Payments to grow your worldrld

Virtual threads

From observability to production readiness

WORLDLINE

François JOUBAUD

Lead developer, ex manager

- Focus on Java, observability
- Production incident analysis
- Jeyzer solution author

Get in touch:

✉ francois.joubaud@worldline.com
francois.joubaud@jeyzer.org

in linkedin.com/in/francois-joubaud

Jeyzer – what is it ?

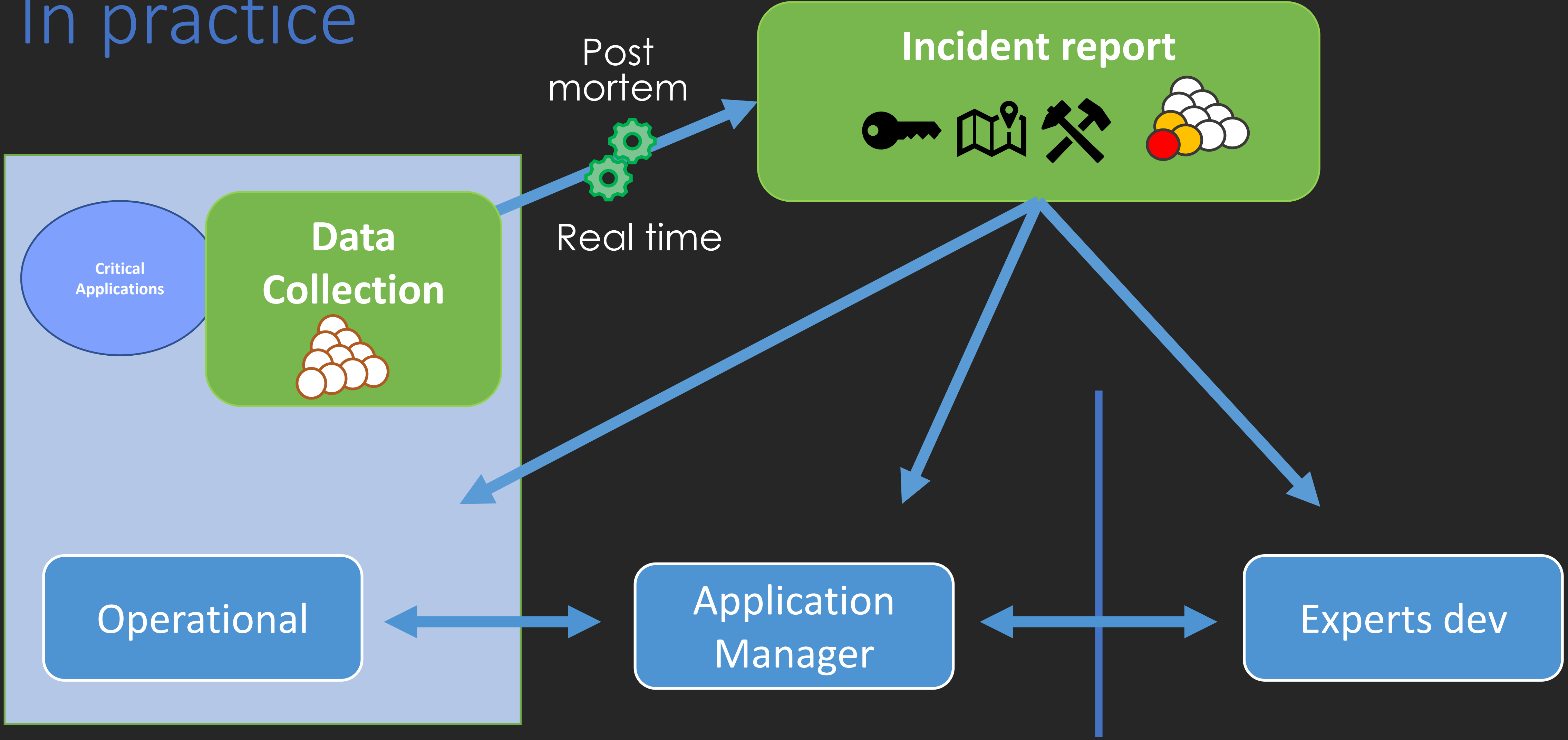
- Incident analysis solution for Java servers
  - Open source – 10Y maturity – 12th release
  - Users : from OPS to R&D

- Sampling based (JFR, JZR, dumps..)

- Outputs : JZR report (Excel) and event dispatching (Mail, JIRA, Zabbix/Grafana..)

- Goals : provide an accurate view of the Java internals
  - Abstract the technical information
  - Filter the real activity
  - Detect problems

In practice


Post mortem
Real time
Incident report
Critical Applications
Data Collection
Operational
Application Manager
Experts dev

Jeyzer – current state

- Jeyzer 3.1 : Virtual thread analysis
- Jeyzer 3.2 : Virtual thread recording agent
- Jeyzer 3.3 : Zabbix/Grafana integration
- The bible : Virtual thread monitoring guide
- Conferences – Paris JUG, Bordeaux JUG, JChateau

Takeaway - key points

Virtual threads -> IO wait

- Simple usage
  Large adoption
- Unleash the beast
  Thread waves & impacts
- Paradigms to come
  StructuredTaskScope

Monitoring goals

- Simple monitoring
  Standard availability
- Control the beast
- Activity flow insights
  Distributed…

Monitoring situation

production usage

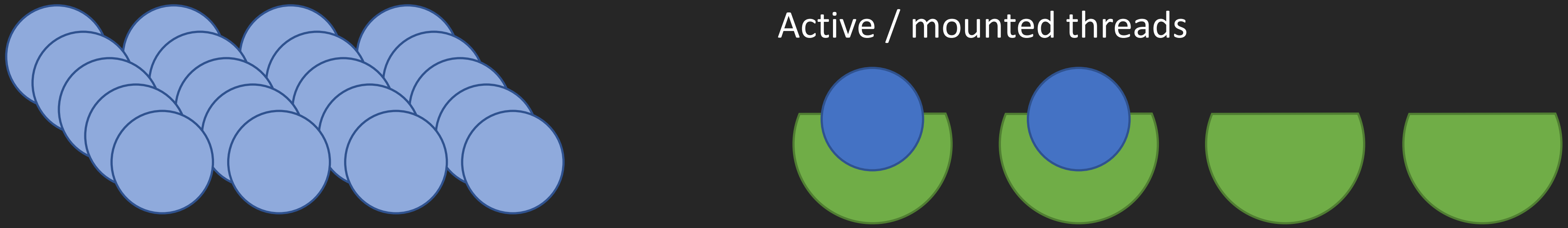
?

Introduction

- Agenda
    - Technical overview
    - From debugging to observability
    - Analysis in Jeyzer : demo cases
    - Adoption in dev and production
    - Conclusion

Technical context

- Use ExecutorService, VirtualThread or StructuredTaskScope to create it
- On calls that imply a wait, virtual threads get **suspended**
- A **carrier** thread executes the **active** virtual threads
- Sleep and IO classes have been enhanced to suspend virtual threads



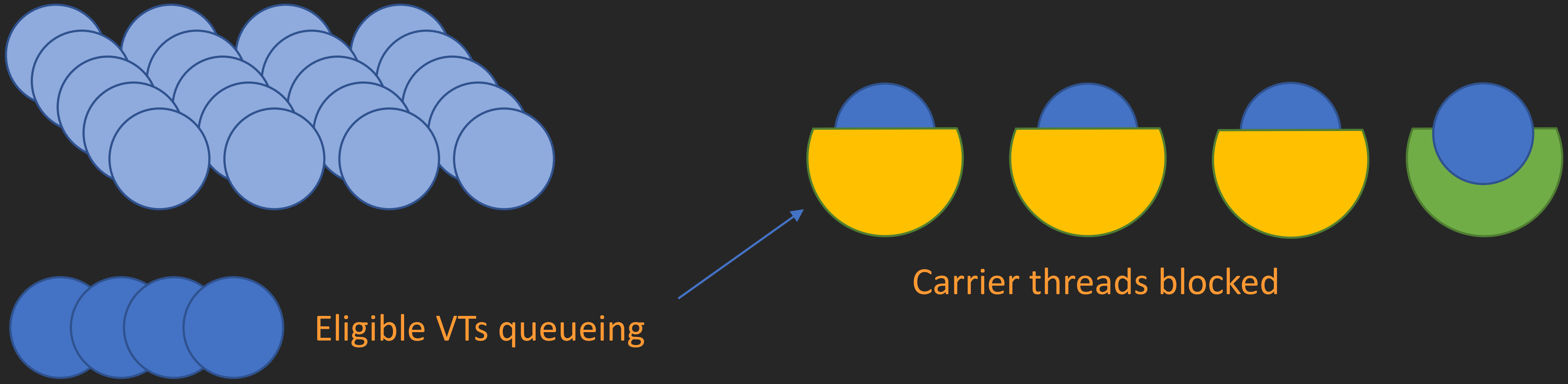
Active / mounted threads

Thousands of suspended VTs

Carrier threads = native threads

Technical context - Pinning

- Pinning operations : locking and native operations
- Pinning virtual threads = active threads blocked = less carrier threads !
- Solution : ReentrantLock usage – virtual thread based



Eligible VTs queueing

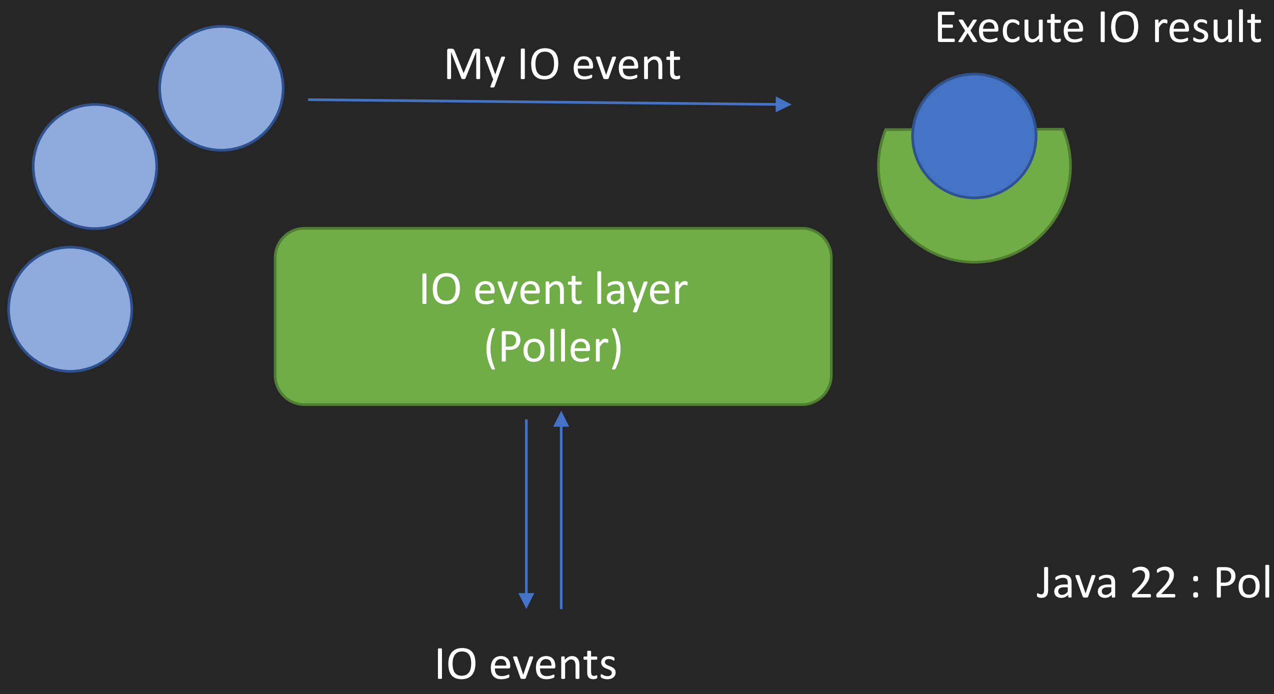
Carrier threads blocked

Technical context - Advanced

- Number of Carrier threads = number of CPUs

  Override : `-Djdk.virtualThreadScheduler.parallelism / maxPoolSize`

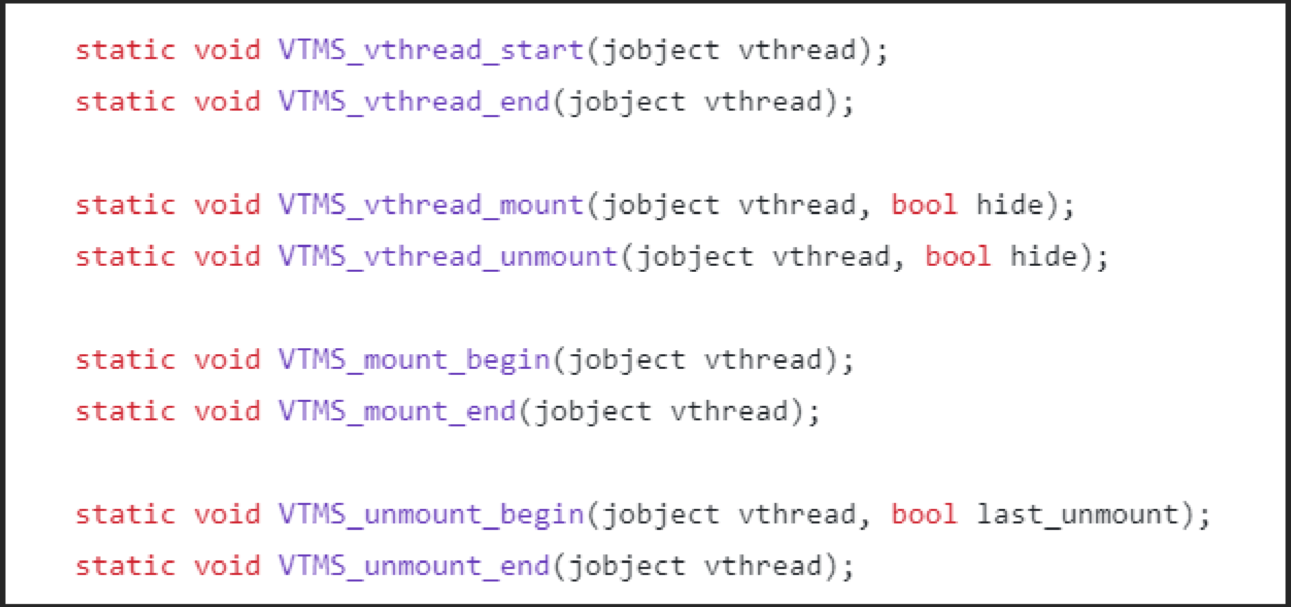
- When dealing with IO, the JVM will start a Poller layer to dispatch the IO events



Java 22 : Poller = virtual threads on Linux

Debugging

- Debugging is NOT :
  - Observability
  - Production compliant



```
static void VTMS_vthread_start(jobject vthread);
static void VTMS_vthread_end(jobject vthread);

static void VTMS_vthread_mount(jobject vthread, bool hide);
static void VTMS_vthread_unmount(jobject vthread, bool hide);

static void VTMS_mount_begin(jobject vthread);
static void VTMS_mount_end(jobject vthread);

static void VTMS_unmount_begin(jobject vthread, bool last_unmount);
static void VTMS_unmount_end(jobject vthread);
```

- Debugging methods : doc
  - JDI / jdb : ThreadReference isVirtual(), no mounted state, events
  - JDWP / remote debug agent : same + all virtual threads option, slow
  - JVMTI : native calls, same + thread CPU time
  - MX diagnostic : HotSpotDiagnosticMXBean.dumpThreads

- Monitoring agents rely on debugging APIs. VT support will pop up !

Observability - Virtual and… invisible

- Virtual threads are **not** visible through :
  - Standard monitoring API : MX runtime*, Thread..
  - Standard tools : Java Flight Recorder, Jstack

- Carrier threads are visible because native
  - Their CPU usage and active allocation give an idea of the virtual activity

- JFR events introduced : insufficient
  - Creation and termination of virtual threads. Little stats interest.
  - Pinning events : interesting to see if virtual threads get incorrectly used

Observability – The JCMD candle

- Virtual threads are visible with Jcmd
  - **Thread dump command line**
  - Available in JDK only
  - Connects locally through VM attach procedure
  - Limited : only stacks, no lock and state information
  - File output - 2 formats
    - JSON : incremental collection, slower, bigger, thread groups and pools
    - TXT : real dump (safe point), stop the world. Exhaustive.

```
> Jcmd <pid> Thread.dump_to_file -format=<json|txt> <dump file path>
```

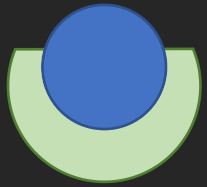
Observability – JCMD dump 1/3

- Carrier thread **active** :

```
#157 "ForkJoinPool-1-worker-10"
java.base/jdk.internal.vm.Continuation.run(Continuation.java:260)
java.base/java.lang.VirtualThread.runContinuation(VirtualThread.java:216)
java.base/java.util.concurrent.ForkJoinTask$RunnableExecuteAction.exec(ForkJoinTask.java:1423)
java.base/java.util.concurrent.ForkJoinTask.doExec(ForkJoinTask.java:387)
java.base/java.util.concurrent.ForkJoinPool$WorkQueue.topLevelExec(ForkJoinPool.java:1312)
java.base/java.util.concurrent.ForkJoinPool.scan(ForkJoinPool.java:1843)
java.base/java.util.concurrent.ForkJoinPool.runWorker(ForkJoinPool.java:1808)
java.base/java.util.concurrent.ForkJoinWorkerThread.run(ForkJoinWorkerThread.java:188)
```

Observability – JCMD dump   2/3

- Virtual thread **active** :



```
#31 "" virtual
java.base/java.util.Random.next(Random.java:444)
java.base/java.util.Random.nextDouble(Random.java:698)
java.base/java.lang.Math.random(Math.java:893)
ca.bazlur.loom.lab2.MillionVirtualThreads.consumeCPU(MillionVirtualThreads.java:44)
ca.bazlur.loom.lab2.MillionVirtualThreads.sleep(MillionVirtualThreads.java:66)
ca.bazlur.loom.lab2.MillionVirtualThreads.lambda$0(MillionVirtualThreads.java:17)
java.base/java.util.concurrent.ThreadPerTaskExecutor$ThreadBoundFuture.run(ThreadPerTaskExecutor.java:352)
java.base/java.lang.VirtualThread.run(VirtualThread.java:305)
java.base/java.lang.VirtualThread$VThreadContinuation.lambda$new$0(VirtualThread.java:177)
java.base/jdk.internal.vm.Continuation.enter0(Continuation.java:327)
java.base/jdk.internal.vm.Continuation.enter(Continuation.java:320)
```

Observability – JCMD dump   3/3

- Virtual thread **inactive (could have thousands/millions)** :

```
#52 "" virtual      java.base/jdk.internal.vm.Continuation.yield(Continuation.java:357)
java.base/java.lang.VirtualThread.yieldContinuation(VirtualThread.java:428)
java.base/java.lang.VirtualThread.parkNanos(VirtualThread.java:599)
java.base/java.lang.VirtualThread.doSleepNanos(VirtualThread.java:777)
java.base/java.lang.VirtualThread.sleepNanos(VirtualThread.java:750)
java.base/java.lang.Thread.sleep(Thread.java:525)
java.base/java.util.concurrent.TimeUnit.sleep(TimeUnit.java:446)
ca.bazlur.loom.lab2.MillionVirtualThreads.sleep(MillionVirtualThreads.java:65)
ca.bazlur.loom.lab2.MillionVirtualThreads.lambda$0(MillionVirtualThreads.java:17)
java.base/java.util.concurrent.ThreadPerTaskExecutor$ThreadBoundFuture.run(ThreadPerTaskExecutor.java:352)
java.base/java.lang.VirtualThread.run(VirtualThread.java:305)
java.base/java.lang.VirtualThread$VThreadContinuation.lambda$new$0(VirtualThread.java:177)
java.base/jdk.internal.vm.Continuation.enter0(Continuation.java:327)
java.base/jdk.internal.vm.Continuation.enter(Continuation.java:320)
```

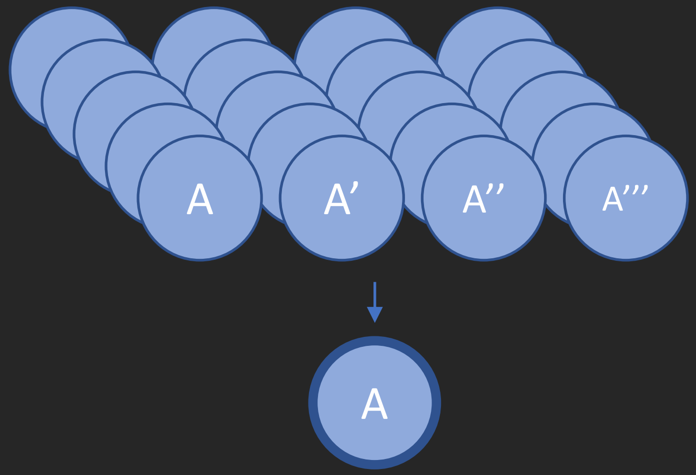
Jeyzer VT support - Recording

- JCMD periodic calls
  - 30 sec for ex
  - Windows script available in Jeyzer 3.1

- Jeyzer Recorder Agent
  - HotSpotDiagnosticMXBean.dumpThreads
  - + process details, recording rolling/archiving/retention
  - Since Jeyzer 3.2

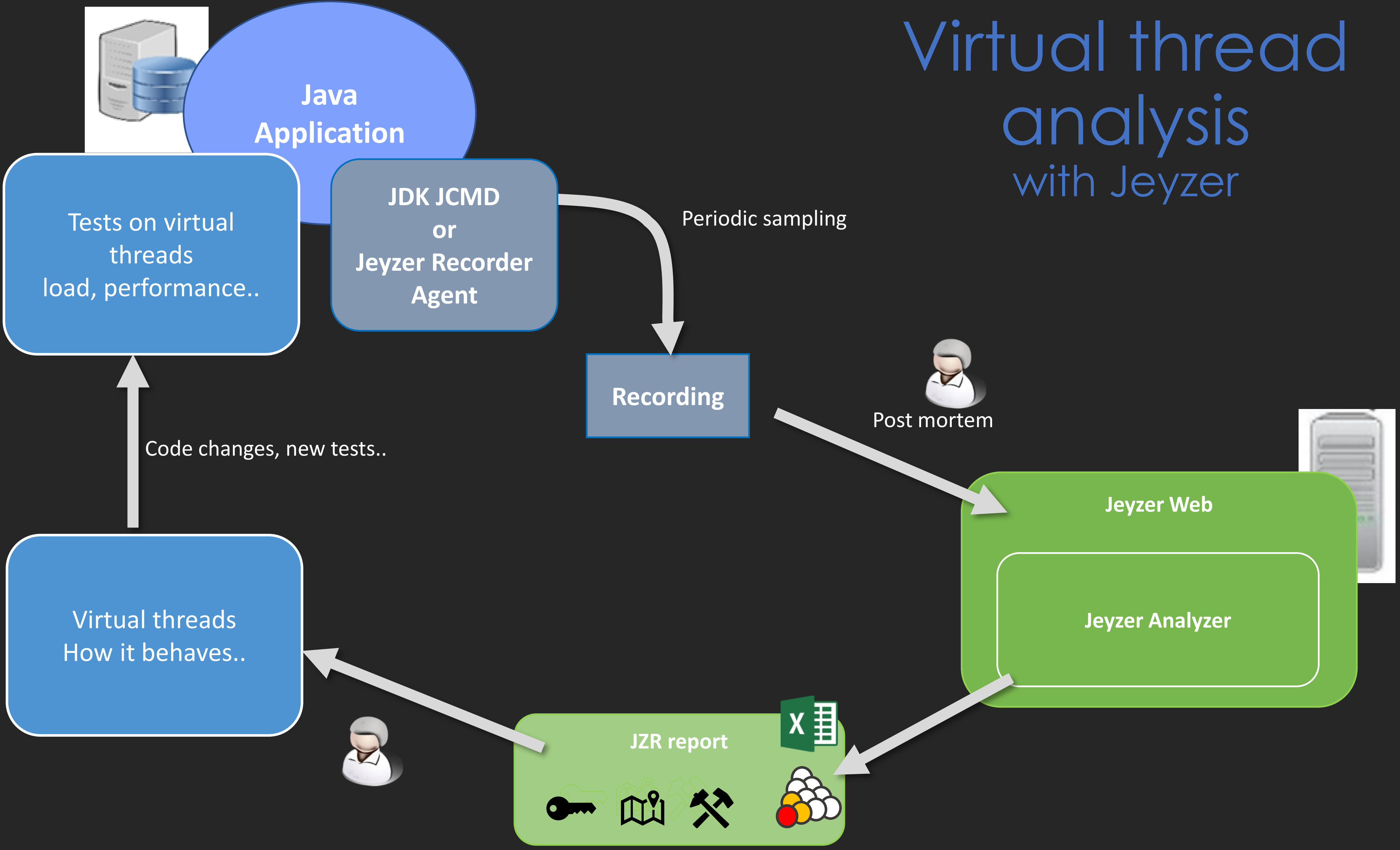
```
-javaagent:…jeyzer-agent.jar=…/config/agent/jeyzer-agent.xml;jeyzer-record-agent-profile=my_app
```

Jeyzer VT support - Analysis



- Supported in Jeyzer 3.1
- JSON and txt support
- **Identical and unmounted** virtual threads are **merged** for readability
- Carrier threads are filtered
- Analysis detection rules
  - Virtual thread usage : indication for non dev people
  - Virtual thread not visible warning : apply for standard recording methods
  - CPU usage on mounted threads (JFR) : core related. 100% CPU for VTs ?
  - **Unmounted virtual thread leak** : required

Virtual thread analysis
with Jeyzer



Jeyzer demo

- **Virtual threads demo** available in Jeyzer
  - Inspired from various Loom project demos
  - Educational purpose
  - Jeyzer analysis validation

- Several standard cases analyzed :
  - Pressure : creation, CPU bound, *IO bound*
  - Locking : synchronized, reentrant locking, *deadlock*

- Jeyzer Java profiles updated for virtual threads

Virtual threads – Under pressure

- ImageDownload demo : virtual vs native threads
    - Test : high connection load on remote server to get 5000 images
    - Virtual threads : 50% connection errors + open connection/VT “leakage”
    - Pseudo leak disappears when :
        - Connectivity down
        - Server connection timeout ?
    - Difference :
        - Executors.newFixedThreadPool(**100**) // native
        - Executors.newVirtualThreadPerTaskExecutor(); // virtual. **No safety belt**
    - Executors.newFixedThreadPool(**5000**) : same behavior
    - Solution :
        - add throttling code (ex: semaphore) in the VT implementation to reduce the load
          Semaphore semaph = new Semaphore(**100**);

Virtual threads – Manage the beast

- Migration to virtual threads is easy
- Standard functional tests will be green…
- Performance tests are **mandatory**
  - **Unleash the beast** : possible local and backend load issues to handle
  - Internal CPU / memory / **virtual thread monitoring**
- Impacts
  - Double check the fail-over on VT operations
  - **Slow down the beast** : add throttling with semaphore protection
  - Perf tests will also highlight the slow parts of your system : optimizations
- Monitoring / profiling / application behavior : go for Jeyzer

Virtual threads – Production ready ?

- Like for threads, we must have a view on virtual threads in case of production issue
- The only solution for now is JCMD or the Jeyzer Recorder Agent

So… are virtual threads production ready ?

Answer is **unlikely**

Reasons :

- JCMD is not production compliant : is JDK installed in production ?
- Jeyzer immediate acceptance in production environments ?

Virtual threads – What is missing ?

- Monitoring enhancements for production readiness
    - Proper high level monitoring API
    - Merge the unmounted virtual threads
        - Decrease memory footprint
        - Increase readability
    - Let’s wait for standard support in all monitoring tools/platforms
    - Provide more information around virtual threads :
        - Memory, CPU..
        - Locking
        - Deadlock detection
        - Logical units (StructuredTaskScope)
- More safety guard code to slow down the beast

Virtual threads - Conclusion

- Not really ready for production yet
  - **Do not rush in virtual thread usage on existing projects**
  - **Make sure underlying frameworks/containers do not use it**
  - Let’s wait for next Java updates : monitoring will for sure improve

  PS : in a pragmatic way, new LTS adoption in production takes always some time.

- Dev : anticipate the move
  - Assess the performance gains on your application, the move cost
  - Prepare safety belt code, identify the performance vulnerable operations
  - Follow the evolutions (StructuredTaskScope, ScopedValue)
  - **New** projects may start using it, assuming live on Java 21+ or Java 25

Takeaway - key points

Virtual threads -> IO wait

- Simple usage
  Large adoption
- Unleash the beast
  Thread waves & impacts
- Paradigms to come
  StructuredTaskScope

Monitoring goals

- Simple monitoring
  Standard availability
- Control the beast
- Activity flow insights
  Distributed…

Monitoring situation

production usage

- Poor ecosystem
  as of now

- Choice : risk, volumes, complexity…

Q&A time

The end

Thank you !!

And have a good virtual observability