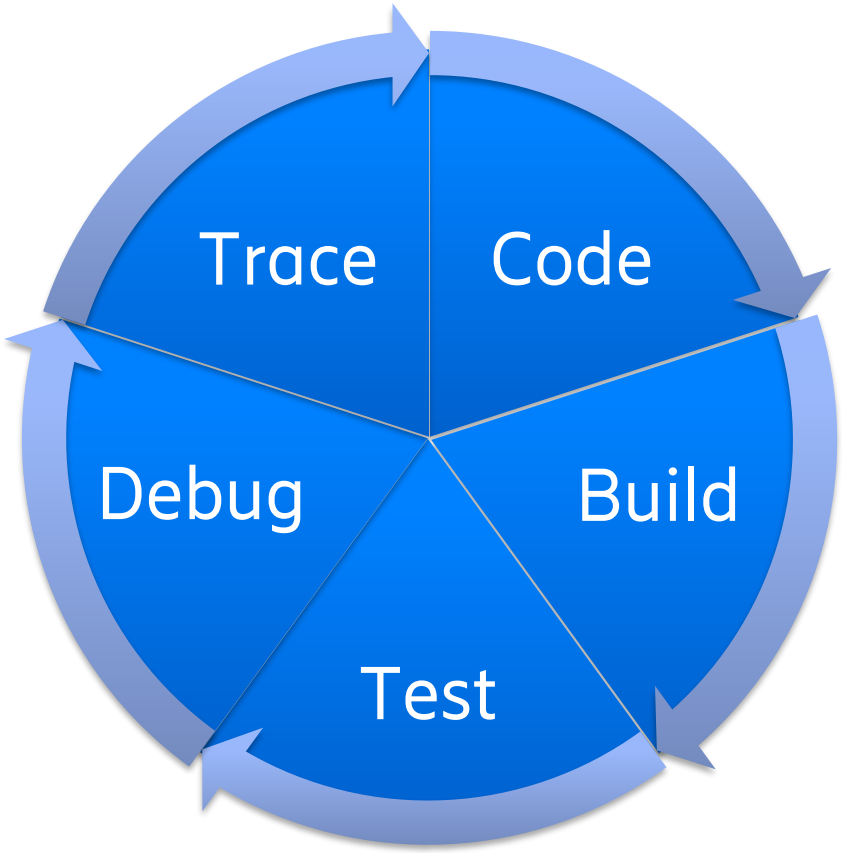
Complete software design loop with Theia and Trace Compass

Bernd Hufmann, Ericsson AB

Agenda

- Background
- Why? What? How? - Tracing inside SW design loop
- Prototypes with Theia
- Demo
- Opportunities
- Q&A

Basic software design loop



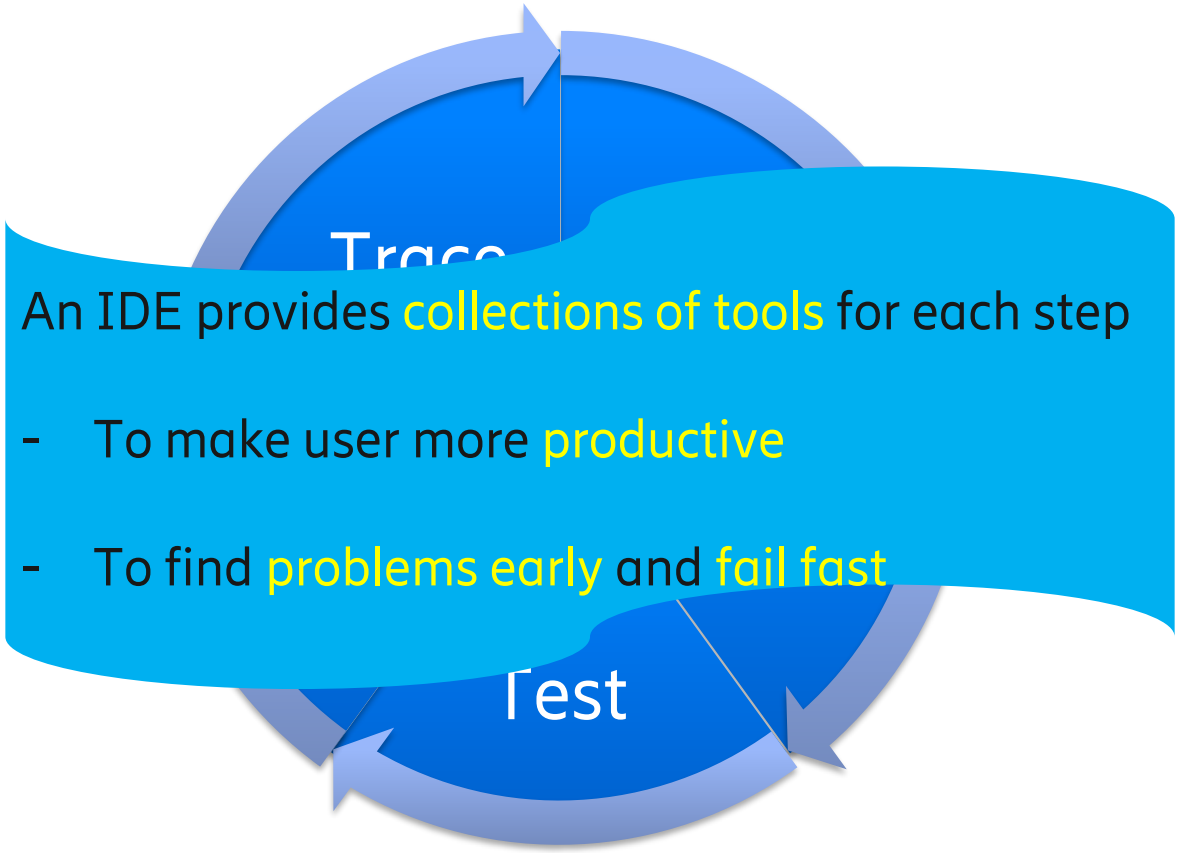
Design and implement new feature

Troubleshoot new feature

Build new feature

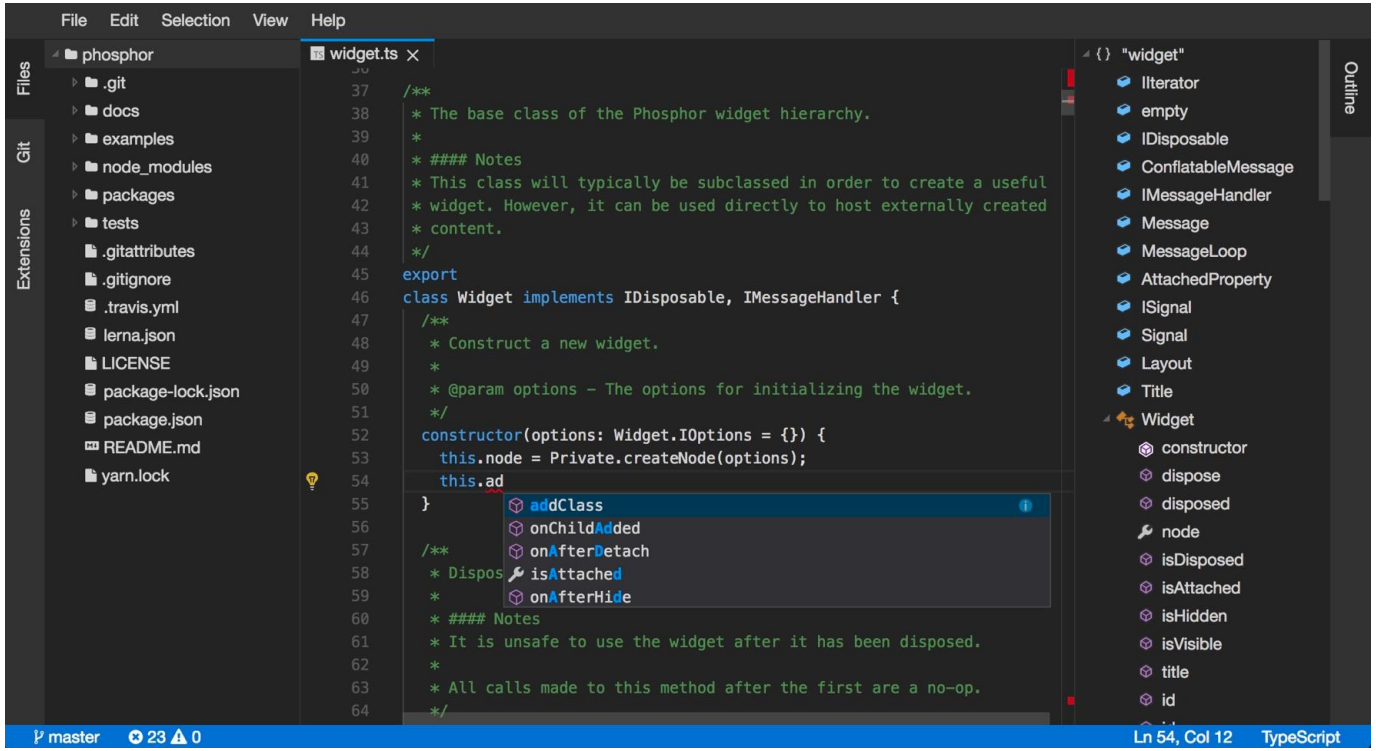
Validate new feature

Integrated software environment (IDE)

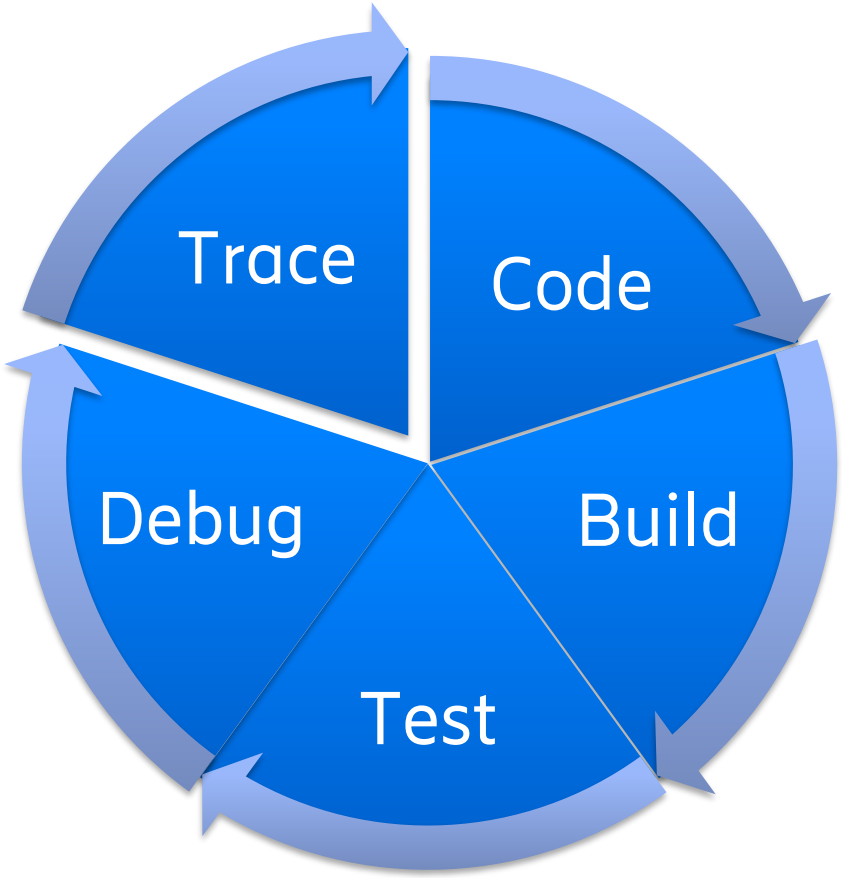


Eclipse Theia

- Extensible platform
- Uses modern web-technologies
- Allows integration with IDE workflow
  - LSP – for language support
  - DAP – for debug support
- Deployment
  - Browser or desktop
  - Workspace managers like Eclipse Che
- Open Source and community
- Vendor neutral

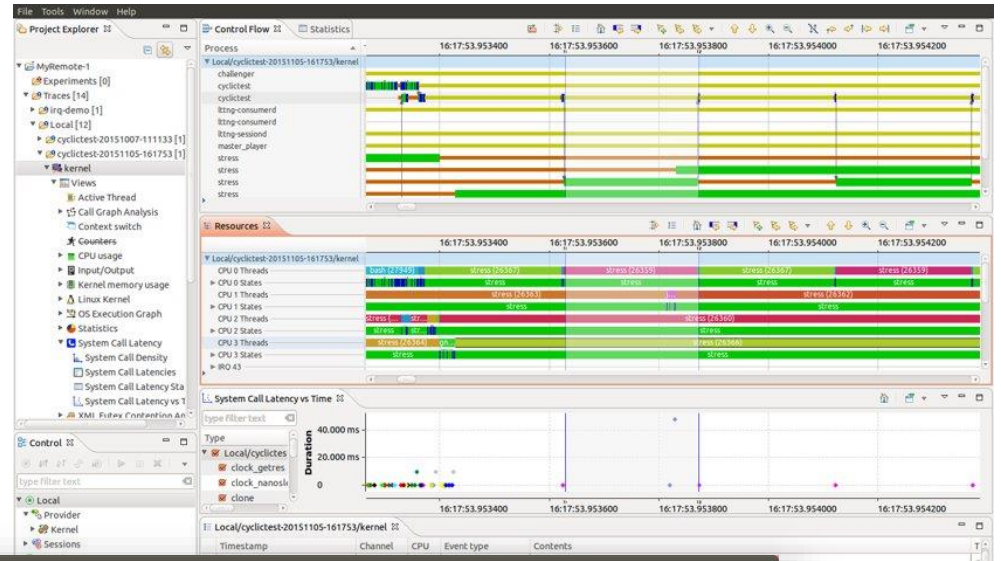


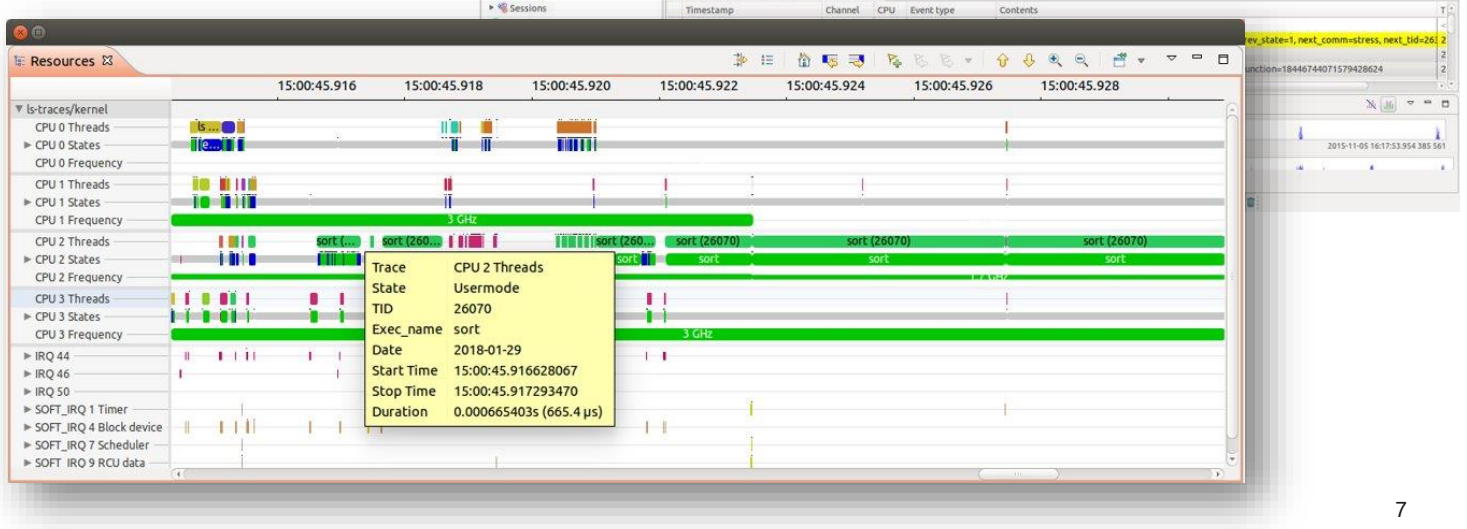
Tracing inside software design loop



What is tracing?

- Trace
  - Series of events over time
  - Event collected at tracepoints during program execution
  - Each event has a time, 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





Tracing inside SW design loop
Motivation

Why?

- Integrated trouble-shooting
- Tracing “at user’s fingertip”
- No need to switch tool
- Find problems early
- Make user more productive

What?

- Trace collection
- Trace analysis
- Trace visualization
- Integration with code editing and debug

How?

- Client-server architecture
- Trace Server Protocol
- Theia front-end
- Trace Server

Tracing inside SW design loop

Why?

- Integrated trouble-shooting
- Tracing “at user’s fingertip“
- No need to switch tool
- Find problems early
- Make user more productive

What?

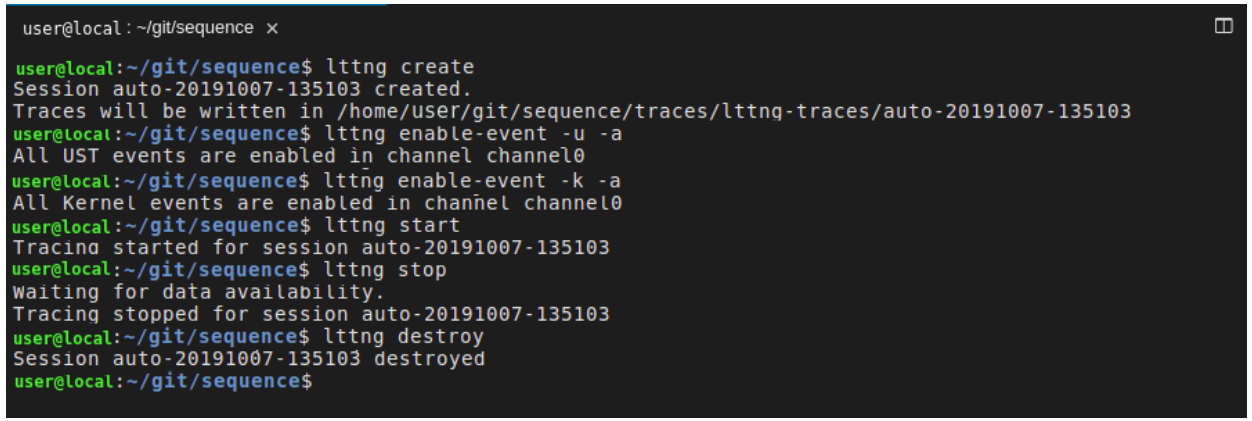
- Trace collection
- Trace analysis
- Trace visualization
- Integration with code editing and debug

How?

- Client-server architecture
- Trace Server Protocol
- Theia front-end
- Trace Server

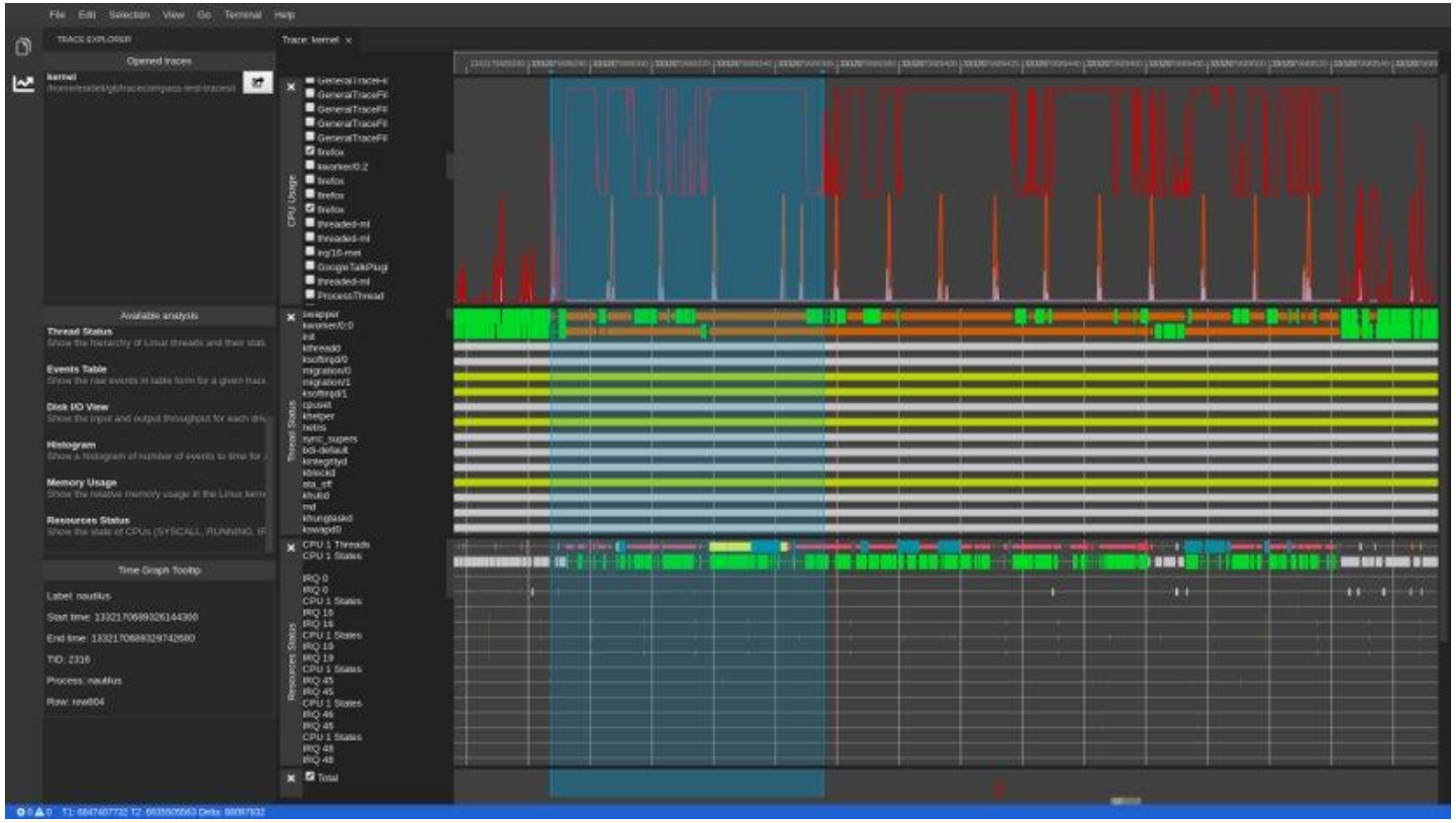
Trace collection

- Configure tracer and run with tracing
- Collect traces and show them in IDE -> Discoverability
- Challenges
  - Configure and trace collection varies from tracer to tracer
    - E.g. LTTng ≠ Perf ≠ Ftrace
  - Local vs remote
- Command-line interface for trace collection
- Flexibility through Theia's terminal

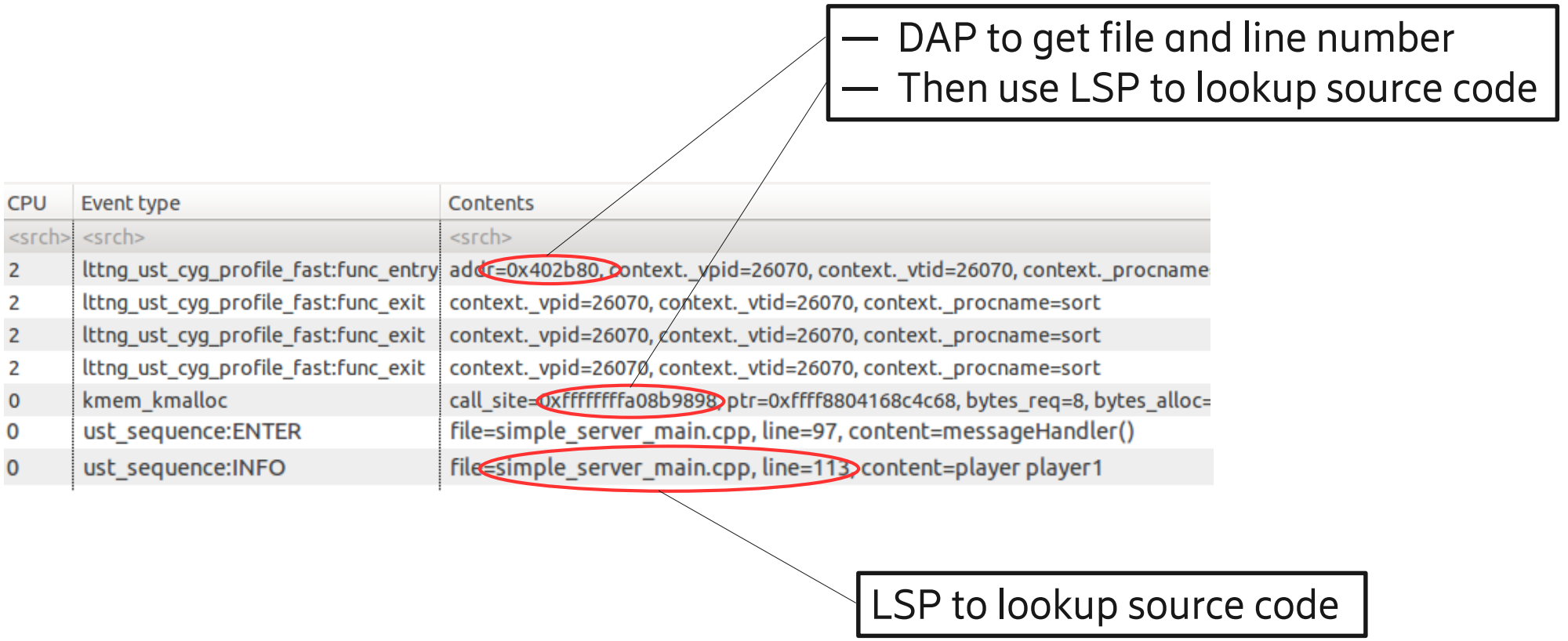


Trace analysis and visualization

- Follow program execution
- Find high resources utilization (memory or CPU)
- Find low resource utilization
  - starvation, deadlock...
- Performance analysis
- Latency analysis
- Profiling
- Statistics
- Regions of interest
- Compare results



Correlation
Leveraging LSP and DAP



Tracing inside SW design loop

Why?

- Integrated trouble-shooting
- Tracing “at user’s fingertip“
- No need to switch tool
- Find problems early
- Make user more productive

What?

- Trace collection
- Trace analysis
- Trace visualization
- Integration with code editing and debug

How?

- Client-server architecture
- Trace Server Protocol
- Theia front-end
- Trace Server

TRACE COMPASS

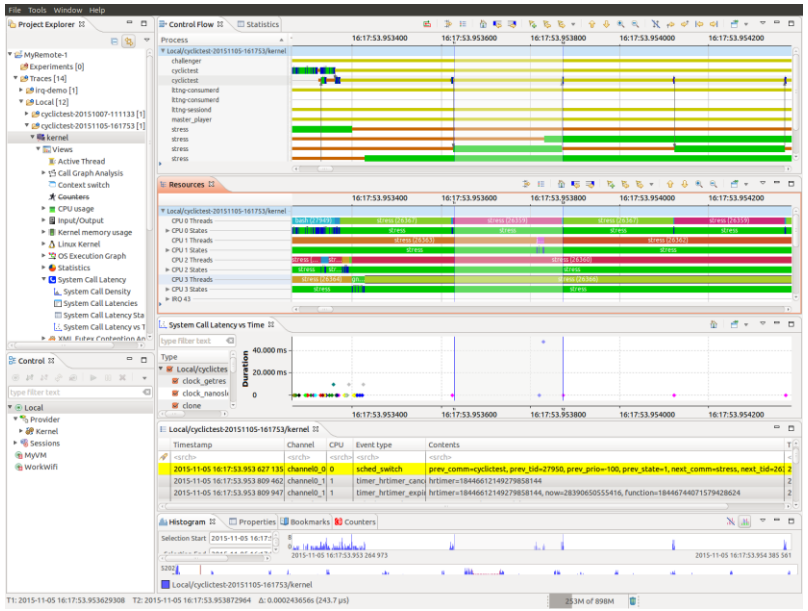
Eclipse Trace Compass is an open source application to solve performance and reliability issues by reading and analyzing traces and logs of a system.

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.

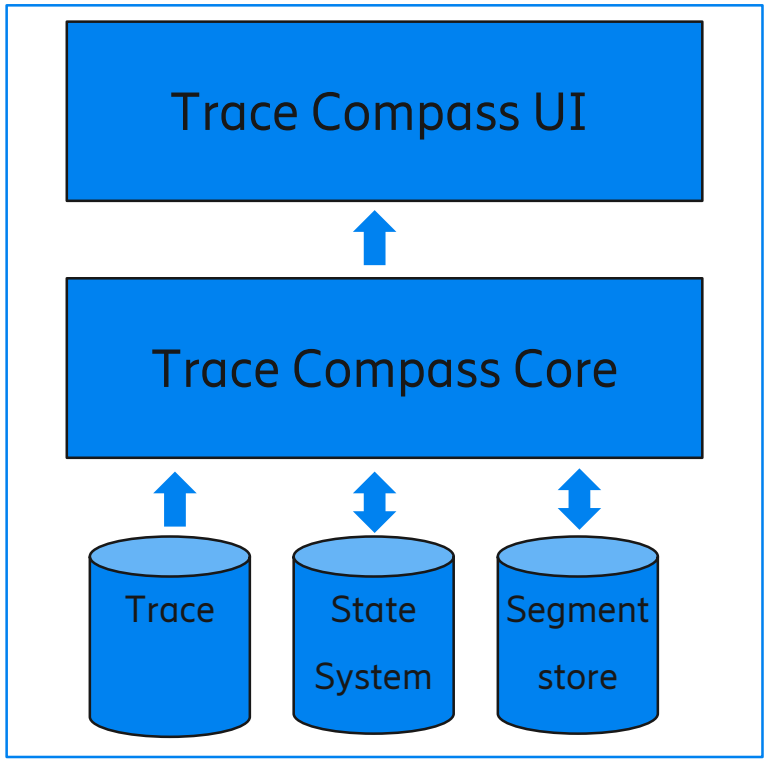
Key characteristics

— Handles trace larger than available memory

— Correlate traces from heterogenous system



Monolithic application

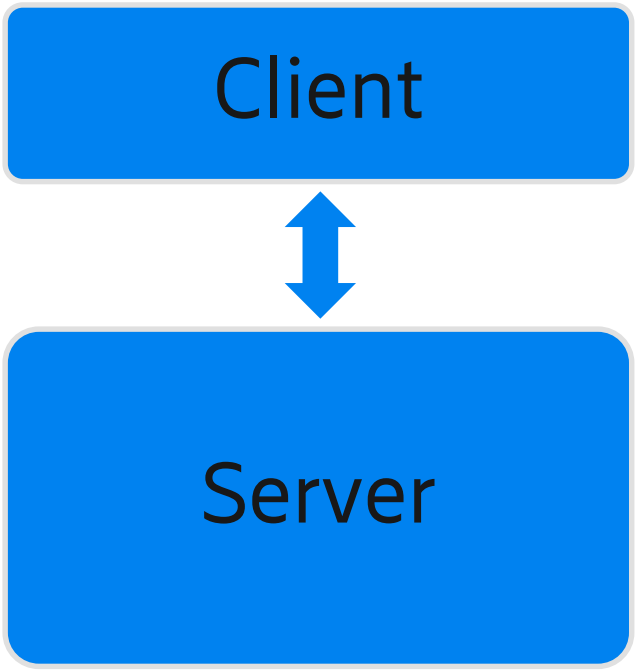


- Standalone RCP
- Plug-ins for Eclipse IDEs

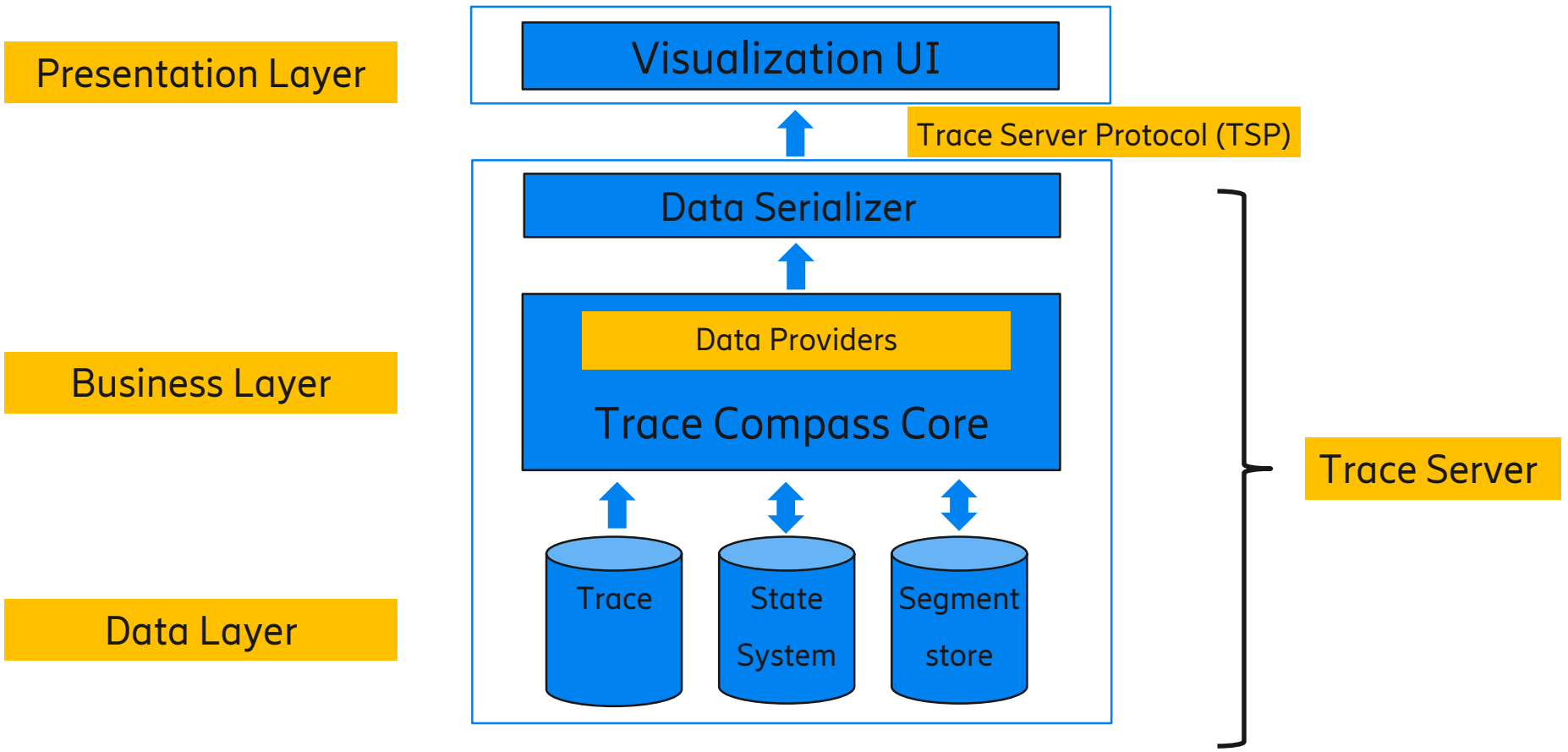
- Challenges
  - Limited to Eclipse
  - Scalability
  - Higher coupling
    - Re-use
    - Maintenance

Towards client-server architecture

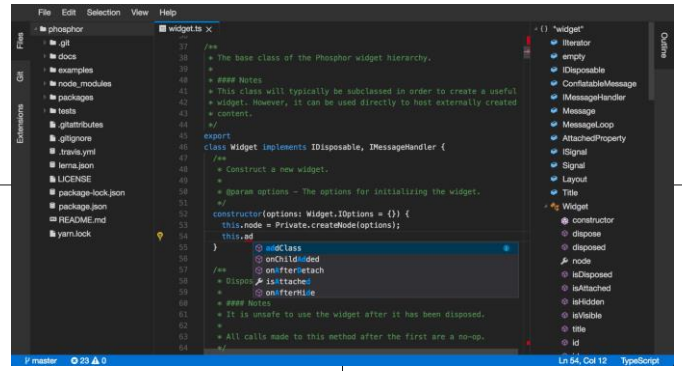
- One trace ≠ one client ≠ one view
  - Different types of clients
  - Different visualizations for same trace
  - Prevent loading entire trace into client
- Domain specific logic is already implemented in Trace Compass
  - Service based solution
  - Avoid re-implementing it in the client
  - Preserve investment made



Client-server architecture



Trace Server Protocol (TSP)



Language Server Protocol (LSP)

Debug Adapter Protocol (DAP)

Trace Server Protocol (TSP)

Language Server

Debug Server

Trace Server

Trace Server Protocol (TSP)

- Protocol to handle communication between backend and frontend
- Allowing traces to reside and be analysed on the backend.
- Exchange visualization data between a client and a server
- Trace management, available data providers, server-side filtering and searching
- https://github.com/theia-ide/trace-server-protocol
- Integration with Theia using tsp-typescript-client
  - TSP ready client to perform your requests
  - Abstract the technology used (REST, HTTP)
  - NPM package available
  - https://github.com/theia-ide/tsp-typescript-client

Trace Server Protocol (TSP)

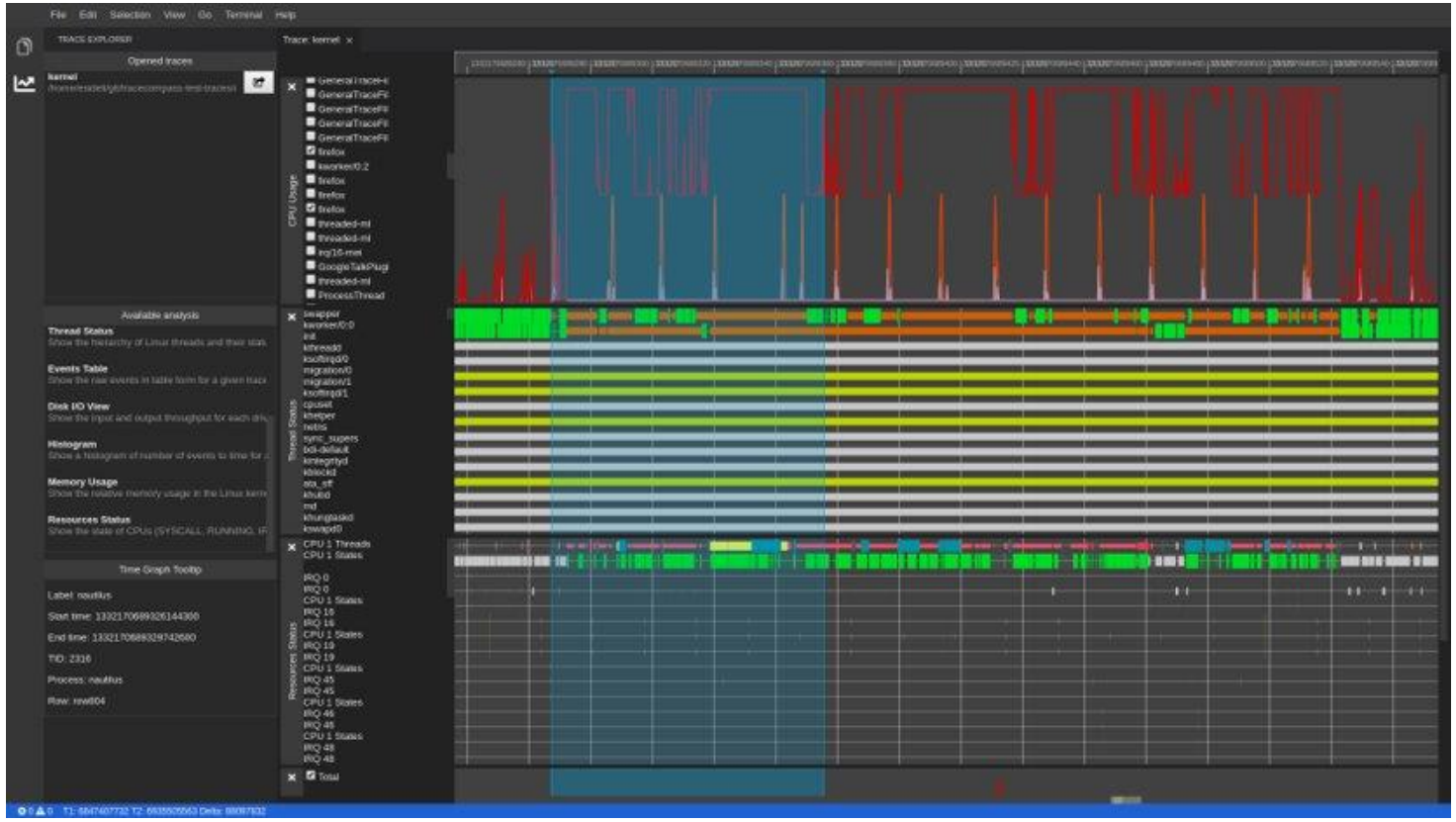
- Protocol to handle communication between backend and frontend
- Allowing traces to reside and be analysed on the backend.
- Exchange visualization data between a client and a server
- Trace management, available data providers, server-side filtering and searching
- https://github.com/theia-ide/trace-server-protocol
- Integration with Theia using tsp-typescript-client
  - TSP ready client to perform your requests
  - Abstract the technology used (REST, HTTP)
  - NPM package available
  - https://github.com/theia-ide/tsp-typescript-client

Disclaimer

- Current implementation ≠ protocol
- It's a REST API
- HTTP protocol

Theia front-end

- Theia based prototype using the TSP
- Prototype available on GitHub (https://github.com/theia-ide/theia-trace-extension
- Opportunity for a new UI/UX
- React
- Chart.js
- agGrid
- New time graph library

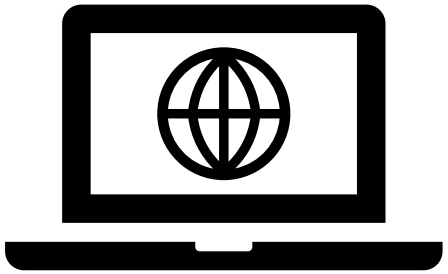


Demo

- Theia C/C++ application
- Theia trace extension installed
- Trace server is running

New opportunities

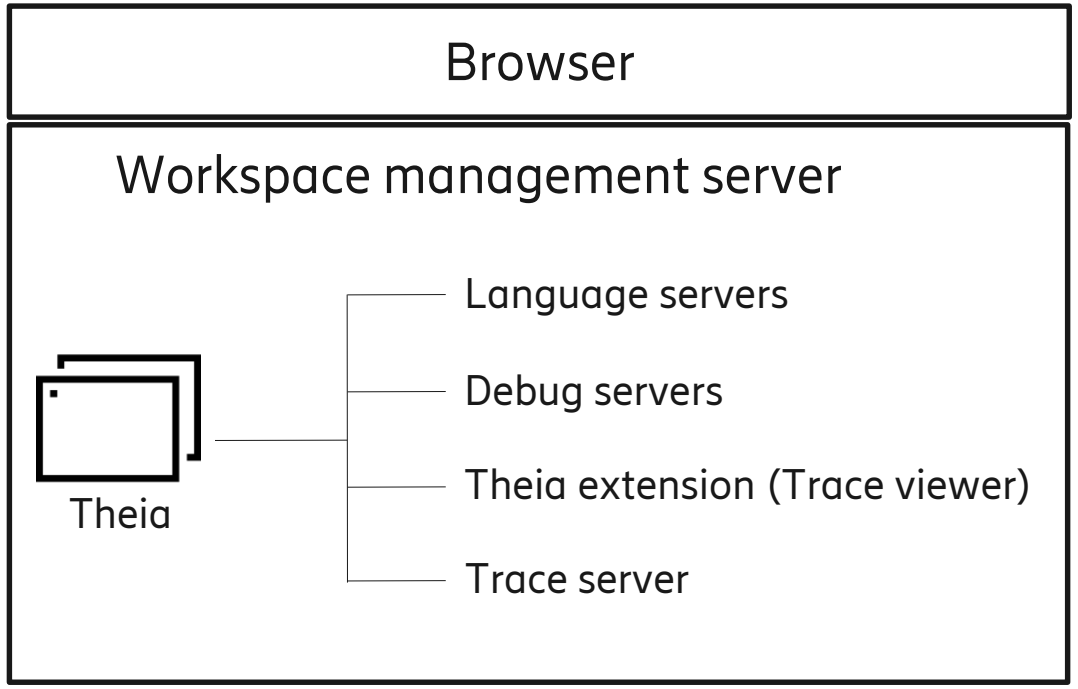
- Integration with other web-based solutions
  - Cloud IDE
  - Dashboards integration (CI, bug reports tools)
- Deployment
  - Single click instead of desktop installation
  - Accessible from various devices
- Leverage modern, state-of the art UI technologies
- Scalability
  - Handle traces larger than local disk
- Increased security



Integration with workspace management applications

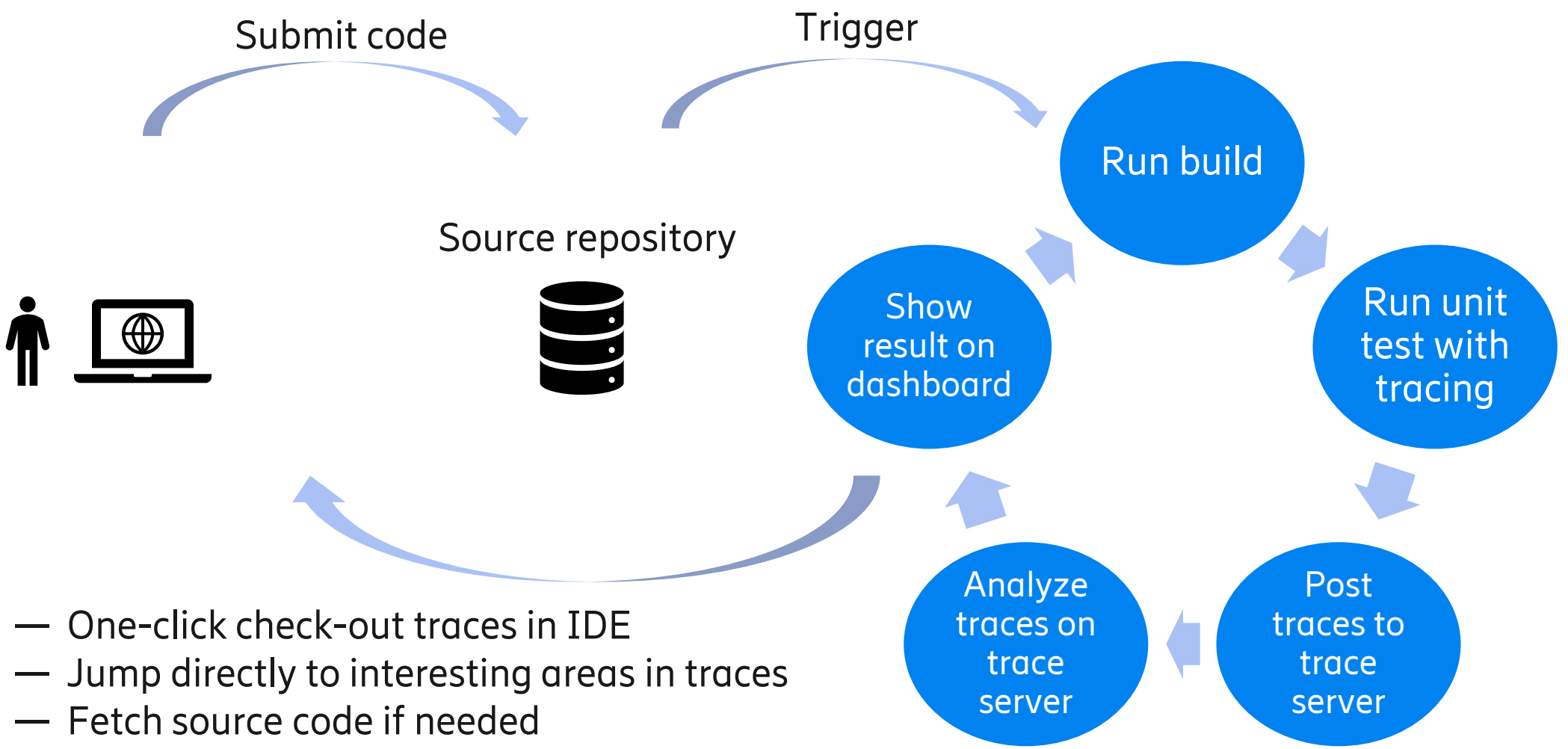
Eclipse Che / GitPod

- Prepare workspaces for trouble-shooting sessions
  - Setup cloud IDE
  - Get source code
    - LSP
  - Setup debuggers
    - DAP
  - Setup trace viewer
    - TSP
- Share trouble-shooting sessions (workspaces)



Dashboard integration

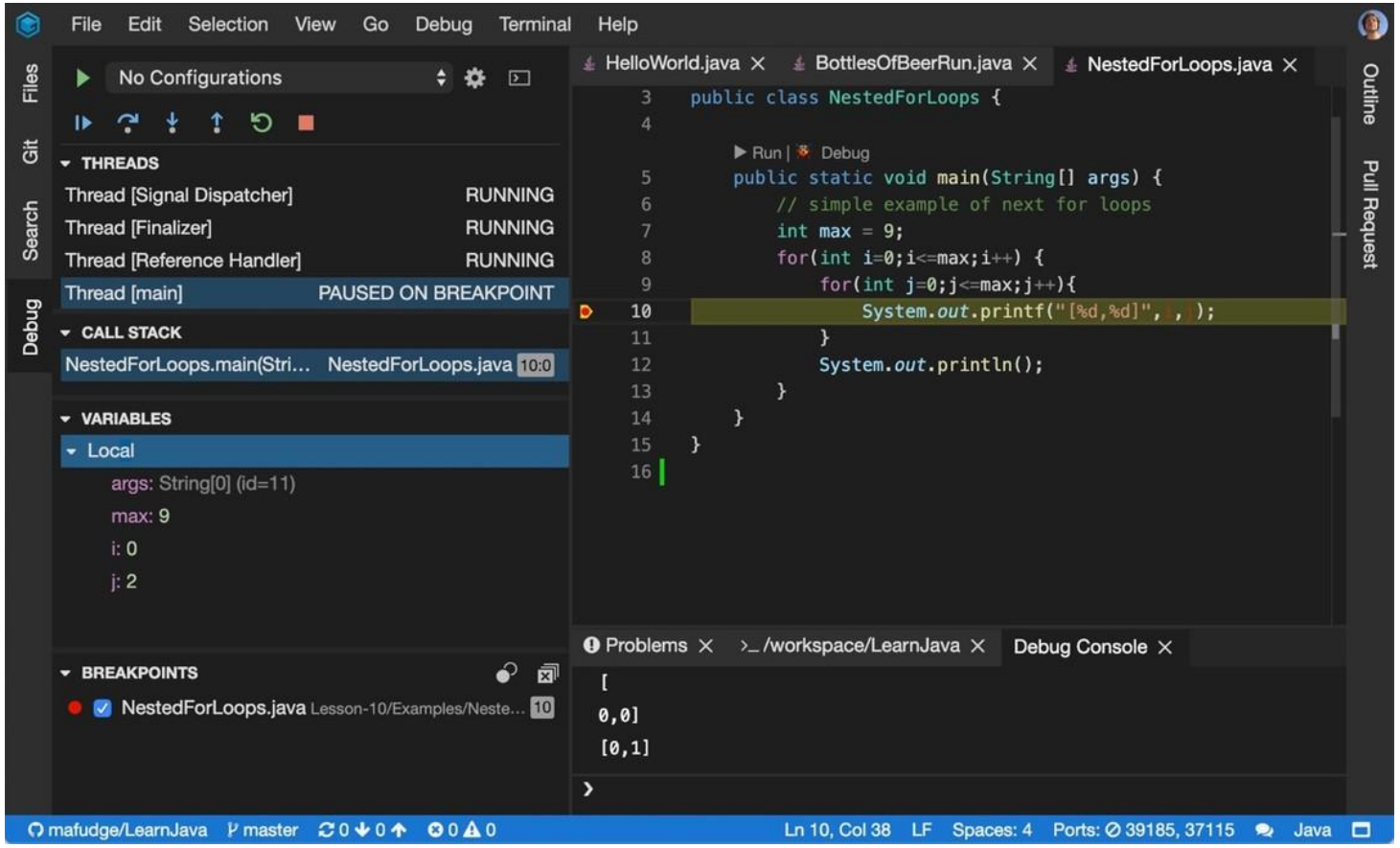
For Example CI or TR tools



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

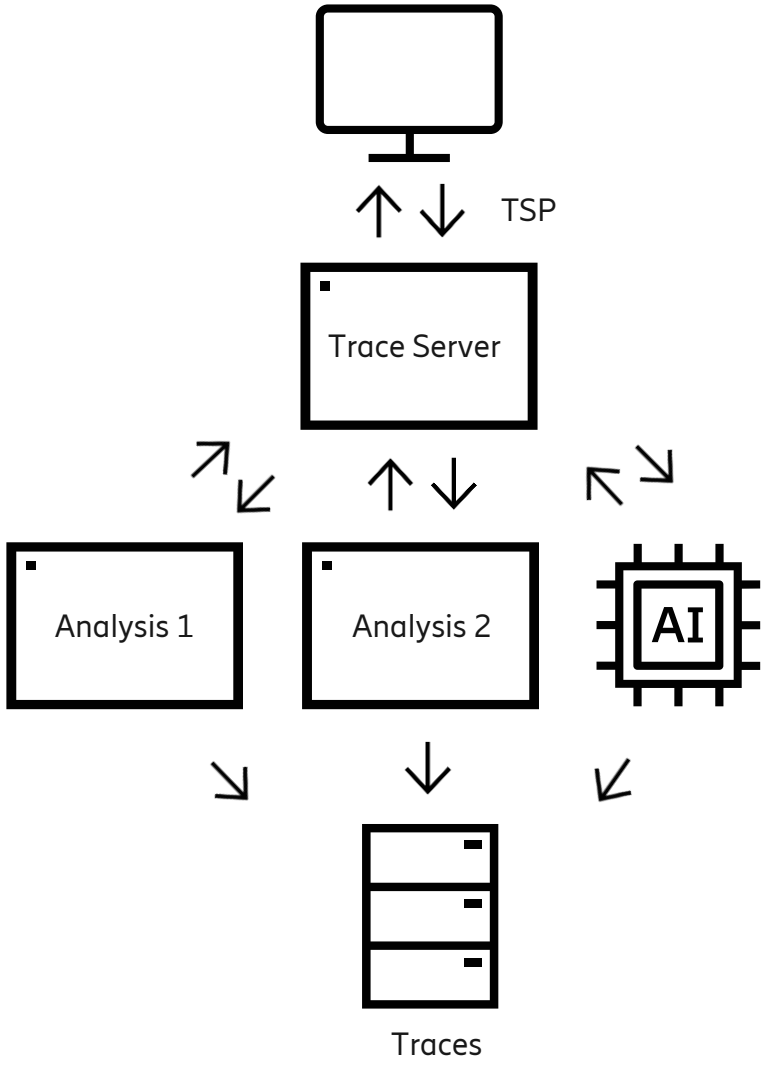
Trace & Debug

- Crash dump analysis
  - Show traces before crash
  - Open core file with debugger
- Navigate source code using traces
  - Using source locations
- Collect traces when hitting breakpoint



Higher scalability

- Enables micro-services
- Distributed architecture
- Parallel, distributed analysis
  - Different traces
  - Same traces, different analysis
- Analyze traces that exceed local disk space



Take-aways

- Tracing is a proven and efficient trouble-shooting technique
- Having it in the SW design loop will help find and understand bugs early
- Client-server architecture allows tracing to be integrated in SW design loop with Theia
- New opportunities open up with the client-server architecture
- Maximal re-use of Trace Compass domain specific logic in trace server

References

- Trace Compass
  - http://tracecompass.org
  - https://projects.eclipse.org/projects/tools.tracecompass
  - https://projects.eclipse.org/projects/tools.tracecompass.incubator
- Trace Server Protocol
  - https://github.com/theia-ide/trace-server-protocol
  - https://github.com/theia-ide/tsp-typescript-client
- Theia frontend prototype
  - https://github.com/theia-ide/theia-trace-extension

Contacts

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