Evaluating Software Model Checking Tools

(It's a Dirty Job, But Someone's Got to Do It)

George Avrunin

Department of Mathematics
University of Massachusetts

The Need for Evaluation

- Different finite-state verification tools, based on different approaches

The Need for Evaluation

- Different finite-state verification tools, based on different approaches
- Performance varies significantly on different problems/properties
  - we can't predict the performance of the tools very well

The Need for Evaluation

- Different finite-state verification tools, based on different approaches
- Performance varies significantly on different problems/properties
  - we can't predict the performance of the tools very well
- So we can't provide much guidance to users about
  - which FSV tool to apply, or even
  - whether FSV will be useful for their particular problems

The Need for Evaluation

- Different finite-state verification tools, based on different approaches
- Performance varies significantly on different problems/properties
  - we can't predict the performance of the tools very well
- So we can't provide much guidance to users about
  - which FSV tool to apply, or even
  - whether FSV will be useful for their particular problems
- Evaluation also suggests improvements in tools, new research directions, etc.

Analytic Comparisons Are Not Enough

- Underlying algorithms are not well-understood
  - We don't know very much about predicting the size of OBDDs, or the time required for integer programming.

Analytic Comparisons Are Not Enough

- Underlying algorithms are not well-understood
  - We don't know very much about predicting the size of OBDDs, or the time required for integer programming.
- Need evaluations of *tools*, not just approaches.

Analytic Comparisons Are Not Enough

- Underlying algorithms are not well-understood
  - We don't know very much about predicting the size of OBDDs, or the time required for integer programming.
- Need evaluations of *tools*, not just approaches.
- Need evaluations on *typical* problems.
  - (domain-specific) benchmarks

Analytic Comparisons Are Not Enough

- Underlying algorithms are not well-understood
  - We don't know very much about predicting the size of OBDDs, or the time required for integer programming.
- Need evaluations of *tools*, not just approaches.
- Need evaluations on *typical* problems.
  - (domain-specific) benchmarks

So, while analytical comparisons would be extremely useful, we need to do empirical evaluation.

But Empirical Comparisons Are Hard . . .

- Tools have different formalisms, possibly with different expressive power, for describing the system . . .
  - Are the different tools really analyzing the same system?

But Empirical Comparisons Are Hard . . .

- Tools have different formalisms, possibly with different expressive power, for describing the system . . .
  - Are the different tools really analyzing the same system?
- and different formalisms for specifying the property.
  - Are they really checking the same property?

But Empirical Comparisons Are Hard . . .

- Tools have different formalisms, possibly with different expressive power, for describing the system . . .
  - Are the different tools really analyzing the same system?
- and different formalisms for specifying the property.
  - Are they really checking the same property?
- Effective use of even a single tool on a system of realistic size and complexity is a major project
  - need significant expertise with the tool

But Empirical Comparisons Are Hard . . .

- Tools have different formalisms, possibly with different expressive power, for describing the system . . .
  - Are the different tools really analyzing the same system?
- and different formalisms for specifying the property.
  - Are they really checking the same property?
- Effective use of even a single tool on a system of realistic size and complexity is a major project
  - need significant expertise with the tool
- Would like to measure factors affecting ease of use, not just performance (time/memory)
  - but *sound* experiments with human subjects are difficult to design correctly and very expensive

A Case Study—Chiron GUI Development System

Client-server system

- Client comprises
  - application for which interface is being constructed
  - ADTs that organize the data and functionality of application
  - *artists* that maintain mappings between ADTs and visual objects appearing on screen
  - runtime components providing coordination

A Case Study—Chiron GUI Development System

Client-server system

- Client comprises
  - application for which interface is being constructed
  - ADTs that organize the data and functionality of application
  - *artists* that maintain mappings between ADTs and visual objects appearing on screen
  - runtime components providing coordination
- Server manages aspects that are not artist- or application-specific

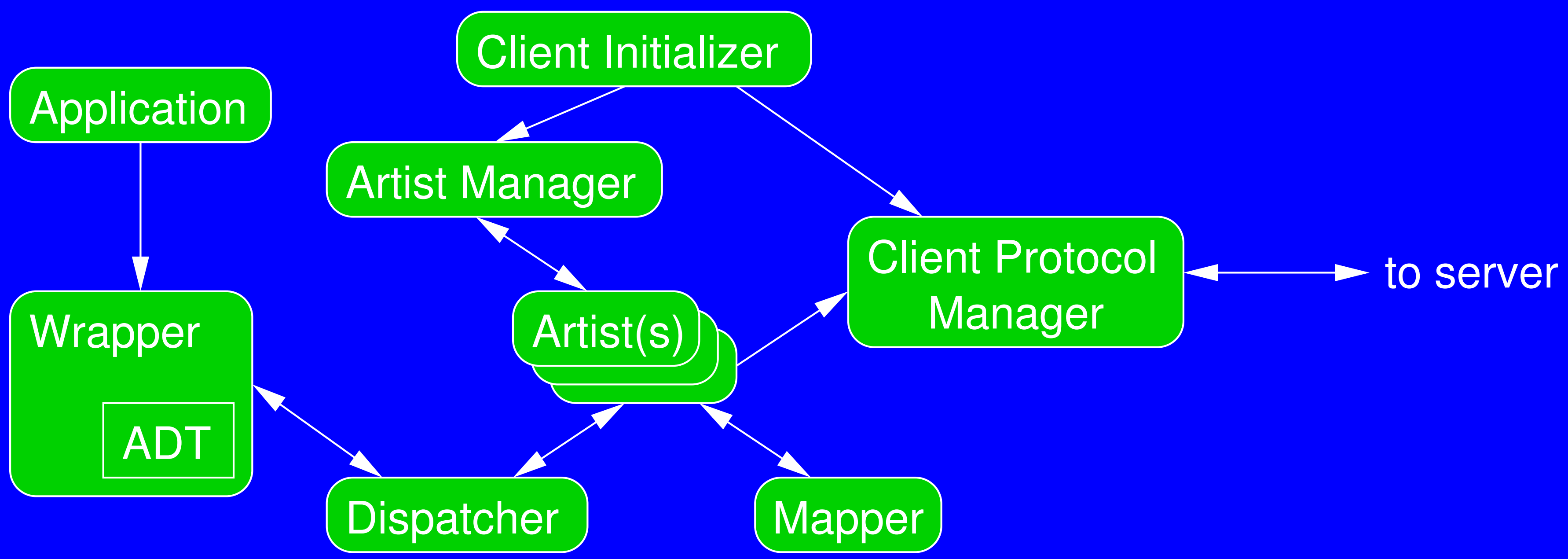
A Case Study—Chiron GUI Development System

Client-server system

- Client comprises
  - application for which interface is being constructed
  - ADTs that organize the data and functionality of application
  - *artists* that maintain mappings between ADTs and visual objects appearing on screen
  - runtime components providing coordination
- Server manages aspects that are not artist- or application-specific

Architecture is highly concurrent and even a toy interface is more than 1000 lines of Ada

Runtime Architecture of a Chiron Client



The Tools

- SMV
  - symbolic (BDD-based) model checker
- SPIN
  - explicit state model checker
- INCA
  - generates a system of inequalities that must have integer solutions if the property is violated, and uses integer linear programming to check
- FLAVERS
  - uses data flow analysis techniques to propagate states of property automaton and various constraint automata through graph representing control flow

Properties

We chose 10 properties that focus on event dispatching mechanism

Properties

We chose 10 properties that focus on event dispatching mechanism

- No formal spec
  - but extensive contact with Chiron developers
  - believe examples are realistic and reflect developers' understanding

Properties

We chose 10 properties that focus on event dispatching mechanism

- No formal spec
  - but extensive contact with Chiron developers
  - believe examples are realistic and reflect developers' understanding
- Sample properties:
  - Freedom from deadlock
  - If artist a_1 is registered for event e_1 and dispatcher receives e_1, it will not receive another event before passing e_1 to a_1.
  - Having received e_1, dispatcher does not notify artists of e_2.
  - Dispatcher does not notify an artist of an event the artist is not registered for.

Formalizing the Properties

Need to formalize natural language requirements in CTL, LTL, INCA query language, FLAVERS QRE

Formalizing the Properties

Need to formalize natural language requirements in CTL, LTL, INCA query language, FLAVERS QRE

- formalizations need to be *consistent*

Formalizing the Properties

Need to formalize natural language requirements in CTL, LTL, INCA query language, FLAVERS QRE

- formalizations need to be *consistent*
- would like to choose formalizations that lead to best tool performance

Formalizing the Properties

Need to formalize natural language requirements in CTL, LTL, INCA query language, FLAVERS QRE

- formalizations need to be *consistent*
- would like to choose formalizations that lead to best tool performance
- used specification patterns system wherever possible (but don't know this leads to best tool performance)

A Property Example

If artist a_1 is registered for event e_1 and dispatcher receives e_1, it will not receive another event before passing e_1 to a_1.

In the pattern system, this is "Existence of P between Q and R"

A Property Example

If artist a_1 is registered for event e_1 and dispatcher receives e_1, it will not receive another event before passing e_1 to a_1.

In the pattern system, this is "Existence of P between Q and R"

LTL:

```
[](((afternotifyartistse1 && isregistereda1e1) &&
    <>(notifyartistse1 || notifyartistse2)) -> (!(notifyartistse1 ||
    notifyartistse2) U notifyclienteventa1e1))
```

A Property Example

If artist a_1 is registered for event e_1 and dispatcher receives e_1, it will not receive another event before passing e_1 to a_1.

In the pattern system, this is "Existence of P between Q and R"

LTL:

```
[](((afternotifyartistse1 && isregistereda1e1) &&
    <>(notifyartistse1 || notifyartistse2)) -> (!(notifyartistse1 ||
    notifyartistse2) U notifyclienteventa1e1))
```

CTL:

```
AG((afternotifyartistse1 & isregistereda1e1) ->
    !E[!notifyclienteventa1e1 U (notifyartistse1 | notifyartistse2)])
```

Property Example, continued

INCA:

```
(defquery "p02a" "nofair" (omega-star-less (sequence
  (interval :initial t :open t
    :constraints
      '((>= "accept(a1;dispatcher.register_event;art1;e1)"
      (+ 1 "accept(a1;dispatcher.unregister_event;art1;e1)")))
    :ends-with '((rendezvous "adt_wrapper;dispatcher.notify_artists;e1;")))
  (interval
    :ends-with '((and
      (or (prefix "call") (prefix "accept"))
      (contains "adt_wrapper;dispatcher.notify_artists")))
    :forbid '((rendezvous "dispatcher;a1.notify_client_event;e1"))))))
```

Property Example, continued

FLAVERS:

{dispatcher_register_event_a1_e1, dispatcher_notify_artists_e2
dispatcher_unregister_event_a1_e1, artist1_notify_client_event_e1,
dispatcher_notify_artists_e1}

none

.*;
dispatcher_register_event_a1_e1;
[-dispatcher_unregister_event_a1_e1]*;
dispatcher_notify_artists_e1;
[-artist1_notify_client_event_e1]*;
[dispatcher_notify_artists_e1, dispatcher_notify_artists_e2];
.*

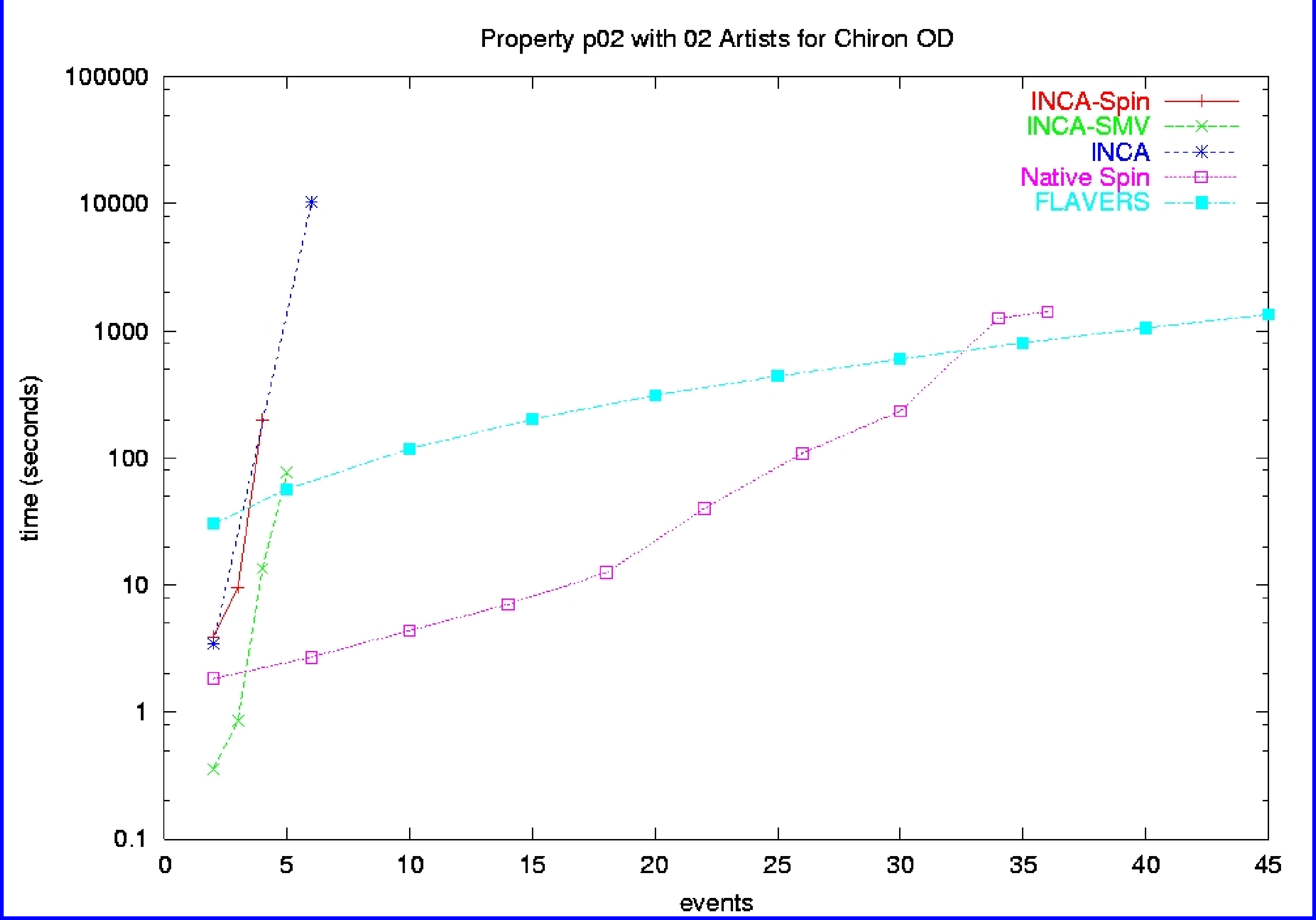
Building the Models

- Abstraction done at source code level by hand
  - also converted one dynamic data structure of bounded size to a static one

Building the Models

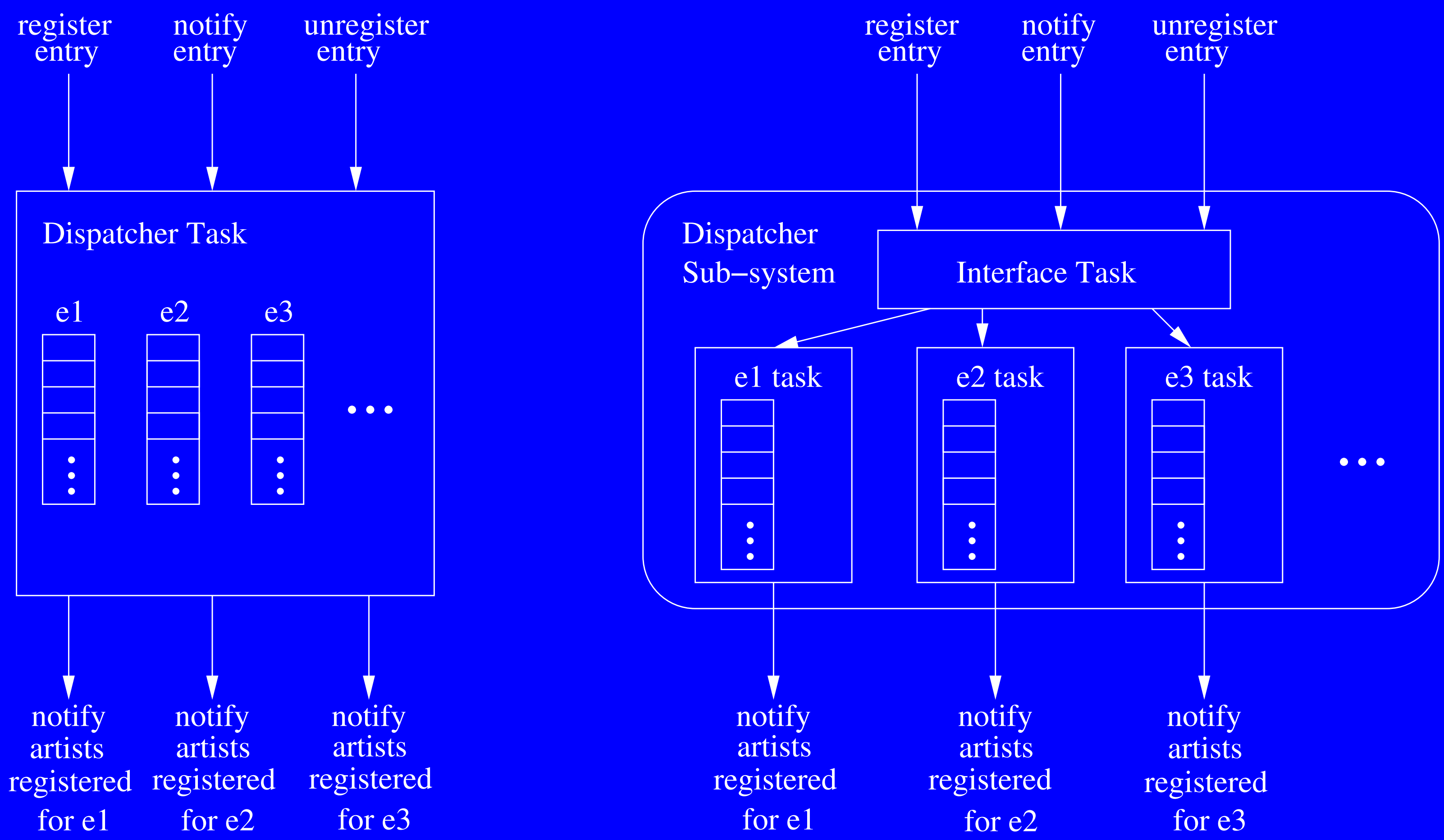
- Abstraction done at source code level by hand
  - also converted one dynamic data structure of bounded size to a static one
- Used Corbett's translators [*TSE* 1996] to build SMV transition relation and Promela code
  - from FSAs built by INCA
  - extensive effort validating translators

Times

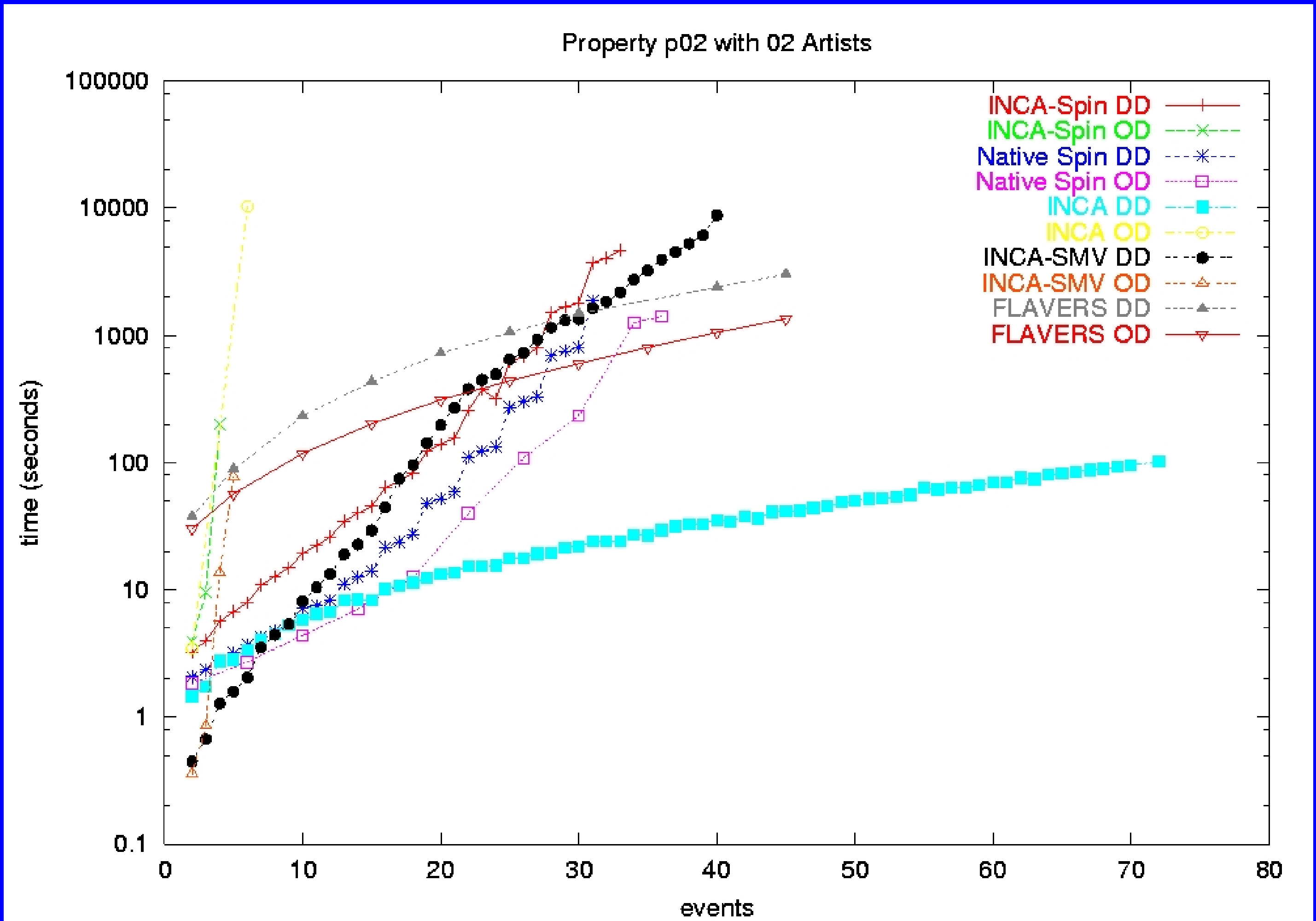


Property p02 with 02 Artists for Chiron OD
INCA-Spin
INCA-SMV
INCA
Native Spin
FLAVERS
time (seconds)
events
100000
10000
1000
100
10
1
0.1
0
5
10
15
20
25
30
35
40
45

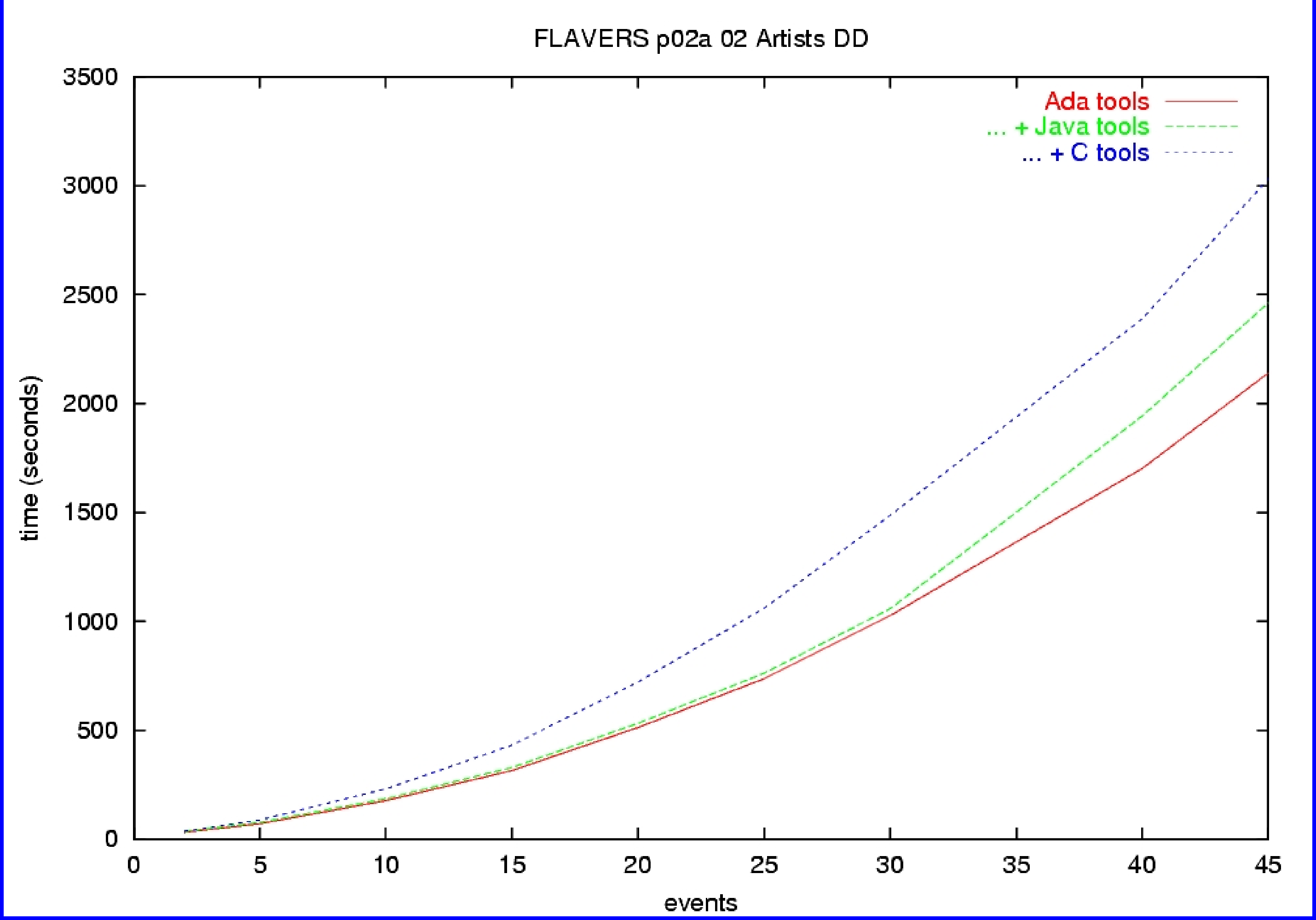
Decomposing the Dispatcher Task



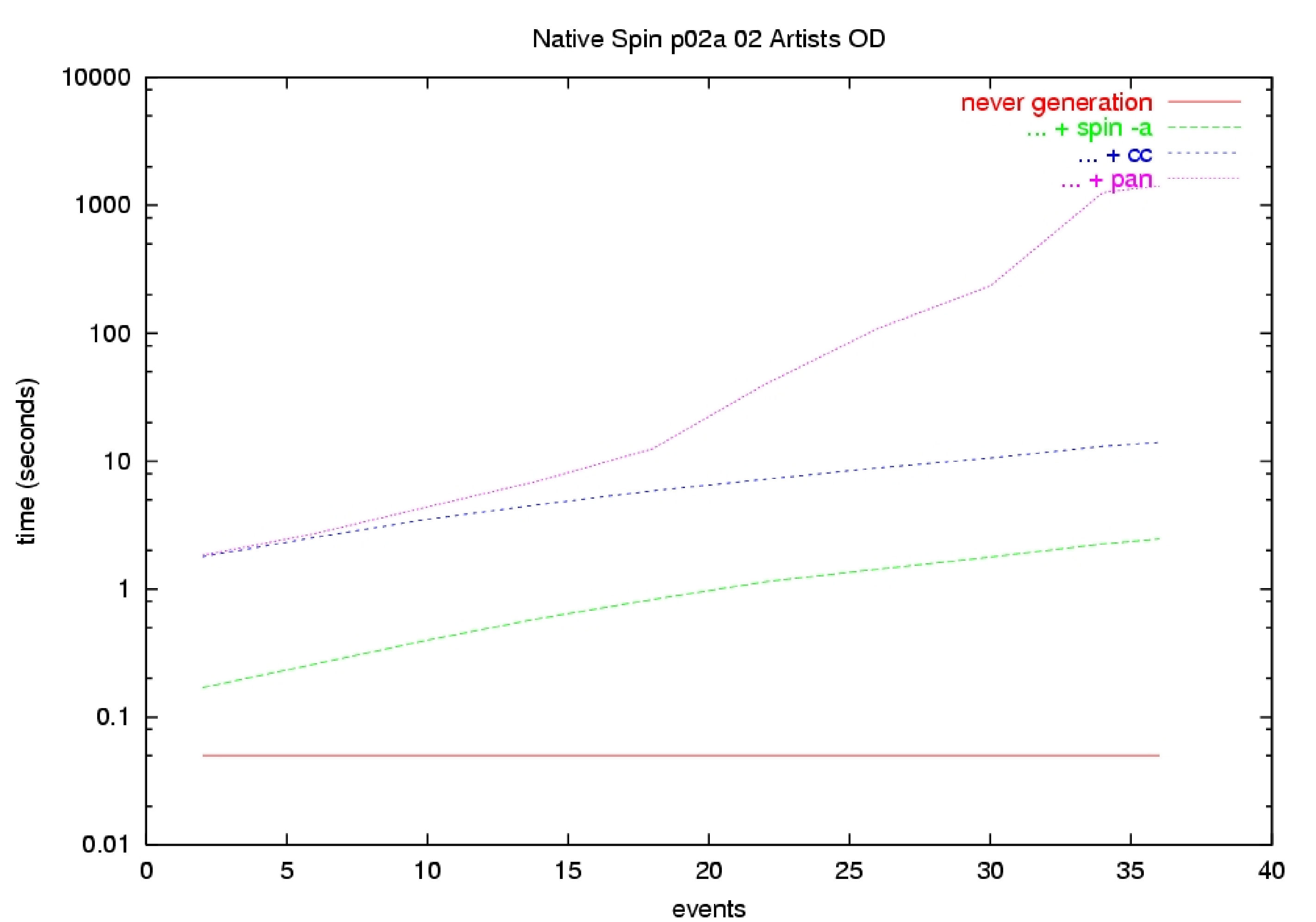
Times For Both Versions of Dispatcher



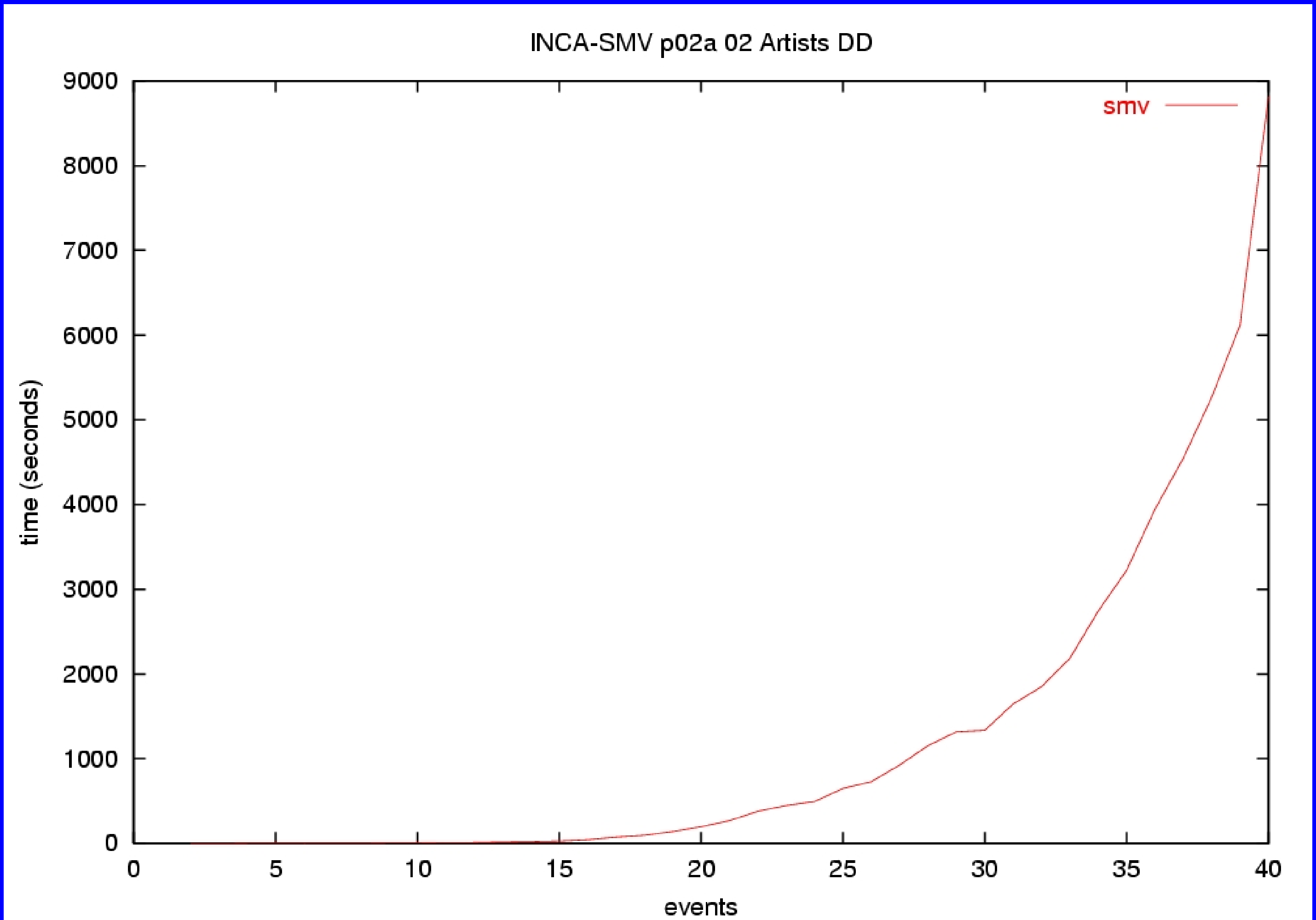
Components of FLAVERS Time



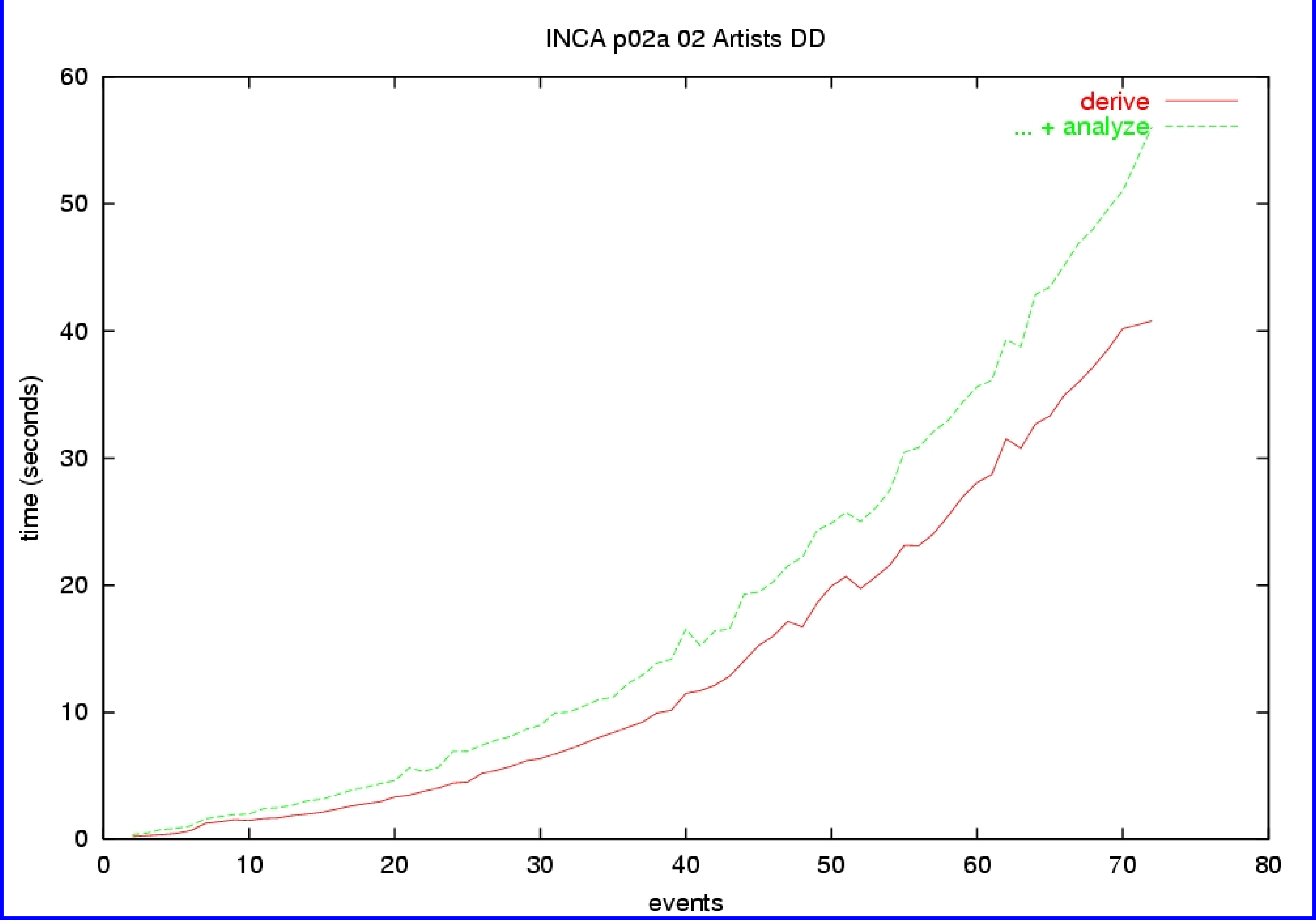
Components of SPIN Time



Components of SMV Time



Components of INCA Time



Some of the Things We Learned

Disclaimer: Certainly can't draw any general conclusions about the tools from these experiments

- Potential sources of bias are many and subtle!!

Some of the Things We Learned

Disclaimer: Certainly can't draw any general conclusions about the tools from these experiments

- Potential sources of bias are many and subtle!!
- INCA, and hence Corbett's translators, break down when one task has a large number of states, but very effective when all tasks are fairly small

Some of the Things We Learned

Disclaimer: Certainly can't draw any general conclusions about the tools from these experiments

- Potential sources of bias are many and subtle!!
- INCA, and hence Corbett's translators, break down when one task has a large number of states, but very effective when all tasks are fairly small
- Tools are very sensitive to details of model
  - substantial variation for all tools on "equivalent" systems

Some of the Things We Learned

Disclaimer: Certainly can't draw any general conclusions about the tools from these experiments

- Potential sources of bias are many and subtle!!
- INCA, and hence Corbett's translators, break down when one task has a large number of states, but very effective when all tasks are fairly small
- Tools are very sensitive to details of model
  - substantial variation for all tools on "equivalent" systems
- Relatively little variation across properties
  - but algorithms and abstractions tailored to property can be helpful

Some of the Things We Learned, continued

- Slicing likely to be very useful
  - FLAVERS models only variables needed to verify property, but didn't have (automated) way to do this for the other tools

Some of the Things We Learned, continued

- Slicing likely to be very useful
  - FLAVERS models only variables needed to verify property, but didn't have (automated) way to do this for the other tools

- Comparisons complicated by details of application of different tools and by evolution of tools.
  - *All* of the tools have changed during this study

Some of the Things We Learned, continued

- Slicing likely to be very useful
  - FLAVERS models only variables needed to verify property, but didn't have (automated) way to do this for the other tools

- Comparisons complicated by details of application of different tools and by evolution of tools.
  - *All* of the tools have changed during this study

- Properties of interest were relatively complicated.