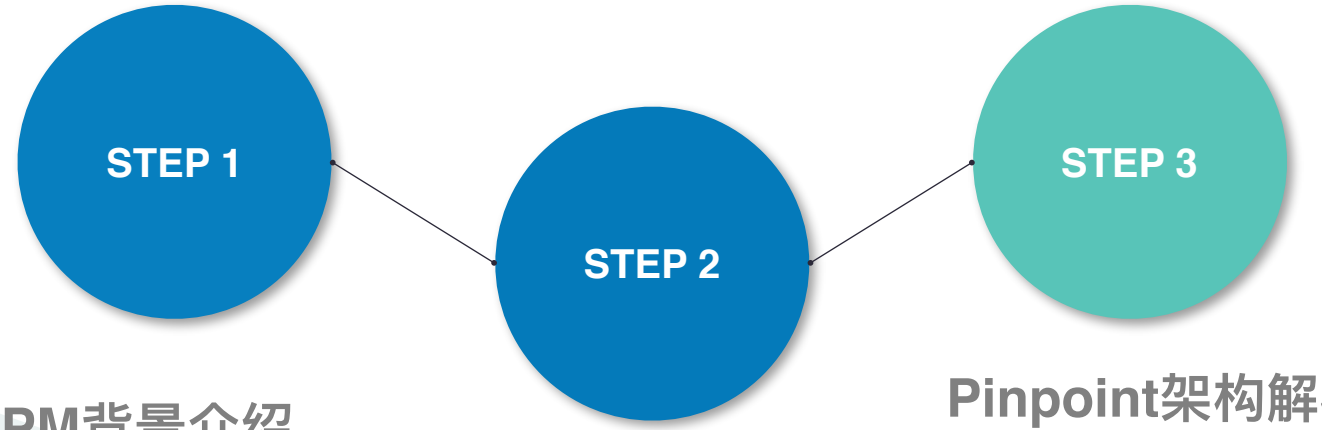
APM技术分享

分享人：刘晓东

目录 /CONTENTS

字节码增强技术分享



APM背景介绍

Pinpoint架构解析

STEP 1

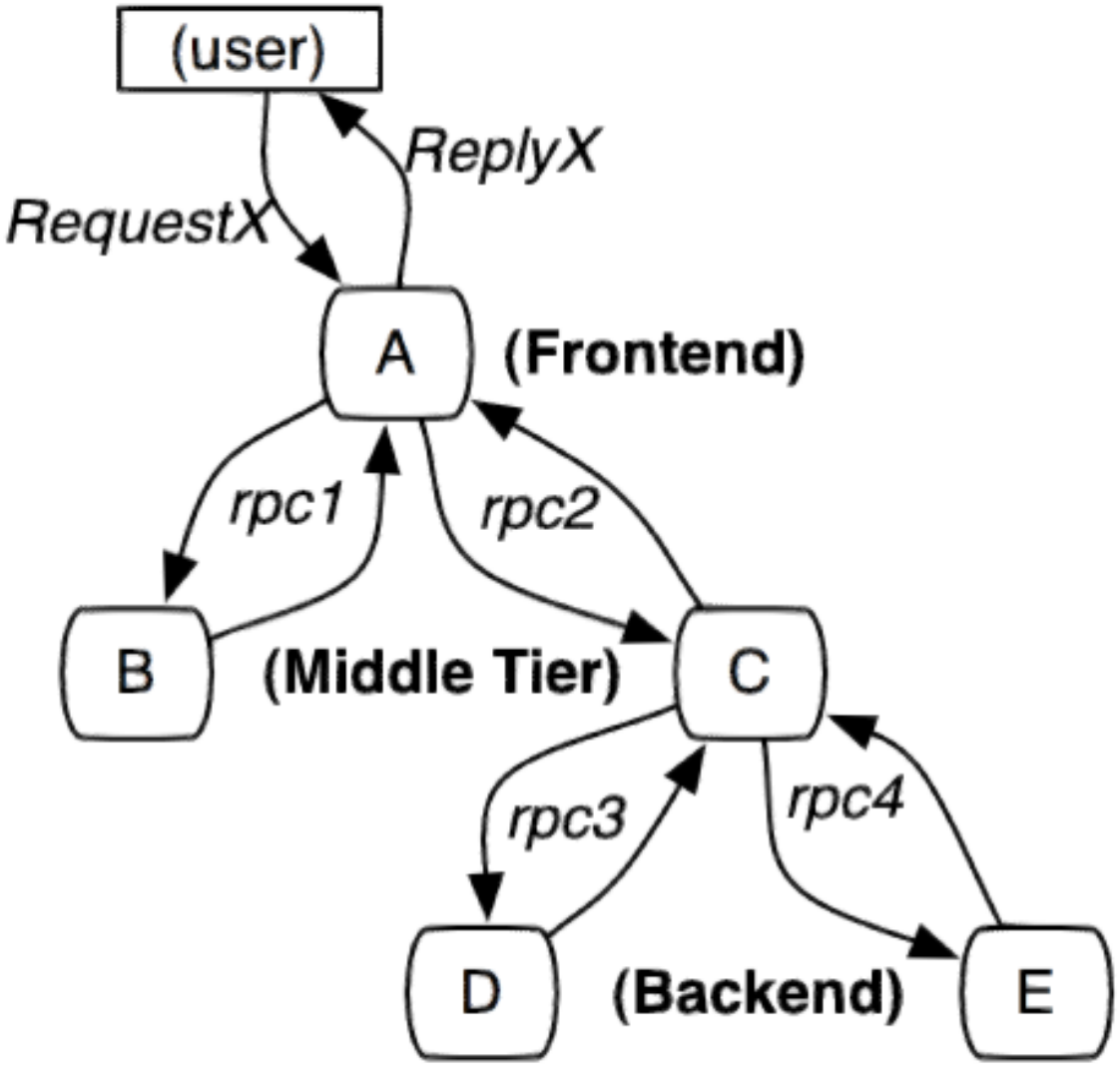
APM技术概述

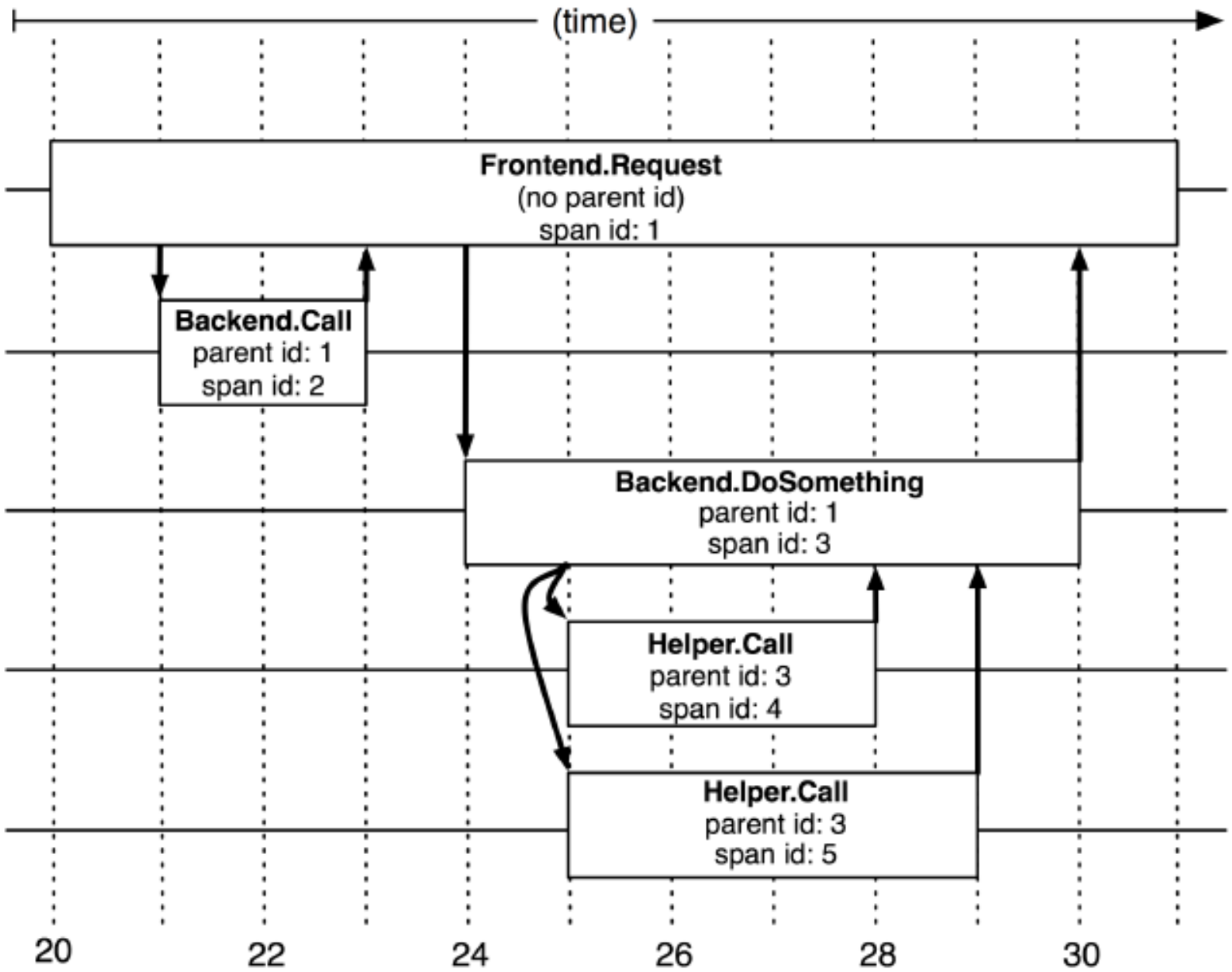
背景

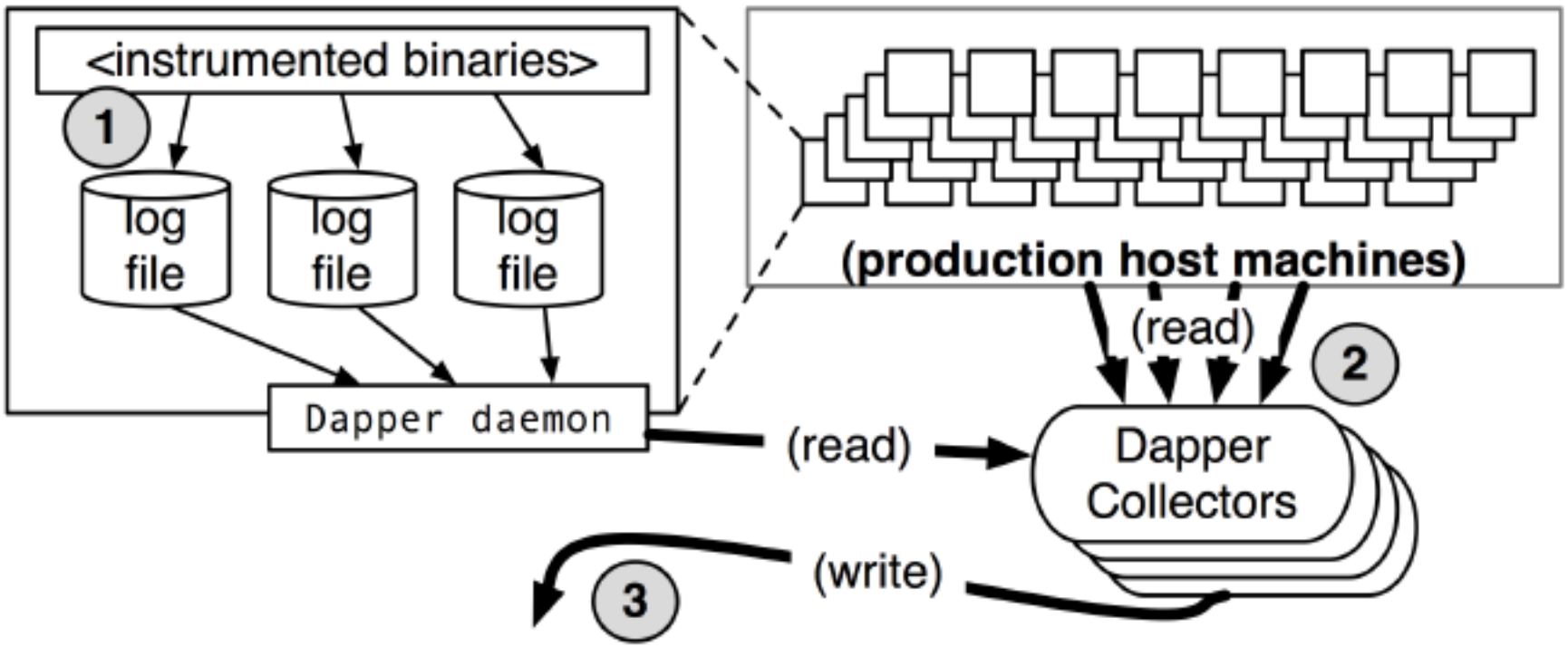
APM工具全称为：Application Performance Management，即应用性能分析工具。

当代互联网系统，特别是微服务化后，通常会包含很多个个不同的模块，模块的交互会比较复杂。为了更好的分析系统的行为、及时发现排查性能问题，APM工具逐渐发展起来。

目前主流的APM工具都是基于谷歌的Drapper论文（大规模分布系统的追踪系统）发展起来的，包括：Zipkin、Pinpoint、Skywalking以及Cat等。


(user)
ReplyX
RequestX
A
(Frontend)
rpc1
rpc2
B
(Middle Tier)
C
rpc3
rpc4
D
(Backend)
E

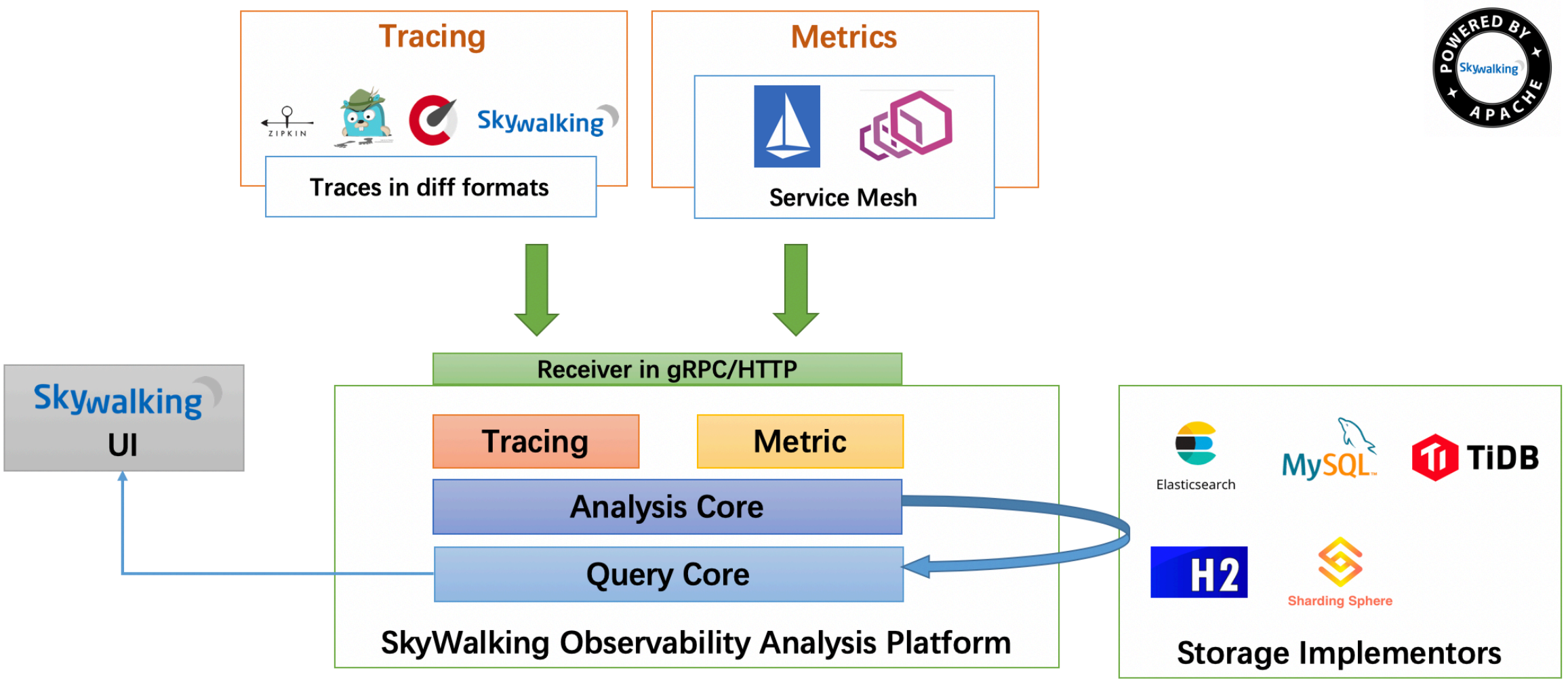

(time)
Frontend.Request
(no parent id)
span id: 1
Backend.Call
parent id: 1
span id: 2
Backend.DoSomething
parent id: 1
span id: 3
Helper.Call
parent id: 3
span id: 4
Helper.Call
parent id: 3
span id: 5
20
22
24
26
28
30



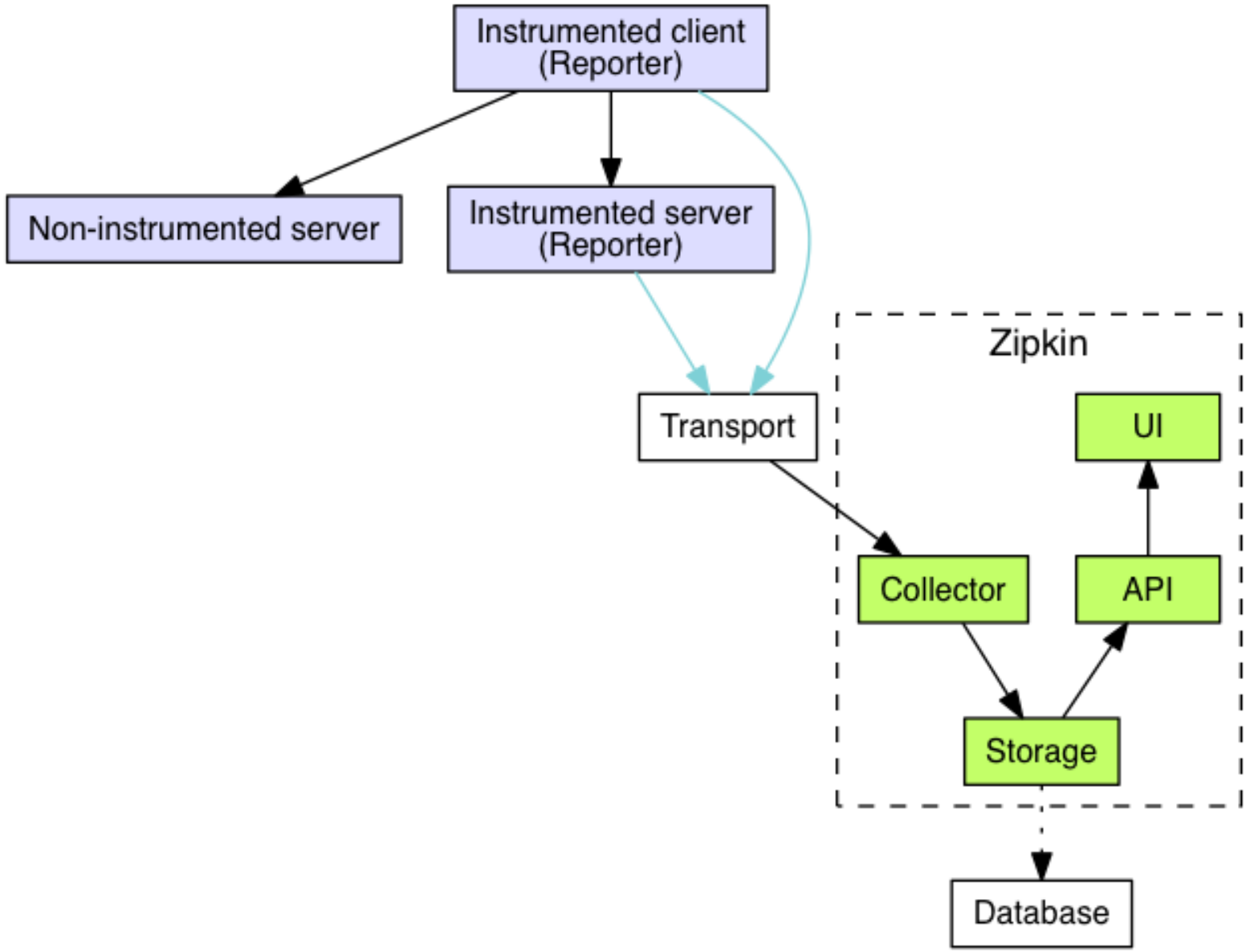
trace id	span 12	span 23	span 34	span 45	span 56	...
123456	*nil*	*nil*	**<data>**	**<data>**	*nil*	...
246802	**<data>**	*nil*	*nil*	*nil*	**<data>**	...
357913	*nil*	**<data>**	*nil*	*nil*	*nil*	...
...	...	...	...	...	...	...

(Central Bigtable repository for trace data)

框架对比-Skywalking



框架对比-Zipkin


Instrumented client
(Reporter)
Non-instrumented server
Instrumented server
(Reporter)
Zipkin
Transport
UI
Collector
API
Storage
Database

框架对比

	优点	缺点
Skywalking	Java采用字节码注入、支持ServiceMesh、插件丰富、加入Apache孵化器、性能损耗低	暂无明显缺点
Pinpoint	Java字节码注入支持、插件丰富、UI展示功能强大	比Skywalking性能损耗高，不符合OpenTracing规范
Zipkin	有twiter背书	有代码侵入性、监控不够详细
Cat	中文社区	代码侵入性过高

Trace：调用链，由多个Span组成

Span：一次方法调用或者程序间的调用。包含Span名称，起始、结束时间，SpanTag（用户自定义，用于过滤查询使用），SpanLog（包含自定义信息，用于查看排查链路使用）以及SpanContext。

SpanContext：包含当前调用链的状态：比如链路ID、SpanID；以及Baggage Items：跨进程边界传输的数据。

Carrier：真正用于传输SpanContext的载体，可以是TextMap，也可以放到HttpHeader。Trace 注入SpanContext到Carrier中，或者从Carrier中提取出来。

STEP 2

字节码增强技术

什么是字节码增强技术?

Java字节码增强指的是在字节码生成后，通过对其修改，增强其功能。相当于对应用程序的二进制文件进行修改。

字节码增强技术可以用于动态代理、日志记录以及链路追踪等许多地方。

字节码增强框架常用的有Javassist框架以及ASM框架。

Javassist是日本东京工业大学的一位教授创建的编辑、创建字节码的类库。具有操作简单、方便的优点，不用深入了解字节码的结构就可以对字节码做修改。缺点是性能相对较差。

CtClass：是Class文件的抽象映射

CtMethod：是Class中Method的抽象映射

ClassPool：是一个存储CtClass的Hash结构，key为类名，value为CtClass实例

Javassist字节码增强技术

JavassistDemo

```java
package org.example.javassist;
public class HelloAssist {
    public void sayHello(){
        System.out.println("Hello world!");
    }
}

public class JavassistDemo {
    public static void main(String[] args) throws Exception {
        ClassPool classPool = ClassPool.getDefault();
        CtClass ctClass = classPool.getCtClass("org.example.javassist.HelloAssist");
        CtMethod ctMethod = ctClass.getDeclaredMethod("sayHello");
        ctMethod.insertBefore("System.out.println(\"before---->\");");
        ctMethod.insertAfter("System.out.println(\"after---->\");");

        HelloAssist helloAssist = (HelloAssist) ctClass.toClass().newInstance();
        helloAssist.sayHello();
    }
}


```

Modifiers, name, super class, interfaces	
Constant pool: numeric, string and type constants	
Source file name (optional)	
Enclosing class reference	
Annotation*	
Attribute*	
Inner class*	Name
Field*	Modifiers, name, type
	Annotation*
	Attribute*
Method*	Modifiers, name, return and parameter types
	Annotation*
	Attribute*
	Compiled code

Figure 2.1.: Overall structure of a compiled class (* means zero or more)

Java type	Type descriptor
`boolean`	`Z`
`char`	`C`
`byte`	`B`
`short`	`S`
`int`	`I`
`float`	`F`
`long`	`J`
`double`	`D`
`Object`	`Ljava/lang/Object;`
`int[]`	`[I`
`Object[][]`	`[[Ljava/lang/Object;`

Figure 2.2.: Type descriptors of some Java types

ClassReader：解析类文件或者byte数组，当解到Field、Method等的时候，会调用ClassVisitor的visitXXX。事件生产者。

ClassVisitor：采用访问者模式，声明有visitXXX方法，可以重新visitXXX来实现自己的功能。相当于事件过滤器。

ClassWriter：继承自ClassVisitor，可以实现byte数组到类的转换。

ASM字节码增强技术-CoreAPI

```java
public class HelloAsm {
    private String name;

    public void say(){
        System.out.println(name);
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

```java
public class AsmDemoVisitor extends ClassVisitor {

    public AsmDemoVisitor(int api, ClassVisitor classVisitor)
        super(api, classVisitor);
    }

    @Override
    public MethodVisitor visitMethod(int access, String name,
        if (name.equals("say")) {
            return null;
        }
        return super.visitMethod(access, name, descriptor, sig
    }
}
```

ASM字节码增强技术-CoreAPI

```java
public class AsmCoreApiDemo {
    @Test
    public void test() throws IOException {
        ClassReader cr = new ClassReader( className: "org.example.asm.HelloAsm");
        ClassWriter cw = new ClassWriter( flags: 0);
        ClassVisitor cv = new AsmDemoVisitor(Opcodes.ASM4, cw);
        cr.accept(cv, parsingOptions: 0);
        byte[] toByte = cw.toByteArray();
        File file = new File( pathname: "HelloAsm.class");
        FileOutputStream fout = new FileOutputStream(file);
        fout.write(toByte);
        fout.close();
    }
}
```

ASM字节码增强技术-TreeAPI

```java
/**
 * @author 刘晓东
 */
public class AsmTreeApiDemo {
    @Test
    public void test() throws IOException {
        ClassReader cr = new ClassReader( className: "org.example.asm.HelloAsm");

        ClassNode cn = new ClassNode();
        cr.accept(cn, parsingOptions: 0);
        cn.methods.removeIf(methodNode -> "say".equals(methodNode.name));

        ClassWriter cw = new ClassWriter( flags: 0);
        cn.accept(cw);
        byte[] toByte = cw.toByteArray();
        File file = new File( pathname: "HelloAsm.class");
        FileOutputStream fout = new FileOutputStream(file);
        fout.write(toByte);
        fout.close();
    }
}
```

JavaAgent

```java
//org.example.App
public class App {
    public static void main( String[] args ) {
        System.out.println( "Hello World!" );
    }
}

//maven demo
<plugin>
    <artifactId>maven-jar-plugin</artifactId>
    <version>3.0.2</version>
    <configuration>
        <archive>
            <manifest>
                <addClasspath>true</addClasspath>
                <classpathPrefix>lib/</classpathPrefix>
                <mainClass>org.example.App</mainClass>
            </manifest>
        </archive>
    </configuration>
</plugin>
```

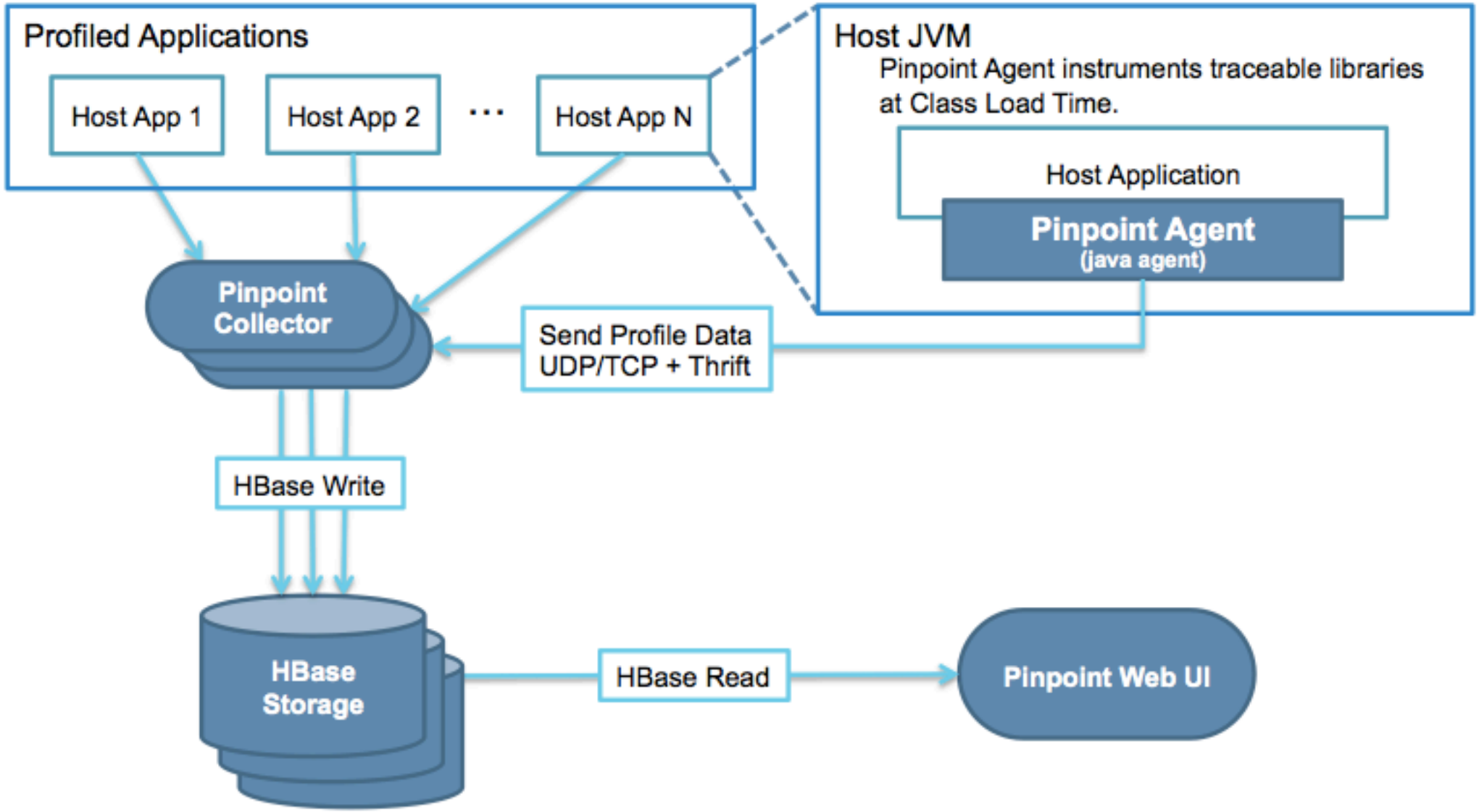
```java
//agent demo
public class PreApp
{
  public static void premain(String agentArgs, Instrumentation instrumentation){
    instrumentation.addTransformer(new ClassFileTransformer() {
      @Override
      public byte[] transform(ClassLoader loader, String className, Class<?> classBeingRedefined,
ProtectionDomain protectionDomain, byte[] classfileBuffer) throws IllegalClassFormatException {
        //此处需要指定类名，不然会打印多次before main。同时类名中的 . 需要替换为 /
        if("org/example/App".equals(className)){
          System.out.println("before main--->");
        }
        return classfileBuffer;
      }
    });
  }
}

//maven demo
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-jar-plugin</artifactId>
    <version>3.0.2</version>
    <configuration>
    <archive>
        <manifest>
        <addClasspath>true</addClasspath>
        </manifest>
        <manifestEntries>
        <Premain-Class>
            org.example.PreApp
        </Premain-Class>
        </manifestEntries>
    </archive>
    </configuration>
</plugin>
```

```
java -javaagent:agent-demo-1.0-SNAPSHOT.jar  -jar demo-1.0-SNAPSHOT.jar
```

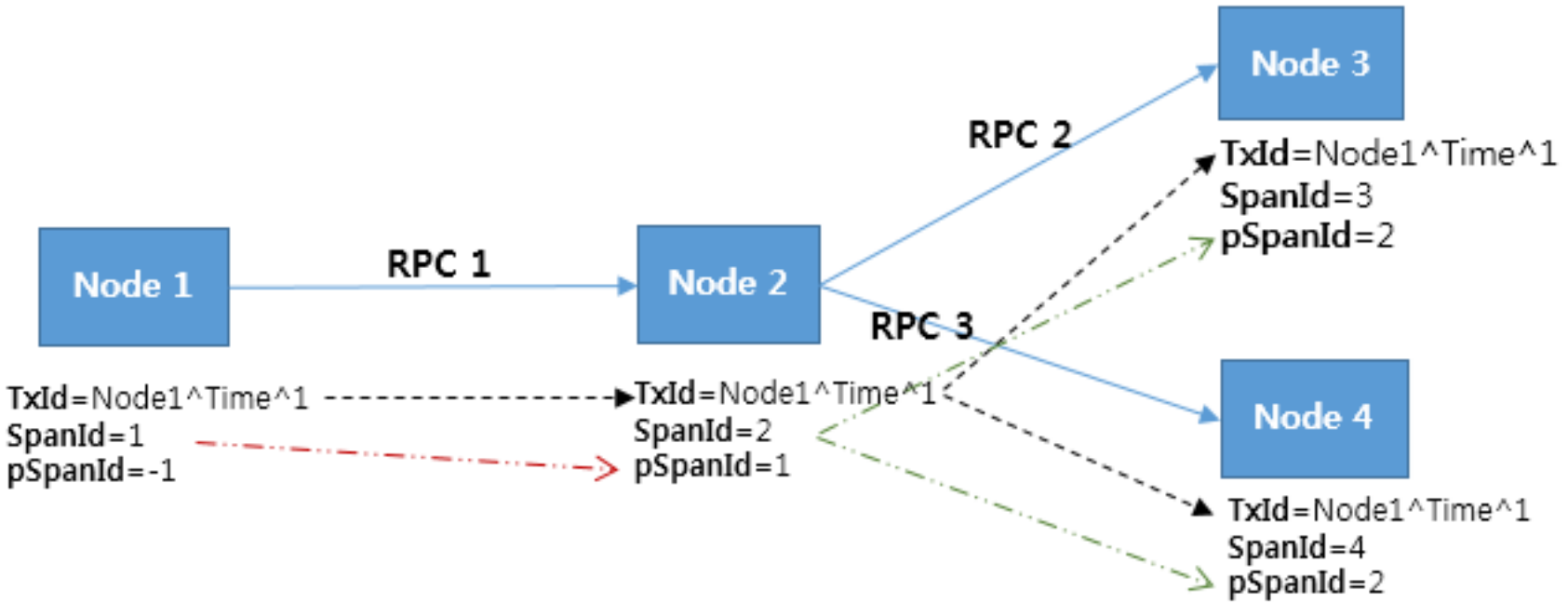
STEP 3

Pinpoint架构分析

Pinpoint架构

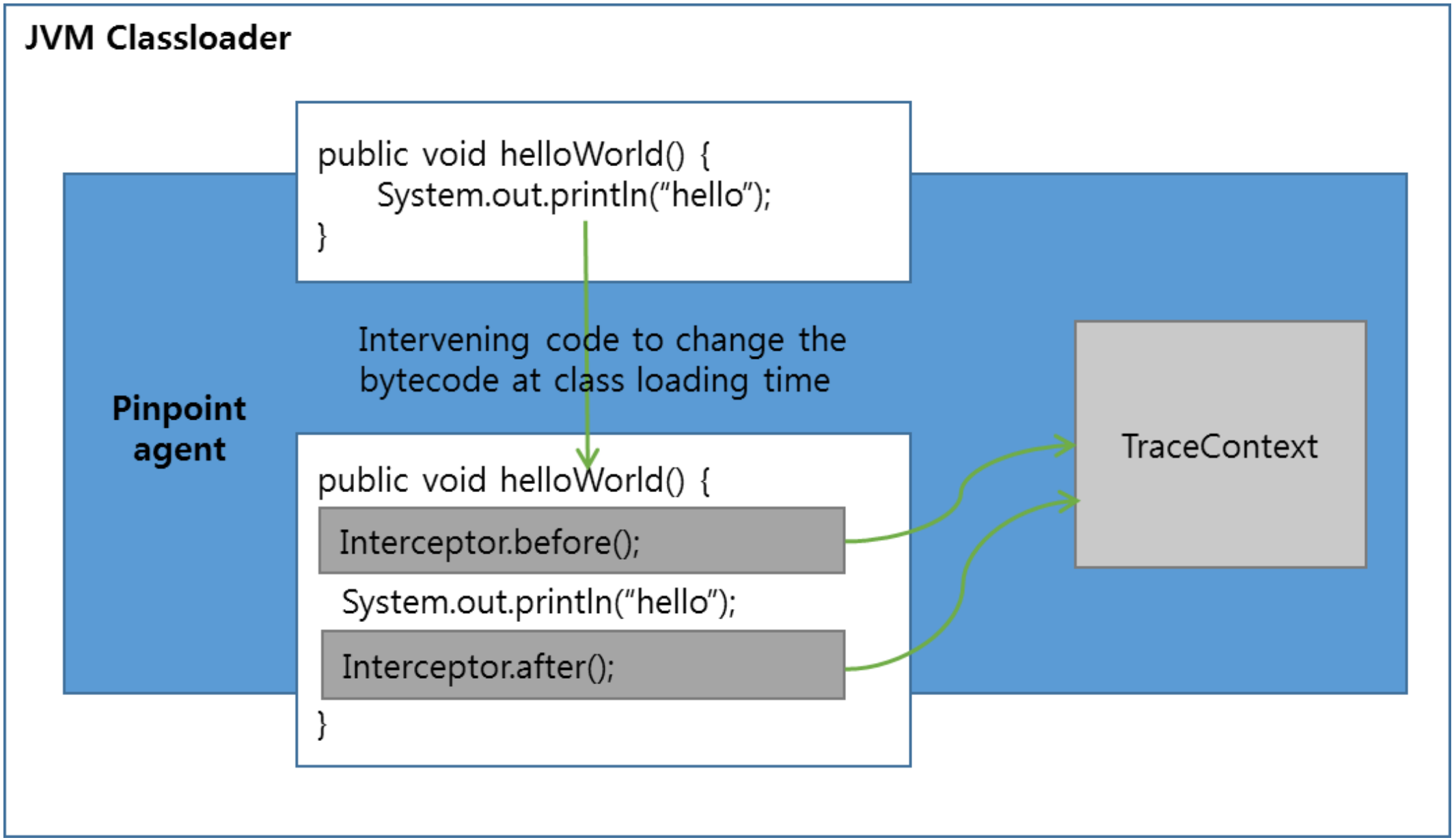

Profiled Applications
Host App 1
Host App 2
Host App N
Host JVM
Pinpoint Agent instruments traceable libraries at Class Load Time.
Host Application
Pinpoint Agent
(java agent)
Pinpoint Collector
Send Profile Data
UDP/TCP + Thrift
HBase Write
HBase Storage
HBase Read
Pinpoint Web UI

Pinpoint追踪链路



注：下一个Span的ID由上一个Span生成

Pinpoint代码增强



启动过程

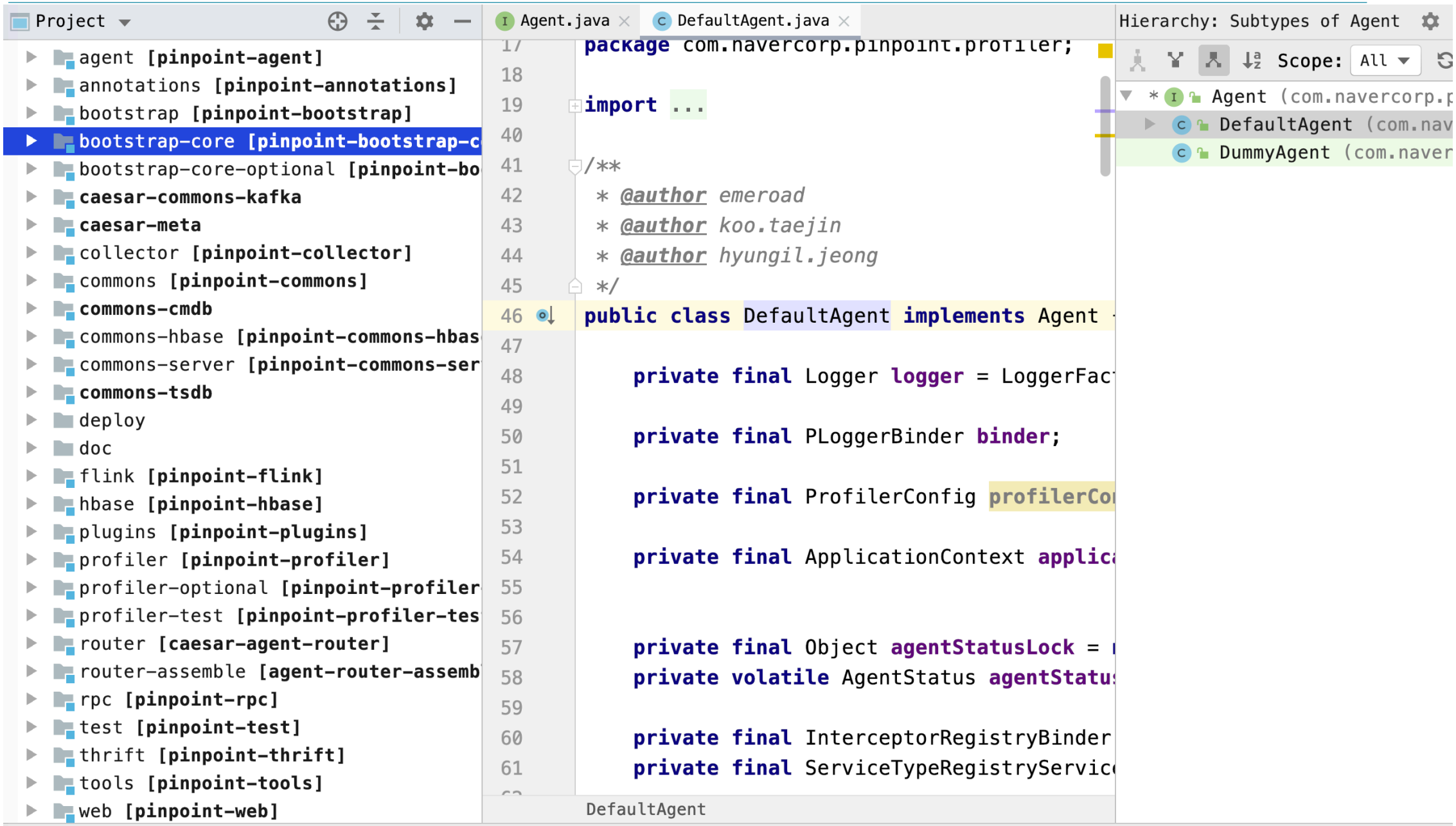
启动时：

1. 通过javaagent和应用程序一起启动
2. 加载plugin目录下的所有插件
3. 初始化插件并注册相关transformCallBack
4. 加载到拦截的类时，执行增强方法，注入增强代码

启动后：

1. 执行到拦截的方法时，执行注入的before以及after逻辑
2. 记录追踪数据并进行上报

Pinpoint目录结构



Pinpoint插件分析

1. TraceMetadataProvider
   com.navercorp.pinpoint.plugin.gson.GsonMetadataProvider
2. ProfilerPlugin
   com.navercorp.pinpoint.plugin.gson.GsonPlugin

```
public class GsonMetadataProvider implements TraceMetadataProvider {
    /**
     * @see TraceMetadataProvider#setup(TraceMetadataSetupContext)
     */
    @Override
    public void setup(TraceMetadataSetupContext context) {
        context.addServiceType(GsonPlugin.GSON_SERVICE_TYPE);
        context.addAnnotationKey(GsonPlugin.GSON_ANNOTATION_KEY_JSON_LENGTH);
    }
}
```

Pinpoint插件分析

```
@Override
public void setup(ProfilerPluginSetupContext context) {
    GsonConfig config = new GsonConfig(context.getConfig());
    logger.debug("[Gson] Initialized config={}", config);

    if (config.isProfile()) {
        transformTemplate.transform( className: "com.google.gson.Gson", (instrumentor, loader, className, classBeingRedefined, protectionDomain, clas
                InstrumentClass target = instrumentor.getInstrumentClass(loader, className, classfileBuffer);

                for (InstrumentMethod m : target.getDeclaredMethods(MethodFilters.name( ...names: "fromJson"))) {
                    m.addScopedInterceptor( interceptorClassName: "com.navercorp.pinpoint.plugin.gson.interceptor.FromJsonInterceptor", GSON_SCOPE);
                }

                for (InstrumentMethod m : target.getDeclaredMethods(MethodFilters.name( ...names: "toJson"))) {
                    m.addScopedInterceptor( interceptorClassName: "com.navercorp.pinpoint.plugin.gson.interceptor.ToJsonInterceptor", GSON_SCOPE);
                }

                return target.toBytecode();
        });
    }
}
```

Pinpoint插件分析

```java
@Override
public void before(Object target, Object arg0, Object arg1) {
    if (logger.isDebugEnabled()) {
        logger.beforeInterceptor(target, new Object[] {arg0, arg1});
    }
    final Trace trace = traceContext.currentTraceObject();
    if (trace == null) {
        return;
    }
    trace.traceBlockBegin();
}
@Override
public void after(Object target, Object arg0, Object arg1, Object result, Throwable throwable) {
    if (logger.isDebugEnabled()) {
        logger.afterInterceptor(target, new Object[] {arg0, arg1}, result, throwable);
    }
    final Trace trace = traceContext.currentTraceObject();
    if (trace == null) {
        return;
    }
    try {
        SpanEventRecorder recorder = trace.currentSpanEventRecorder();
        recorder.recordServiceType(GsonPlugin.GSON_SERVICE_TYPE);
        recorder.recordApi(descriptor);
        recorder.recordException(throwable);

        if (arg0 != null && arg0 instanceof String) {
            recorder.recordAttribute(GsonPlugin.GSON_ANNOTATION_KEY_JSON_LENGTH, ((String) arg0).length());
        }
    } finally {
        trace.traceBlockEnd();
    }
}
```

Pinpoint数据收集

DataSender.java

```
1  /.../
16
17 package com.navercorp.pinpoint.profiler.sender;
18
19 import org.apache.thrift.TBase;
20
21 /**
22  * @author emeroad
23  * @author netspider
24  */
25 public interface DataSender {
26
27     boolean send(TBase<?, ?> data);
28
29     void stop();
30
31 }
32
```

Hierarchy: Subtypes of DataSender

Scope: All

- DataSender (com.navercorp.pinpoint.profiler.sender)
  - NioUDPDataSender (com.navercorp.pinpoint.profiler.sender)
  - ListenableDataSender (com.navercorp.pinpoint.test)
  - UdpDataSender (com.navercorp.pinpoint.profiler.sender)
    - BufferedUdpDataSender (com.navercorp.pinpoint.profiler.sender)
      - Anonymous in sendMessage_getLimit() in UdpDataSenderTest (com.
  - AbstractDataSender (com.navercorp.pinpoint.profiler.sender)
    - NioUDPDataSender (com.navercorp.pinpoint.profiler.sender)
    - TcpDataSender (com.navercorp.pinpoint.profiler.sender)
    - UdpDataSender (com.navercorp.pinpoint.profiler.sender)
      - BufferedUdpDataSender (com.navercorp.pinpoint.profiler.sende
        - Anonymous in sendMessage_getLimit() in UdpDataSenderTest (cc
    - SpanStreamUdpSender (com.navercorp.pinpoint.profiler.sender)
  - EnhancedDataSender (com.navercorp.pinpoint.profiler.sender)
    - LoggingDataSender (com.navercorp.pinpoint.profiler.sender)
    - TcpDataSender (com.navercorp.pinpoint.profiler.sender)
    - TestTcpDataSender (com.navercorp.pinpoint.test)
    - CountingDataSender (com.navercorp.pinpoint.profiler.sender)
    - EmptyDataSender (com.navercorp.pinpoint.profiler.sender)

ThreadLocal

```
public T get() {
    Thread t = Thread.currentThread();
    ThreadLocalMap map = getMap(t);
    if (map != null) {
        ThreadLocalMap.Entry e = map.getEntry(this);
        if (e != null) {
            @SuppressWarnings("unchecked")
            T result = (T)e.value;
            return result;
        }
    }
    return setInitialValue();
}

private T setInitialValue() {
    T value = initialValue();
    Thread t = Thread.currentThread();
    ThreadLocalMap map = getMap(t);
    if (map != null)
        map.set(this, value);
    else
        createMap(t, value);
    return value;
}
ThreadLocalMap getMap(Thread t) {
    return t.threadLocals;
}
void createMap(Thread t, T firstValue) {
    t.threadLocals = new ThreadLocalMap(this, firstValue);
}
```

```
public void set(T value) {
    Thread t = Thread.currentThread();
    ThreadLocalMap map = getMap(t);
    if (map != null)
        map.set(this, value);
    else
        createMap(t, value);
}


 public void remove() {
     ThreadLocalMap m = getMap(Thread.currentThread());
     if (m != null)
         m.remove(this);
 }
```

感谢观看