On CS Undergrad Teaching and Visualization Research

Part 1 (teaching)

A Mini-lecture on
sorting in different programming paradigms

Part 2 (research)

Visual Data Analysis:
real users, real data, right methods

Michael Sedlmair
Visualization and Data Analysis group (VDA)
universität wien

Part 1: Teaching

A Mini-lecture on Sorting in different programming paradigms

Assumptions wrt previous knowledge

- Target audience:
  - Mainly 2nd term CS students,
  - some 1st term
- Previous knowledge of
  - Programming: Java
  - Algorithms: Complexity analysis, data structures, recursion
- Learning goals
  - Repetition/introduction of sorting algorithms
  - First contact with programming paradigms,
    specifically imperative VS. declarative programming

Outline

- Sorting algorithms
  - Selection Sort
  - Quicksort
- Sorting in different programming paradigms
  - Imperative VS. declarative
  - Quicksort in Java (imperative)
  - Quicksort in Haskell (declarative/functional)

Sorting algorithms

Sorting

Sorting

- One of the standard algorithmic problems in Computer Science
- Goal: Put an unsorted list of elements *u* in ascending/ descending order.
- Example:
  [7,3,3,1,5] —> [1,3,3,5,7]
  element u_1,
  value_of(u_1) =7
- *(Note: special case of sorting numbers in ascending order)*

Sorting algorithms

- Simple sorts
  - **Selection Sort**
  - Insertion Sort
  - Bubble Sort
  - …
- More efficient sorts
  - **Quicksort**
  - Merge Sort
  - Heap Sort
  - Radixsort
  - …

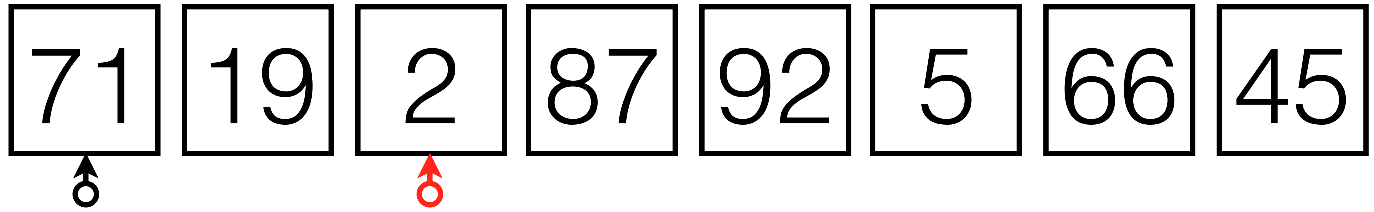
Selection Sort

- Split list into 2 parts:
  - *L*, sorted part of the list (left)
  - *R*, rest of the list (right)
- At the beginning $L = \varnothing$
- Identify the smallest element in *R* and swap it with the first element of *R*
- Done when $|R| = 1$

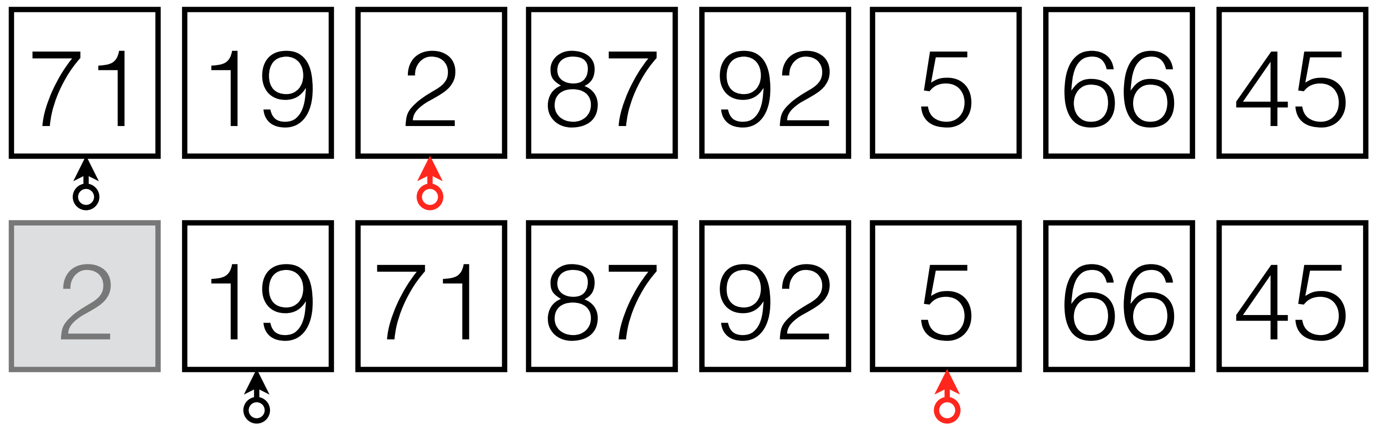
Selection Sort

71	19	2	87	92	5	66	45

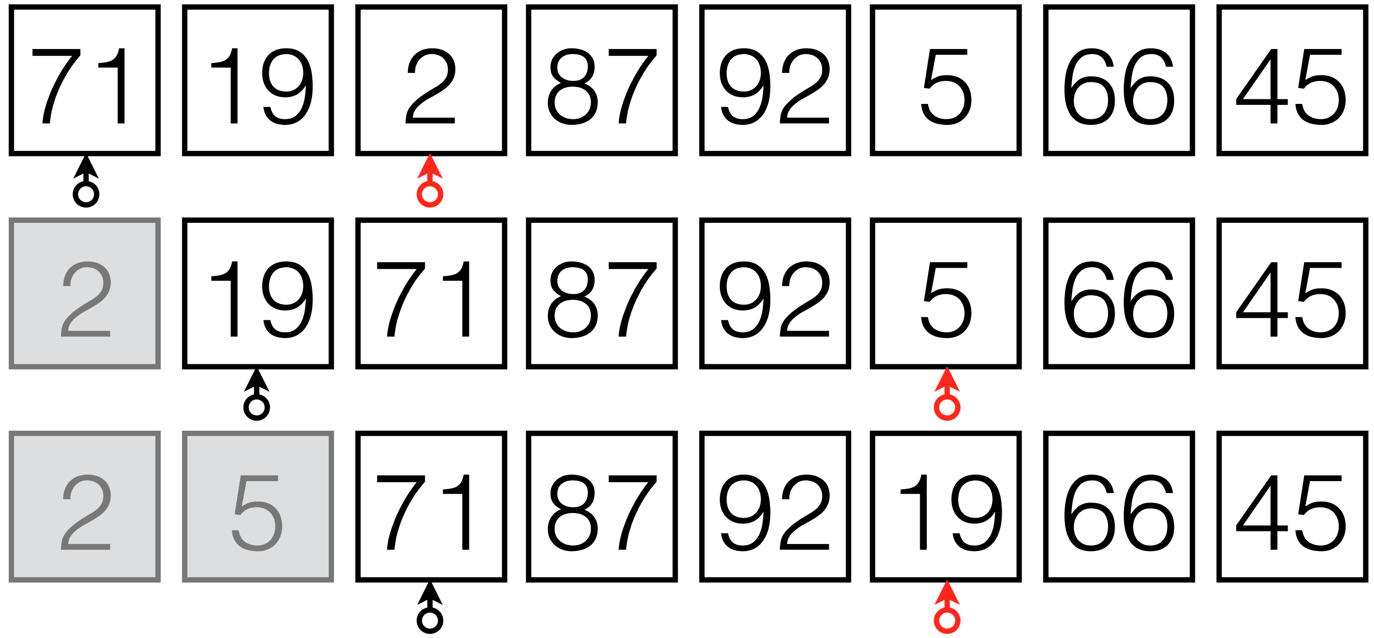
Selection Sort



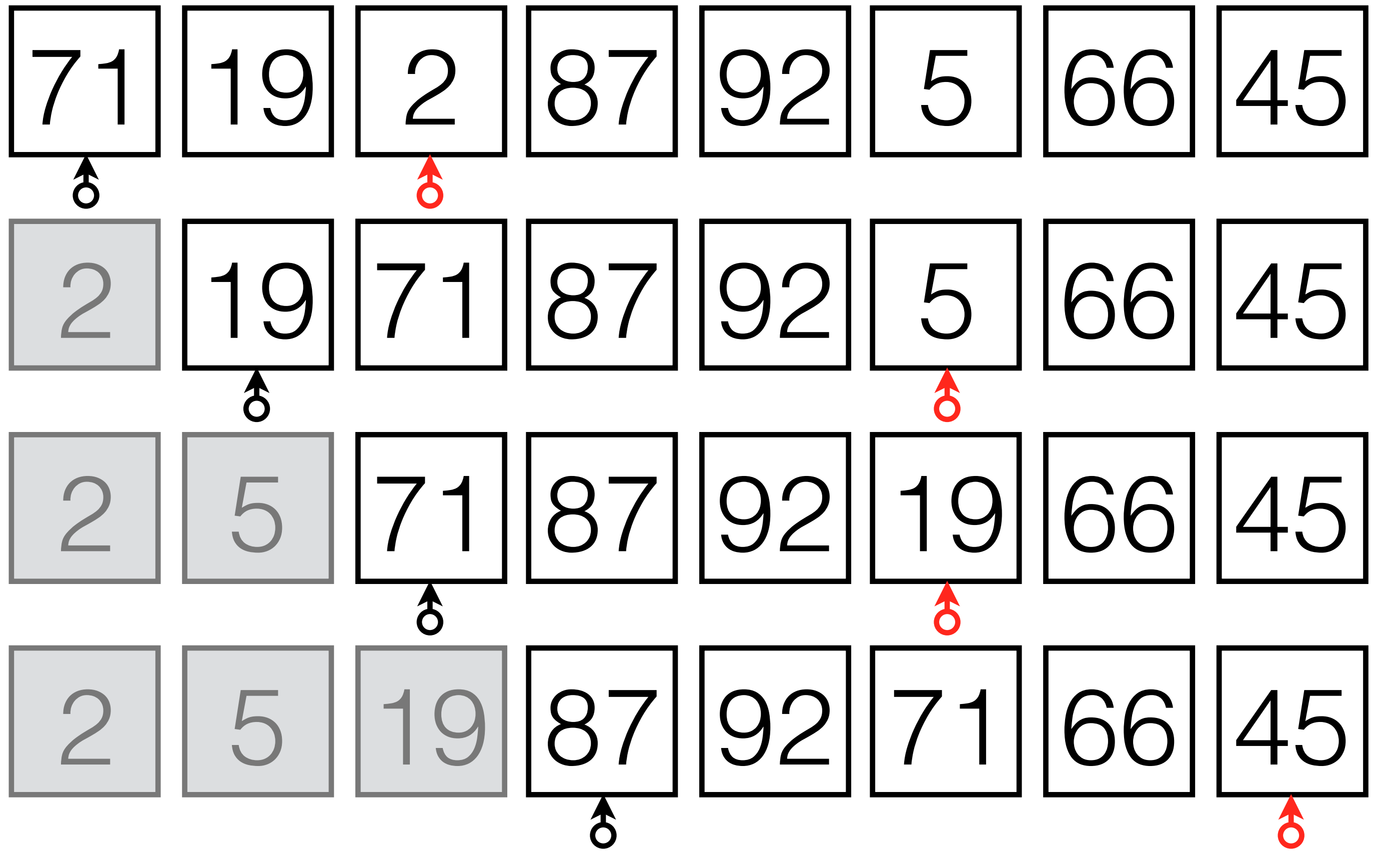
Selection Sort



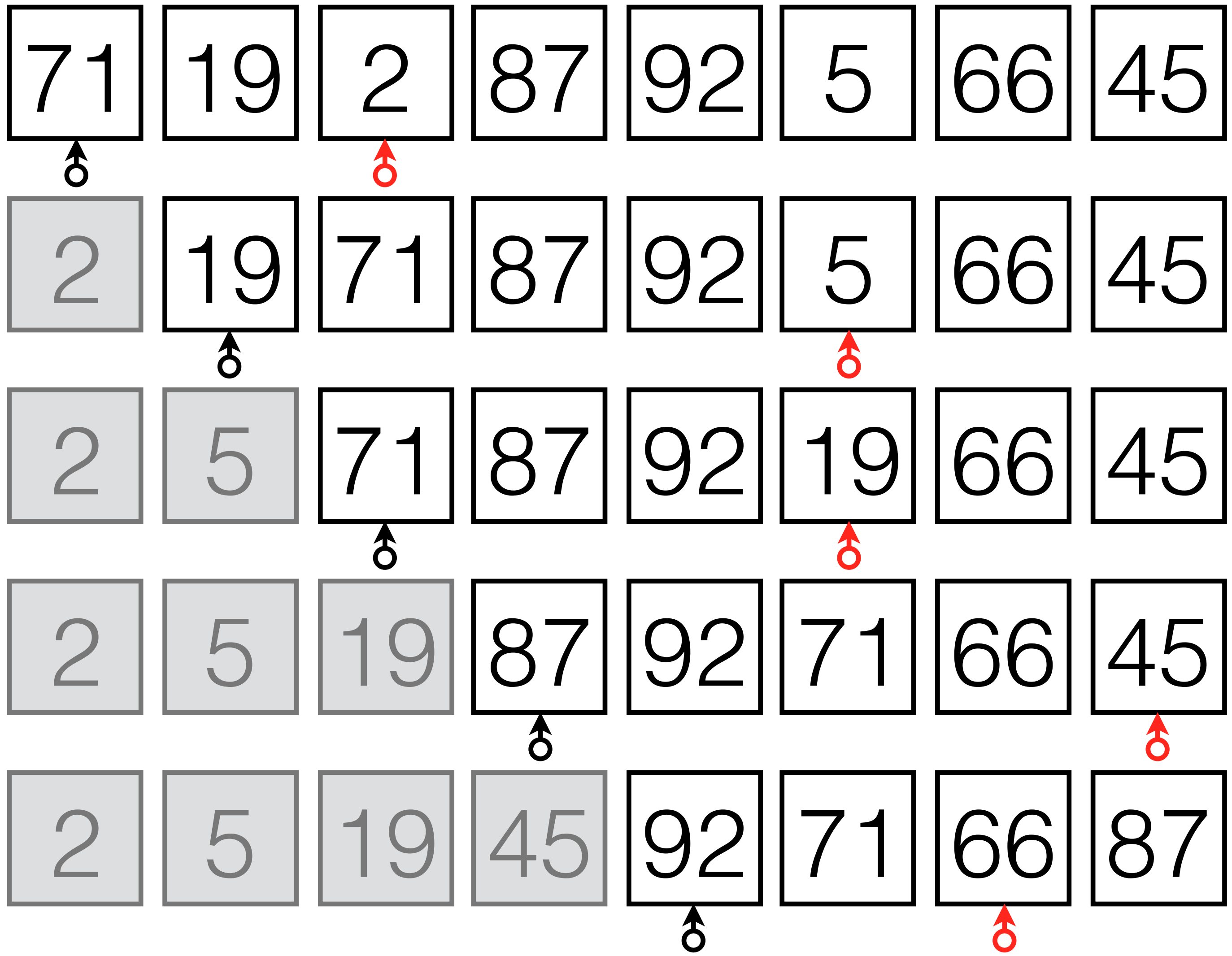
Selection Sort



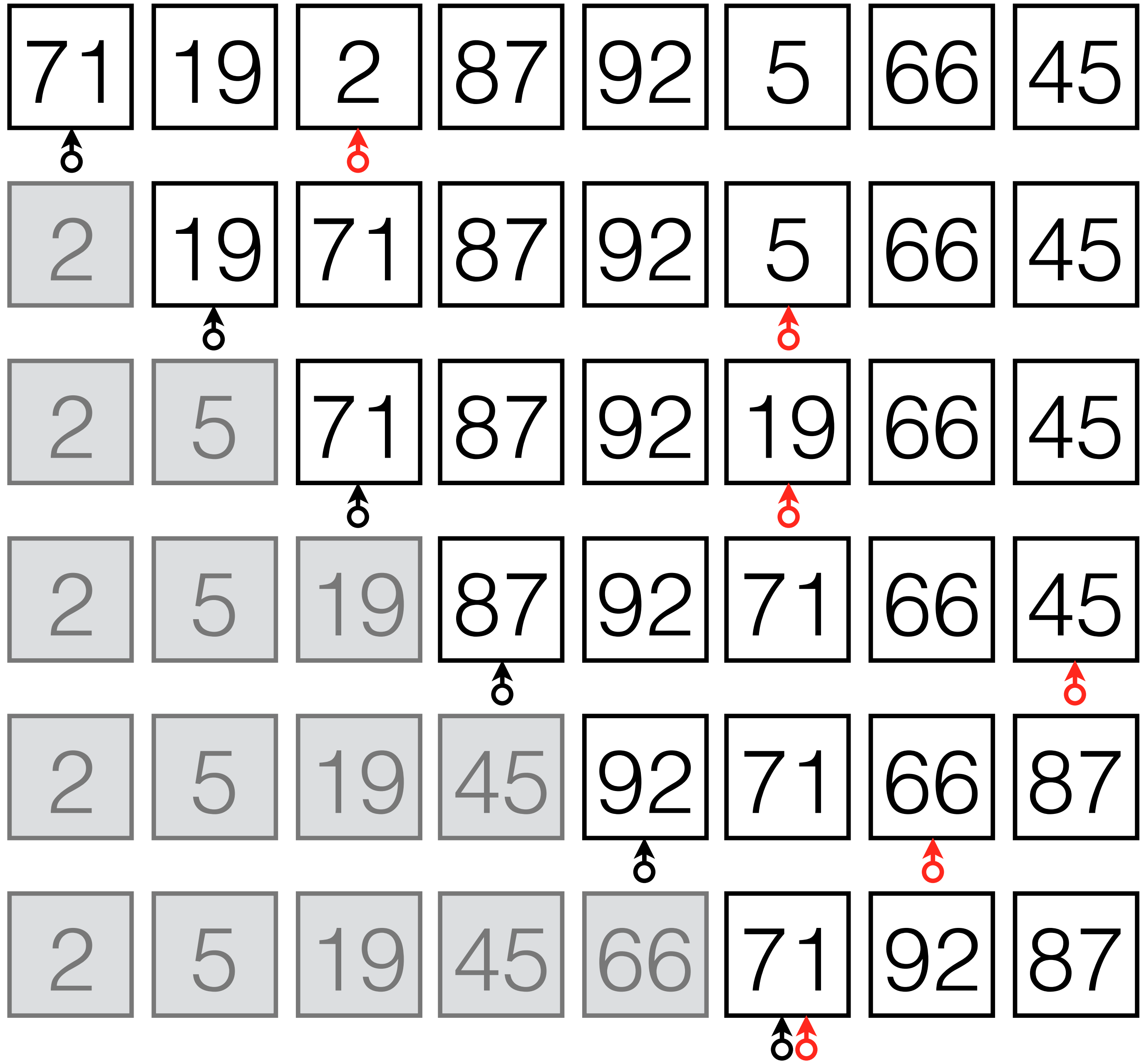
Selection Sort



Selection Sort

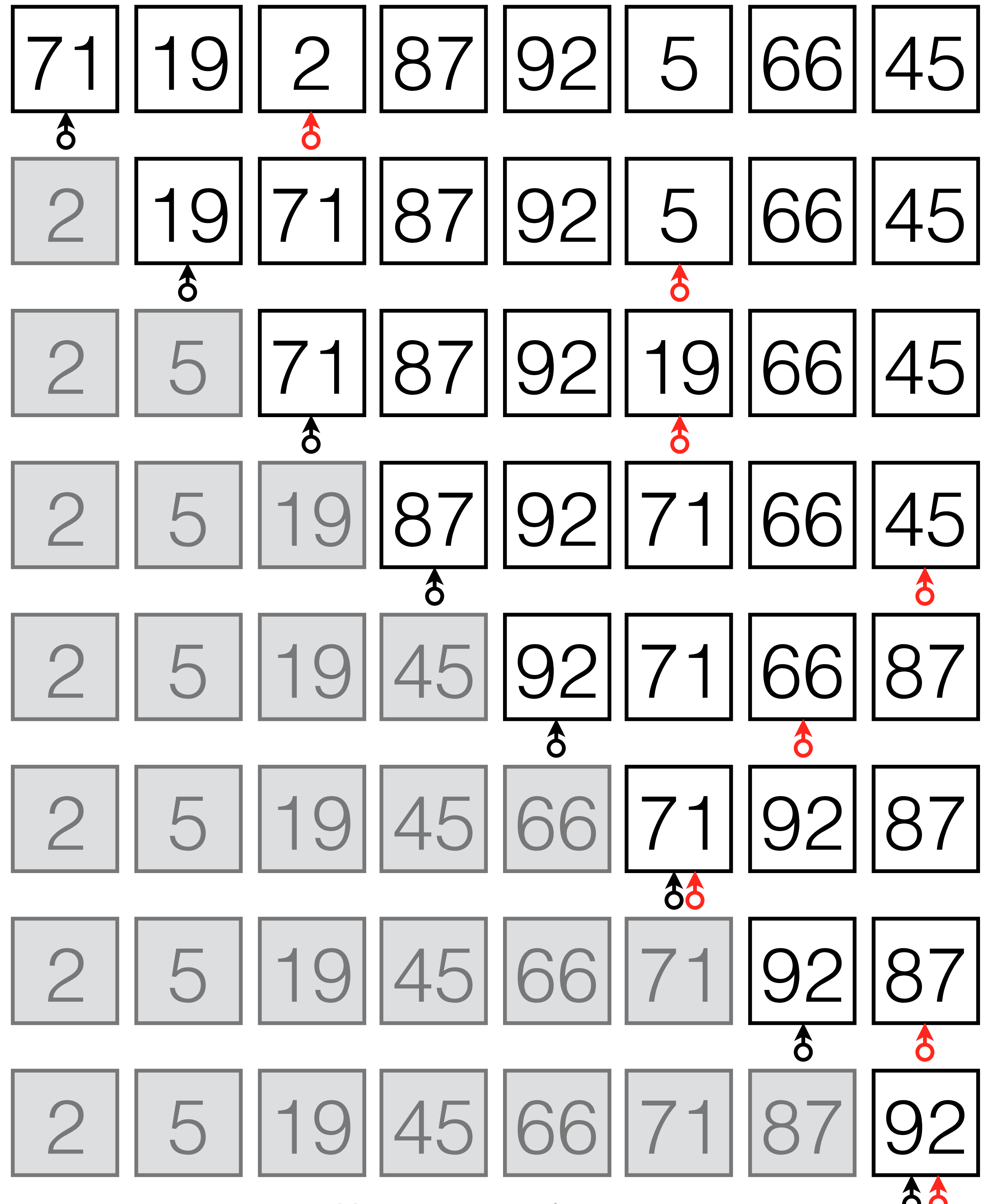


Selection Sort



Selection Sort

Selection Sort



Selection Sort: Strengths / Weaknesses

- Strengths
  - easy to understand
  - “in-place” (memory usage: *n*)

Selection Sort: Strengths / Weaknesses

- Strengths
  - easy to understand
  - “in-place” (memory usage: *n*)

- Biggest Weakness
  - Expensive for large lists: $O(n^2)$

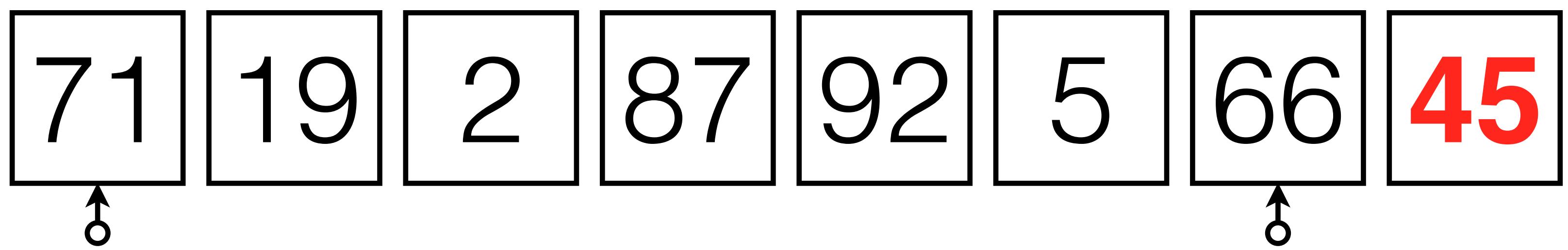
Quicksort

- Divide-and-conquer algorithms
- Determine an arbitrary *pivot element* u_p (e.g., last element of a list)
- Swap elements in the list, so that
    - all elements left of u_p are smaller than u_p
    - all elements right of u_p are greater than u_p
- The *pivot* u_p is then already at its final position
- Recursively sort left and right parts of the list

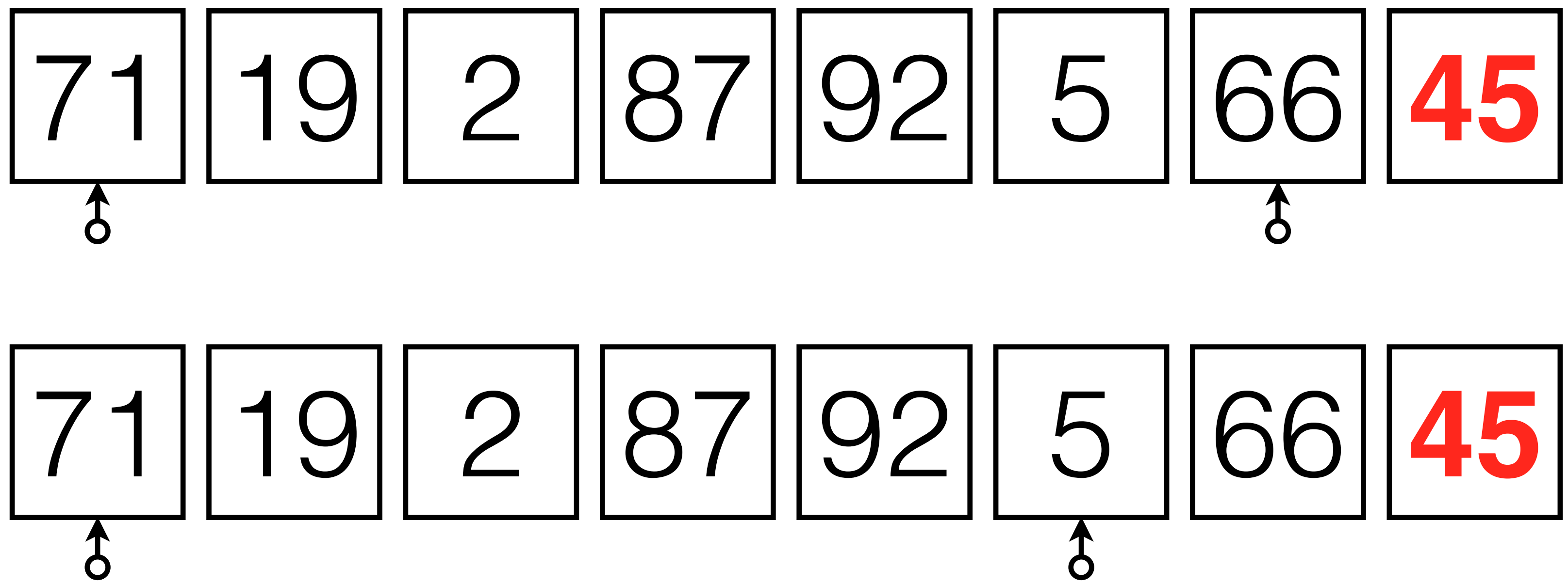
Quicksort

- Moving the pivot:
  - Go from left to right until you find an element u_i with value_of(u_i) > value_of(u_p)
  - Go from right to left until you find an element u_j with value_of(u_j) < value_of(u_p)
  - Swap these elements
  - Repeat until *i a*nd *j* meet
  - Swap *pivot element* with the element at position *i* if value_of(u_p) < value_of(u_i)

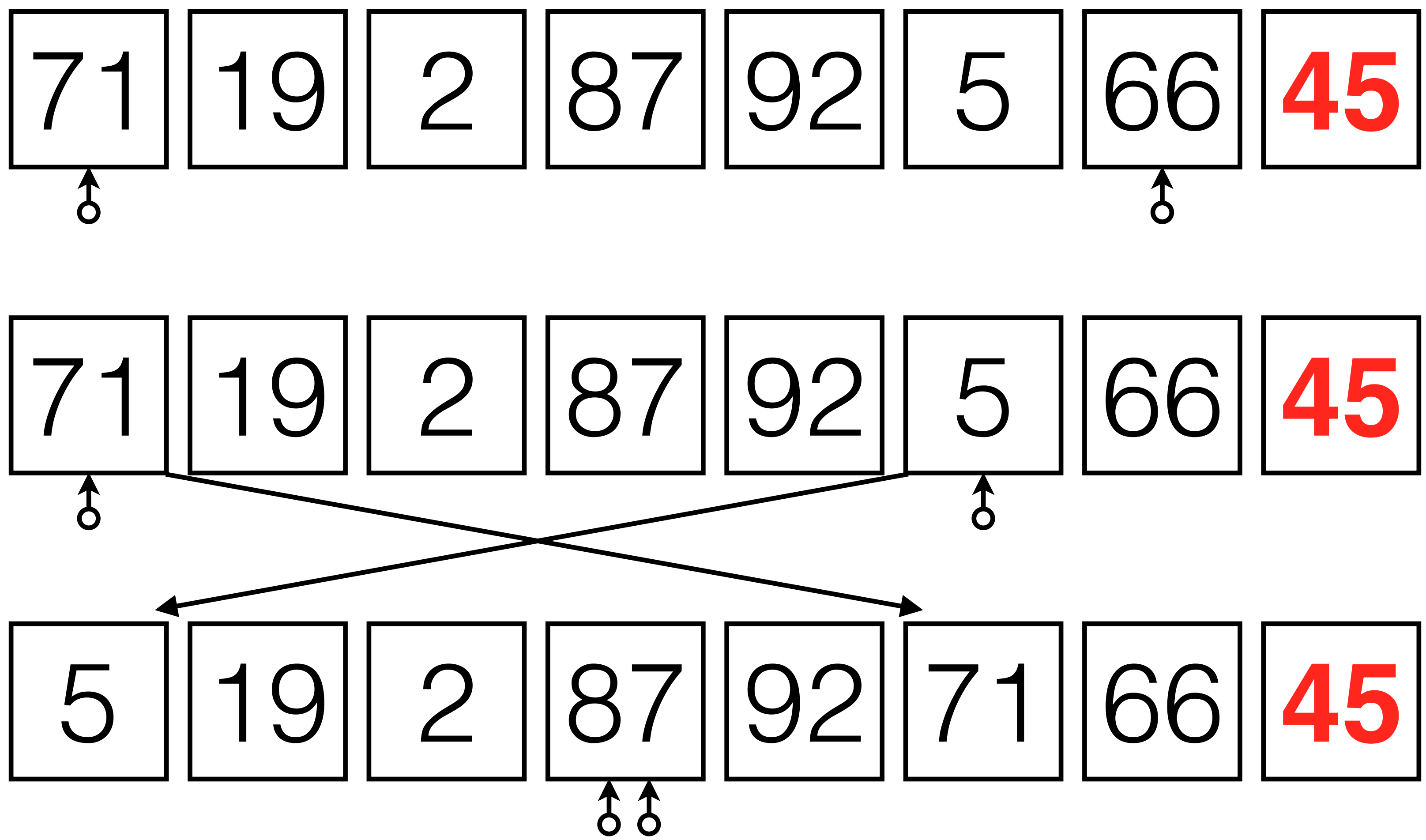
Quicksort



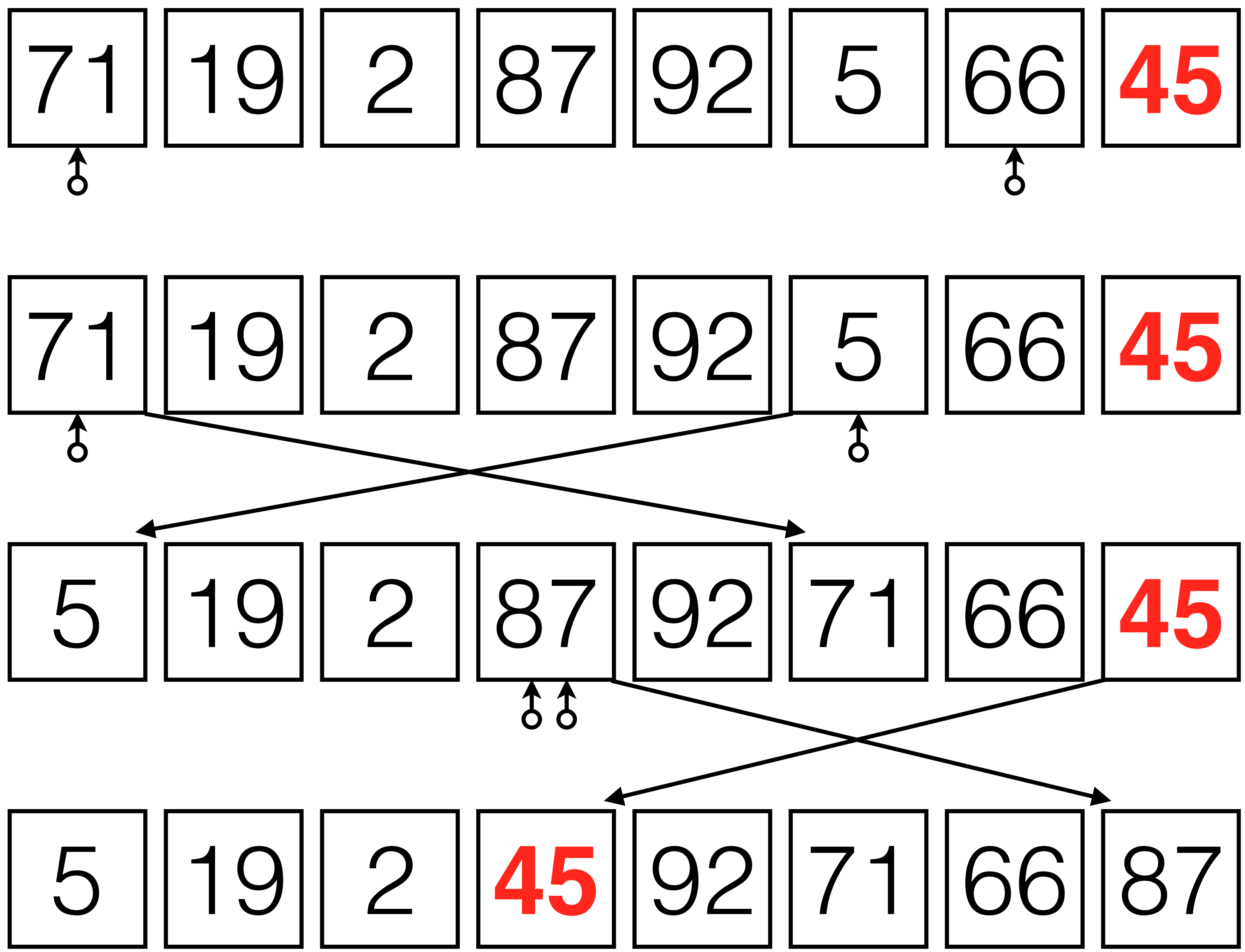
Quicksort



Quicksort

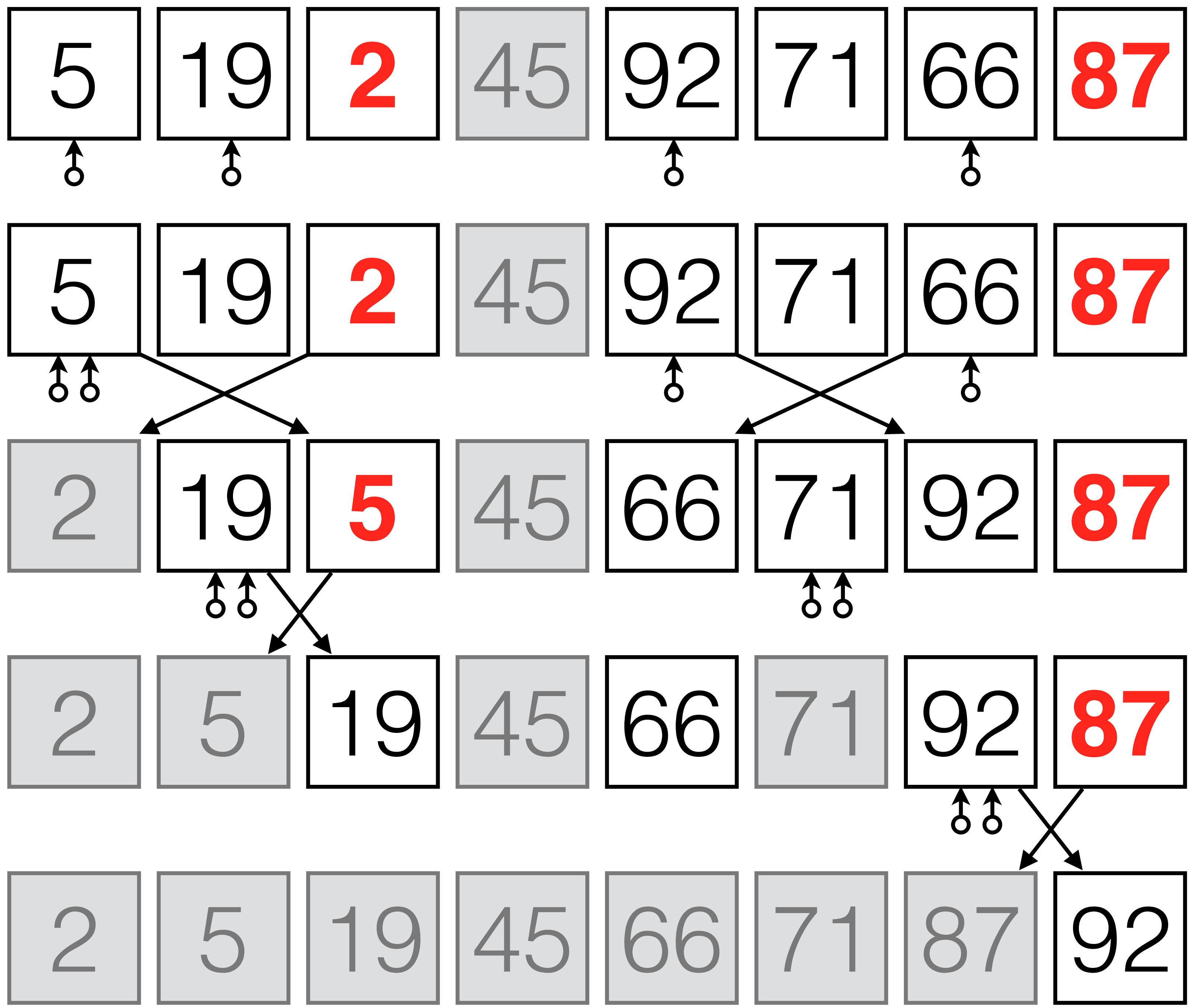


Quicksort



71 19 2 87 92 5 66 **45**
71 19 2 87 92 5 66 **45**
5 19 2 87 92 71 66 **45**
5 19 2 **45** 92 71 66 87

Quicksort



2 5 19 45 66 71 87 92

Quicksort: Strength / Weaknesses

- Strength:
  - Average complexity: $O(n \cdot \log_2(n))$
  - “in-place” memory

Quicksort: Strength / Weaknesses

- Strength:
  - Average complexity: $O(n \cdot \log_2(n))$
  - "in-place" memory
- Weaknesses:
  - Worst case complexity: $O(n^2)$
  - not efficient for small lists

Sorting in different programming paradigms

(using the example Quicksort)

Programming paradigms

- Fundamental style or “way” of programming
- Some examples
  - Imperative — Give an explicit sequence of commands
  - Declarative — Programs specify the result, not how to get it.
  - Object-Oriented — Computation is effected by sending messages to objects; objects have state and behavior.
  - …
- Not mutually exclusive concepts! Specifically, languages can follow more than one paradigm

Imperative VS. declarative

<u>**Imperative**</u> VS. declarative

- Step-by-step recipe
- Describe the <u>how</u>: Control flow in imperative programming is explicit; commands show how the computation takes place.
- Examples: <u>Java</u>, C++, C, …

Imperativ: Quicksort in Java

```java
public static void qsort (int[] val, int low, int high) {

    if (high - low > 0){
        //get pivot element from the end of the list
        int pivot = high;

        //initialize pointers
        int l = low, h = high - 1;

        while (l < h) {
            //set pointers
            while ((l < h) && (val[l] <= val[pivot])) {
                l++;
            }
            while ((l < h) && (val[h] >= val[pivot])) {
                h--;
            }

            //swap values
            if (l != h) {
                swap(val, l, h);
            } else if (val[l] > val[pivot]) {
                swap(val, l, pivot);
            }
        }
        //recursion
        quicksort(val, low, l - 1);
        quicksort(val, l + 1, high);
    }
}
```

71	19	2	87	92	5	66	**45**

Imperativ: Quicksort in Java

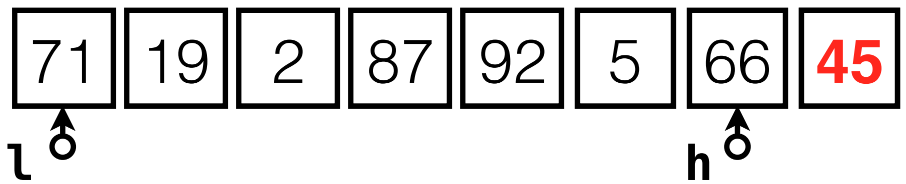
```java
public static void qsort (int[] val, int low, int high) {

    if (high - low > 0){
        //get pivot element from the end of the list
        int pivot = high;

        //initialize pointers
        int l = low, h = high - 1;

        while (l < h) {
            //set pointers
            while ((l < h) && (val[l] <= val[pivot])) {
                l++;
            }
            while ((l < h) && (val[h] >= val[pivot])) {
                h--;
            }

            //swap values
            if (l != h) {
                swap(val, l, h);
            } else if (val[l] > val[pivot]) {
                swap(val, l, pivot);
            }
        }
        //recursion
        quicksort(val, low, l - 1);
        quicksort(val, l + 1, high);
    }
}
```



Imperativ: Quicksort in Java

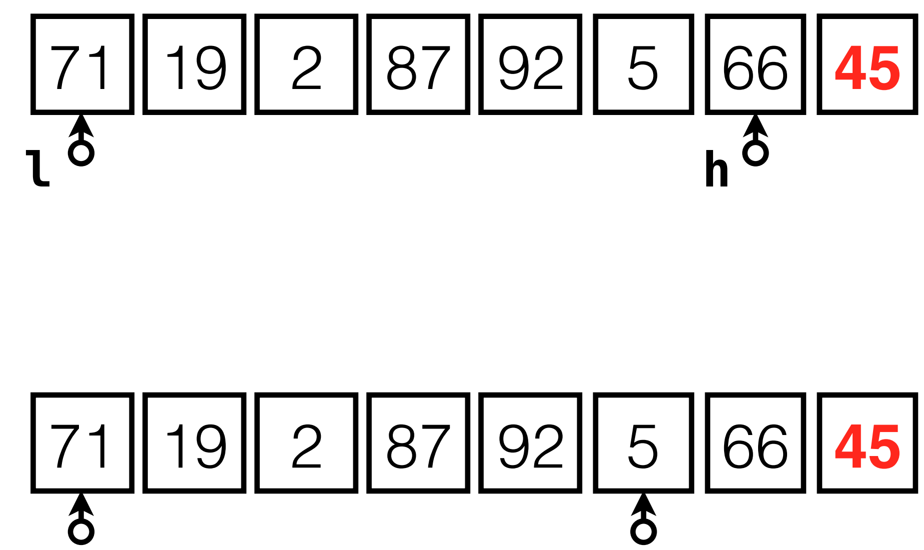
```
public static void qsort (int[] val, int low, int high) {

    if (high - low > 0){
        //get pivot element from the end of the list
        int pivot = high;

        //initialize pointers
        int l = low, h = high - 1;

        while (l < h) {
            //set pointers
            while ((l < h) && (val[l] <= val[pivot])) {
                l++;
            }
            while ((l < h) && (val[h] >= val[pivot])) {
                h--;
            }

            //swap values
            if (l != h) {
                swap(val, l, h);
            } else if (val[l] > val[pivot]) {
                swap(val, l, pivot);
            }
        }
        //recursion
        quicksort(val, low, l - 1);
        quicksort(val, l + 1, high);
    }
}
```



Imperativ: Quicksort in Java

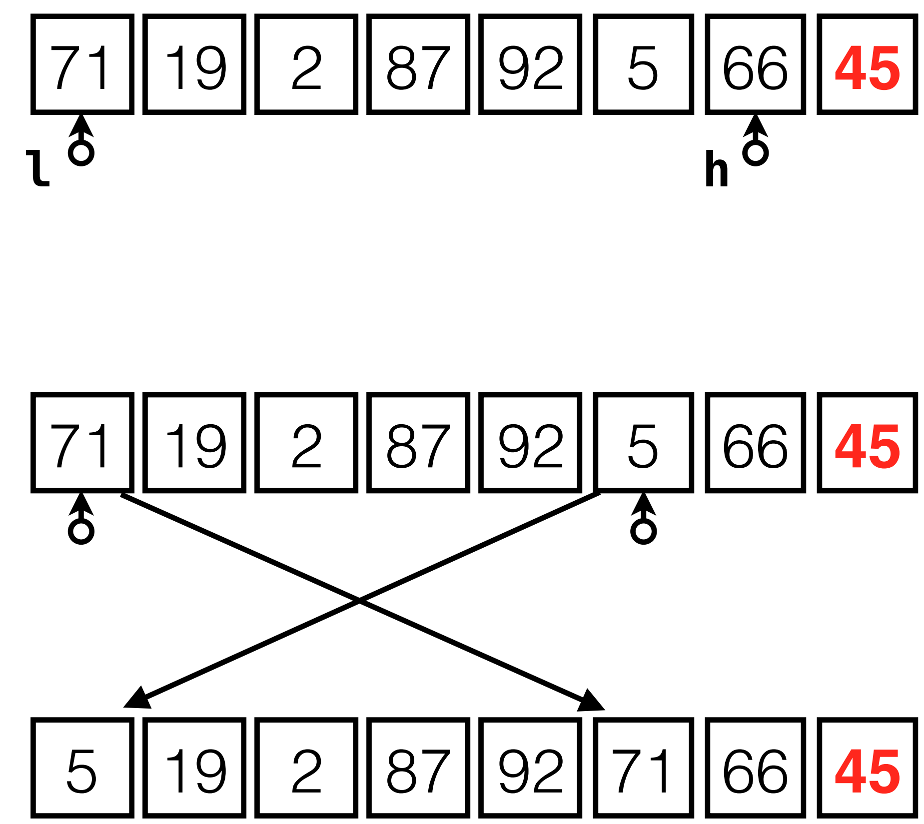
```java
public static void qsort (int[] val, int low, int high) {

    if (high - low > 0){
        //get pivot element from the end of the list
        int pivot = high;

        //initialize pointers
        int l = low, h = high - 1;

        while (l < h) {
            //set pointers
            while ((l < h) && (val[l] <= val[pivot])) {
                l++;
            }
            while ((l < h) && (val[h] >= val[pivot])) {
                h--;
            }

            //swap values
            if (l != h) {
                swap(val, l, h);
            } else if (val[l] > val[pivot]) {
                swap(val, l, pivot);
            }
        }
        //recursion
        quicksort(val, low, l - 1);
        quicksort(val, l + 1, high);
    }
}
```



Imperativ: Quicksort in Java

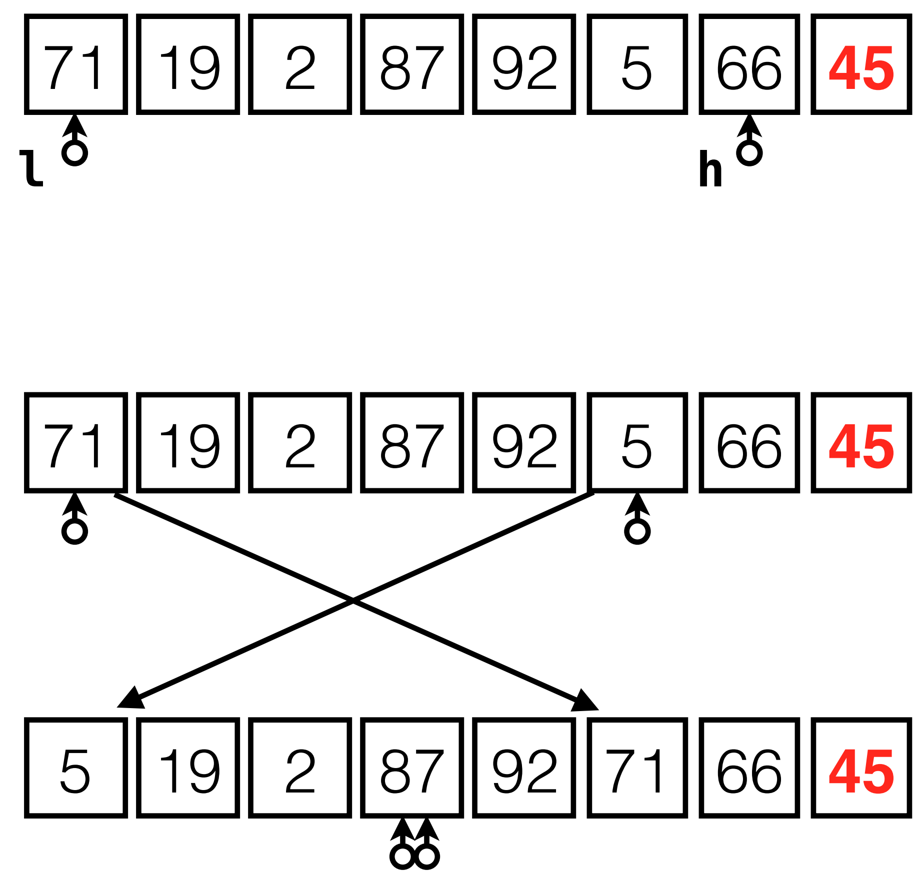
```java
public static void qsort (int[] val, int low, int high) {

    if (high - low > 0){
        //get pivot element from the end of the list
        int pivot = high;

        //initialize pointers
        int l = low, h = high - 1;

        while (l < h) {
            //set pointers
            while ((l < h) && (val[l] <= val[pivot])) {
                l++;
            }
            while ((l < h) && (val[h] >= val[pivot])) {
                h--;
            }

            //swap values
            if (l != h) {
                swap(val, l, h);
            } else if (val[l] > val[pivot]) {
                swap(val, l, pivot);
            }
        }
        //recursion
        quicksort(val, low, l - 1);
        quicksort(val, l + 1, high);
    }
}
```



Imperativ: Quicksort in Java

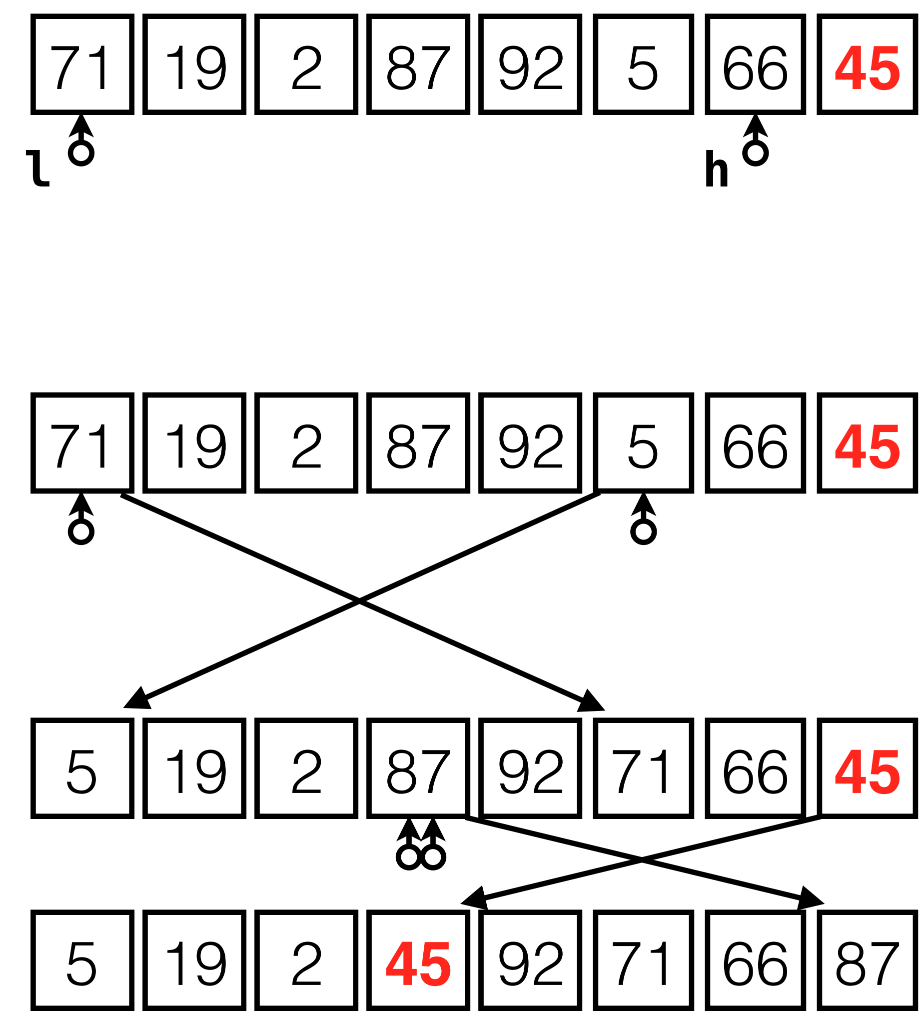
```
public static void qsort (int[] val, int low, int high) {

    if (high - low > 0){
        //get pivot element from the end of the list
        int pivot = high;

        //initialize pointers
        int l = low, h = high - 1;

        while (l < h) {
            //set pointers
            while ((l < h) && (val[l] <= val[pivot])) {
                l++;
            }
            while ((l < h) && (val[h] >= val[pivot])) {
                h--;
            }

            //swap values
            if (l != h) {
                swap(val, l, h);
            } else if (val[l] > val[pivot]) {
                swap(val, l, pivot);
            }
        }
        //recursion
        quicksort(val, low, l - 1);
        quicksort(val, l + 1, high);
    }
}
```



Imperativ: Quicksort in Java

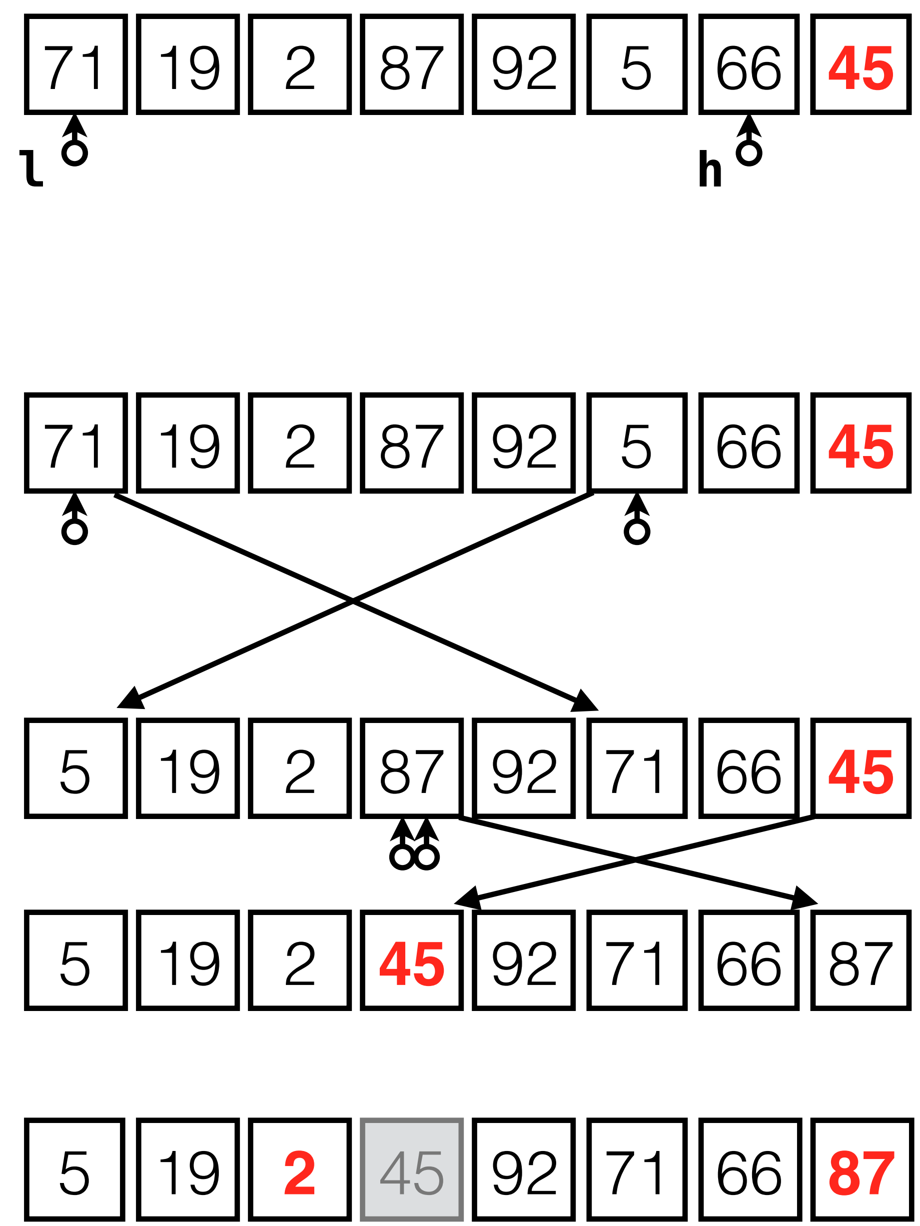
```java
public static void qsort (int[] val, int low, int high) {

    if (high - low > 0){
        //get pivot element from the end of the list
        int pivot = high;

        //initialize pointers
        int l = low, h = high - 1;

        while (l < h) {
            //set pointers
            while ((l < h) && (val[l] <= val[pivot])) {
                l++;
            }
            while ((l < h) && (val[h] >= val[pivot])) {
                h--;
            }

            //swap values
            if (l != h) {
                swap(val, l, h);
            } else if (val[l] > val[pivot]) {
                swap(val, l, pivot);
            }
        }
        //recursion
        quicksort(val, low, l - 1);
        quicksort(val, l + 1, high);
    }
}
```

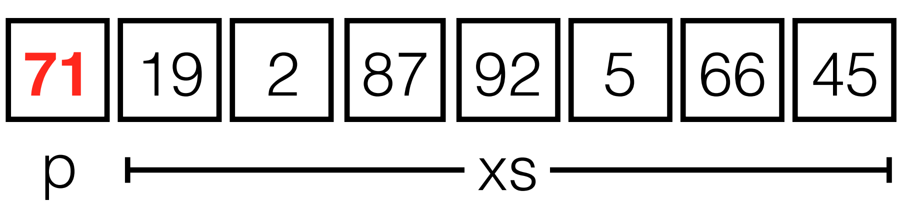


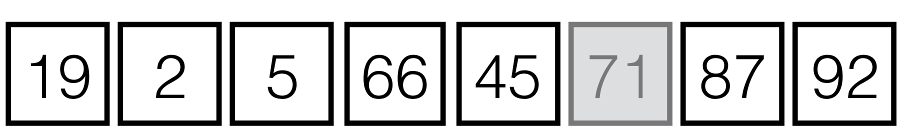
Imperative VS. **declarative**

- What not the how
- *"if you can describe the problem, you have solved the problem" [Brian Tarbox]*
- Higher-level abstraction
- Main benefit:
  - Code is much shorter
  - Less error-prone
- Example: Haskell (functional)

Declarative/functional: Quicksort in Haskell

```haskell
qsort []     = []
qsort (p:xs) = qsort lesser ++ [p] ++ qsort greater
    where
        lesser  = [ y | y <- xs, y < p ]
        greater = [ y | y <- xs, y >= p ]
```





- `[]` is the empty list
- `(p:xs)`
- `[p]` is a list with one element
- `++` concatenation of two list
- `lesser` / `greater` filters the list

Summary Mini-Lecture

- Sorting
  - One of the canonical algorithmic problems in Computer Science
  - Many algorithms exist
  - Examples: Selection Sort (simple), Quicksort (more efficient)
- Programming paradigms
  - Imperative VS. declarative
  - Quicksort in Java VS. Quicksort in Haskell

Further references

- Quicksort in 42 different programming languages:
  https://en.wikibooks.org/wiki/Algorithm_Implementation/Sorting/Quicksort
- Books
  - Knuth, Donald E. (1998), Sorting and Searching, The Art of Computer Programming 3 (2nd ed.), Addison-Wesley.
  - Tate, Bruce A. (2010), Seven Languages in Seven Weeks: A Pragmatic Guide to Learning Programming Languages (1st ed.), Pragmatic Bookshelf.
- Other lectures & **special thanks (!)**
  - Konzepte der Informatik, 2013, Prof. Deussen
    https://streaming.uni-konstanz.de/lectures/wintersemester-2013/inf-11700-20132/
  - Konzepte der Programmierung, 2014, Prof. Scholl
    https://streaming.uni-konstanz.de/lectures/sommersemester-2014/inf-12070-20141/
  - ACM curriculum
    https://www.acm.org/education/CS2013-final-report.pdf

Part 2: Research

Visual Data Analysis: real users, real data, right methods

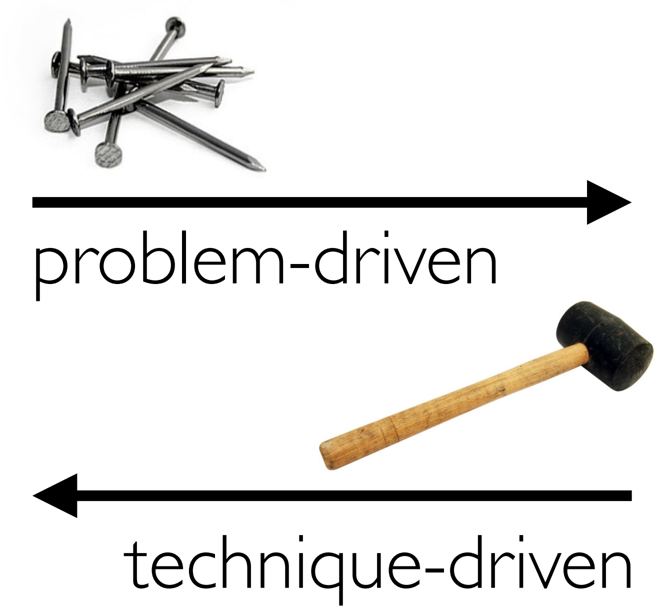
data

has become ubiquitous

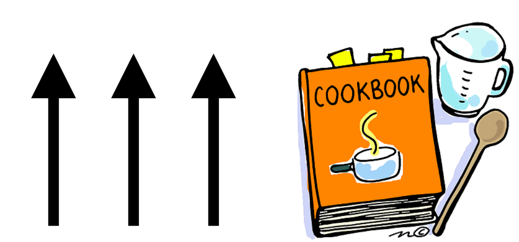
visualization & data analysis

My Focus

- **real users/real data**
  - problem-driven: design studies
  - technique-driven: high-dimensional data analysis

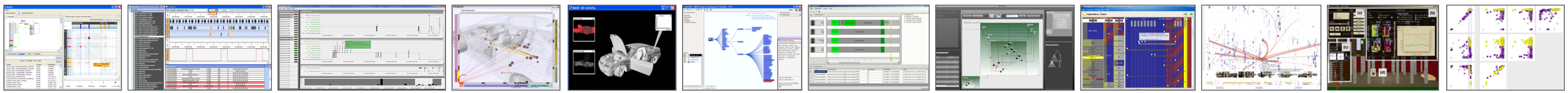


- **right methods**
  - innovate and refine research methods for visual data analysis

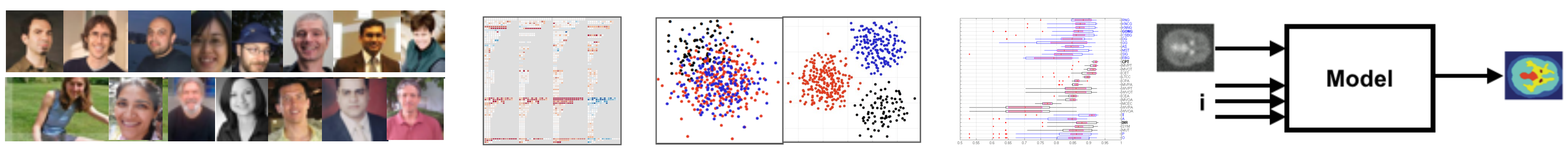


real users/real data:

Applied visualization projects (design studies)

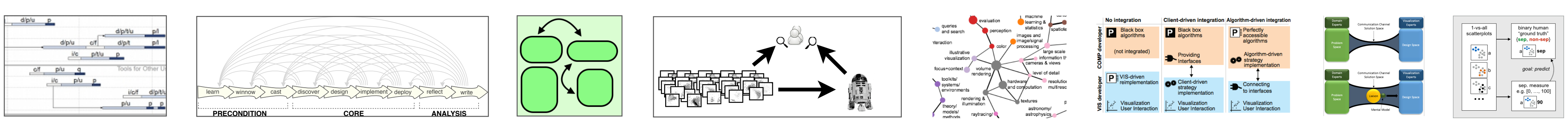


High-dimensional data analysis techniques



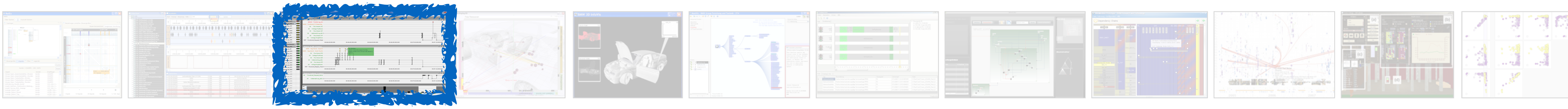
right methods:

Novel and refined research methods



real users/real data

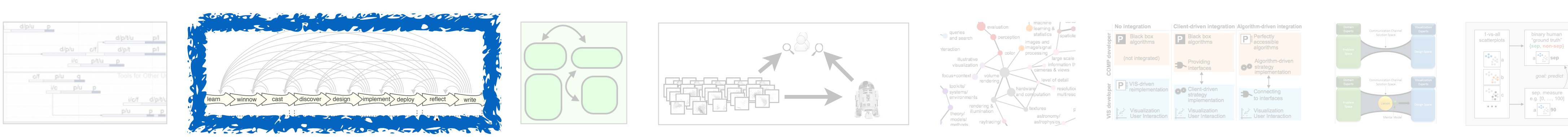
Applied visualization projects (design studies)



High-dimensional data analysis techniques

right methods:

Novel and refined research methods



real users/real data:

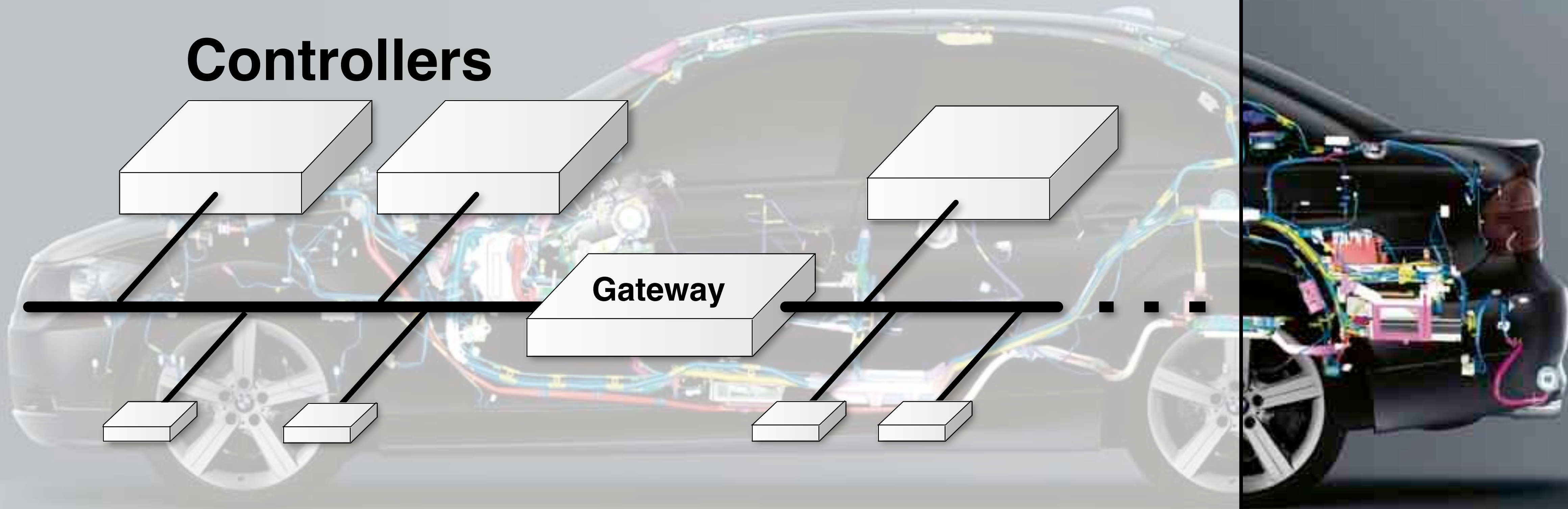
Cardiogram: Visual Analytics for Automotive Engineers

M. Sedlmair, P. Isenberg, D. Baur, M. Mauerer, C. Pigorsch, A. Butz *(CHI 2011).*

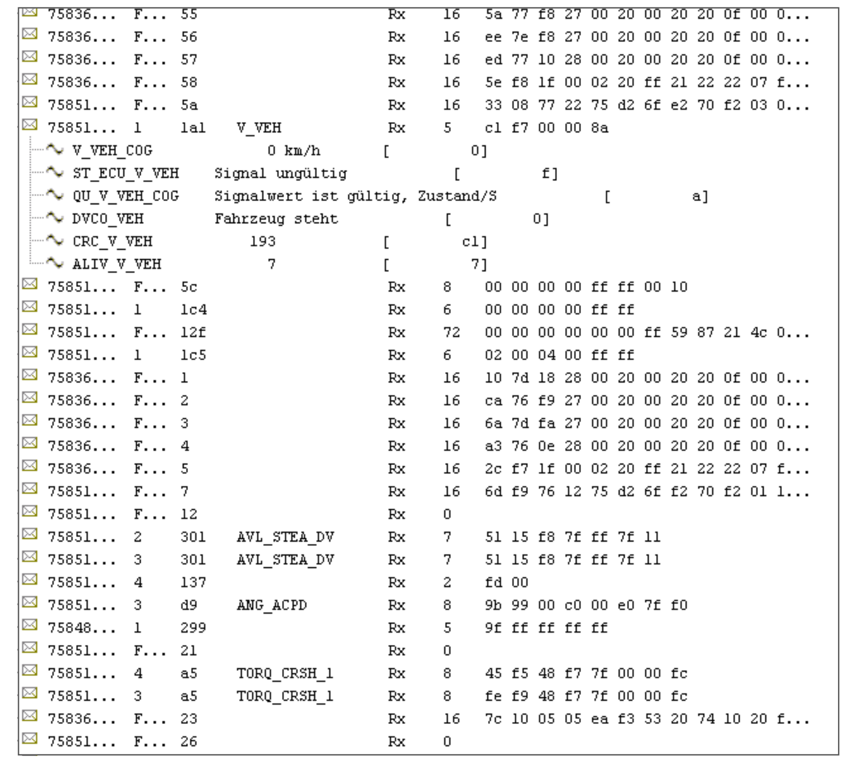
more and more electronics...

... enabled by in-car communication networks



~70 Controllers / Car

Problem Characterization

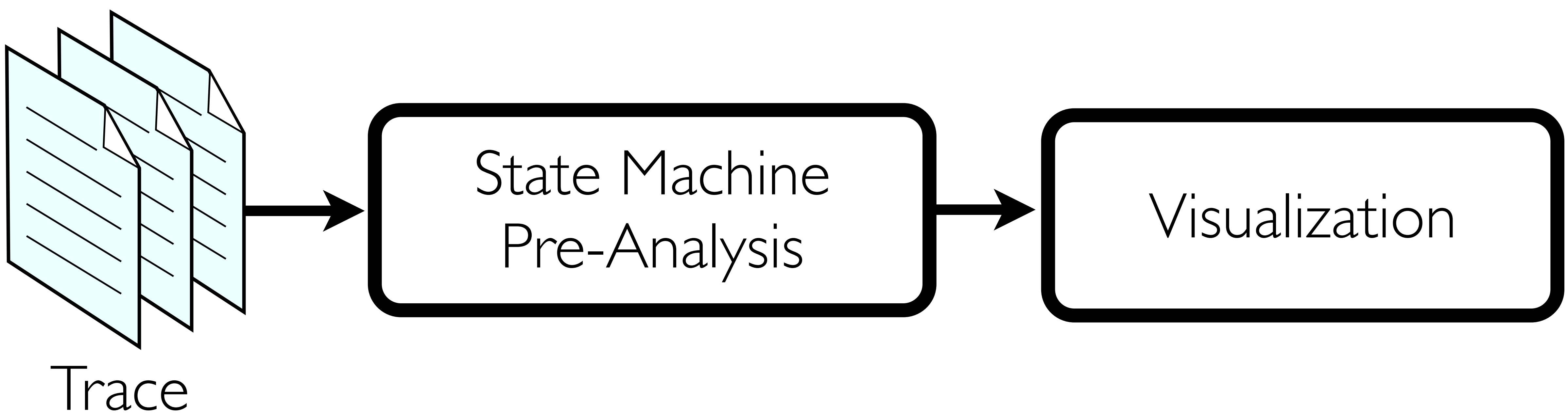


- **data**: recorded traces (15k messages/sec)
- **task**: finding errors
- **process**: test drives
- **current practices**: mainly textual lists

Design Requirements

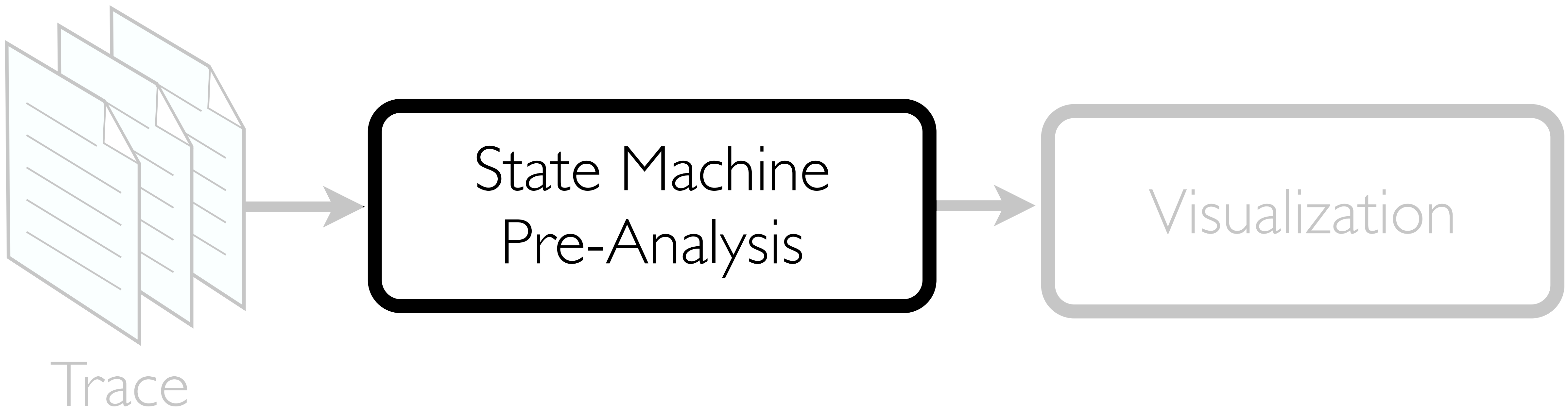
- **Main Challenges**
    - handling masses of test traces (large and many)
    - understanding correlation between trace and car behavior
    - ...

Cardiogram
overview

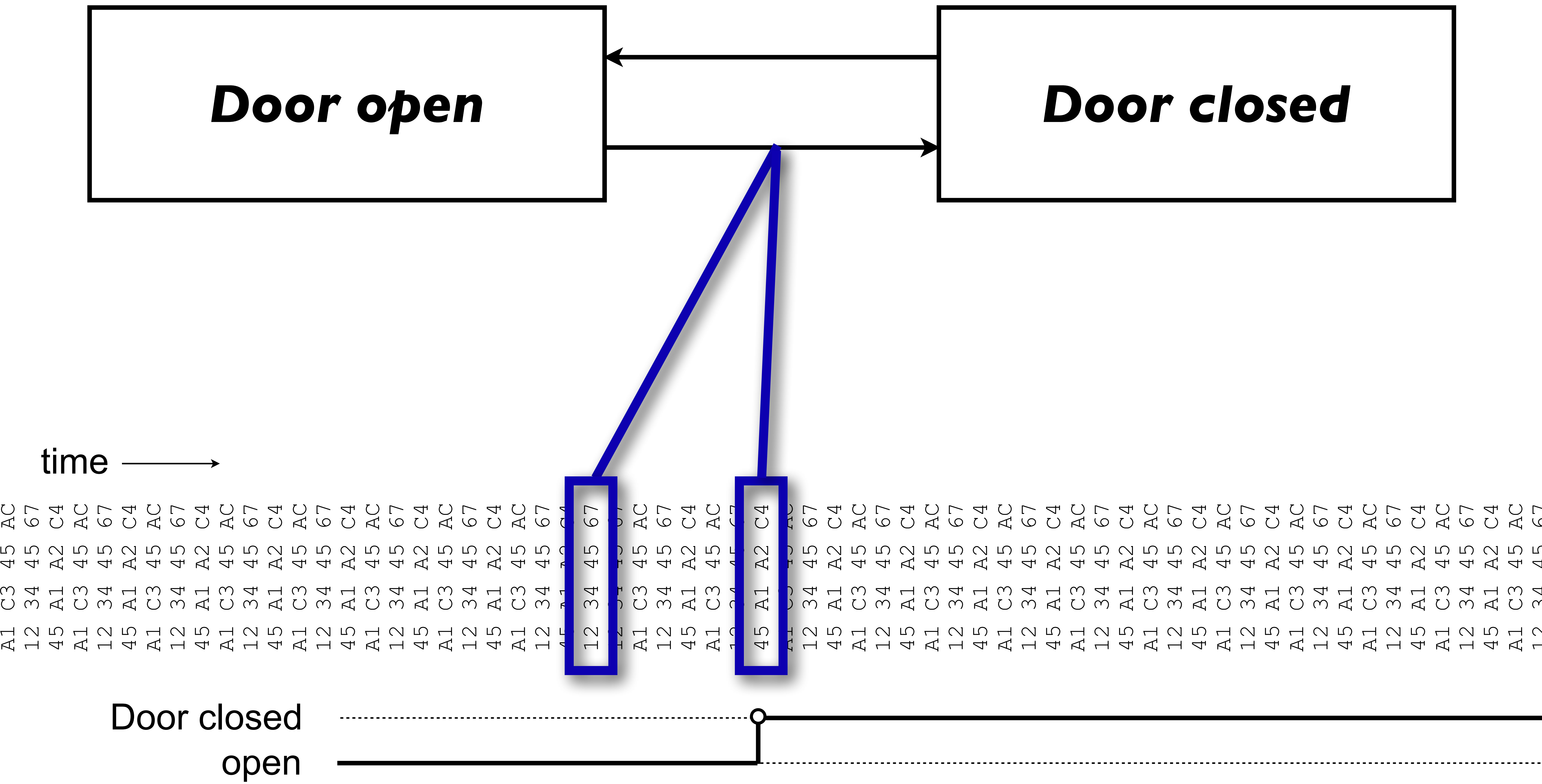


Cardiogram

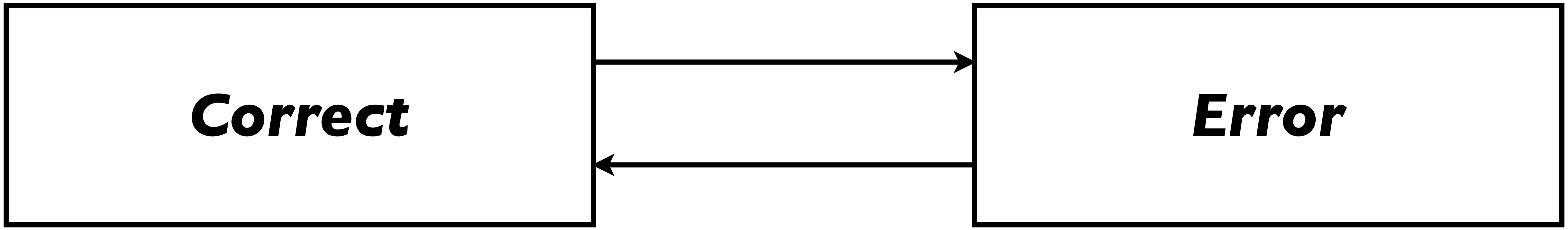
state machine pre-analysis



State Machines: Behavior

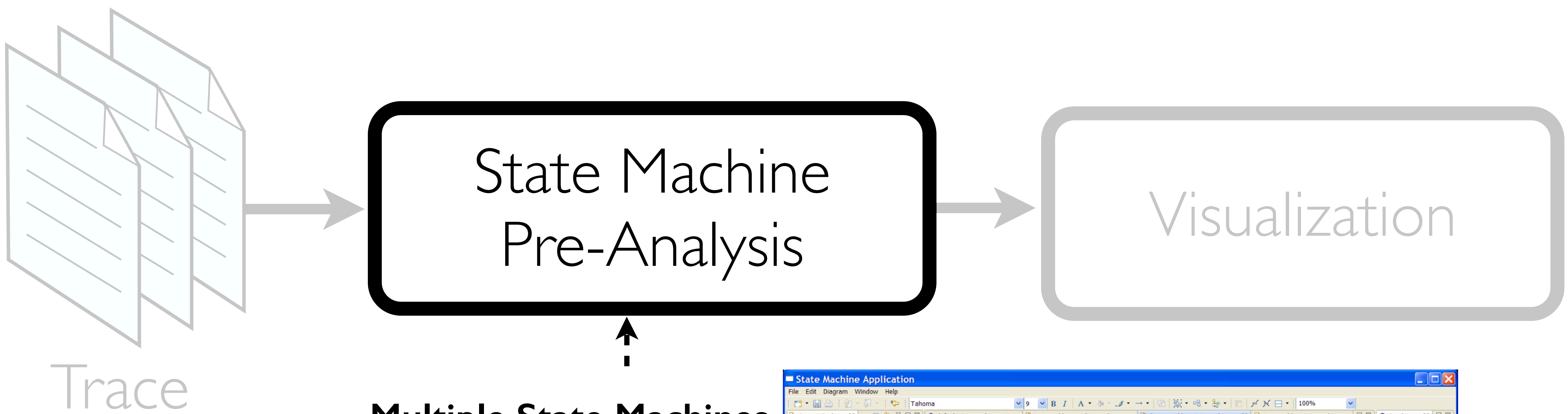


State Machines: Error Detection



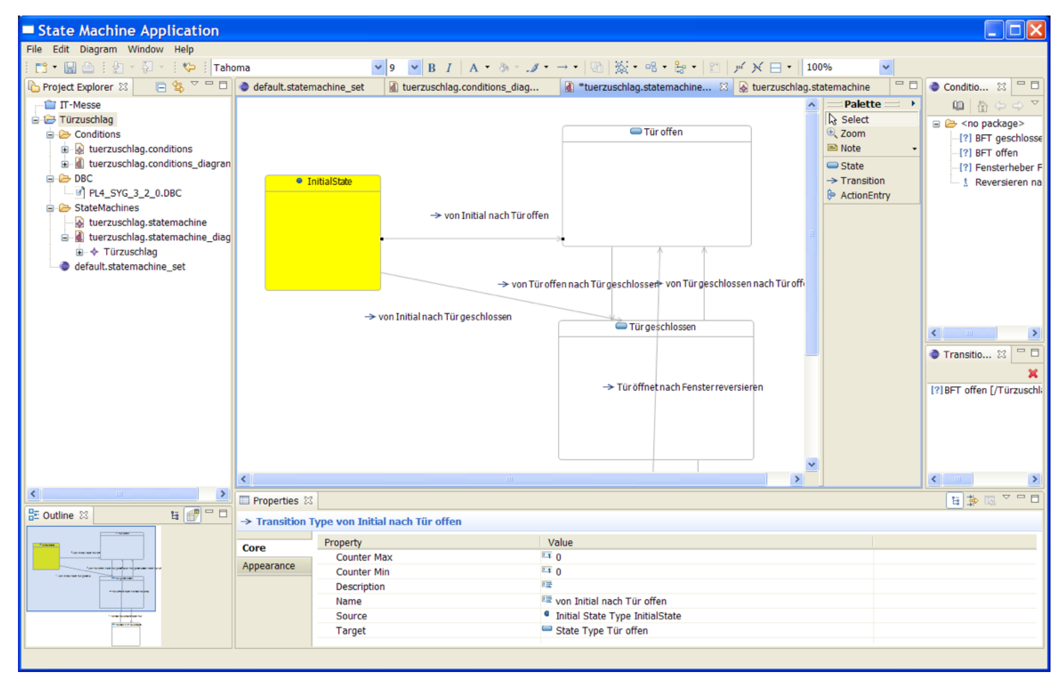
Cardiogram

create/select state machines



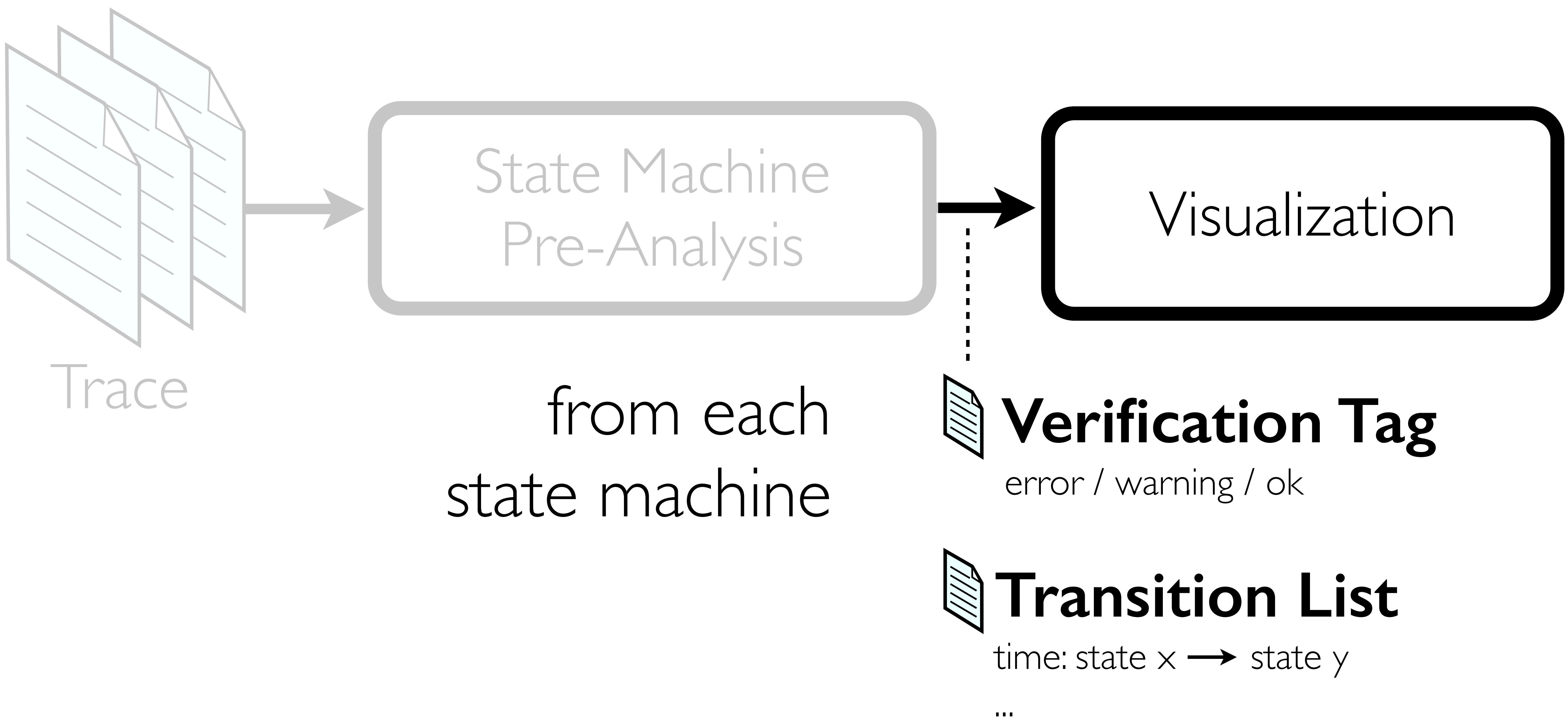
Multiple State Machines

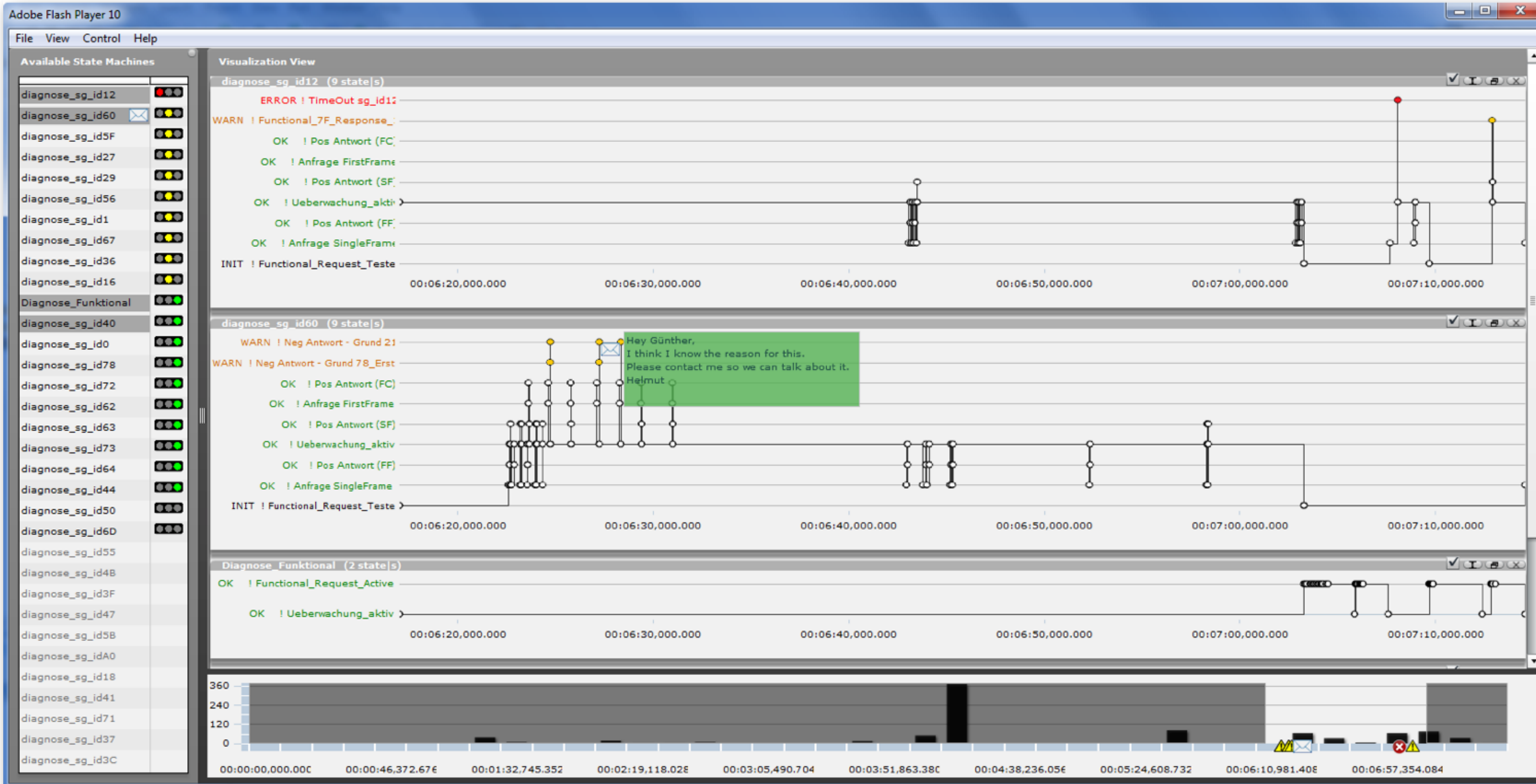
- State Machine 1
- State Machine 2
- State Machine 3
... (dozens)

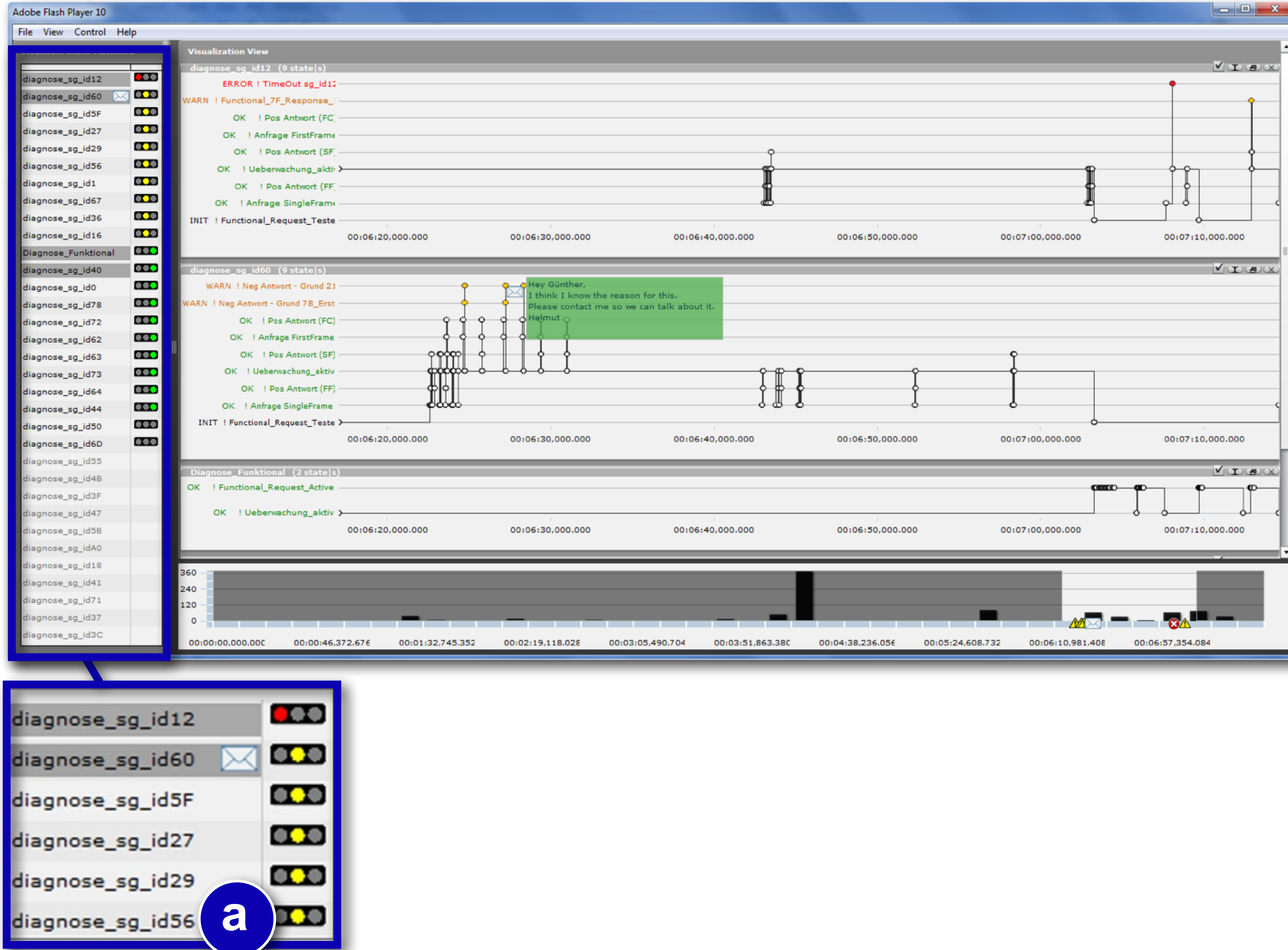


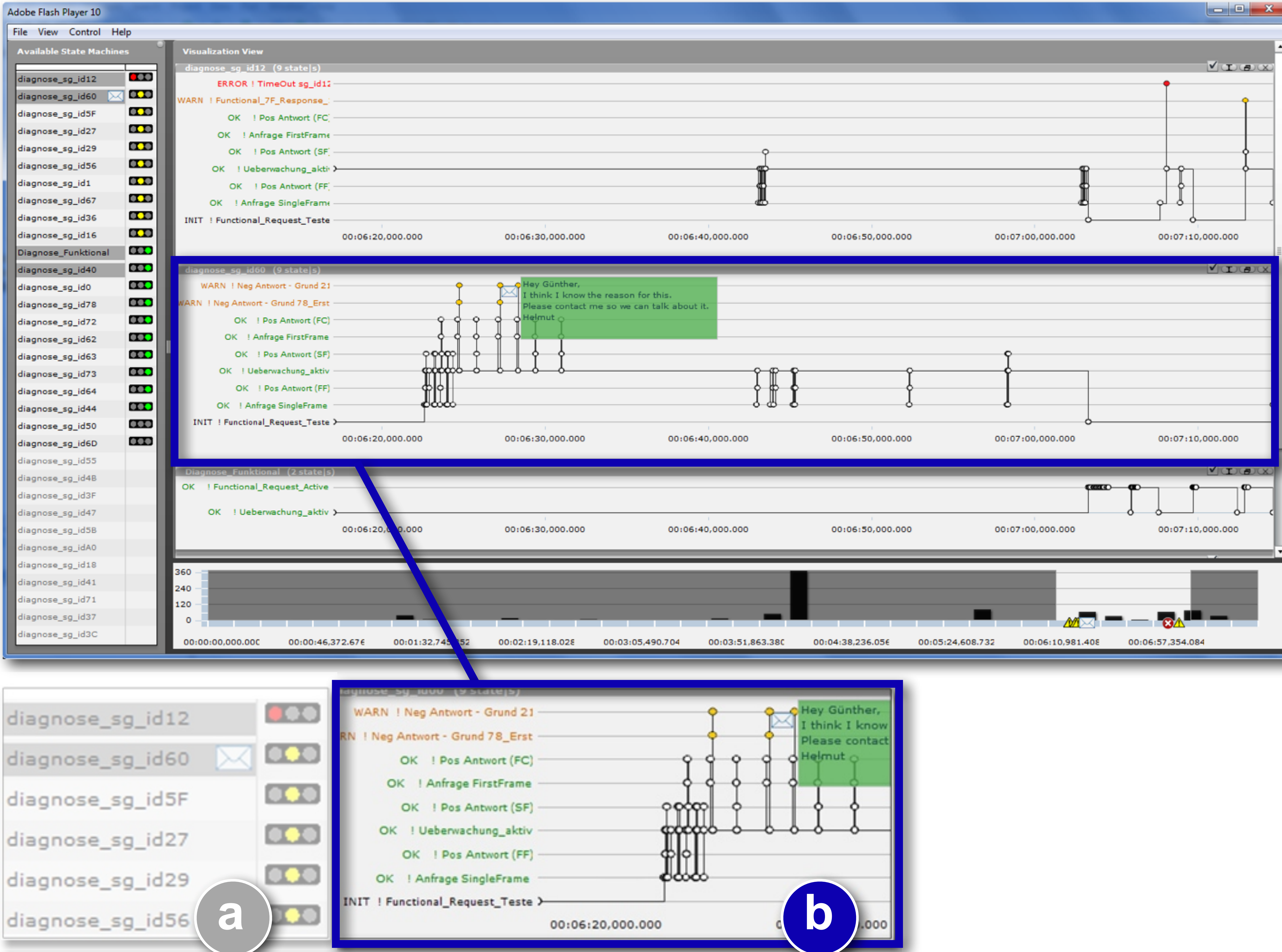
Cardiogram

data abstraction/reduction

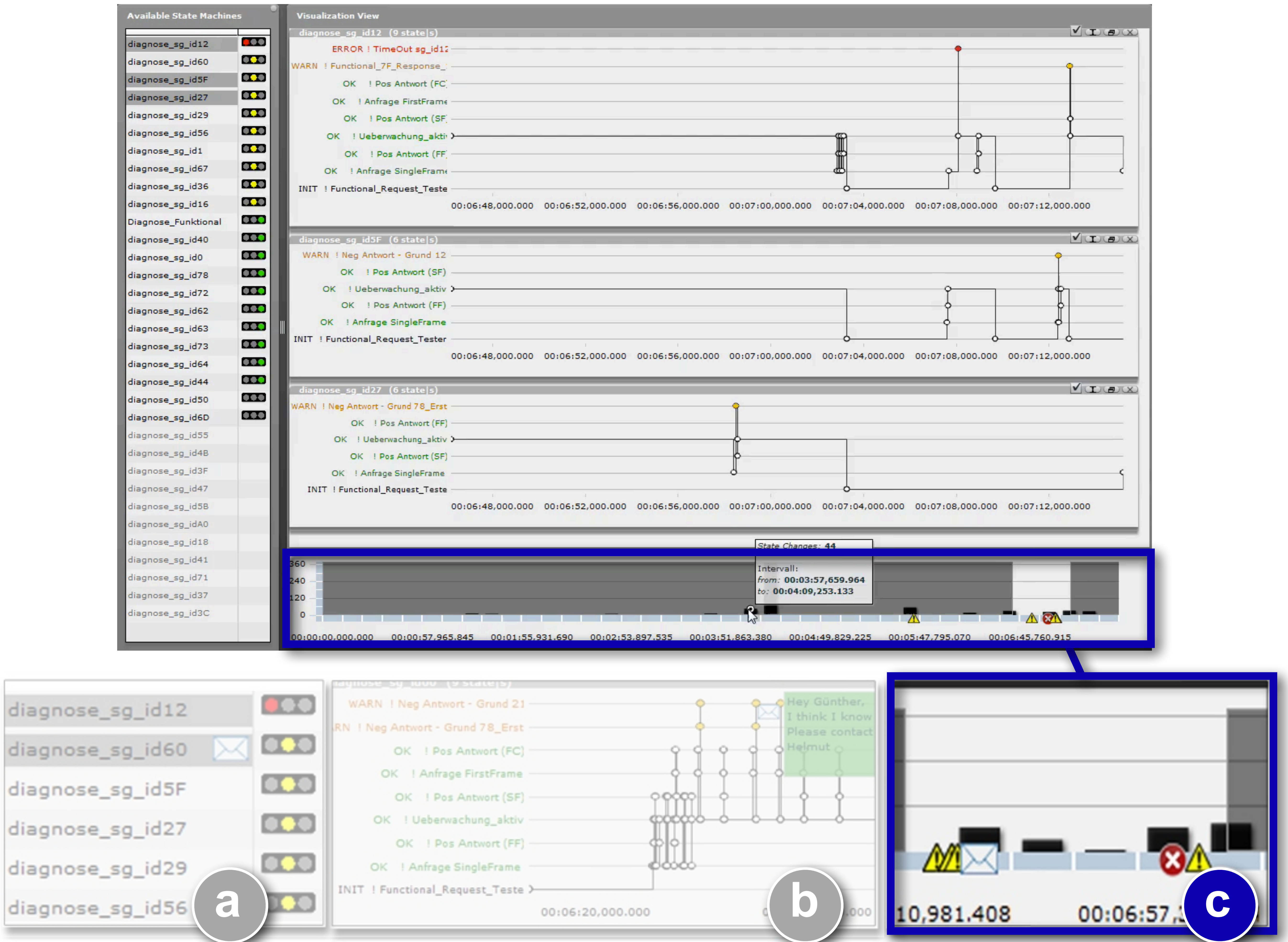



Adobe Flash Player 10
File View Control Help
Available State Machines
diagnose_sg_id12
diagnose_sg_id60
diagnose_sg_id5F
diagnose_sg_id27
diagnose_sg_id29
diagnose_sg_id56
diagnose_sg_id1
diagnose_sg_id67
diagnose_sg_id36
diagnose_sg_id16
Diagnose_Funktional
diagnose_sg_id40
diagnose_sg_id0
diagnose_sg_id78
diagnose_sg_id72
diagnose_sg_id62
diagnose_sg_id63
diagnose_sg_id73
diagnose_sg_id64
diagnose_sg_id44
diagnose_sg_id50
diagnose_sg_id6D
diagnose_sg_id55
diagnose_sg_id4B
diagnose_sg_id3F
diagnose_sg_id47
diagnose_sg_id5B
diagnose_sg_idA0
diagnose_sg_id18
diagnose_sg_id41
diagnose_sg_id71
diagnose_sg_id37
diagnose_sg_id3C
Visualization View
diagnose_sg_id12 (9 state(s)
ERROR ! TimeOut sg_id12
WARN ! Functional_7F_Response_
OK ! Pos Antwort (FC
OK ! Anfrage FirstFrame
OK ! Pos Antwort (SF
OK ! Ueberwachung_akti
OK ! Pos Antwort (FF
OK ! Anfrage SingleFrame
INIT ! Functional_Request_Teste
00:06:20,000.000
00:06:30,000.000
00:06:40,000.000
00:06:50,000.000
00:07:00,000.000
00:07:10,000.000
diagnose_sg_id60 (9 state(s)
WARN ! Neg Antwort - Grund 21
WARN ! Neg Antwort - Grund 78_Erst
OK ! Pos Antwort (FC)
OK ! Anfrage FirstFrame
OK ! Pos Antwort (SF)
OK ! Ueberwachung_aktiv
OK ! Pos Antwort (FF)
OK ! Anfrage SingleFrame
INIT ! Functional_Request_Teste
Hey Günther,
I think I know the reason for this.
Please contact me so we can talk about it.
Helmut
00:06:20,000.000
00:06:30,000.000
00:06:40,000.000
00:06:50,000.000
00:07:00,000.000
00:07:10,000.000
Diagnose_Funktional (2 state(s)
OK ! Functional_Request_Active
OK ! Ueberwachung_aktiv
00:06:20,000.000
00:06:30,000.000
00:06:40,000.000
00:06:50,000.000
00:07:00,000.000
00:07:10,000.000
360
240
120
0
00:00:00,000.000
00:00:46,372.676
00:01:32,745.352
00:02:19,118.028
00:03:05,490.704
00:03:51,863.380
00:04:38,236.056
00:05:24,608.732
00:06:10,981.408
00:06:57,354.084


Adobe Flash Player 10
File View Control Help
Visualization View
diagnose_sg_id12 (9 state(s))
ERROR ! TimeOut sg_id12
WARN ! Functional_7F_Response_
OK ! Pos Antwort (FC)
OK ! Anfrage FirstFrame
OK ! Pos Antwort (SF)
OK ! Ueberwachung_aktiv
OK ! Pos Antwort (FF)
OK ! Anfrage SingleFrame
INIT ! Functional_Request_Teste
00:06:20,000.000
00:06:30,000.000
00:06:40,000.000
00:06:50,000.000
00:07:00,000.000
00:07:10,000.000
diagnose_sg_id60 (9 state(s))
WARN ! Neg Antwort - Grund 21
WARN ! Neg Antwort - Grund 78_Erst
Hey Günther,
I think I know the reason for this.
Please contact me so we can talk about it.
Helmut
Diagnose_Funktional (2 state(s))
OK ! Functional_Request_Active
OK ! Ueberwachung_aktiv
360
240
120
0
00:00:00,000.000
00:00:46,372.676
00:01:32,745.352
00:02:19,118.028
00:03:05,490.704
00:03:51,863.380
00:04:38,236.056
00:05:24,608.732
00:06:10,981.408
00:06:57,354.084
diagnose_sg_id12
diagnose_sg_id60
diagnose_sg_id5F
diagnose_sg_id27
diagnose_sg_id29
diagnose_sg_id56
diagnose_sg_id1
diagnose_sg_id67
diagnose_sg_id36
diagnose_sg_id16
Diagnose_Funktional
diagnose_sg_id40
diagnose_sg_id0
diagnose_sg_id78
diagnose_sg_id72
diagnose_sg_id62
diagnose_sg_id63
diagnose_sg_id73
diagnose_sg_id64
diagnose_sg_id44
diagnose_sg_id50
diagnose_sg_id6D
diagnose_sg_id55
diagnose_sg_id4B
diagnose_sg_id3F
diagnose_sg_id47
diagnose_sg_id5B
diagnose_sg_idA0
diagnose_sg_id18
diagnose_sg_id41
diagnose_sg_id71
diagnose_sg_id37
diagnose_sg_id3C
a



Adobe Flash Player 10
File View Control Help
Available State Machines
Visualization View
diagnose_sg_id12
diagnose_sg_id60
diagnose_sg_id5F
diagnose_sg_id27
diagnose_sg_id29
diagnose_sg_id56
diagnose_sg_id1
diagnose_sg_id67
diagnose_sg_id36
diagnose_sg_id16
Diagnose_Funktional
diagnose_sg_id40
diagnose_sg_id0
diagnose_sg_id78
diagnose_sg_id72
diagnose_sg_id62
diagnose_sg_id63
diagnose_sg_id73
diagnose_sg_id64
diagnose_sg_id44
diagnose_sg_id50
diagnose_sg_id6D
diagnose_sg_id55
diagnose_sg_id4B
diagnose_sg_id3F
diagnose_sg_id47
diagnose_sg_id5B
diagnose_sg_idA0
diagnose_sg_id18
diagnose_sg_id41
diagnose_sg_id71
diagnose_sg_id37
diagnose_sg_id3C
diagnose_sg_id12 (9 state(s))
ERROR ! TimeOut sg_id12
WARN ! Functional_7F_Response_
OK ! Pos Antwort (FC)
OK ! Anfrage FirstFrame
OK ! Pos Antwort (SF)
OK ! Ueberwachung_aktiv
OK ! Pos Antwort (FF)
OK ! Anfrage SingleFrame
INIT ! Functional_Request_Teste
00:06:20,000.000
00:06:30,000.000
00:06:40,000.000
00:06:50,000.000
00:07:00,000.000
00:07:10,000.000
diagnose_sg_id60 (9 state(s))
WARN ! Neg Antwort - Grund 21
WARN ! Neg Antwort - Grund 78_Erst
Hey Günther,
I think I know the reason for this.
Please contact me so we can talk about it.
Helmut
Diagnose_Funktional (2 state(s))
OK ! Functional_Request_Active
OK ! Ueberwachung_aktiv
360
240
120
0
00:00:00,000.000
00:00:46,372.676
00:01:32,745.352
00:02:19,118.028
00:03:05,490.704
00:03:51,863.380
00:04:38,236.056
00:05:24,608.732
00:06:10,981.408
00:06:57,354.084
a
b


Available State Machines
Visualization View
diagnose_sg_id12
diagnose_sg_id60
diagnose_sg_id5F
diagnose_sg_id27
diagnose_sg_id29
diagnose_sg_id56
diagnose_sg_id1
diagnose_sg_id67
diagnose_sg_id36
diagnose_sg_id16
Diagnose_Funktional
diagnose_sg_id40
diagnose_sg_id0
diagnose_sg_id78
diagnose_sg_id72
diagnose_sg_id62
diagnose_sg_id63
diagnose_sg_id73
diagnose_sg_id64
diagnose_sg_id44
diagnose_sg_id50
diagnose_sg_id6D
diagnose_sg_id55
diagnose_sg_id4B
diagnose_sg_id3F
diagnose_sg_id47
diagnose_sg_id5B
diagnose_sg_idA0
diagnose_sg_id18
diagnose_sg_id41
diagnose_sg_id71
diagnose_sg_id37
diagnose_sg_id3C
diagnose_sg_id12 (9 state(s))
ERROR ! TimeOut sg_id12
WARN ! Functional_7F_Response_
OK ! Pos Antwort (FC)
OK ! Anfrage FirstFrame
OK ! Pos Antwort (SF)
OK ! Ueberwachung_aktiv
OK ! Pos Antwort (FF)
OK ! Anfrage SingleFrame
INIT ! Functional_Request_Tester
00:06:48,000.000 00:06:52,000.000 00:06:56,000.000 00:07:00,000.000 00:07:04,000.000 00:07:08,000.000 00:07:12,000.000
diagnose_sg_id5F (6 state(s))
WARN ! Neg Antwort - Grund 12
OK ! Pos Antwort (SF)
OK ! Ueberwachung_aktiv
OK ! Pos Antwort (FF)
OK ! Anfrage SingleFrame
INIT ! Functional_Request_Tester
diagnose_sg_id27 (6 state(s))
WARN ! Neg Antwort - Grund 78_Erst
OK ! Pos Antwort (FF)
OK ! Ueberwachung_aktiv
OK ! Pos Antwort (SF)
OK ! Anfrage SingleFrame
INIT ! Functional_Request_Teste
State Changes: 44
Intervall:
from: 00:03:57,659.964
to: 00:04:09,253.133
360
240
120
0
00:00:00,000.000 00:00:57,965.845 00:01:55,931.690 00:02:53,897.535 00:03:51,863.380 00:04:49,829.225 00:05:47,795.070 00:06:45,760.915
WARN ! Neg Antwort - Grund 21
WARN ! Neg Antwort - Grund 78_Erst
OK ! Pos Antwort (FC)
OK ! Anfrage FirstFrame
OK ! Pos Antwort (SF)
OK ! Ueberwachung_aktiv
OK ! Pos Antwort (FF)
OK ! Anfrage SingleFrame
INIT ! Functional_Request_Teste
00:06:20,000.000
Hey Günther,
I think I know
Please contact
Helmut
10,981.408
a
b
c

Longitudinal field study

1 year, 15 engineers

Longitudinal field study

results

- externalization and sharing of expert knowledge

Longitudinal field study

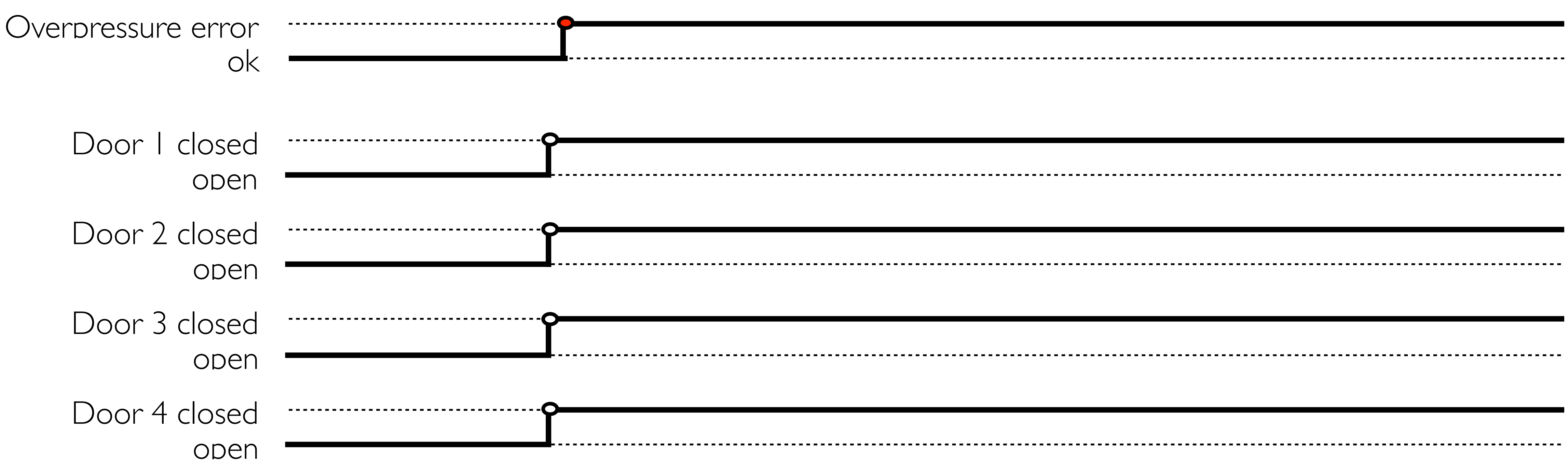
results

- externalization and sharing of expert knowledge
- complete coverage of traces vs. sparse sampling

Longitudinal field study

results

- externalization and sharing of expert knowledge
- complete coverage of traces vs. sparse sampling
- understand behavioral correlations



Longitudinal field study

results

- externalization and sharing of expert knowledge
- complete coverage of traces vs. sparse sampling
- understand behavioral correlations

deployed and adopted

how to?

“A **design study** is a project in which visualization researchers analyze a specific real-world problem faced by domain experts, design a visualization system that supports solving this problem, validate the design, and reflect about lessons learned in order to refine visualization design guidelines.”

[Sedlmair et al.: Design Study Methodology: Reflections from the Trenches and the Stacks, InfoVis 2012]

design studies:

long and winding road with many pitfalls!

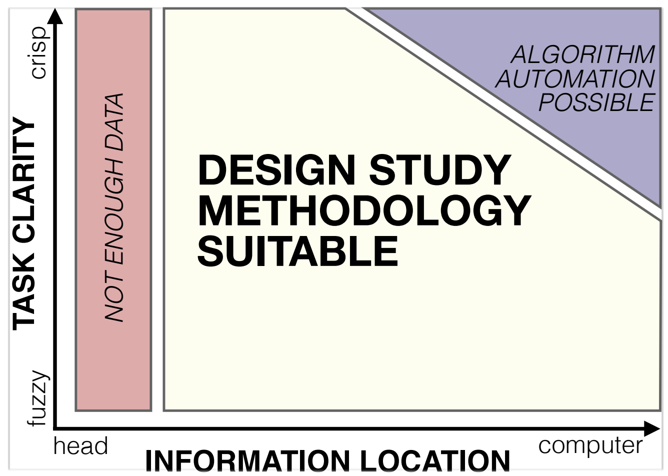
right methods:

Design Study Methodology: Reflection from the Trenches and the Stacks

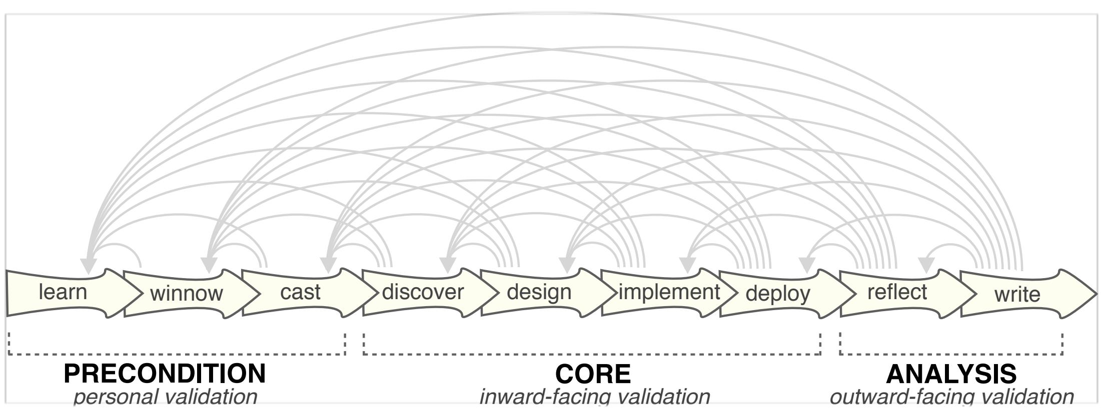
M. Sedlmair, M. Meyer, T. Munzner.
TVCG (Proc. InfoVis 2012). **Best paper honorable mention.**



main contributions
guiding other researchers with ...



Definitions



9-stage framework

PF-1	premature advance: jumping forward over stages	general
PF-2	premature start: insufficient knowledge of vis literature	learn
PF-3	premature commitment: collaboration with wrong people	winnow
PF-4	no real data available (yet)	winnow
PF-5	insufficient time available from potential collaborators	winnow
PF-6	no need for visualization: problem can be automated	winnow
PF-7	researcher expertise does not match domain problem	winnow
PF-8	no need for research: engineering vs. research project	winnow
PF-9	no need for change: existing tools are good enough	winnow

32 pitfalls

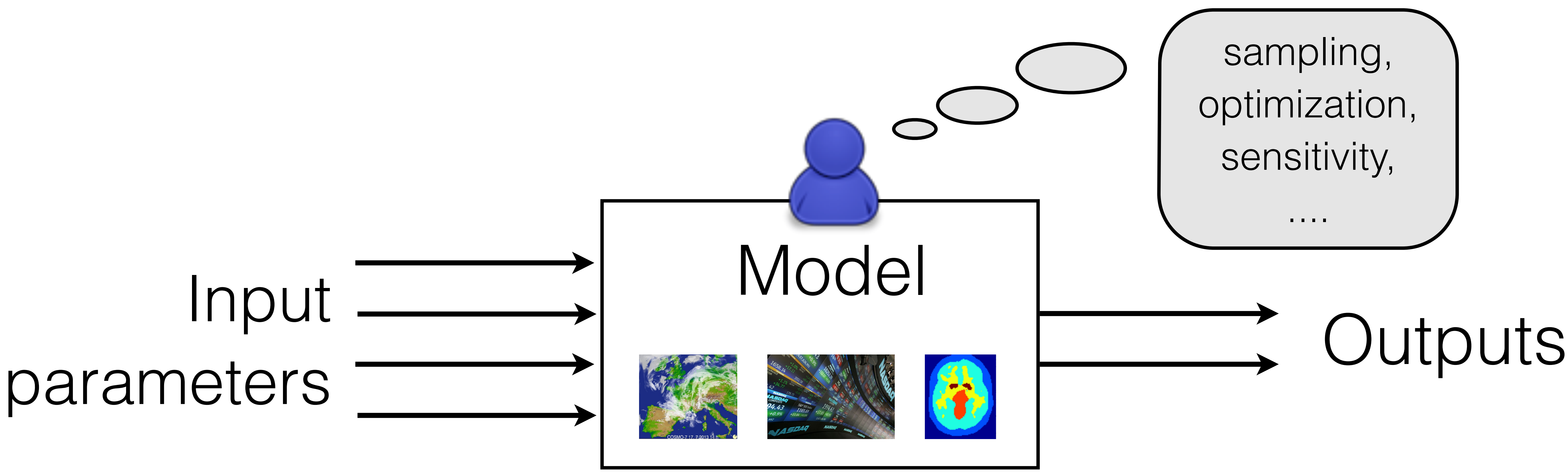
future work

short-term

1. Design studies

- real users/real data:
  - contribute to their fields
  - improve our guidelines
- right methods:
  - validate, improve, extend design study methodology

Design studies:
Visual Parameter Space Analysis



Sedlmair, *Heinzl, Bruckner, Piringer, Möller: Visual Parameter Space Analysis: A Conceptual Framework, InfoVis 2014*

Bergner, ***Sedlmair****, Möller, Abdolyousef, Saad: ParaGlide: Interactive Parameter Space Partitioning for Computer Simulations, IEEE TVCG, 2013*

Design studies
Data-driven Journalism

- *users:* journalists
- *tasks:* story creation
- *data:* text over time, e.g., parliament speeches

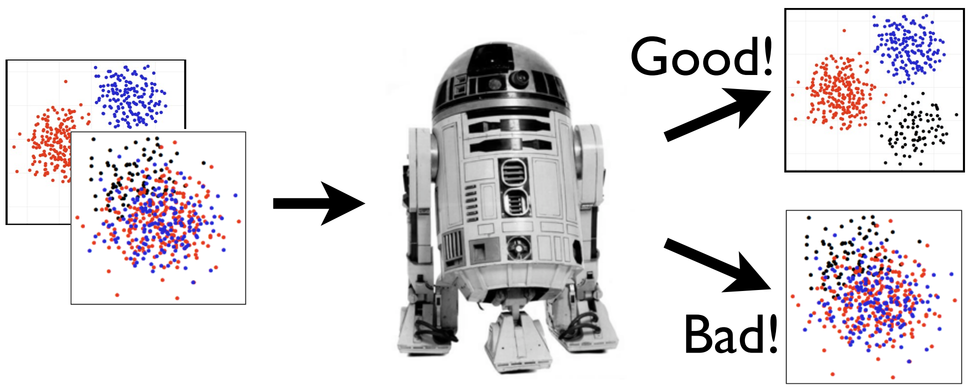
FFG-funded VALiD project (690.000 Euro)
*Aigner (PI), **Sedlmair (Co-PI),** Ausserhofer (Co-PI), Popper (Co-PI)*

2. High-dim data analysis techniques

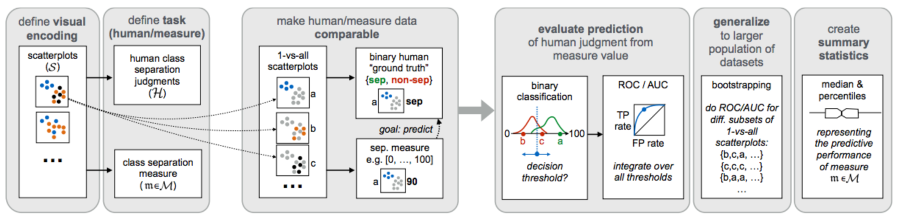
- real users/real data:
  - understand techniques in real circumstances
  - build better techniques
- right method
  - combine approaches from HCI, Vis, Psychology, and Machine Learning

High-dim techniques: **visual quality measures**

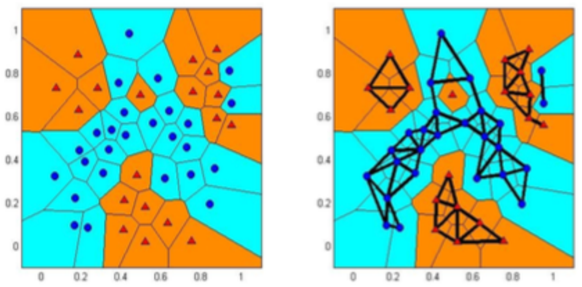
- real users/real data
  *[**Sedlmair**, Tatu, Munzner, Tory: A Taxonomy of Visual Cluster Separation Factors, EuroVis 2012]*



- right method
  *[**Sedlmair**, Aupetit, Data-driven Evaluation of Visual Quality Measures, EuroVis 2015]*



- new techniques
  *[Aupetit, **Sedlmair**: Learning human visual judgment of class separation in colored scatterplots for Visual Analytics, NIPS 2015 - in submission]*



High-dim techniques:
other ...

- clustering
- dimension reduction
- sensitivity analysis
- …

future work

long-term

Vision:

The “cookbook” for Visual Data Analysis

abstract data & task

↓

data analysis technique

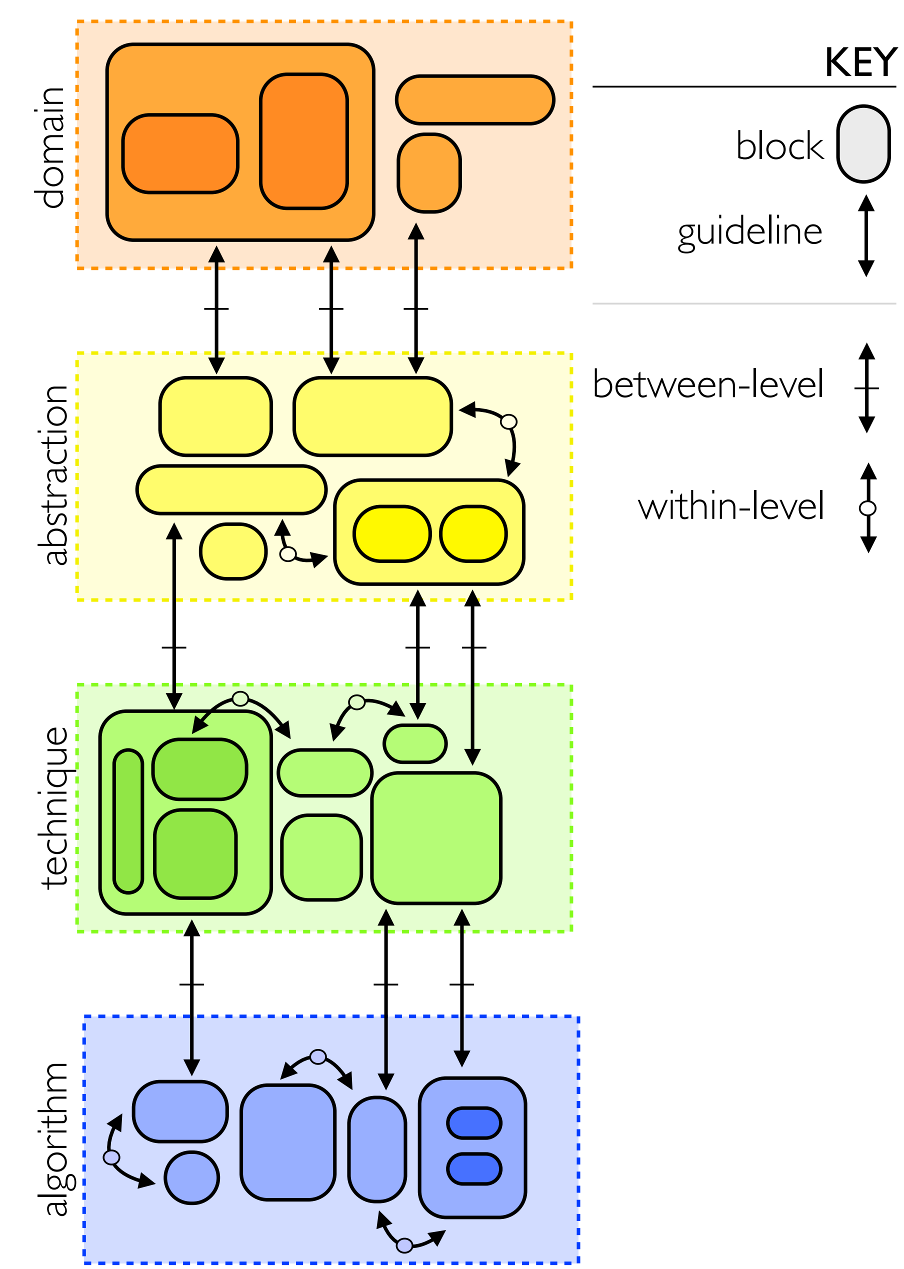
The “cookbook” for Visual Data Analysis

blocks and guidelines

*Meyer, **Sedlmair**, Quinan, Munzner:*
The Nested Blocks and Guidelines Model,
IVS Journal 2013.

design decision patterns

ongoing project:
with Heidi Lam & Miriah Meyer



The "cookbook" for Visual Data Analysis

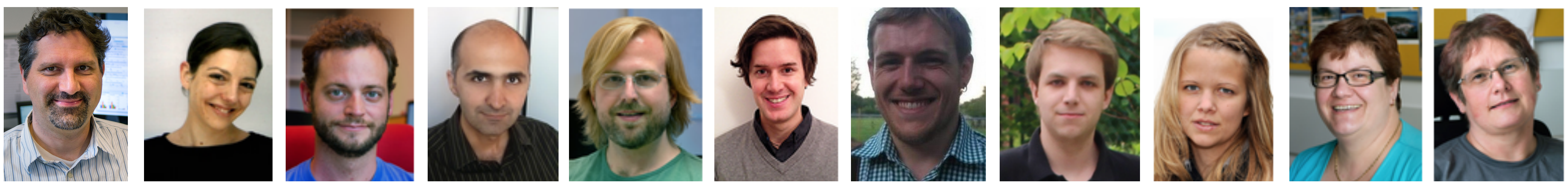
natural science
vs. design research

vast amounts of
real users/real data
work necessary

[Sir Francis Bacon 1561-1626]

Thanks to you & to …

VDA Group @ University of Vienna



Many, many other collaborators

…

Questions?

On CS Undergrad Teaching and Visualization Research

1. Sorting in different programming paradigms

2. Visual Data Analysis: real users, real data, right methods

Michael Sedlmair

Visualization and Data Analysis group (VDA)

universität wien