University of Sheffield

Dynamic Analysis

Software Reengineering
(COM3523 / COM6523)

The University of Sheffield

Static Analysis has its limitations

Too much information.

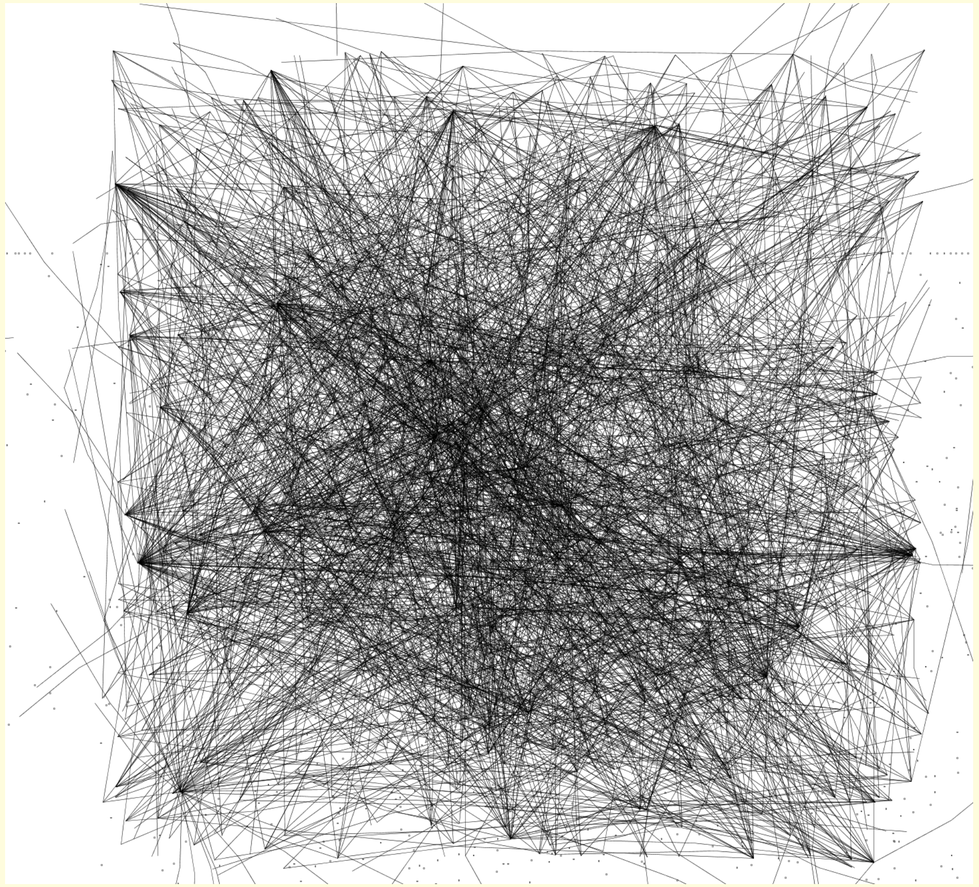
- Will consider events in the code that are impossible in practice.
- What if we are only interested in a particular use-case?

Inherently **conservative**.

- Can produce lots of relations that are spurious or infeasible.
- E.g. - Produce method calls that are not possible for any actual program execution.

Many things are impossible to determine from source code analysis alone.

- Performance-related questions.
- Issues wrt. particular usage scenarios.



Call graph for Apache Commons Math.

Dynamic Analysis

Extracting information from **program executions**.

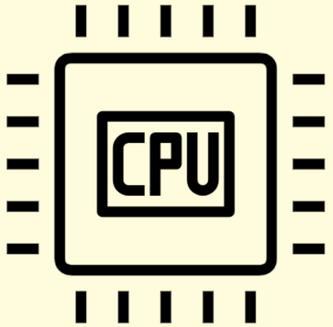
- Function calls, input-events, network-signals, …
- Call stack depth, object states, output values, parameter inputs, memory usage, clock-time, etc.

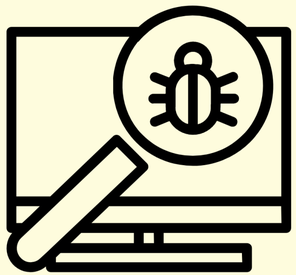
Provide information about ``software behaviour''.

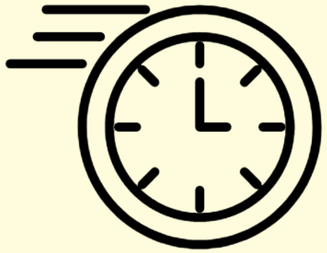
- Which elements are executed, when, for how long, …

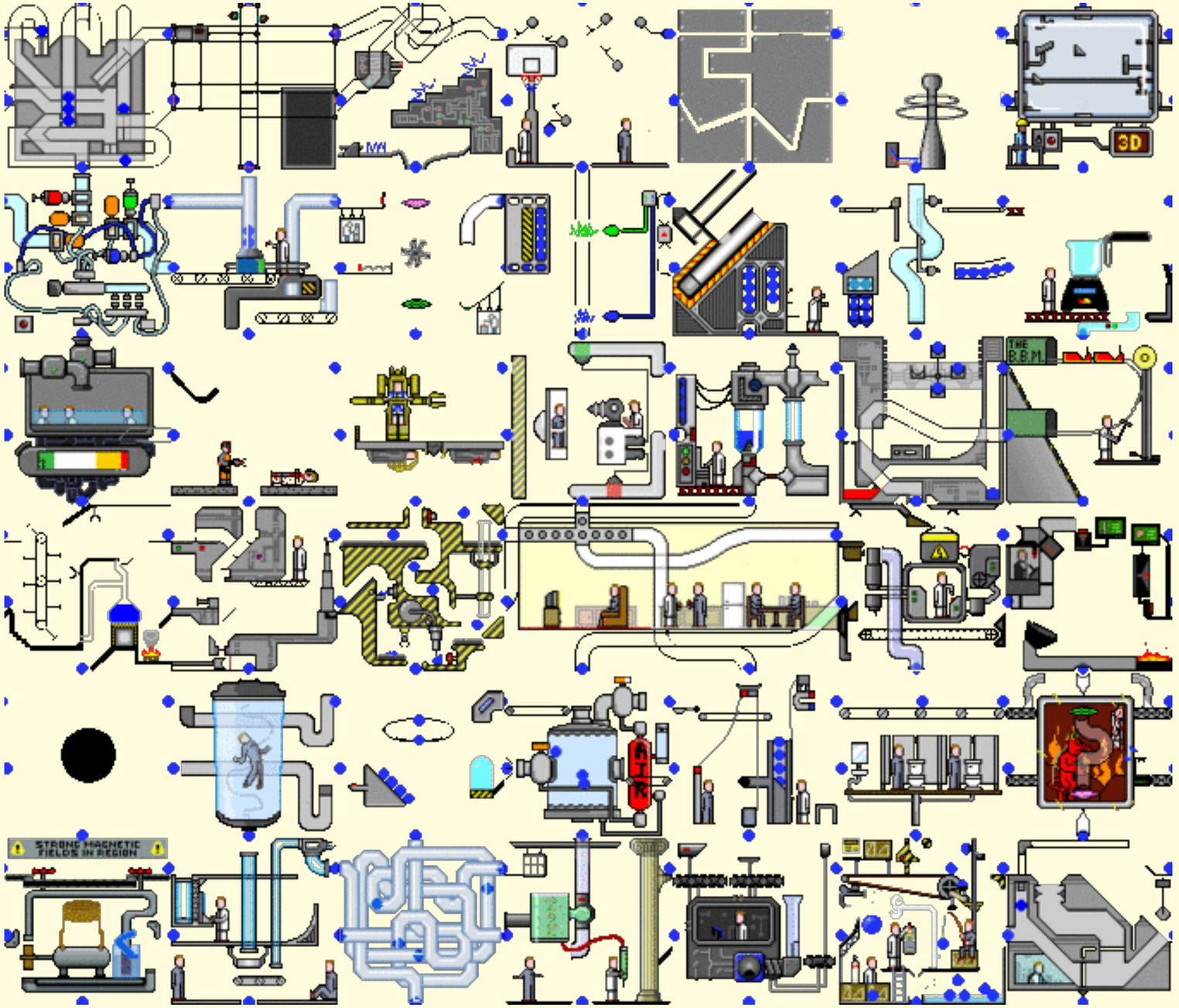
Common dynamic analyses:

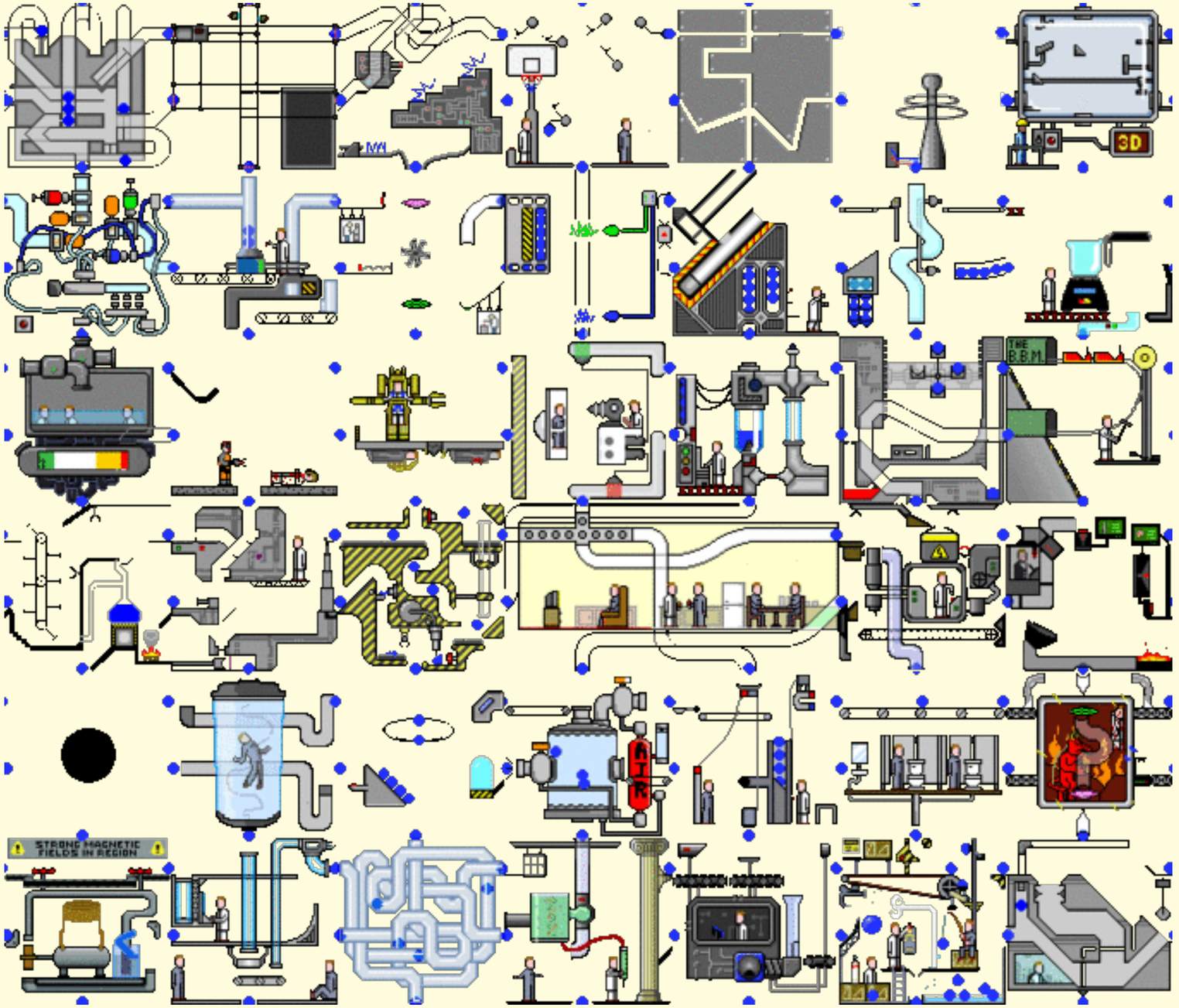
- Testing
- Profiling
- Logging
- Debugging








3D
THE B.B.M.
STRONG MAGNETIC FIELDS IN REGION


3D
THE B.B.M.
STRONG MAGNETIC FIELDS IN REGION

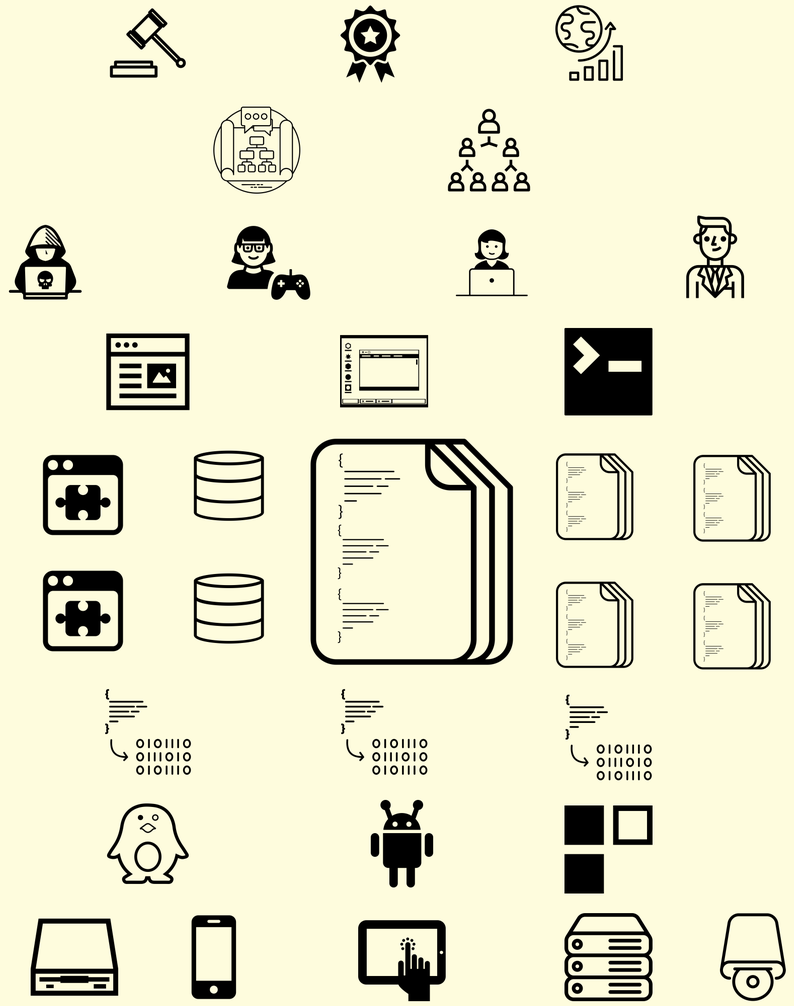
Execution Traces

A Trace is a “record” of an execution.

- Collection of time-stamped events.
- Can be at any level in the system - from hardware to business process.
- Commonly comprise an event (e.g. a method call), coupled with some data (e.g. system state information, performance data)

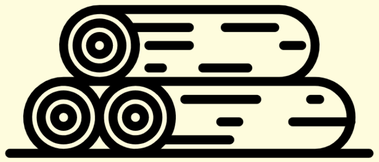
Can apply to distributed systems, or monolithic ones.

- Reasoning about order of trace-elements becomes harder in distributed / concurrent systems.



Loggers

Tools to automate the collection of runtime data.



Need a better alternative than System.out.println(…)!

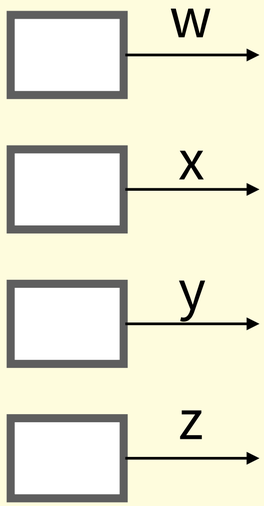
Can specify:

Where information is funnelled - console, file, database, …

Logging instructions tend to be inserted into code by developers.

When logging occurs - for info, debugging, errors,…

How information is presented / formatted.



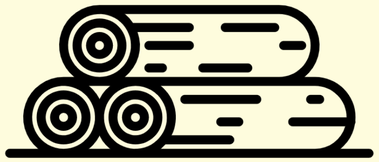
Lots of logging frameworks

java.util.logging, slf4j, log4j, …

Still need to insert log-statements into code…

Loggers

Tools to automate the collection of runtime data.



Need a better alternative than System.out.println(…)!

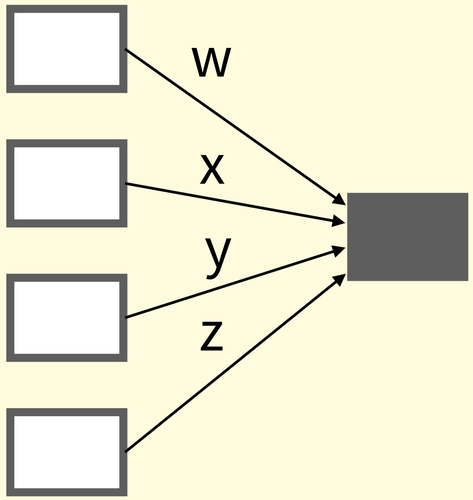
Can specify:

Where information is funnelled - console, file, database, …

Logging instructions tend to be inserted into code by developers.

When logging occurs - for info, debugging, errors,…

How information is presented / formatted.



Lots of logging frameworks

java.util.logging, slf4j, log4j, …

Still need to insert log-statements into code…

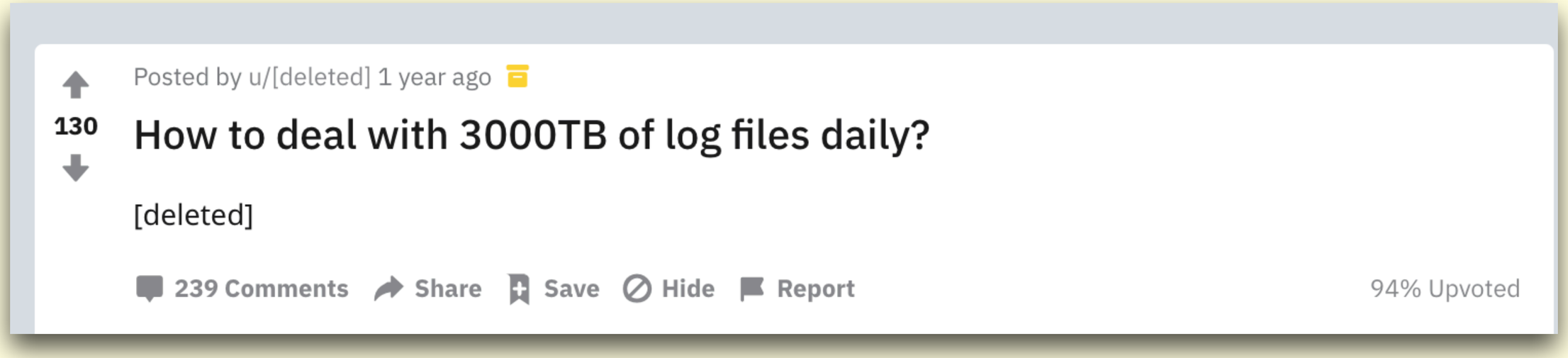
Precision is important

The size of a trace file is impossible to predict in general.

- See “Halting Problem”.

Large traces are harder to analyse and manage.

- Important to be selective about where, when and what to log.



Trade-offs galore

	Human effort to set-up / remove	Detail / level of information about application	Performance impact	Trace volume

Trade-offs galore

	Human effort to set-up / remove	Detail / level of information about application	Performance impact	Trace volume
Print-statements inserted by developer	😭	😐	🙂	🙂

Trade-offs galore

	Human effort to set-up / remove	Detail / level of information about application	Performance impact	Trace volume
Print-statements inserted by developer	😭	😐	🙂	🙂
Using dTrace or bTrace to record OS-level events	🙂	😐	🙂	😐

Trade-offs galore

	Human effort to set-up / remove	Detail / level of information about application	Performance impact	Trace volume
Print-statements inserted by developer	😭	😐	🙂	🙂
Using dTrace or bTrace to record OS-level events	🙂	😐	🙂	😐
Automatically instrumenting code	🙂	🤩	😐	😐

Trade-offs galore

	Human effort to set-up / remove	Detail / level of information about application	Performance impact	Trace volume
Print-statements inserted by developer	😭	😐	🙂	🙂
Using dTrace or bTrace to record OS-level events	🙂	😐	🙂	😐
Automatically instrumenting code	🙂	🤩	😐	😐
Recording inputs and outputs - truly “black-box”	🤩	😐	🤩	🤩

The Heisenbug

A bug that disappears / alters its behaviour when you attempt to observe it.

The Heisenbug

A bug that disappears / alters its behaviour when you attempt to observe it.

Named after the “Heisenberg Effect” in quantum mechanics.

- The act of observing a system inevitably changes its state.

The Heisenbug

A bug that disappears / alters its behaviour when you attempt to observe it.

Named after the “Heisenberg Effect” in quantum mechanics.

- The act of observing a system inevitably changes its state.

A risk when the dynamic analysis interferes with the application environment.

The Heisenbug

A bug that disappears / alters its behaviour when you attempt to observe it.

Named after the “Heisenberg Effect” in quantum mechanics.

- The act of observing a system inevitably changes its state.

A risk when the dynamic analysis interferes with the application environment.

Examples:

- Time-sensitive programs - dynamic analysis can slow the program down.
- Programs that rely on disk access - trace storage can interfere with this.
- Programs that rely on compiler optimisations - trace instrumentation can alter the optimisations,
- …

The Heisenbug

A bug that disappears / alters its behaviour when you attempt to observe it.

Named after the “Heisenberg Effect” in quantum mechanics.

- The act of observing a system inevitably changes its state.

A risk when the dynamic analysis interferes with the application environment.

Examples:

- Time-sensitive programs - dynamic analysis can slow the program down.
- Programs that rely on disk access - trace storage can interfere with this.
- Programs that rely on compiler optimisations - trace instrumentation can alter the optimisations,
- …

Reinforces need to be selective when instrumenting a program.

Sound but imprecise versus unsound but precise

Static analysis:

Conservative - considers every possible (and many impossible) executions.

Imprecise (e.g. lots of redundant calls in a call graph).

Sound. Every possible call is in the call graph.

Dynamic analysis:

Partial - only considers a single execution.

Precise - it it occurs in the trace it **must** be possible.

Unsound - observations may not generalise to **every** execution.

Ernst, Michael D. "Static and dynamic analysis: Synergy and duality." WODA 2003: ICSE Workshop on Dynamic Analysis. 2003.

Sound but imprecise versus unsound but precise

Static analysis:

Conservative - considers every possible (and many impossible) executions.

Imprecise (e.g. lots of redundant calls in a call graph).

Sound. Every possible call is in the call graph.

Dynamic analysis:

Partial - only considers a single execution.

Precise - it it occurs in the trace it **must** be possible.

Unsound - observations may not generalise to **every** execution.

Ernst, Michael D. "Static and dynamic analysis: Synergy and duality." WODA 2003: ICSE Workshop on Dynamic Analysis. 2003.

Automatically Instrumenting Java Code

Aspects

A way to modularise software in terms of **“cross-cutting concerns”**.

- Motivated by the high amount of duplicate code in systems.

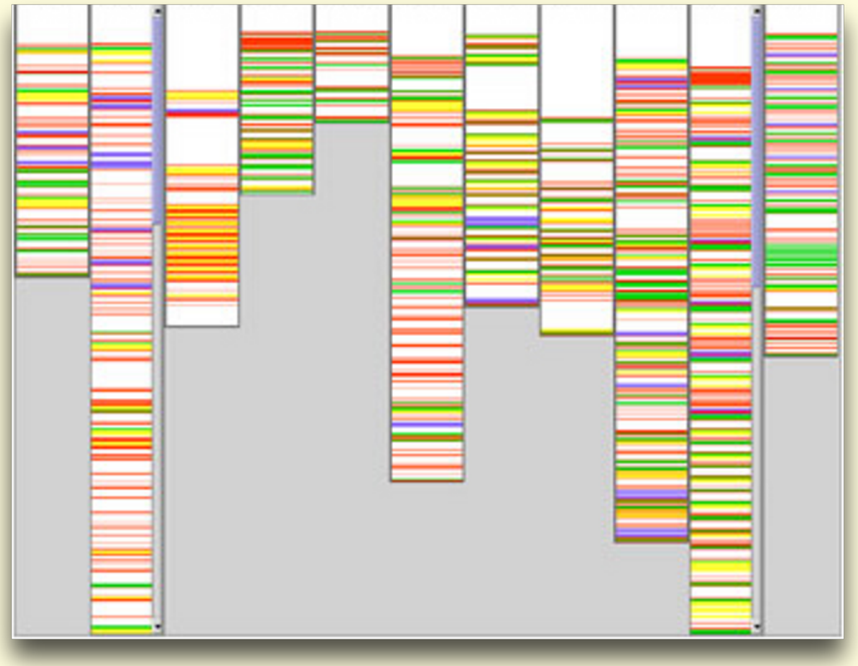
Capture code that deals with a particular ‘aspect’ (known as an **“advice”**).

- Fragment of code to be used throughout a program.
- E.g. File-handling, data-base management, logging…

Advice is **“woven”** into software execution via **“join points”**.

- E.g. method entry or exit points, field accesses, etc.

A specification that links advice to a join-point is known as a **“point cut”**.



Cross-cutting concerns from a 19k component. (from Bruntink, van Duersen and Tourwé)

Using Aspects in Java

There are many dedicated Aspect-Oriented Programming Libraries that could be used.

- AspectJ is one of the main ones for Java.
- Too heavyweight.

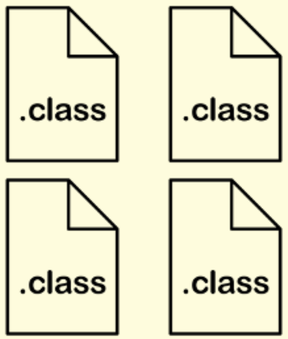
Java includes the ability to write “Transformers” that modify classes during execution.

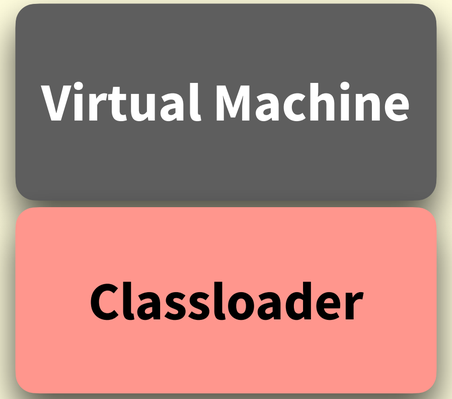
- Provide the infrastructure to implement basic aspect-oriented concepts, including automated logging.

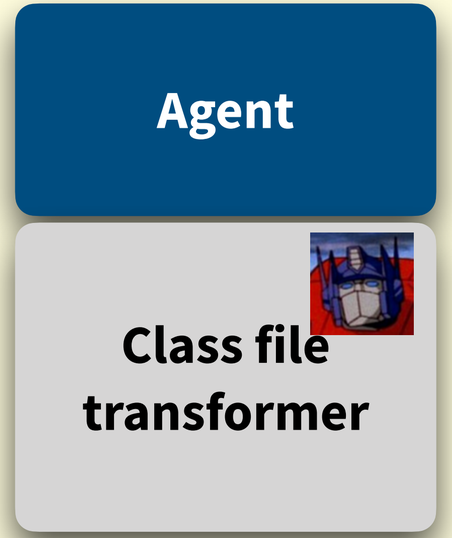
Deployed as “agents”.

- “Attached” to a Java program when it is run, have the ability to alter a class when it is loaded.

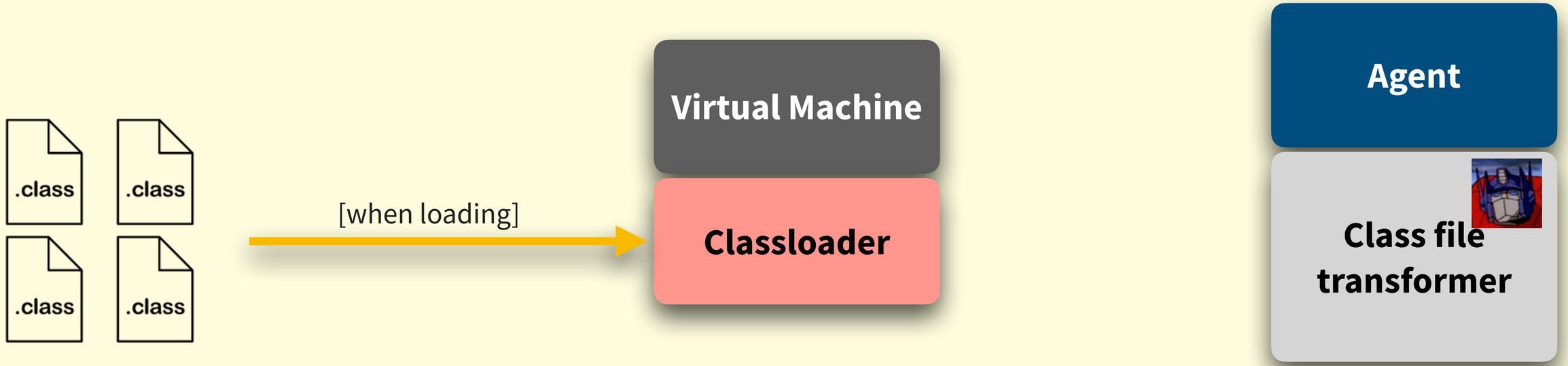
How it works


.class
.class
.class
.class

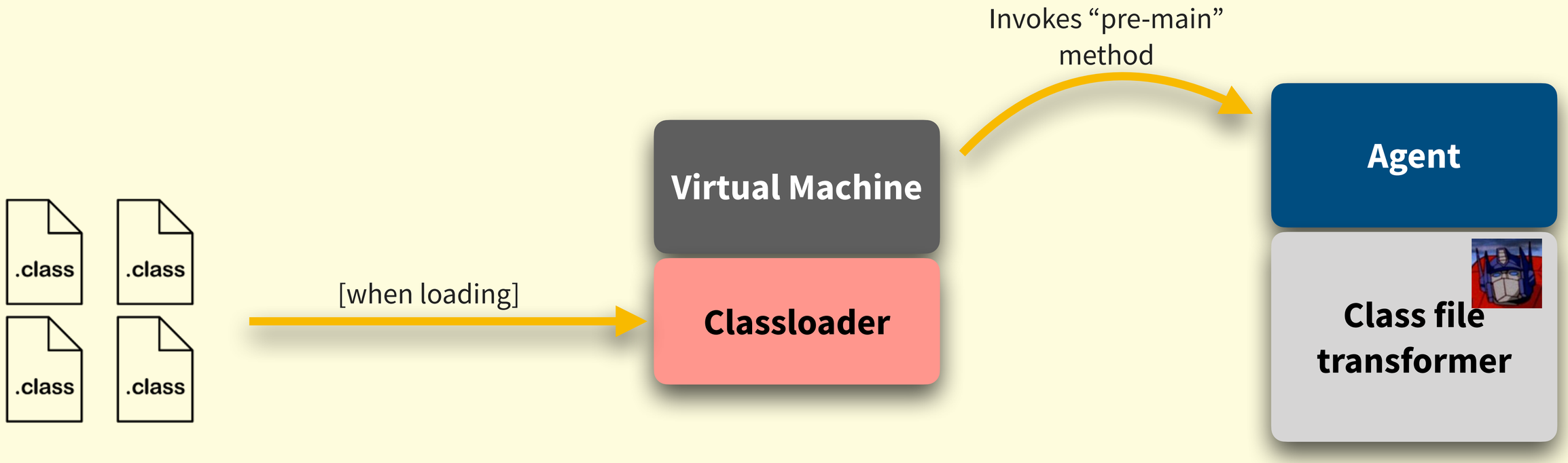

Virtual Machine
Classloader


Agent
Class file transformer

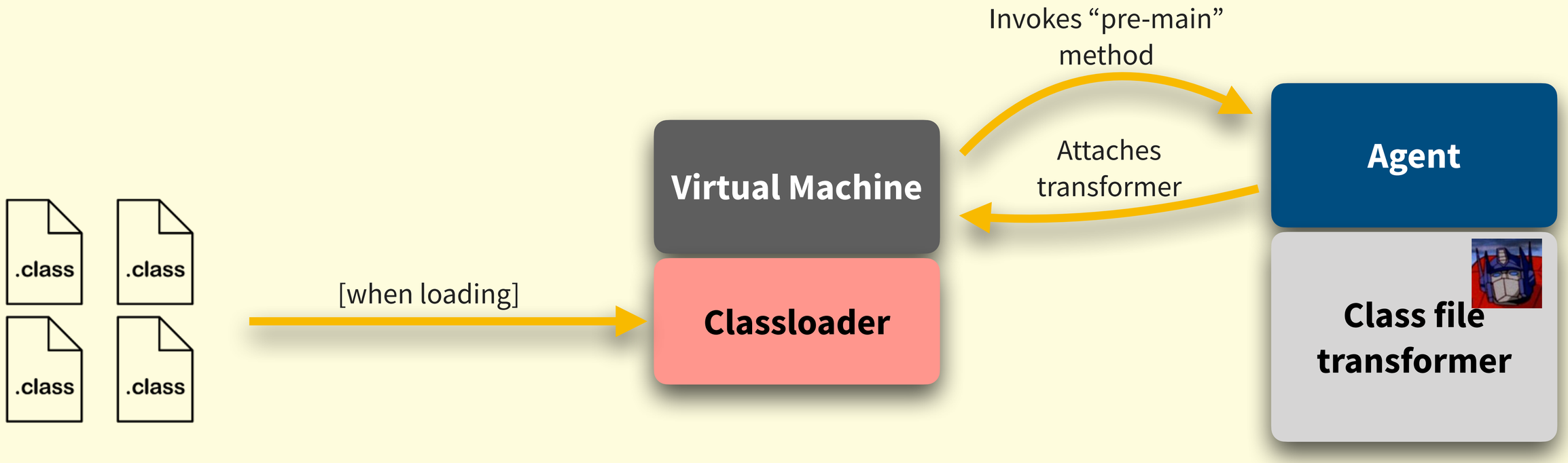
How it works


.class
.class
.class
.class
[when loading]
Virtual Machine
Classloader
Agent
Class file transformer

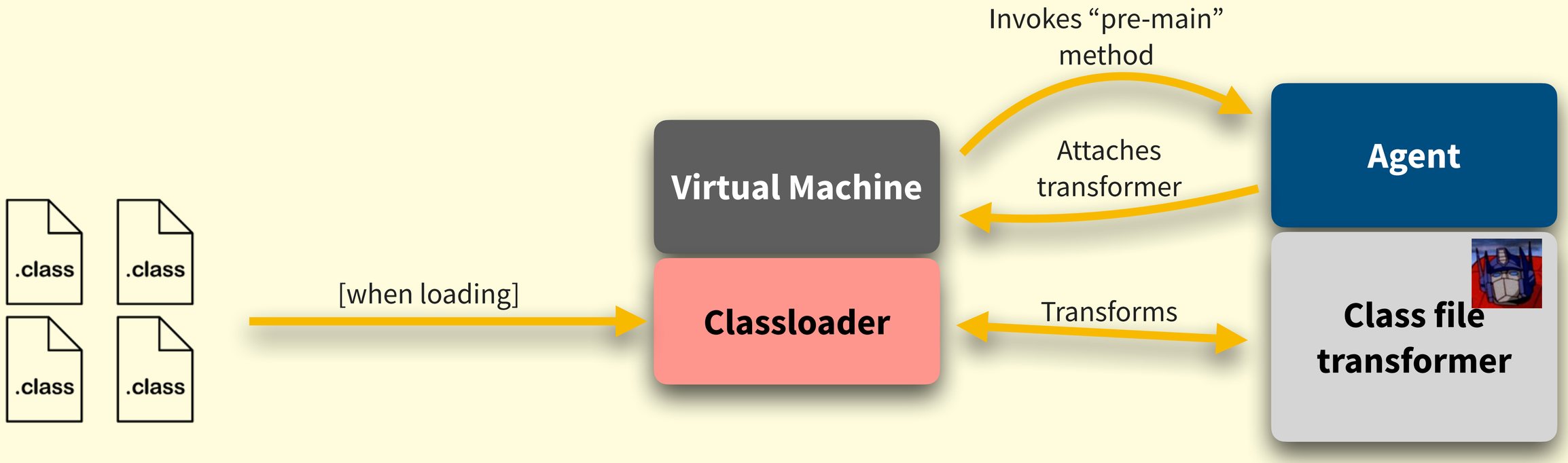
How it works


Invokes “pre-main” method
.class
.class
.class
.class
[when loading]
Virtual Machine
Classloader
Agent
Class file transformer

How it works


Invokes “pre-main” method
Attaches transformer
Virtual Machine
Classloader
Agent
Class file transformer
.class
.class
.class
.class
[when loading]

How it works


Invokes “pre-main” method
Attaches transformer
Virtual Machine
Agent
.class
.class
.class
.class
[when loading]
Classloader
Transforms
Class file transformer

Demo

Answering Questions with Dynamic Analysis

Gauging Importance

Count the number of occurrences of a class or a method in a trace.

Those that are executed frequently are obvious targets for optimisation.

- May be executed within a loop, or called by multiple client classes.

Particularly important to weigh up against corresponding static analysis metrics.

- If it's a small method, less scope for optimisation.
- If it's big and cumbersome *and* frequently executed, it's a key reengineering target.

What to plot:

- Name of method / class versus number of occurrences in the trace.

Identifying bottlenecks

Which methods might be inefficient CPU / Memory hoggers?

A trace can link a trace-point to non-functional details.

- Memory / CPU availability, etc.

Can also estimate clock-time required for a method.

- By calculating time-difference of instrumentation-point to the time recorded for subsequent point.

Since a single method can execute frequently, we need to take the average across all points.

- Can be done via a pivot-table in Excel.

What to plot:

- Method vs. average resource consumption.

Phase analysis

Identify distinctive “phases” of execution.

- Which segments of a trace correspond to specific “functionalities” in the system?

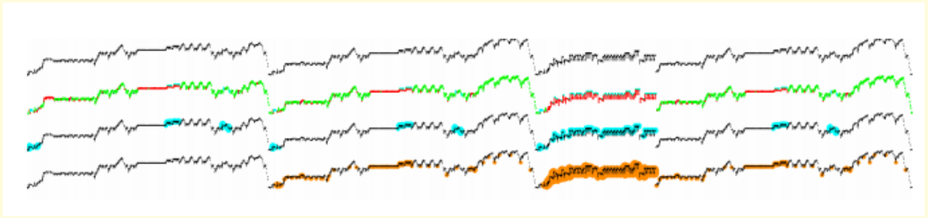
Need to identify patterns within sequences of variables.

- Tricky with multiple variables and noisy data.

Can start by visualising trace as a time-series

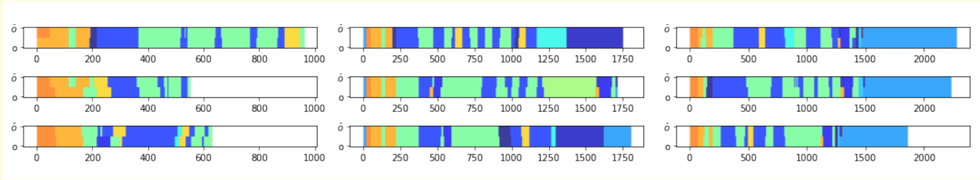
- Can visualise internal data-state variables.
- Can also visualise state of the VM (e.g. stack-depth or memory usage).

Can be used to explore which classes / methods feature in different phases.



Depth-encoded traces.

Kuhn & Greevy. "Exploiting the analogy between traces and signal processing." ***ICSM*** **2006.**



Ataiefard, Mashhadi, Hemmati & Walkinshaw "Deep State Inference: Toward Behavioral Model Inference of Black-Box Software Systems." ***IEEE Transactions on Software Engineering*** **48.12 (2021)**

Feature Identification

Which methods are relevant to a specific run-time feature?

E.g. which methods are particularly relevant to loading a file in MS Word?

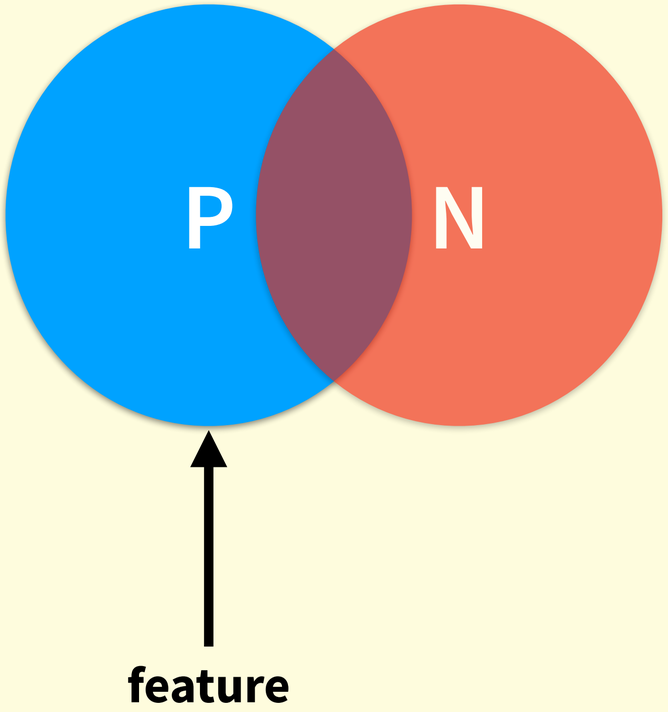
Cannot simply take *all* of the methods in a trace that happens to involve loading a file.

Would include lots of irrelevant methods.

"Software Reconnaissance"

Two sets of traces: One set that involves the feature (P), and one set that definitely doesn't (N).

Subtract all of the elements in N from P.



Wilde, Norman, and Michael C. Scully. "Software reconnaissance: Mapping program features to code." Journal of Software Maintenance: Research and Practice 7.1 (1995): 49-62.

Key Takeaways

Dynamic analysis is the process of recording runtime data.

Commonly operates on “execution traces”.

- Recordings of software executions, captured as sequences of events.

Can be much more precise than static analysis, but can also be less sound.

- Relies on the analyst “covering” all of the relevant software behaviour.

Traces can be very large.