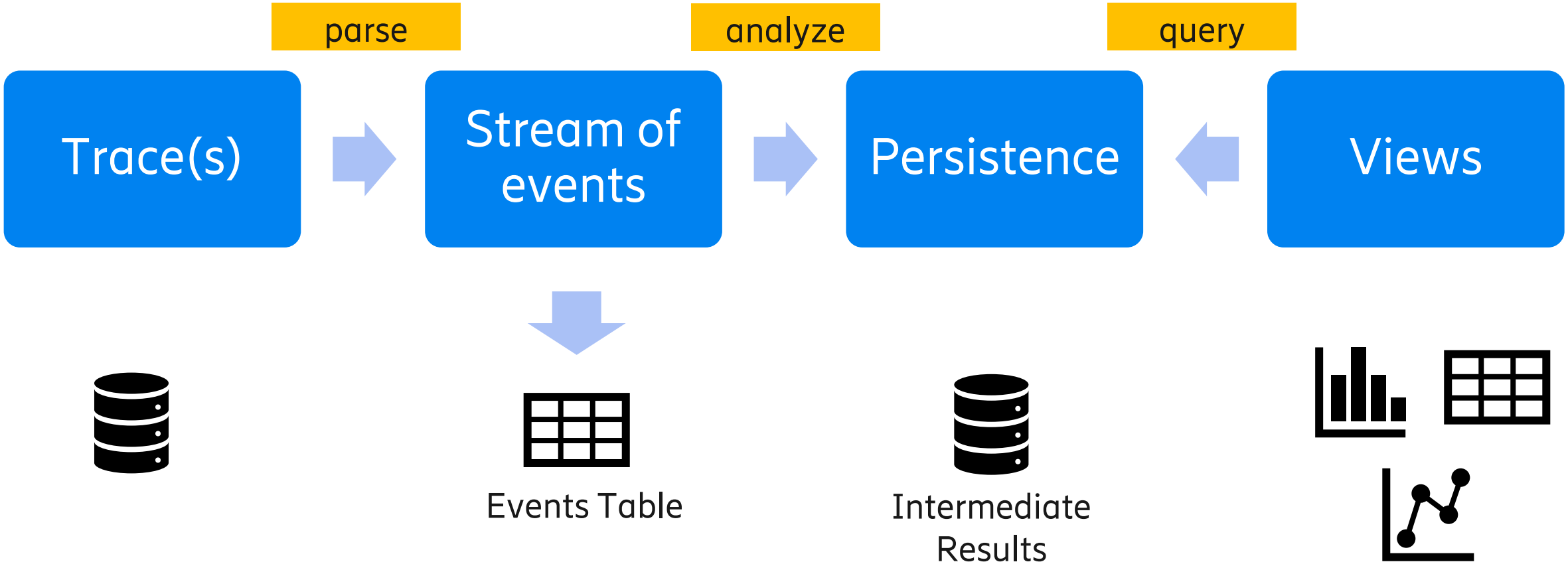
"One of these things is not like the others, one of these things is not the same"

Application *One*	≠	Application *Two*
Trace format *A*	≠	Trace format *B*
Analysis *Foo*	≠	Analysis *Bar*
Graph Baz	≠	Graph *Qux*
Problem *X*	≠	Problem *Y*

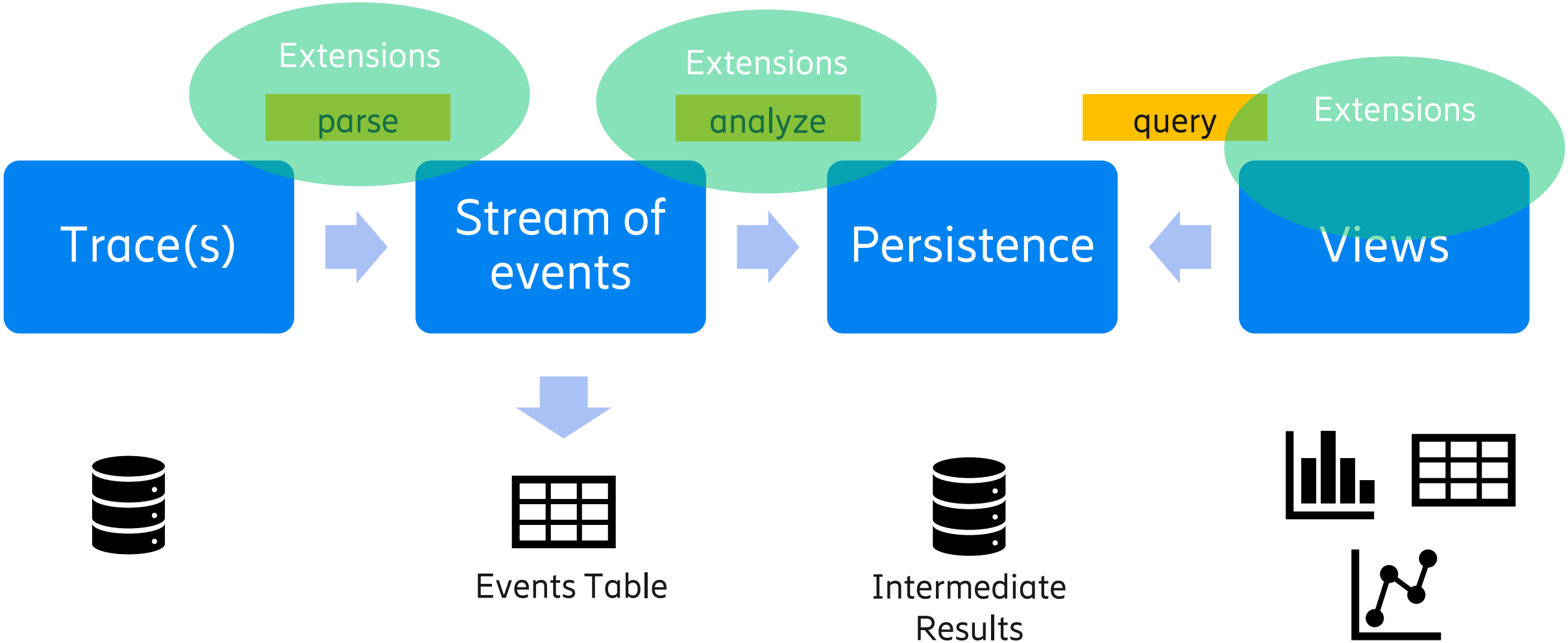
But many re-usable concepts!

Give flexibility to the users

Trace analysis and visualization flow

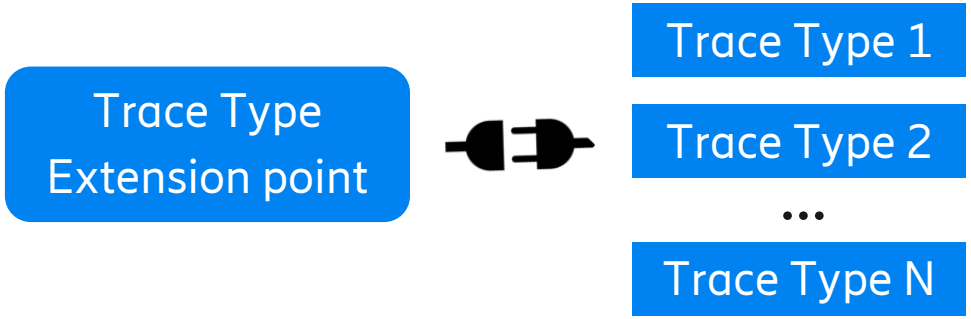


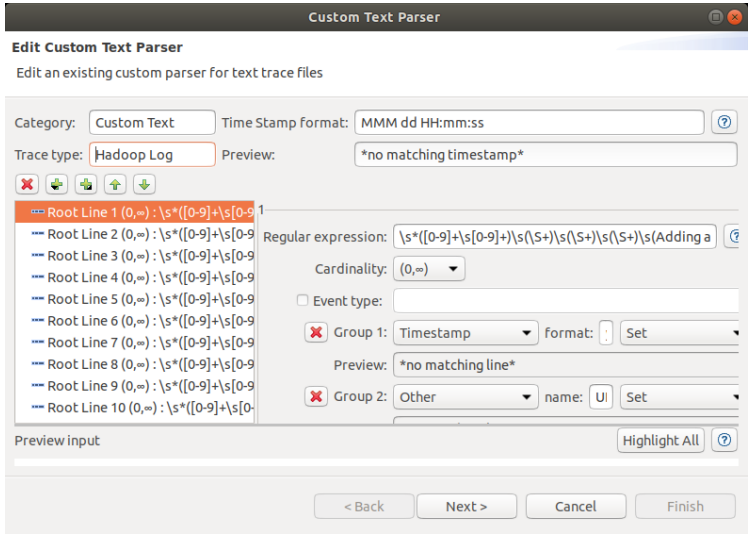
Trace analysis and visualization flow



Trace parser extensibility

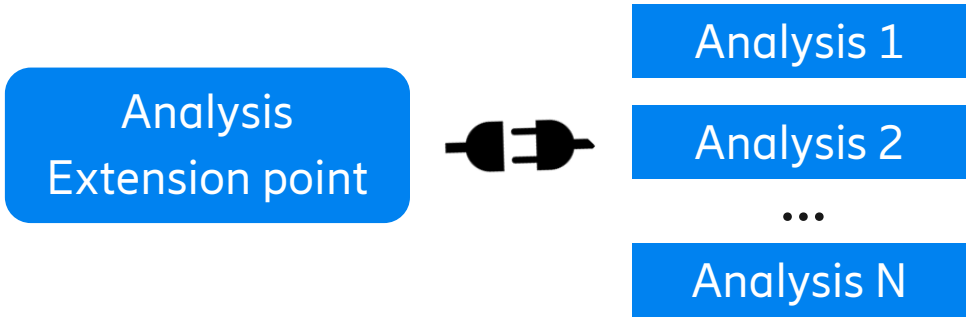
- Extension point to define trace parsers
  - Automatic trace type detection
  - UI to select and manage trace types
- Java APIs for parsers
- Build-in parsers, e.g. Common Trace Format (CTF)
- Custom Text and XML parser wizard
  - Create parser on-the-fly from UI



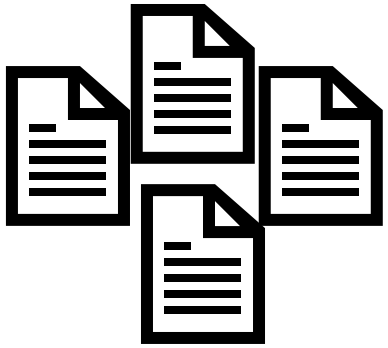


Analysis extensibility

- Extension point to define trace analysis
  - Manage analysis in Eclipse
  - Show available analysis in UI
- Java APIs
- Data persistence
  - state system
  - segment store
- XML defined analysis
  - Create them on-the-fly

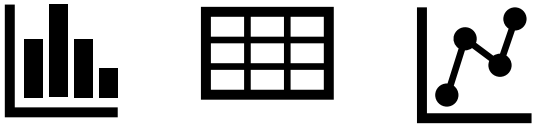


XML definitions

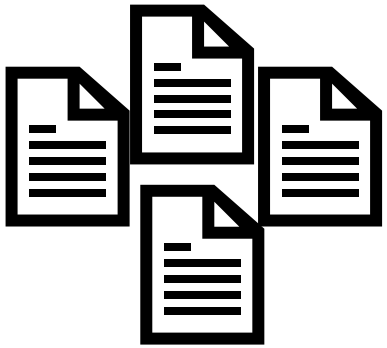


Visualization extensibility

- Java APIs for common views
- Default implementations
- Re-usable widgets
- XML defined views
  - Create them on-the-fly

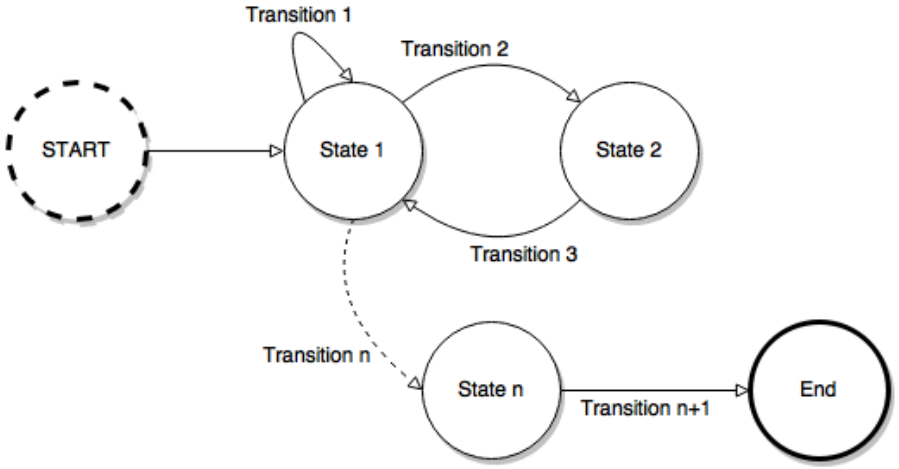


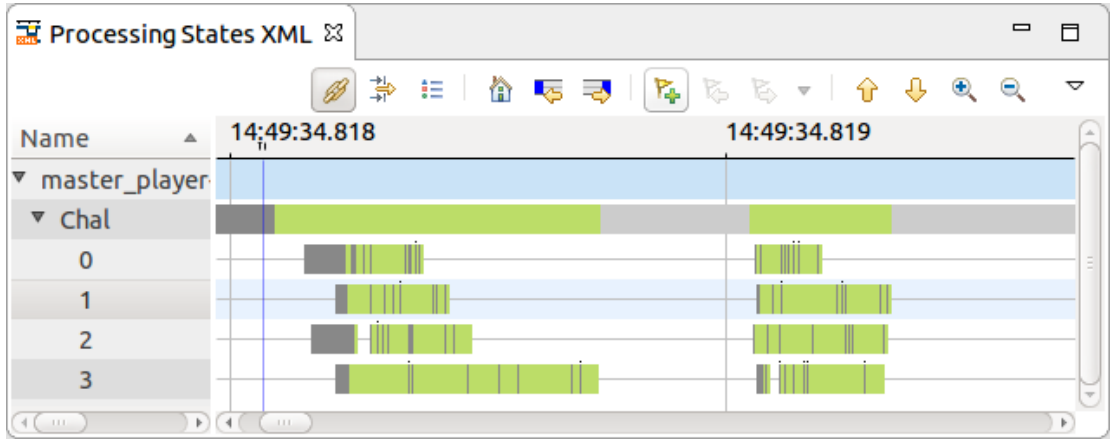
XML definitions

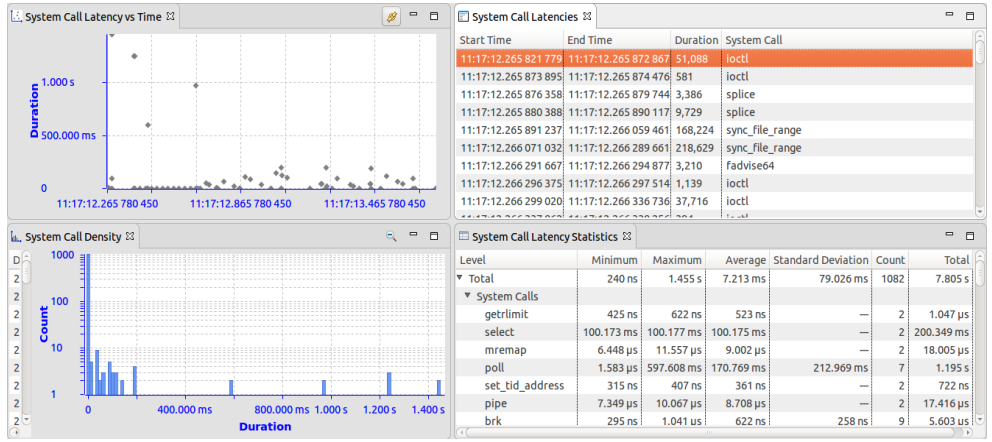


In-depth: XML analysis and views

- Find a sequence of data within a trace
- Generate state systems
- Do timing analysis
- Define specialized views
- Pattern analysis







XML analysis and views
Challenges

- Finite number of available analyses
- Some flexibility with XML analyses
- Very verbose
- Hard to read
- Hard to debug
- But it works!

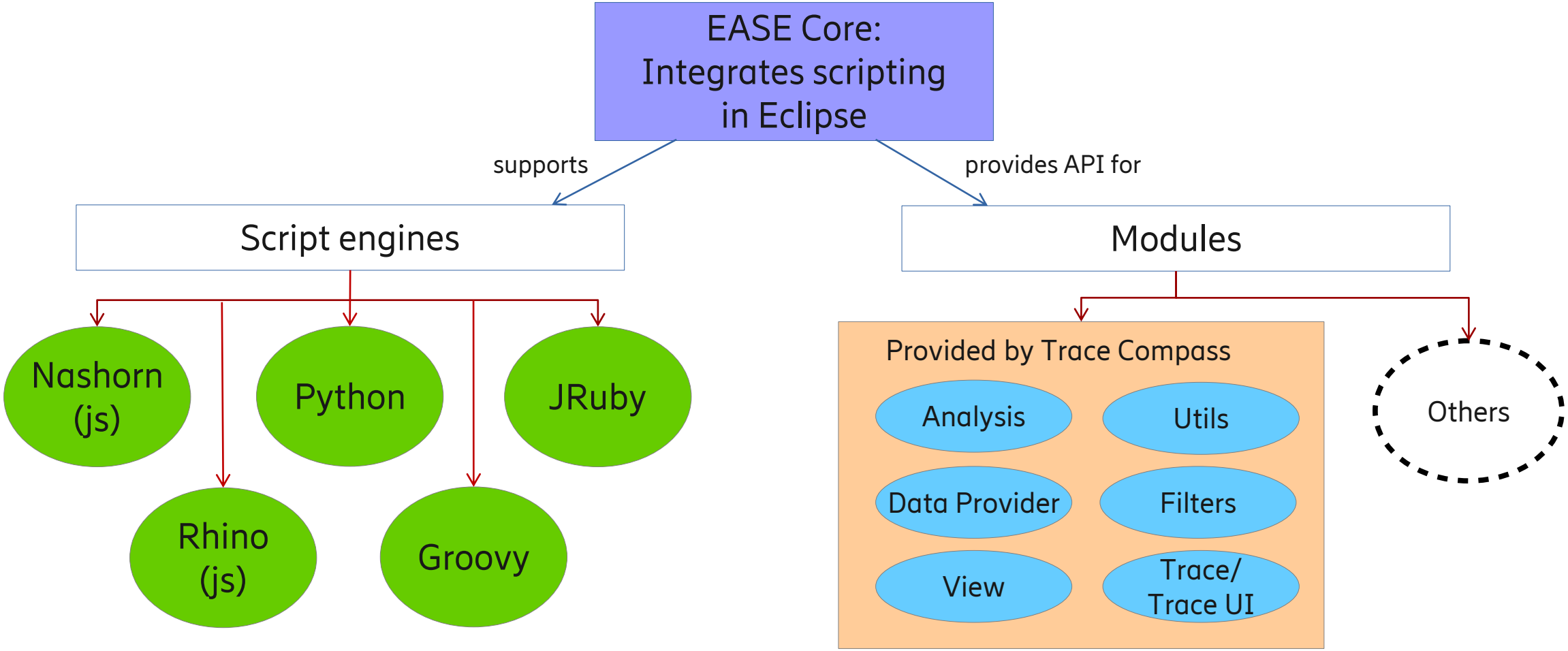
```
        <stateChange>
            <stateAttribute type="constant" value="#CurrentScenario" />
            <stateAttribute type="constant" value="type" />
            <stateValue type="eventField" value="evName" />
        </stateChange>
        <stateChange>
            <stateAttribute type="constant" value="#CurrentScenario" />
            <stateAttribute type="constant" value="component" />
            <stateValue type="script" value="cat == null ? 'UNKNOWN' : cat" scriptEngine="nashorn" >
                <stateValue id="cat" type="eventField" value="cat" />
            </stateValue>
        </stateChange>
    </action>

    <action id="push_event_type">
        <!-- Push the current event to the thread's callstack -->
        <stateChange>
                <stateAttribute type="location" value="CurrentThread" />
                <stateAttribute type="constant" value="CallStack" />
                <stateValue type="eventField" value="evName" stack="push"/>
        </stateChange>
        <stateChange>
                <stateAttribute type="location" value="CurrentThread" />
                <stateAttribute type="constant" value="cpu" />
                <stateValue type="eventField" value="cpu" />
        </stateChange>
    </action>

    <action id="pop_event_type">
        <!-- Pop the current event from the callstack -->
        <stateChange>
                <stateAttribute type="location" value="CurrentThread" />
                <stateAttribute type="constant" value="CallStack" />
                <stateValue type="eventField" value="evName" stack="pop"/>
        </stateChange>
    </action>

    <!-- FSMs -->

    <fsm id="tgThread" initial="Wait_thread_start">
        <state id="Wait_thread_start">
            <!-- The state will stay here until we have a thread start event -->
            <transition event="*" cond="is_start" target="in_thread" action="entering_thread:push_event_type"/>
        </state>
        <state id="in_thread" >
            <!-- The state will loop on itself until the thread ends and increment the operations that happen during the execution -->
            <transition event="*" cond="thread_thread:is_start" target="in_thread" action="push_event_type"/>
            <transition event="*" cond="thread_thread:is_end:last_out" target="end_thread" action="pop_event_type"/>
            <transition event="*" cond="thread_thread:is_end" target="in_thread" action="pop_event_type"/>
        </state>
```

XML analysis and views
Challenges

- Finite number of available analyses
- Some flexibility with XML analyses
- Very verbose
- Hard to read
- Hard to debug
- But it works!
- Ultimate flexibility: Scripting

```python
# load proper Trace Compass modules
loadModule('/TraceCompass/Analysis');
loadModule('/TraceCompass/View');
loadModule('/TraceCompass/DataProvider');

from py4j.java_gateway import JavaClass

# Create an analysis for this script
analysis = getAnalysis("activetid_python.js")

if analysis is None:
    print("Trace is null")
    exit()

# Get the analysis's state system so we can fill it, true indicates to re-use an existing state system, false would create
ss = analysis.getStateSystem(False)

# The state system methods require a vararg array. This puts the string in a vararg array to call those methods
def strToVarargs(str):
    object_class = java.lang.String
    object_array = gateway.new_array(object_class, 1)
    object_array[0] = str
    return object_array

# The analysis itself is in this function
def runAnalysis():
    # Get the event iterator for the trace
    iter = analysis.getEventIterator()

    # Parse all events
    event = None
    while iter.hasNext():
        # The python java gateway keeps a reference to the Java objects it sends to python. To avoid OutOfMemoryException,
        if not(event is None):
            gateway.detach(event)

        event = iter.next();

        # Do something when the event is a sched_switch
        if event.getName() == "sched_switch":
            # This function is a wrapper to get the value of field CPU in the event, or return null if the field is not pr
            cpu = getFieldValue(event, "CPU")
            tid = getFieldValue(event, "next_tid")
            if (not(cpu is None) and not(tid is None)):
                # Write the tid to the state system, for the attribute corresponding to the cpu
                quark = ss.getQuarkAbsoluteAndAdd(strToVarargs(str(cpu)))
                # modify the value, tid is a long, so "" + tid make sure it's a string for display purposes
                ss.modifyAttribute(event.getTimestamp().toNanos(), str(tid), quark)
```

EASE – Eclipse Advanced Scripting Environment



EASE highlights

- Execute scripts in context of Eclipse
- Supports multiple scripting languages
- Provide access to your application classes (java)
- Interact with your workbench
- Integrate scripts into UI (e.g. toolbar, menu) using keywords
- Extend easily existing application
- Rapid prototyping
- Plug-in mechanism to add custom modules
- Easily share scripts

Example scripts: Demo

```
int main(int argc, char** argv) {
  // Initialize the MPI environment
  MPI_Init(NULL, NULL);
  // Find out rank, size
  int world_rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
  int world_size;
  MPI_Comm_size(MPI_COMM_WORLD, &world_size);

  int token;
  // Receive from the lower process and send to the higher process. Take care
  // of the special case when you are the first process to prevent deadlock.
  if (world_rank != 0) {
    MPI_Recv(&token, 1, MPI_INT, world_rank - 1, 0, MPI_COMM_WORLD,
             MPI_STATUS_IGNORE);
    printf("Process %d received token %d from process %d\n", world_rank, token,
           world_rank - 1);
  } else {
    // Set the token's value if you are process 0
    token = -1;
  }
  MPI_Send(&token, 1, MPI_INT, (world_rank + 1) % world_size, 0,
           MPI_COMM_WORLD);
  // Now process 0 can receive from the last process. This makes sure that at
  // least one MPI_Send is initialized before all MPI_Recvs (again, to prevent
  // deadlock)
  if (world_rank == 0) {
    MPI_Recv(&token, 1, MPI_INT, world_size - 1, 0, MPI_COMM_WORLD,
             MPI_STATUS_IGNORE);
    printf("Process %d received token %d from process %d\n", world_rank, token,
           world_size - 1);
  }
  MPI_Finalize();
}
```

Initialization Complete

Receive entry

Receive exit

Send entry

Send exit

Receive entry

Receive exit

Instrumented application with custom tracepoints

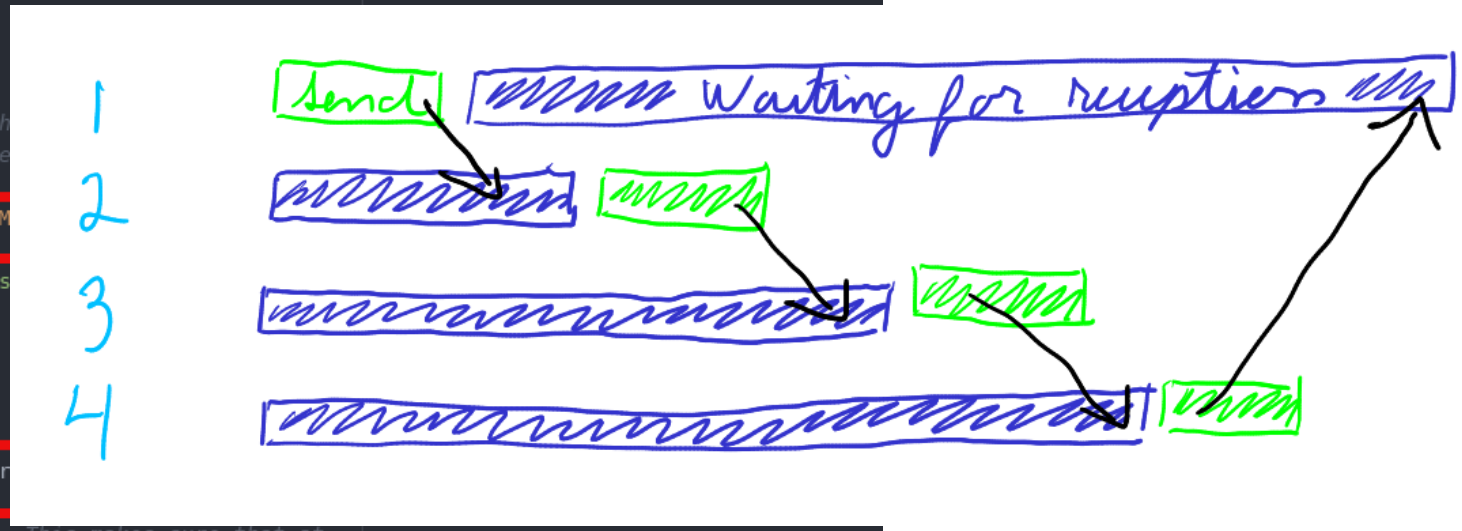
Example scripts: Demo

```
int main(int argc, char** argv) {
  // Initialize the MPI environment
  MPI_Init(NULL, NULL);
  // Find out rank, size
  int world_rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
  int world_size;
  MPI_Comm_size(MPI_COMM_WORLD, &world_size);

  int token;
  // Receive from the lower process and send to the h
  // of the special case when you are the first proce
  if (world_rank != 0) {
    MPI_Recv(&token, 1, MPI_INT, world_rank - 1, 0, M
             MPI_STATUS_IGNORE);
    printf("Process %d received token %d from process
           world_rank - 1);
  } else {
    // Set the token's value if you are process 0
    token = -1;
  }
  MPI_Send(&token, 1, MPI_INT, (world_rank + 1) % wor
           MPI_COMM_WORLD);
  // Now process 0 can receive from the last process. This makes sure that at
  // least one MPI_Send is initialized before all MPI_Recvs (again, to prevent
  // deadlock)
  if (world_rank == 0) {
    MPI_Recv(&token, 1, MPI_INT, world_size - 1, 0, MPI_COMM_WORLD,
             MPI_STATUS_IGNORE);
    printf("Process %d received token %d from process %d\n", world_rank, token,
           world_size - 1);
  }
  MPI_Finalize();
}
```

Initialization Complete

Receive entry

Receive exit



The view we want to get

Example scripts: Demo

```
// Load the proper modules
loadModule("/TraceCompass/Trace")

// Get the active trace
var trace = getActiveTrace()

// Get an event iterator on that trace
var iter = getEventIterator(trace)

// Iterate through the events
var event = null
while (iter.hasNext()) {
        event = iter.next()

        // For each event, print the name and the field names
        eventString = event.getName() + " --> ( "

        var fieldsIterator = event.getContent().getFieldNames().iterator()
        while (fieldsIterator.hasNext()) {
                eventString += fieldsIterator.next() + " "
        }
        eventString += ")"

        print(eventString);
}
```

Step 1:
Read events from the trace

Example scripts: Demo

```diff
@@ -11,20 +11,37 @@ var trace = getActiveTrace()

 // Get an event iterator on that trace
 var iter = getEventIterator(trace)
+// Associate a TID with an mpi worker
+var tidToWorkerMap = {};

 // Iterate through the events
 var event = null
 while (iter.hasNext()) {
        event = iter.next()

-       // For each event, print the name and the field names
-       eventString = event.getName() + " --> ( "
-
-       var fieldsIterator = event.getContent().getFieldNames().iterator()
-       while (fieldsIterator.hasNext()) {
-               eventString += fieldsIterator.next() + " "
+       eventName = event.getName()
+       if (eventName == "ring:init") {
+               tid = getEventFieldValue(event, "context._vtid")
+               worker_id = getEventFieldValue(event, "worker_id")
+               tidToWorkerMap[tid] = worker_id
+               print("Init -> tid: " + tid + ", worker_id: " + worker_id)
+       } else if (eventName == "ring:recv_entry") {
+               tid = getEventFieldValue(event, "context._vtid")
+               worker_id = tidToWorkerMap[tid]
+               print("Entering Reception -> tid: " + tid+ ", worker_id: " + worker_id)
+       } else if (eventName == "ring:recv_exit") {
+               tid = getEventFieldValue(event, "context._vtid")
+               worker_id = tidToWorkerMap[tid]
+               source = getEventFieldValue(event, "source")
+               print("Exiting Reception -> tid: " + tid + ", worker_id: " + worker_id + ", source: " + sour
+       } else if (eventName == "ring:send_entry") {
+               tid = getEventFieldValue(event, "context._vtid")
+               worker_id = tidToWorkerMap[tid]
+               dest = getEventFieldValue(event, "dest")
+               print("Entering Send -> tid: " + tid + ", worker_id: " + worker_id + ", dest: " + dest)
+       } else if (eventName == "ring:send_exit") {
+               tid = getEventFieldValue(event, "context._vtid")
+               worker_id = tidToWorkerMap[tid]
+               print("Exiting Send -> tid: " + tid + ", worker_id: " + worker_id)
        }
-       eventString += ")"
-
-       print(eventString);
 }
```

Step 2:
For each event, get the event field values

Example scripts: Demo

```
@@ -5,6 +5,7 @@

 // Load the proper modules
 loadModule("/TraceCompass/Trace")
+loadModule("/TraceCompass/Analysis")

 // Get the active trace
 var trace = getActiveTrace()
@@ -14,6 +15,11 @@ var iter = getEventIterator(trace)
 // Associate a TID with an mpi worker
 var tidToWorkerMap = {};

+//Get an analysis
+var analysis = createScriptedAnalysis(trace, "ringTimeLine.js")
+// Get the analysis's state system so we can fill it, false indicates to create a new state system even if
+var ss = analysis.getStateSystem(false);
+
 // Iterate through the events
 var event = null
 while (iter.hasNext()) {
@@ -24,24 +30,36 @@ while (iter.hasNext()) {
               tid = getEventFieldValue(event, "context._vtid")
               worker_id = getEventFieldValue(event, "worker_id")
               tidToWorkerMap[tid] = worker_id
-               print("Init -> tid: " + tid + ", worker_id: " + worker_id)
       } else if (eventName == "ring:recv_entry") {
               tid = getEventFieldValue(event, "context._vtid")
               worker_id = tidToWorkerMap[tid]
-               print("Entering Reception -> tid: " + tid+ ", worker_id: " + worker_id)
+               // Save the state of the resource as waiting for reception
+               quark = ss.getQuarkAbsoluteAndAdd(worker_id);
+               ss.modifyAttribute(event.getTimestamp().toNanos(), "Waiting for reception", quark);
       } else if (eventName == "ring:recv_exit") {
               tid = getEventFieldValue(event, "context._vtid")
               worker_id = tidToWorkerMap[tid]
               source = getEventFieldValue(event, "source")
-               print("Exiting Reception -> tid: " + tid + ", worker_id: " + worker_id + ", source: " + sour
+               // Remove the waiting for reception state
+               quark = ss.getQuarkAbsoluteAndAdd(worker_id);
+               ss.removeAttribute(event.getTimestamp().toNanos(), quark);
```

[...]

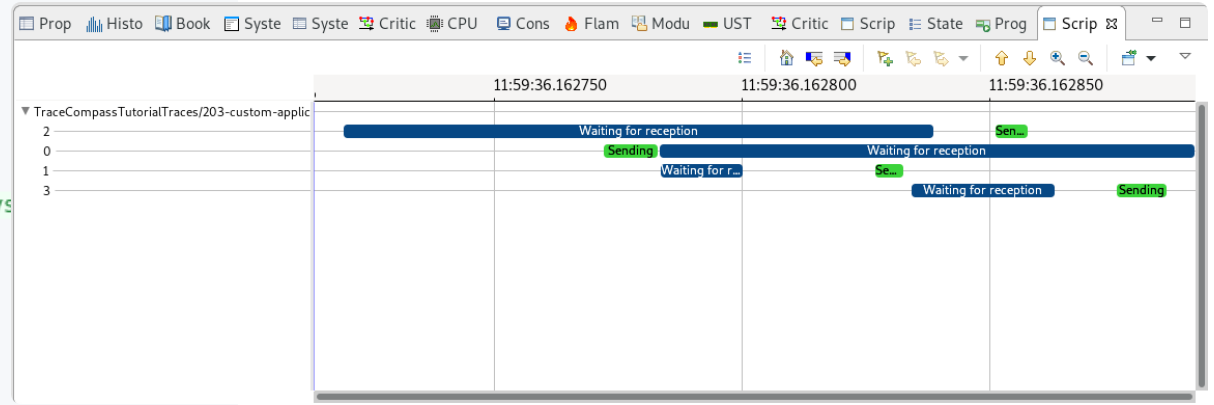
```
+
+// Done parsing the events, close the state system at the time of the last event, it needs to be done manua
+if (event != null) {
+       ss.closeHistory(event.getTimestamp().toNanos());
+}
```

Step 3:
Save the states in a state system (backend from which views get their data)

Example scripts: Demo

```
@@ -6,6 +6,8 @@
 // Load the proper modules
 loadModule("/TraceCompass/Trace")
 loadModule("/TraceCompass/Analysis")
+loadModule("/TraceCompass/DataProvider")
+loadModule("/TraceCompass/View")

 // Get the active trace
 var trace = getActiveTrace()
@@ -63,3 +65,13 @@ while (iter.hasNext()) {
 if (event != null) {
        ss.closeHistory(event.getTimestamp().toNanos());
 }
+
+//Get a time graph provider from this analysis, displaying all attributes
+//Create a map and fill it, because javascript map cannot use the EASE constants as keys
+var map = new java.util.HashMap();
+map.put(ENTRY_PATH, '*');
+provider = createTimeGraphProvider(analysis, map);
+if (provider != null) {
+       // Open a time graph view displaying this provider
+       openTimeGraphView(provider);
+}
```

Step 4:
Show a time graph view for the states

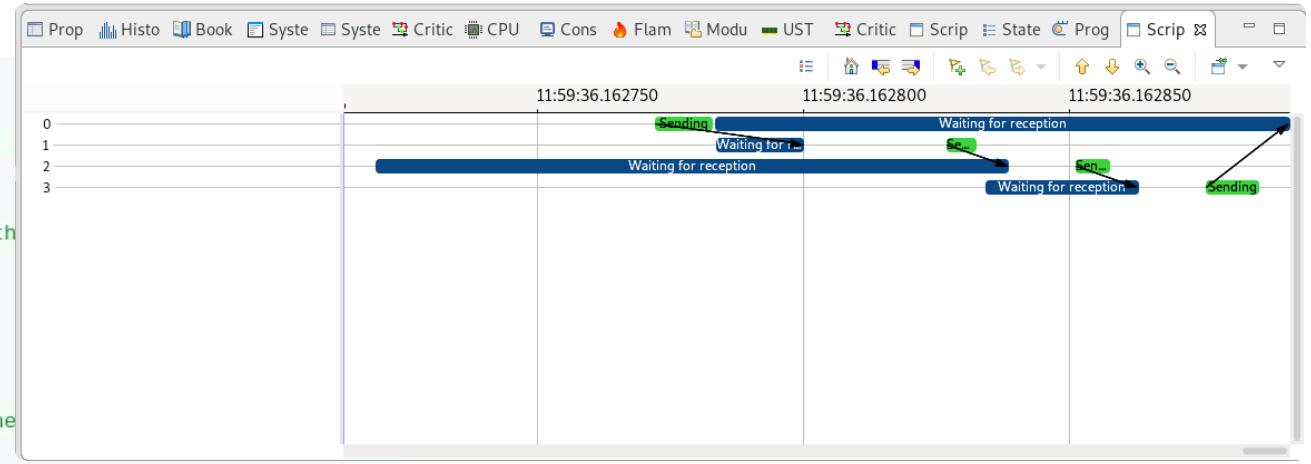


Example scripts: Demo

See [the Trace Compass EASE Tutorial lab]
for complete diff and script

Step 5:
Add the arrows

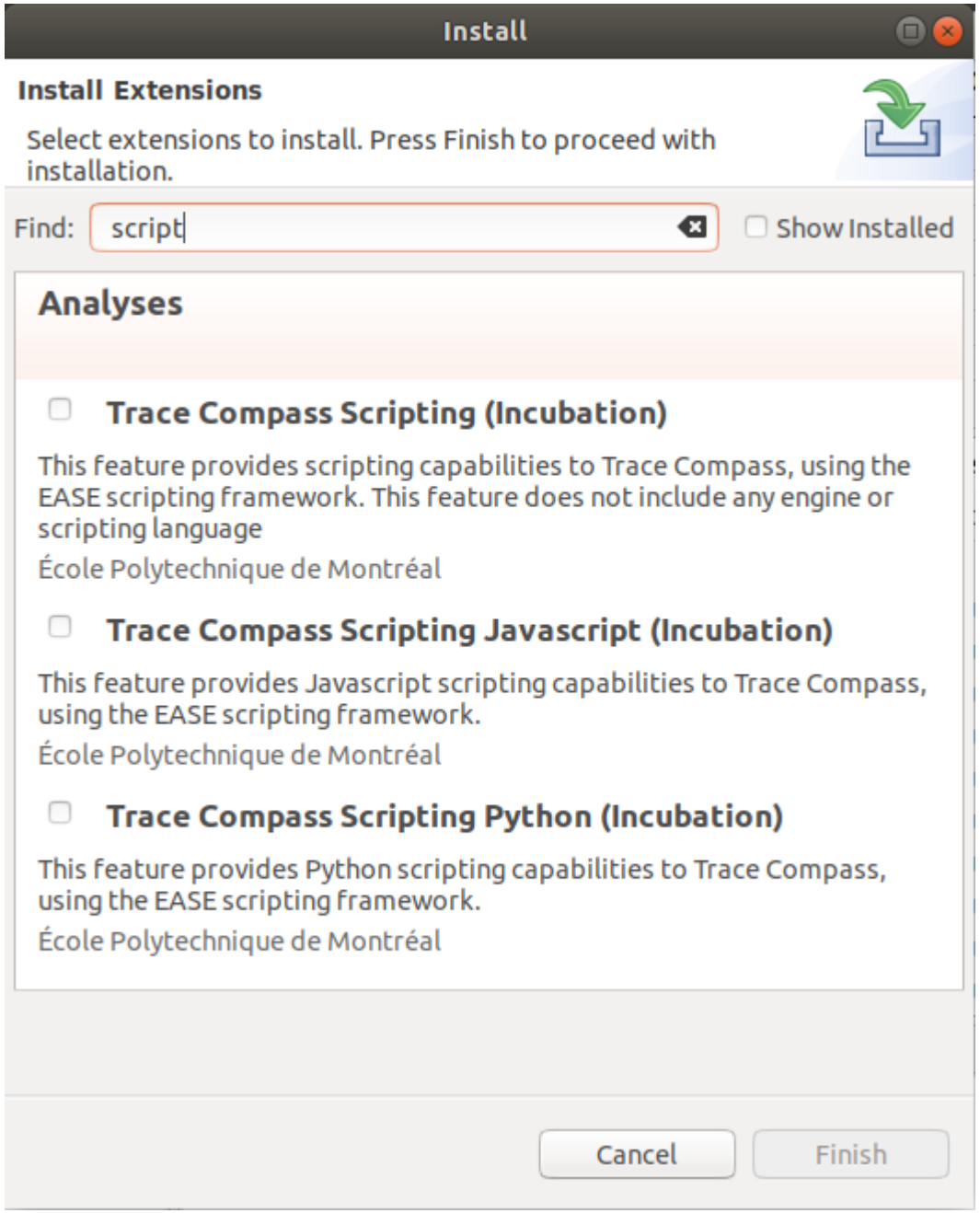
```
+     [...]
+
+       duration = arrow["endTime"] - startTime;
+       // Add the arrow to the arrows list
+       tgArrows.getList().add(createArrow(srcId, dstId, startTime, duration,
+}
+
+// A function used to return the entries to the data provider. It receives th
+function getEntries(parameters) {
+       // The list is static once built, return all entries
+       return tgEntries.getList();
+}
+
+// A function used to return the arrows to the data provider. It receives the
+function getArrows(parameters) {
+       // Just return all the arrows, the view will take those in the range
+       return tgArrows.getList();
+}
+
+// Create a scripted data provider for this analysis, using script functions to get the entries, row model
+provider = createScriptedTimeGraphProvider(analysis, getEntries, null, getArrows);
 if (provider != null) {
                        // Open a time graph view displaying this provider
                        openTimeGraphView(provider);
```



How-to install

Trace Compass add-ons

— In RCP use" Menu "Tools -> Add-ons..."
— In Eclipse IDE use: [Update Site]



Takeaways

- Trace Compass is extensible
- EASE provides scripting capabilities to Eclipse applications
- Perfect match: EASE and Trace Compass
- EASE Scripting is now available for Trace Compass users
- Quickly create your own script and share it with colleagues (and community ☺)

References

- Trace Compass
  - http://tracecompass.org
  - https://projects.eclipse.org/projects/tools.tracecompass
  - https://projects.eclipse.org/projects/tools.tracecompass.incubator
- Trace Compass Scripting
  - https://archive.eclipse.org/tracecompass.incubator/doc/org.eclipse.tracecompass.incubator.scripting.doc.user/User-Guide.html
  - http://versatic.net/tracecompass/introducingEase.html
  - https://github.com/tahini/tracecompass-ease-scripting
  - https://github.com/tuxology/tracevizlab/
- Eclipse EASE
  - https://projects.eclipse.org/projects/technology.ease

Contacts

- Presenter
  - Geneviève Bastien: gbastien@versatic.net
  - Bernd Hufmann: bernd.hufmann@ericsson.com
- Mailing list
  - tracecompass-dev@eclipse.org
- IRC
  - oftc.net #tracecompass