Introducing:
Trace Server Protocol

Bernd Hufmann, Ericsson AB

2018-10-25

Agenda

- Motivation and goal
- Background
- Trace Server Protocol (TSP)
- Opportunities
- Ongoing work / Demo
- Open discussions

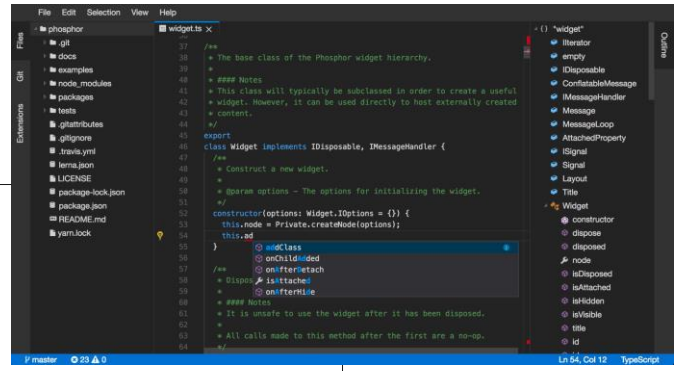
Motivation

- Increasing popularity of web-based technologies
- Integration with next generation IDEs (e.g. Theia)
  - Success of Language Server Protocol
- Automated trace analysis
  - CI environment
  - Trouble reports
- Scale trace analysis for traces larger than local disk space

Goals of presentation

- Present idea of Trace Server Protocol
- Create awareness and interest in the community
- Collect early feedback
- Collaboration

Next generation IDE



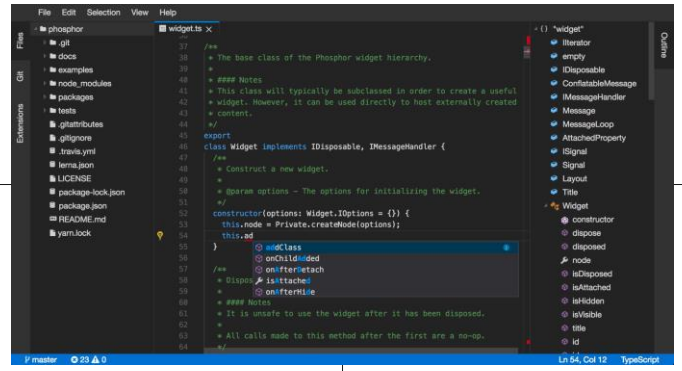
Language Server Protocol (LSP)

Debug Adapter Protocol (DAP)

Language Server

Debug Server

Next generation IDE



Language Server Protocol (LSP)

Debug Adapter Protocol (DAP)

Trace Server Protocol (TSP)

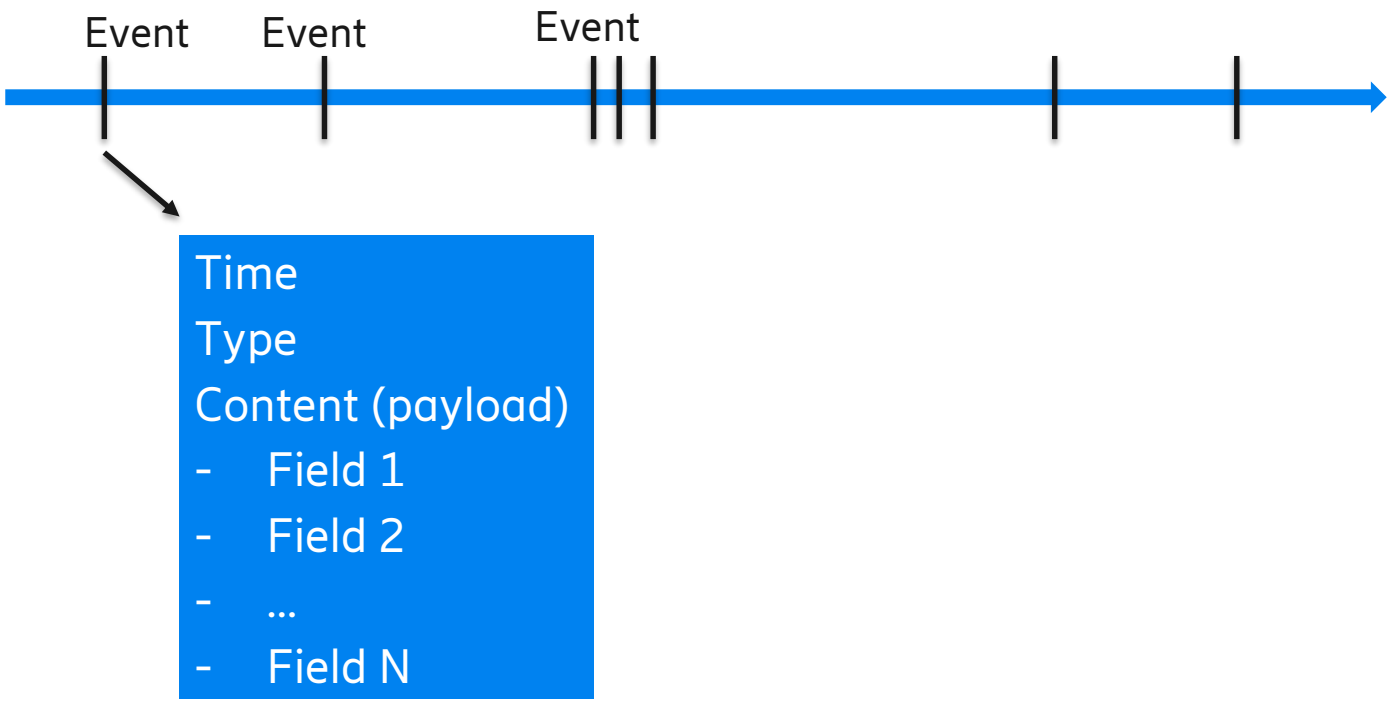
Language Server

Debug Server

Trace Server

What is a trace?

- A series of events over time
- Events collected at tracepoints during program execution
- Each event has a type and content fields (payload)



What can we do with events?

— Show the raw events in an events table

ls-traces

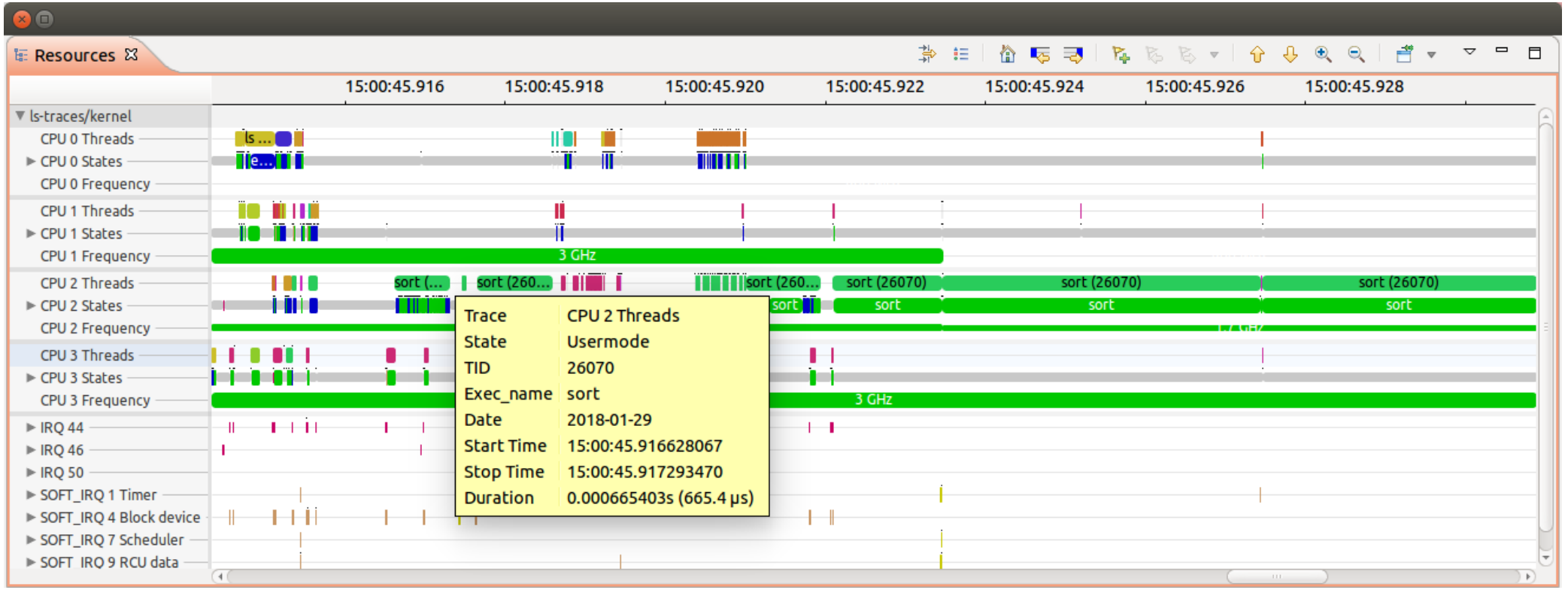
Timestamp	Channel	CPU	Event type	Contents
<srch>	<srch>	<srch>	<srch>	<srch>
2018-01-29 15:00:45.956 153 666	userspace_2	2	lttng_ust_cyg_profile_fast:func_entry	addr=0x402b80, context._vpid=26070, context._vtid=26070, context._procname=
2018-01-29 15:00:45.956 153 919	userspace_2	2	lttng_ust_cyg_profile_fast:func_exit	context._vpid=26070, context._vtid=26070, context._procname=sort
2018-01-29 15:00:45.956 154 187	userspace_2	2	lttng_ust_cyg_profile_fast:func_exit	context._vpid=26070, context._vtid=26070, context._procname=sort
2018-01-29 15:00:45.956 154 468	userspace_2	2	lttng_ust_cyg_profile_fast:func_exit	context._vpid=26070, context._vtid=26070, context._procname=sort
2018-01-29 15:00:45.956 154 567	kernel_0	0	kmem_kmalloc	call_site=0xffffffffa08b9898, ptr=0xffff8804168c4c68, bytes_req=8, bytes_alloc=
2018-01-29 15:00:45.956 154 715	userspace_2	2	lttng_ust_cyg_profile_fast:func_exit	context._vpid=26070, context._vtid=26070, context._procname=sort
2018-01-29 15:00:45.956 154 955	kernel_1	1	syscall_entry_recvmsg	fd=6, msg=140722071796864, flags=0, context._perf_cpu_migrations=32
2018-01-29 15:00:45.956 154 996	kernel_0	0	syscall_entry_poll	timeout_msecs=-1, nfds=1, fds_length=1, overflow=0, fds=[[fd=3, raw_events=0x
2018-01-29 15:00:45.956 155 031	userspace_2	2	lttng_ust_cyg_profile_fast:func_exit	context._vpid=26070, context._vtid=26070, context._procname=sort
2018-01-29 15:00:45.956 155 231	kernel_1	1	syscall_exit_recvmsg	ret=-11, msg=140722071796864, context._perf_cpu_migrations=32
2018-01-29 15:00:45.956 155 356	userspace_2	2	lttng_ust_cyg_profile_fast:func_exit	context._vpid=26070, context._vtid=26070, context._procname=sort
2018-01-29 15:00:45.956 155 386	kernel_0	0	kmem_kfree	call_site=0xffffffffa08b9a96, ptr=0xffff8804168c4c68, context._perf_cpu_migrat
2018-01-29 15:00:45.956 155 673	userspace_2	2	lttng_ust_cyg_profile_fast:func_exit	context._vpid=26070, context._vtid=26070, context._procname=sort
2018-01-29 15:00:45.956 155 955	userspace_2	2	lttng_ust_cyg_profile_fast:func_entry	addr=0x40a140, context._vpid=26070, context._vtid=26070, context._procname=
2018-01-29 15:00:45.956 156 229	userspace_2	2	lttng_ust_cyg_profile_fast:func_entry	addr=0x409fd0, context._vpid=26070, context._vtid=26070, context._procname=
2018-01-29 15:00:45.956 156 628	kernel_2	2	syscall_entry_write	fd=1, buf=140656486031360, count=14, context._perf_cpu_migrations=40
2018-01-29 15:00:45.956 157 240	kernel_1	1	kmem_kmalloc	call_site=0xffffffffa08b9898, ptr=0xffff880416db0b00, bytes_req=112, bytes_all
2018-01-29 15:00:45.956 157 306	kernel_2	2	workqueue_queue_work	work=0xffff880035e11a08, function=0xffffffff8145c700, req_cpu=256, context.

What can we do with events?

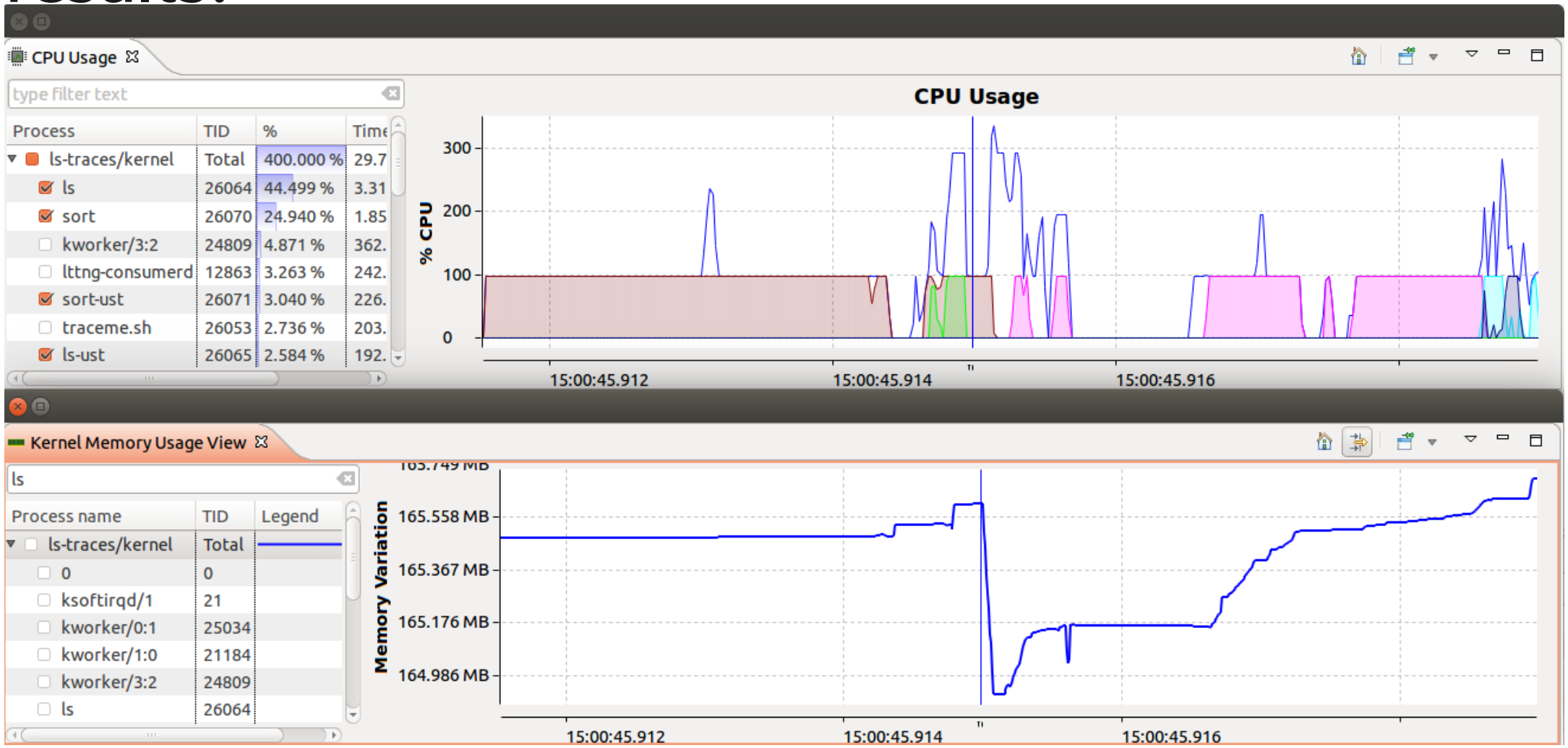
- Use as input for further analysis, e.g.
  - Write state machines
  - Create patterns
  - Analyze timing (measuring time between events)
  - Create execution graphs like critical path
  - Follow resource usage
  - ....

What can we do with the analysis results?

— Create all sorts of visualization graphs



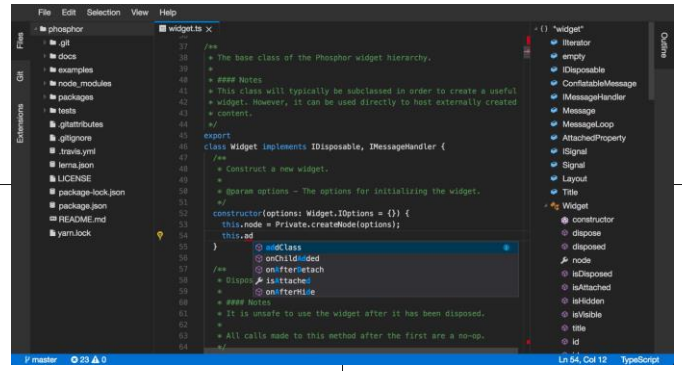
What can we do with the analysis results?



Tracing use cases

- Help developers, trouble-shooters, system admins etc. to solve complex problem
  - Profile applications using trace data
  - Find long executions
  - Get critical path executions
  - Correlate traces from different nodes
  - Investigate real-time deadlines
  - Find memory usage issues
  - Find high processor load
  - Investigate concurrency problems
  - …

Next generation IDE



Language Server Protocol (LSP)

Debug Adapter Protocol (DAP)

Trace Server Protocol (TSP)

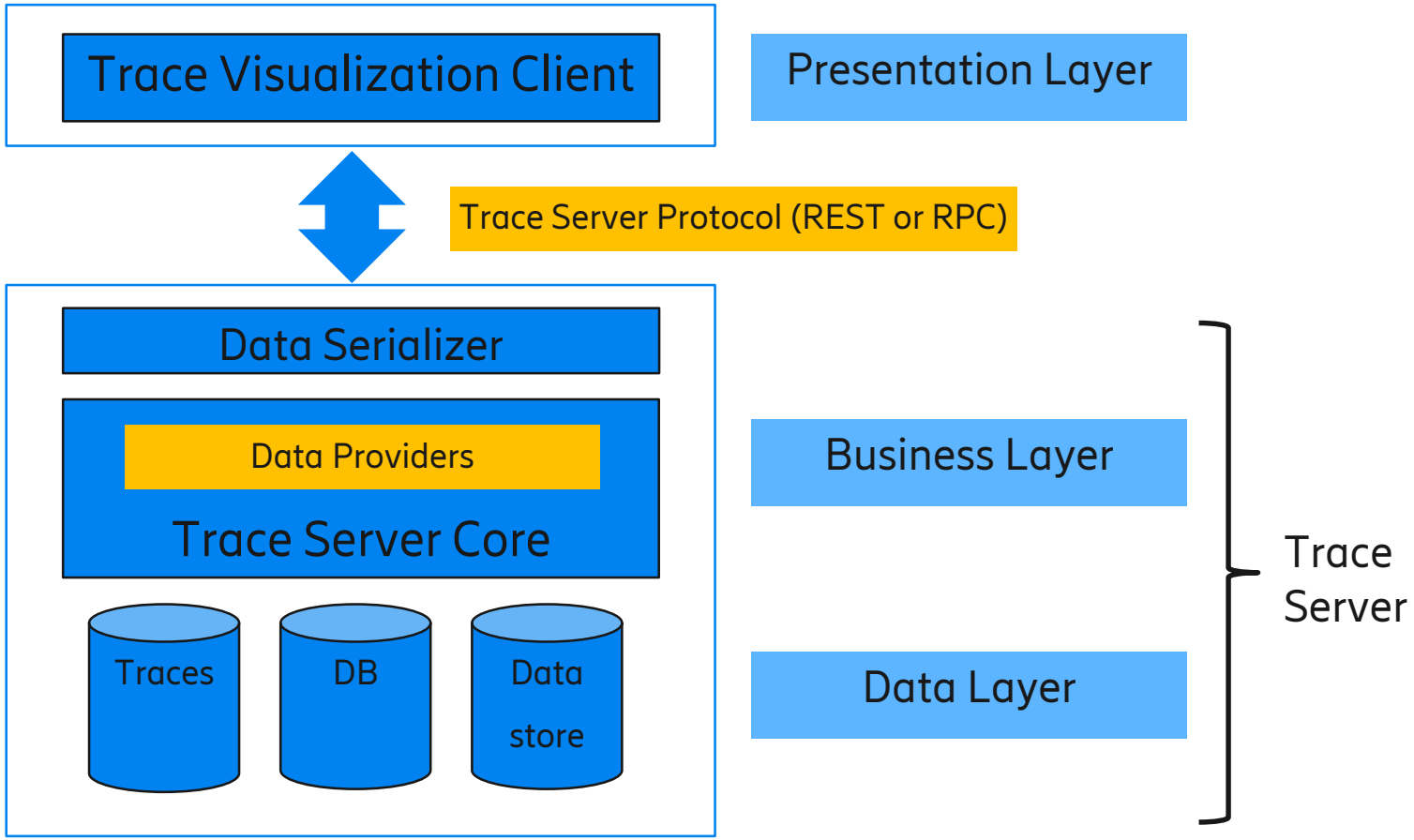
Language Server

Debug Server

Trace Server

Proposed architecture

- Layered architecture
- Separation of UI and core
- Trace Server Protocol
  - REST API or RPC



Scope of Trace Server Protocol (TSP)

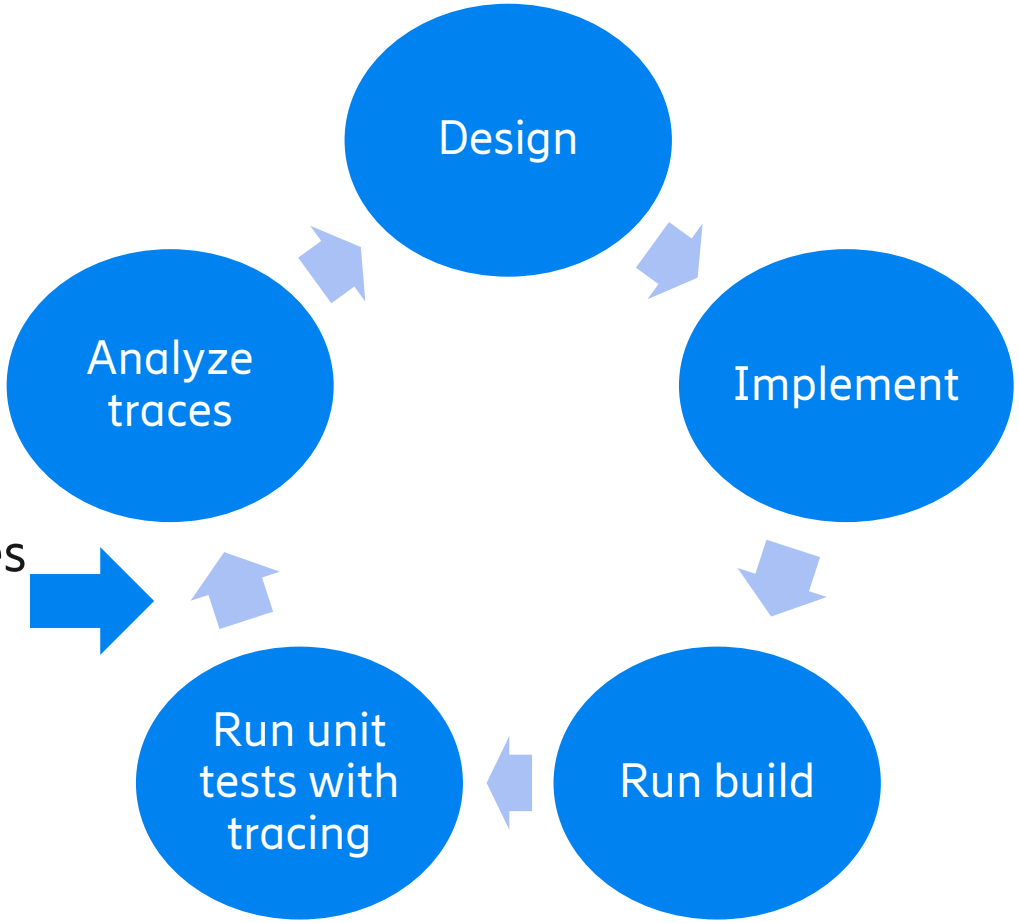
- Handle the communication between **core** and **UI** of trace viewer
- Exchange **visualization** data between a client and a server
- Trace file and trace set management
- Manage available visualizations
- Provide data for various visualization types
- Define filter and queries
- Manage trace annotations (e.g. bookmarks)

Not in TSP scope

- Interface to the compiler
- Interface to the debugger
- Interface to the tracer

Opportunities

Integrated feedback in design workflow



- Automatically detect traces
- Show trace files to user

Leveraging LSP and DAP

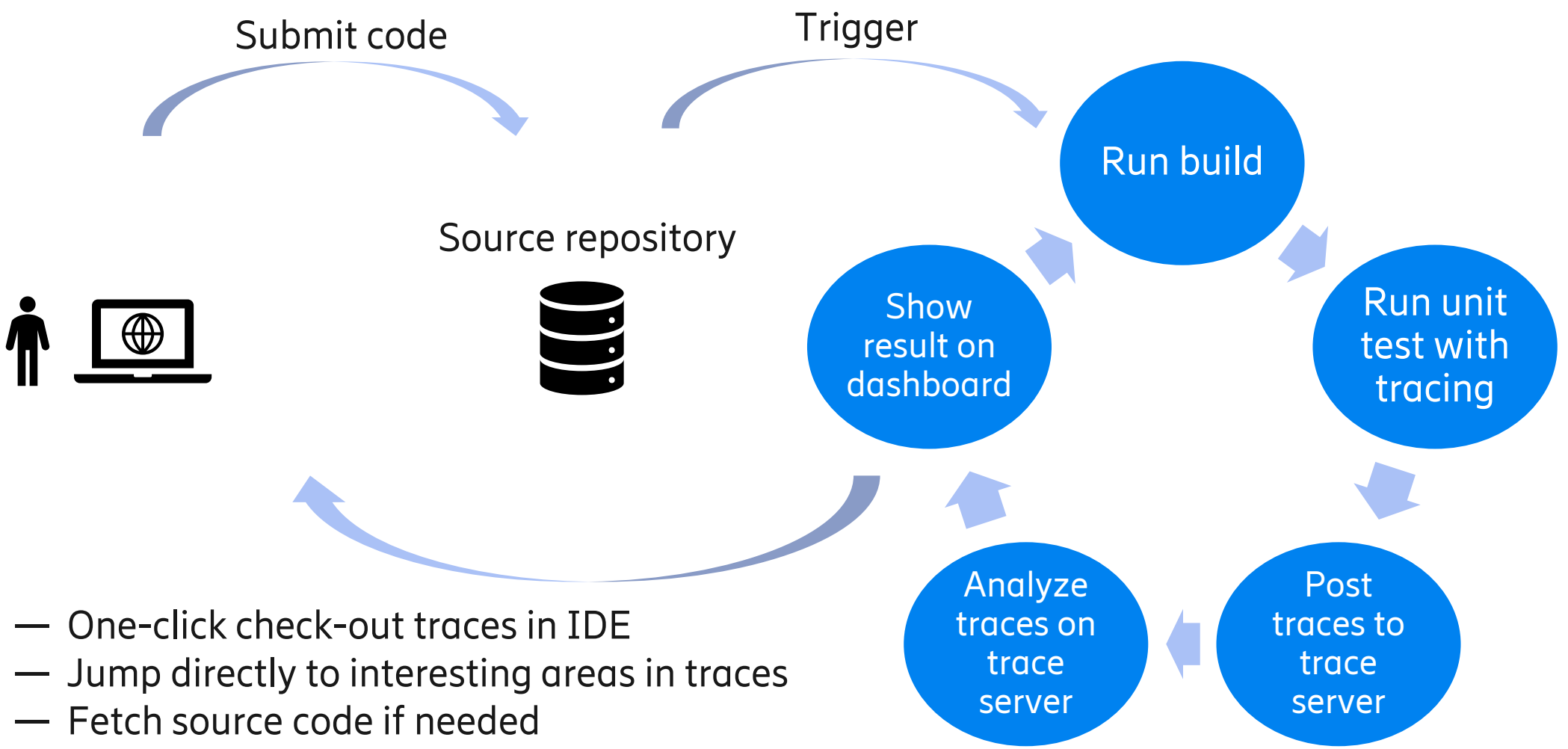
- Use DAP to get file and line number
- Then use LSP to lookup source code

CPU	Event type	Contents
<srch>	<srch>	<srch>
2	lttng_ust_cyg_profile_fast:func_entry	addr=0x402b80, context._vpid=26070, context._vtid=26070, context._procname
2	lttng_ust_cyg_profile_fast:func_exit	context._vpid=26070, context._vtid=26070, context._procname=sort
2	lttng_ust_cyg_profile_fast:func_exit	context._vpid=26070, context._vtid=26070, context._procname=sort
2	lttng_ust_cyg_profile_fast:func_exit	context._vpid=26070, context._vtid=26070, context._procname=sort
0	kmem_kmalloc	call_site=0xffffffffa08b9898, ptr=0xffff8804168c4c68, bytes_req=8, bytes_alloc=
0	ust_sequence:ENTER	file=simple_server_main.cpp, line=97, content=messageHandler()
0	ust_sequence:INFO	file=simple_server_main.cpp, line=113, content=player player1

- Use LSP to lookup source code

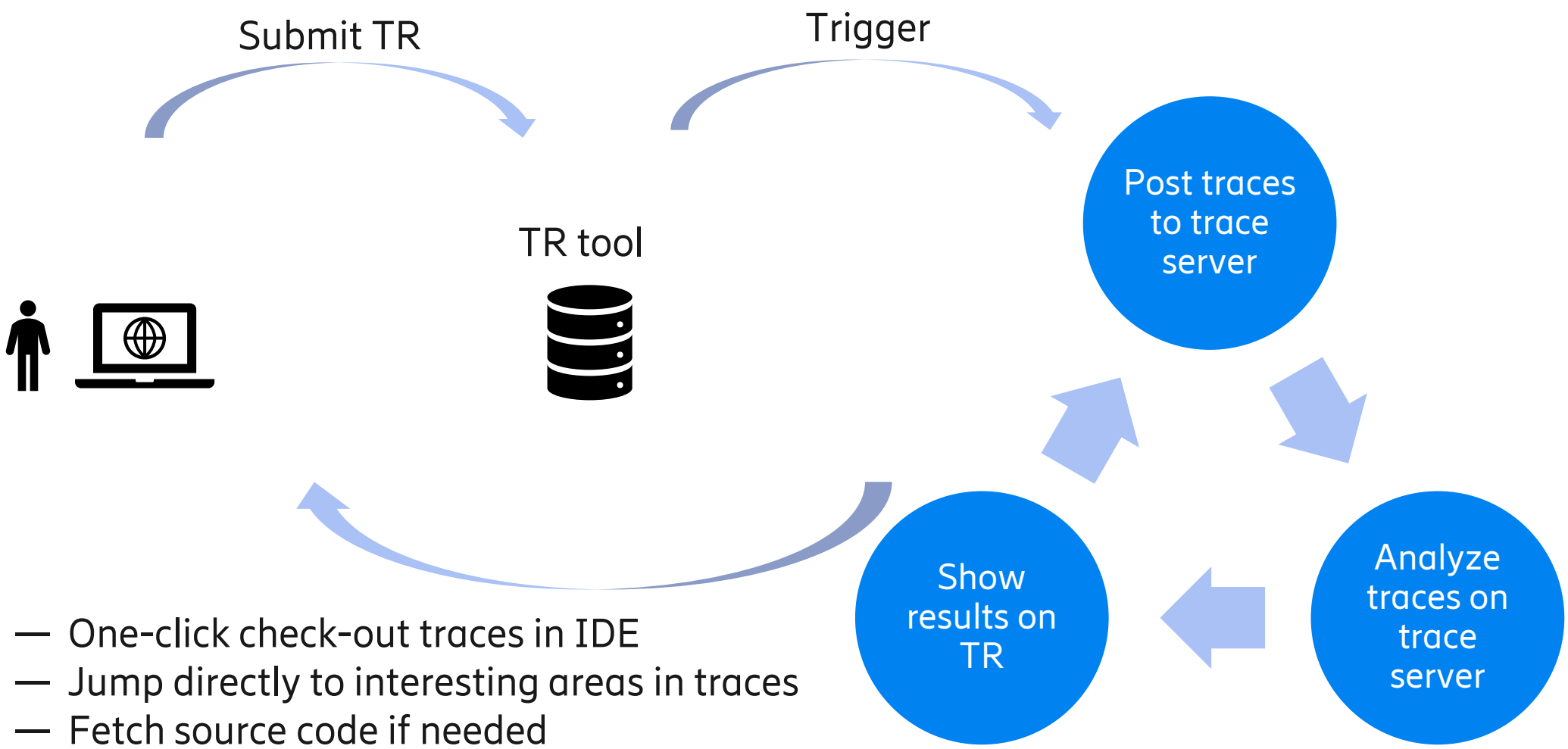
— Source code lookup from different graphs
— Automatic source code lookup when stepping

Continuous integration



- One-click check-out traces in IDE
- Jump directly to interesting areas in traces
- Fetch source code if needed

Integration with TR tools



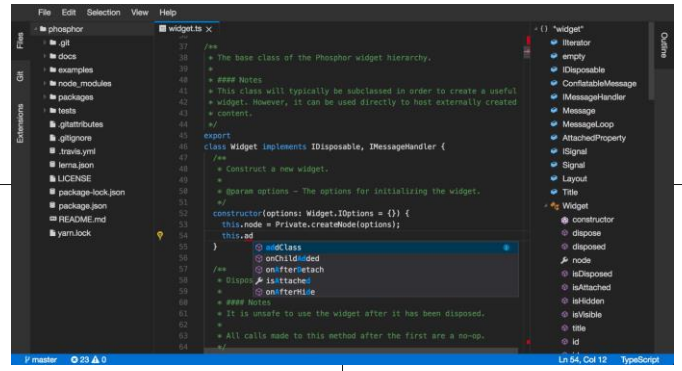
- One-click check-out traces in IDE
- Jump directly to interesting areas in traces
- Fetch source code if needed

Integration with workspace management applications

Eclipse Che

- Easily prepare workspaces for trouble-shooting sessions
  - Use cloud IDE
  - Get source code
    - Languages servers
  - Setup debuggers
    - Debug server
- Share trouble-shooting sessions (workspaces)

Enables micro-services architecture



Language Server Protocol (LSP)

Debug Adapter Protocol (DAP)

Trace Server Protocol (TSP)

Language Server

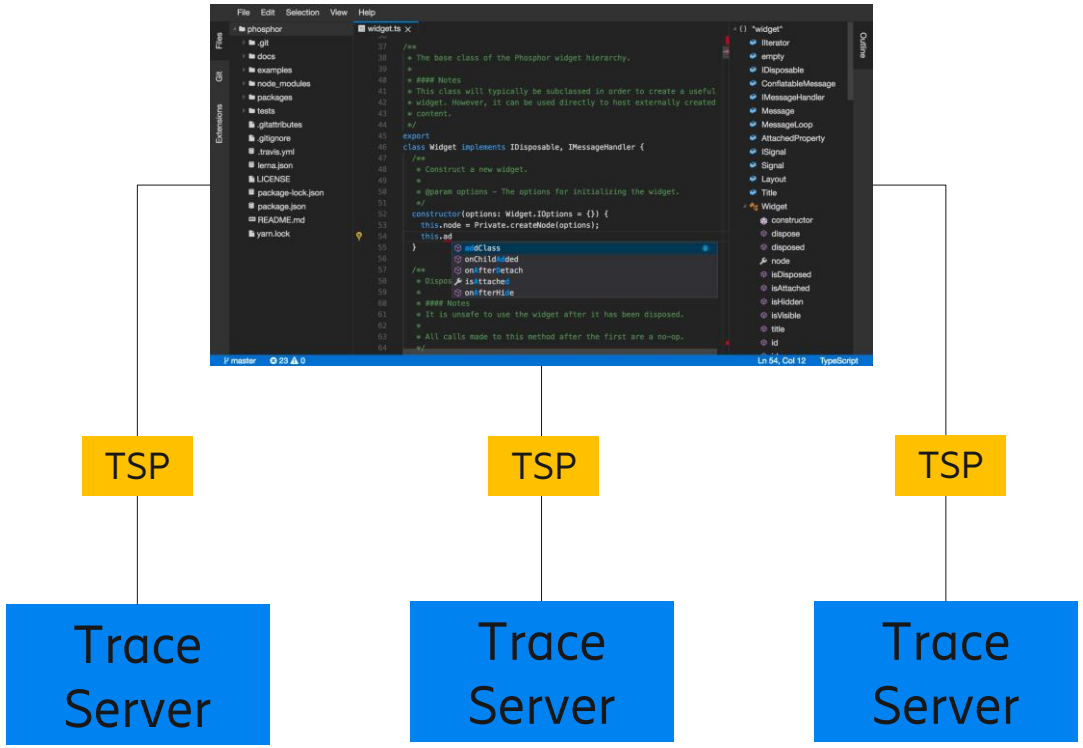
Debug Server

Microservice

Trace Server

Higher scalability

- Distributed architecture
- Parallel analysis
- Distributed analysis
- Multiple trace servers analyzing
  - different traces
  - same traces but different analysis
- Analyze traces that exceed local disk space



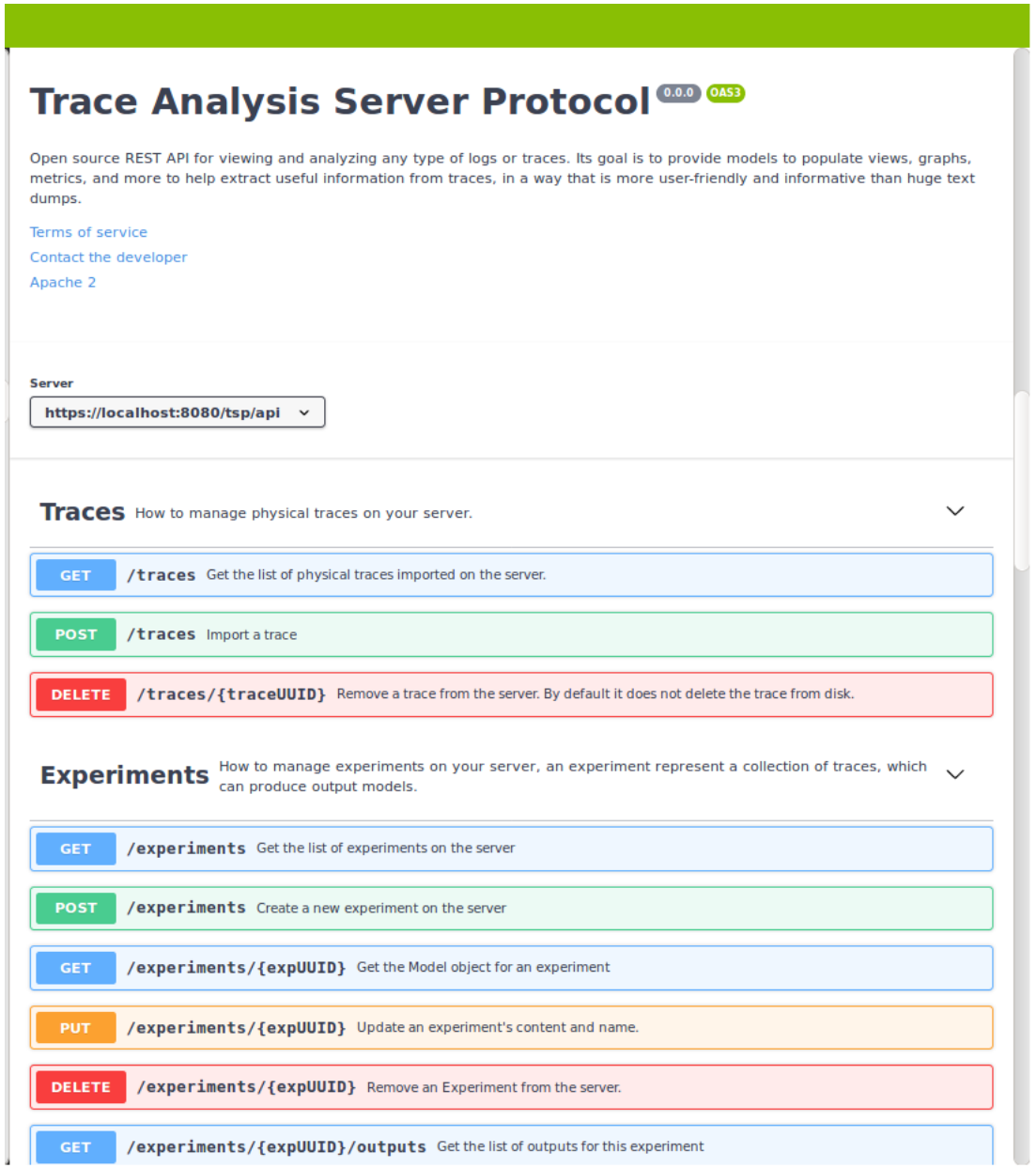
Other opportunities

- Server or client in other programming languages (e.g. Python, Go, R)
- Light-weight / single function trace servers
- Leveraging modern UI technologies
- Thin UI clients

Proof-of-concept

Trace server protocol

- Specification (preliminary)
  - https://github.com/theia-ide/trace-server-protocol
- TSP TypeScript client library
  - https://github.com/theia-ide/tsp-typescript-client



Trace server PoC

- Trace server based on Trace Compass
  - Available in Trace Compass incubator project
  - Eclipse RCP using Trace Compass core components
  - REST over HTTP (JSON)
  - Jersey Rest library

- Framework for IDE-like applications
- Runs in the cloud or on the desktop
- Uses modern web technologies
- Supports multiple languages via LSP
- Supports debugging via DAP
- Extensible (via extension or plug-ins)
- https://www.theia-ide.org/

Trace visualization front-end PoC

- Implemented as extension to Theia
- TSP TypeScript client library
- Timeline-chart library for Gantt-chart type views
  - newly created
- Chart.js for XY-charts
- agGrid for Events table

Research Results

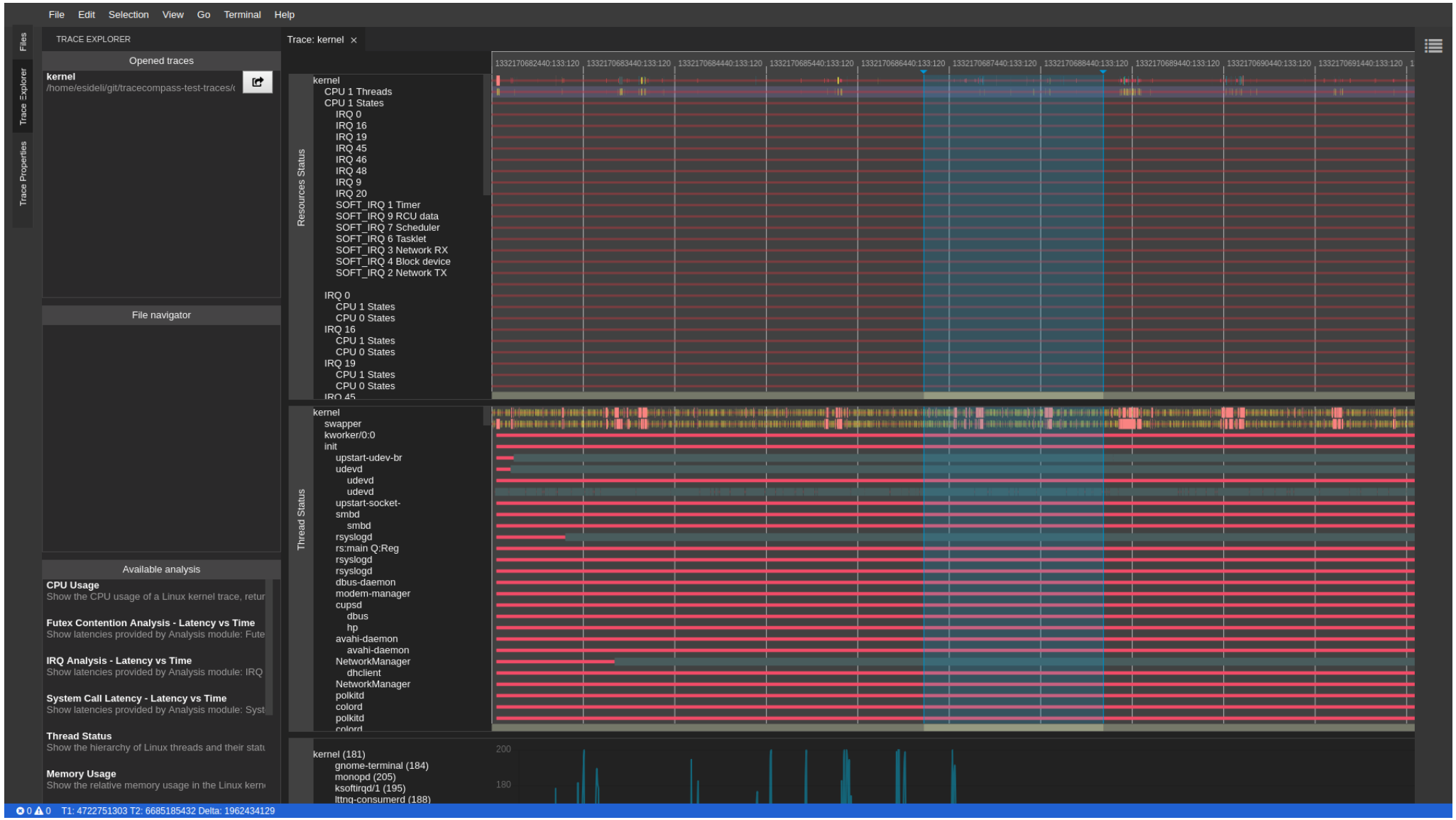
Data Exchange Protocols

- Results for a resolution of 1000 data points
- Time overhead is of the serialization/deserialization/network (localhost) vs the time of just getting the data

		JSON	JSON (Gzip)	Protobuf
XY Charts (1 series)	Response Size (KB)	80	30	30
	Time Overhead (%)	5.2	7.8	4.8
XY Charts (25 series)	Response Size (KB)	800	50	300
	Time Overhead (%)	25.3	34.3	9.2
Time graph row data (25 rows)	Response Size (KB)	1150	100	120
	Time Overhead (%)	9.6	13.3	3.5

Demo

Screenshot


File Edit Selection View Go Terminal Help
Files
Trace Explorer
Trace Properties
TRACE EXPLORER
Opened traces
kernel
/home/esideli/git/tracecompass-test-traces/
File navigator
Available analysis
CPU Usage
Show the CPU usage of a Linux kernel trace, retur
Futex Contention Analysis - Latency vs Time
Show latencies provided by Analysis module: Fute
IRQ Analysis - Latency vs Time
Show latencies provided by Analysis module: IRQ
System Call Latency - Latency vs Time
Show latencies provided by Analysis module: Syst
Thread Status
Show the hierarchy of Linux threads and their statu
Memory Usage
Show the relative memory usage in the Linux kern
Trace: kernel
Resources Status
kernel
CPU 1 Threads
CPU 1 States
IRQ 0
IRQ 16
IRQ 19
IRQ 45
IRQ 46
IRQ 48
IRQ 9
IRQ 20
SOFT_IRQ 1 Timer
SOFT_IRQ 9 RCU data
SOFT_IRQ 7 Scheduler
SOFT_IRQ 6 Tasklet
SOFT_IRQ 3 Network RX
SOFT_IRQ 4 Block device
SOFT_IRQ 2 Network TX
IRQ 0
CPU 1 States
CPU 0 States
IRQ 16
CPU 1 States
CPU 0 States
IRQ 19
CPU 1 States
CPU 0 States
Thread Status
kernel
swapper
kworker/0:0
init
upstart-udev-br
udevd
udevd
udevd
upstart-socket-
smbd
smbd
rsyslogd
rs:main Q:Reg
rsyslogd
rsyslogd
dbus-daemon
modem-manager
cupsd
dbus
hp
avahi-daemon
avahi-daemon
NetworkManager
dhclient
NetworkManager
polkitd
colord
polkitd
colord
kernel (181)
gnome-terminal (184)
monopd (205)
ksoftirqd/1 (195)
lttng-consumerd (188)
200
180
0 0 T1: 4722751303 T2: 6685185432 Delta: 1962434129

Potential Collaborations

- Trace Server Protocol
- Front-end libraries
  - E.g. Time graph (has started)
- Front-end (e.g. Theia)
- Trace server backend

Challenges

- Define a generic TSP and thin front-end
- Efficient data exchange (large amount of data, frequency of updates)
- How to customize for special / domain specific use cases, e.g.
  - Display of custom values
  - Special sorting
  - Server-side filtering
  - Command executions (e.g. follow thread)
- Security aspects

References (prototype repositories)

- Trace server protocol
  - https://github.com/theia-ide/trace-server-protocol
- TypeScript library implementing Protocol
  - https://github.com/theia-ide/tsp-typescript-client
- Timeline-chart library
  - https://github.com/theia-ide/timeline-chart
- Theia integration prototype
  - https://github.com/delislesim/theia-trace-extension

Other References

- Project pages
  - http://tracecompass.org
  - http://projects.eclipse.org/projects/tools.tracecompass
  - http://projects.eclipse.org/projects/tools.tracecompass.incubator
- Documentation
  - Trace Compass User Guide
  - Trace Compass Developer Guide
- Eclipse Theia
  - https://www.theia-ide.org/
  - https://projects.eclipse.org/projects/ecd.theia

Contacts

- Presenter
  - Bernd Hufmann: Bernd.Hufmann@ericsson.com
- Mailing list
  - tracecompass-dev@eclipse.org
- IRC
  - oftc.net #tracecompass
- Mattermost
  - https://mattermost-test.eclipse.org/eclipse/channels/trace-compass