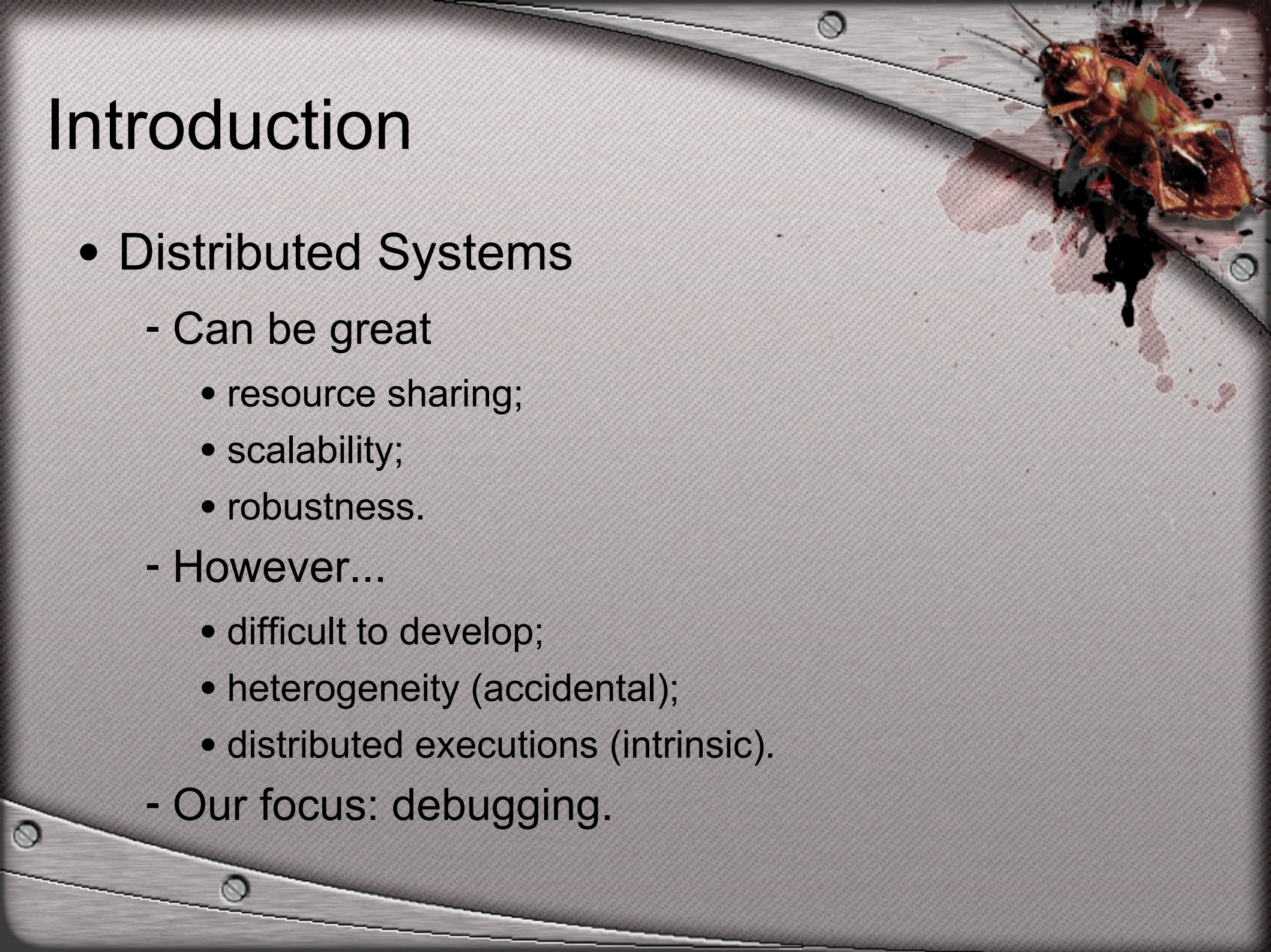
Univesidade de São Paulo
Instituto de Matemática
e Estatística

An Eclipse-Based Tool for Symbolic Debugging of Distributed Object Systems

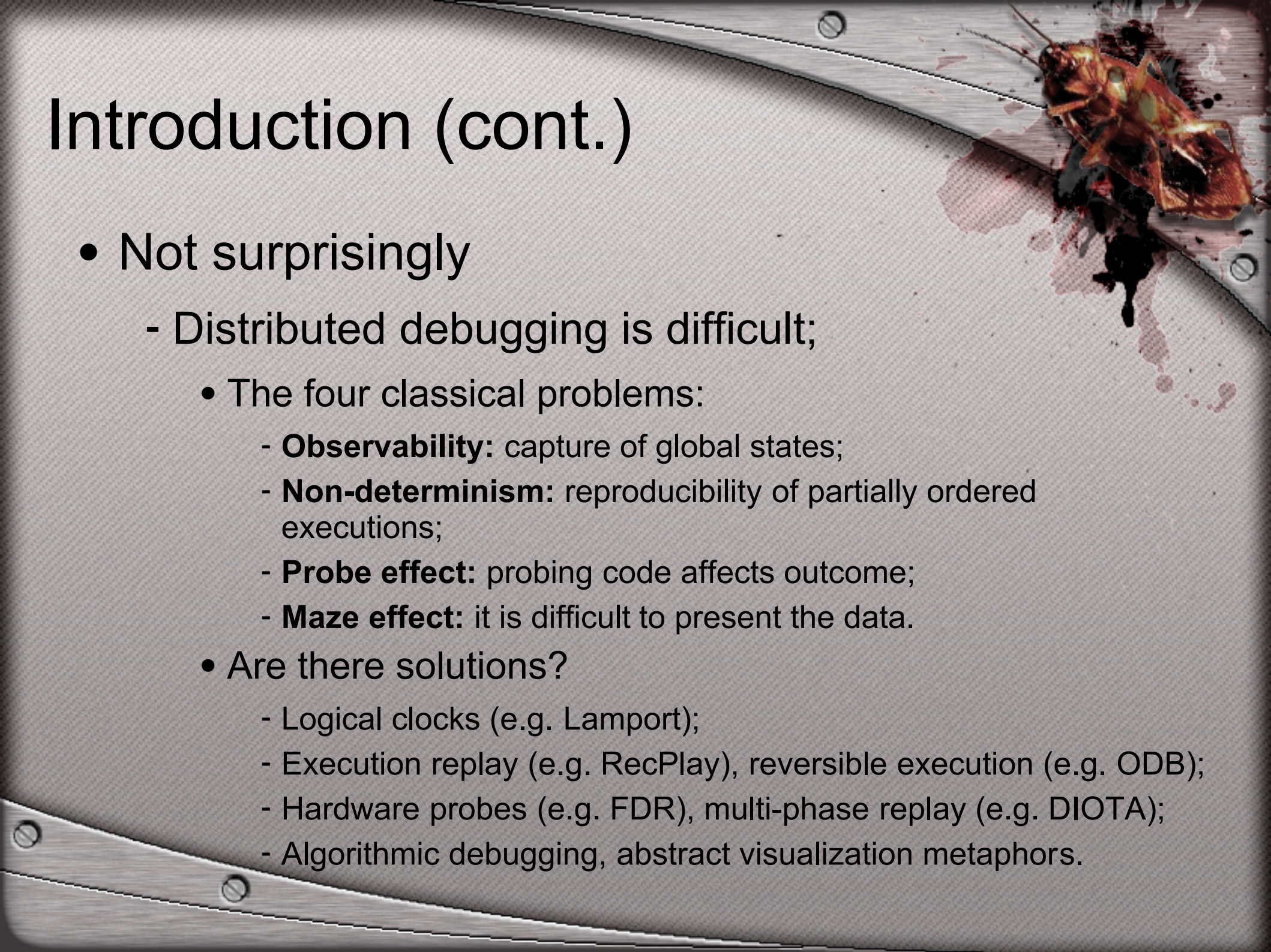
9th International
Symposium on Distributed
Objects, Middleware and Applications

Introduction



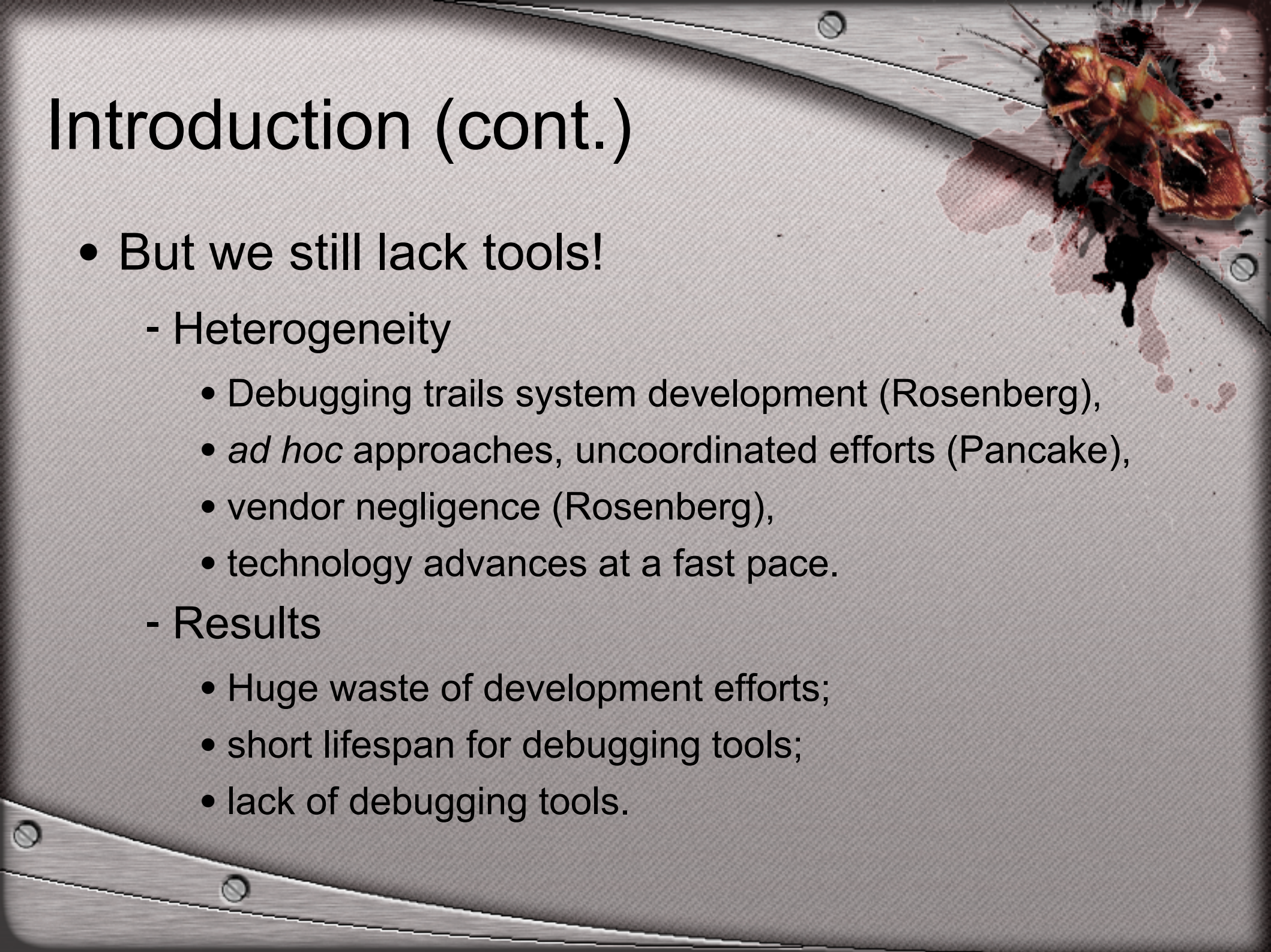
- Distributed Systems
  - Can be great
    - resource sharing;
    - scalability;
    - robustness.
  - However...
    - difficult to develop;
    - heterogeneity (accidental);
    - distributed executions (intrinsic).
  - Our focus: debugging.

Introduction (cont.)



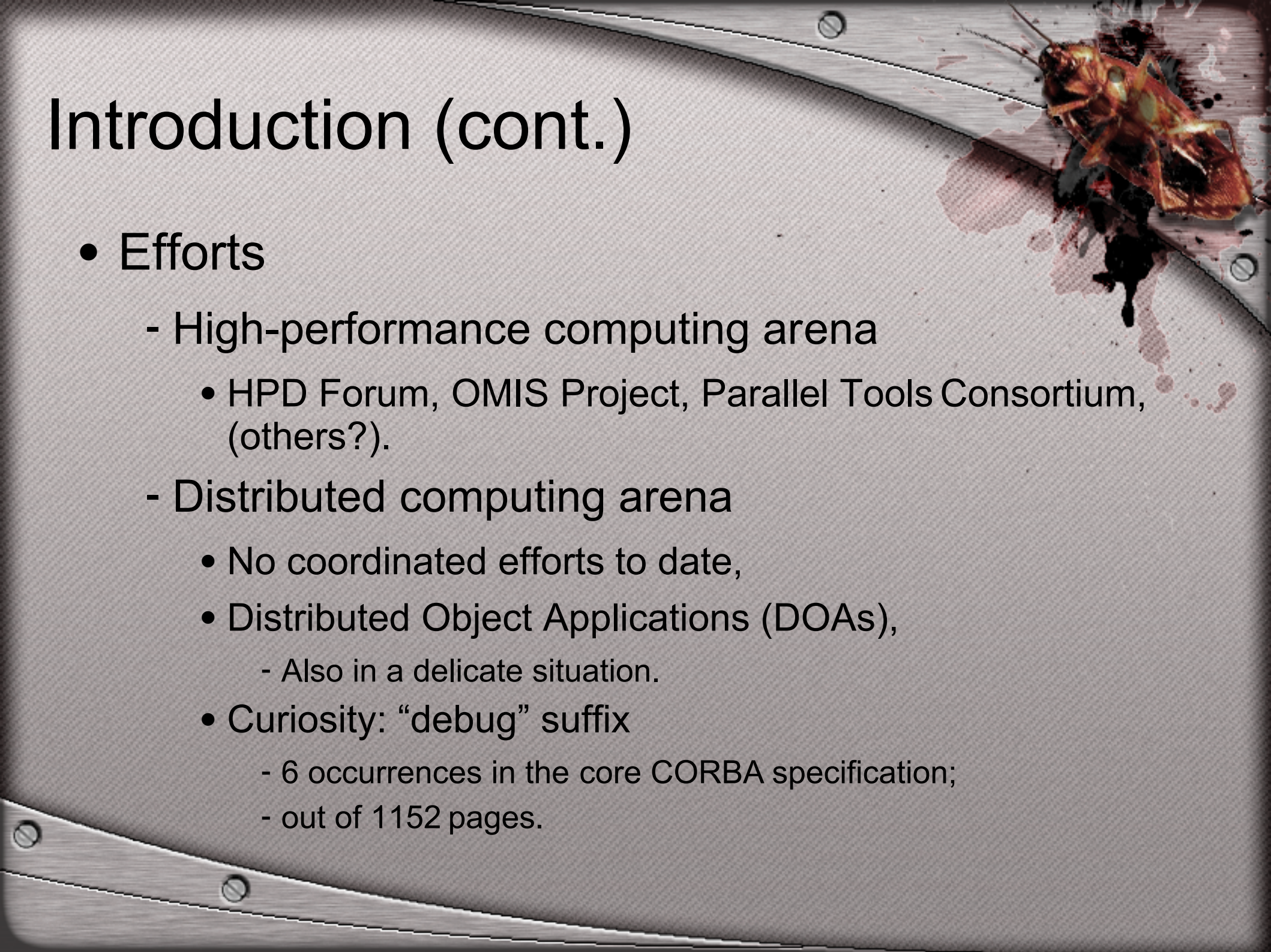
- Not surprisingly
  - Distributed debugging is difficult;
    - The four classical problems:
      - **Observability:** capture of global states;
      - **Non-determinism:** reproducibility of partially ordered executions;
      - **Probe effect:** probing code affects outcome;
      - **Maze effect:** it is difficult to present the data.
    - Are there solutions?
      - Logical clocks (e.g. Lamport);
      - Execution replay (e.g. RecPlay), reversible execution (e.g. ODB);
      - Hardware probes (e.g. FDR), multi-phase replay (e.g. DIOTA);
      - Algorithmic debugging, abstract visualization metaphors.

Introduction (cont.)



- But we still lack tools!
  - Heterogeneity
    - Debugging trails system development (Rosenberg),
    - *ad hoc* approaches, uncoordinated efforts (Pancake),
    - vendor negligence (Rosenberg),
    - technology advances at a fast pace.
  - Results
    - Huge waste of development efforts;
    - short lifespan for debugging tools;
    - lack of debugging tools.

Introduction (cont.)

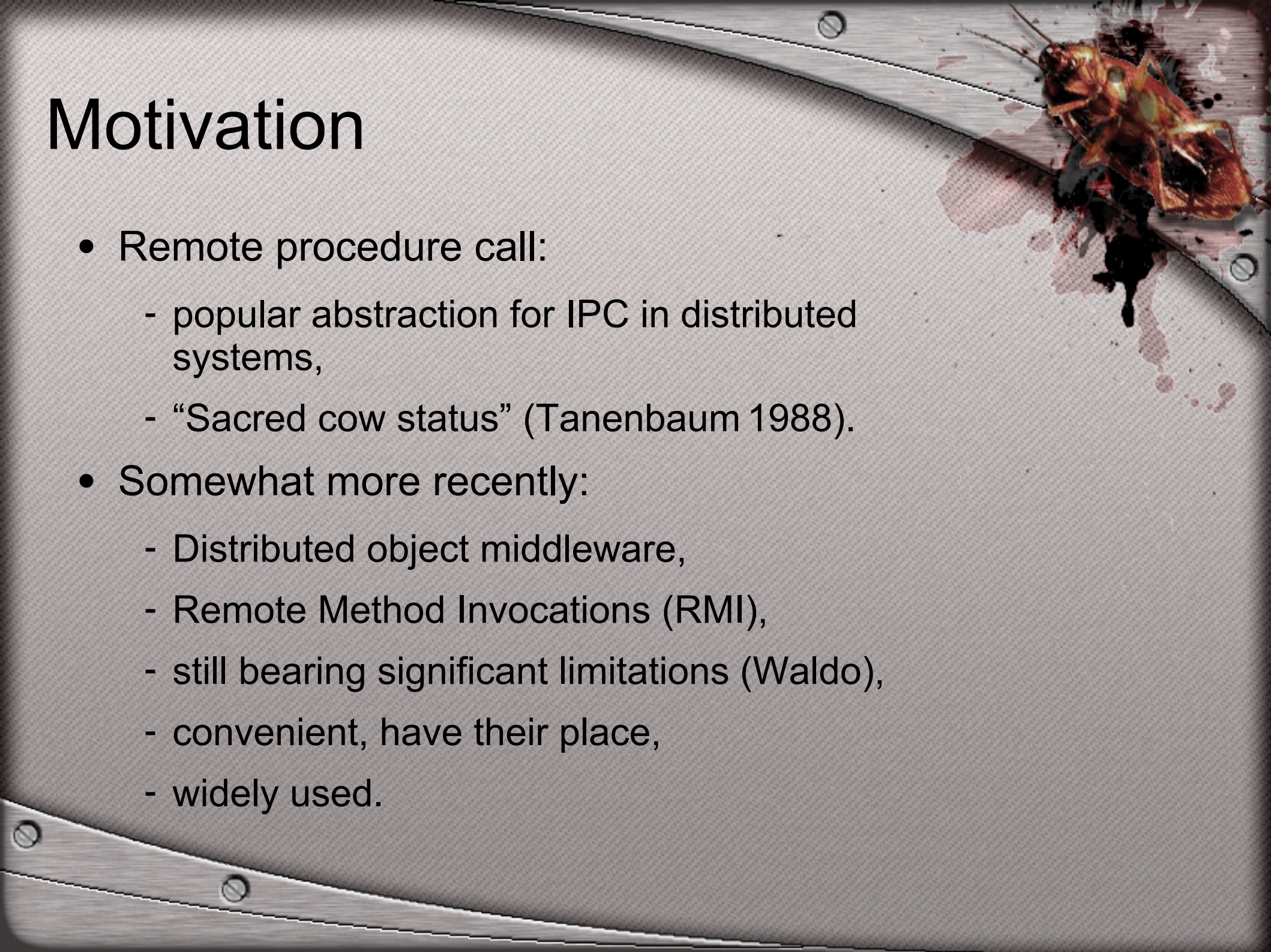


- Efforts
  - High-performance computing arena
    - HPD Forum, OMIS Project, Parallel Tools Consortium, (others?).
  - Distributed computing arena
    - No coordinated efforts to date,
    - Distributed Object Applications (DOAs),
      - Also in a delicate situation.
    - Curiosity: “debug” suffix
      - 6 occurrences in the core CORBA specification;
      - out of 1152 pages.

Objectives

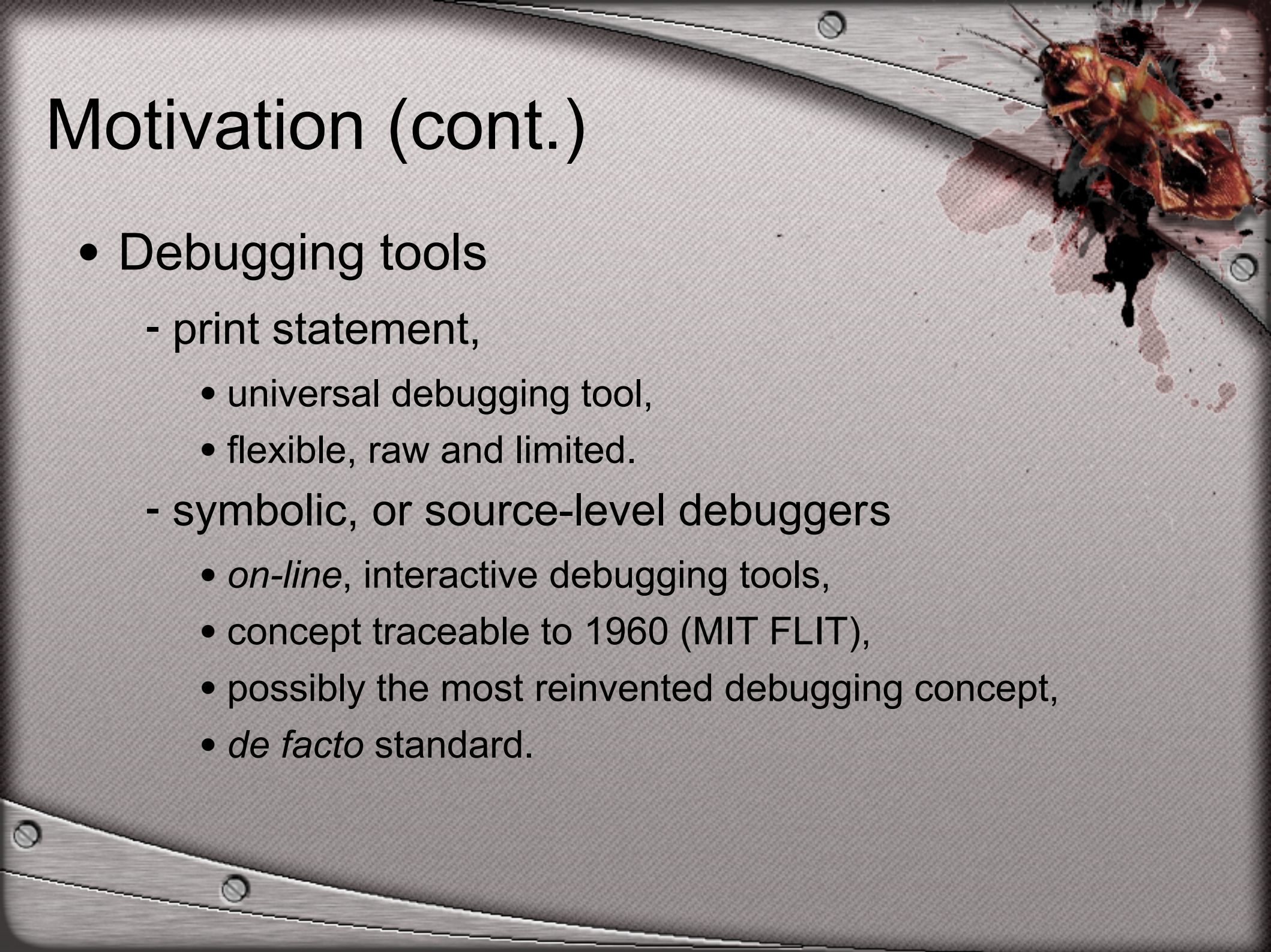
- Main objective of this work:
  - Changing the world and the way people think!
    - No single effort could do this.
  - Wasting efforts building yet another tool.
    - That will be born dead.
    - Oh wait, that's not it either.
  - Build a useful tool with what is available;
    - portable;
    - extensible;
    - simple.

Motivation



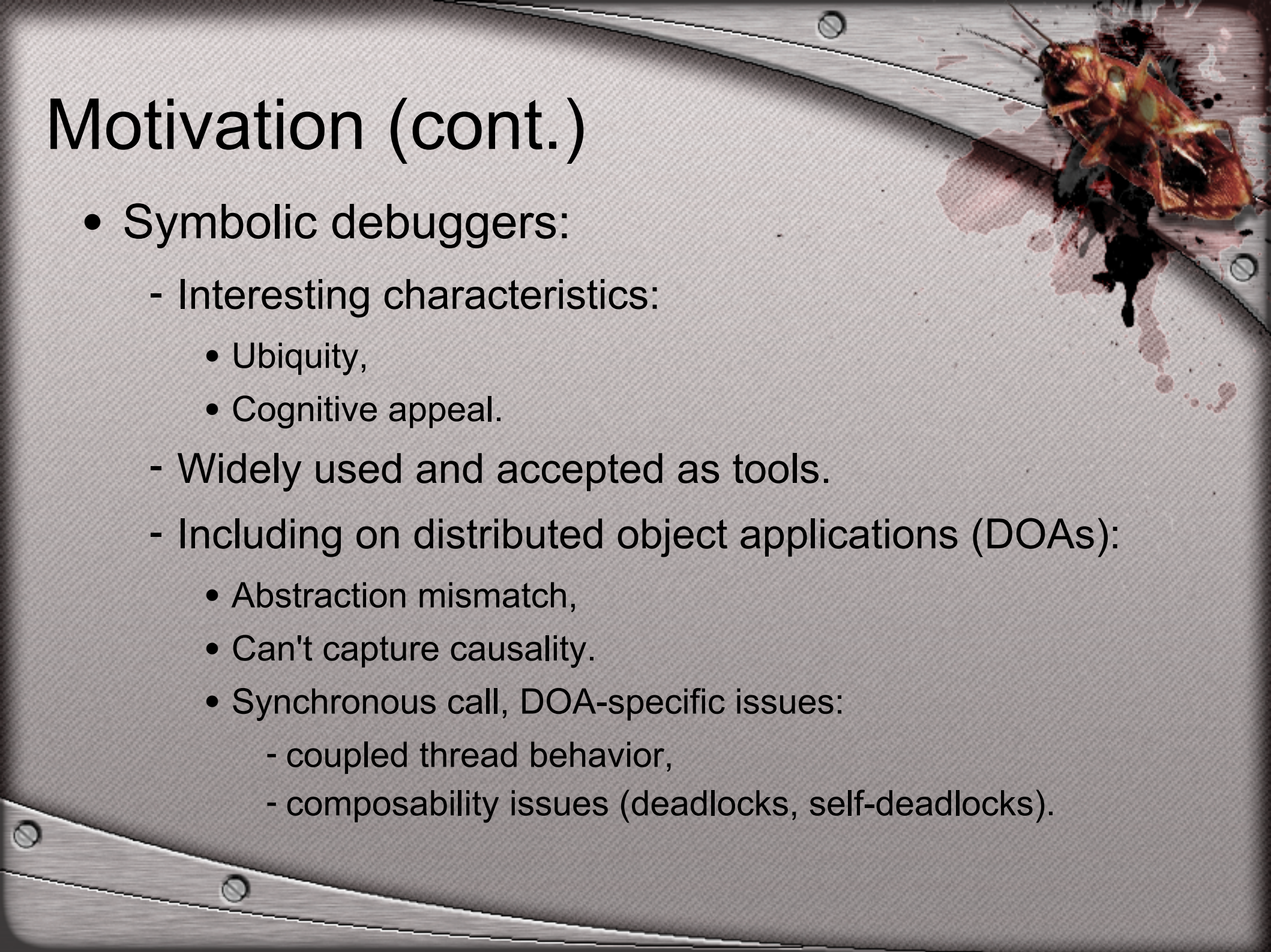
- Remote procedure call:
  - popular abstraction for IPC in distributed systems,
  - “Sacred cow status” (Tanenbaum 1988).
- Somewhat more recently:
  - Distributed object middleware,
  - Remote Method Invocations (RMI),
  - still bearing significant limitations (Waldo),
  - convenient, have their place,
  - widely used.

Motivation (cont.)



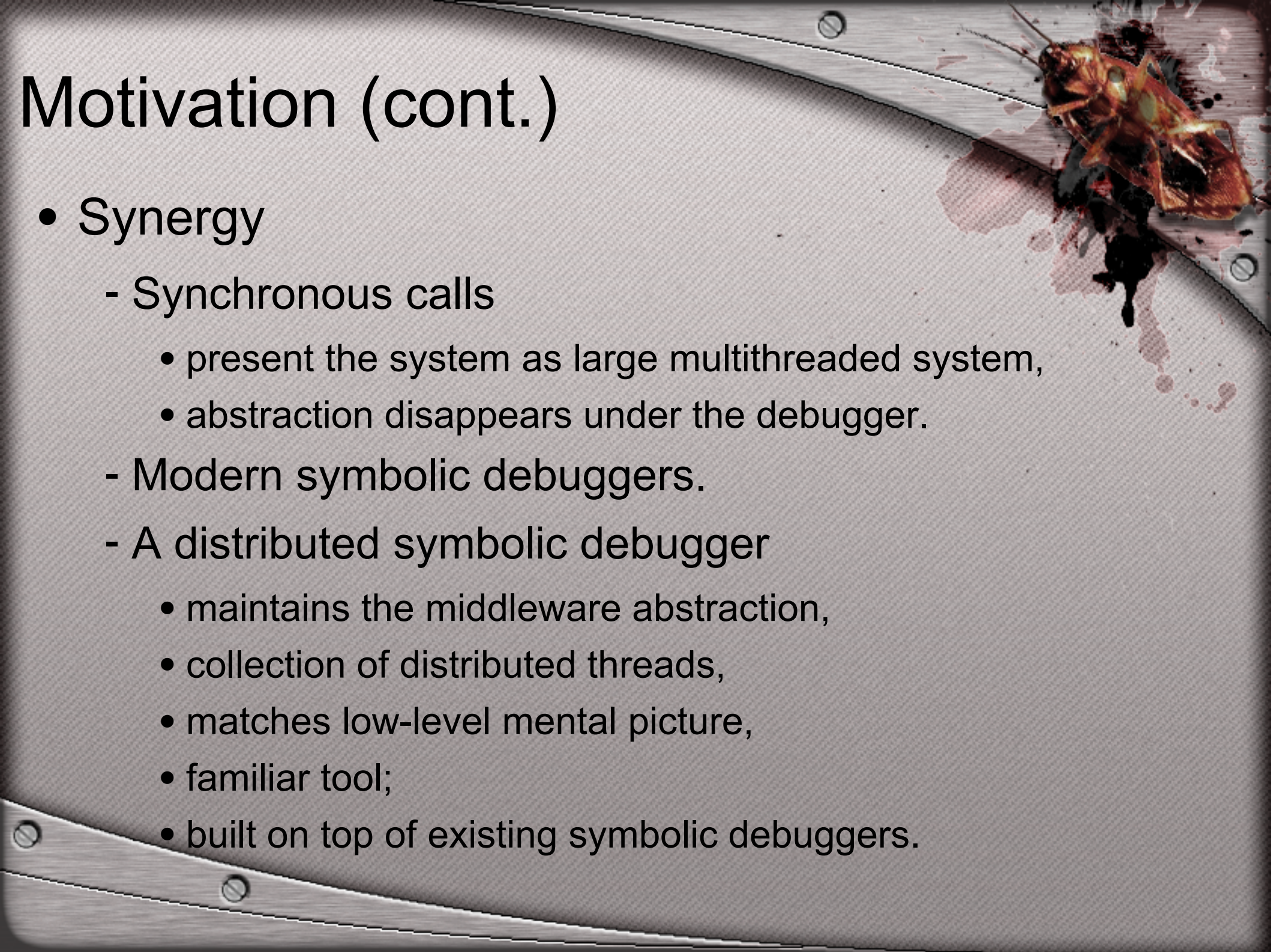
- Debugging tools
  - print statement,
    - universal debugging tool,
    - flexible, raw and limited.
  - symbolic, or source-level debuggers
    - *on-line*, interactive debugging tools,
    - concept traceable to 1960 (MIT FLIT),
    - possibly the most reinvented debugging concept,
    - *de facto* standard.

Motivation (cont.)



- Symbolic debuggers:
  - Interesting characteristics:
    - Ubiquity,
    - Cognitive appeal.
  - Widely used and accepted as tools.
  - Including on distributed object applications (DOAs):
    - Abstraction mismatch,
    - Can't capture causality.
    - Synchronous call, DOA-specific issues:
      - coupled thread behavior,
      - composability issues (deadlocks, self-deadlocks).

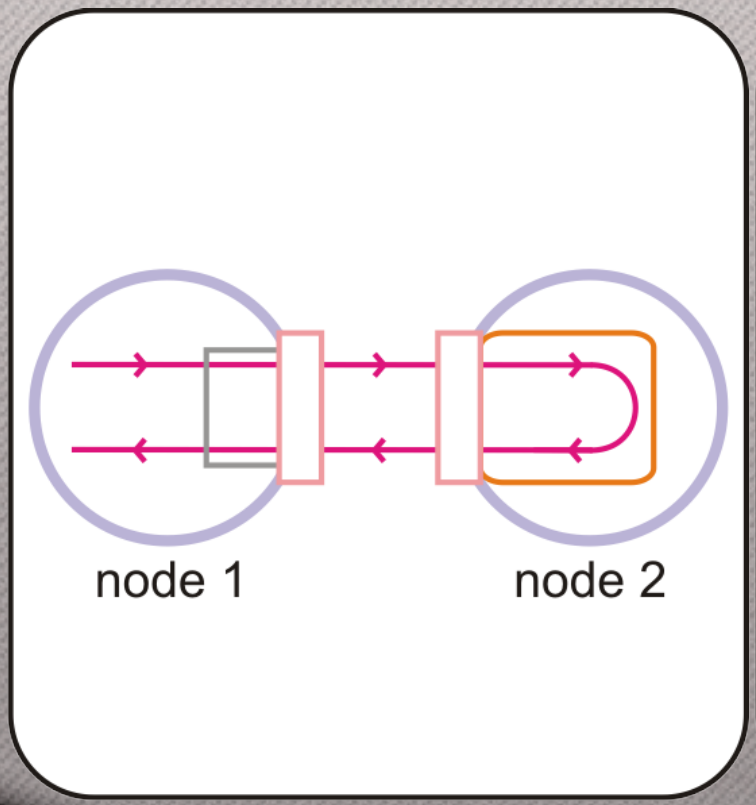
Motivation (cont.)

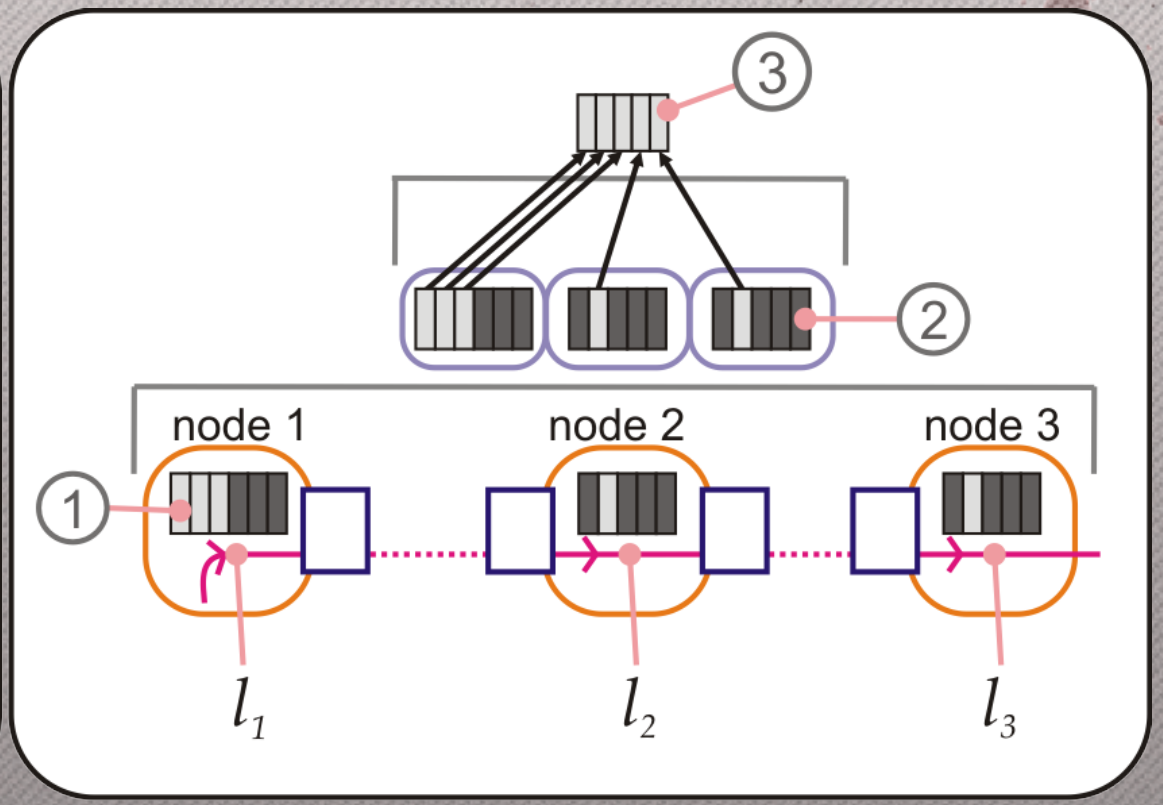


- Synergy
  - Synchronous calls
    - present the system as large multithreaded system,
    - abstraction disappears under the debugger.
  - Modern symbolic debuggers.
  - A distributed symbolic debugger
    - maintains the middleware abstraction,
    - collection of distributed threads,
    - matches low-level mental picture,
    - familiar tool;
    - built on top of existing symbolic debuggers.

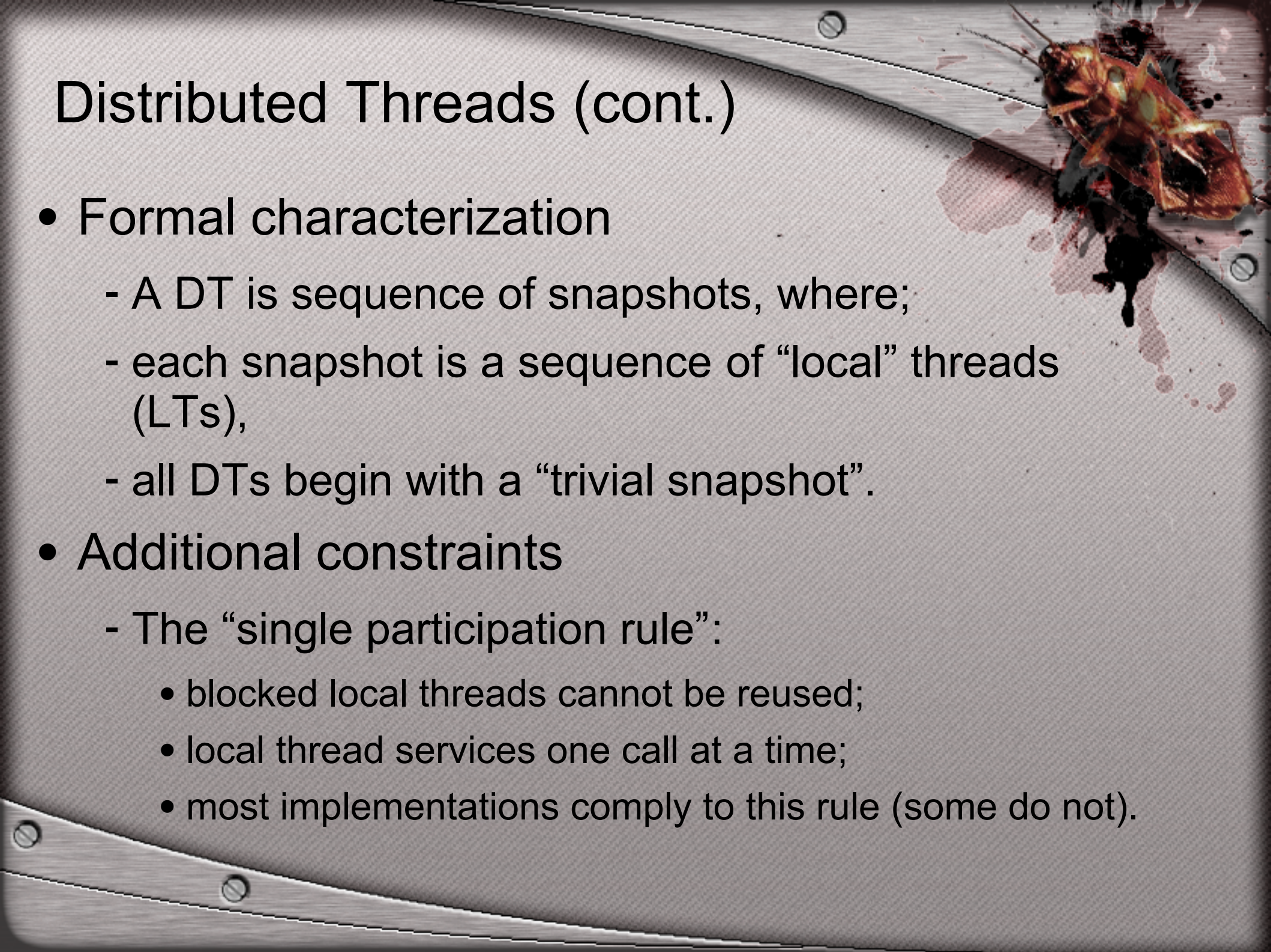
Distributed Threads

- Distributed thread (DT)
  - Virtual thread that crosses multiple nodes.





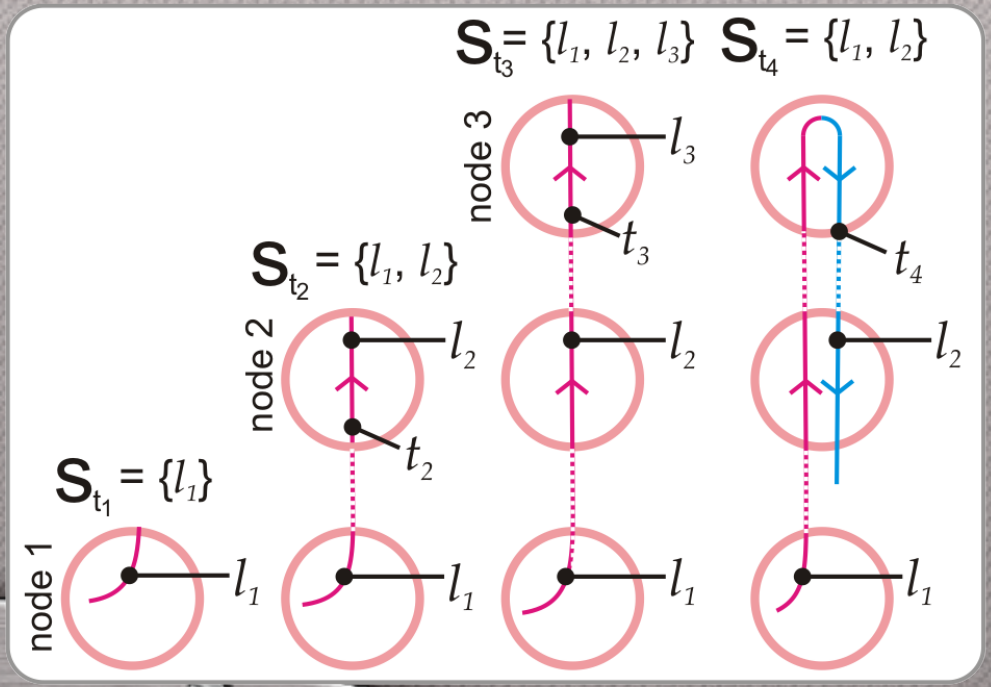
Distributed Threads (cont.)

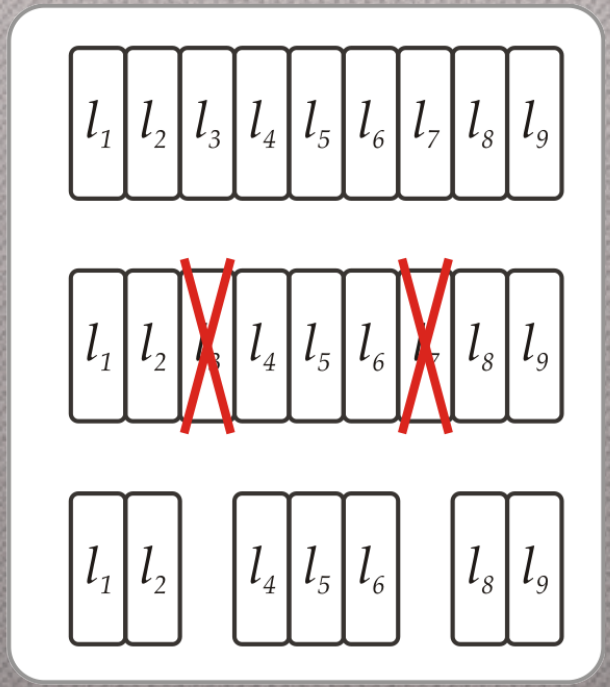


- Formal characterization
  - A DT is sequence of snapshots, where;
  - each snapshot is a sequence of “local” threads (LTs),
  - all DTs begin with a “trivial snapshot”.
- Additional constraints
  - The “single participation rule”:
    - blocked local threads cannot be reused;
    - local thread services one call at a time;
    - most implementations comply to this rule (some do not).

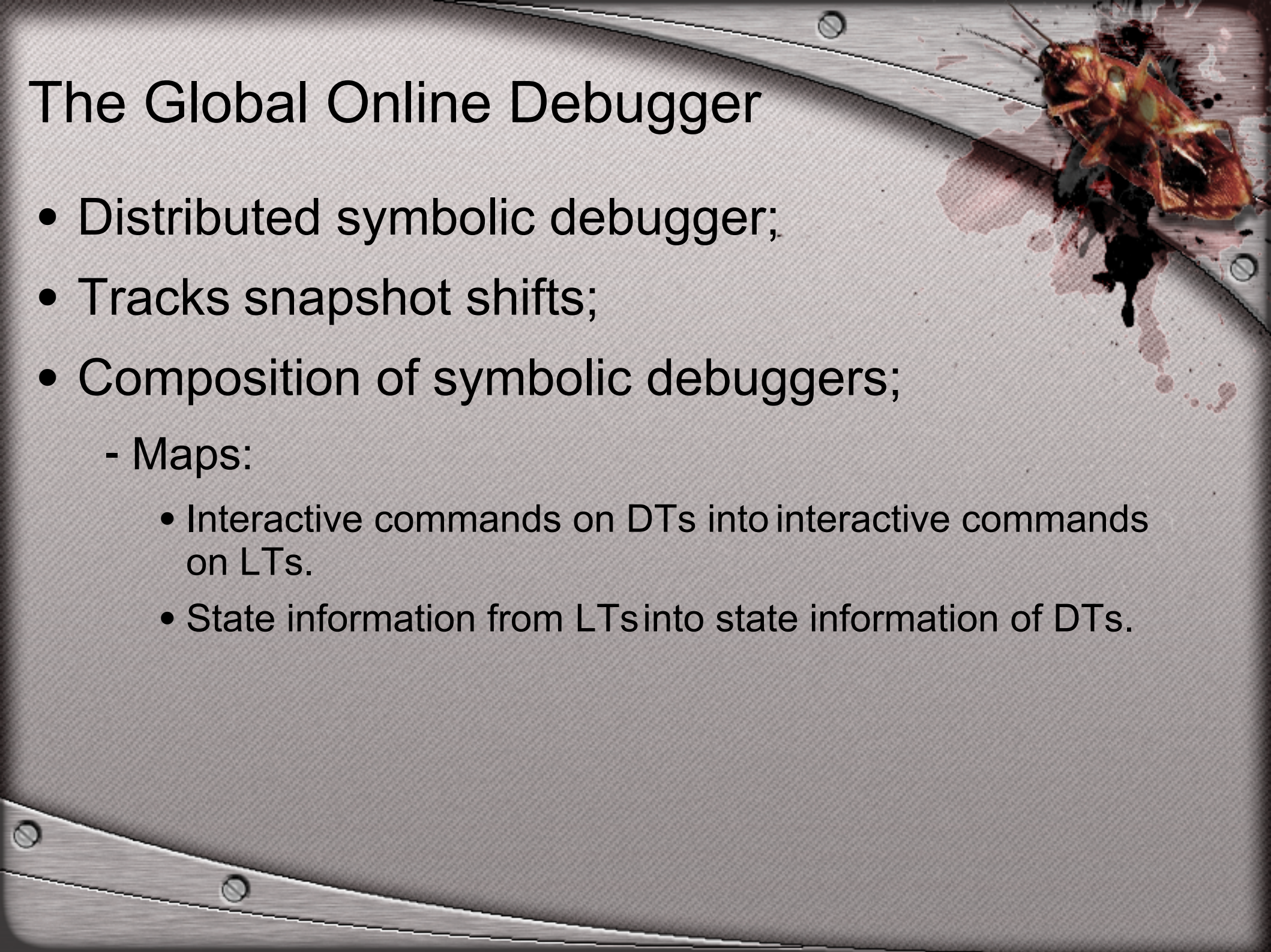
Distributed Threads (cont.)

- The current snapshot of a DT changes when:
  - A new remote call begins being handled;
  - some remote call returns;
  - some reply message is lost (breaks DT in two);
  - some node fails (breaks DT in possibly many pieces).



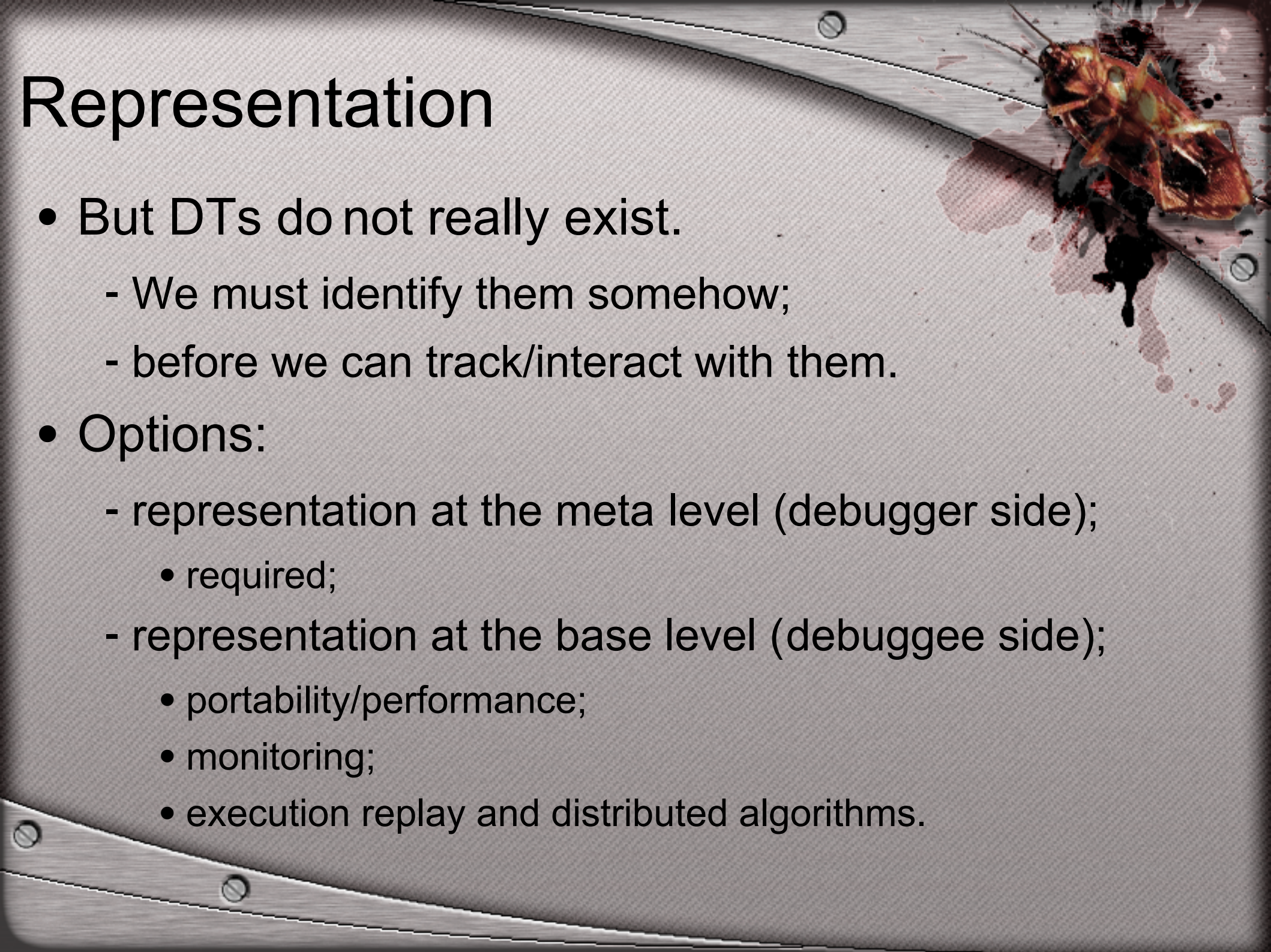


The Global Online Debugger



- Distributed symbolic debugger;
- Tracks snapshot shifts;
- Composition of symbolic debuggers;
  - Maps:
    - Interactive commands on DTs into interactive commands on LTs.
    - State information from LTs into state information of DTs.

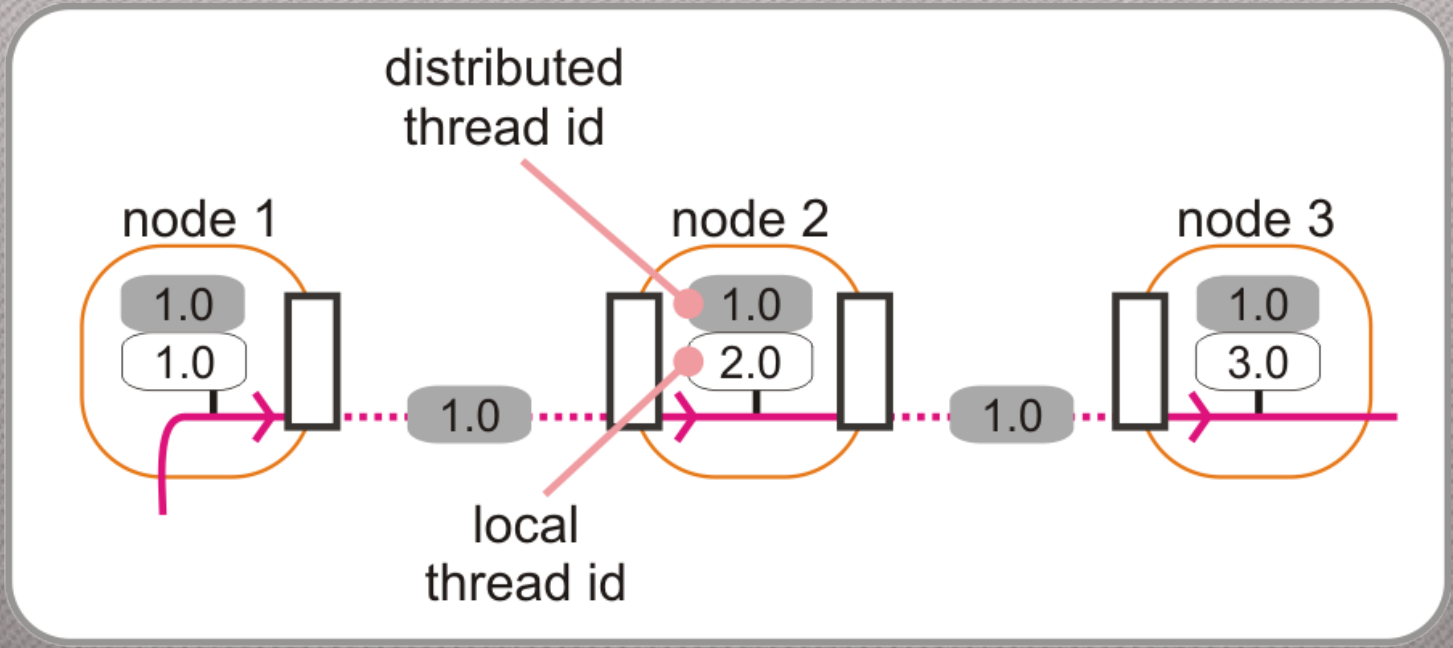
Representation



- But DTs do not really exist.
  - We must identify them somehow;
  - before we can track/interact with them.
- Options:
  - representation at the meta level (debugger side);
    - required;
  - representation at the base level (debuggee side);
    - portability/performance;
    - monitoring;
    - execution replay and distributed algorithms.

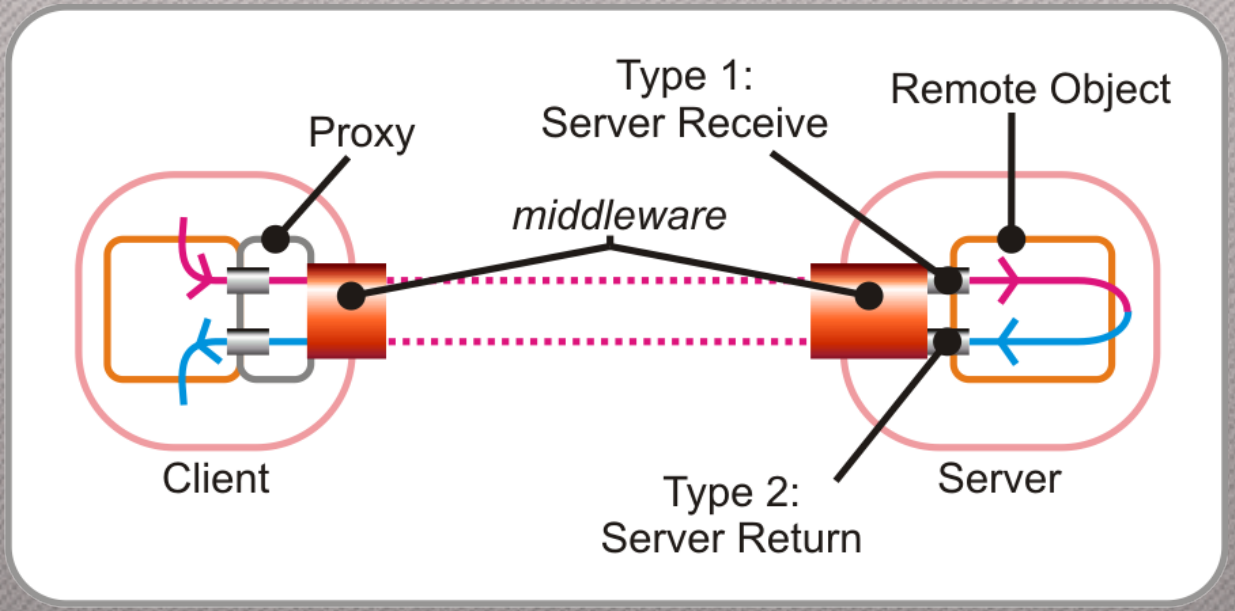
Representation (base-level)

- Each local thread has a unique id;
- first thread in the call chain propagates its id;
- *middleware requirement: passing of context information (metadata) with each request.



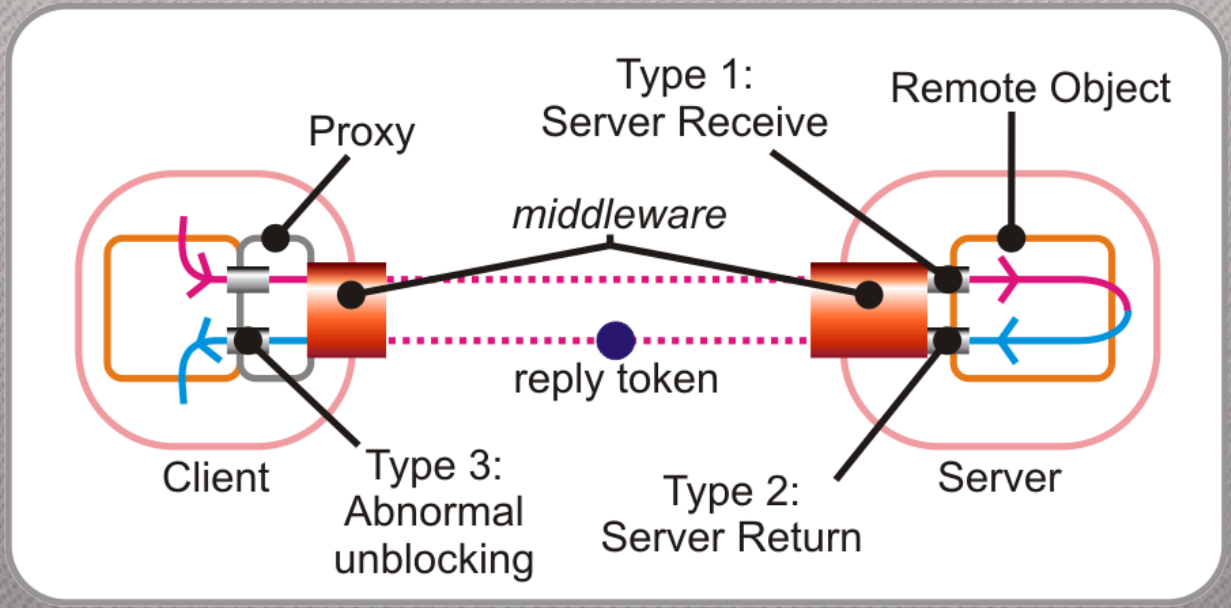
Tracking

- Normal shifts:
  - Two types of events: server receive, server return.



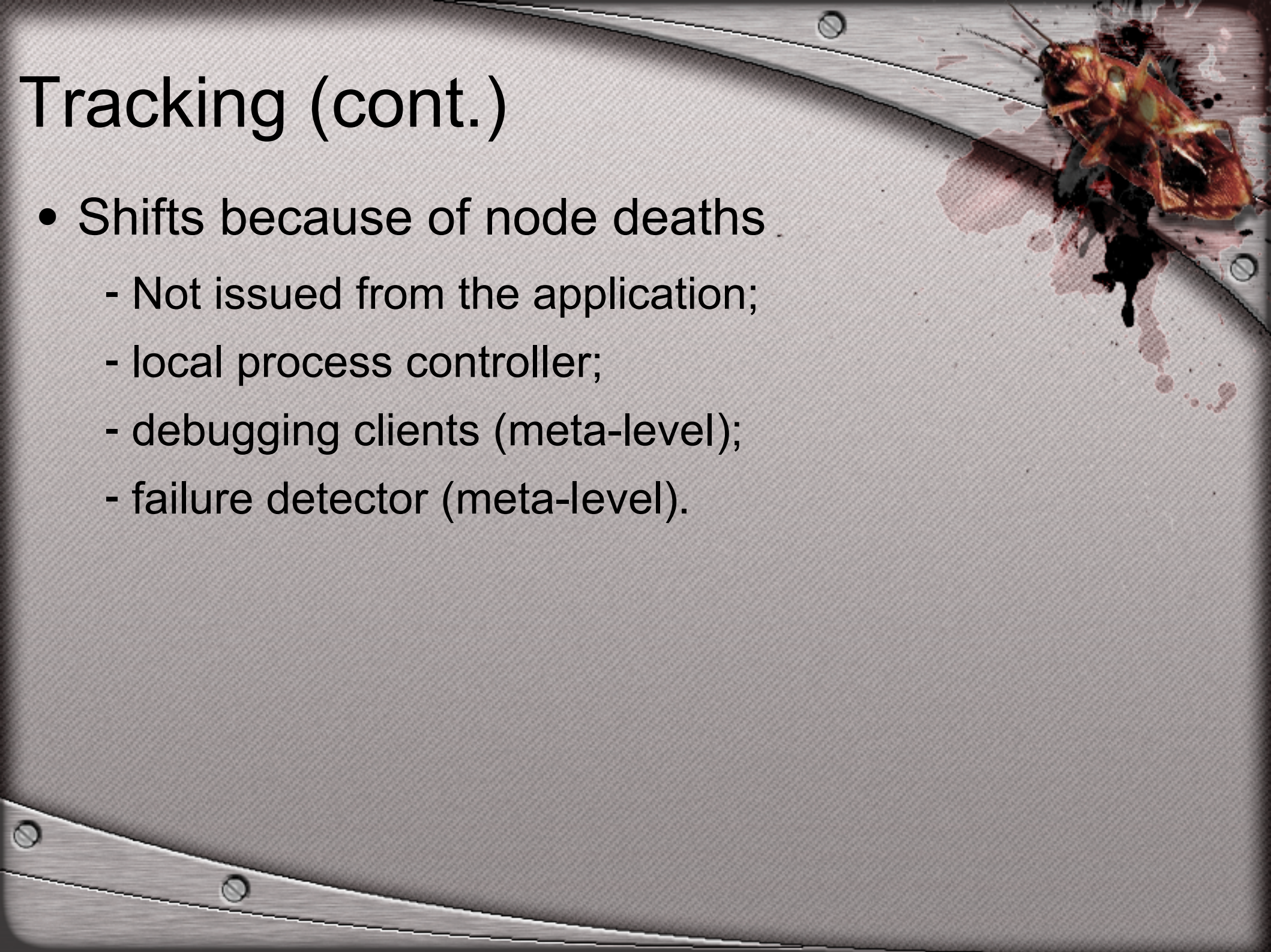
Tracking (cont.)

- Shifts because of lost reply messages:
  - Client detects “abnormal unblockings”:
    - blocked local thread returns;
    - no reply token = no reply message (or middleware bug).



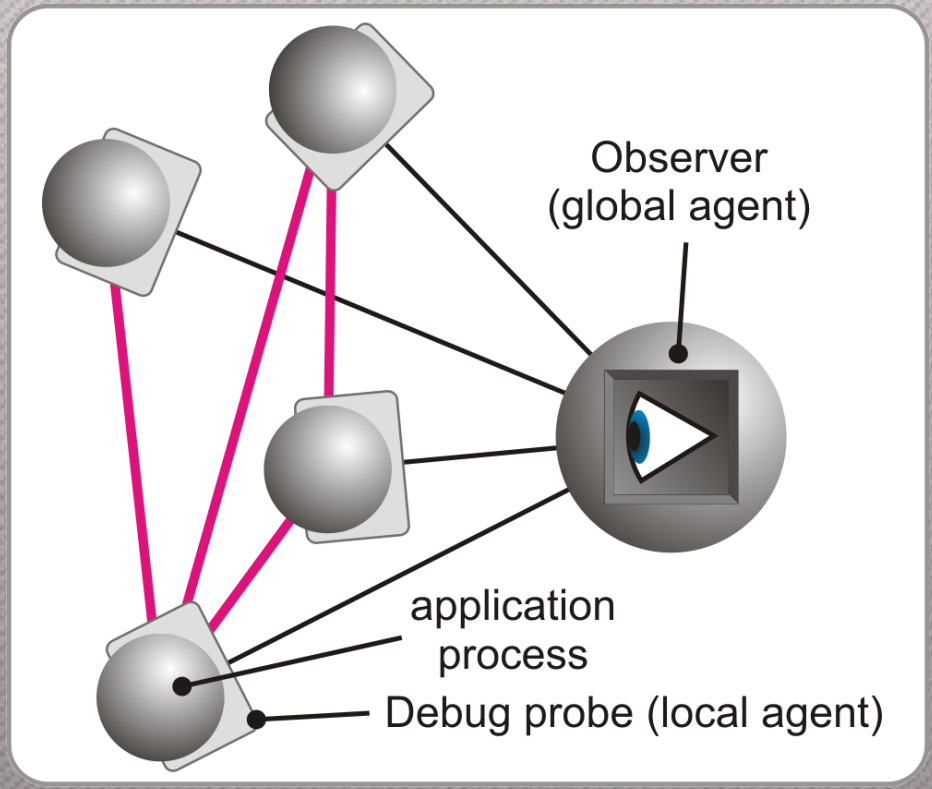
Tracking (cont.)

- Shifts because of node deaths
  - Not issued from the application;
  - local process controller;
  - debugging clients (meta-level);
  - failure detector (meta-level).

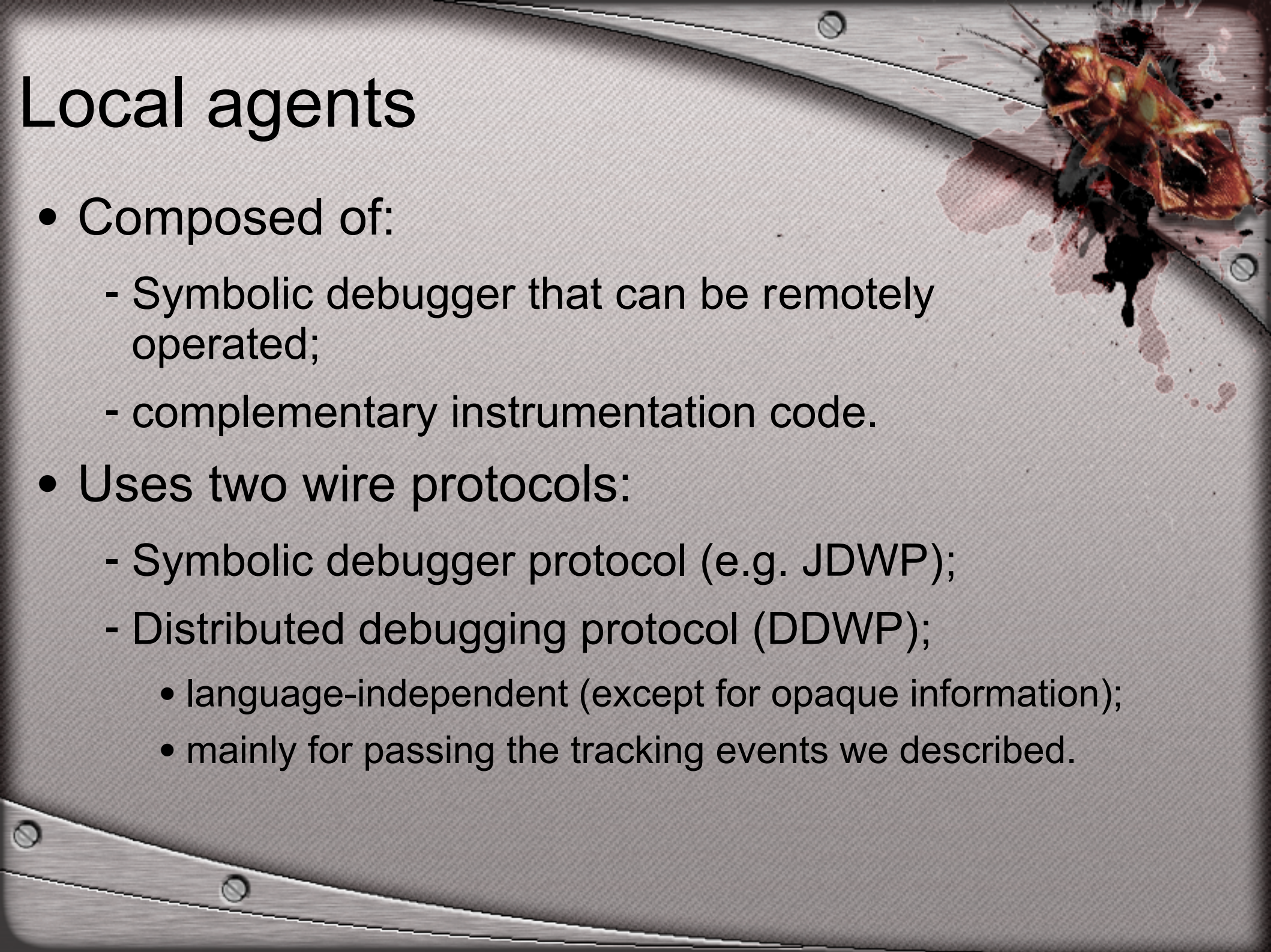


The Global On-line Debugger

- Architectural overview of the G.O.D.:



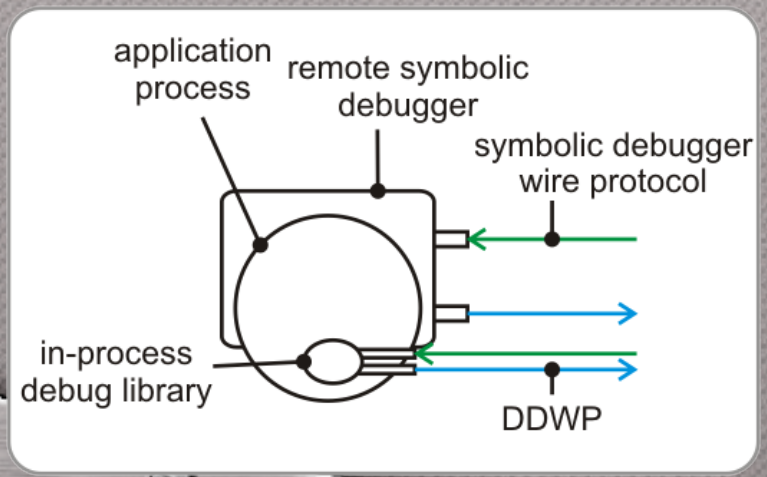
Local agents

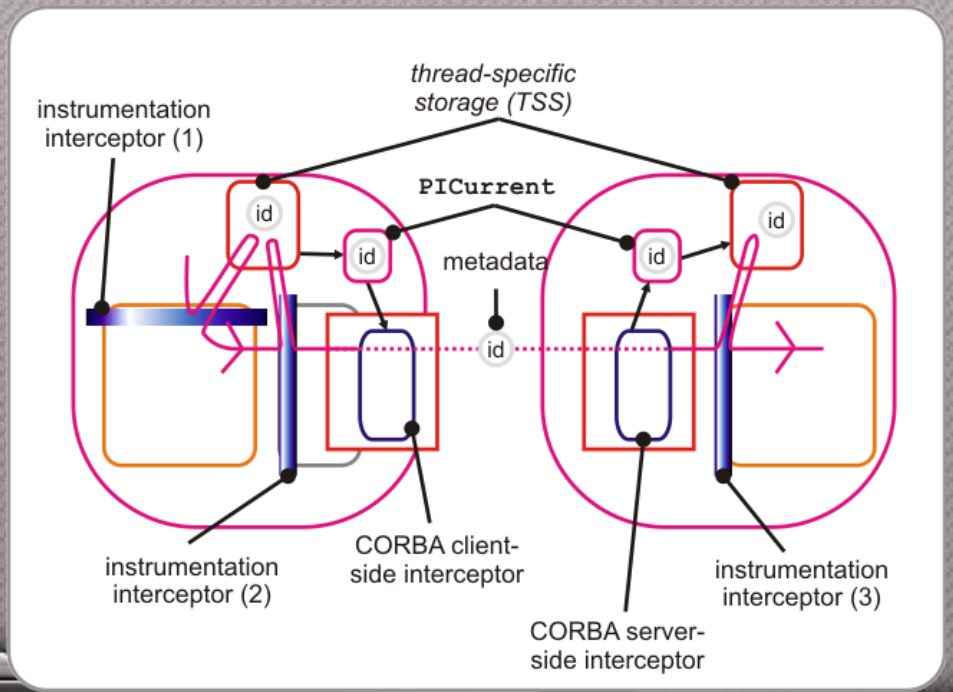


- Composed of:
  - Symbolic debugger that can be remotely operated;
  - complementary instrumentation code.
- Uses two wire protocols:
  - Symbolic debugger protocol (e.g. JDWP);
  - Distributed debugging protocol (DDWP);
    - language-independent (except for opaque information);
    - mainly for passing the tracking events we described.

Java/CORBA local agents

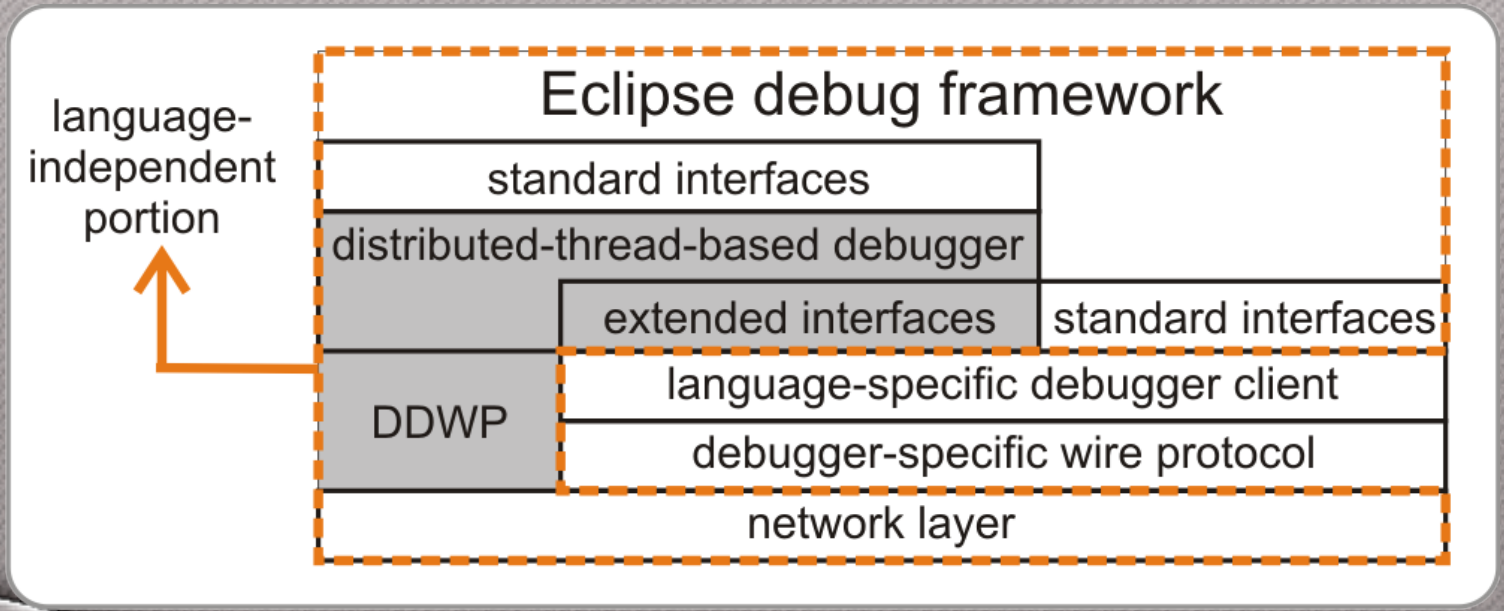
- Somewhat straightforward;
  - dynamic instrumentation (BCEL);
    - interceptors added at each **`Runnable.run()`**;
    - at each proxy (tracking, id passing);
    - at each remote object (tracking, id passing, tokens);
    - interceptors call debug library;
    - ids passed by combining TSS and service contexts.



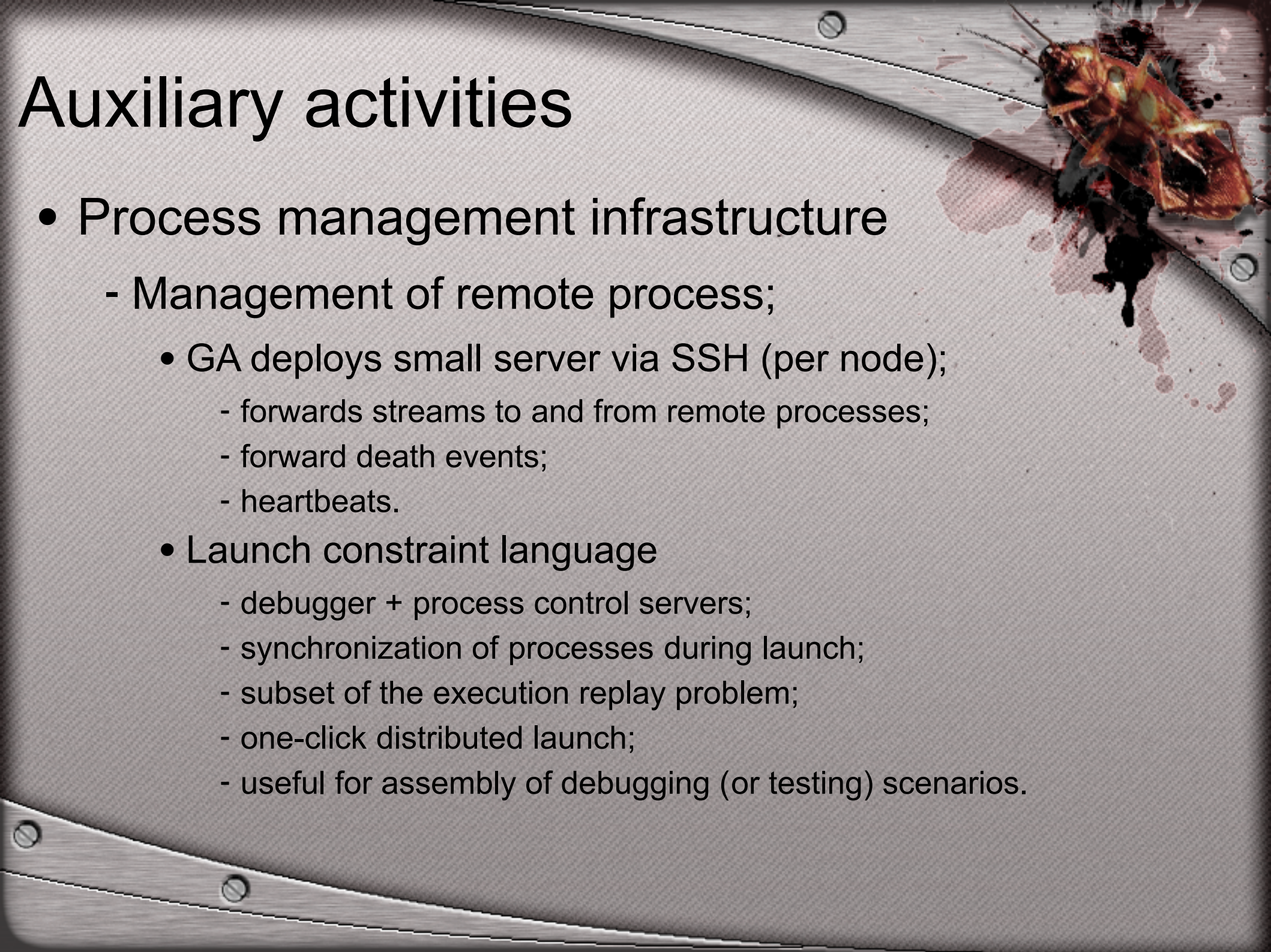


Global agent (GA)

- “Composite debugger”;
  - based on the generic Eclipse meta-model;
  - language-agnostic object-model for reifying runtime entities;
  - reification of a virtual process, based on distributed threads.

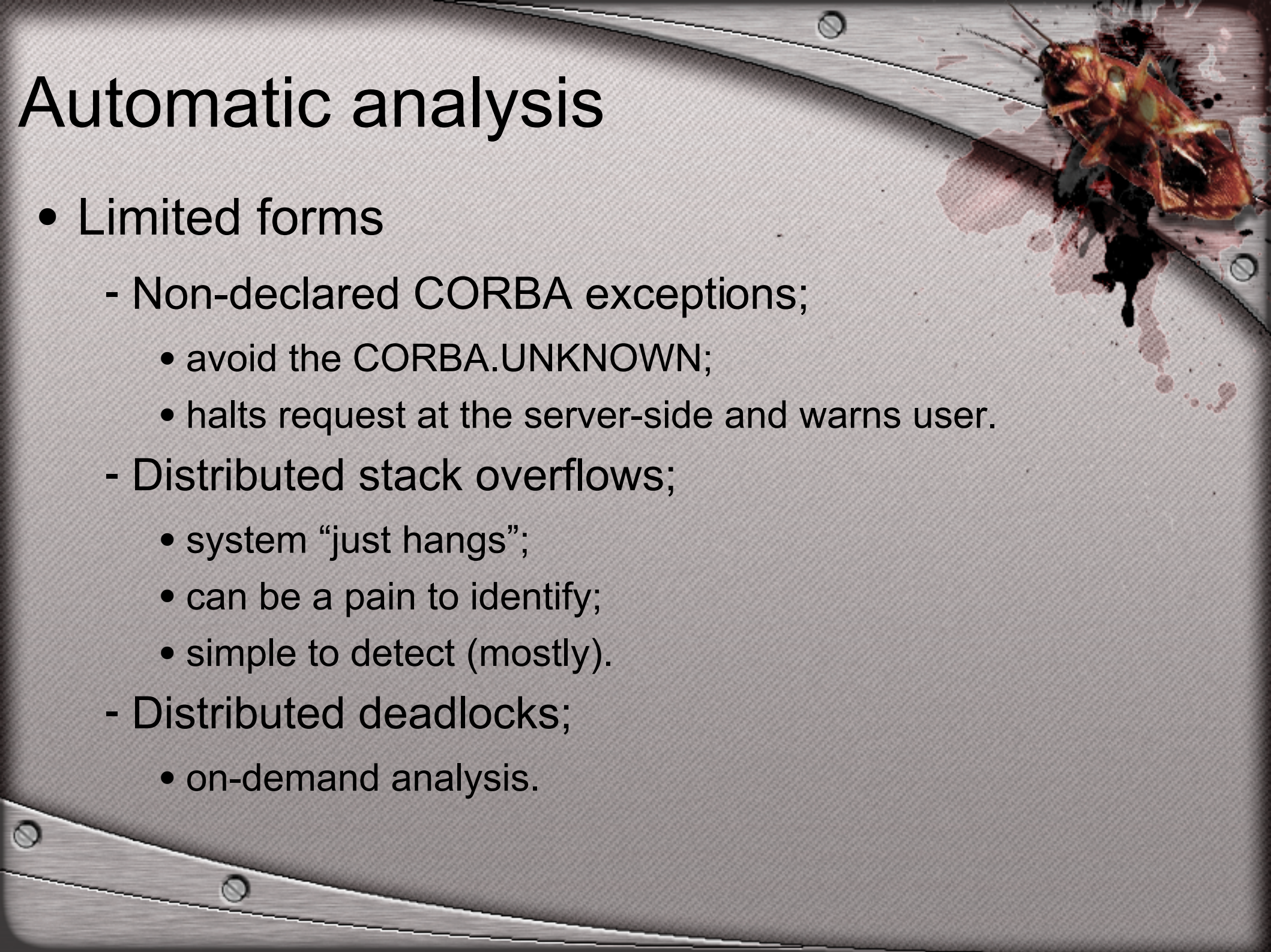


Auxiliary activities



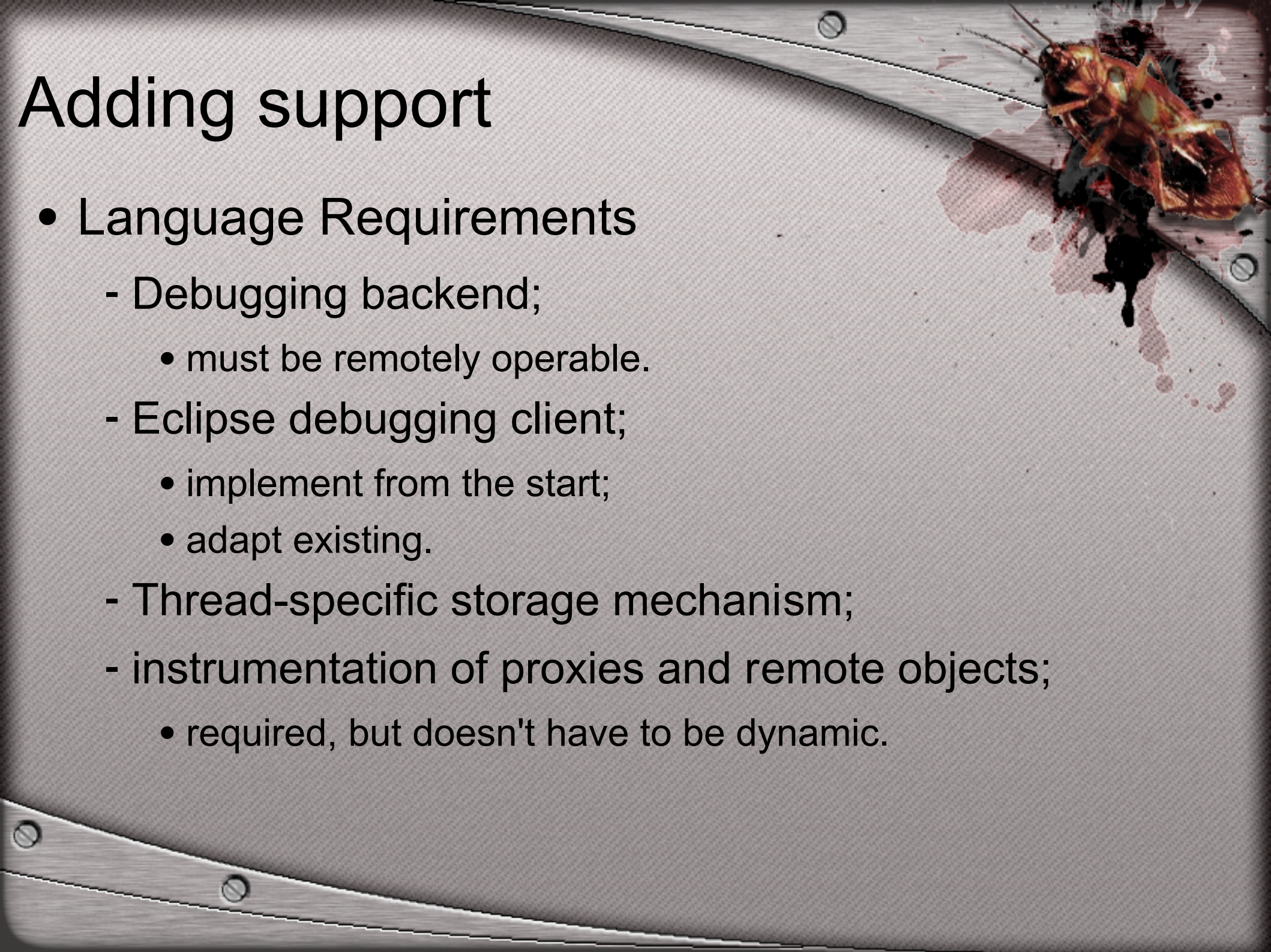
- Process management infrastructure
  - Management of remote process;
    - GA deploys small server via SSH (per node);
      - forwards streams to and from remote processes;
      - forward death events;
      - heartbeats.
    - Launch constraint language
      - debugger + process control servers;
      - synchronization of processes during launch;
      - subset of the execution replay problem;
      - one-click distributed launch;
      - useful for assembly of debugging (or testing) scenarios.

Automatic analysis



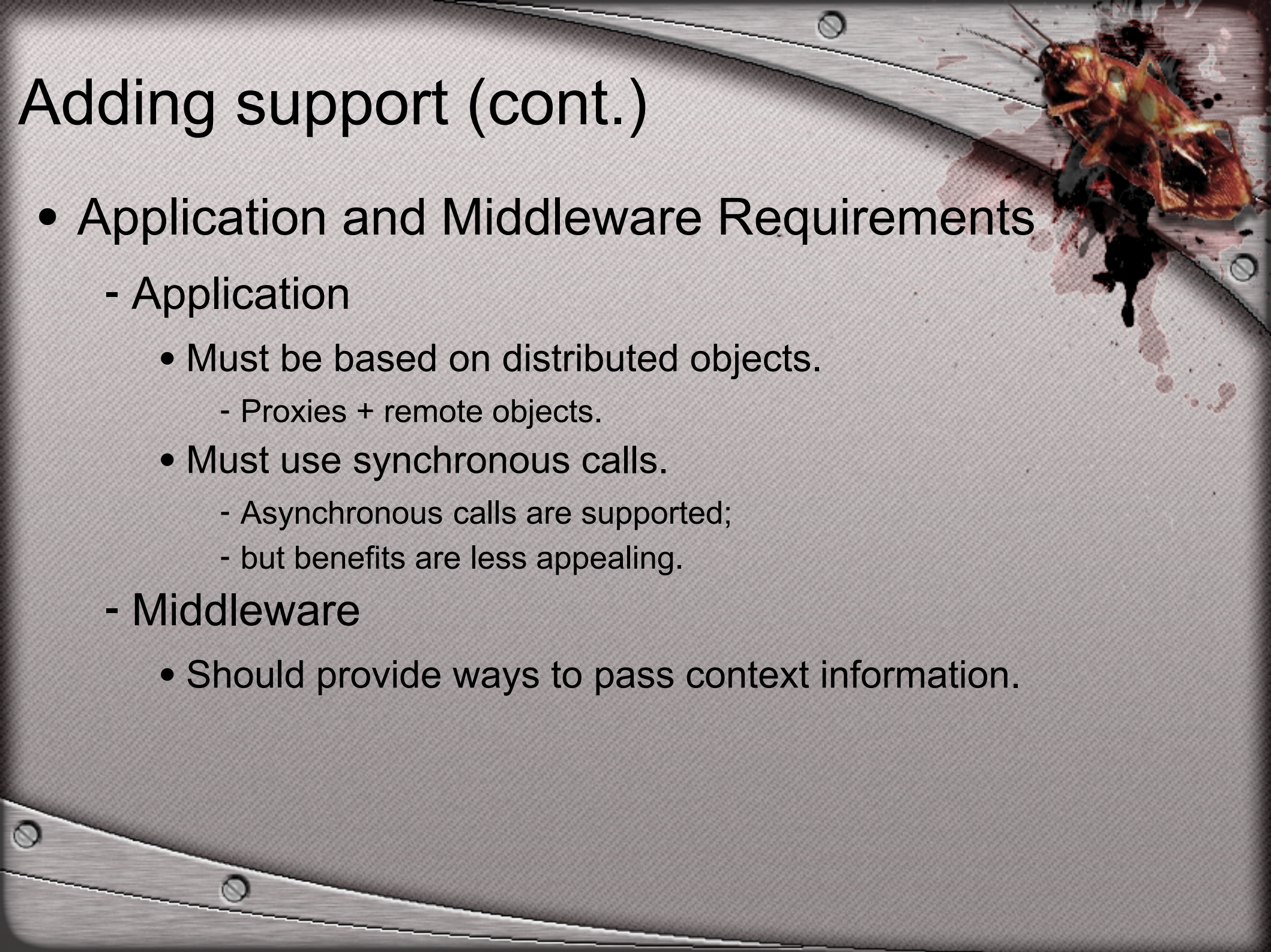
- Limited forms
  - Non-declared CORBA exceptions;
    - avoid the CORBA.UNKNOWN;
    - halts request at the server-side and warns user.
  - Distributed stack overflows;
    - system “just hangs”;
    - can be a pain to identify;
    - simple to detect (mostly).
  - Distributed deadlocks;
    - on-demand analysis.

Adding support



- Language Requirements
  - Debugging backend;
    - must be remotely operable.
  - Eclipse debugging client;
    - implement from the start;
    - adapt existing.
  - Thread-specific storage mechanism;
  - instrumentation of proxies and remote objects;
    - required, but doesn't have to be dynamic.

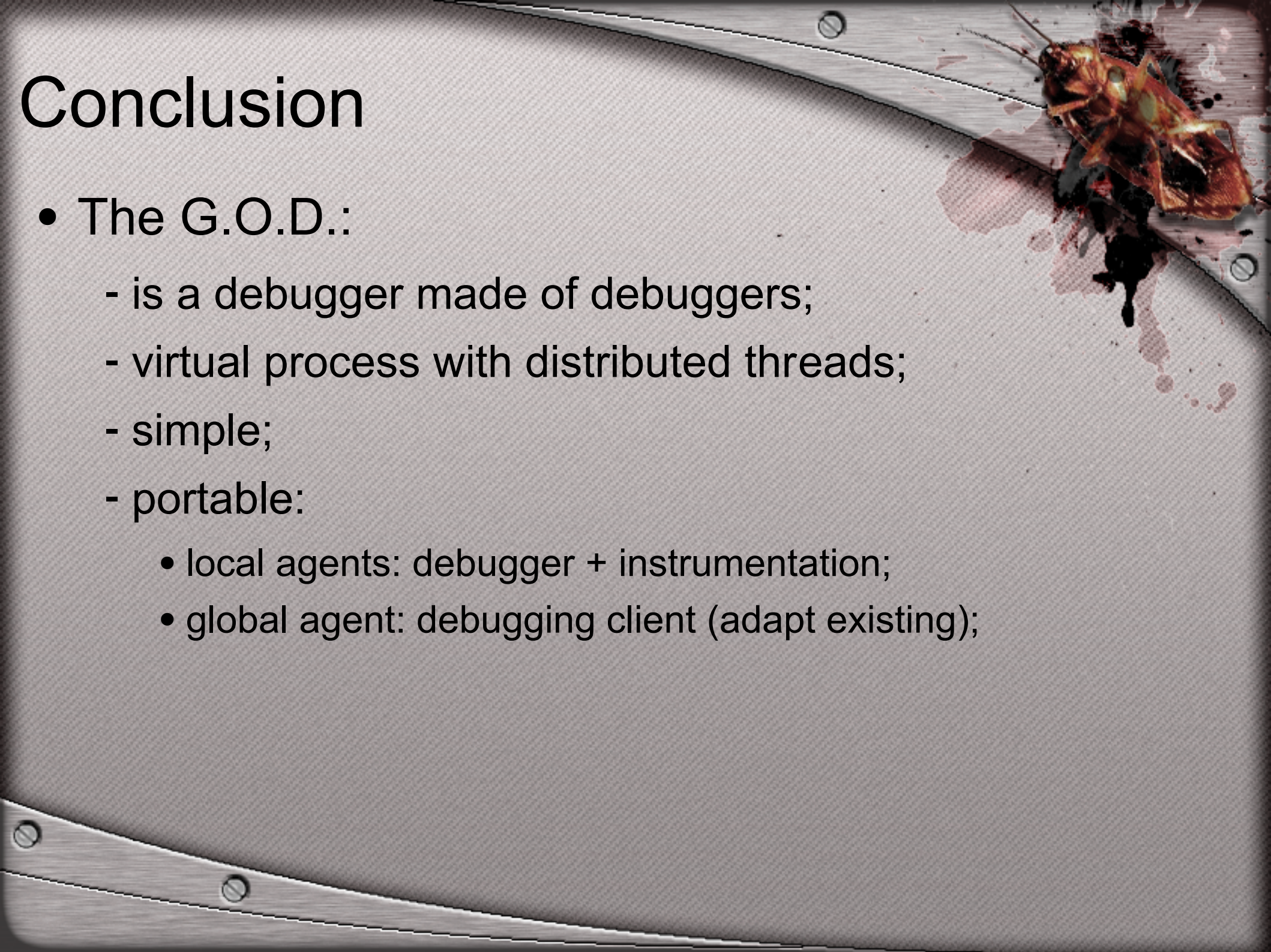
Adding support (cont.)



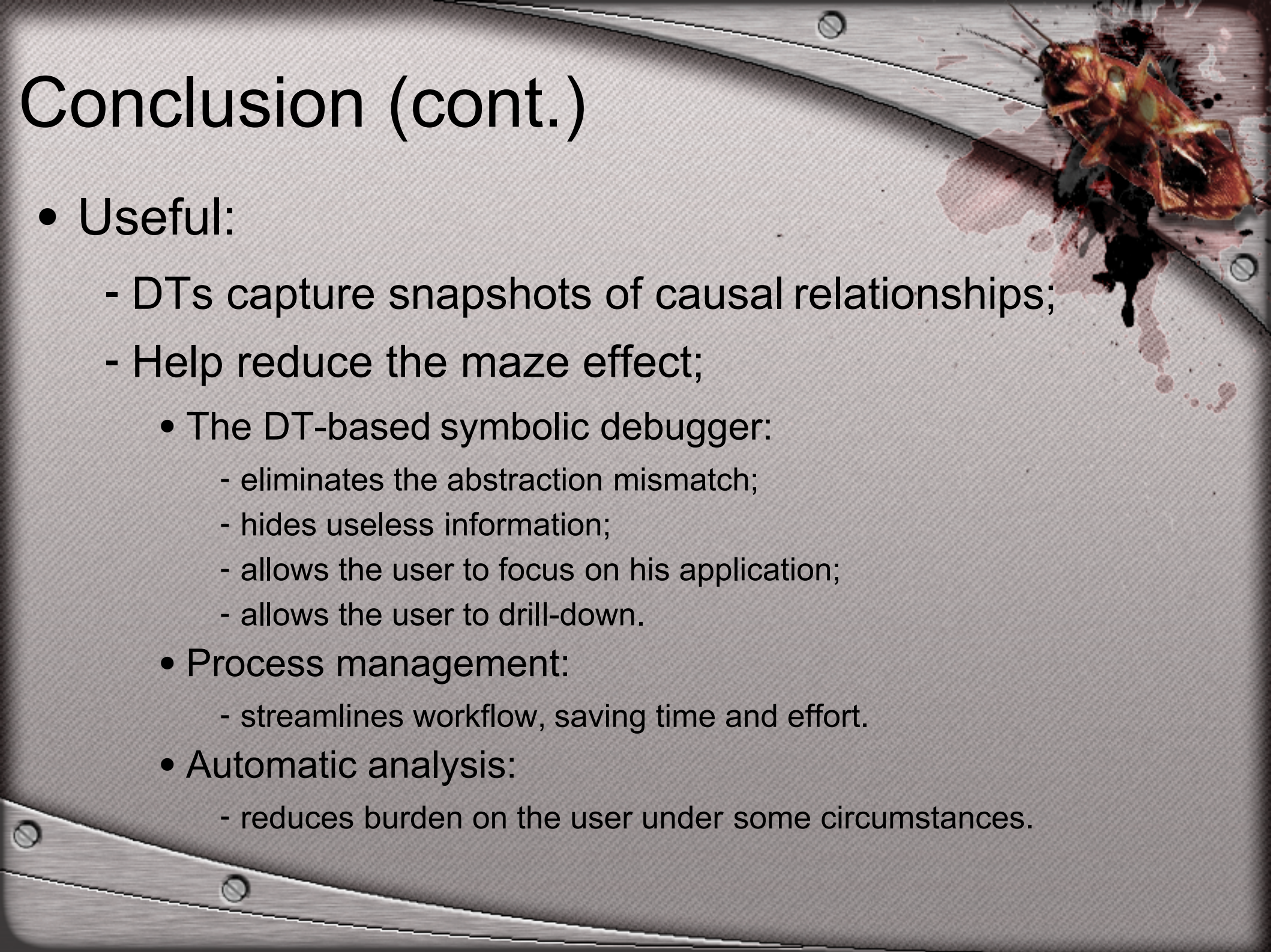
- Application and Middleware Requirements
  - Application
    - Must be based on distributed objects.
      - Proxies + remote objects.
    - Must use synchronous calls.
      - Asynchronous calls are supported;
      - but benefits are less appealing.
  - Middleware
    - Should provide ways to pass context information.

Conclusion

- The G.O.D.:
  - is a debugger made of debuggers;
  - virtual process with distributed threads;
  - simple;
  - portable:
    - local agents: debugger + instrumentation;
    - global agent: debugging client (adapt existing);



Conclusion (cont.)



- Useful:
  - DTs capture snapshots of causal relationships;
  - Help reduce the maze effect;
    - The DT-based symbolic debugger:
      - eliminates the abstraction mismatch;
      - hides useless information;
      - allows the user to focus on his application;
      - allows the user to drill-down.
    - Process management:
      - streamlines workflow, saving time and effort.
    - Automatic analysis:
      - reduces burden on the user under some circumstances.

Future work

- Porting.
- Classical problems:
  - we barely scratched them;
  - visualization:
    - more scalable mechanisms;
    - more automatic analysis mechanisms;
    - DT are too low-level (maze of DTs comes fast).
  - probe effect and non-determinism:
    - too difficult to tackle in a portable manner;
    - threads + shared memory.

Thank you!

- Help is appreciated!
- Source-code is available under the EPL.
- Project web site (there's a screencast there):
  - http://god.incubadora.fapesp.br
- Google and eclipse.org:
  - Summer of Code Grant
- IBM:
  - Eclipse Innovation Grant