Tracing distributed service calls: implementing APM for the JVM

Disclaimer

INSTANA 

I am contracting for the APM vendor Instana and gained most of my experience working with APM while being with the company. In order to discuss what I have been factually working with, I cannot avoid showcasing the tool I helped to make. I am not paid to feature Instana in this presentation.

Outline

I. Inventory
II. Tracing (micro-)services
III. Implementing APM
IV. Advanced topics

Where we are coming from: the “distributed monolith”.

Where we are coming from: the “distributed monolith”.

Where we are coming from: the “distributed monolith”.

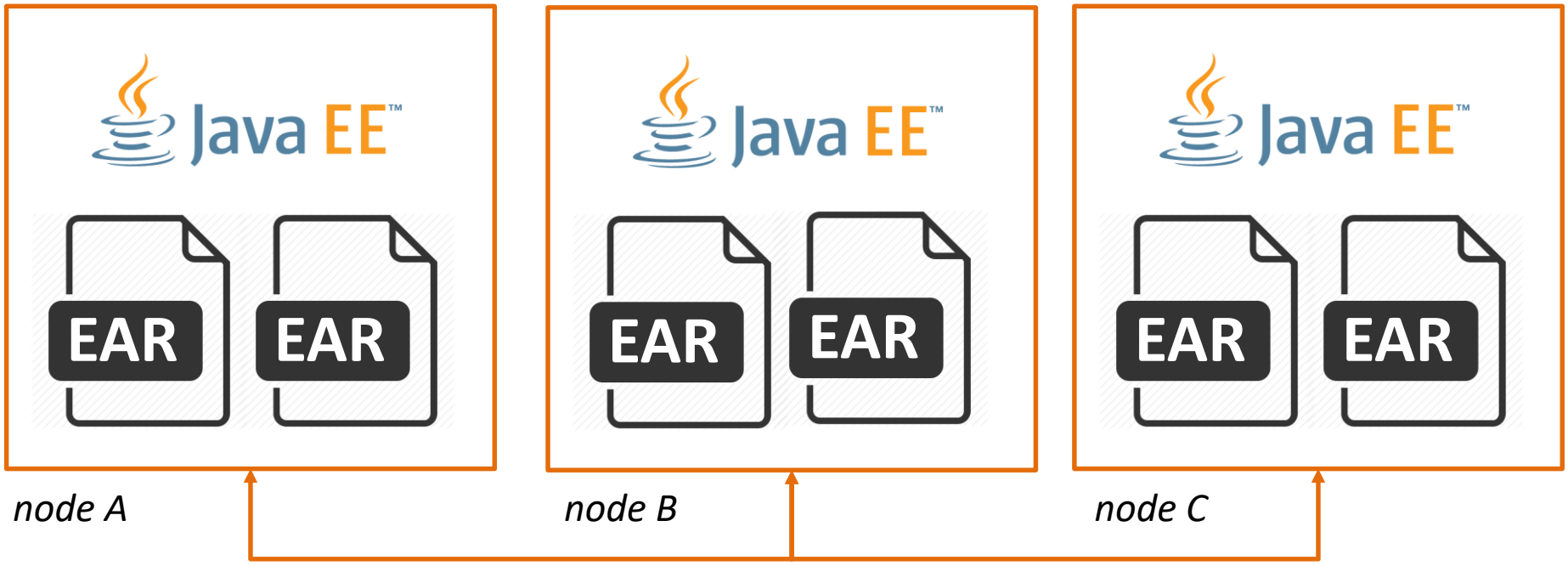
Where we are coming from: the “distributed monolith”.

node A

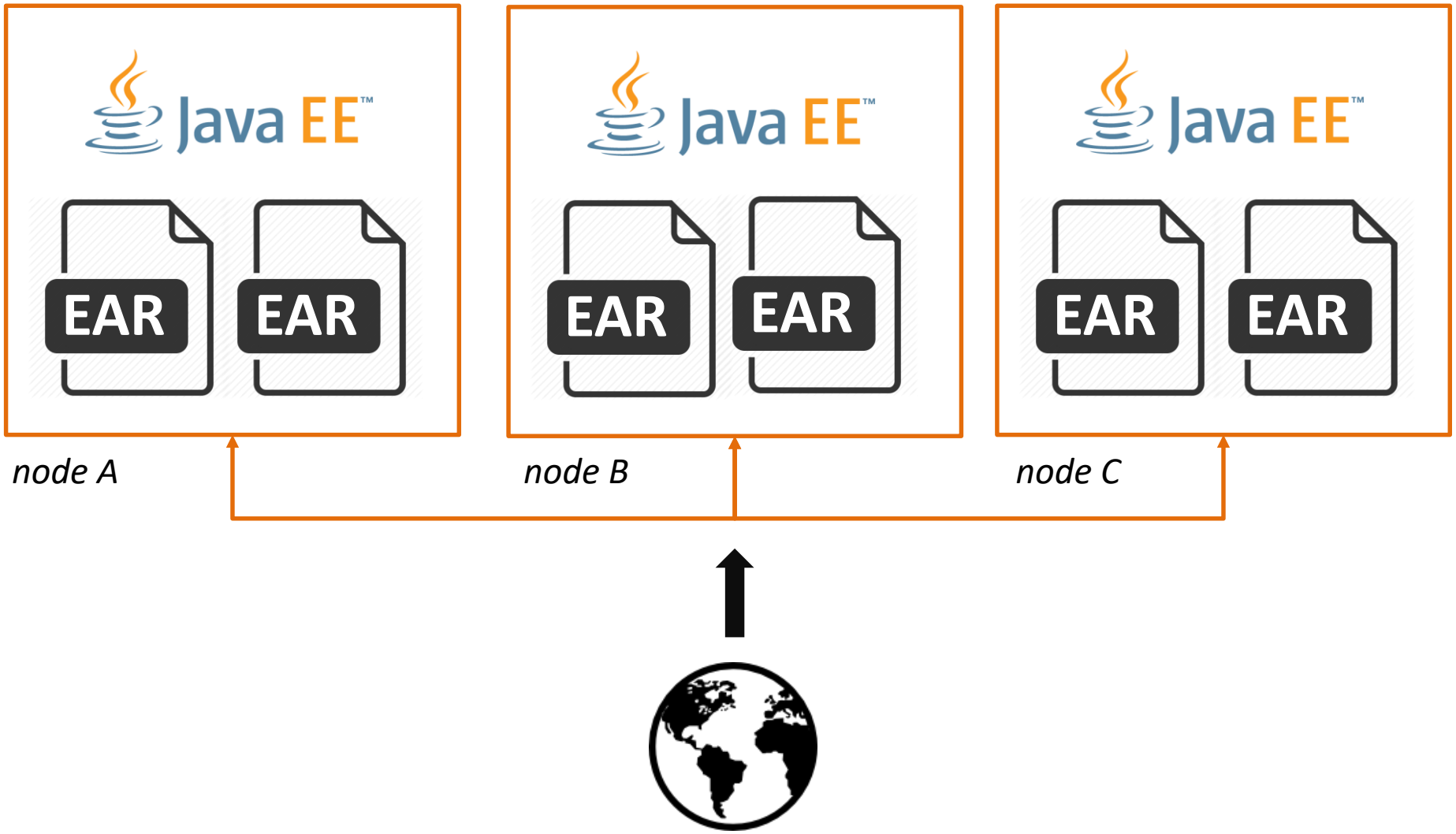
node B

node C

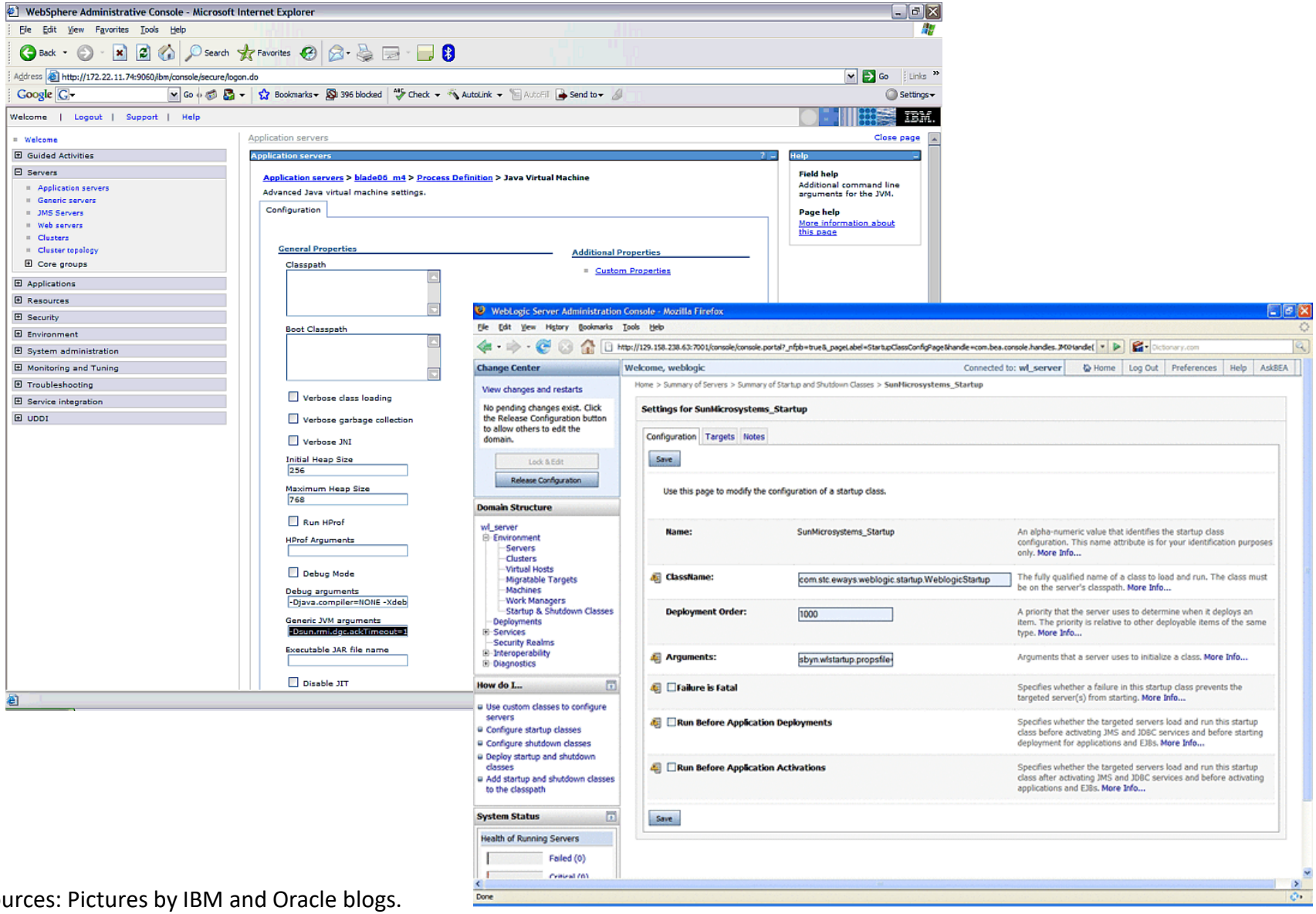
Where we are coming from: the “distributed monolith”.



Where we are coming from: the “distributed monolith”.

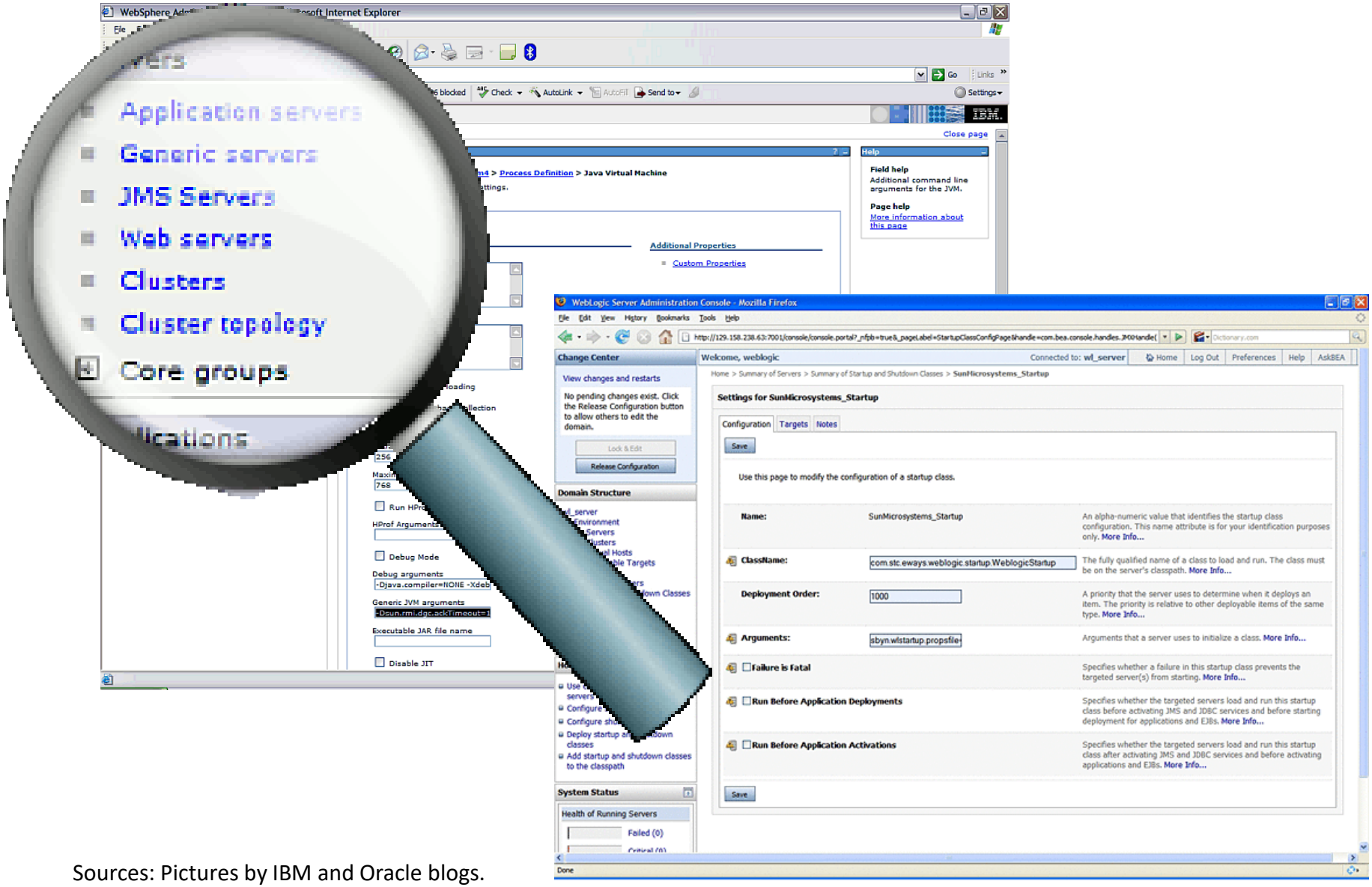


Where we are coming from: distribution by cloning.



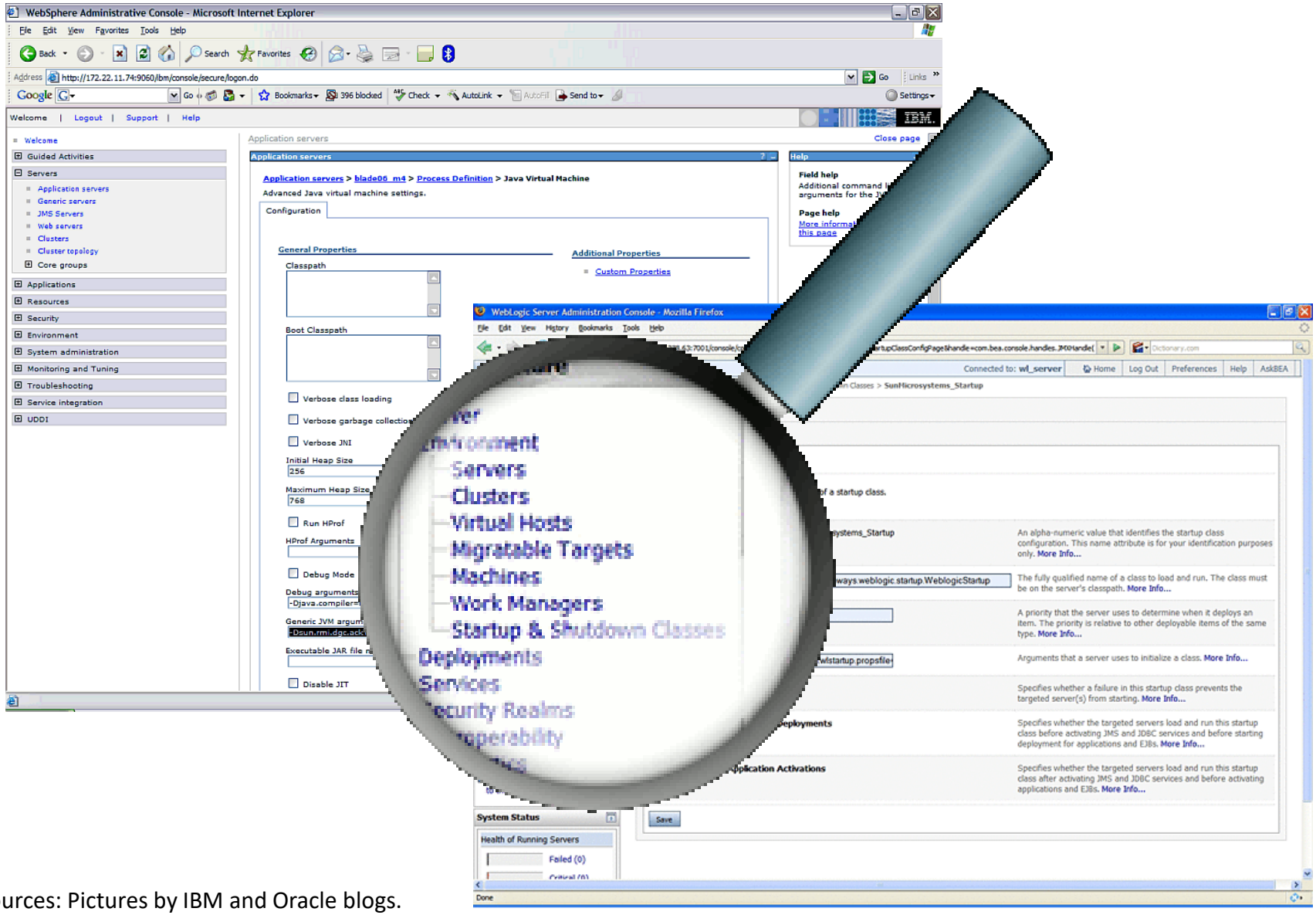
Sources: Pictures by IBM and Oracle blogs.

Where we are coming from: distribution by cloning.



Sources: Pictures by IBM and Oracle blogs.

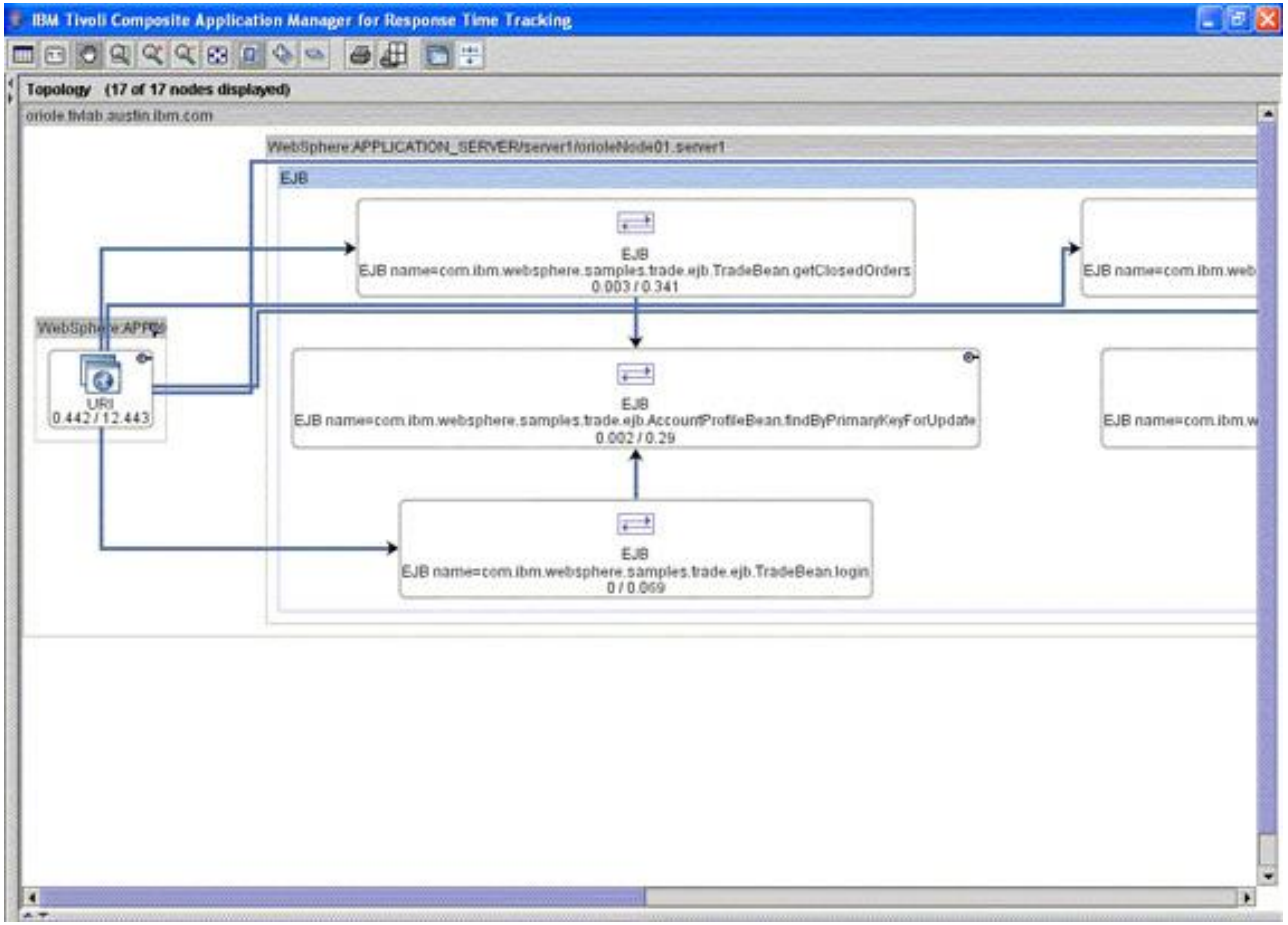
Where we are coming from: distribution by cloning.



Sources: Pictures by IBM and Oracle blogs.

Where we are coming from: inferred tracing.

Use of standard APIs: limits development to app server's capabilities.
Sacrifice freedom in development but ease operation.
Standard APIs allow server to interpret program semantics.



Sources: Picture by IBM blog.

Where we transition to: greenfield “micro”-services.

Sources: Lagom is a Lightbend trademark, Spring Boot is a Pivotal trademark.

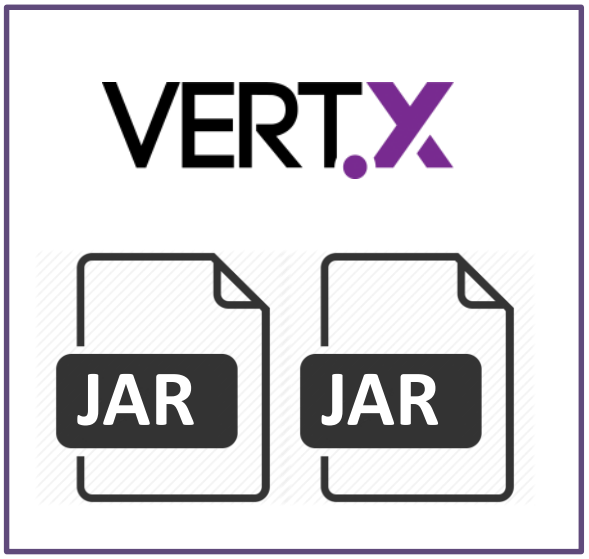
Where we transition to: greenfield “micro”-services.

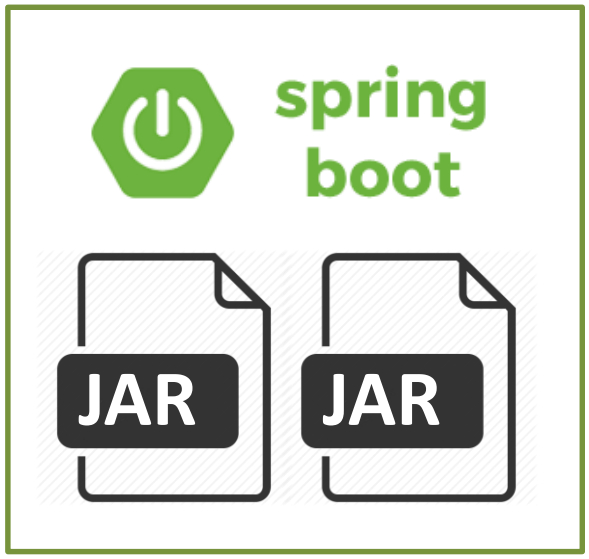
Sources: Lagom is a Lightbend trademark, Spring Boot is a Pivotal trademark.

Where we transition to: greenfield “micro”-services.

Sources: Lagom is a Lightbend trademark, Spring Boot is a Pivotal trademark.

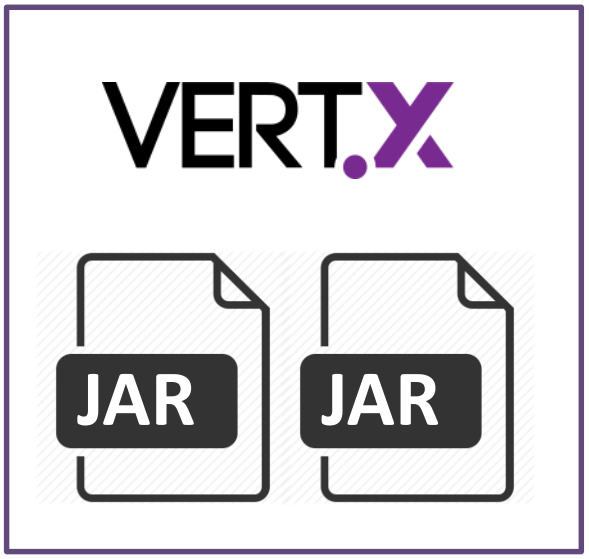
Where we transition to: greenfield “micro”-services.





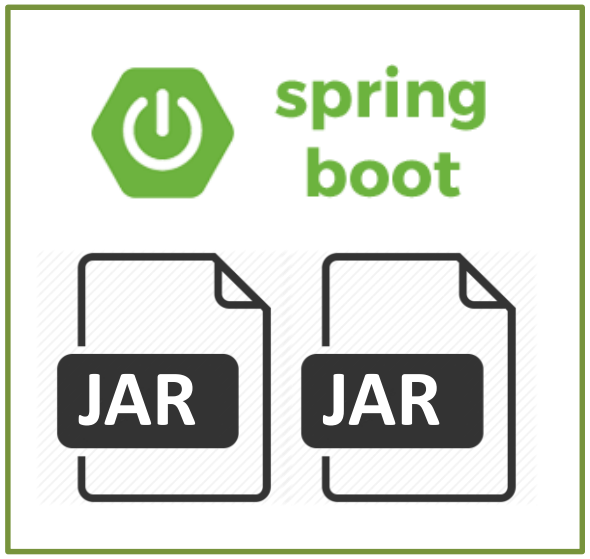
Sources: Lagom is a Lightbend trademark, Spring Boot is a Pivotal trademark.

Where we transition to: greenfield “micro”-services.



service A

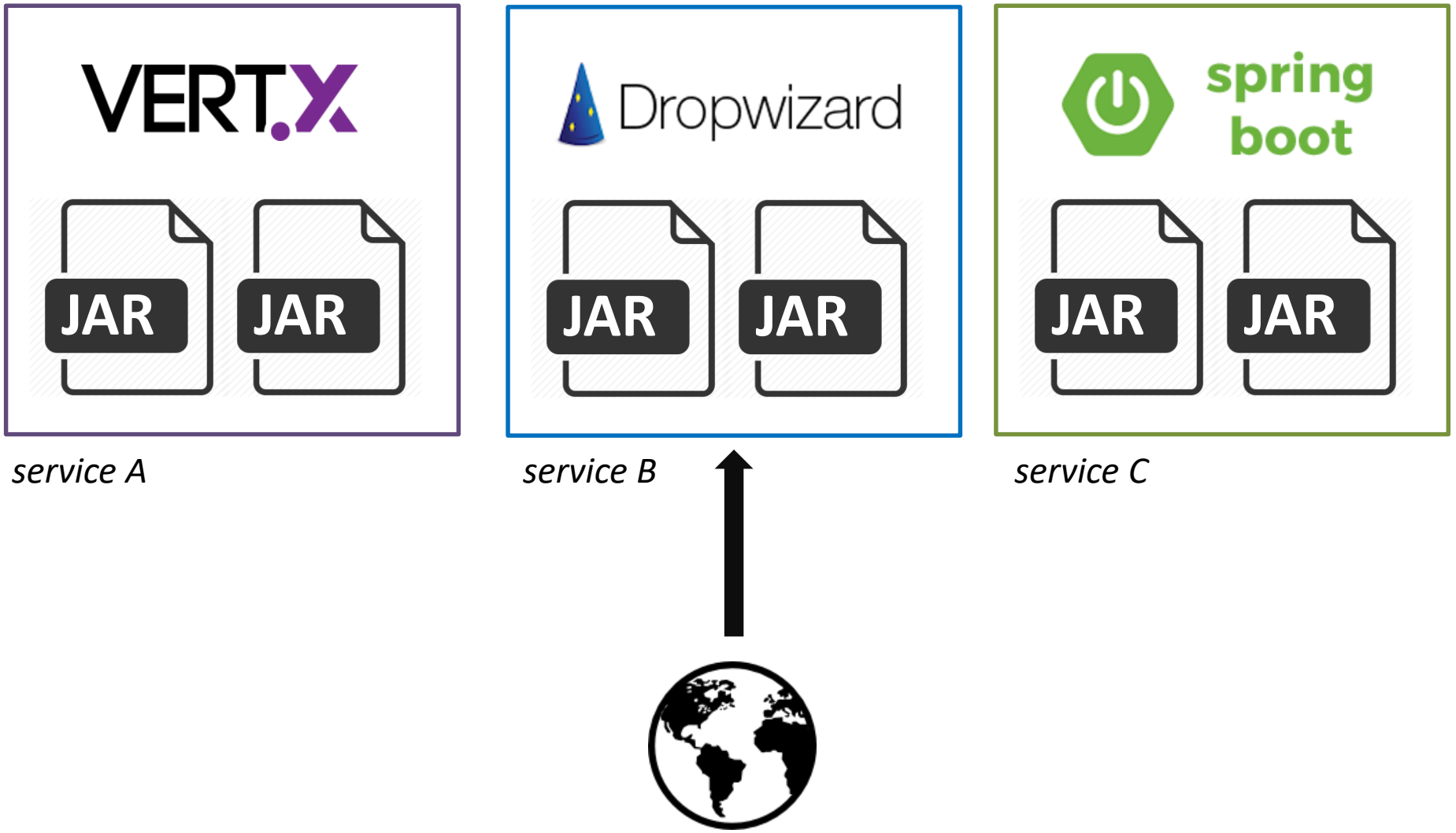
service B



service C

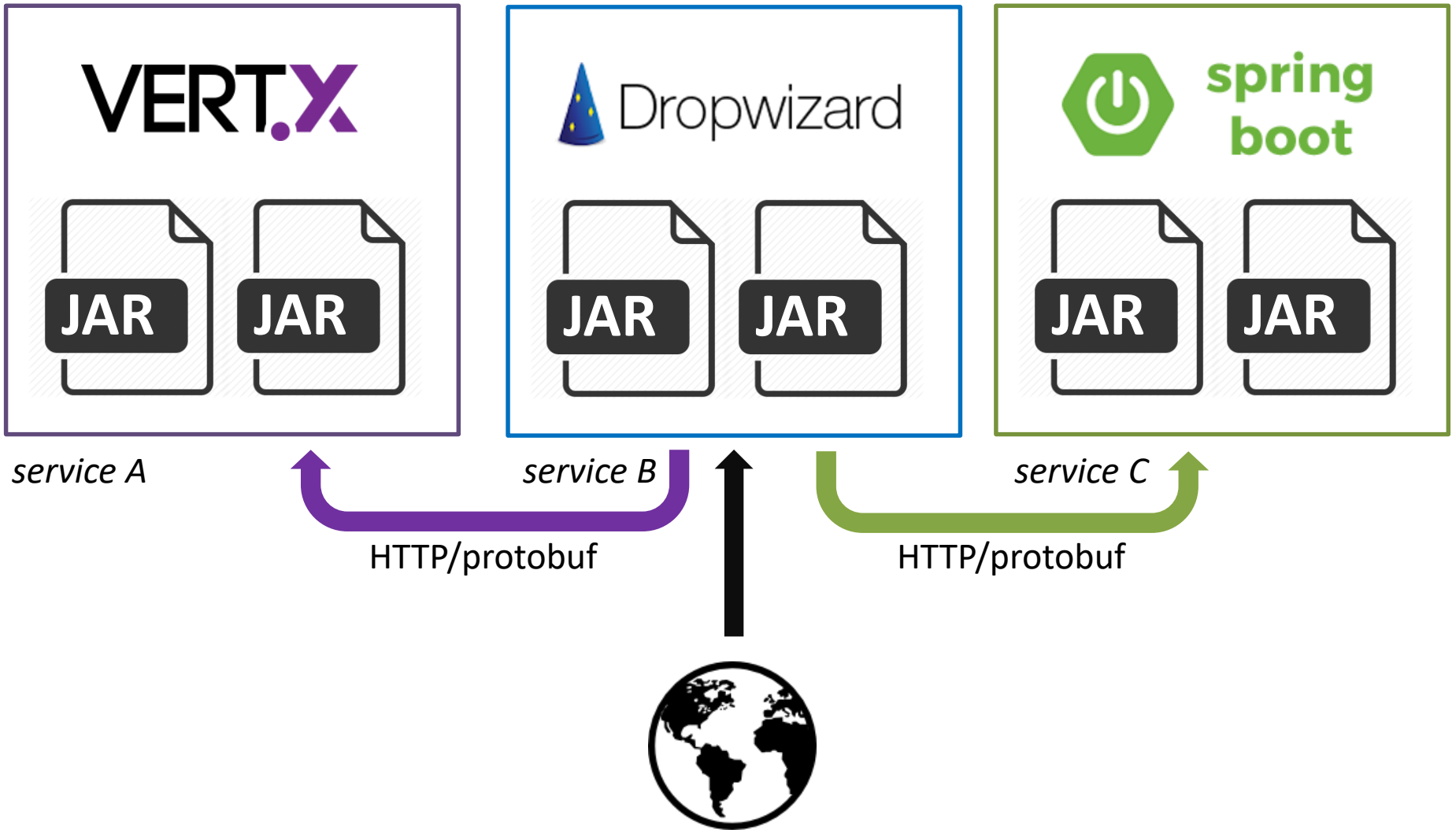
Sources: Lagom is a Lightbend trademark, Spring Boot is a Pivotal trademark.

Where we transition to: greenfield “micro”-services.



Sources: Lagom is a Lightbend trademark, Spring Boot is a Pivotal trademark.

Where we transition to: greenfield “micro”-services.



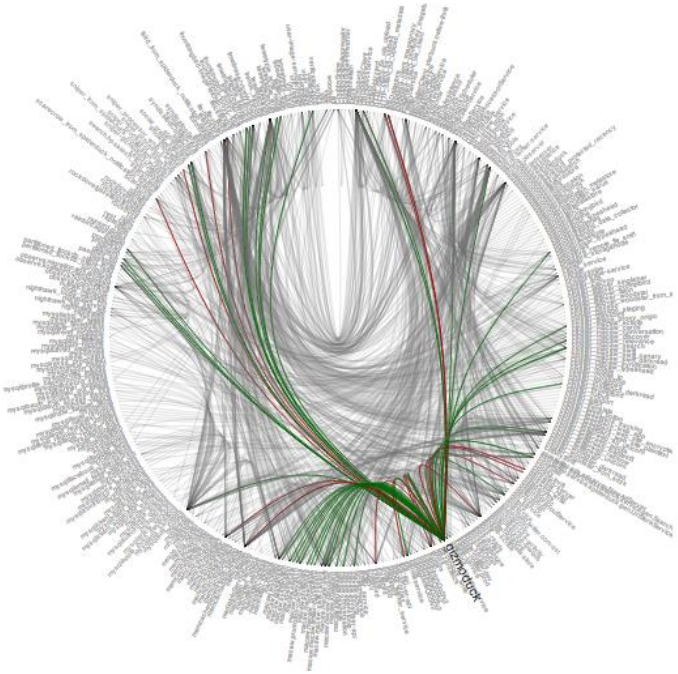
Sources: Lagom is a Lightbend trademark, Spring Boot is a Pivotal trademark.

Where we transition to: “The wheel of doom”

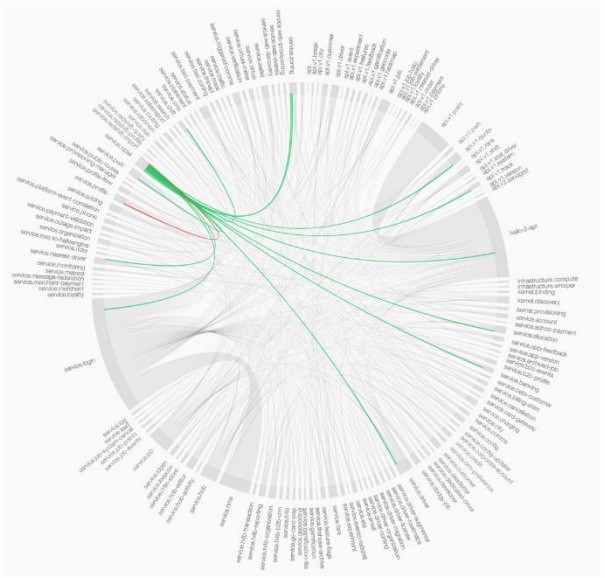
Hailo

Sources: Screenshots from Zipkin on Twitter Blog and Hailo Blog.

Where we transition to: “The wheel of doom”



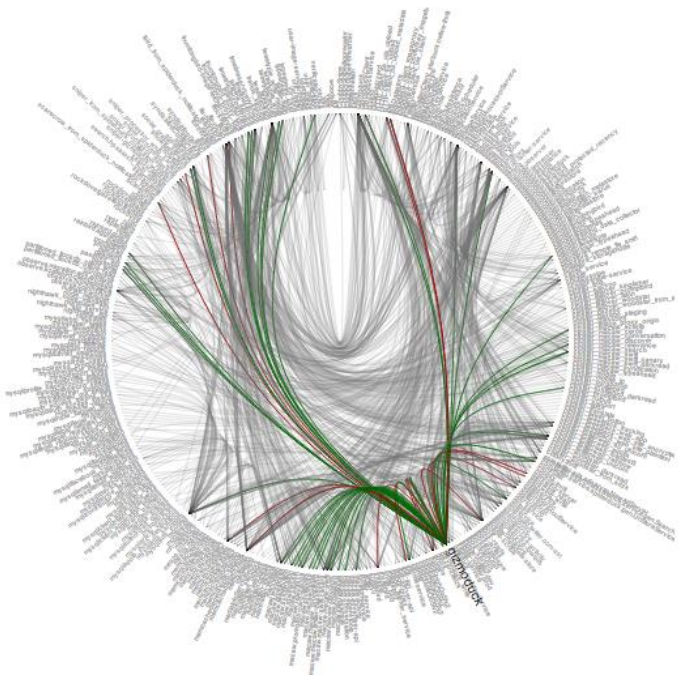
Twitter



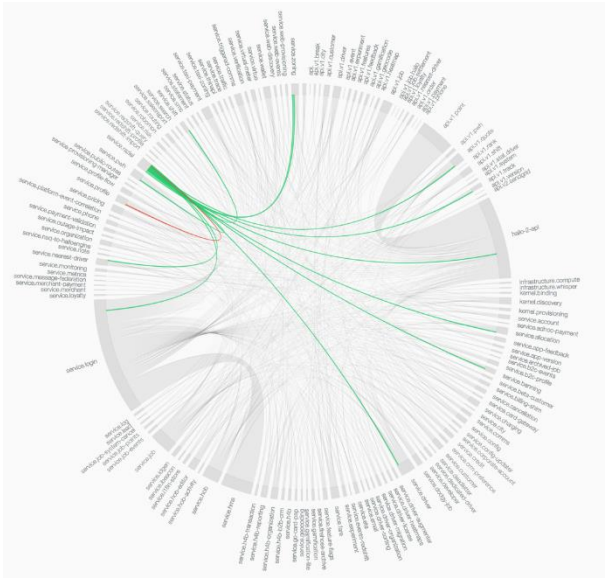
Hailo

Sources: Screenshots from Zipkin on Twitter Blog and Hailo Blog.

Where we transition to: “The wheel of doom”



Twitter



Hailo

The Empire

Simple bits, complex interaction: **death star topologies**.

Sources: Screenshots from Zipkin on Twitter Blog and Hailo Blog. Death Star by “Free Star Wars Fan Art Collection”.

Where we transition to: simple services, complex operation.

1. Writing distributed services can ease development but adds challenges on integration.
2. Distributed services without standardized APIs cannot easily be observed in interaction.
3. Distributed services make it harder to collect structured monitoring data.
4. Distributed (micro-)services require DevOps to successfully run in production.

Sources: Picture by Reddit.

The next big thing: serverless architecture?

Sources: AWS Lambda is an Amazon trademark.

The next big thing: serverless architecture?

Sources: AWS Lambda is an Amazon trademark.

The next big thing: serverless architecture?

Sources: AWS Lambda is an Amazon trademark.

The next big thing: serverless architecture?

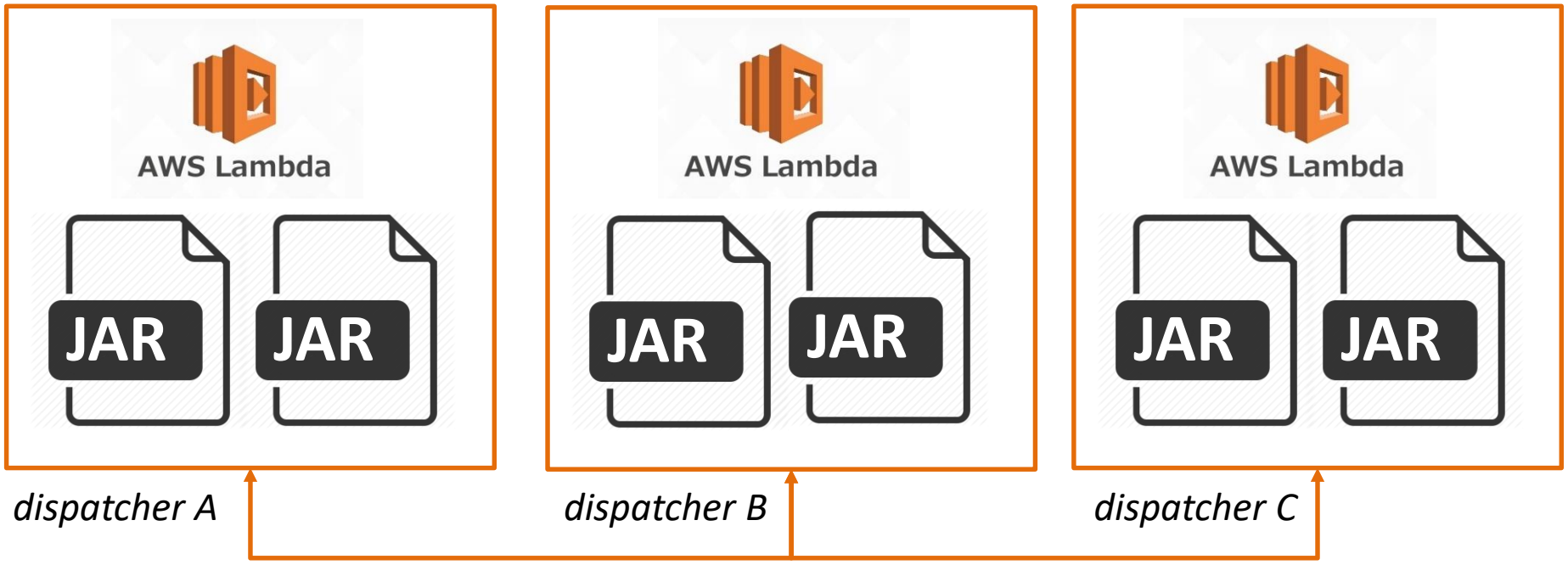
dispatcher A

dispatcher B

dispatcher C

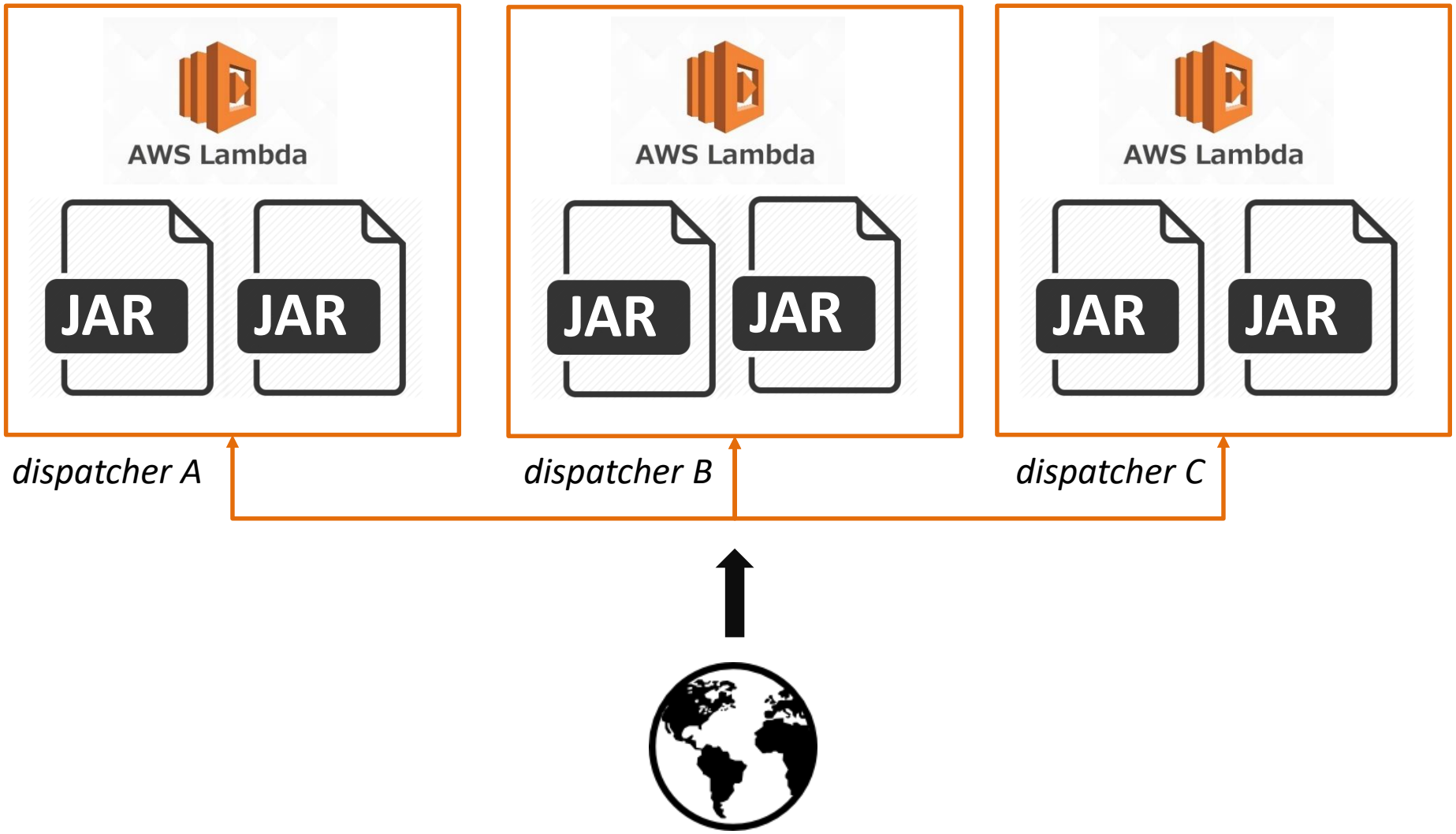
Sources: AWS Lambda is an Amazon trademark.

The next big thing: serverless architecture?



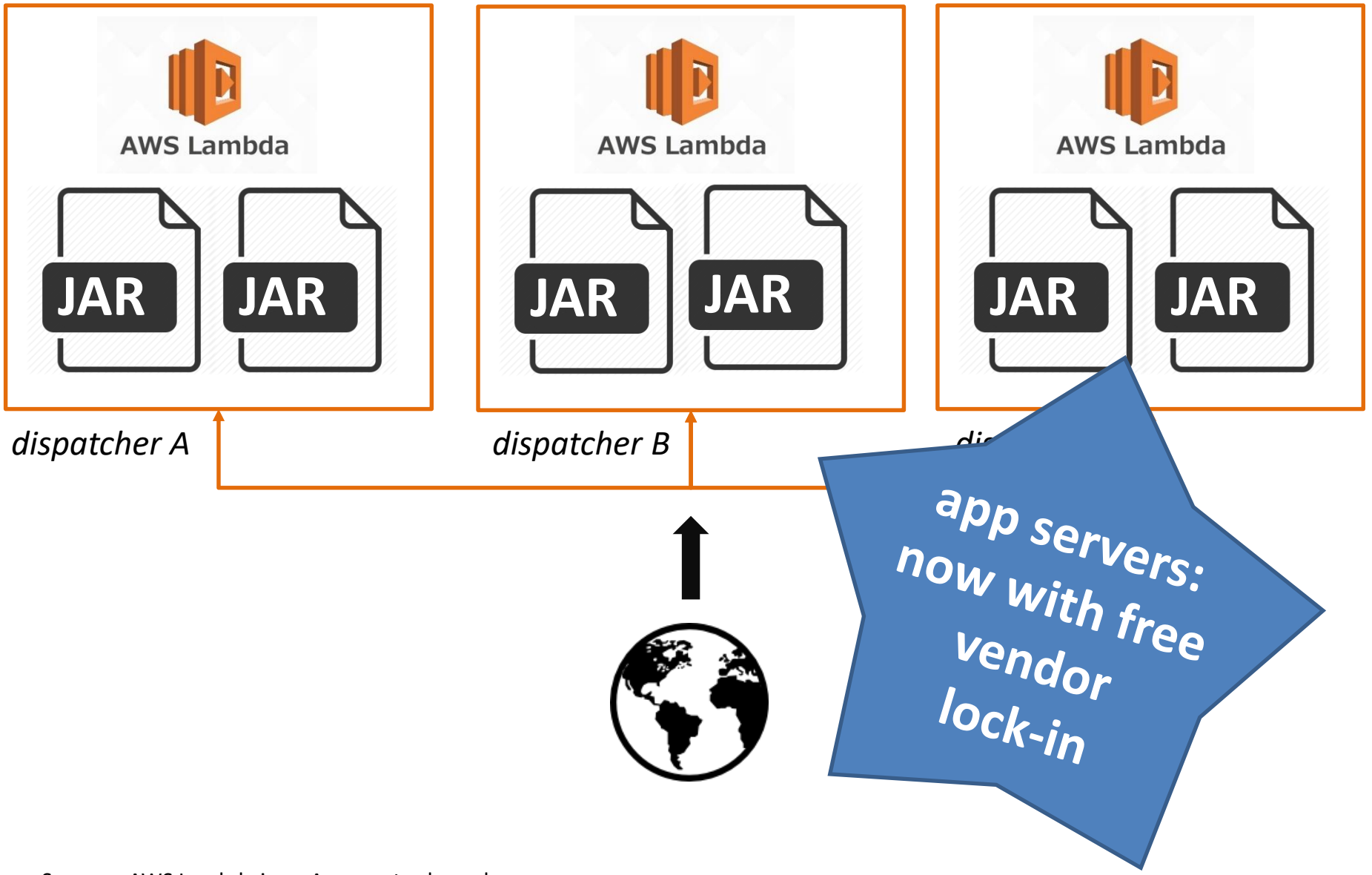
Sources: AWS Lambda is an Amazon trademark.

The next big thing: serverless architecture?



Sources: AWS Lambda is an Amazon trademark.

The next big thing: serverless architecture?



Sources: AWS Lambda is an Amazon trademark.

The next big thing: serverless architecture?

silver bullet syndrome

Sources: Picture by Golden Eagle Coin. Concept from: “silver bullet syndrome” by Hadi Hariri.

Outline

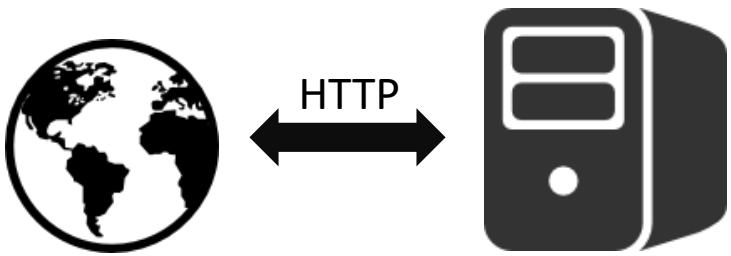
I. Inventory
II. Tracing (micro-)services
III. Implementing APM
IV. Advanced topics

What information do we want?

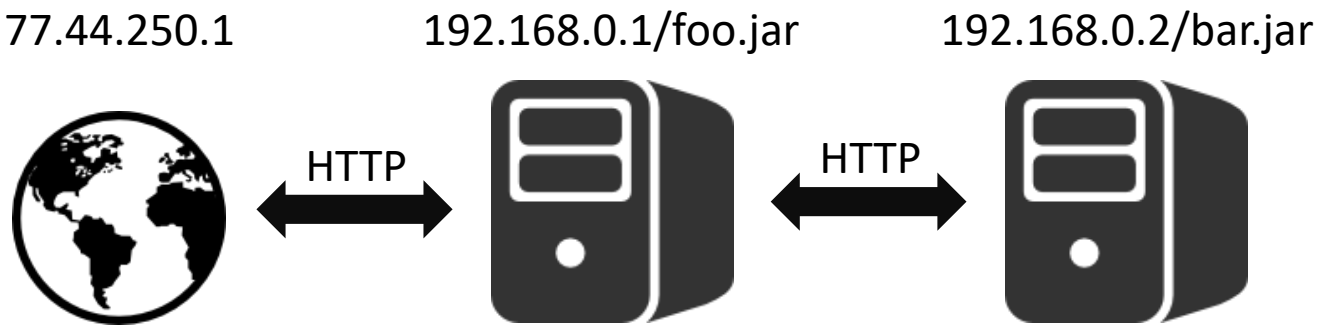
192.168.0.1/foo.jar

What information do we want?

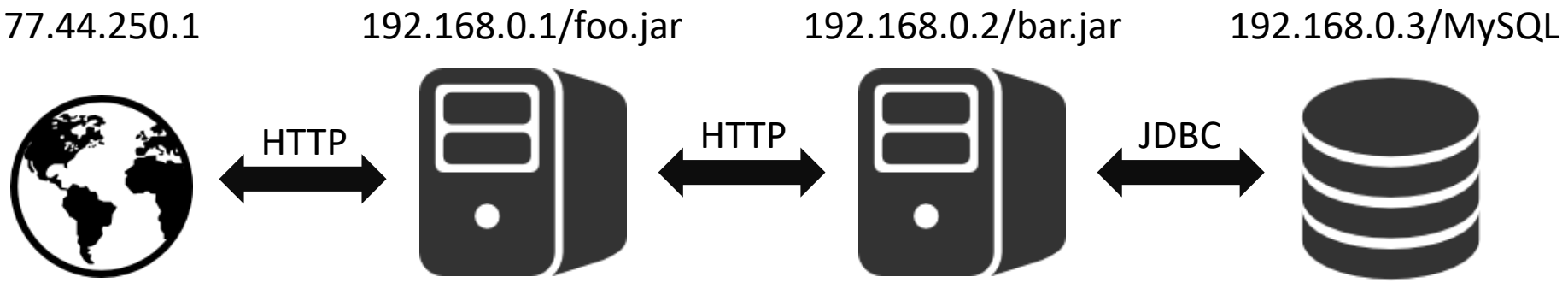
77.44.250.1 192.168.0.1/foo.jar



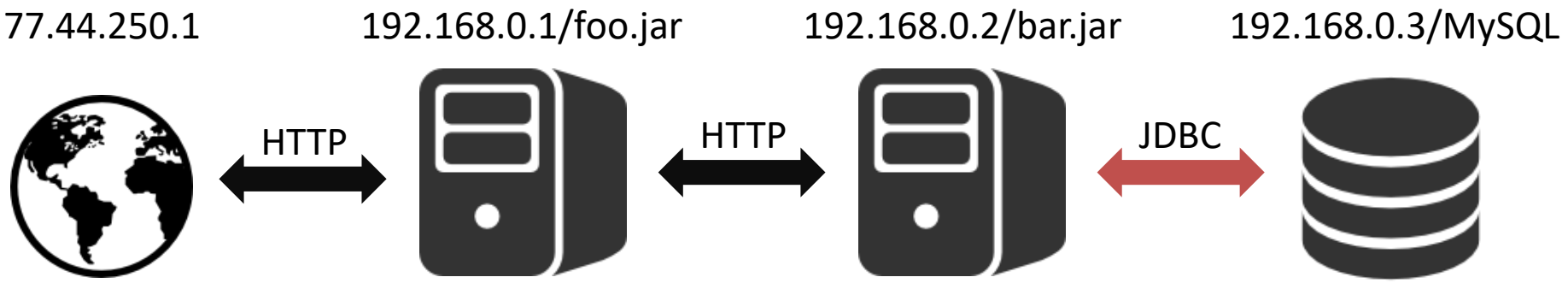
What information do we want?



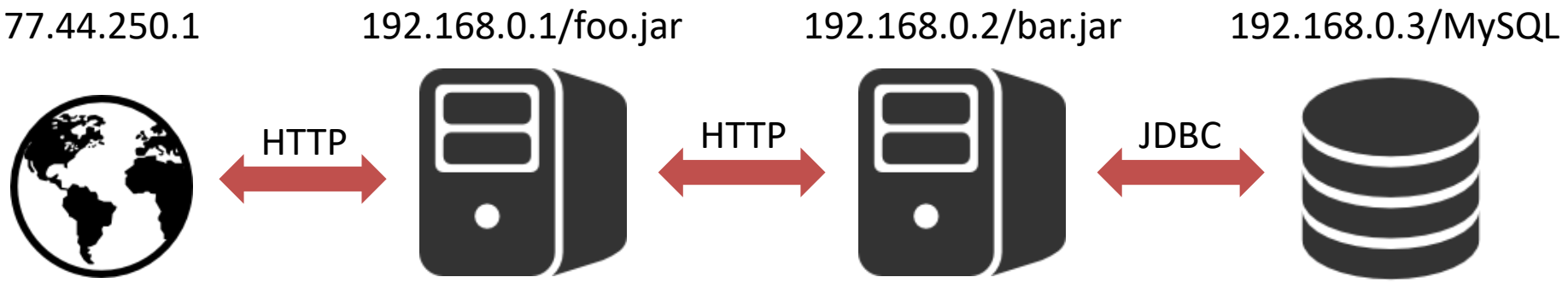
What information do we want?



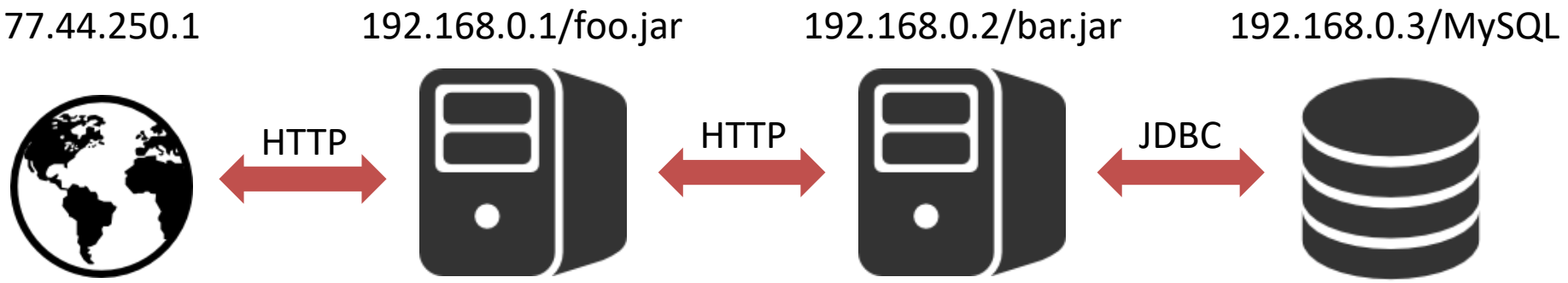
What information do we want?



What information do we want?

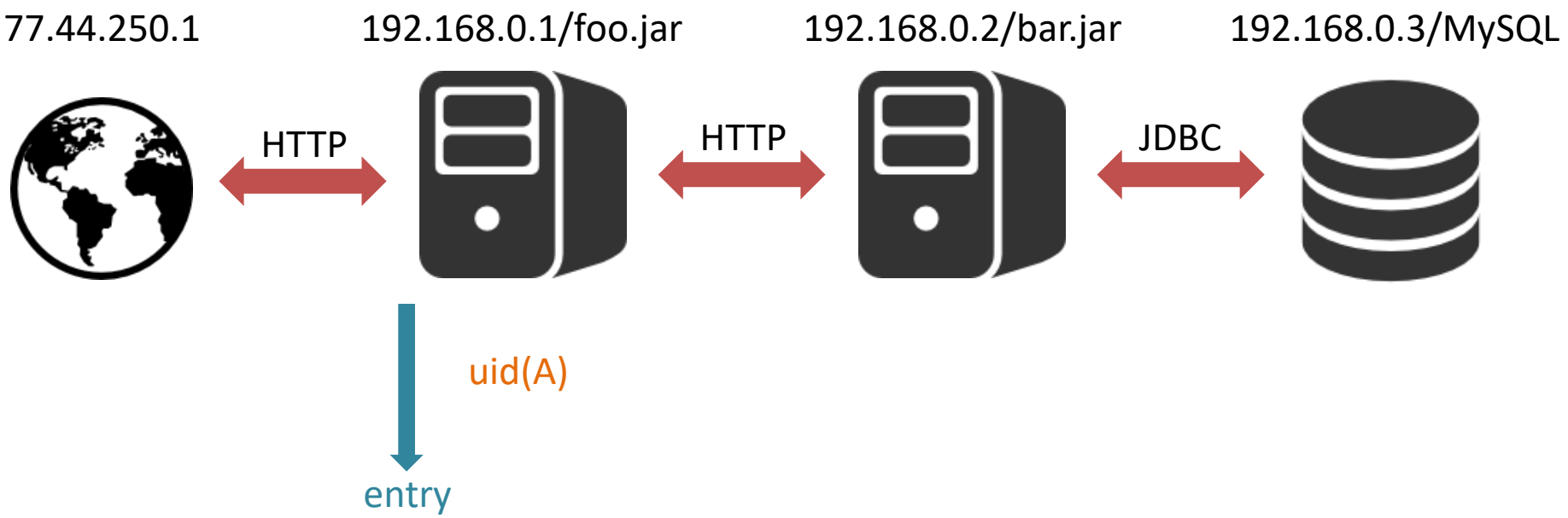


What information do we want?

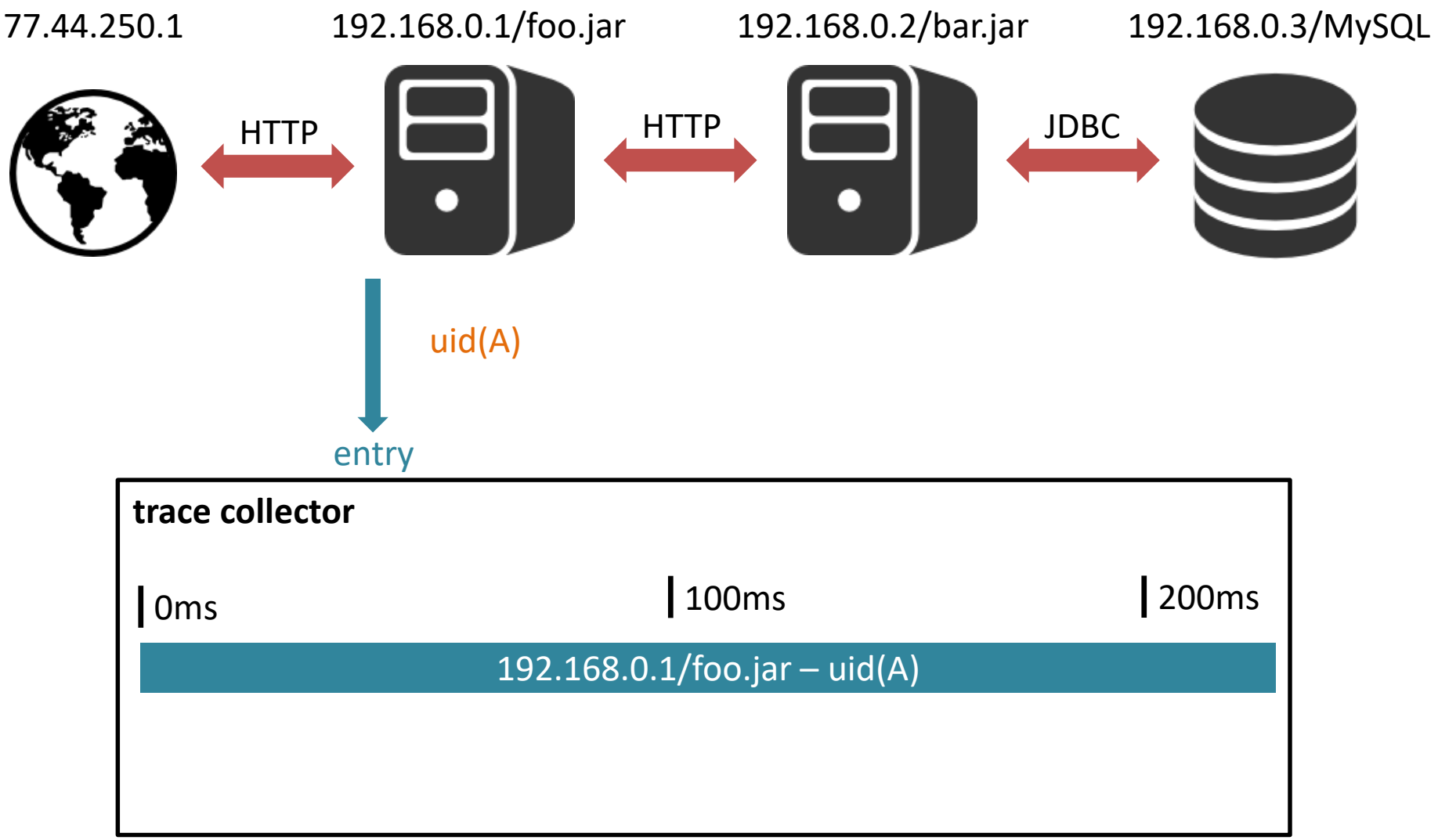


trace collector

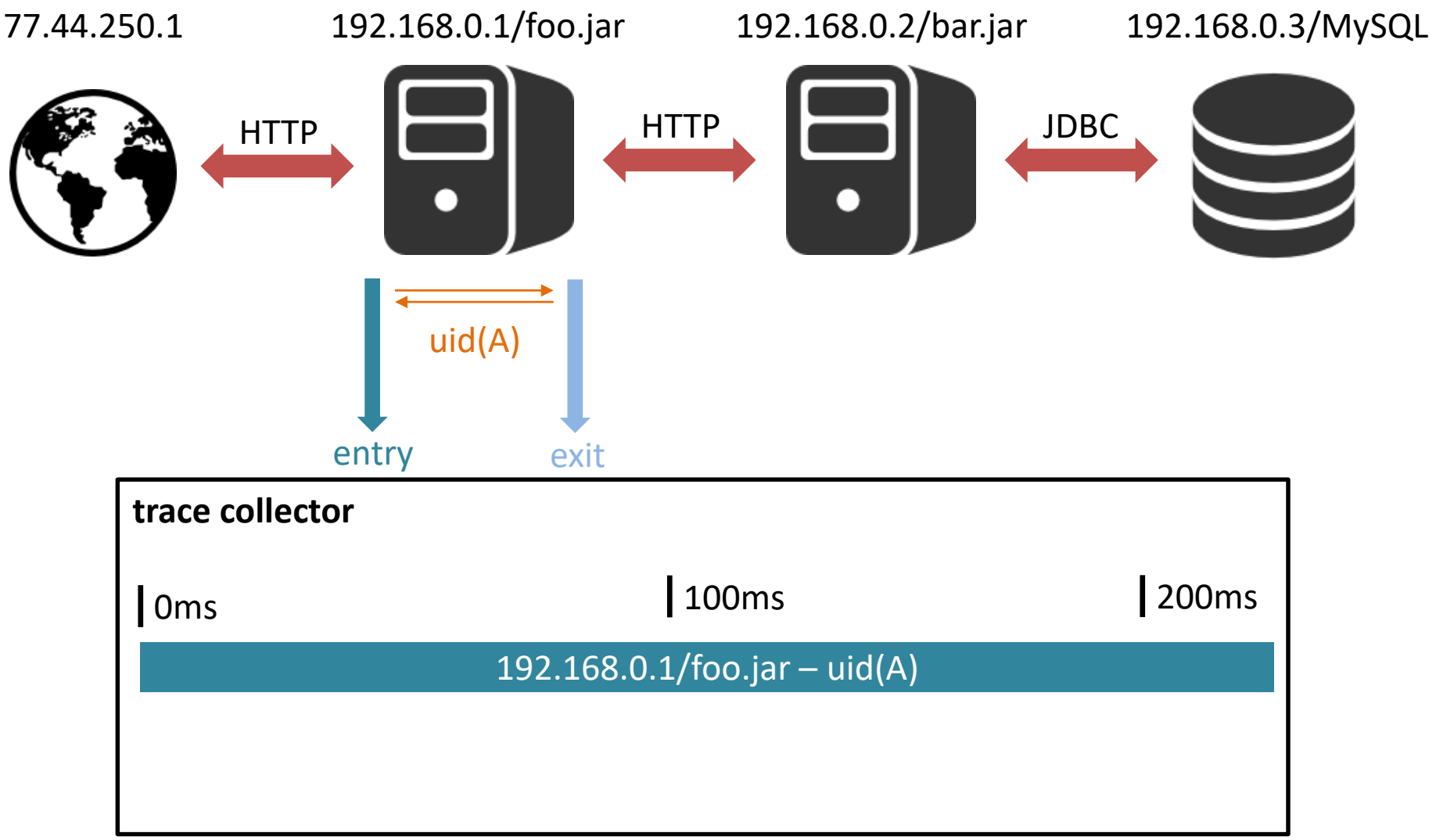
What information do we want?



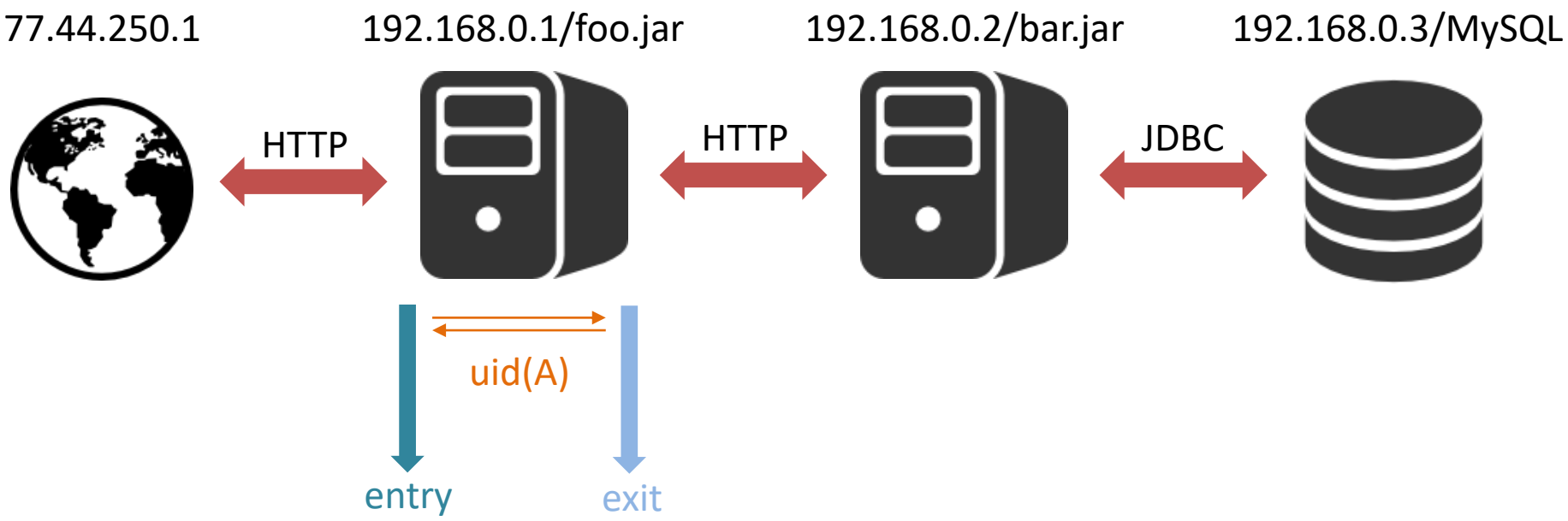
What information do we want?



What information do we want?



What information do we want?



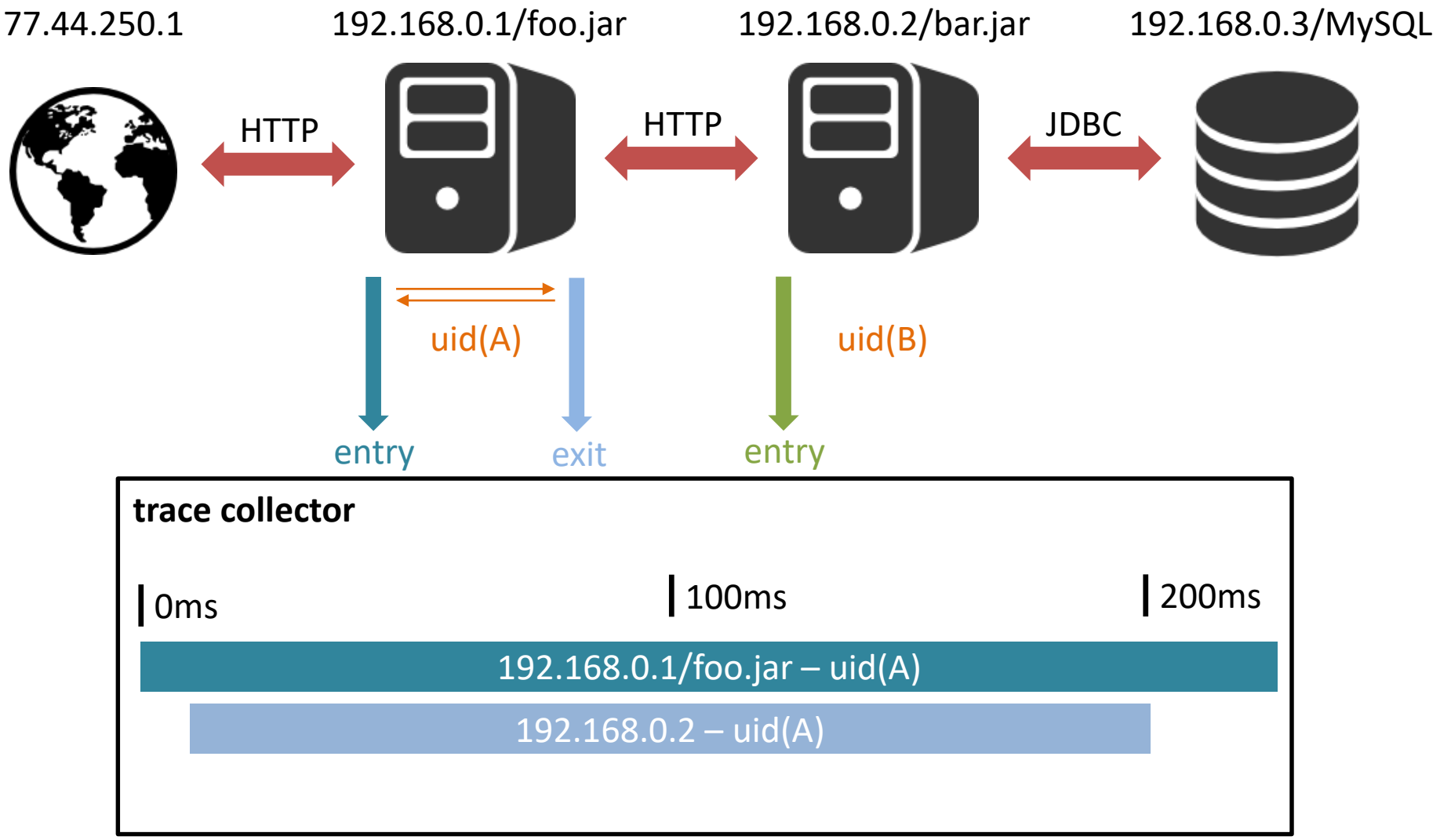
trace collector

0ms | 100ms | 200ms

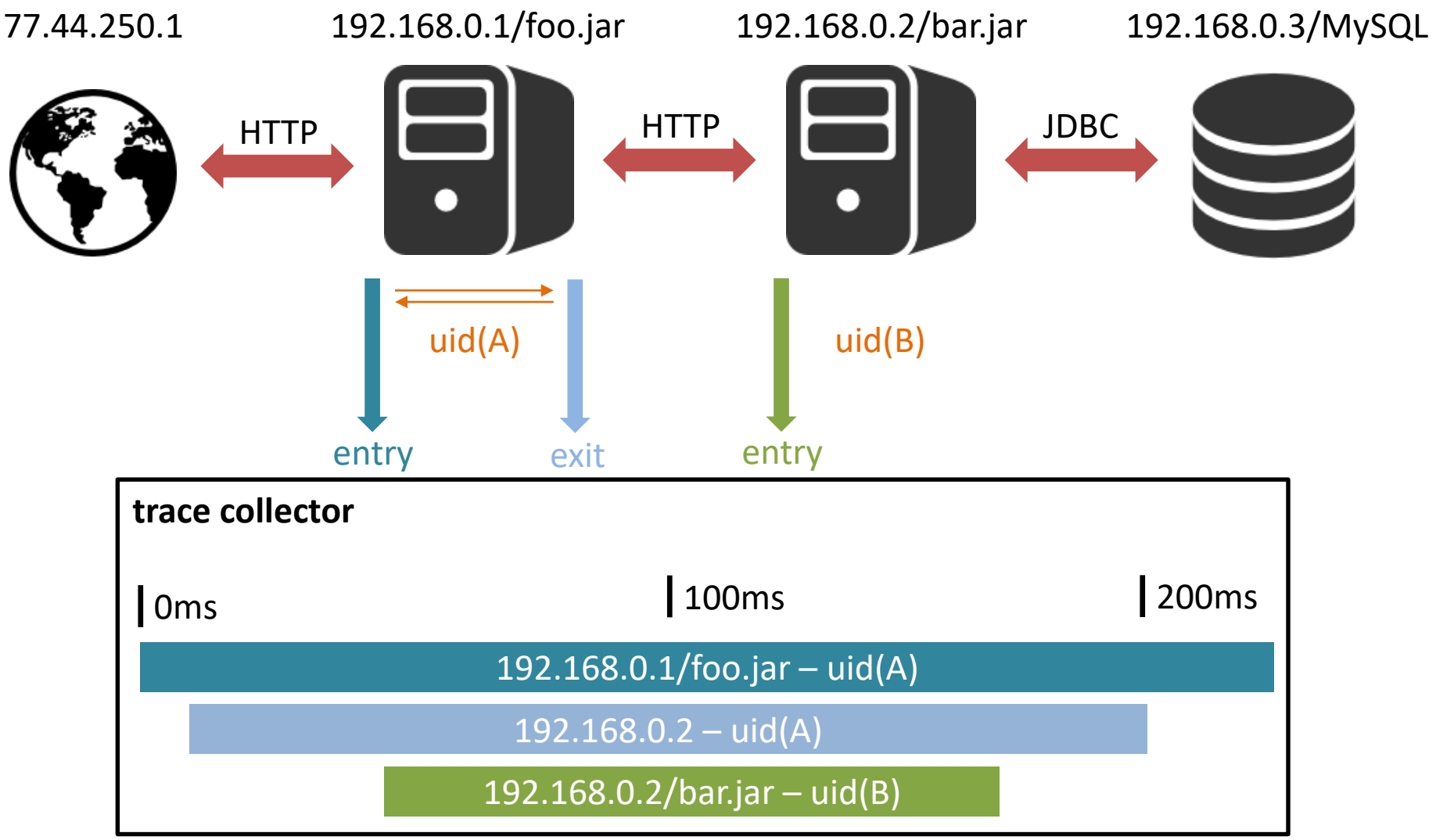
192.168.0.1/foo.jar – uid(A)

192.168.0.2 – uid(A)

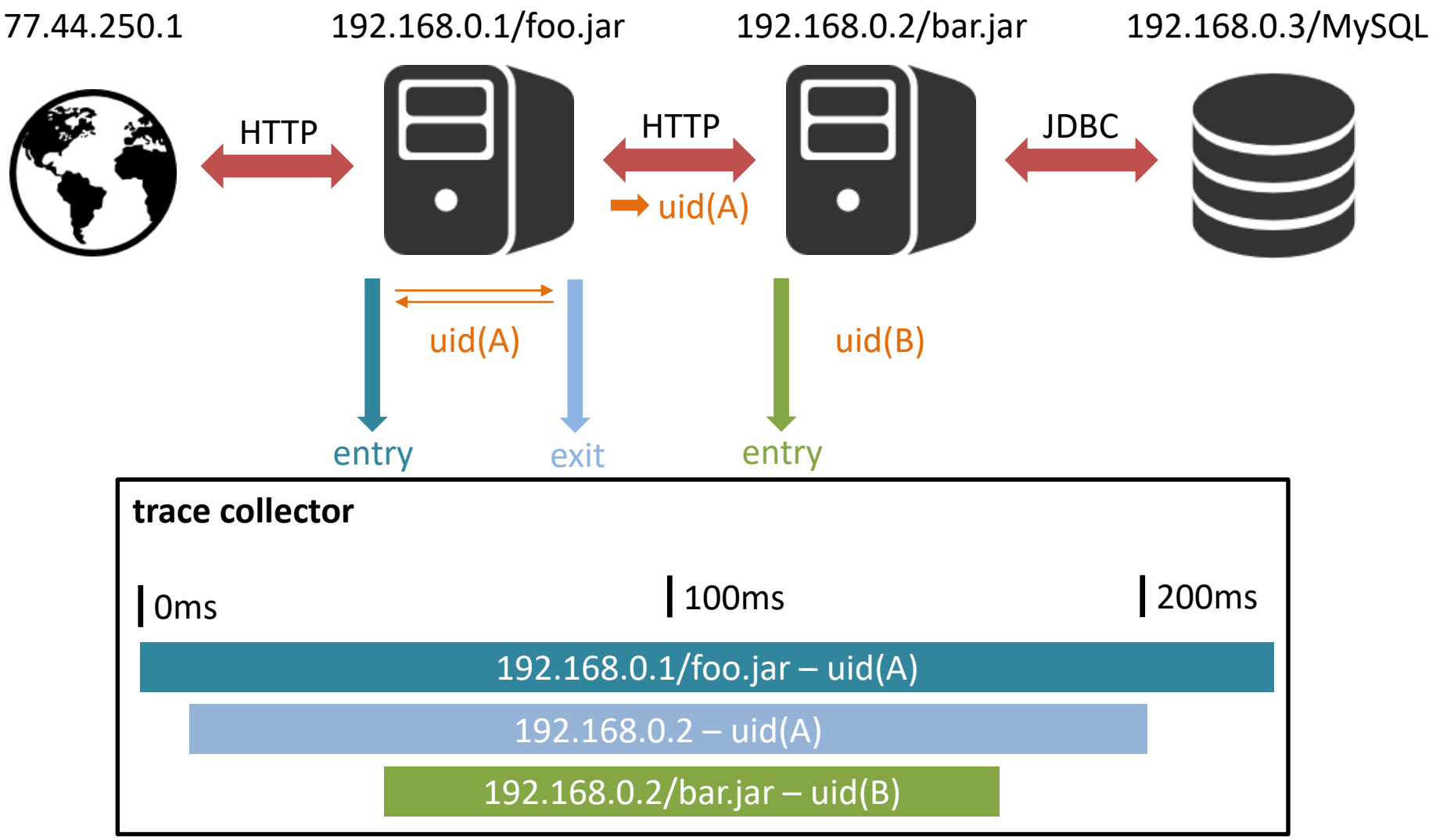
What information do we want?



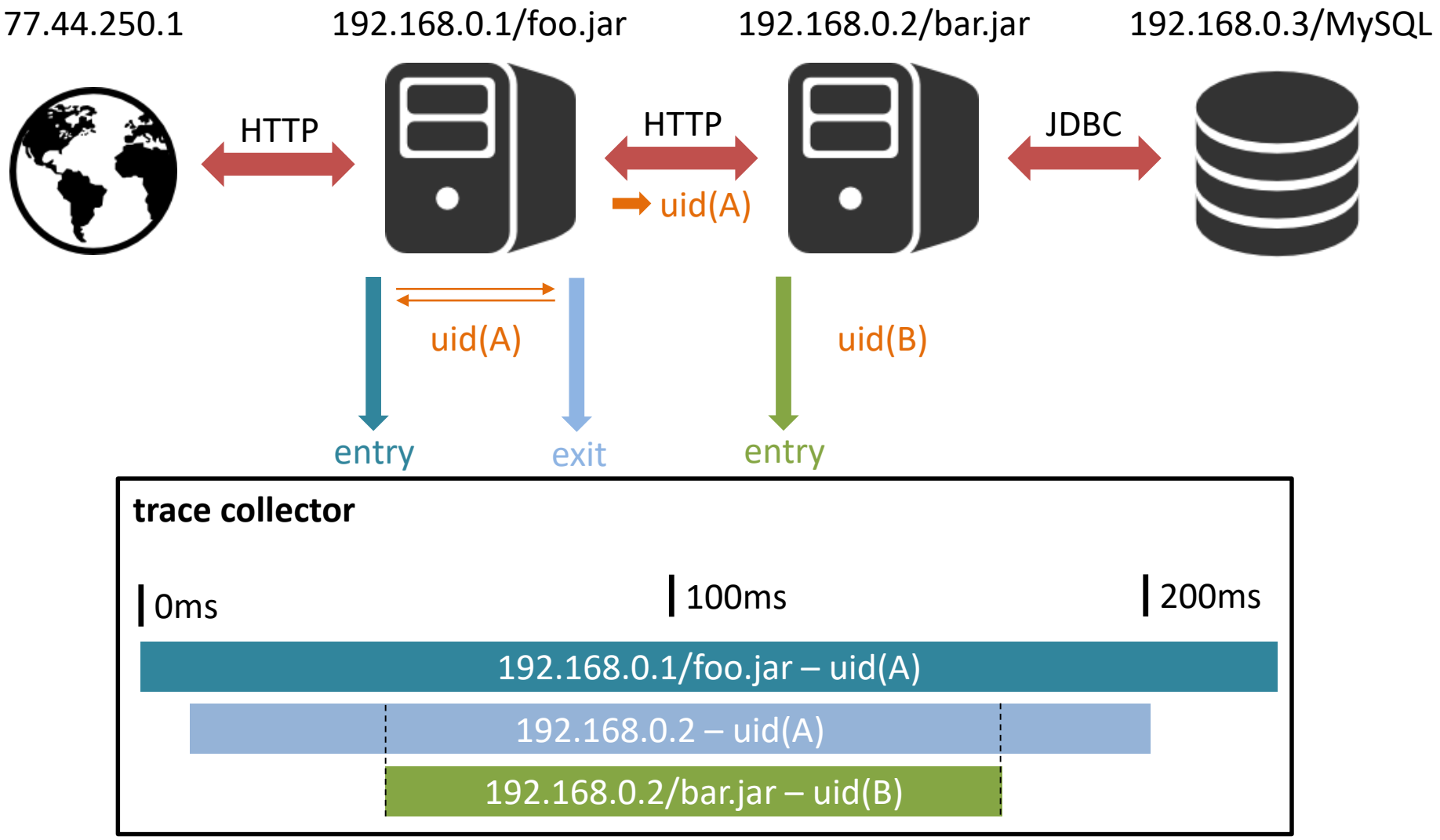
What information do we want?



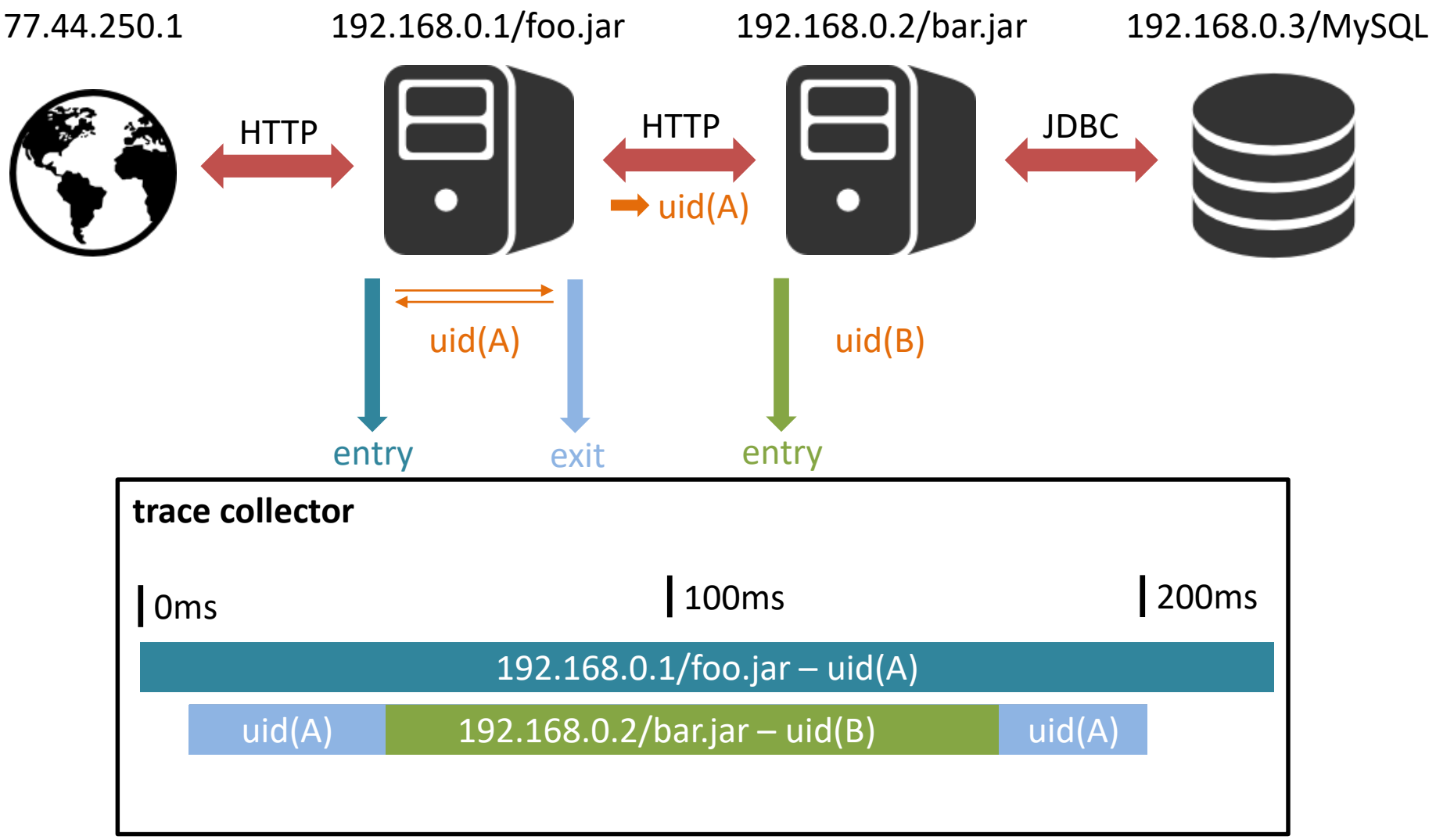
What information do we want?



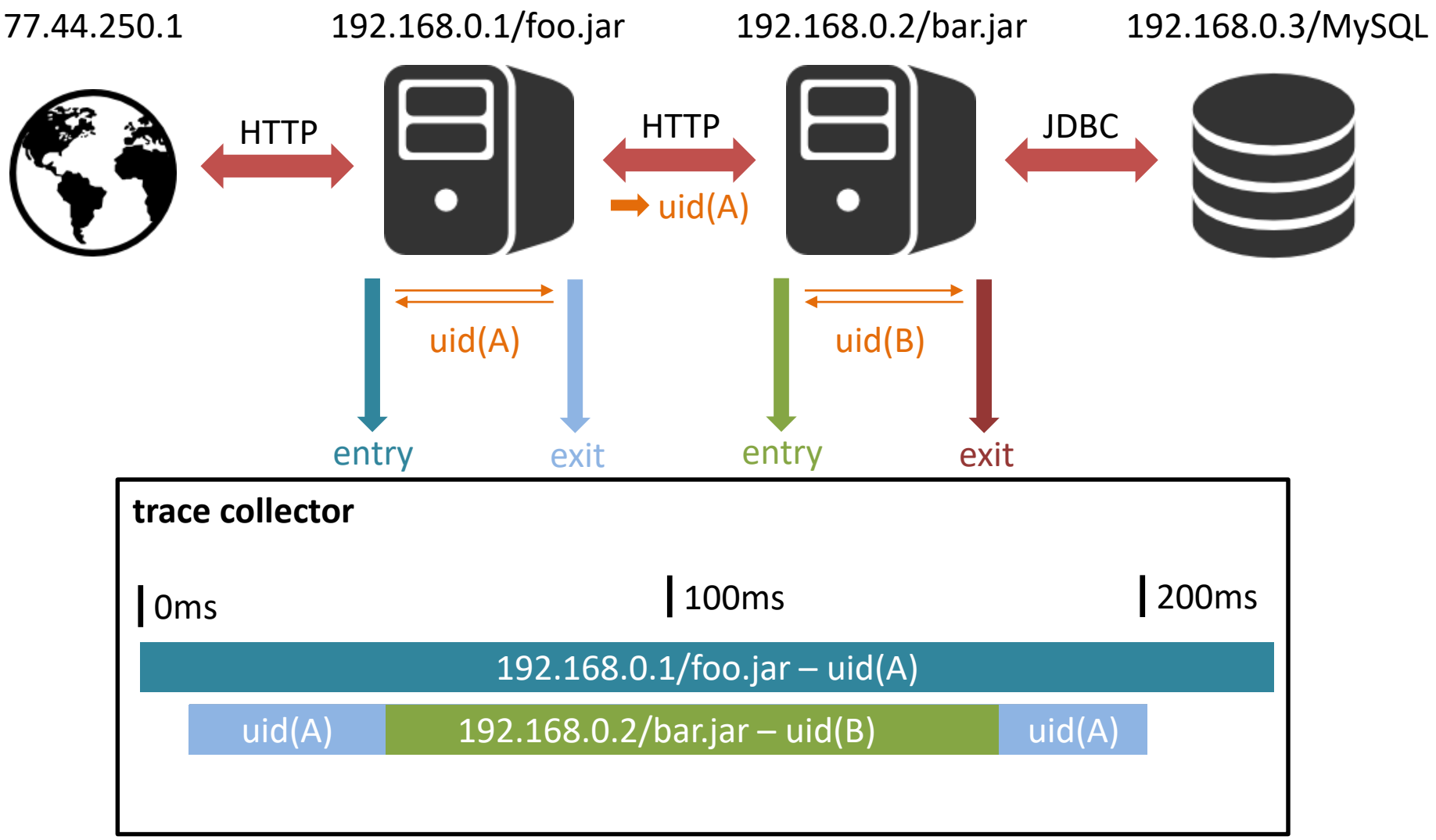
What information do we want?



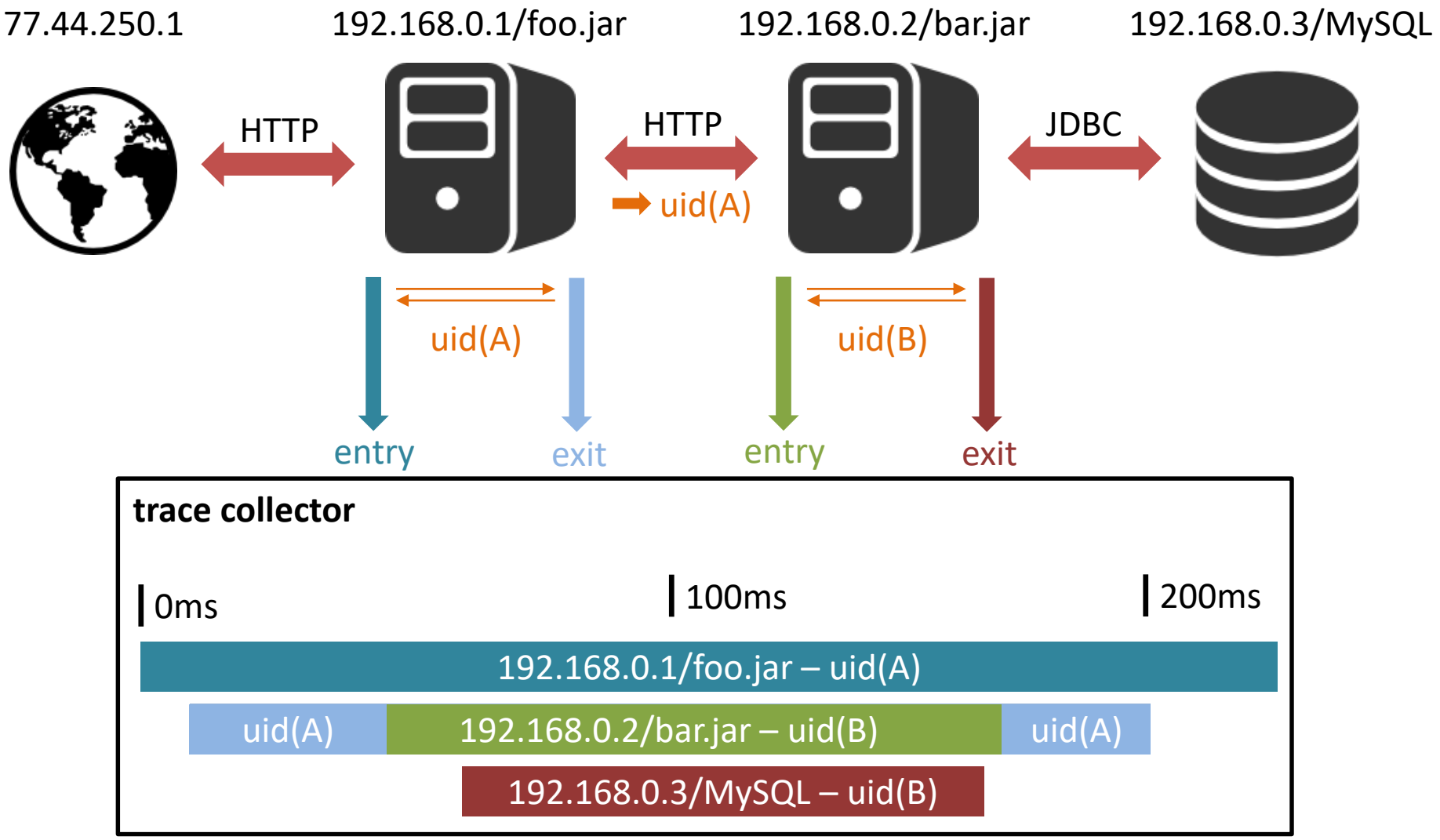
What information do we want?



What information do we want?



What information do we want?



How do we get it?

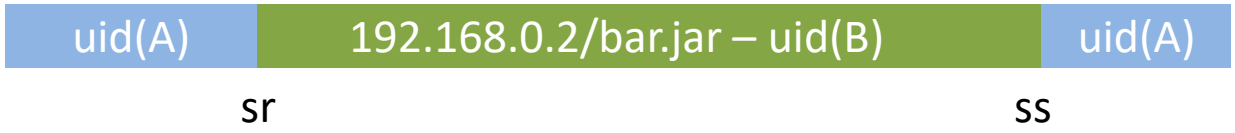
Source: Logo from zipkin.io.

How do we get it?

uid(A)	192.168.0.2/bar.jar – uid(B)	uid(A)

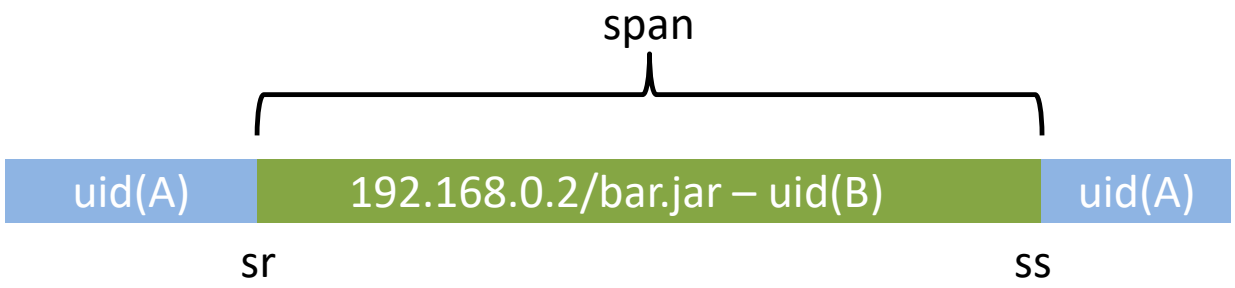
Source: Logo from zipkin.io.

How do we get it?



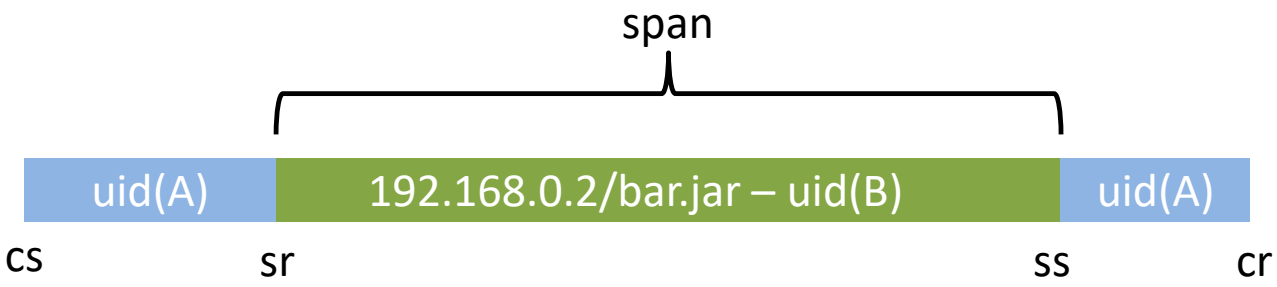
Source: Logo from zipkin.io.

How do we get it?



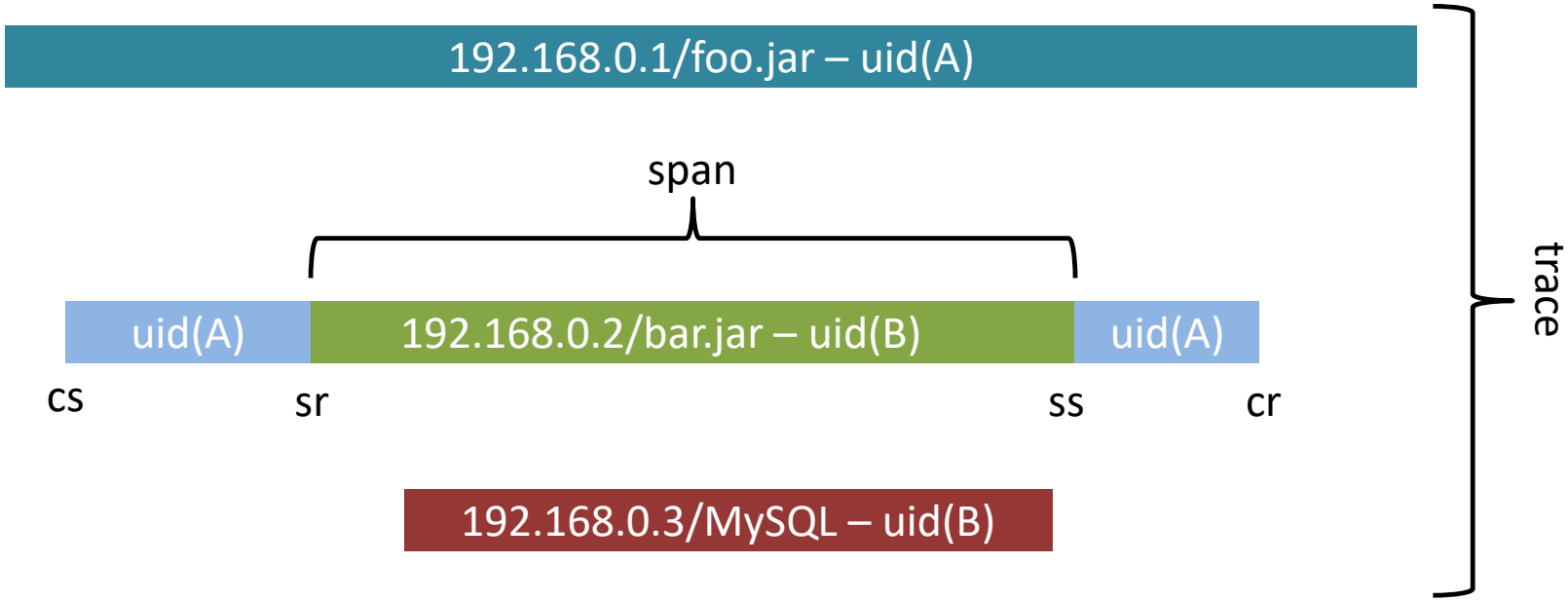
Source: Logo from zipkin.io.

How do we get it?



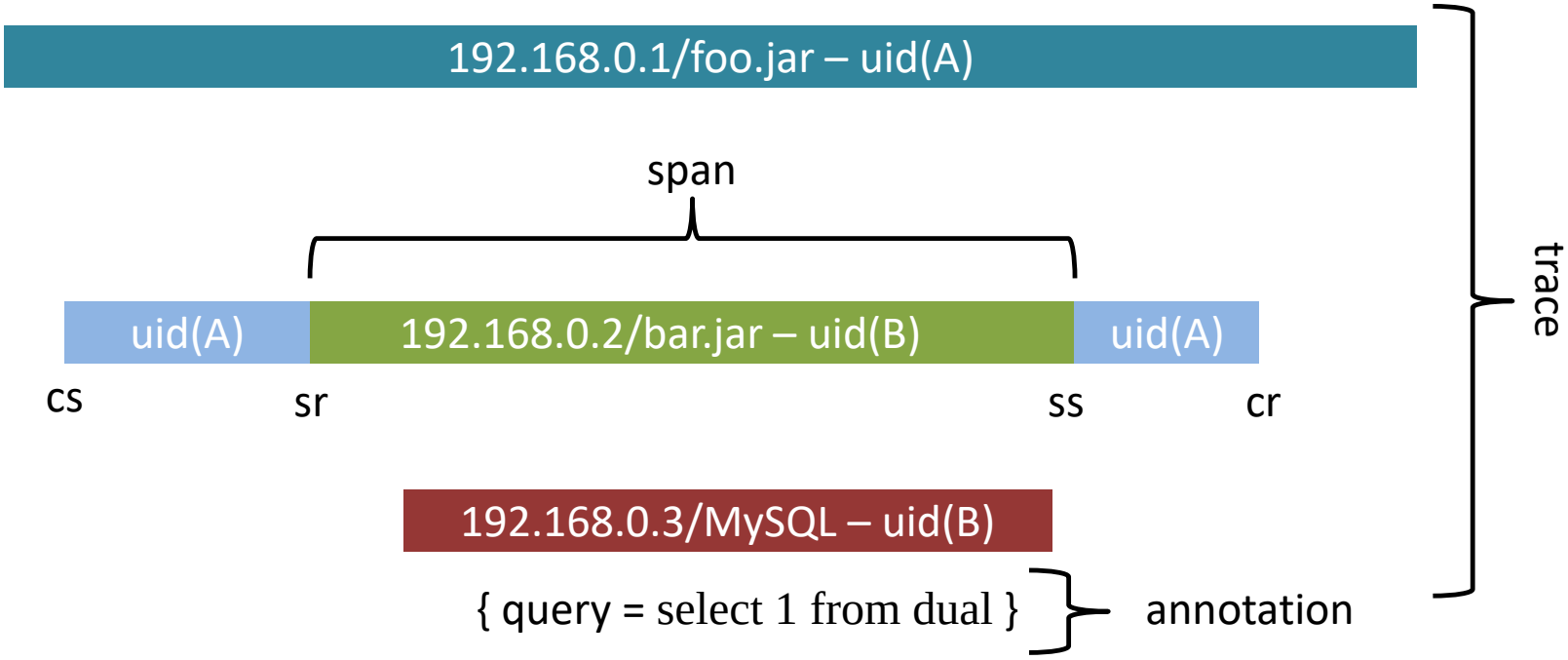
Source: Logo from zipkin.io.

How do we get it?



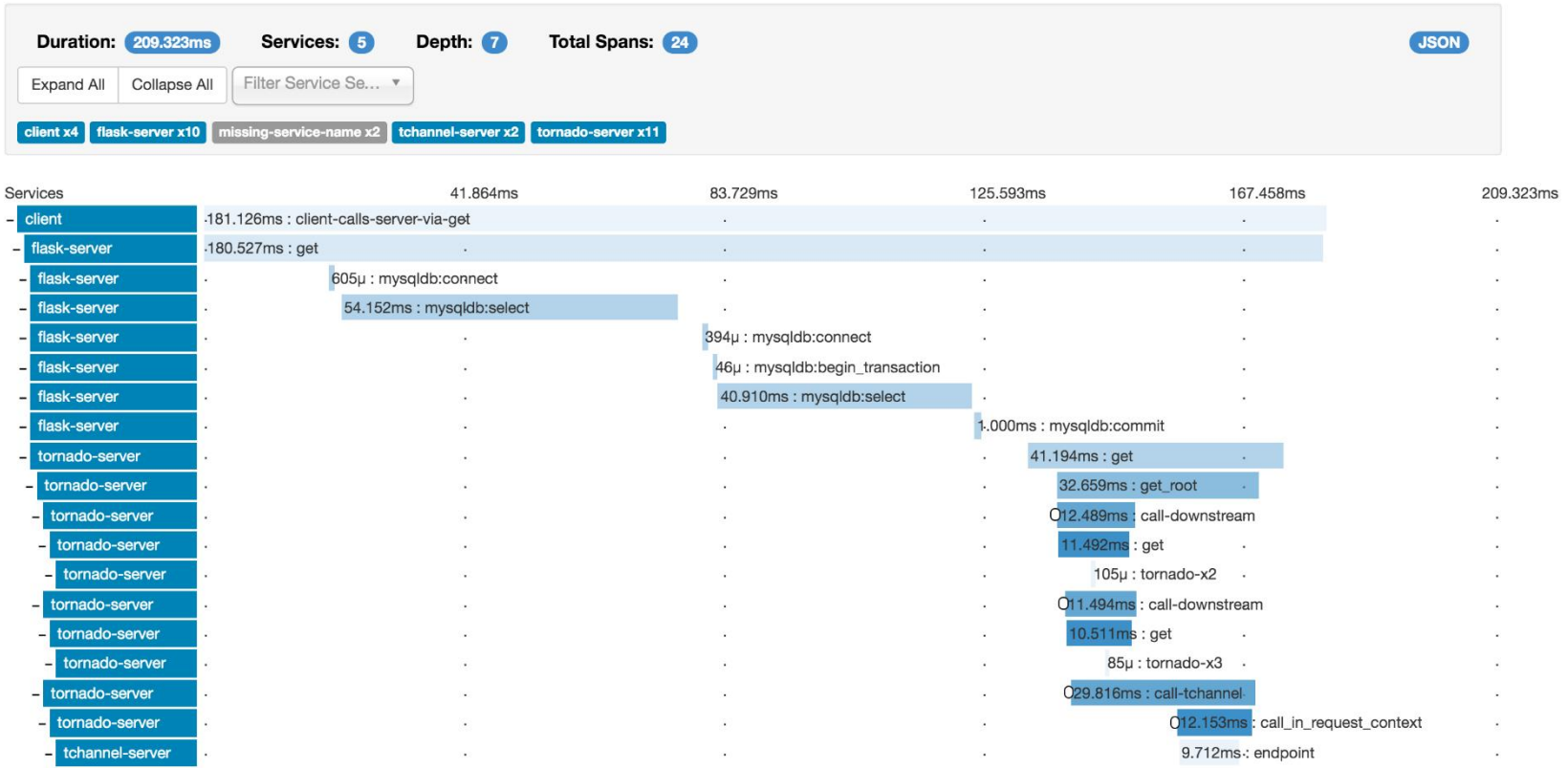
Source: Logo from zipkin.io.

How do we get it?



Source: Logo from zipkin.io.

How do we get it?



Source: Zipkin screenshot.

How do we get it?

```
Span.Builder span = Span.builder()
   .traceId(42L)
   .name("foo")
   .id(48L)
   .timestamp(System.currentTimeMillis());

long now = System.nanoTime();
// do hard work

span = span.duration(System.nanoTime() - now);
// send span to server
```

How do we get it?

```
Span.Builder span = Span.builder()
  .traceId(42L)
  .name("foo")
  .id(48L)
  .timestamp(System.currentTimeMillis());

long now = System.nanoTime();
// do hard work

span = span.duration(System.nanoTime() - now);
// send span to server
```

How do we get it?

```
Span.Builder span = Span.builder()
   .traceId(42L)
   .name("foo")
   .id(48L)
   .timestamp(System.currentTimeMillis());

long now = System.nanoTime();
// do hard work

span = span.duration(System.nanoTime() - now);
// send span to server
```

How do we get it?

```
Span.Builder span = Span.builder()
   .traceId(42L)
   .name("foo")
   .id(48L)
   .timestamp(System.currentTimeMillis());

long now = System.nanoTime();
// do hard work

span = span.duration(System.nanoTime() - now);
// send span to server
```

Several competing APIs:

1. Most popular are Zipkin (core) and Brave.
2. Some libraries such as Finagle (RPC) offer built-in Zipkin-compatible tracing.
3. Many plugins exist to add tracing as a drop-in to several libraries.
4. Multiple APIs exist for different non-JVM languages.

A standard to the rescue.

Source: Logo from opentracing.io.

A standard to the rescue.


```
Span span = tracer.buildSpan("foo")
  .asChildOf(parentSpan.context())
  .withTag("bar", "qux")
  .start();

// do hard work

span.finish();
```

Source: Logo from opentracing.io.

A standard to the rescue.


```
Span span = tracer.buildSpan("foo")
  .asChildOf(parentSpan.context())
  .withTag("bar", "qux")
  .start();

// do hard work

span.finish();
```

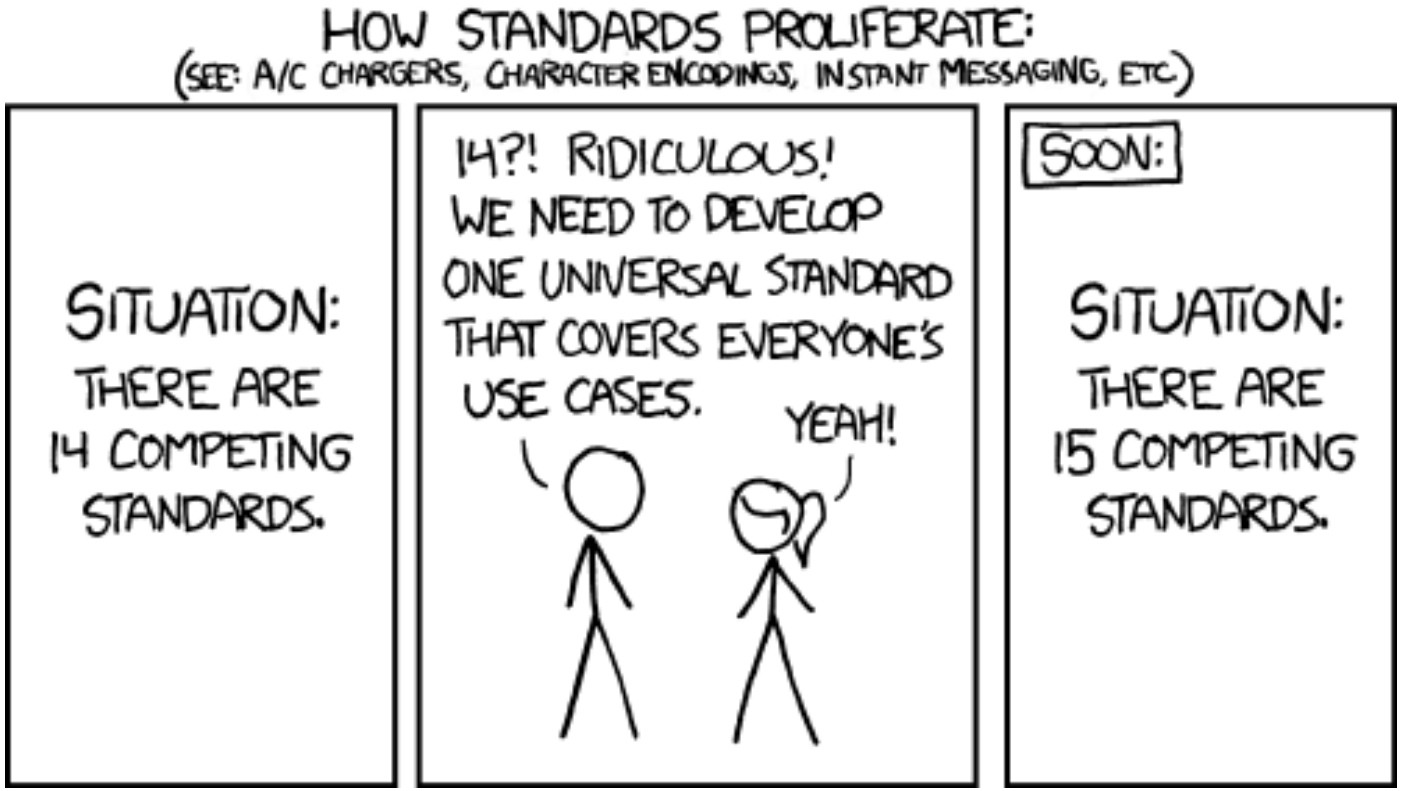
Source: Logo from opentracing.io.

A standard to the rescue.


```
Span span = tracer.buildSpan("foo")
  .asChildOf(parentSpan.context())
  .withTag("bar", "qux")
  .start();

// do hard work

span.finish();
```

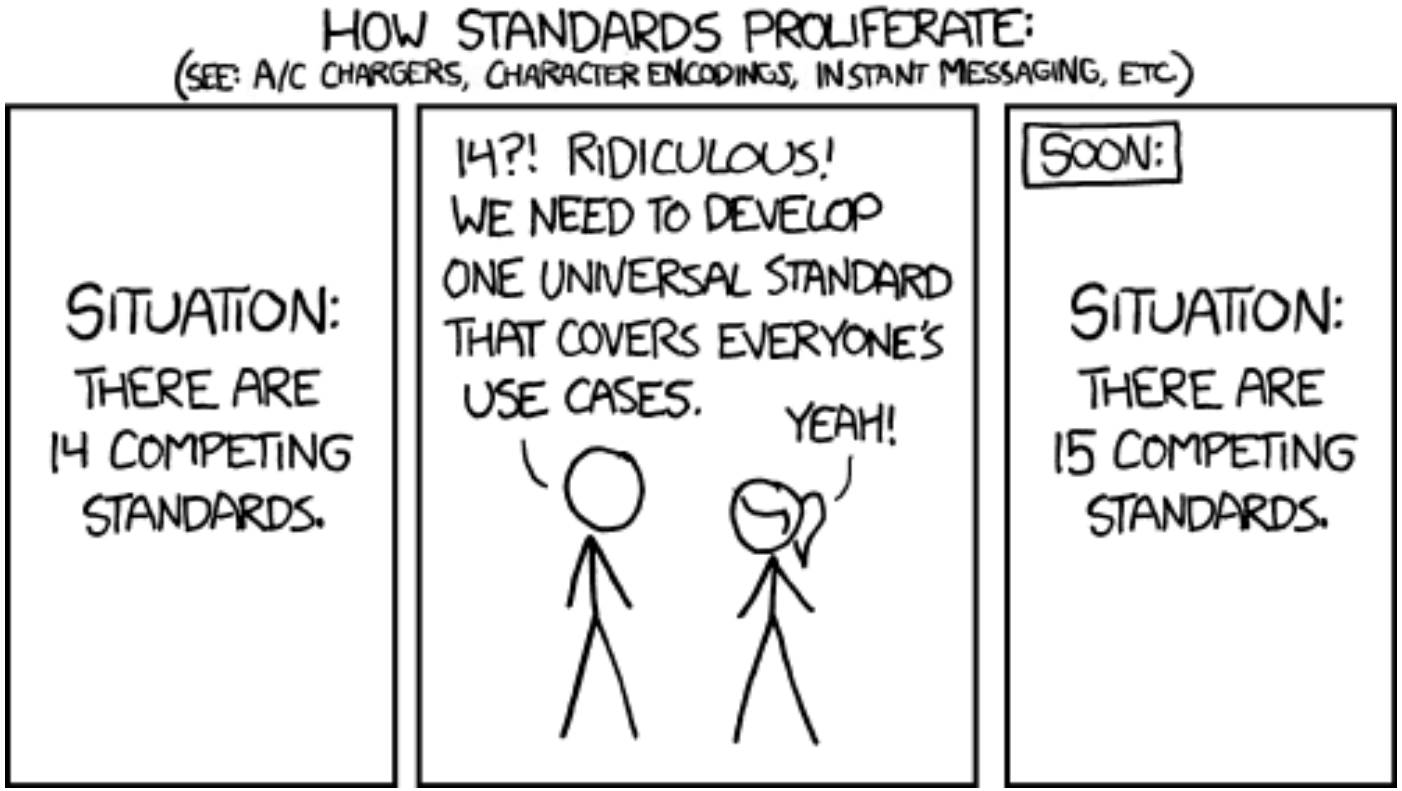
Source: Logo from opentracing.io.

A standard to the rescue.



Source: “Standards” by xkcd.

A standard to the rescue.



Problems:

1. Single missing element in chain breaks entire trace.
2. Requires explicit hand-over on every context switch.
   (Span typically stored in thread-local storage.)

Source: “Standards” by xkcd.

Outline

I. Inventory
II. Tracing (micro-)services
III. Implementing APM
IV. Advanced topics

Drop-in tracing.

```
public class TracingAgent {

  public static void premain(String arg, Instrumentation inst) {
    inst.addTransformer(
      (classLoader, typeName, type, pd, classFile) -> {
        if (shouldTraceClass(typeName)) {
          return addTracing(classFile);
        } else {
          return null;
        }
      });
  }

  private static boolean shouldTraceClass(String typeName) {
    return false; // TODO: implement
  }

  private static byte[] addTracing(byte[] binary) {
    return binary; // TODO: implement
  }
}
```

Drop-in tracing.

```
public class TracingAgent {

  public static void premain(String arg, Instrumentation inst) {
    inst.addTransformer(
      (classLoader, typeName, type, pd, classFile) -> {
        if (shouldTraceClass(typeName)) {
          return addTracing(classFile);
        } else {
          return null;
        }
      });
  }

  private static boolean shouldTraceClass(String typeName) {
    return false; // TODO: implement
  }

  private static byte[] addTracing(byte[] binary) {
    return binary; // TODO: implement
  }
}
```

Drop-in tracing.

```
public class TracingAgent {

  public static void premain(String arg, Instrumentation inst) {
    inst.addTransformer(
      (classLoader, typeName, type, pd, classFile) -> {
        if (shouldTraceClass(typeName)) {
          return addTracing(classFile);
        } else {
          return null;
        }
      });
  }

  private static boolean shouldTraceClass(String typeName) {
    return false; // TODO: implement
  }

  private static byte[] addTracing(byte[] binary) {
    return binary; // TODO: implement
  }
}
```

Drop-in tracing.

```
public class TracingAgent {

  public static void premain(String arg, Instrumentation inst) {
    inst.addTransformer(
      (classLoader, typeName, type, pd, classFile) -> {
        if (shouldTraceClass(typeName)) {
          return addTracing(classFile);
        } else {
          return null;
        }
      });
  }

  private static boolean shouldTraceClass(String typeName) {
    return false; // TODO: implement
  }

  private static byte[] addTracing(byte[] binary) {
    return binary; // TODO: implement
  }
}
```

Drop-in tracing.

```
public class TracingAgent {

  public static void premain(String arg, Instrumentation inst) {
    inst.addTransformer(
      (classLoader, typeName, type, pd, classFile) -> {
        if (shouldTraceClass(typeName)) {
          return addTracing(classFile);
        } else {
          return null;
        }
      });
  }

  private static boolean shouldTraceClass(String typeName) {
    return false; // TODO: implement
  }

  private static byte[] addTracing(byte[] binary) {
    return binary; // TODO: implement
  }
}
```

High-level instrumentation with Byte Buddy.

Code generation and manipulation library:

1. Apache 2.0 licensed.
2. Mature: Over 2 million downloads per year.
3. Requires zero byte-code competence.
4. Safe code generation (no verifier errors).
5. High-performance library (even faster than vanilla-ASM).
6. Already supports Java 9 (experimental).
7. Offers fluent API and type-safe instrumentation.

Check out **http://bytebuddy.net** and **https://github.com/raphw/byte-buddy**

Java agents with Byte Buddy

```
class Foo {
  String bar() { return "bar"; }
}

assertThat(new Foo().bar(), is("Hello World!"));
```

Java agents with Byte Buddy

```
class Foo {
  String bar() { return "bar"; }
}

assertThat(new Foo().bar(), is("Hello World!"));
```


```
public static void premain(String argument,
    Instrumentation instrumentation) {
  new AgentBuilder.Default()
    .type(named("Foo"))
    .transform( (builder, type, classLoader) ->
      builder.method(named("bar"))
             .intercept(value("Hello World!"));
    )
    .installOn(instrumentation);
}
```

Java agents with Byte Buddy

```
class Foo {
  String bar() { return "bar"; }
}

assertThat(new Foo().bar(), is("Hello World!"));
```


```
public static void premain(String argument,
    Instrumentation instrumentation) {
  new AgentBuilder.Default()
    .type(named("Foo"))
    .transform( (builder, type, classLoader) ->
      builder.method(named("bar"))
             .intercept(value("Hello World!"));
    )
    .installOn(instrumentation);
}
```

Java agents with Byte Buddy

```
class Foo {
  String bar() { return "bar"; }
}

assertThat(new Foo().bar(), is("Hello World!"));
```


```
public static void premain(String argument,
    Instrumentation instrumentation) {
  new AgentBuilder.Default()
    .type(named("Foo"))
    .transform( (builder, type, classLoader) ->
      builder.method(named("bar"))
             .intercept(value("Hello World!"));
    )
    .installOn(instrumentation);
}
```

Java agents with Byte Buddy

```
class Foo {
  String bar() { return "bar"; }
}

assertThat(new Foo().bar(), is("Hello World!"));
```


```
public static void premain(String argument,
    Instrumentation instrumentation) {
  new AgentBuilder.Default()
    .type(named("Foo"))
    .transform( (builder, type, classLoader) ->
      builder.method(named("bar"))
             .intercept(value("Hello World!"));
    )
    .installOn(instrumentation);
}
```

Inlining code with Byte Buddy advice

```
class ServletAdvice {

  @OnMethodEnter
  static void enter(@Argument(0) HttpServletRequest request) {
    String traceId = request.getHeader("X-Trace-Id");
    String method = request.getMethod();
    String uri = request.getRequestURI();
    if (traceId != null) {
      ServletTracer.continueTrace(traceId, method, uri);
    } else {
      ServletTracer.startTrace(method, uri);
    }
  }

  @OnMethodExit
  static void exit(@Argument(1) HttpServletResponse response) {
    ServletTracer.complete(response.getStatusCode());
  }
}
```

Inlining code with Byte Buddy advice

```
class ServletAdvice {

  @OnMethodEnter
  static void enter(@Argument(0) HttpServletRequest request) {
    String traceId = request.getHeader("X-Trace-Id");
    String method = request.getMethod();
    String uri = request.getRequestURI();
    if (traceId != null) {
      ServletTracer.continueTrace(traceId, method, uri);
    } else {
      ServletTracer.startTrace(method, uri);
    }
  }

  @OnMethodExit
  static void exit(@Argument(1) HttpServletResponse response) {
    ServletTracer.complete(response.getStatusCode());
  }
}
```

Inlining code with Byte Buddy advice

```
class ServletAdvice {

  @OnMethodEnter
  static void enter(@Argument(0) HttpServletRequest request) {
    String traceId = request.getHeader("X-Trace-Id");
    String method = request.getMethod();
    String uri = request.getRequestURI();
    if (traceId != null) {
      ServletTracer.continueTrace(traceId, method, uri);
    } else {
      ServletTracer.startTrace(method, uri);
    }
  }

  @OnMethodExit
  static void exit(@Argument(1) HttpServletResponse response) {
    ServletTracer.complete(response.getStatusCode());
  }
}
```

Inlining code with Byte Buddy advice

```
public class ServletTraceAgent {

  public static void premain(String arg,
                             Instrumentation inst) {
    new AgentBuilder.Default()
      .type(isSubTypeOf(Servlet.class))
      .transform( (builder, type, classLoader) ->
         builder.visit(Advice.to(ServletAdvice.class)
                             .on(named("service"))));
      ).installOn(inst);
  }
}
```

Inlining code with Byte Buddy advice

```
public class ServletTraceAgent {

  public static void premain(String arg,
                             Instrumentation inst) {
    new AgentBuilder.Default()
      .type(isSubTypeOf(Servlet.class))
      .transform( (builder, type, classLoader) ->
        builder.visit(Advice.to(ServletAdvice.class)
                            .on(named("service"))));
      ).installOn(inst);
  }
}
```

Inlining code with Byte Buddy advice

```
public class ServletTraceAgent {

  public static void premain(String arg,
                             Instrumentation inst) {
    new AgentBuilder.Default()
      .type(isSubTypeOf(Servlet.class))
      .transform( (builder, type, classLoader) ->
        builder.visit(Advice.to(ServletAdvice.class)
                            .on(named("service"))));
      ).installOn(inst);
  }
}
```

Inlining code with Byte Buddy advice

```
public class ServletTraceAgent {

  public static void agentmain(String arg,
                               Instrumentation inst) {
    new AgentBuilder.Default()
      .disableClassFormatChanges()
      .with(AgentBuilder.RedefinitionStrategy.RETRANSFORMATION)
      .type(isSubTypeOf(Servlet.class))
      .transform( (builder, type, classLoader) ->
         builder.visit(Advice.to(ServletAdvice.class)
                             .on(named("service"))));
      ).installOn(inst);
  }
}
```

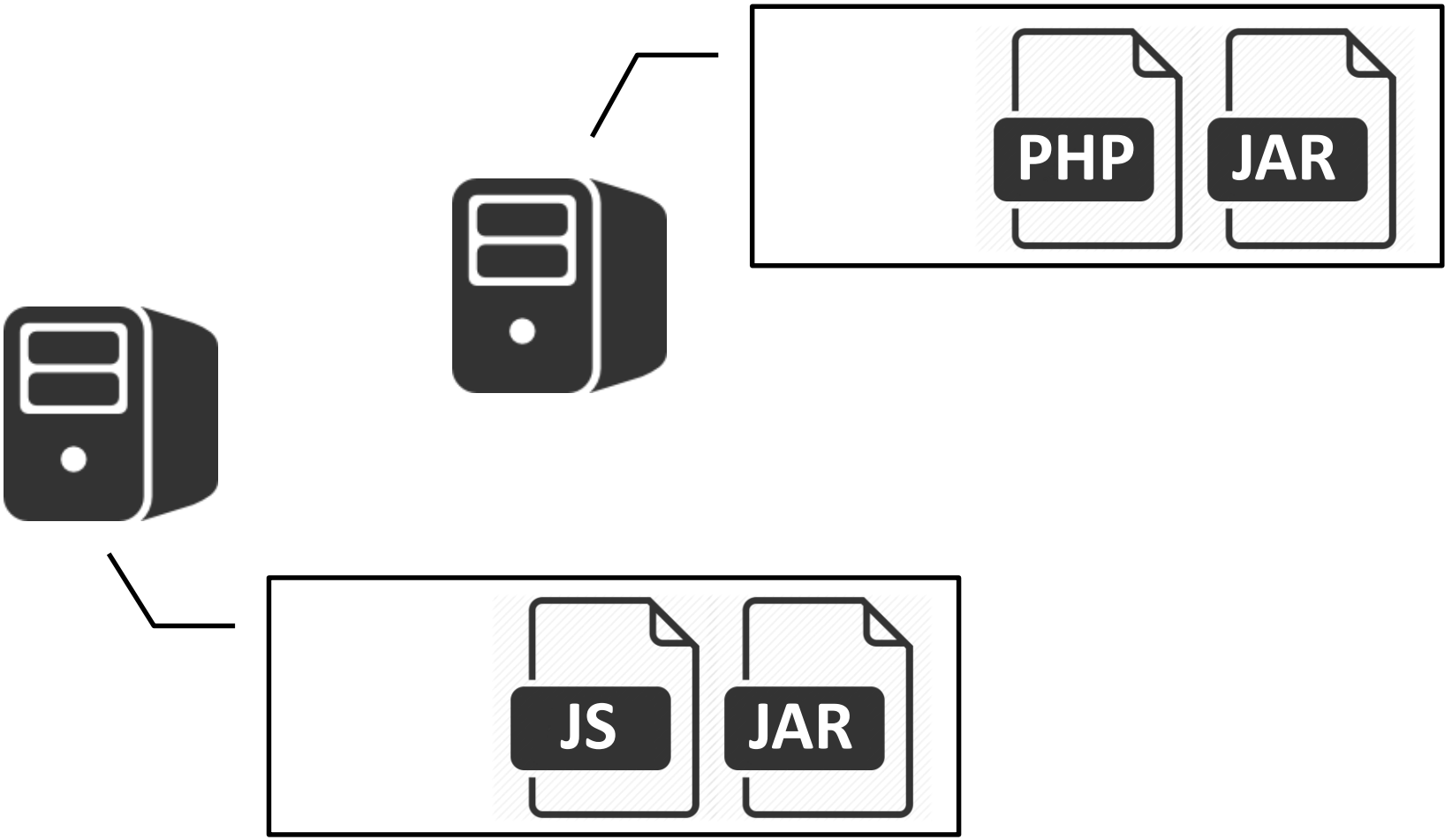
Inlining code with Byte Buddy advice

```
public class ServletTraceAgent {

  public static void agentmain(String arg,
                               Instrumentation inst) {
    new AgentBuilder.Default()
      .disableClassFormatChanges()
      .with(AgentBuilder.RedefinitionStrategy.RETRANSFORMATION)
      .type(isSubTypeOf(Servlet.class))
      .transform( (builder, type, classLoader) ->
        builder.visit(Advice.to(ServletAdvice.class)
                            .on(named("service"))));
      ).installOn(inst);
  }
}
```

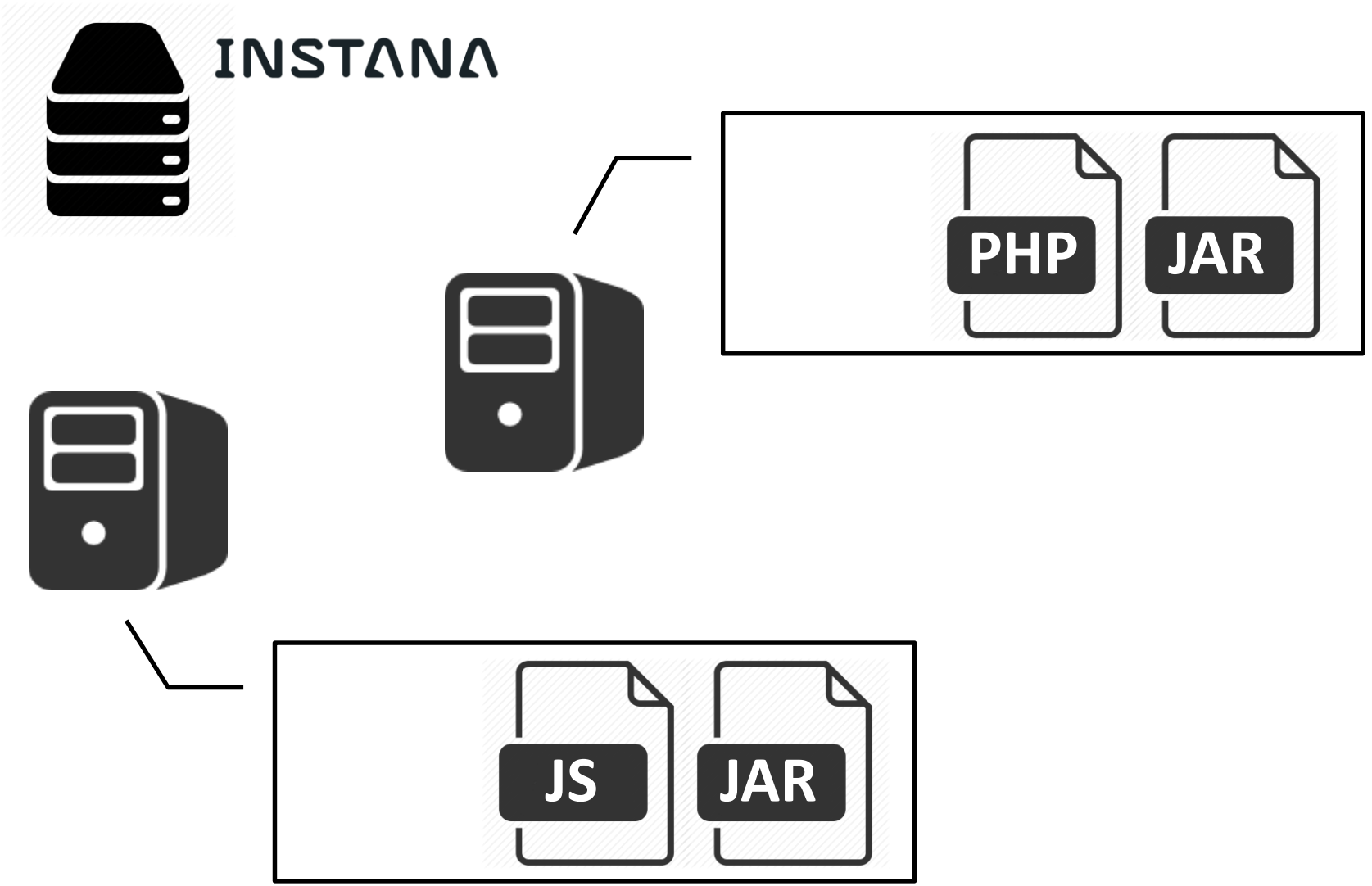
Inlining code with Byte Buddy advice

```
public class ServletTraceAgent {

  public static void agentmain(String arg,
                               Instrumentation inst) {
    new AgentBuilder.Default()
      .disableClassFormatChanges()
      .with(AgentBuilder.RedefinitionStrategy.RETRANSFORMATION)
      .type(isSubTypeOf(Servlet.class))
      .transform( (builder, type, classLoader) ->
        builder.visit(Advice.to(ServletAdvice.class)
                            .on(named("service"))));
      ).installOn(inst);
  }
}
```

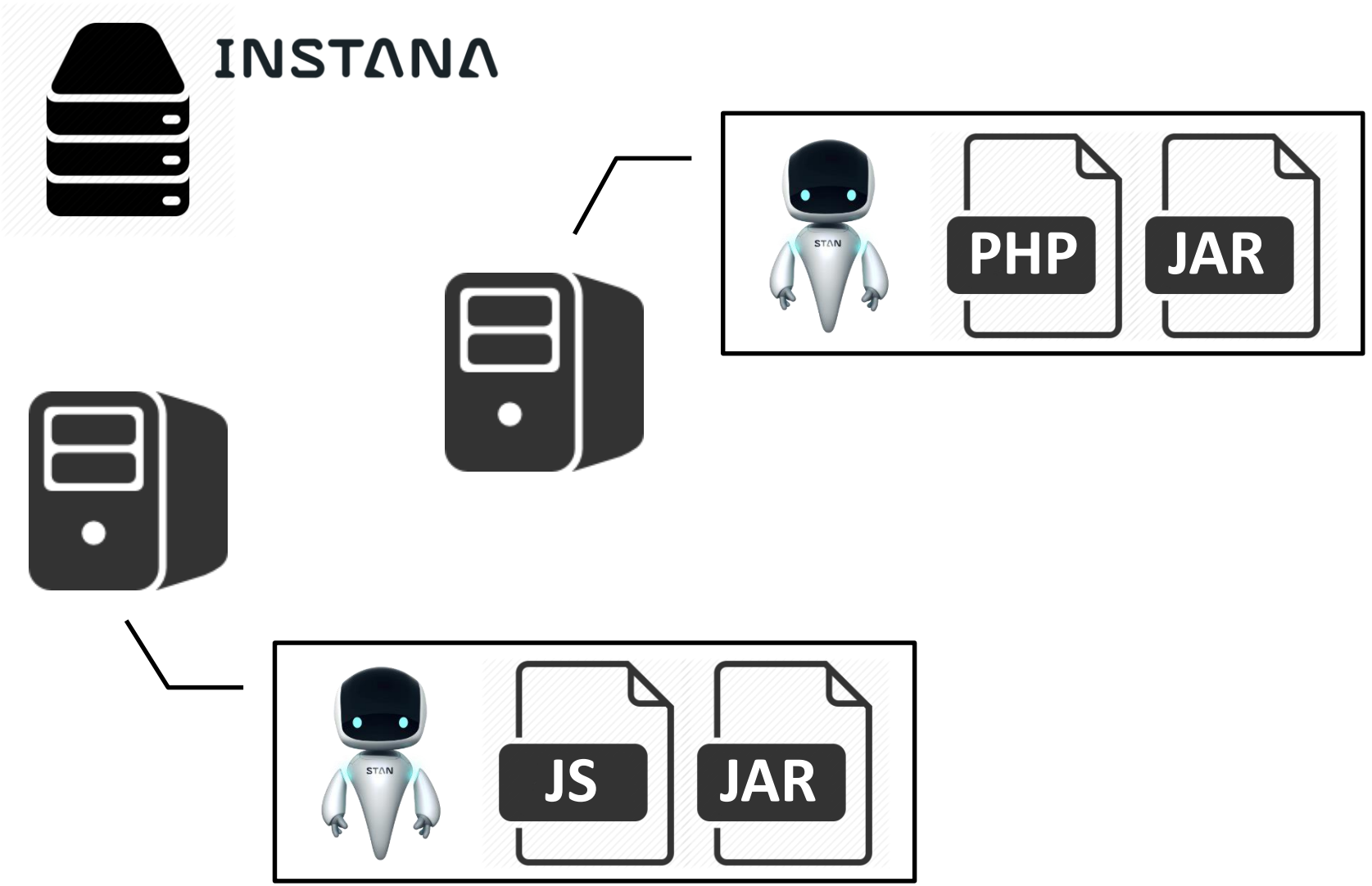
APM architecture: example of Instana



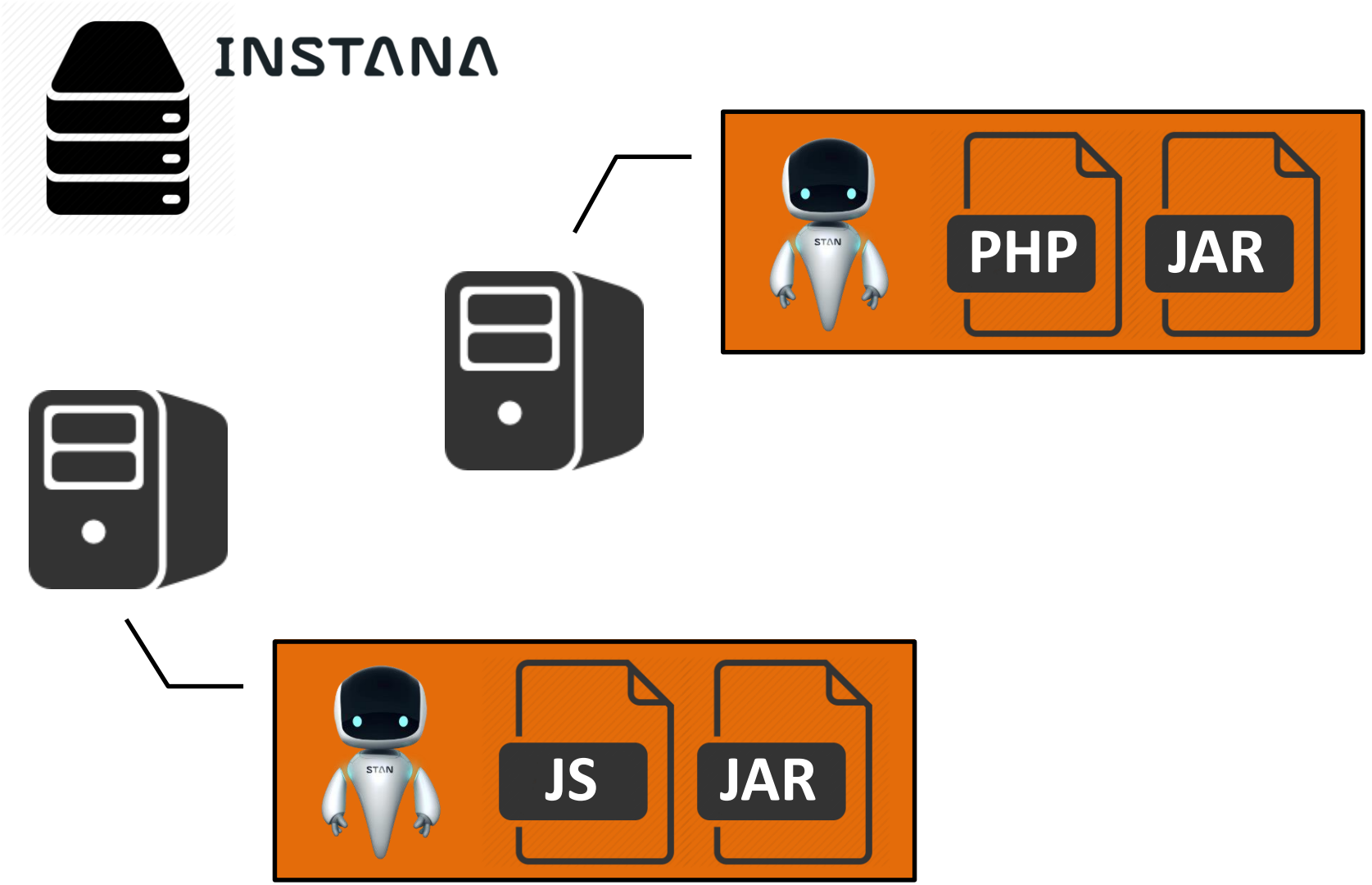
APM architecture: example of Instana



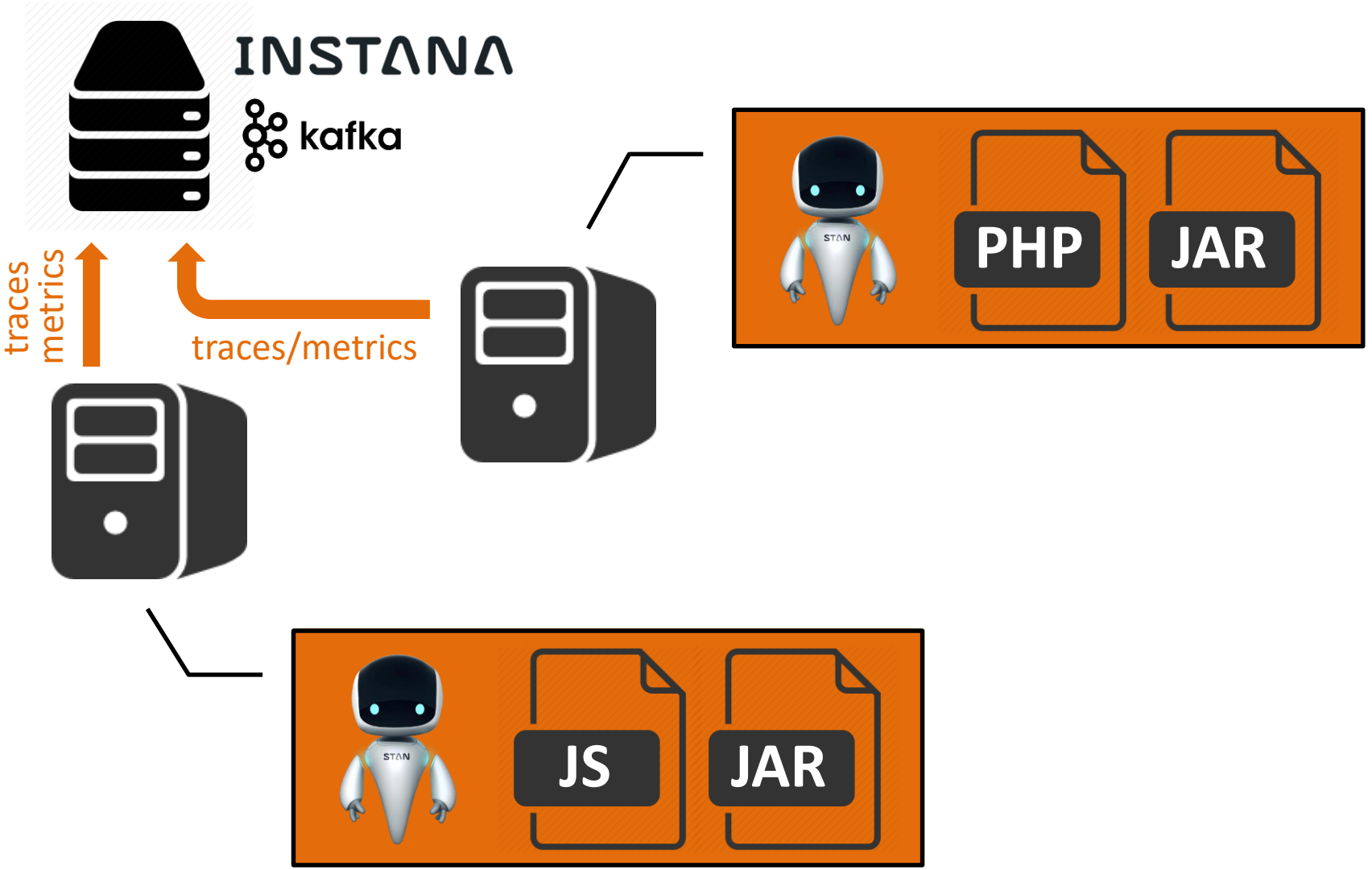
APM architecture: example of Instana



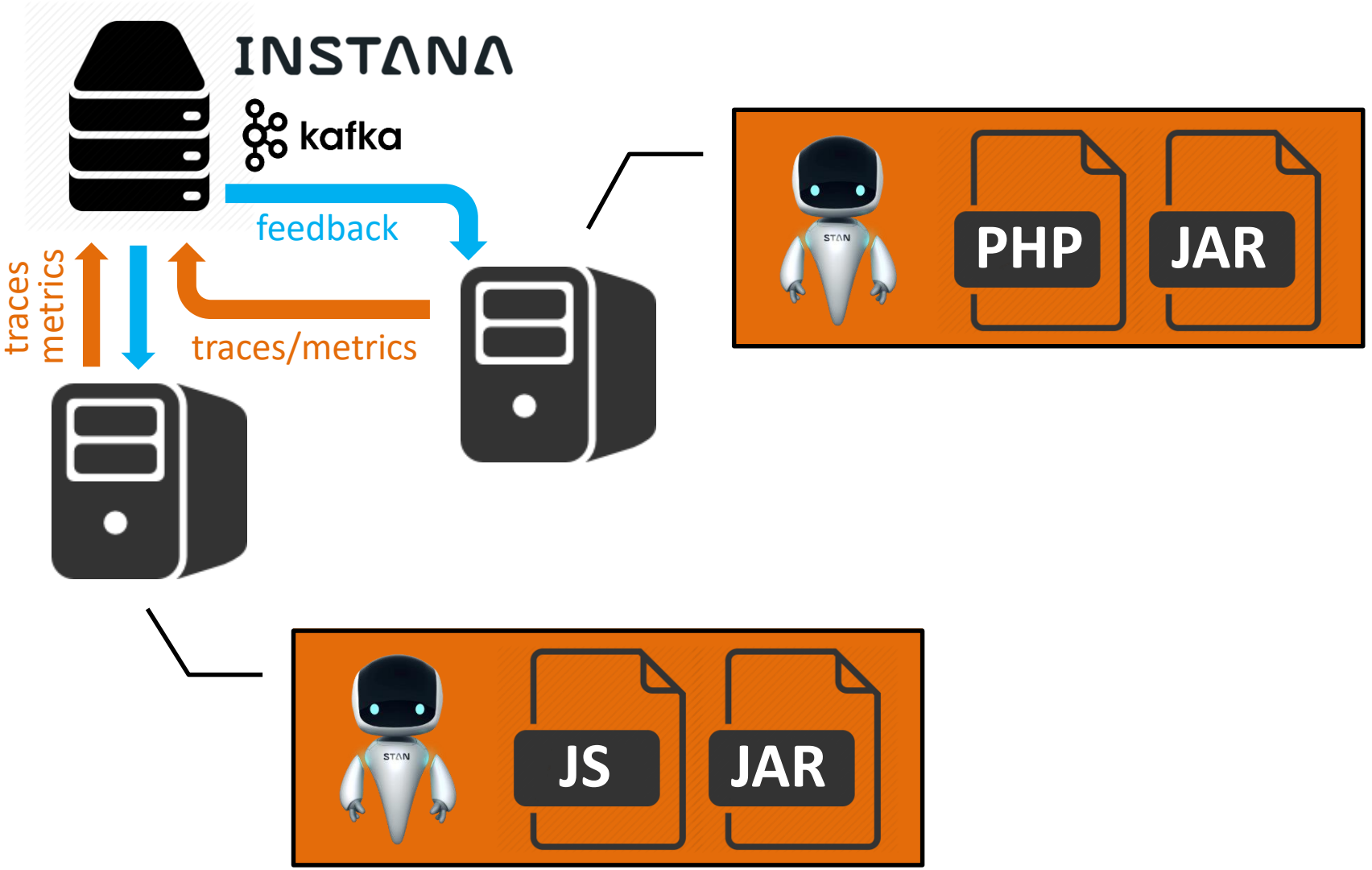
APM architecture: example of Instana



APM architecture: example of Instana

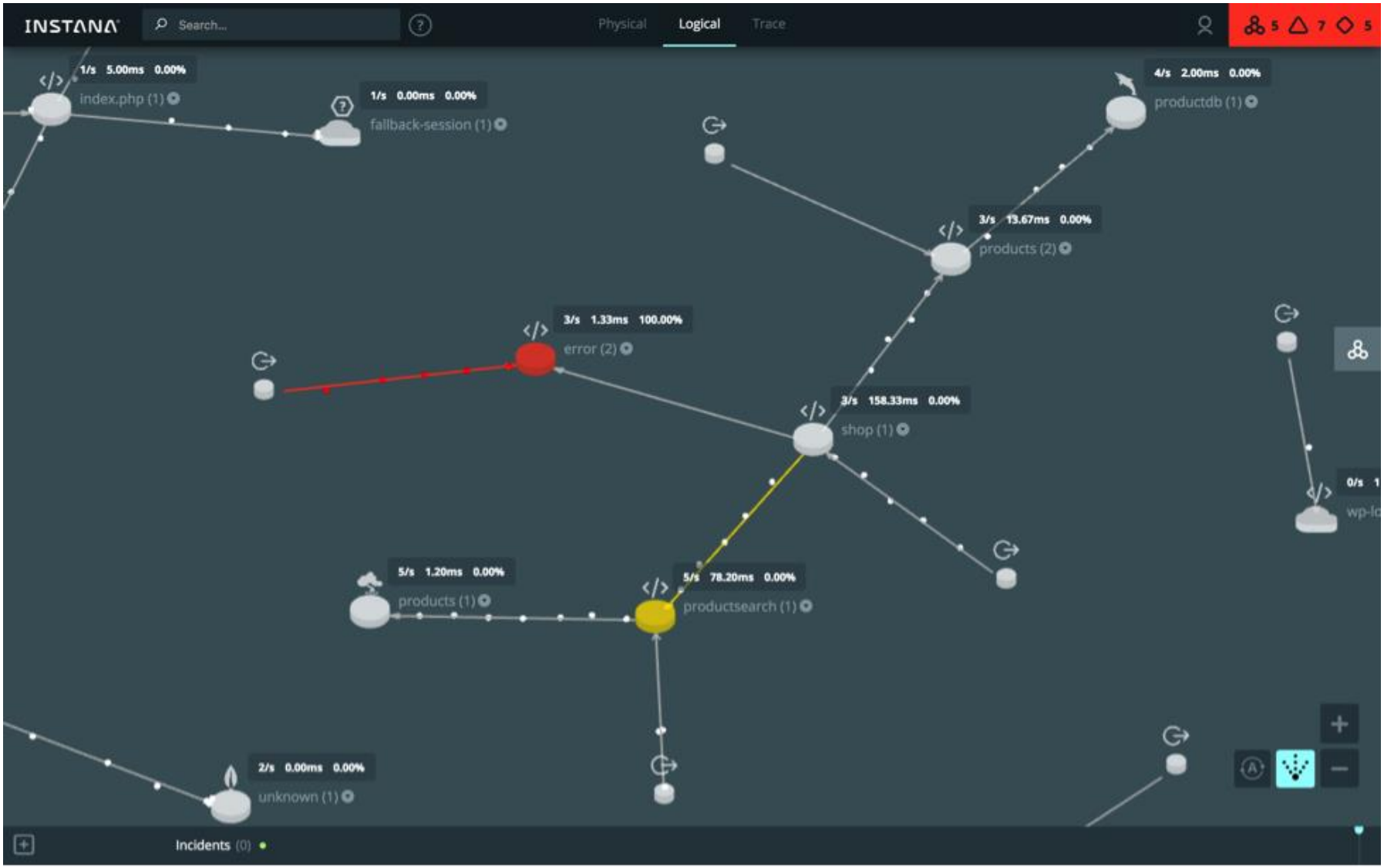


APM architecture: example of Instana



Trace view in Instana (example)

Logical view in Instana (example)



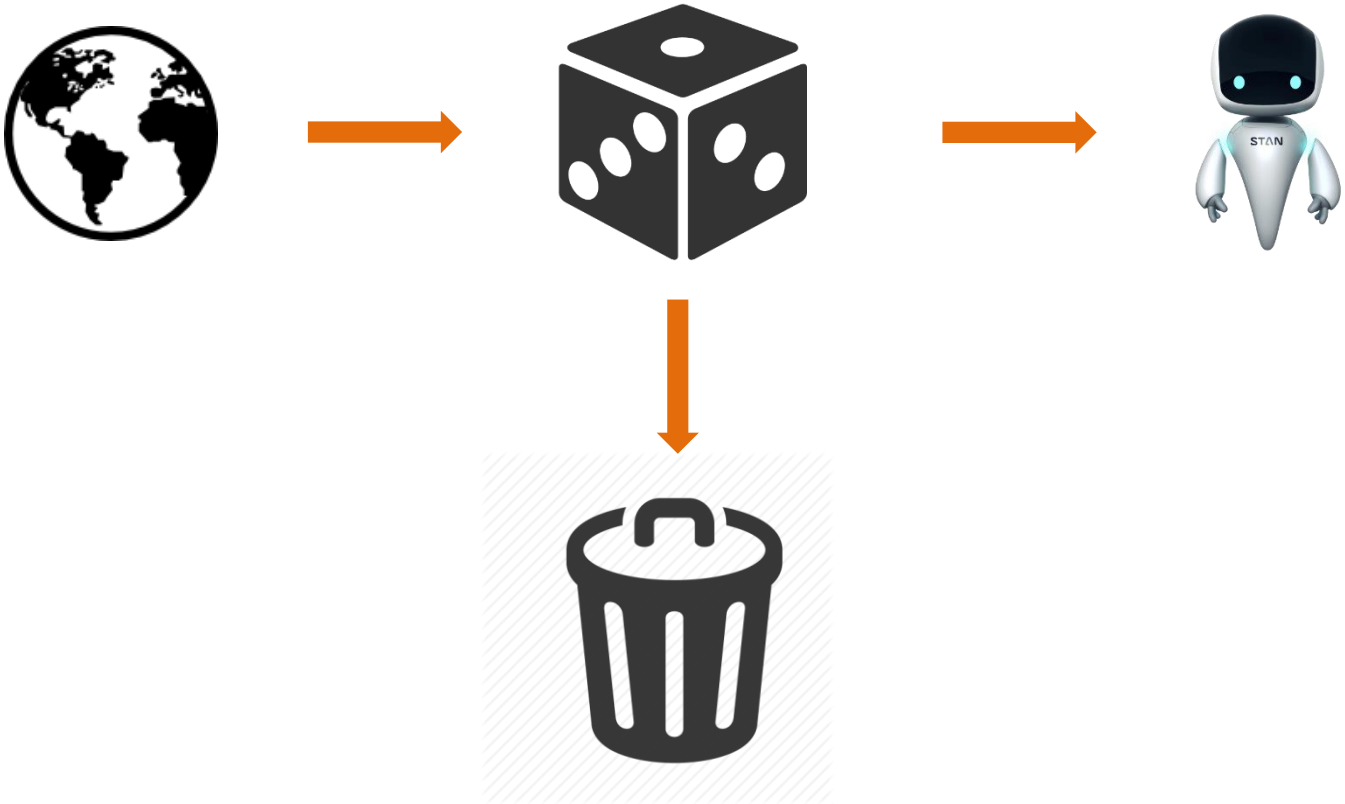
Outline

I. Inventory
II. Tracing (micro-)services
III. Implementing APM
IV. Advanced topics

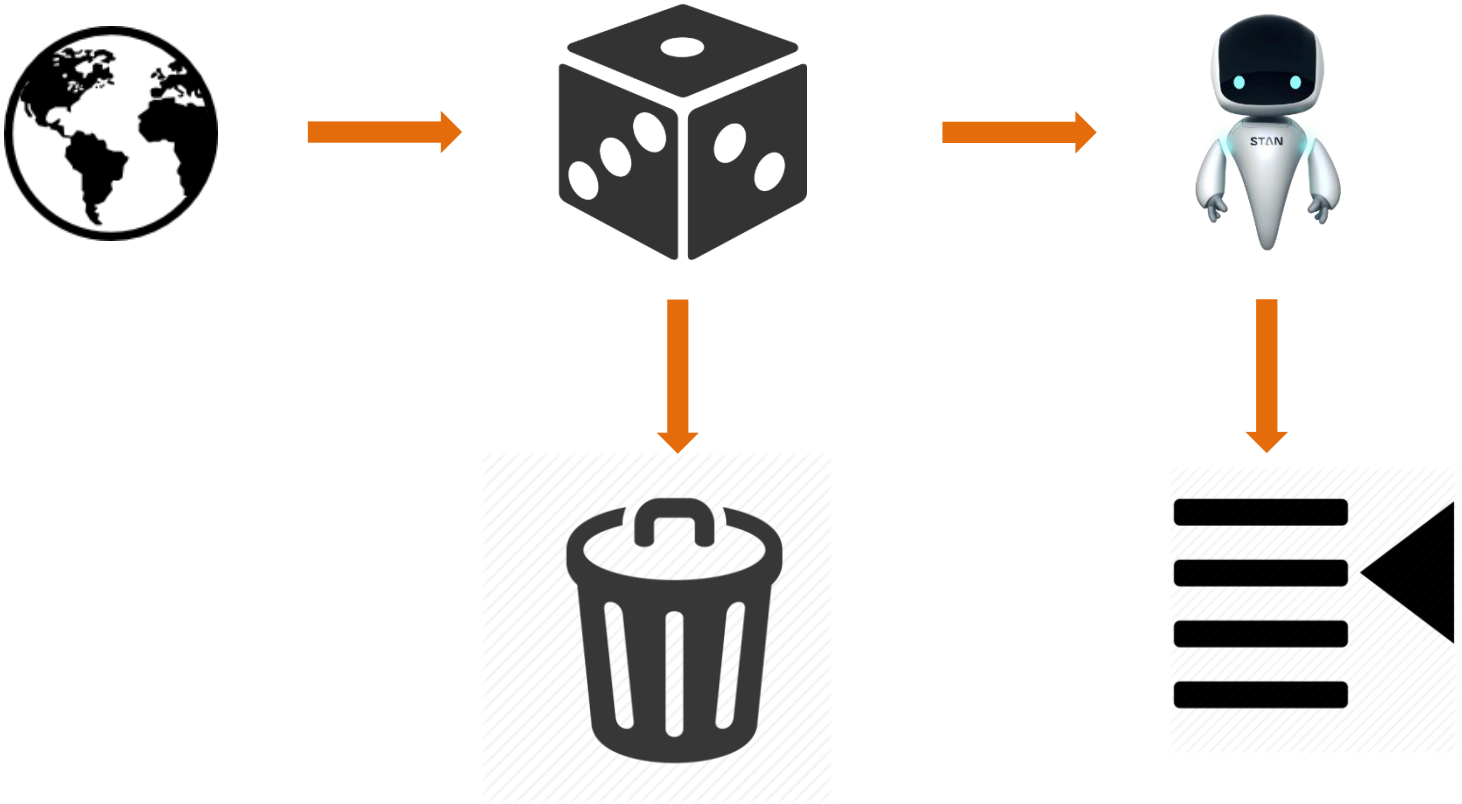
(Adaptive) sampling

(Adaptive) sampling

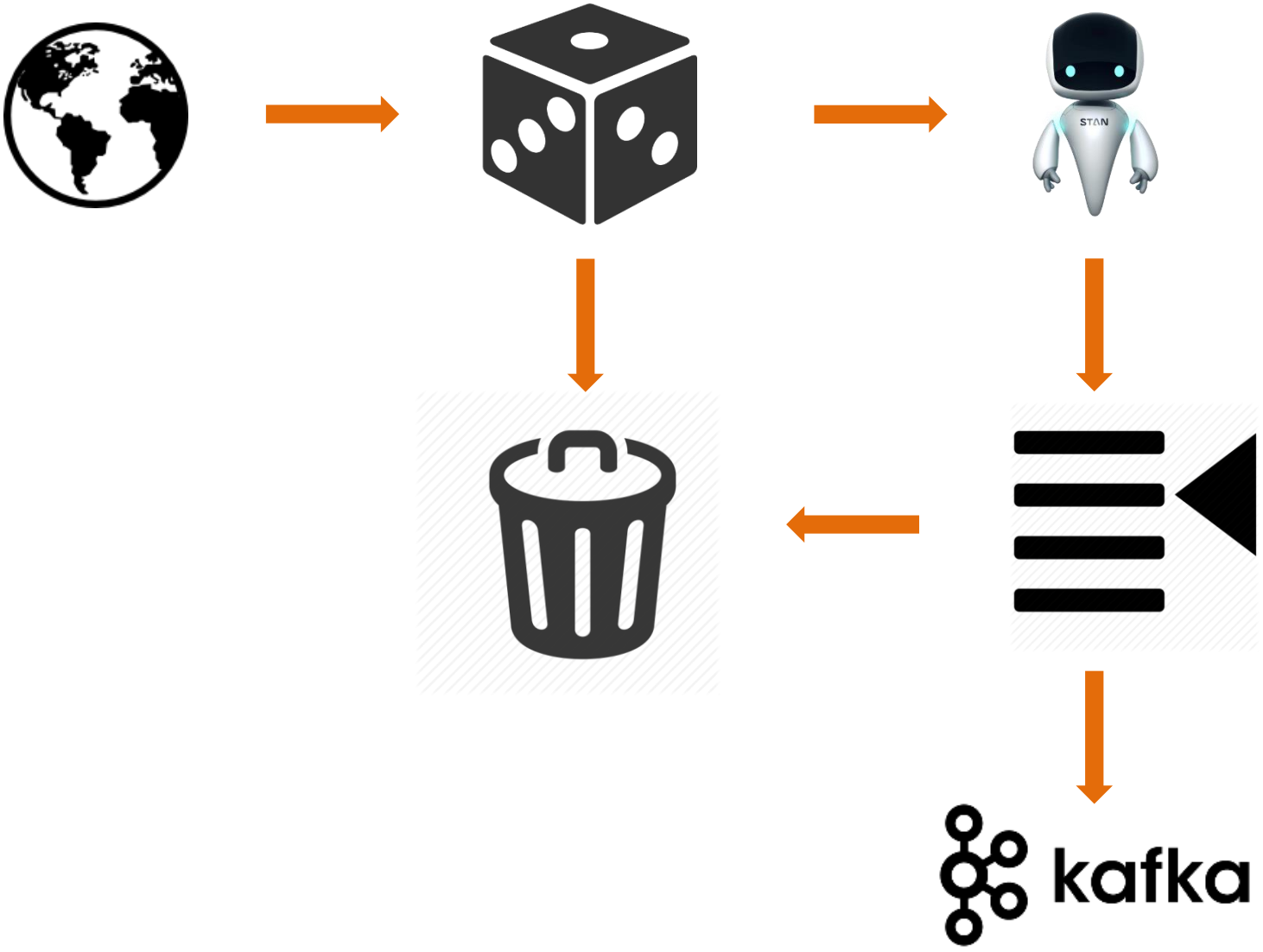
(Adaptive) sampling



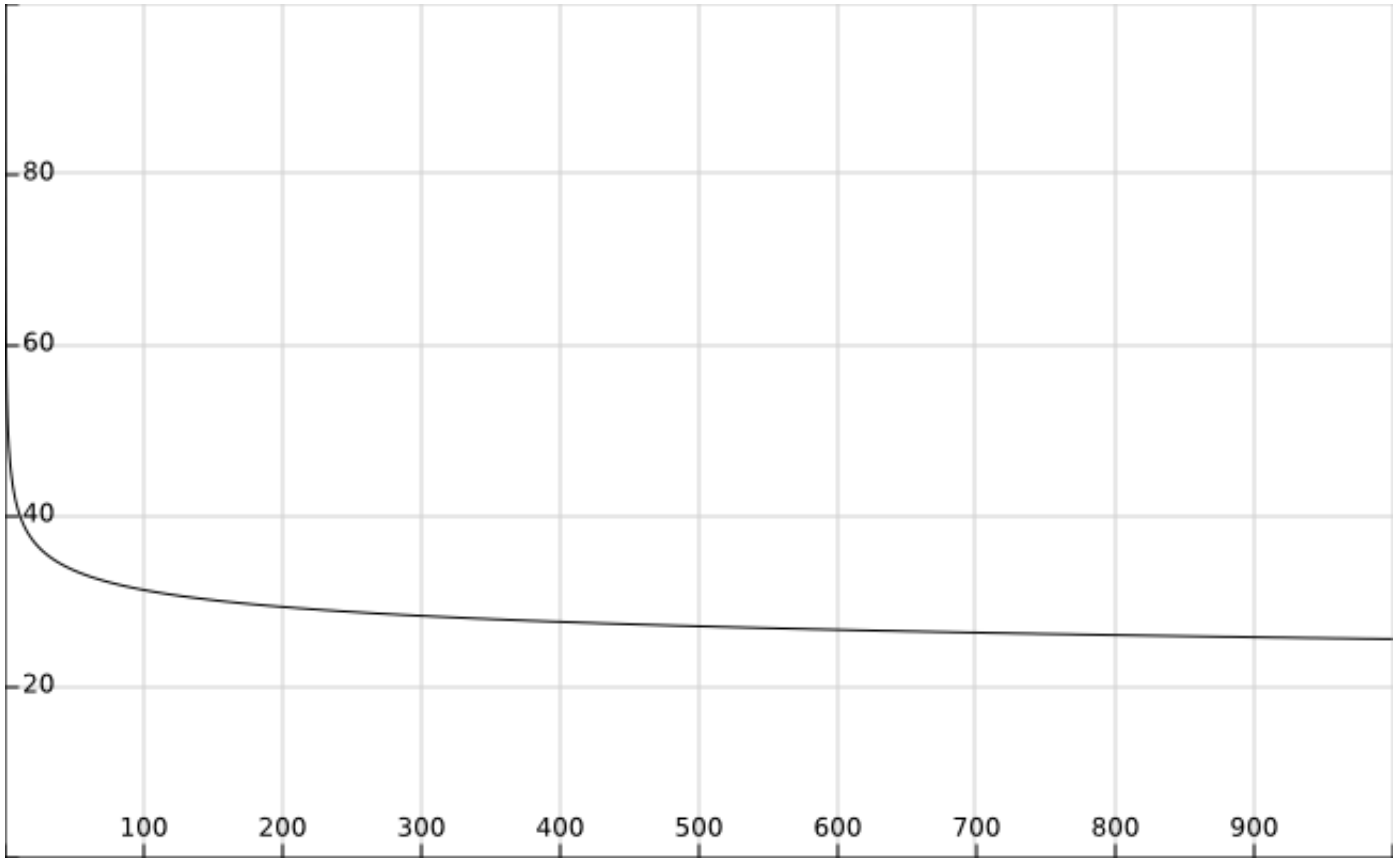
(Adaptive) sampling



(Adaptive) sampling



(Adaptive) sampling: events per second (without queue-bound)



(Adaptive) sampling: marketing “X% overhead”

```
class MyApp {
  void foo() {
    while (true) {
      handleWebRequest();
    }
  }
}
```

```
class MyOtherApp {
  void foo() {
    while (true) {
      Thread.sleep(100L);
    }
  }
}
```

(Adaptive) sampling: marketing “X% overhead”

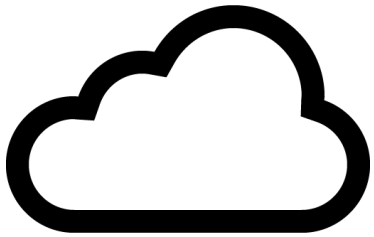
```
class MyApp {
  void foo() {
    while (true) {
      handleWebRequest();
    }
  }
}
```

```
class MyOtherApp {
  void foo() {
    while (true) {
      Thread.sleep(100L);
    }
  }
}
```

(Adaptive) sampling: marketing “X% overhead”

```
class MyApp {
  void foo() {
    while (true) {
      handleWebRequest();
    }
  }
}
```

```
class MyOtherApp {
  void foo() {
    while (true) {
      Thread.sleep(100L);
    }
  }
}
```



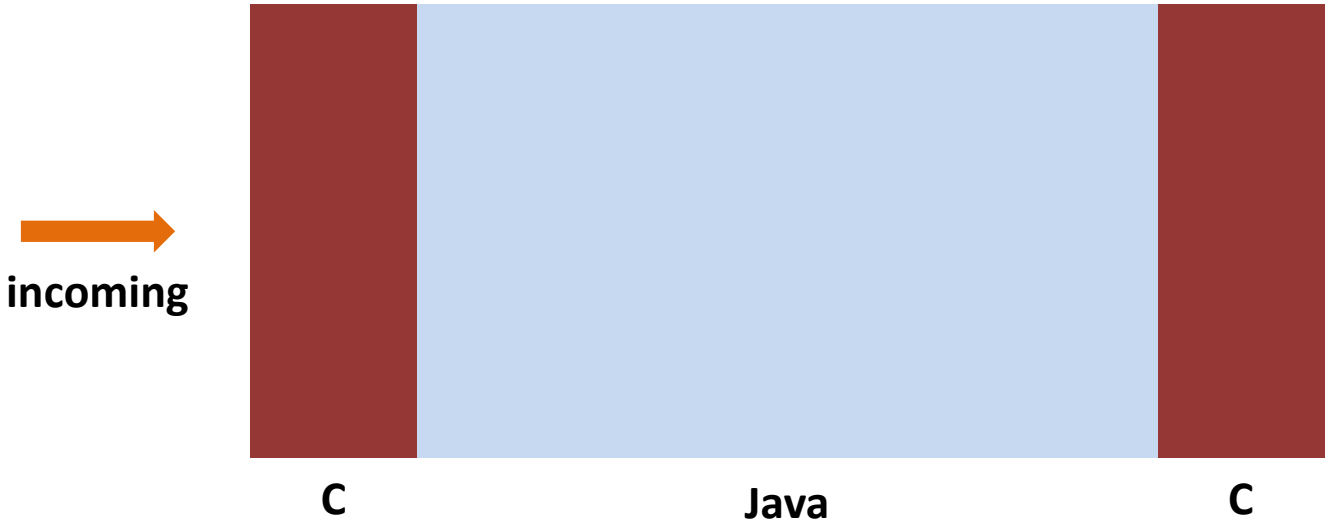
(Adaptive) sampling: marketing “X% overhead”

```
class MyApp {
  void foo() {
    while (true) {
      handleWebRequest();
    }
  }
}
```

```
class MyOtherApp {
  void foo() {
    while (true) {
      Thread.sleep(100L);
    }
  }
}
```

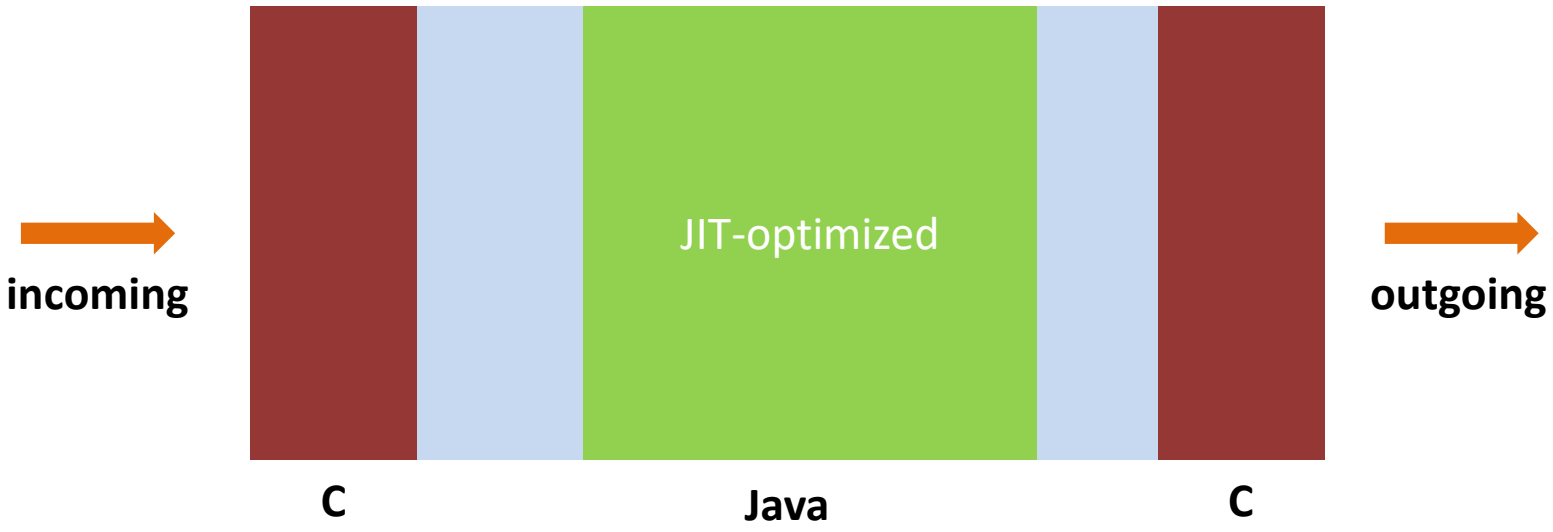
JIT-friendly tracing

JIT-friendly tracing

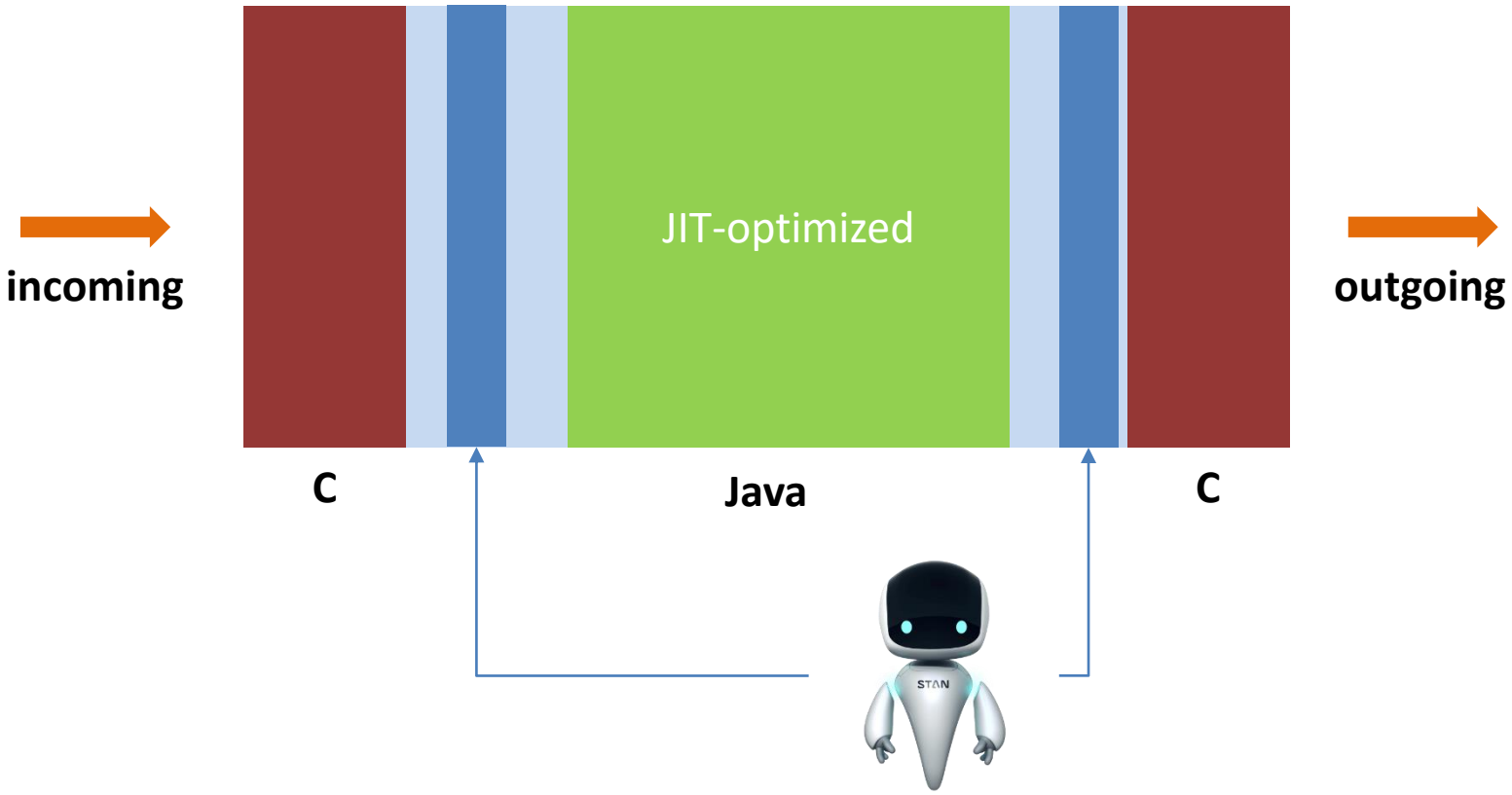


JIT-friendly tracing

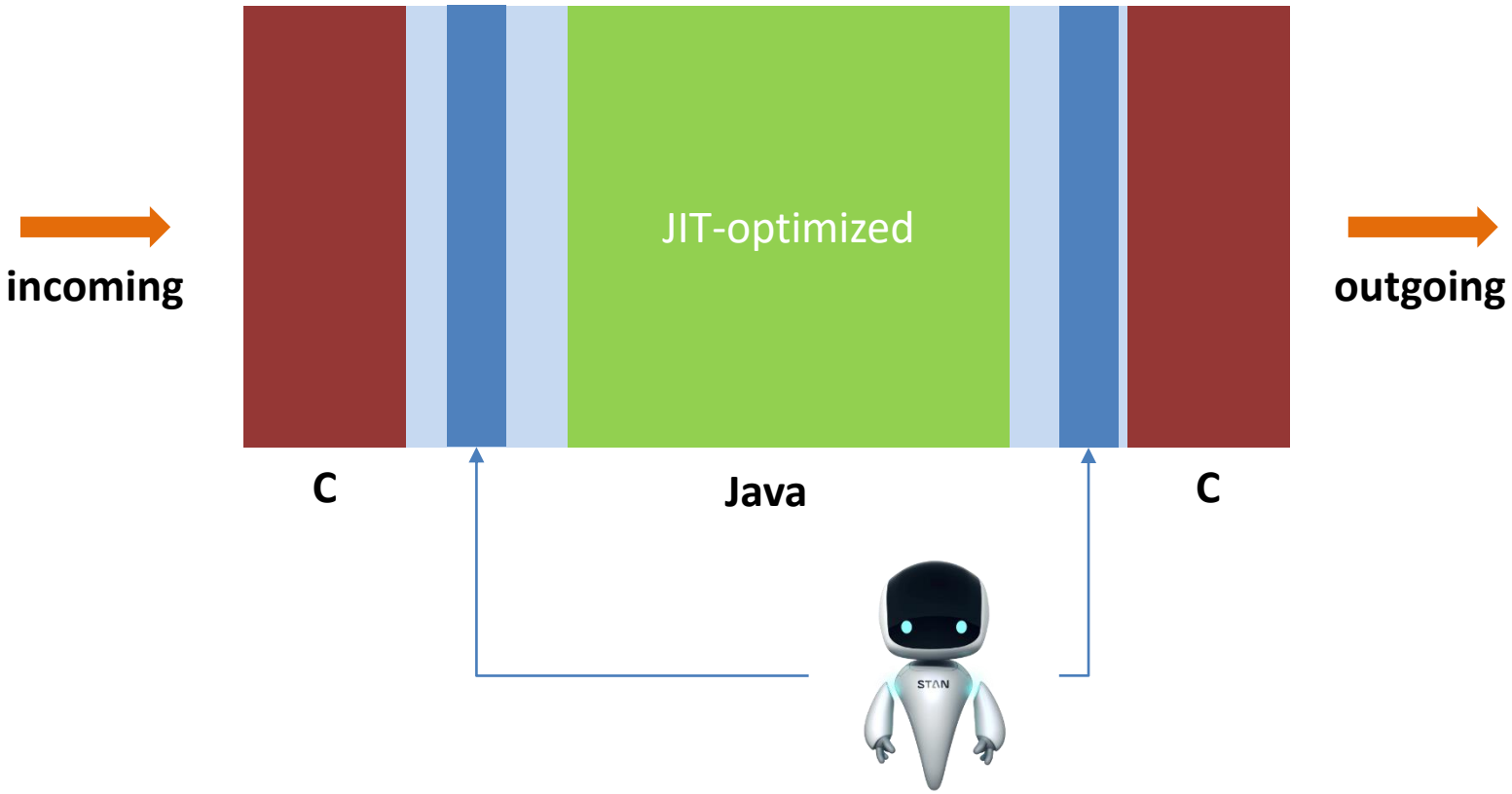
JIT-friendly tracing



JIT-friendly tracing



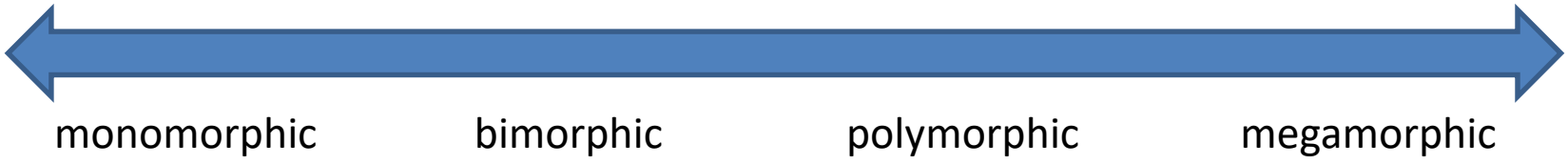
JIT-friendly tracing



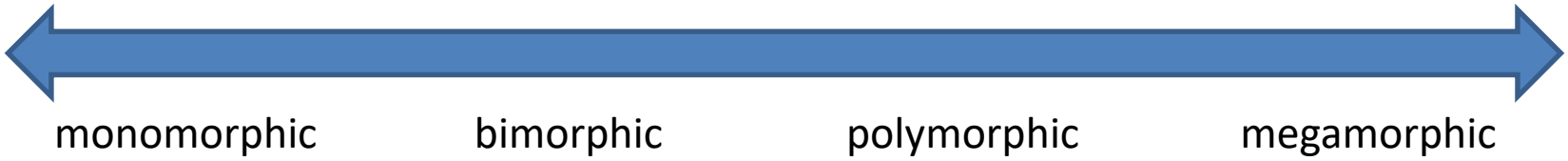
-XX:MaxInlineSize=35 (auto-reduced)
-XX:FreqInlineSize=325
-XX:InlineSmallCode=2000

-XX:MaxInlineLevel=9

JIT-friendly tracing: enforcing monomorphism

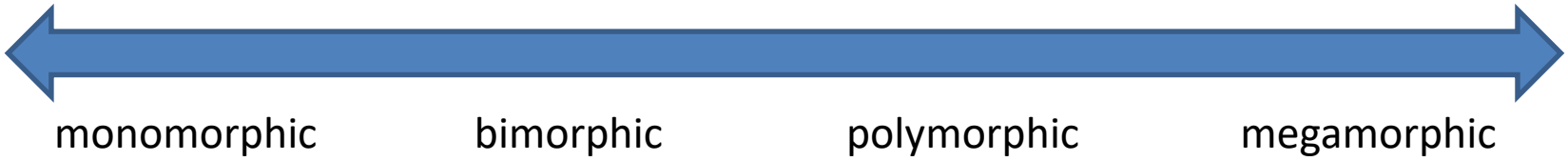


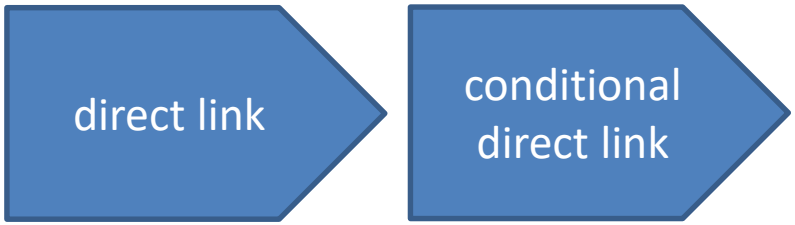
JIT-friendly tracing: enforcing monomorphism



(about 90%)

JIT-friendly tracing: enforcing monomorphism

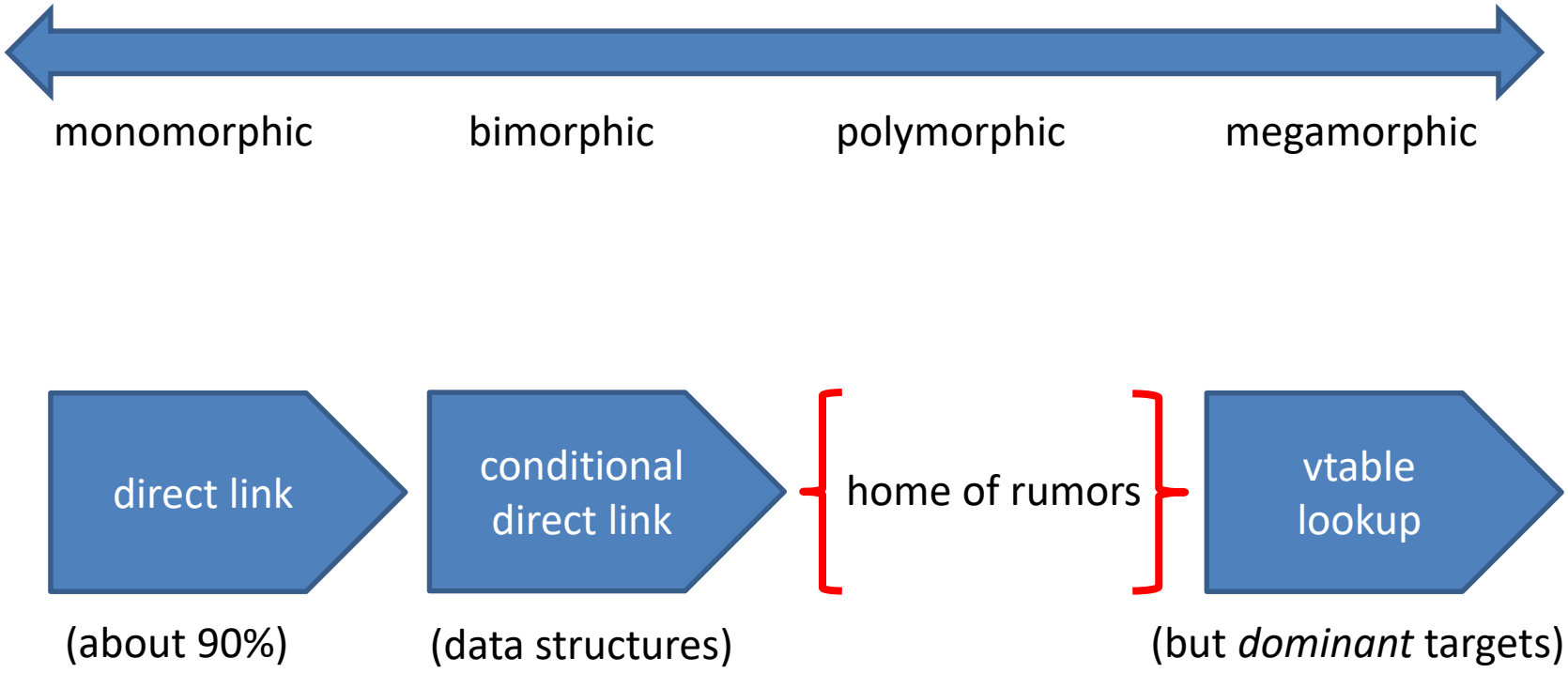




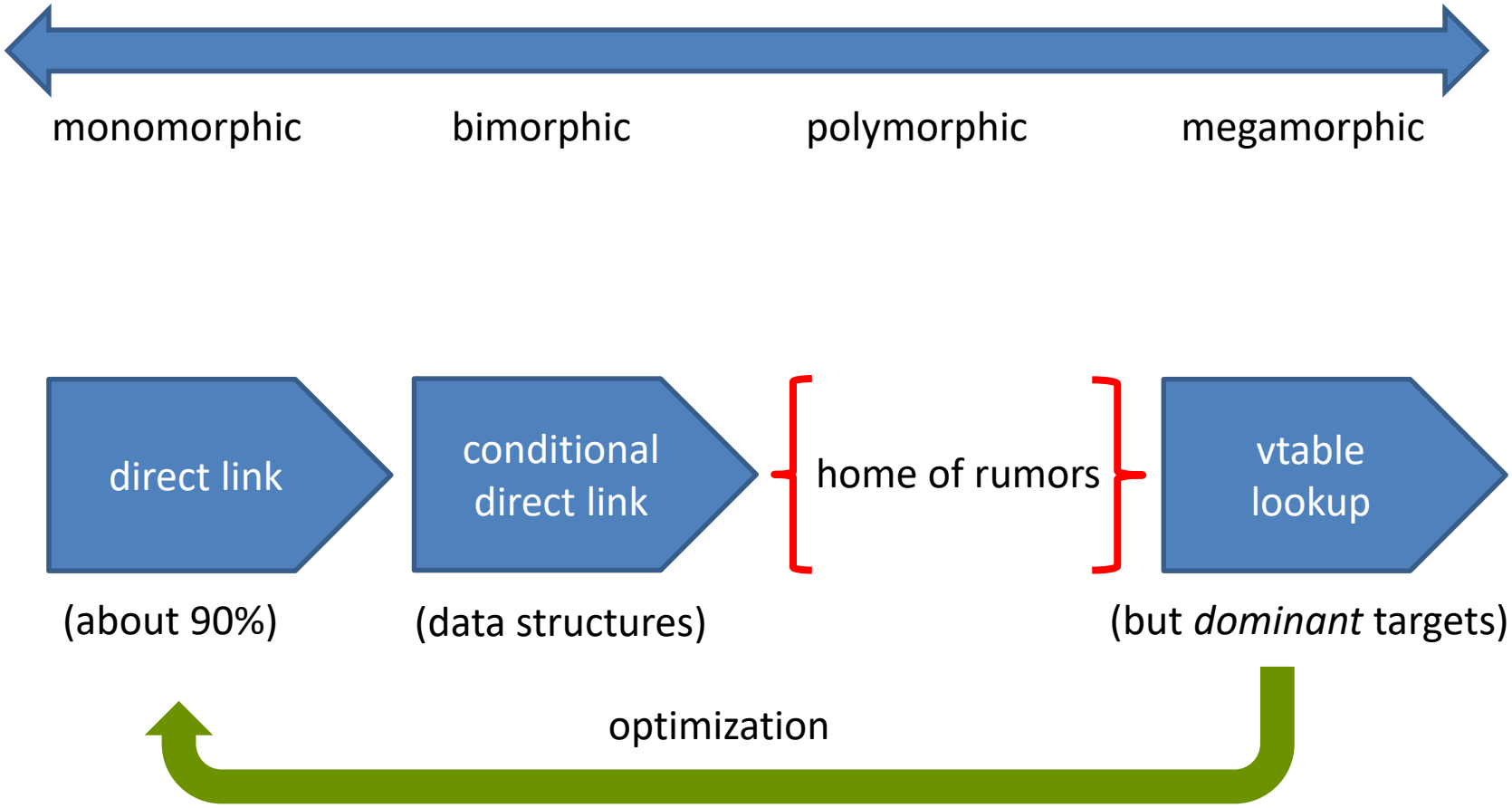
(about 90%)

(data structures)

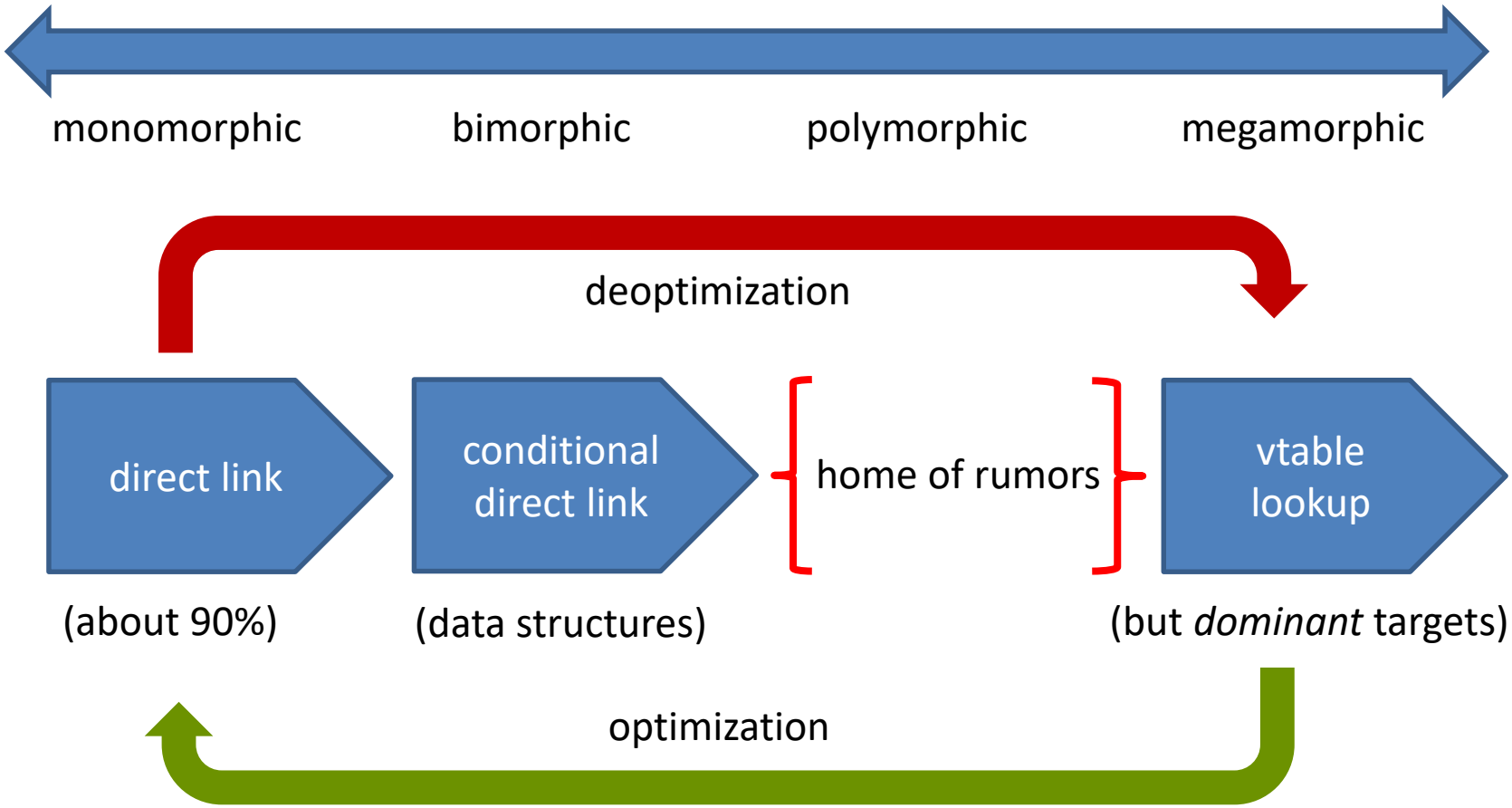
JIT-friendly tracing: enforcing monomorphism



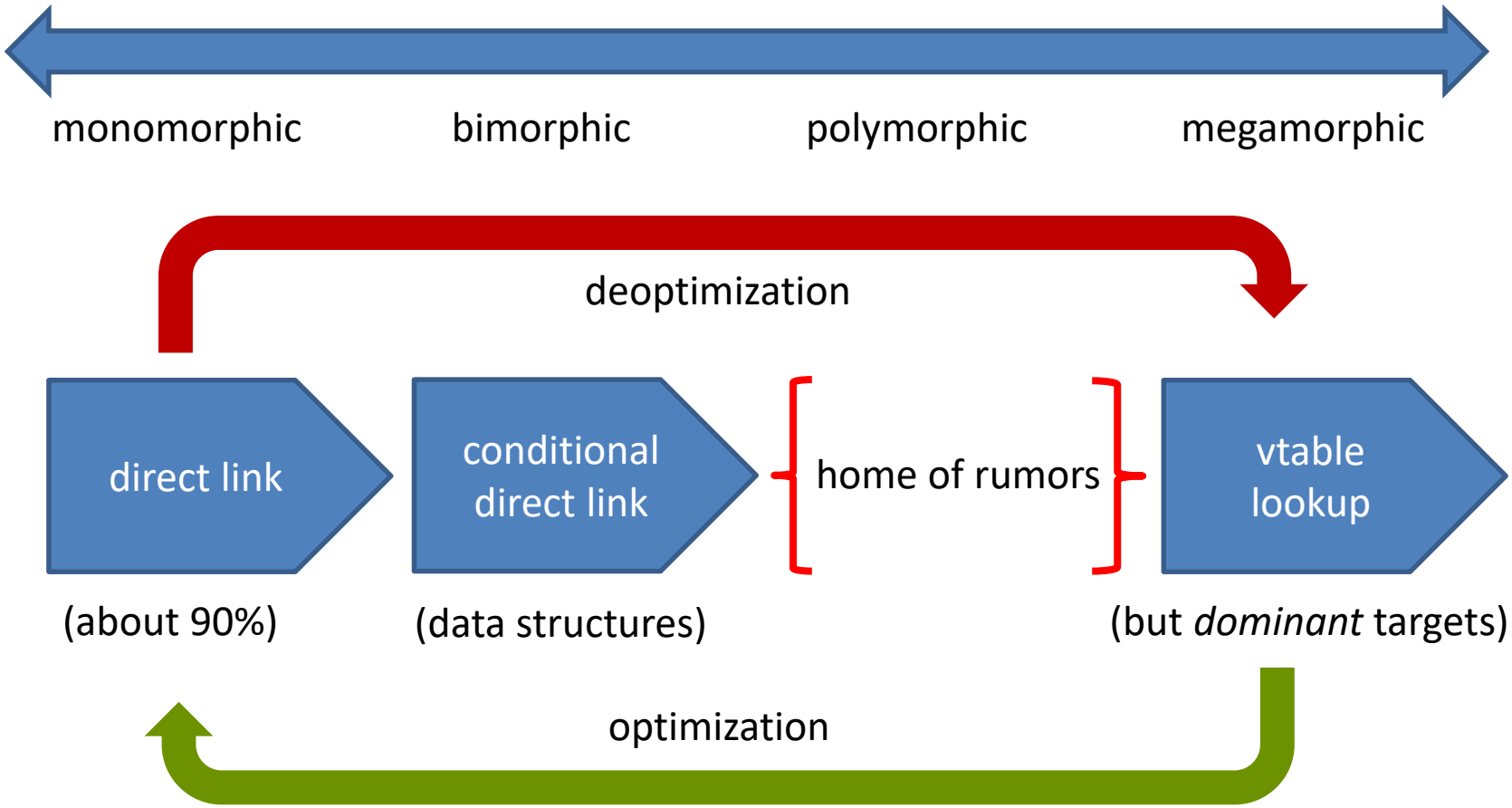
JIT-friendly tracing: enforcing monomorphism



JIT-friendly tracing: enforcing monomorphism



JIT-friendly tracing: enforcing monomorphism



Most available tracers know three types of spans: client, server and local.
This often leads to “trace call megamorphism” in production systems.

JIT-friendly tracing: copy&paste monomorphism

```
class ServletAdvice {

  @OnMethodEnter
  static void enter(@Argument(0) HttpServletRequest request) {
    String traceId = request.getHeader("X-Trace-Id");
    String method = request.getMethod();
    String uri = request.getRequestURI();
    if (traceId != null) {
      ServletTracer.continueTrace(traceId, method, uri);
    } else {
      ServletTracer.startTrace(method, uri);
    }
  }

  @OnMethodExit
  static void exit(@Argument(1) HttpServletResponse response) {
    ServletTracer.complete(response.getStatusCode());
  }
}
```

JIT-friendly tracing: copy&paste monomorphism

```
class ServletAdvice {

  @OnMethodEnter
  static void enter(@Argument(0) HttpServletRequest request) {
    String traceId = request.getHeader("X-Trace-Id");
    String method = request.getMethod();
    String uri = request.getRequestURI();
    if (traceId != null) {
      ServletTracer.continueTrace(traceId, method, uri);
    } else {
      ServletTracer.startTrace(method, uri);
    }
  }

  @OnMethodExit
  static void exit(@Argument(1) HttpServletResponse response) {
    ServletTracer.complete(response.getStatusCode());
  }
}
```

JIT-friendly tracing: copy&paste monomorphism

```
class ServletAdvice {

  @OnMethodEnter
  static void enter(@Argument(0) HttpServletRequest request) {
    String traceId = request.getHeader("X-Trace-Id");
    String method = request.getMethod();
    String uri = request.getRequestURI();
    if (traceId != null) {
      ServletTracer.continueTrace(traceId, method, uri);
    } else {
      ServletTracer.startTrace(method, uri);
    }
  }

  @OnMethodExit
  static void exit(@Argument(1) HttpServletResponse response) {
    ServletTracer.complete(response.getStatusCode());
  }
}
```

Memory-friendly tracing

```
package com.twitter.zipkin.gen;

public class Span implements Serializable {

  public volatile Long startTick;
  private long trace_id;
  private String name;
  private long id;
  private Long parent_id;
  private List<Annotation> annotations = emptyList();
  private List<BinaryAnnotation> b_annotations = emptyList();
  private Boolean debug;
  private Long timestamp;
  private Long duration;

  // ...
}
```

Memory-friendly tracing

```
package com.twitter.zipkin.gen;

public class Span implements Serializable {

  public volatile Long startTick;
  private long trace_id;
  private String name;
  private long id;
  private Long parent_id;
  private List<Annotation> annotations = emptyList();
  private List<BinaryAnnotation> b_annotations = emptyList();
  private Boolean debug;
  private Long timestamp;
  private Long duration;

  // ...
}
```

Memory-friendly tracing

```
package com.twitter.zipkin.gen;

public class Span implements Serializable {

  public volatile Long startTick;
  private long trace_id;
  private String name;
  private long id;
  private Long parent_id;
  private List<Annotation> annotations = emptyList();
  private List<BinaryAnnotation> b_annotations = emptyList();
  private Boolean debug;
  private Long timestamp;
  private Long duration;

  // ...
}
```

Memory-friendly tracing

```
package com.twitter.zipkin.gen;

public class Span implements Serializable {

  public volatile Long startTick;
  private long trace_id;
  private String name;
  private long id;
  private Long parent_id;
  private List<Annotation> annotations = emptyList();
  private List<BinaryAnnotation> b_annotations = emptyList();
  private Boolean debug;
  private Long timestamp;
  private Long duration;

  // ...
}
```

Memory-friendly tracing: Zero-garbage tracer

Span per event
Immutable events
Some privitives
Linked list attachments
Allocation rate correlates with events
Vulnerable to false-sharing
User-thread centric
Scala-style model

Span (container) per thread
Fully mutable events
All primitives (void ids)
Raw-data array annotations
Allocation rate correlates with sampled events
Ensures thread-locality
Tracer-thread centric
Java-style model

Memory-friendly tracing: Zero-garbage tracer

Span per event	Span (container) per thread
Immutable events	Fully mutable events
Some privitives	All primitives (void ids)
Linked list attachments	Raw-data array annotations
Allocation rate correlates with events	Allocation rate correlates with sampled events
Vulnerable to false-sharing	Ensures thread-locality
User-thread centric	Tracer-thread centric
Scala-style model	Java-style model

Memory-friendly tracing: Zero-garbage tracer

Span per event	Span (container) per thread
Immutable events	Fully mutable events
Some privitives	All primitives (void ids)
Linked list attachments	Raw-data array annotations
Allocation rate correlates with events	Allocation rate correlates with sampled events
Vulnerable to false-sharing	Ensures thread-locality
User-thread centric	Tracer-thread centric
Scala-style model	Java-style model

Memory-friendly tracing: Zero-garbage tracer

Span per event	Span (container) per thread
Immutable events	Fully mutable events
Some privitives	All primitives (void ids)
Linked list attachments	Raw-data array annotations
Allocation rate correlates with events	Allocation rate correlates with sampled events
Vulnerable to false-sharing	Ensures thread-locality
User-thread centric	Tracer-thread centric
Scala-style model	Java-style model

Memory-friendly tracing: Zero-garbage tracer

Span per event	Span (container) per thread
Immutable events	Fully mutable events
Some privitives	All primitives (void ids)
Linked list attachments	Raw-data array annotations
Allocation rate correlates with events	Allocation rate correlates with sampled events
Vulnerable to false-sharing	Ensures thread-locality
User-thread centric	Tracer-thread centric
Scala-style model	Java-style model

Memory-friendly tracing: Zero-garbage tracer

Span per event	Span (container) per thread
Immutable events	Fully mutable events
Some privitives	All primitives (void ids)
Linked list attachments	Raw-data array annotations
Allocation rate correlates with events	Allocation rate correlates with sampled events
Vulnerable to false-sharing	Ensures thread-locality
User-thread centric	Tracer-thread centric
Scala-style model	Java-style model

Memory-friendly tracing: Zero-garbage tracer

Span per event	Span (container) per thread
Immutable events	Fully mutable events
Some privitives	All primitives (void ids)
Linked list attachments	Raw-data array annotations
Allocation rate correlates with events	Allocation rate correlates with sampled events
Vulnerable to false-sharing	Ensures thread-locality
User-thread centric	Tracer-thread centric
Scala-style model	Java-style model

Memory-friendly tracing: Zero-garbage tracer

Span per event	Span (container) per thread
Immutable events	Fully mutable events
Some privitives	All primitives (void ids)
Linked list attachments	Raw-data array annotations
Allocation rate correlates with events	Allocation rate correlates with sampled events
Vulnerable to false-sharing	Ensures thread-locality
User-thread centric	Tracer-thread centric
Scala-style model	Java-style model

Memory-friendly tracing: Zero-garbage tracer

Span per event	Span (container) per thread
Immutable events	Fully mutable events
Some privitives	All primitives (void ids)
Linked list attachments	Raw-data array annotations
Allocation rate correlates with events	Allocation rate correlates with sampled events
Vulnerable to false-sharing	Ensures thread-locality
User-thread centric	Tracer-thread centric
Scala-style model	Java-style model

Span identification

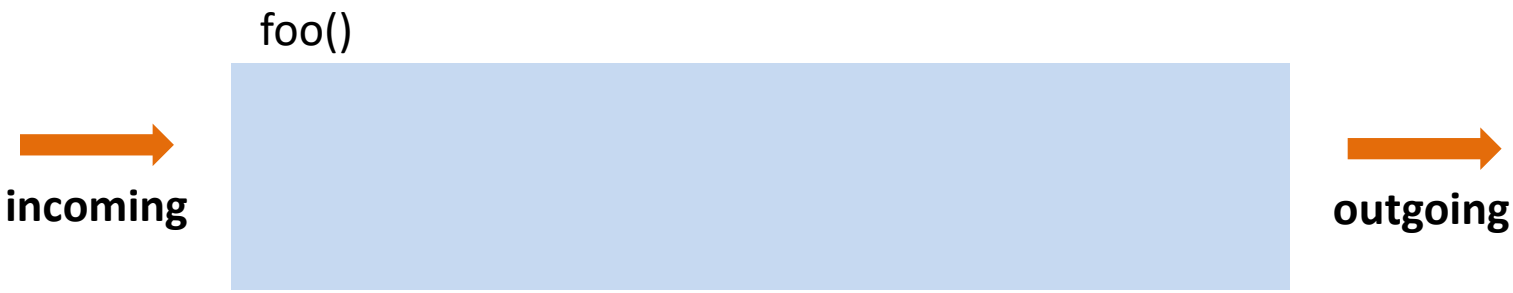
```
class MyBatchFramework {

  void doBatchJob() {
    // do hard work...
  }
}
```

Span identification

```
class MyBatchFramework {
  @com.instana.sdk.Span("trace-me")
   void doBatchJob() {
     // do hard work...
   }
}
```

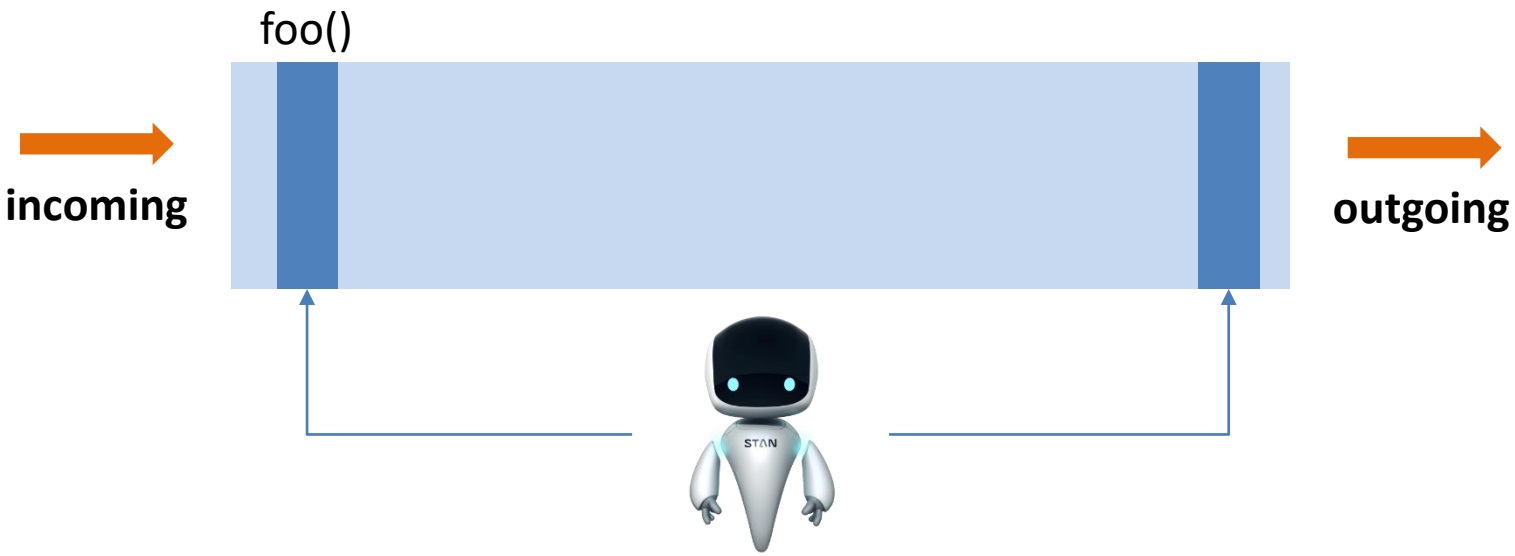
Span identification

```
class MyBatchFramework {
  @com.instana.sdk.Span("trace-me")
   void doBatchJob() {
     // do hard work...
   }
}
```

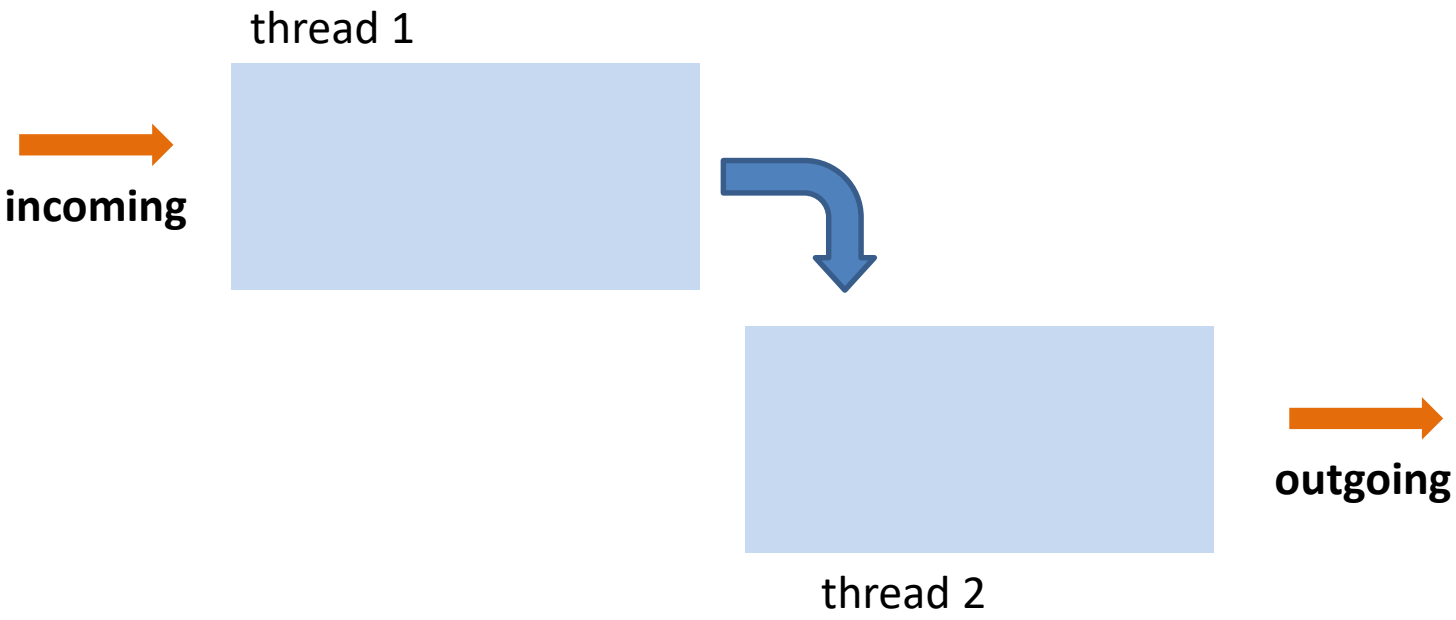


Span identification

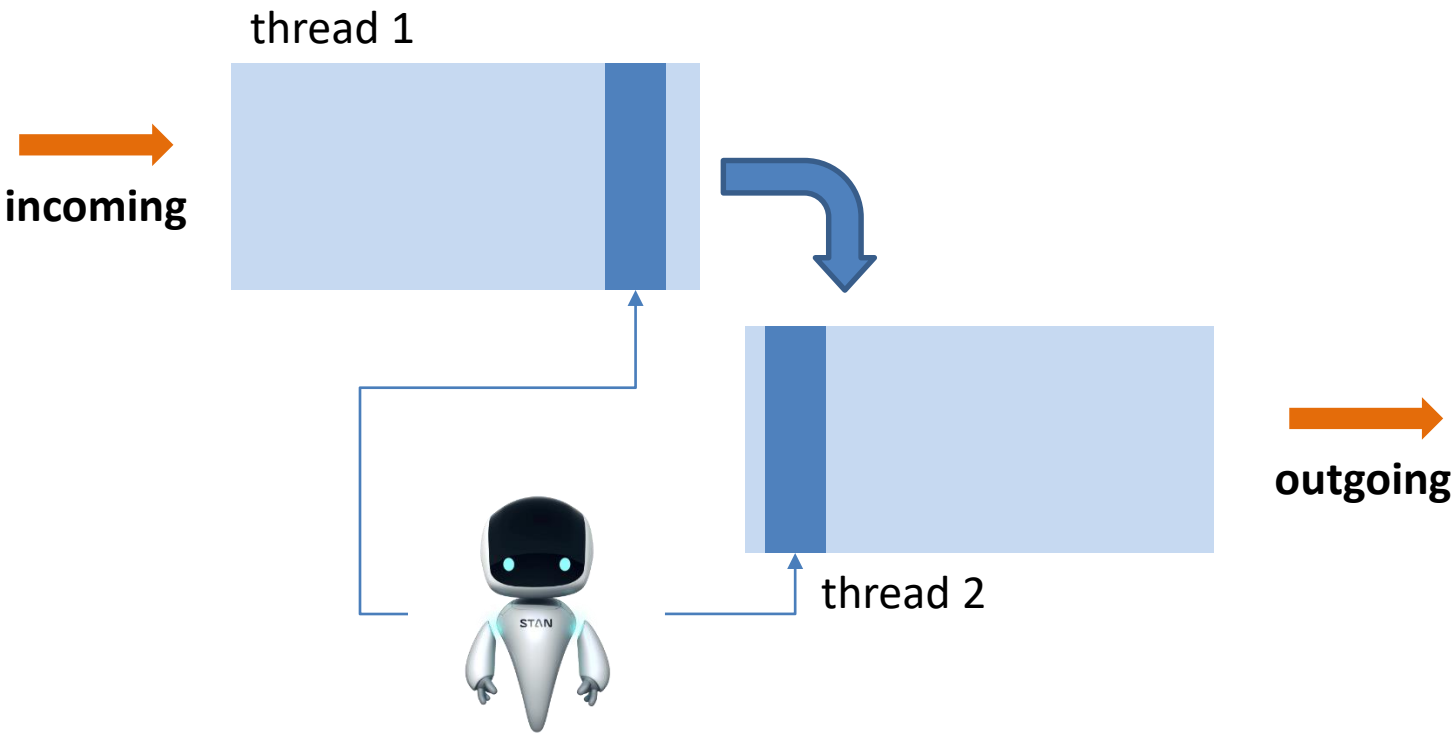
```
class MyBatchFramework {
  @com.instana.sdk.Span("trace-me")
   void doBatchJob() {
     // do hard work...
   }
}
```



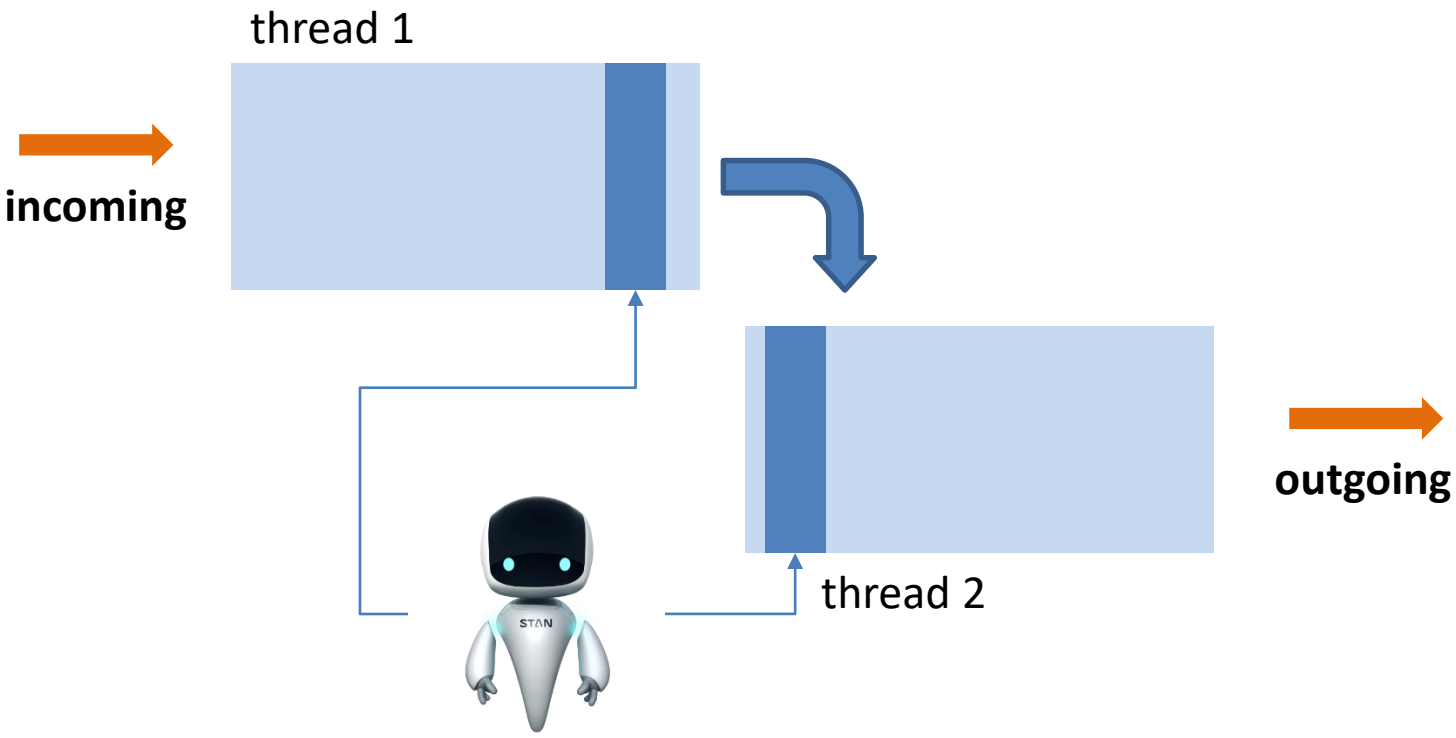
Context-switch tracing



Context-switch tracing



Context-switch tracing



OPENTRACING

Requires explicit context hand-over upon each context-switch.

Tracing sandboxed applications

```
class AccessController {

  public static void checkPermission(Permission permission)
      throws AccessControlException {

    AccessControlContext stack =
        getStackAccessControlContext();

    // perform check based on stack
}
```

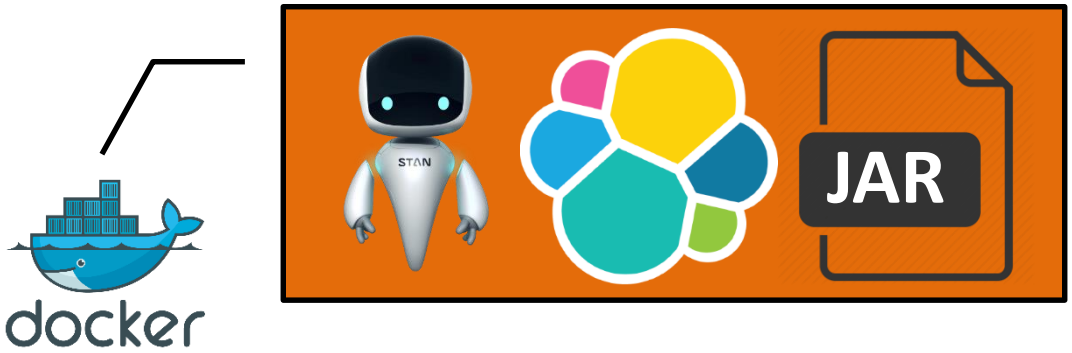
Tracing sandboxed applications

```
class AccessController {

  public static void checkPermission(Permission permission)
      throws AccessControlException {

    if (isInstanaSandboxed()) {
      return;
    }

    AccessControlContext stack =
        getStackAccessControlContext();

    // perform check based on stack privileges
}
```

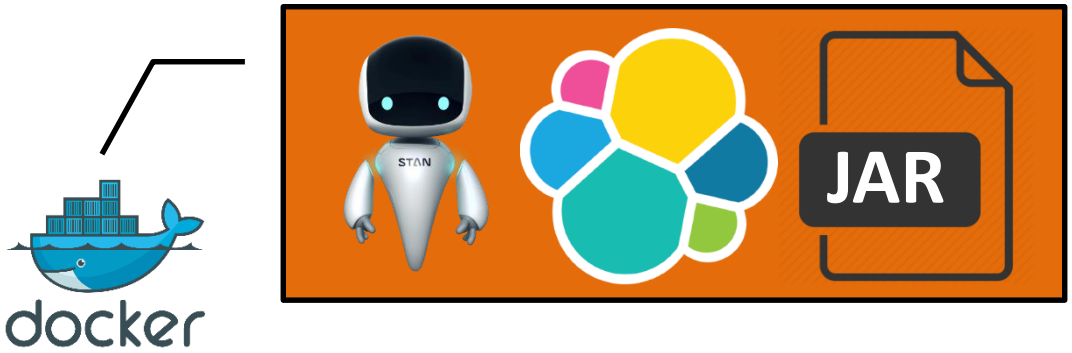
Testing instrumentation and trace collection

Testing instrumentation and trace collection

Testing instrumentation and trace collection

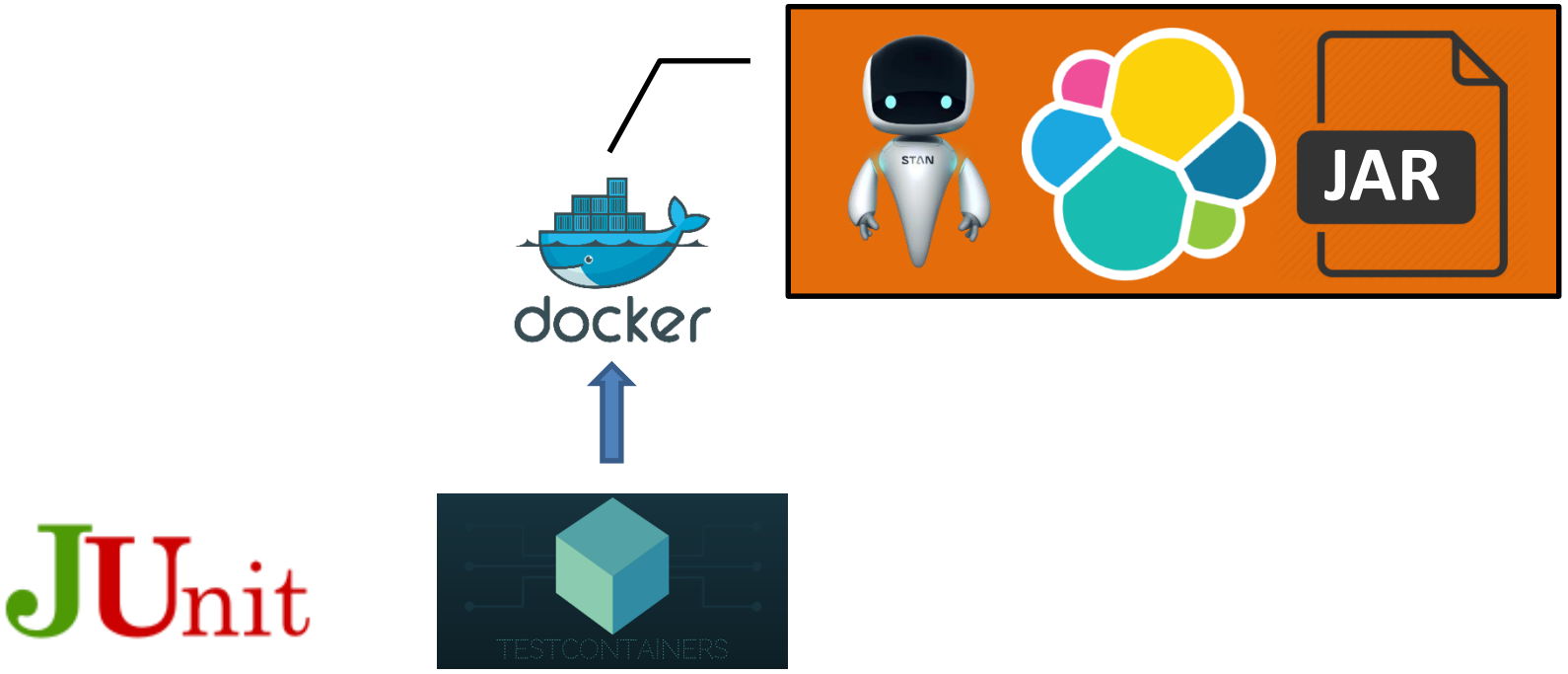


Testing instrumentation and trace collection

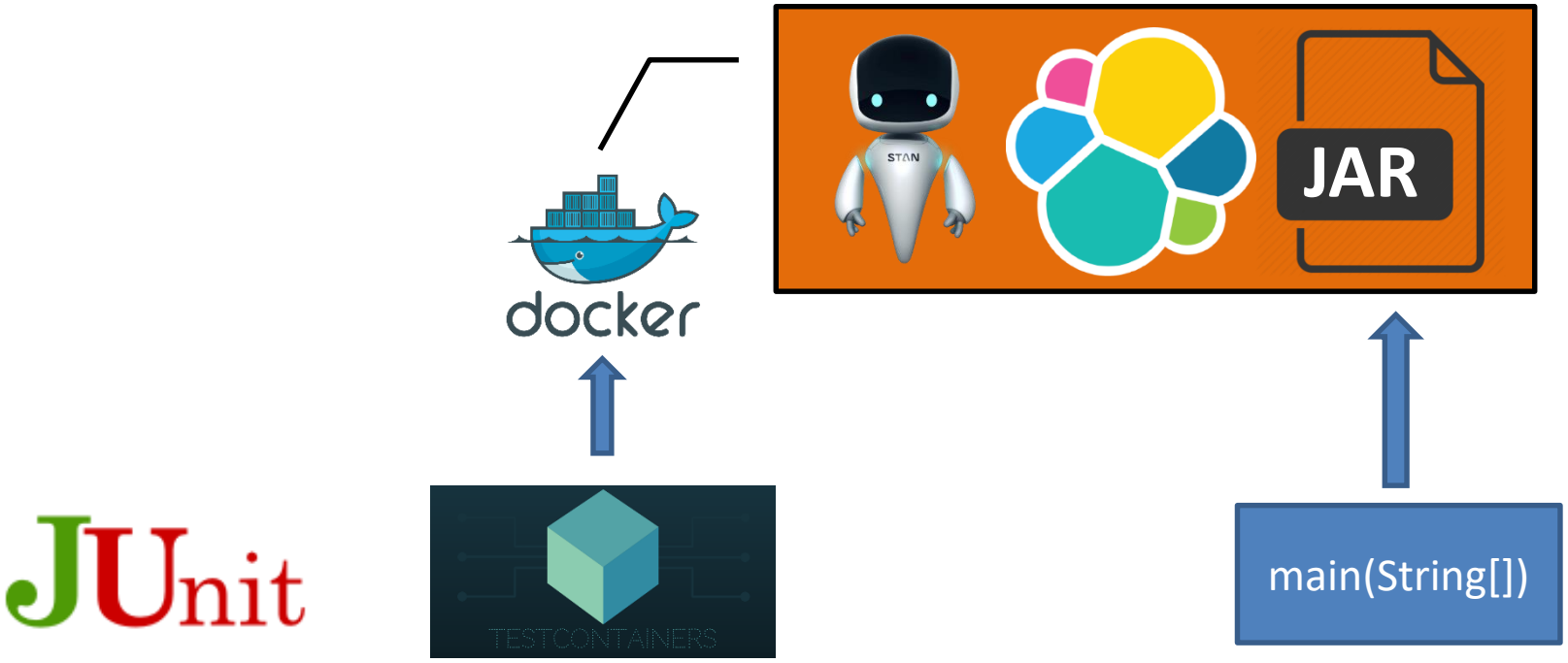


JUnit

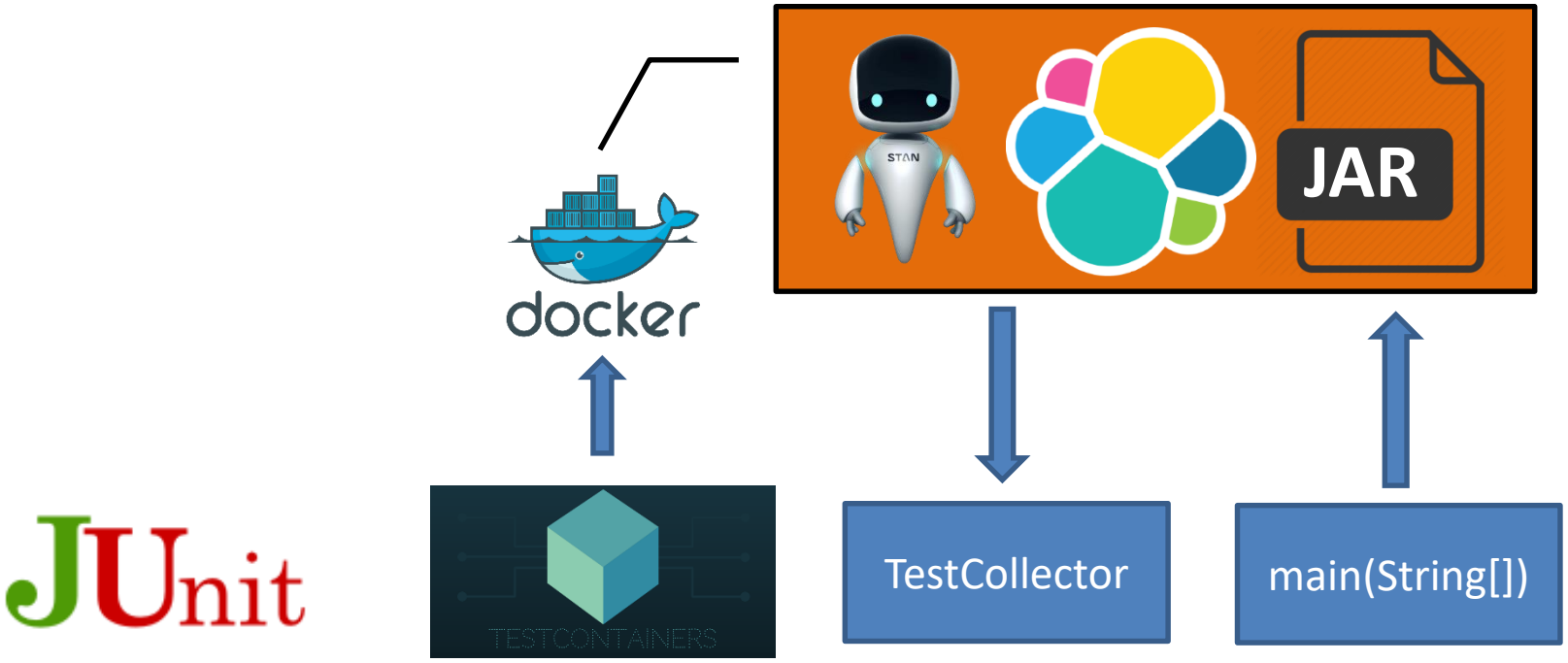
Testing instrumentation and trace collection



Testing instrumentation and trace collection



Testing instrumentation and trace collection



http://rafael.codes
@rafaelcodes

---

http://documents4j.com
https://github.com/documents4j/documents4j

---

http://bytebuddy.net
https://github.com/raphw/byte-buddy