Bringing Mainframe into the Future: OpenTelemetry for Db2 for z/OS

Jørn Thyssen

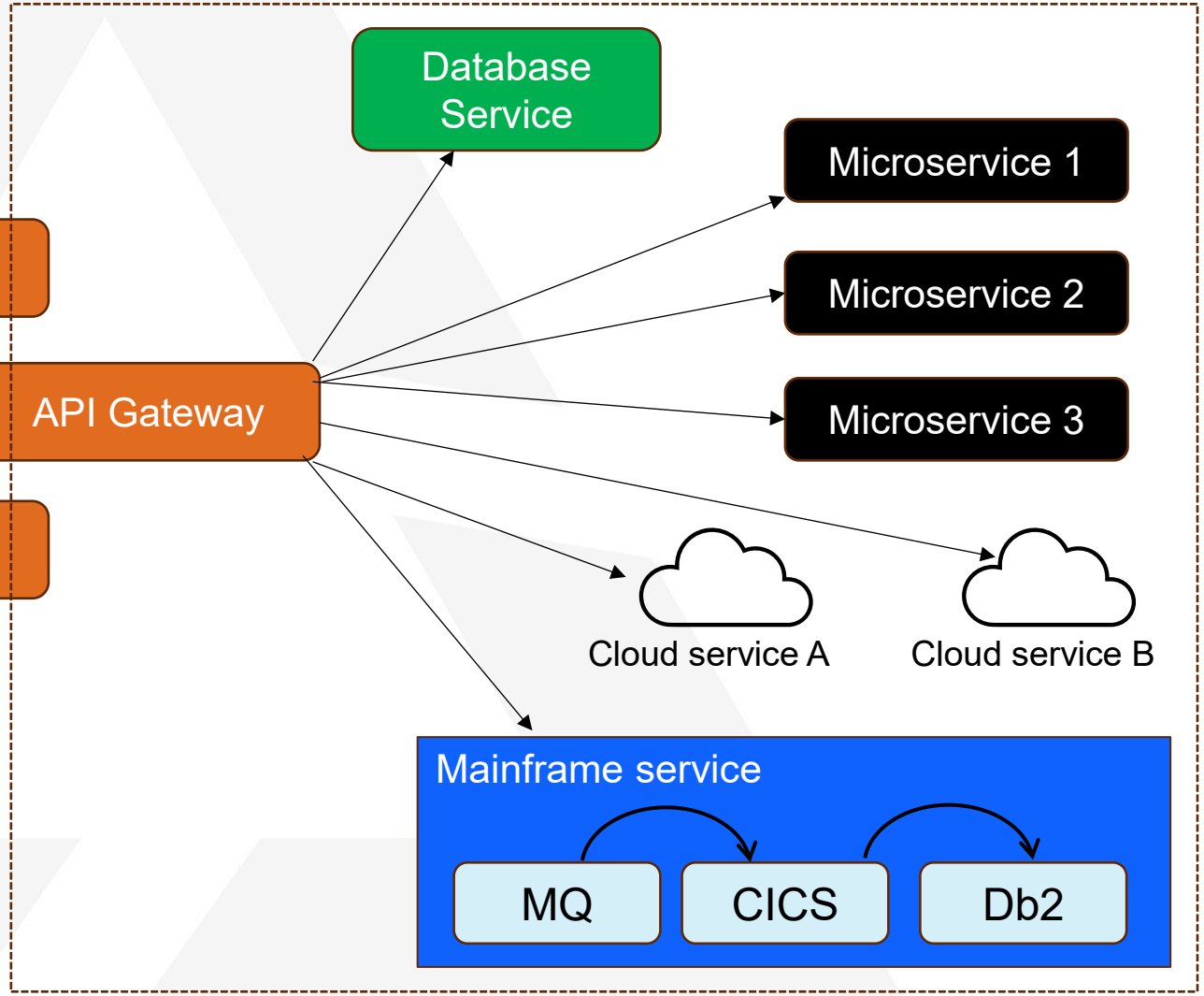
jthyssen@rocketsoftware.com

Senior Technical Staff Member, Rocket Software

Does that look familiar?

Today's application landscape is a mix of API-driven services

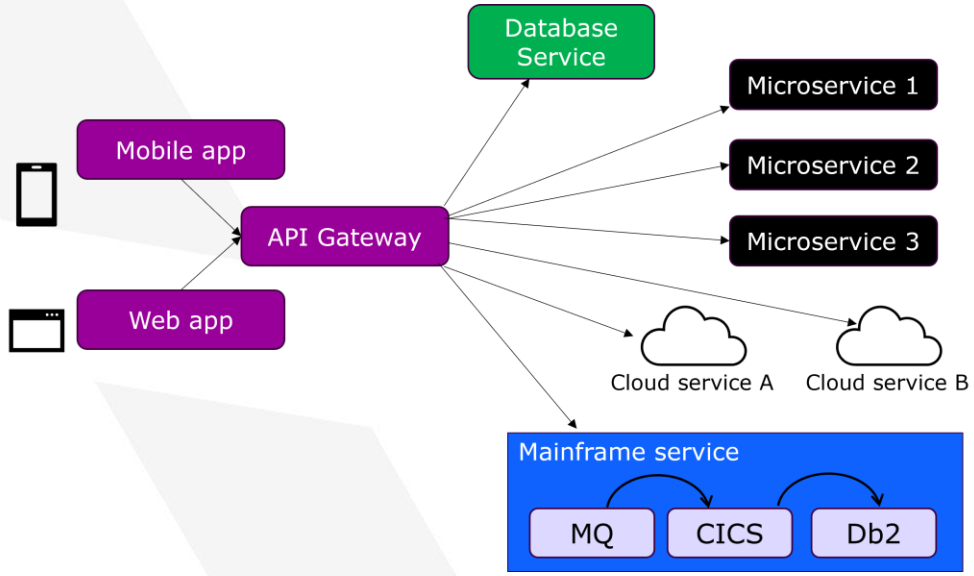
One big black box for users invoking services via mobile or web apps



Different platforms, operating systems, languages, architectural patterns, monitoring, tracing, ...

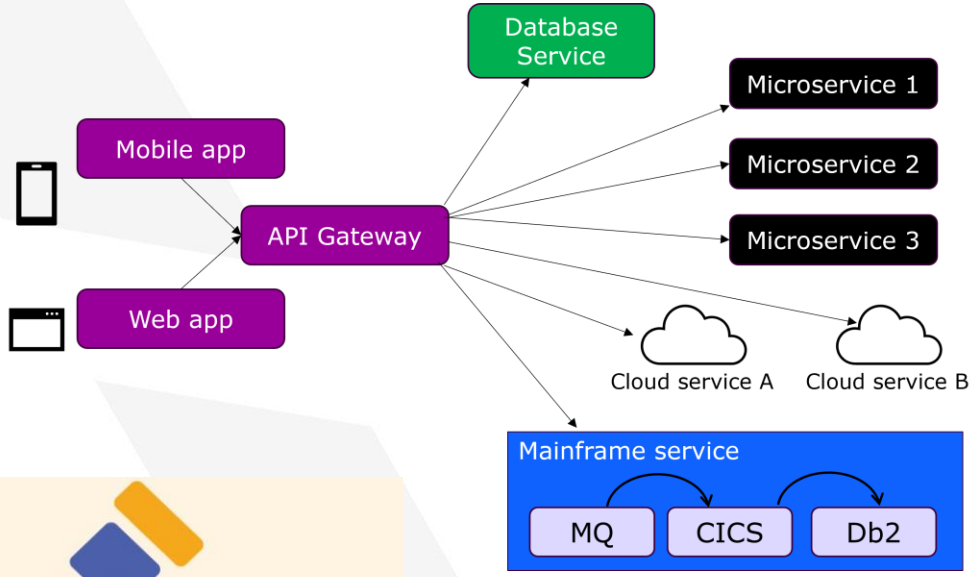
Just a few Challenges

- **Complexity – dozens of services can be involved in a single end user request**
- **Heterogenity of technology stack**
- **Different teams own and run different services**
- **Dynamic infrastructure – containers spin up and down**
- **No centralized visibility – every service can have its own concept of logging, tracing, monitoring, ...**
- **Hard to trace service requests**
- **Risk of blind spots and black boxes without any monitoring and tracing**



Where OpenTelemetry enters the Field

- **Complexity – dozens of services can be involved in a single end user request**
- **Heterogenity of technology stack**
- **Different teams own and run different services**
- **Dynamic infrastructure – containers spin up and down**
- **No centralized visibility – every service can have its own concept of logging, tracing, monitoring, ...**
- **Hard to trace service requests**
- **Risk of blind spots and black boxes without ar monitoring and tracing**



OpenTelemetry

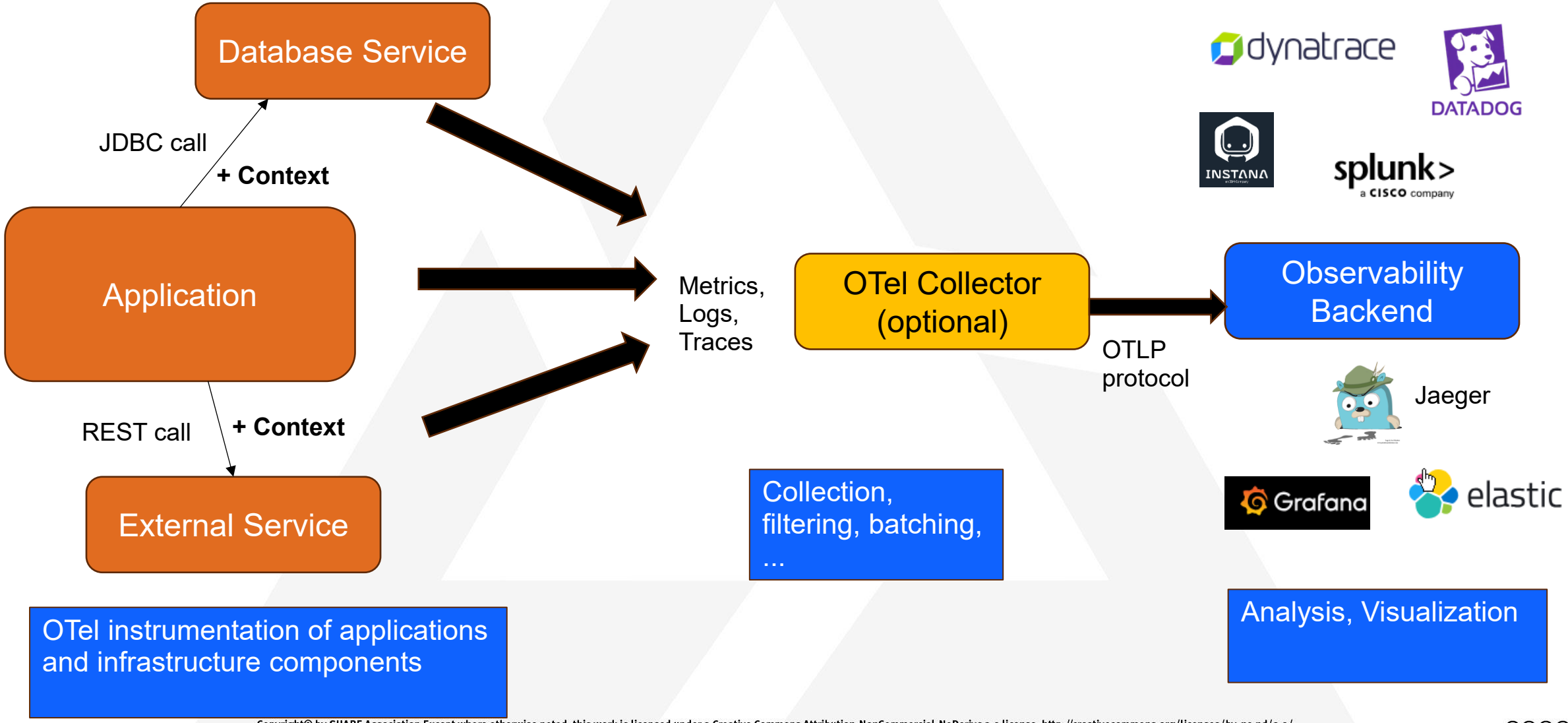
What is OpenTelemetry?

- **Vendor-neutral framework for IT observability**
- **Defines standards for collection and processing of metrics, traces, and log data**
- **Supports cloud, hybrid-, on premise architectures**
- **Defines OTLP, a protocol for telemetry data exchange**
- **Supported by Open Source community, but also by vendor ecosystem**
- **Defines APIs and maintains SDKs to empower user applications for OpenTelemetry**

Benefits of OpenTelemetry

- **Almost no coding required for existing applications to enable OTel instrumentation**
  - Named „zero-code instrumentation“
- **Moderate effort to add tailored instrumentation to an application**
  - SDK/API support for Java, Go, Python, .NET, PHP, Ruby, ...
- **Choice of open source tools and vendor tools for collection, visualization, and analysis**
- **OTel‘s „Trace“ concept allows end-to-end tracing and monitoring across different services and platforms**
- **Does not replace, but complement existing vendor solutions for platform-specific monitoring and observability**

OpenTelemetry in Action



OpenTelemetry – What can be observed?

Metrics

- Numerical data points collected over time
- Gathered from periodic snapshots

Logs

- Text records, either structured or unstructured, which have a timestamp and optional metadata

Traces

- **Track the flow of an application** across distributed systems (or even inside an application)
- **Spans** generated by instrumented applications and components to support distributed tracing

Focus in Db2 context

Baggage

- Contextual information (key/value format) that can be passed across services (typically used in a trace context)

Configuring SMF and the z/OS OTel Emitter

SMF:

- Recording for SMF record types 1157 (third party vendors), 1158 (MQ), 1159 (CICS), 1160 (IMS and IMS Connect), 1161 (Db2) must be active
- In-Memory logstream
- RACF-protected – z/OS OTel emitter STC needs READ access
- All producers – such as Db2 – responsible for accuracy and correctness of generated OTel records

z/OS OTel Emitter:

- Part of z/OS Data Gather (no separate license required)
- Java application (executables shipped in JAR files for USS), requires Java 17 or 21
- Documentation is very basic – see README file in USS
- Configurable:
  - Collector endpoint (=destination of trace records)
  - SMF read buffer size
  - Option for dumping SMF records to STDOUT
  - TLS properties

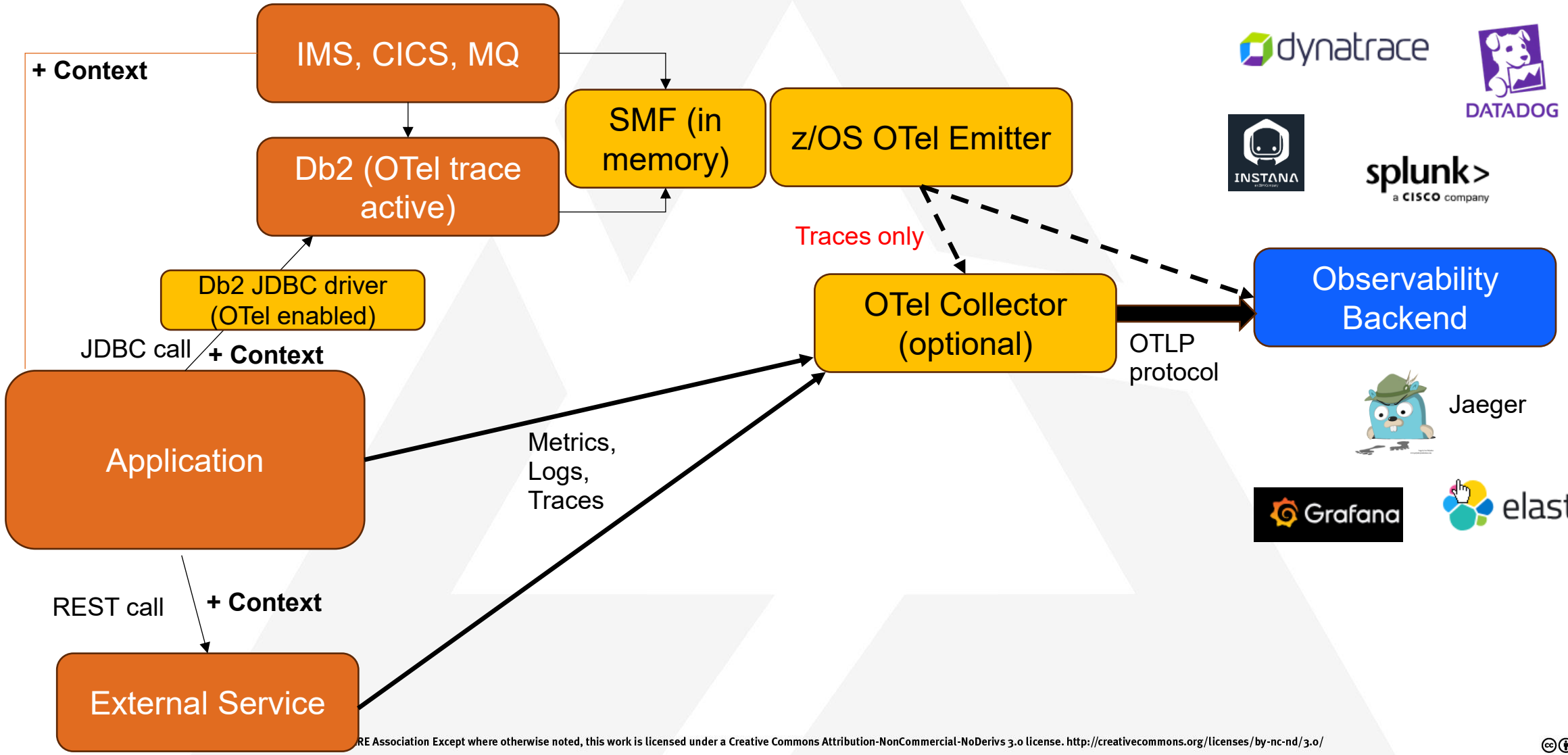
OpenTelemetry and IBM Z

- October 2025: initial support for OpenTelemetry in z/OS 3.2 and 3.1 available (OA66345)
- SMF in-memory logstreams for new record types 1157 to 1161
- New Java-based „z/OS OTel Emitter“ STC (part of z/OS Data Gatherer)
  - Very basic documentation (README file in USS)
- Reads OpenTelemetry trace records - generated by Db2, IMS, CICS, MQ – from SMF and sends them via OTLP to OpenTelemetry collector
- Subsystems (such as Db2) responsible for accurate generation of trace records

Distributed Tracing across IBM z/OS Subsystems

- IBM Z is no more a „black box“ from distributed perspective
- CICS, IMS, CICS Transaction Gateway, MQ, Db2 now can generate their own OpenTelemetry trace records
  - Minimum version level or PTF level for z/OS subsystems required
- Clients (instrumented applications) **must provide trace context** information when calling z/OS applications and services
- z/OS Subsystems can propagate trace context
  - Example: MQ → CICS → Db2
- End-to-end tracing from distributed platforms to IBM Z and inside z/OS subsystems

OpenTelemetry, IBM Z and Db2 for z/OS

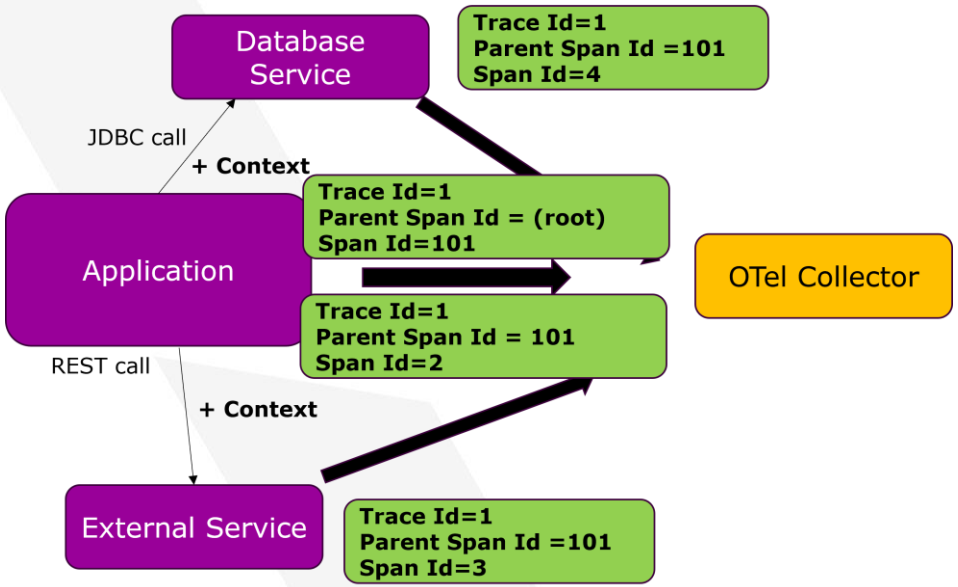


Db2 for z/OS Considerations

- **Db2 for z/OS requires APAR PH67971 and PH68073 for OTel support**
  - Db2 13 only, all function levels
- **New command „-START OTEL EMIT(YES)“ activates generation of OTel trace records to SMF**
- **Records are written when**
  - Db2 receives OTel trace context („context propagation“) from supported client application AND
  - unit of work is committed
- Accounting trace classes 1 and 2 should be active to provide Db2 metrics in OTel trace record
- Context propagation delivered by JDBC T4, Db2 native REST, z/OS Connect REST, CICS-Db2 attachment, CICS liberty-Db2 (JDBC T2), IMS-Db2 attachment, IMS-Db2 Java JDBC T2 environments

OTel Contexts, Traces and Spans

- All OTel instrumented components send their data to the collector **independent** of each other
- "**Context**" is required for correlation and visualization
- OTel Traces are split into "**Spans**"
- Every span represents a unit of work or a unit of operation
- Applications create (and terminate) Spans
- **Context propagation** required to link all spans together to a Trace

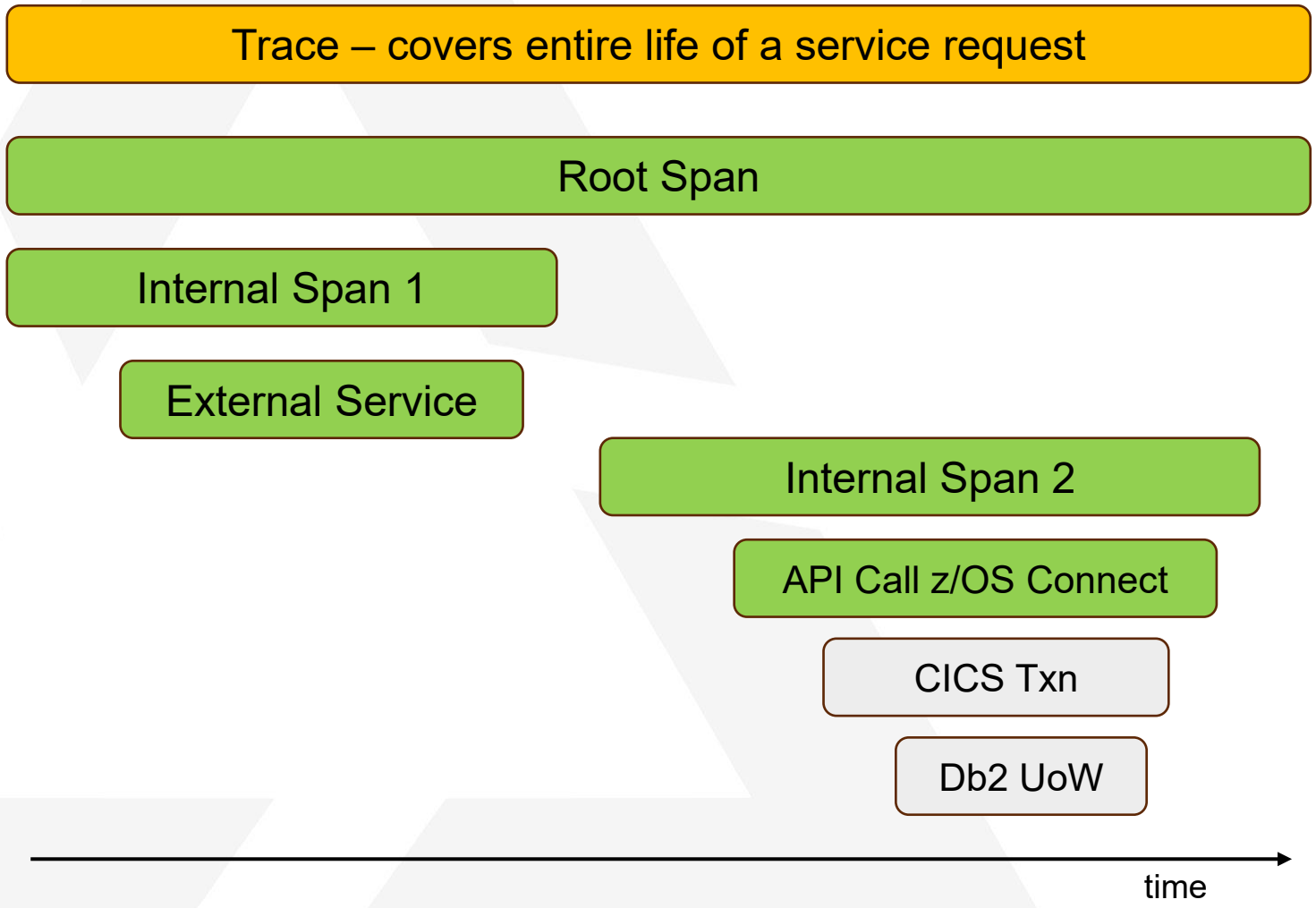


OTel Contexts, Traces and Spans - Example

All spans share same trace id

Every span:

- Has a parent span (or is a root span)
- Has a unique span id
- Has a timestamp and a metered execution time (duration)
- Can have additional attributes (application specific)



Example for a Db2 Span – Generated per UoW


```
"traceID": "04db42ec0272e2862f508b53f577f448",
   "spanID": "e23fe8d4667ad30a",
   "operationName": "Db2 unit of work",
   "references": [
       {
           "refType": "CHILD_OF",
           "traceID": "04db42ec0272e2862f508b53f577f448",
           "spanID": "24b8f5f1a981a9a5"
       }
   ],
   "startTime": 1771237072816004,
   "duration": 13500974,
   "tags": [
       {
           "key": "db.ibm.db2.connection.id",
           "type": "string",
           "value": "SERVER"
       },
       {
           "key": "db.ibm.db2.connection.type",
           "type": "string",
           "value": "DIST"
       },
       {
           "key": "db.ibm.db2.correlation.id",
           "type": "string",
           "value": "SampleQueries_2"
       },
       {
           "key": "db.ibm.db2.cpu.gp",
           "type": "int64",
           "value": 311396
       },
```


ID of Parent Span

Time spent in Db2

Db2-provided information (span attributes)

CPU time on GCP in microseconds


```
{   "key": "db.ibm.db2.cpu.ziip",
    "type": "int64",
    "value": 592503
},
{
    "key": "db.ibm.db2.et",
    "type": "int64",
    "value": 6832684
},
{
    "key": "db.ibm.db2.group.name",
    "type": "string",
    "value": "IDS2"
},
{
    "key": "db.ibm.db2.location",
    "type": "string",
    "value": "RS01IDS2"
},
{
    "key": "db.ibm.db2.luwid",
    "type": "string",
    "value": "GA11E103.G409.E23FE8CFE0C7.0004"
},
{
    "key": "db.ibm.db2.member.name",
    "type": "string",
    "value": "I9A2"
},
{
    "key": "db.ibm.db2.planname",
    "type": "string",
    "value": "DISTSERV"
},...
```


zIIP time in microseconds

Elapsed time in microseconds

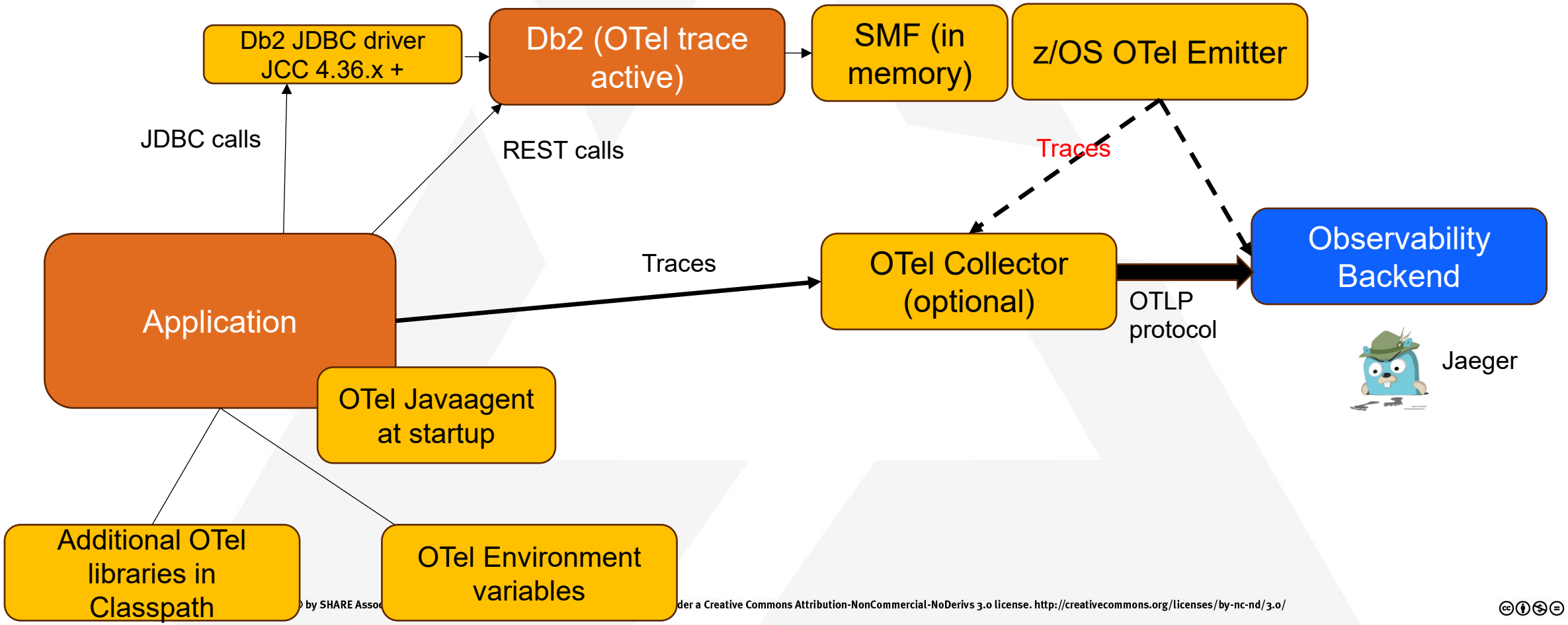
Logical Unit of Work ID

Enabling OTel – Application Perspective

- **Applications must be OTel-instrumented and provide context propagation for Db2 calls**
  - JDBC: Handled by Db2 JCC driver, Db2 Connect V12.1.3 required (JCC architecture level 4.36.x and higher)
- **Zero-code instrumentation:** enables OTel without any code change
  - Requires additional libraries plus configuration changes
  - Support for applications written in Java, JavaScript, Python, PHP, .NET, Go (other languages with some limitations)
- **Manual instrumentation** requires OTel language-specific SDK
  - Consider OTel best practices and semantic conventions

Zero-code Instrumentation – Java Example

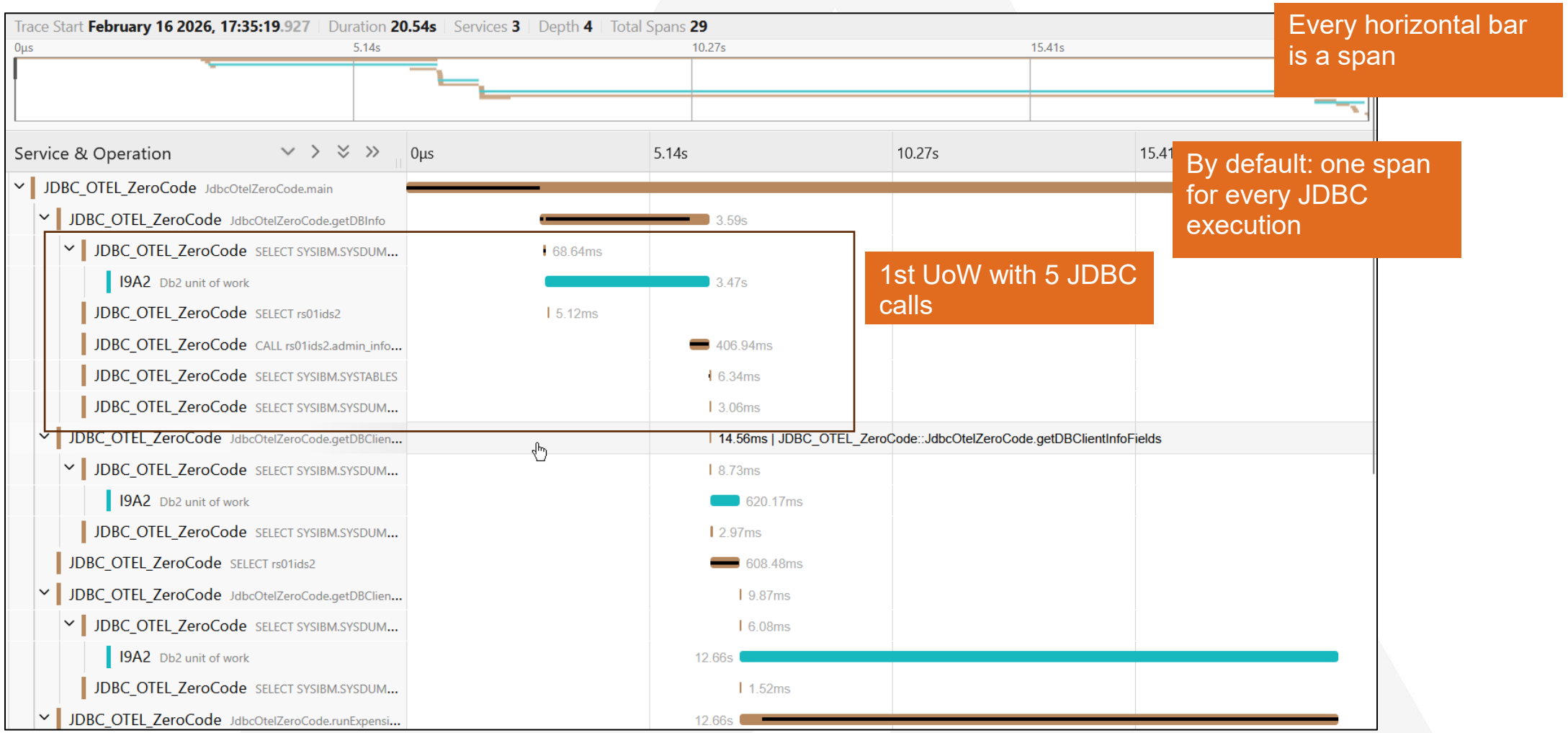
Consider an existing Java application with JDBC calls to Db2 for z/OS and native Db2 REST services



Zero-code Instrumentation – Java Example

- Minimum set of OTel libraries must be added to Java classpath
  - Downloadable from Github
- OTel‘s Javaagent must be specified in JAVA_TOOL_OPTIONS
- Additional environment variables required:
  - OTEL_TRACES_EXPORTER: which protocol to use for trace export (e.g. OTLP)
  - OTEL_EXPORTER_OTLP_ENDPOINT: hostname/port of collector or observability backend
  - OTEL_SERVICE_NAME: meaningful name to locate traces in observability backend
  - OTEL_INSTRUMENTATION_METHODS_INCLUDE: optional – a list of the Java program‘s method for which OTel instrumentation should be activated

Result in JaegerUI (Zero-code Instrumentation)



Manual (or Code-based) Instrumentation

- Requires OpenTelemetry SDK for instrumented application (free download from Github)
- Full control of span creation and hierarchy
- Can provide application-specific attributes to spans
- Developer can align spans to Db2 units of work
- Important: application MUST provide a trace context before Db2 service is called

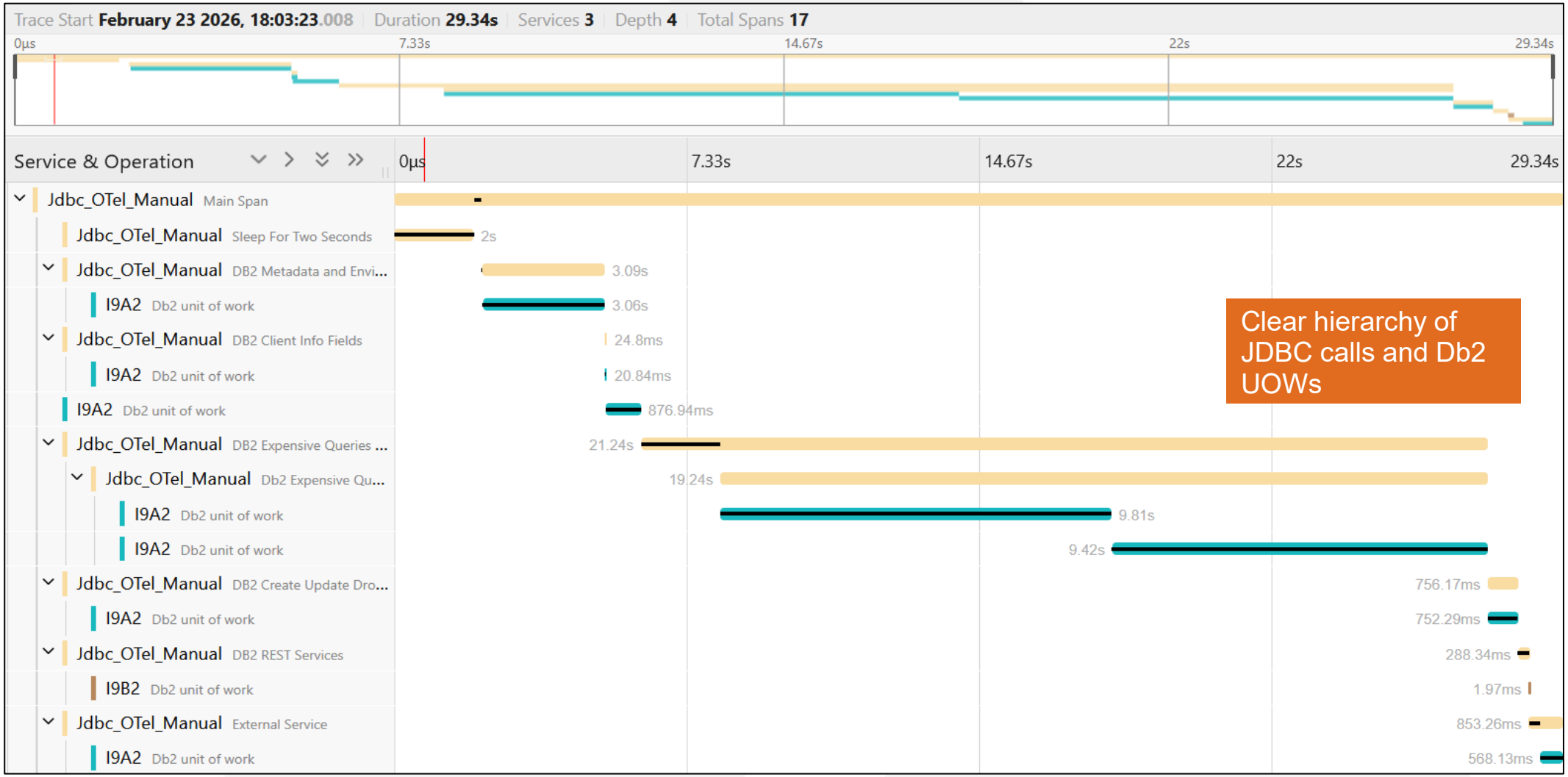
```
Span getDBClientInfoSpan = tracer.spanBuilder("DB2 Client Info
Fields")
                    .setSpanKind(SpanKind.INTERNAL)
                    .setParent(parentContext)
                    .startSpan();
            try (Scope dbScope =
getDBClientInfoSpan.makeCurrent()) {
                getDBClientInfoFields(con);
                con.commit();
            } catch (Exception e) {
            } finally {
                getDBClientInfoSpan.end();
            }
```

Java example:

Method getDBClientInfoFields runs some Db2 queries

Dedicated span created for this method

Result in JaegerUI

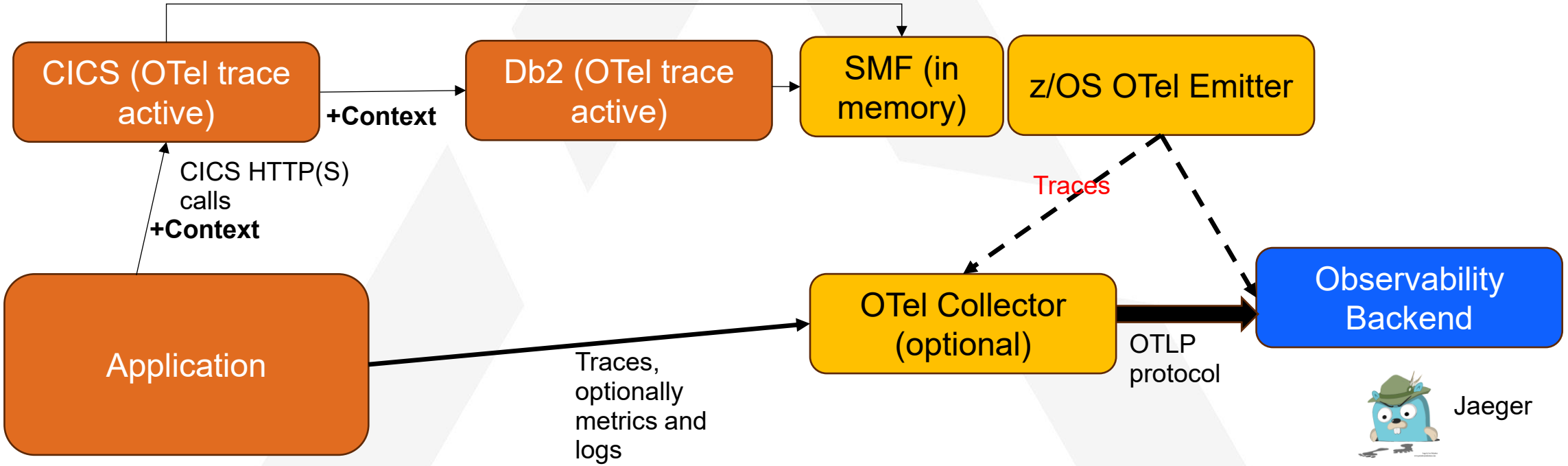


CICS/Db2 example

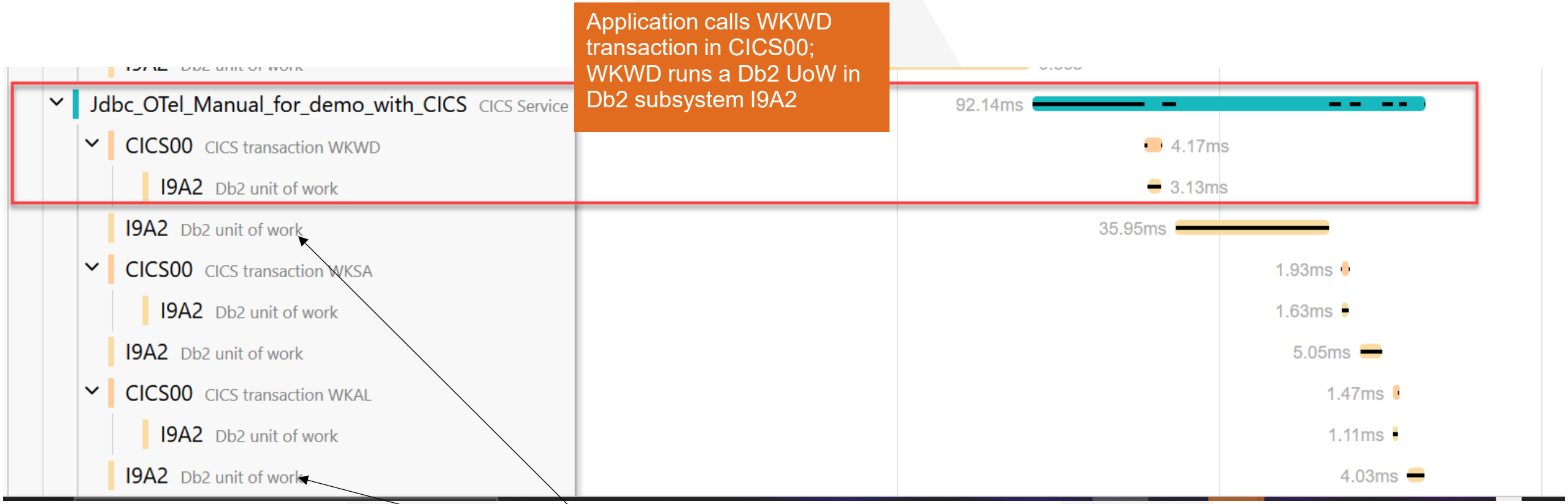
- CICS also supports OTEL from Version 6.3
- Multiple entry point to CICS supported: CICS HTTP, CICS TG, z/OS Connect, ..
- Context propagation: CICS-Db2 attach, CICS Liberty-Db2 attach (JDBC Type 2), …
- CICS offers more granular controls than Db2
  - Propagate yes/no: sending trace context to other subsystems (e.g. Db2)
  - Emit yes/no: generating CICS related spans and emitting them via SMF and z/OS OTel emitter
  - Can be set on region or transaction level

OTel Instrumentation – Java/CICS/Db2 Example

Consider a client application that calls CICS transactions via HTTP interface



Result in JaegerUI



Traditional Db2 Monitoring vs. OpenTelemetry

SHARE
EDUCATE ▸ NETWORK ▸ INFLUENCE

Db2 IFI Instrumentation

- Focused on **metrics**
- Proprietary format
- In-Db2 events only (e.g. Accounting record)
- Can trace SQL flow and related events (getpages, lock requests, I/Os,etc.) with overhead
- No distributed point of view
- Complexity of Db2 IFI handled by vendor products

Db2 OTel Instrumentation

- OTel **traces (with few metrics embedded)**
- OTel trace format
- Contributes to distributed point of view
- Can provide higher level of granularity for basic metrics than accounting rollup records
- Leveraged by Open source tools or vendor products

Considerations and Lessons Learned (1)

- **OTel is no replacement for traditional Db2 Monitoring (yet)**
  - Much more information available from Db2 IFI calls
  - Real time metrics (IFI READS calls) and events (IFI READA calls)
  - OTel: currently very few accounting metrics only – no logs, no Db2 statistics metrics, no events (such as timeouts)
- **Scope of Db2 spans is unit of work**
  - No end-to-end monitoring for single SQL execution (unless UoW contains one SQL only)
  - Application‘s COMMIT strategy drives Db2 span generation
  - OTel can provide more details than traditional accounting record for accounting rollups (DDF and RRSAF)

Considerations and Lessons Learned (2)

- **Currenly only little control on Db2 for z/OS side**
  - Once OTel tracing is on, all OTel instrumented applications can generate Db2 spans
  - Better control on CICS side
- **Time synchronization of all instrumented applications is crucial**
- **Amount of trace records can be a challenge for collection and visualization**
  - Consider OTel sampling
- **Consider instrumentation via Annotations (or manual instrumentation) if zero-code instrumentation is too granular**

Considerations and Lessons Learned (3)

- **Visualization with Zero-code instrumentation can be misleading**
  - Zero-code can create spans for every single JDBC call, but Db2 creates spans per UoW
  - Consider manual instrumentation if accuracy is crucial
- **Not all connection types into Db2 currently supported**
  - Currently, ODBC/CLI is missing
- **According to IBM, only very little overhead for Span generation and emission**
- **OTel is a dynamic topic – many activities are currently going on (OpenSource + vendors)**

Resources

Db2 related blogs:

https://community.ibm.com/community/user/blogs/m-sueli-almeida/2025/09/29/opentelemetry-support-in-db2-for-zos

https://community.ibm.com/community/user/blogs/jrn-thyssen1/2025/12/16/practical-guide-to-opentelemetry-tracing-for-java1

IBM AIOps related podcase:

https://www.buzzsprout.com/2370536/episodes/17544480

OpenTelemetry documentation:

https://opentelemetry.io/

Your feedback is important!

SHARE
EDUCATE • NETWORK • INFLUENCE

Submit a session evaluation for each session you attend:

SHARE mobile app -or- www.share.org/evaluation