CHAOS ENGINEERING

An Introduction to Application-level Chaos Engineering and Tools

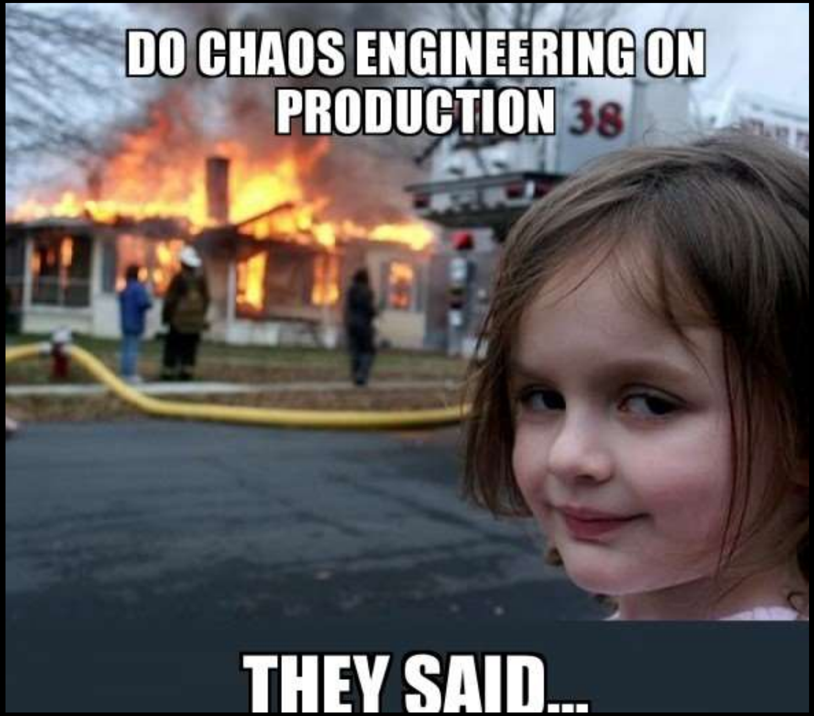
Carried out by – Akhil Yerrapragada (akhily@kth.se) and Nagasudeep Vemula (vemula@kth.se)

PRINCIPLES AND OUTLINE

- Verify that the system works.
- The input can be any kind of perturbation.
- Play with production environment.
- Automate the experiments.
- Make sure less people are infected.

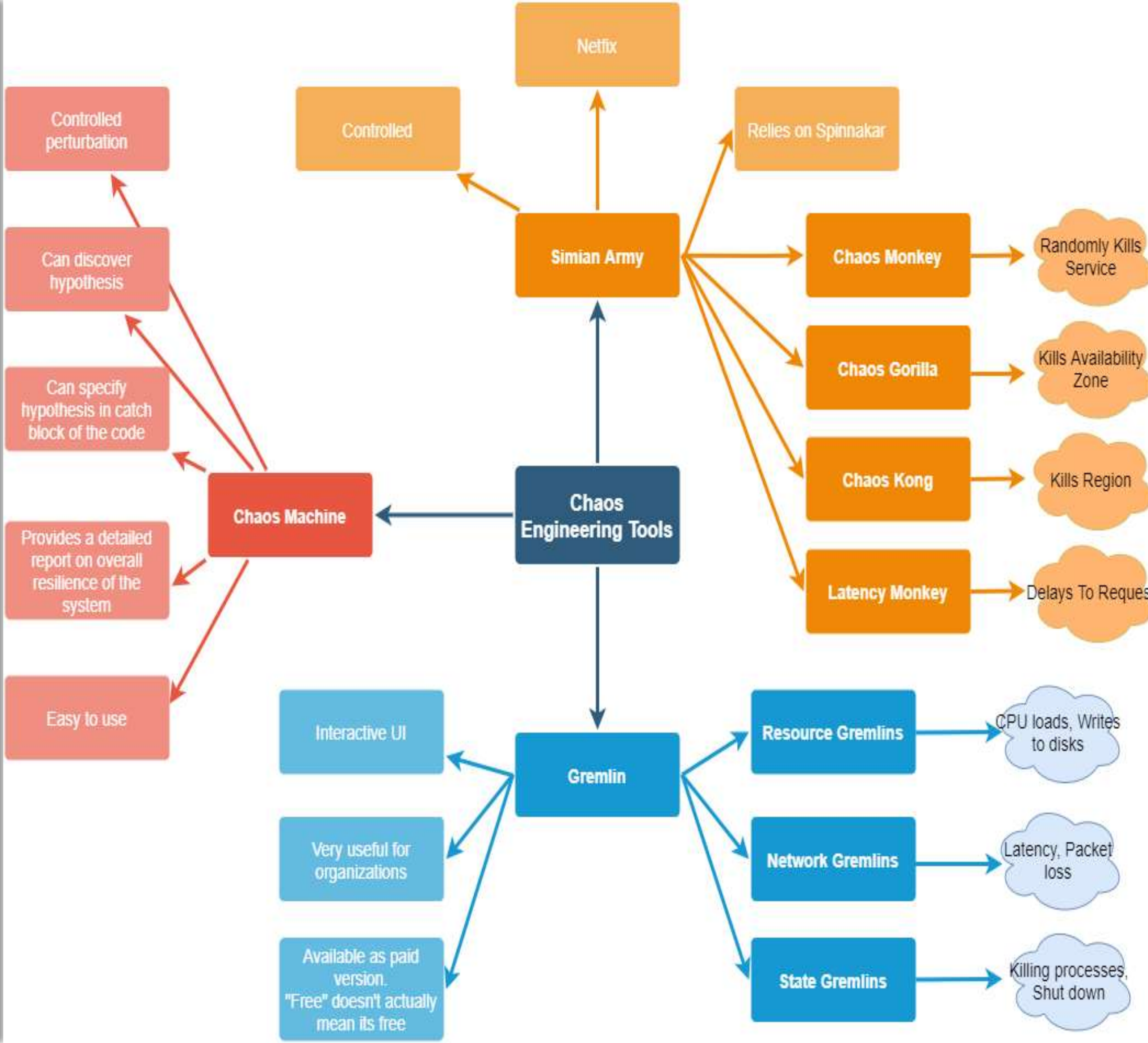
What can we add here?

- Make sure nobody loses anything.
- Do not inject perturbation if you cannot control it.



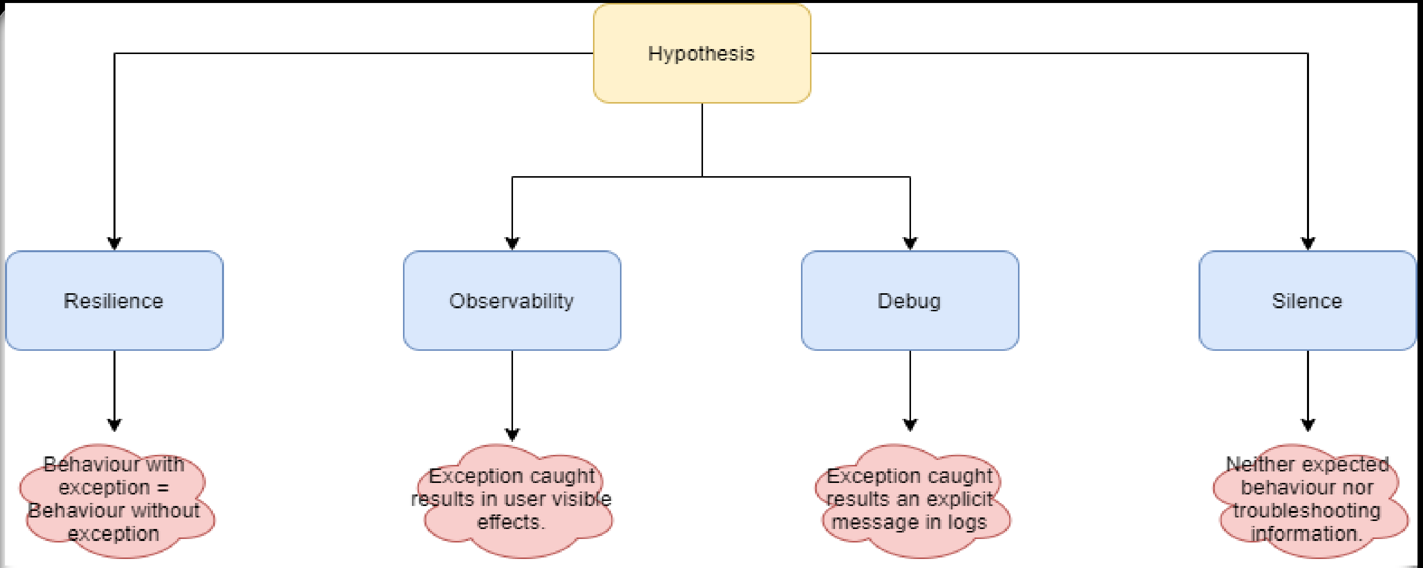
TOOLS TO CREATE CHAOS

- Simian Army
- Gremlin
- Chaos Machine
- Istio
- Pumba etc.….



CHAOS MACHINE

- Can we “try” chaos machine and “catch” exceptions?
- Goals – Specify, Falsify and Discover hypothesis.
- Types of hypothesis -------→
- Can we discover new hypothesis?
- How does it work? ---------→


Hypothesis
Resilience
Observability
Debug
Silence
Behaviour with exception = Behaviour without exception
Exception caught results in user visible effects.
Exception caught results an explicit message in logs
Neither expected behaviour nor troubleshooting information.

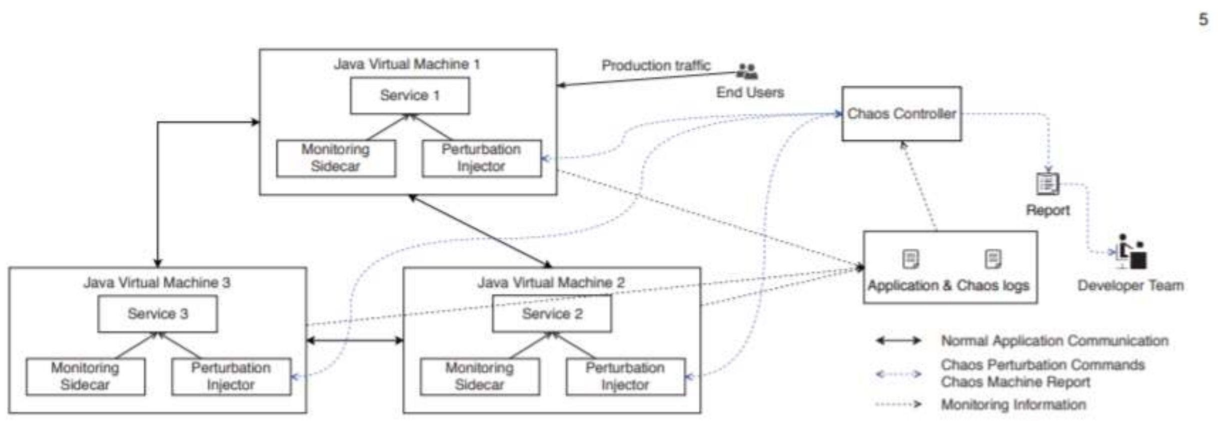
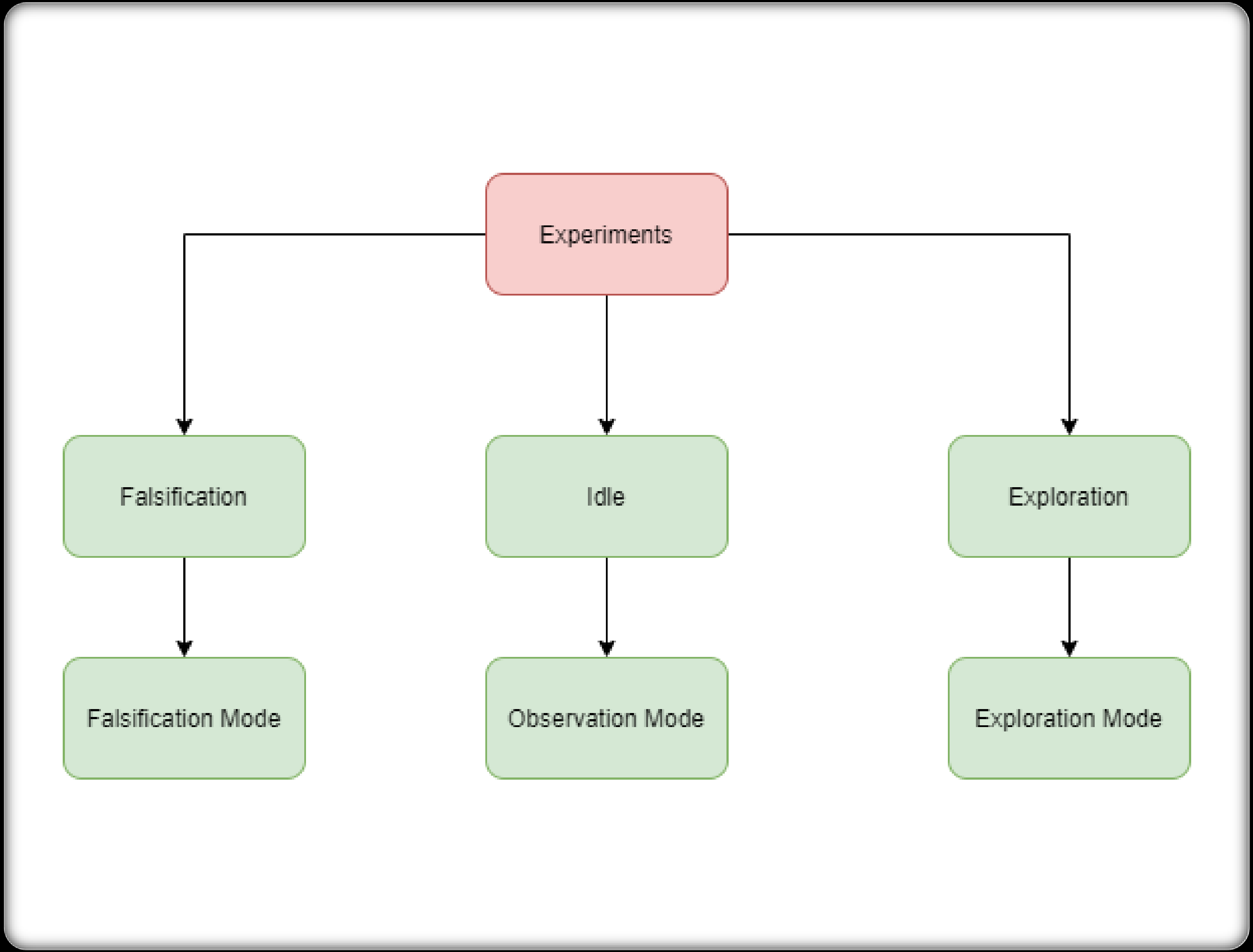


Fig. 1. The components of ChaosMachine

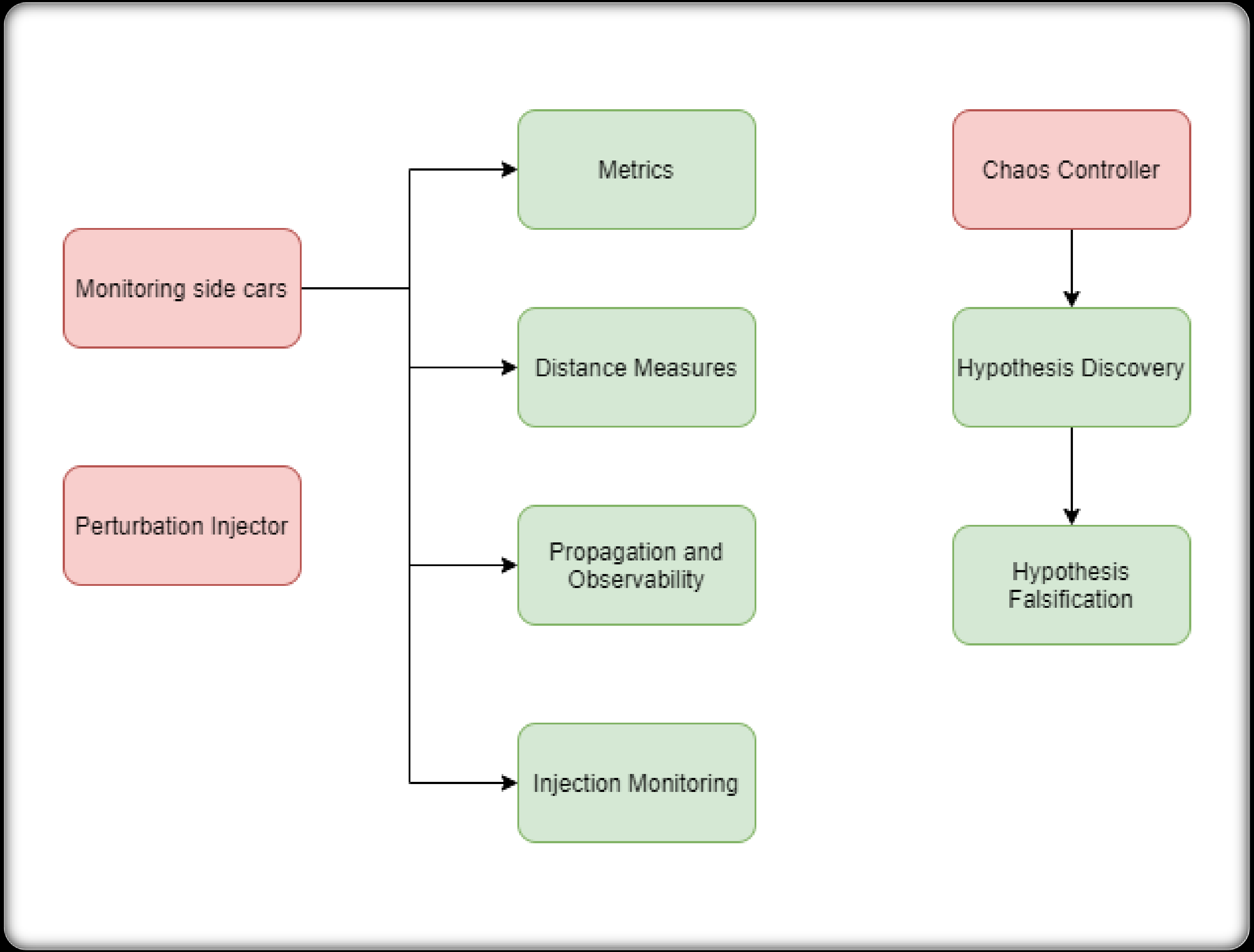
EXPERIMENTS WITH CHAOS MACHINE

- What type of experiments and modes available?
- Default is "Observation Mode"



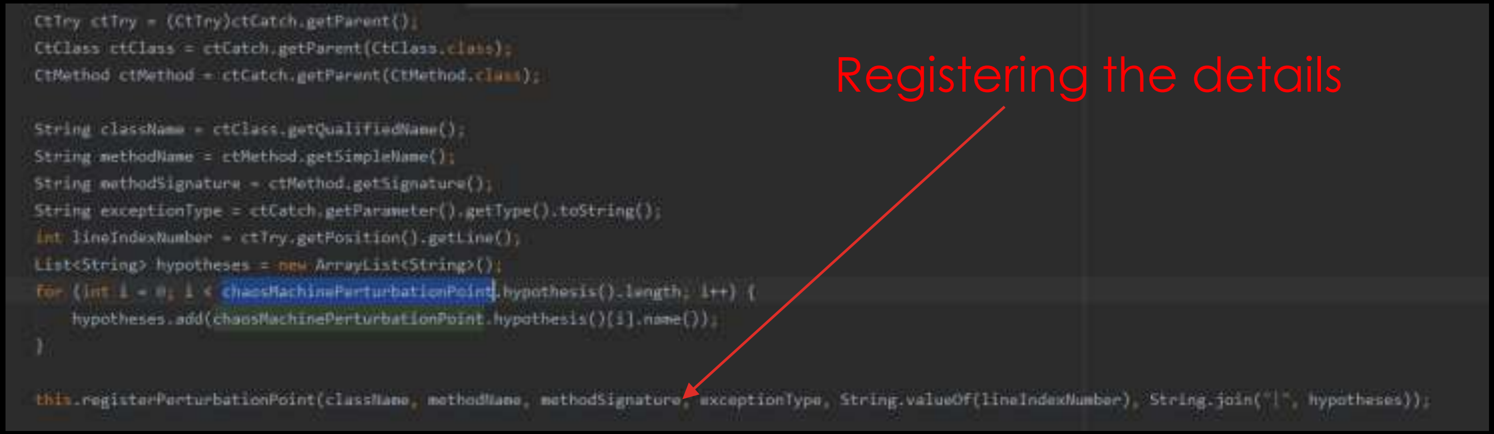
COMPONENTS OF CHAOS MACHINE

- 3 components as described
- The output will be a detailed report on the resilience of the system.

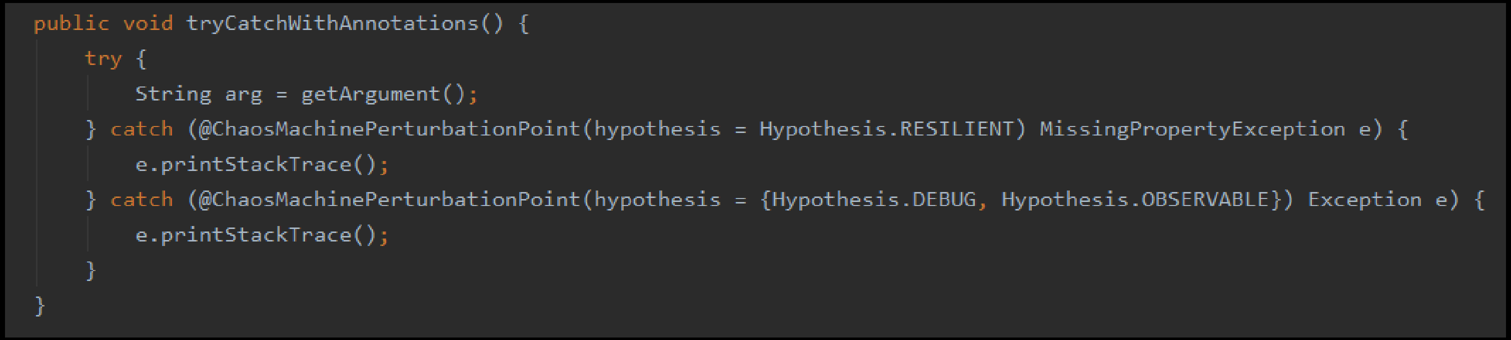


IMPLEMENTATION OF CHAOS MACHINE – ANNOTATION PROCESSOR

- Get the details of class, method, exception type, Index number of the specific try block and the hypothesis defined in it.
- Register the perturbation point details to the CSV file.
- An example of annotation will look something like this.



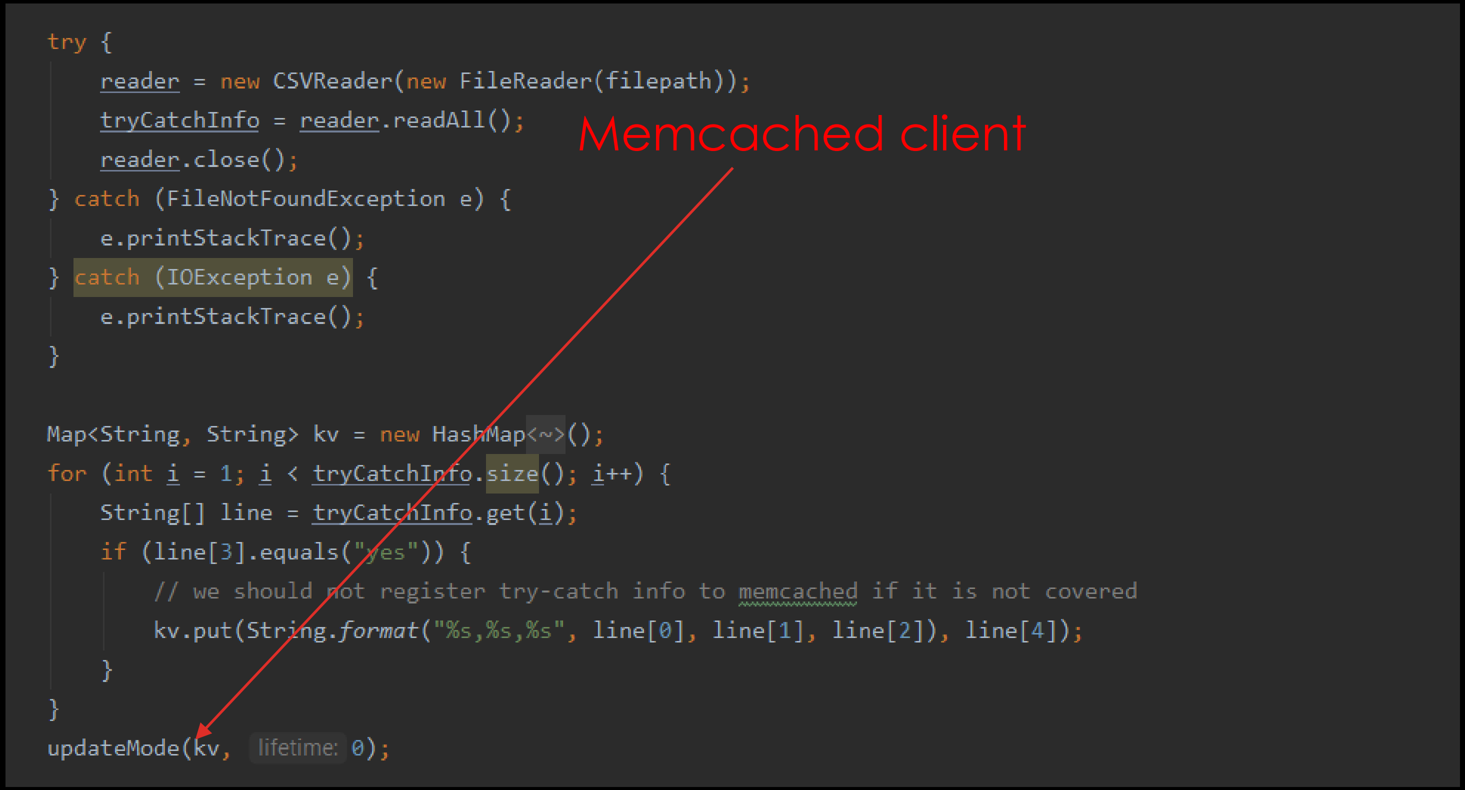
```java
CtTry ctTry = (CtTry)ctCatch.getParent();
CtClass ctClass = ctCatch.getParent(CtClass.class);
CtMethod ctMethod = ctCatch.getParent(CtMethod.class);

String className = ctClass.getQualifiedName();
String methodName = ctMethod.getSimpleName();
String methodSignature = ctMethod.getSignature();
String exceptionType = ctCatch.getParameter().getType().toString();
int lineIndexNumber = ctTry.getPosition().getLine();
List<String> hypotheses = new ArrayList<String>();
for (int i = 0; i < chaosMachinePerturbationPoint.hypothesis().length; i++) {
    hypotheses.add(chaosMachinePerturbationPoint.hypothesis()[i].name());
}

this.registerPerturbationPoint(className, methodName, methodSignature, exceptionType, String.valueOf(lineIndexNumber), String.join("|", hypotheses));
```



```java
public void tryCatchWithAnnotations() {
    try {
        String arg = getArgument();
    } catch (@ChaosMachinePerturbationPoint(hypothesis = Hypothesis.RESILIENT) MissingPropertyException e) {
        e.printStackTrace();
    } catch (@ChaosMachinePerturbationPoint(hypothesis = {Hypothesis.DEBUG, Hypothesis.OBSERVABLE}) Exception e) {
        e.printStackTrace();
    }
}
```

IMPLEMENTATION OF CHAOS MACHINE – CONTROLLER

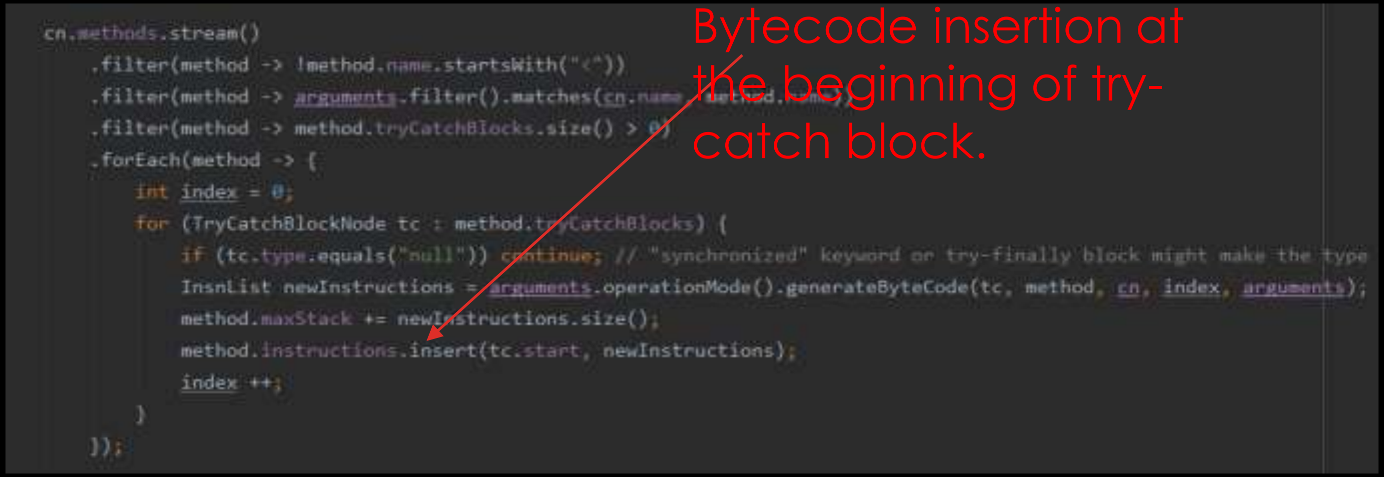
- Use Memcached – distributed memory cached system.
- Fetch the details of CSV.
- Update mode.
- Register it as Memcached client.
- Use JMX as monitoring tool to monitor processCpuTime, etc.
- Use log monitor to check the file updates.



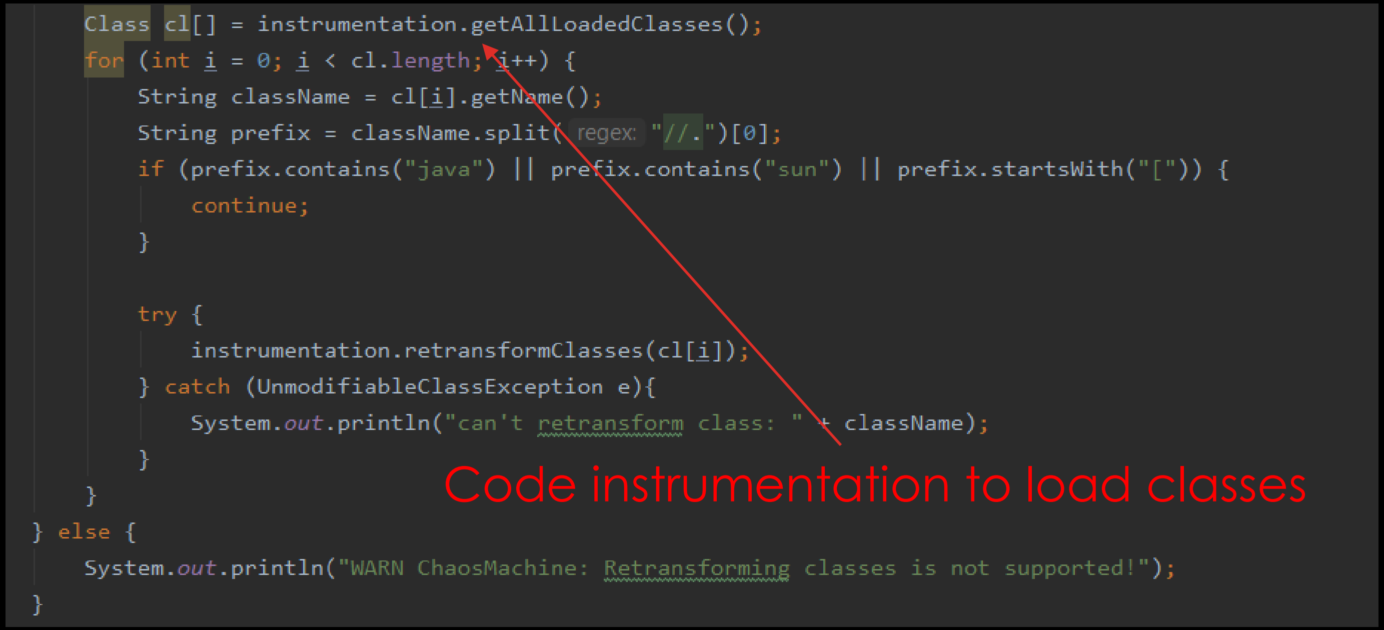
```
try {
    reader = new CSVReader(new FileReader(filepath));
    tryCatchInfo = reader.readAll();
    reader.close();
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}

Map<String, String> kv = new HashMap<~>();
for (int i = 1; i < tryCatchInfo.size(); i++) {
    String[] line = tryCatchInfo.get(i);
    if (line[3].equals("yes")) {
        // we should not register try-catch info to memcached if it is not covered
        kv.put(String.format("%s,%s,%s", line[0], line[1], line[2]), line[4]);
    }
}
updateMode(kv,  lifetime: 0);
```

IMPLEMENTATION OF CHAOS MACHINE – PERTURBATION INJECTOR

- Generate perturbation when chaos controller sends the command.
- Classes are loaded using automated code instrumentation.
- Can be activated and deactivated individually.



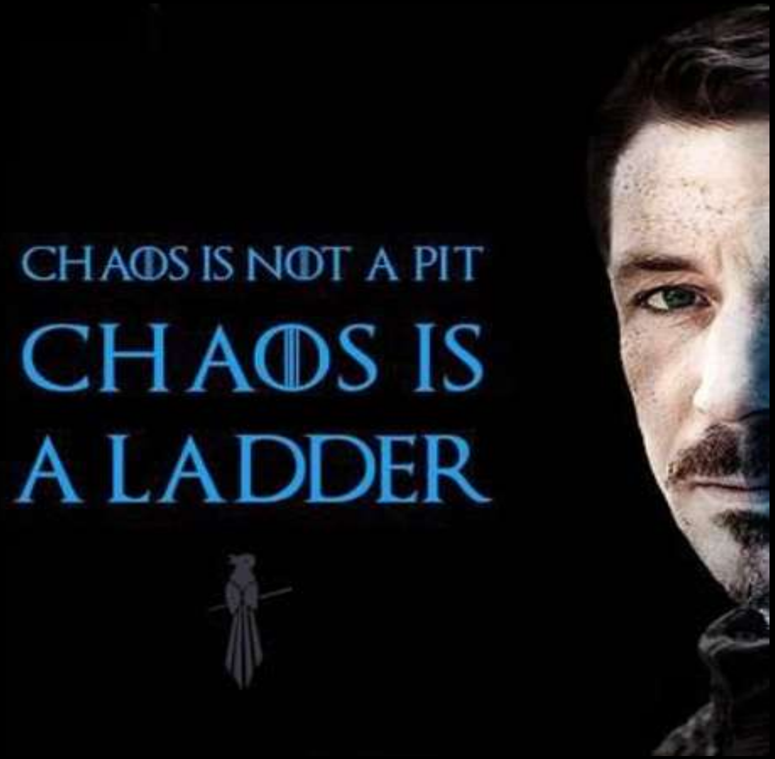
```java
cn.methods.stream()
        .filter(method -> !method.name.startsWith("<"))
        .filter(method -> arguments.filter().matches(cn.name, method.name))
        .filter(method -> method.tryCatchBlocks.size() > 0)
        .forEach(method -> {
            int index = 0;
            for (TryCatchBlockNode tc : method.tryCatchBlocks) {
                if (tc.type.equals("null")) continue; // "synchronized" keyword or try-finally block might make the type
                InsnList newInstructions = arguments.operationMode().generateByteCode(tc, method, cn, index, arguments);
                method.maxStack += newInstructions.size();
                method.instructions.insert(tc.start, newInstructions);
                index ++;
            }
        });
```



```java
    Class cl[] = instrumentation.getAllLoadedClasses();
    for (int i = 0; i < cl.length; i++) {
        String className = cl[i].getName();
        String prefix = className.split( regex: "//.")[0];
        if (prefix.contains("java") || prefix.contains("sun") || prefix.startsWith("[")) {
            continue;
        }

        try {
            instrumentation.retransformClasses(cl[i]);
        } catch (UnmodifiableClassException e){
            System.out.println("can't retransform class: " + className);
        }
    }
} else {
    System.out.println("WARN ChaosMachine: Retransforming classes is not supported!");
}
```

CONCLUSION

- As seen time and again, traditional testing methods just do not seem to lead to reliable services as they cannot predict all the sources of failure



- By making Chaos Engineering part of the DevOps culture, developers and stakeholders continually embrace failure as a way to prepare for and prevent it, resulting in stronger and more resilient applications.

- We strongly believe that in the coming years chaos engineering will transform from something occult in nature to more commonplace this is largely due to the frequency of releases brought about by devops and integrating CE contributes to improved continuous testing.

REFERENCES

- Principles of chaos engineering - https://principlesofchaos.org/?lang=ENcontent
- Gremlin Introduction - https://www.infoq.com/news/2017/12/gremlin-chaos-engineering/
- Failure as a service (FAAS) - https://www.semanticscholar.org/paper/Failure-as-a-Service-(FaaS)%3A-A-Cloud-Service-for-Gunawi-Do/2c7bfc8d75dab44aeab34b1bf5243b192112f502
- Chaos Monkey for Springboot - https://www.baeldung.com/spring-boot-chaos-monkey
- Royal Chaos - https://github.com/KTH/royal-chaos
- A Chaos Engineering System for Live Analysis and Falsification of Exception-handling in the JVM - https://arxiv.org/pdf/1805.05246.pdf