COMPUTATION WORLD 2012

FUTURE COMPUTING / COMPUTATION TOOLS PANEL

EMERGENT COMPUTING PARADIGMS AND THEIR THEORETICAL AND PRACTICAL SUPPORT TOOLS

NICE, 26.7.12

PANELLISTS

Moderator:

Dietmar Fey, University Erlangen-Nuremberg, Germany

Glenn Luecke, Iowa State University, USA

Torsten Ullrich, Fraunhofer Research Centre, Graz, Austria

Daniel Hulme, University College London, UK

Valeriya Gribova, Russian Academy of Sciences, Nowosibirsk, Russia

RESULTS OF PANEL DISCUSSION

- Not only architectures (multi-core, GPU and FPGA accelerators) will become more heterogenous than they are already

- Also tools need diversification due to diverse application scenarios
  - Most abstract levels:
    Expert systems need ontology programming and appropriate tools
  - Less abstract level:
    SaaS (Software as a Service) needs virtualization and appropriate tools for developing service and client
  - More hardware-oriented level:
    Tools have to support optimization more than now
    Automatic transfer of code sequences in optimized structures
    May be autotining is an answer for that sophisticated task
  - Stronger hardware-orientated level (HPC applications):
    The performance to achieve is everything and therefore adequate tools are necessary
  - Hardware architecture level:
    Future Nanotechnology requires tools that support resiliency on different levels (analgoue, digital and system level)

Need Easy-To-Use, Automatic Tools for Error Detection and Performance Assessment

Glenn Luecke
Professor of Mathematics
Director, High Performance Computing Group
Iowa State University, Ames, Iowa, USA
July 26, 2012

Purpose of Tools

- aid expert and non-expert programmers to quickly correct program errors & to develop fast, high performance programs
- tools depend on the programming model used

Current and Future Programming Paradigms

- MPI with Fortran, C/C++
- OpenMP, OpenMP with MPI
- CUDA, CUDA with MPI, CUDA with OpenMP & M
- OpenCL
- OpenACC
- Unified Parallel C, Co-Array Fortran
- Chapel (Cray)
- X10 (IBM)
- Fortress (Oracle) – project stopped last week

Tool Design

- easy-to-use
- low CPU and memory overhead
- scalable
- messages issued must be accurate and contain information needed to easily fix both functional and performance problems identified

Future technological development and its implication for processor architectures.

CONTRIBUTION DIETMAR FEY

- Chair for Computer Architecture
  Friedrich-Alexander-University Erlangen-Nuremberg, Germany

COMPUTING PARADIGM FOR (MIDTERM) FUTURE CIRCUITS

- Use VLSI photonics for communication
- Use memristor for storing
- Use CMOS for processing

STILL CMOS?

Keep aware of nanotechnology / nanocomputing

- Memcomp (Memristor for Computing)
- Quantum Cellular Automata
- Carbon Nano Tube FET

- Memcomp (Memristor for Computing)

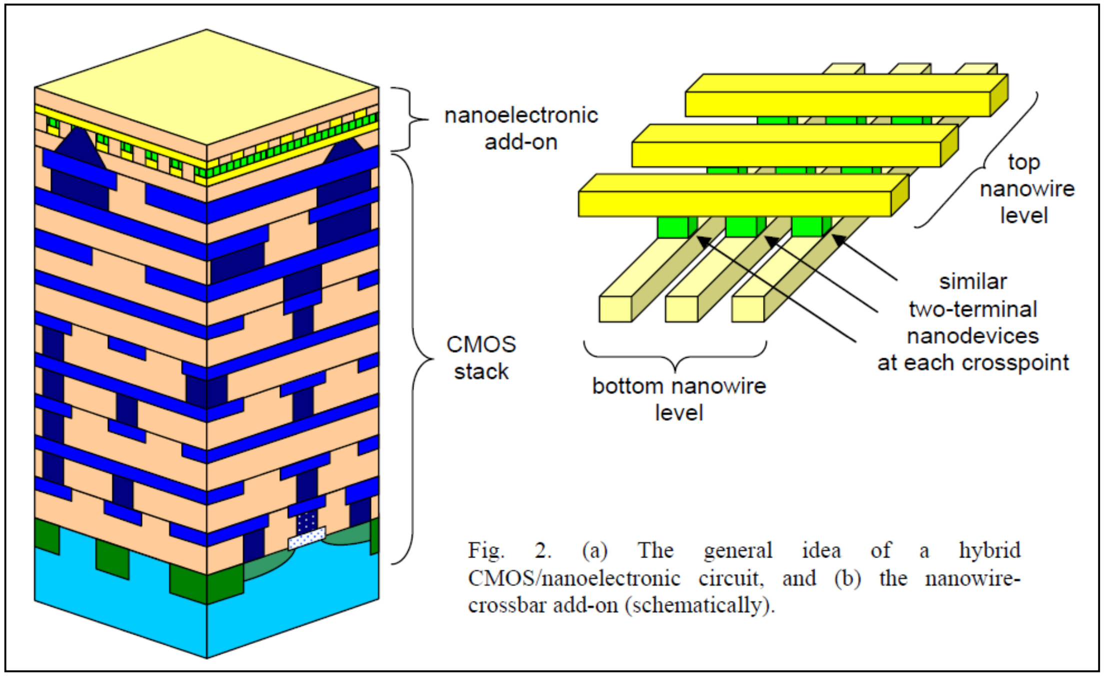
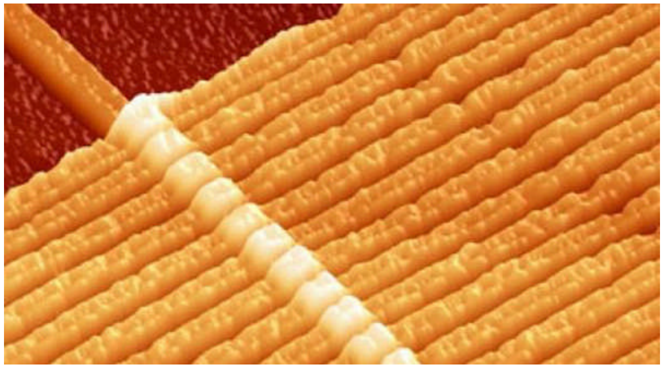


Fig. 2. (a) The general idea of a hybrid CMOS/nanoelectronic circuit, and (b) the nanowire-crossbar add-on (schematically).



IMPLICATIONS ON TOOLS

- Trend will continue
  - Higher frequencies and very aggressive dynamic instruction scheduling techniques are abandoned
  - Replaced by many simpler cores
- Consequences for manufacturability and dependability
- Resiliency fundamental for next-generation systems
  - Reliable systems based on unreliable but high-dense integrated devices

IMPLICATIONS ON ARCHITECTURES AND TOOLS

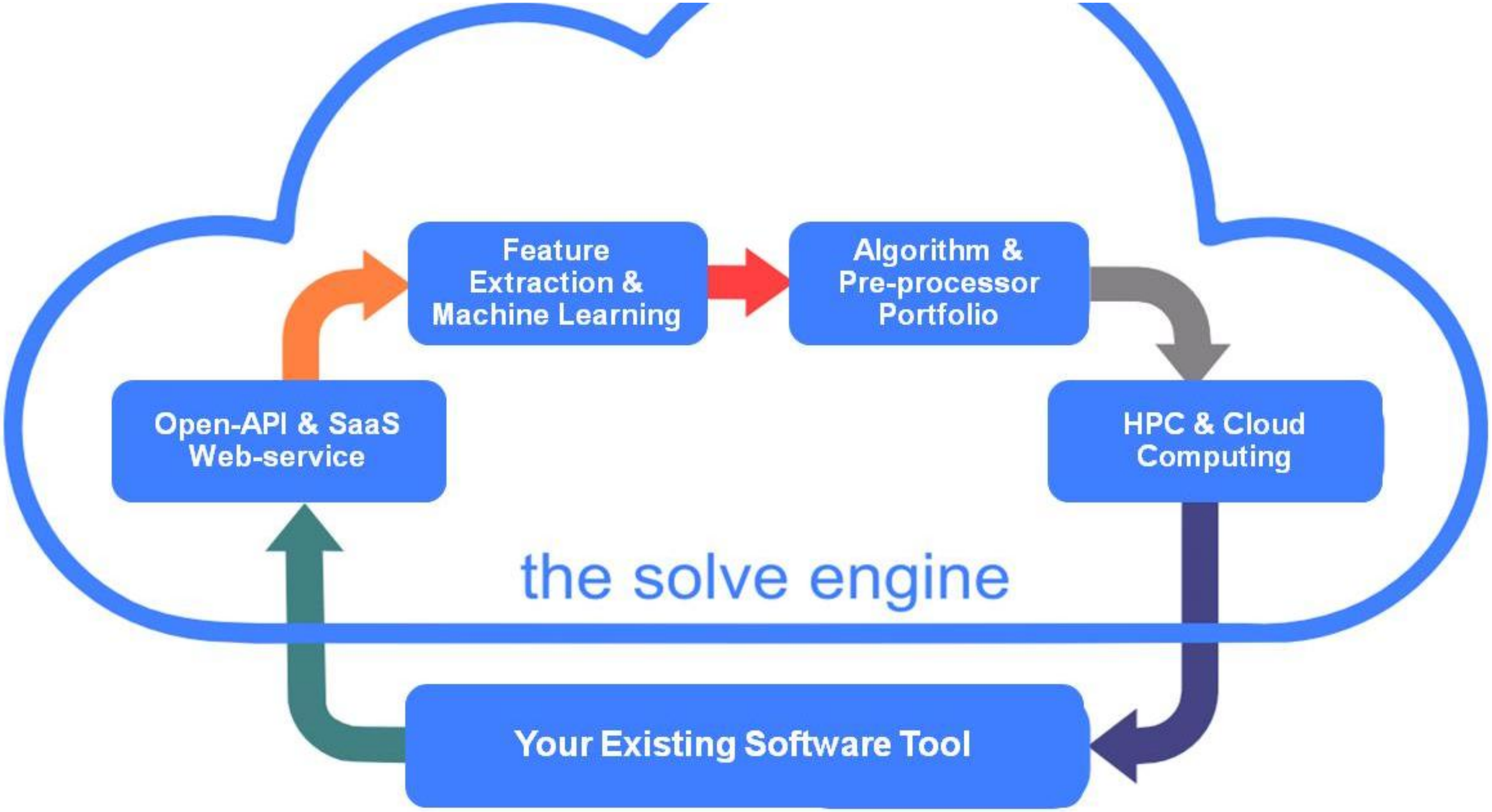
Nanocomputing requires a cross-layer approach

- Technology
- Circuit
- Architectural

Resiliency on different levels is required

- Technological
  - Analogue circuits that observe digital circuits
- Circuit
  - Redundant coding schemes
- Architecture
  - Self-reconfiguration
  - Switching off fault and switching on unused cores

NP-Complete Project



Dr. Daniel Hulme, UCL (University College London)

Computation Tools Panel - ComputationWorld 2012

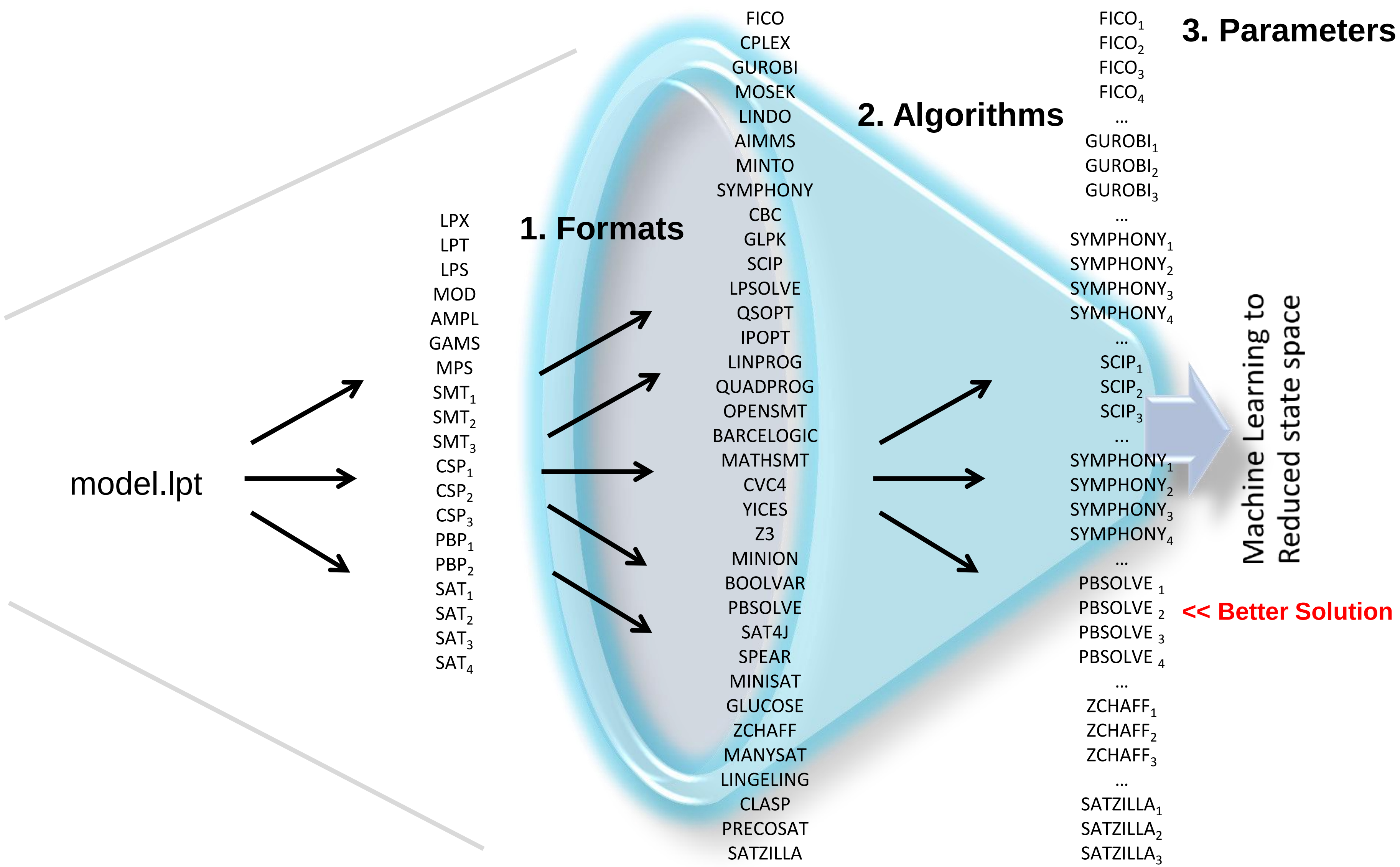
Current Solving Process

ILOG → model.lpt → CPLEX + vanilla parameters → Solution

or

Bespoke Model → Bespoke Algorithm → Solution

New Solving Process



Cloud-based Solving

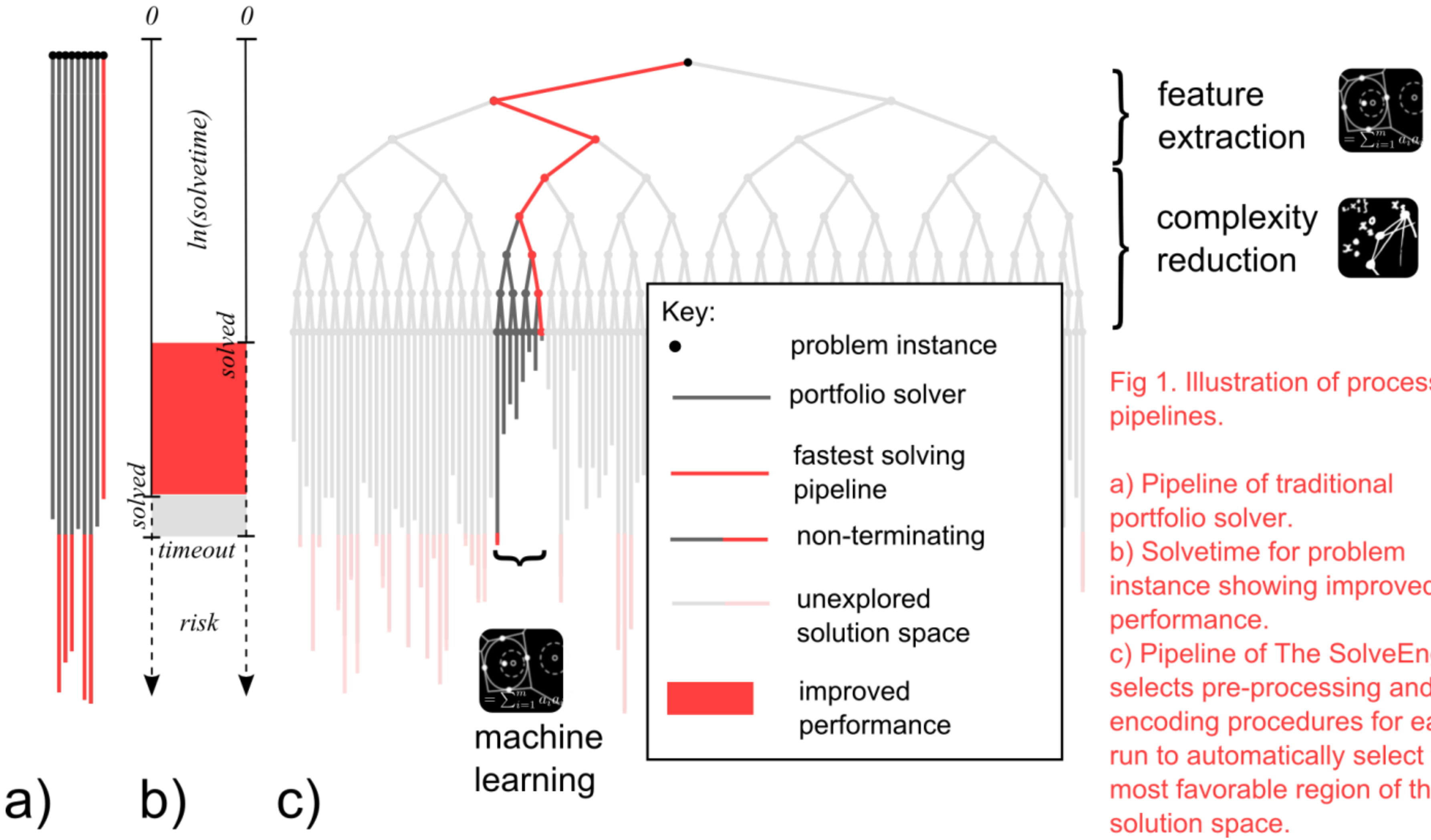
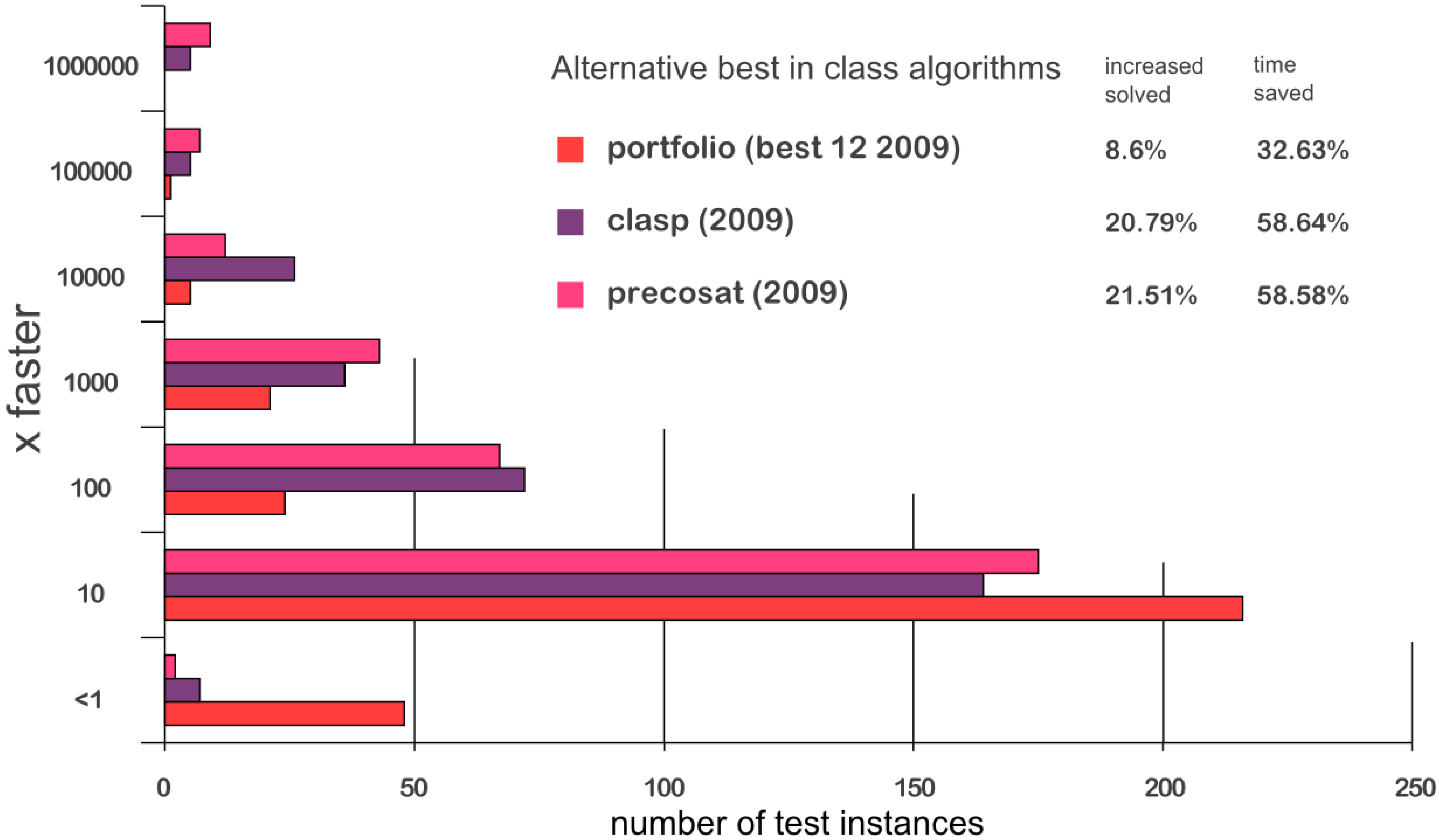


Fig 1. Illustration of processing pipelines.

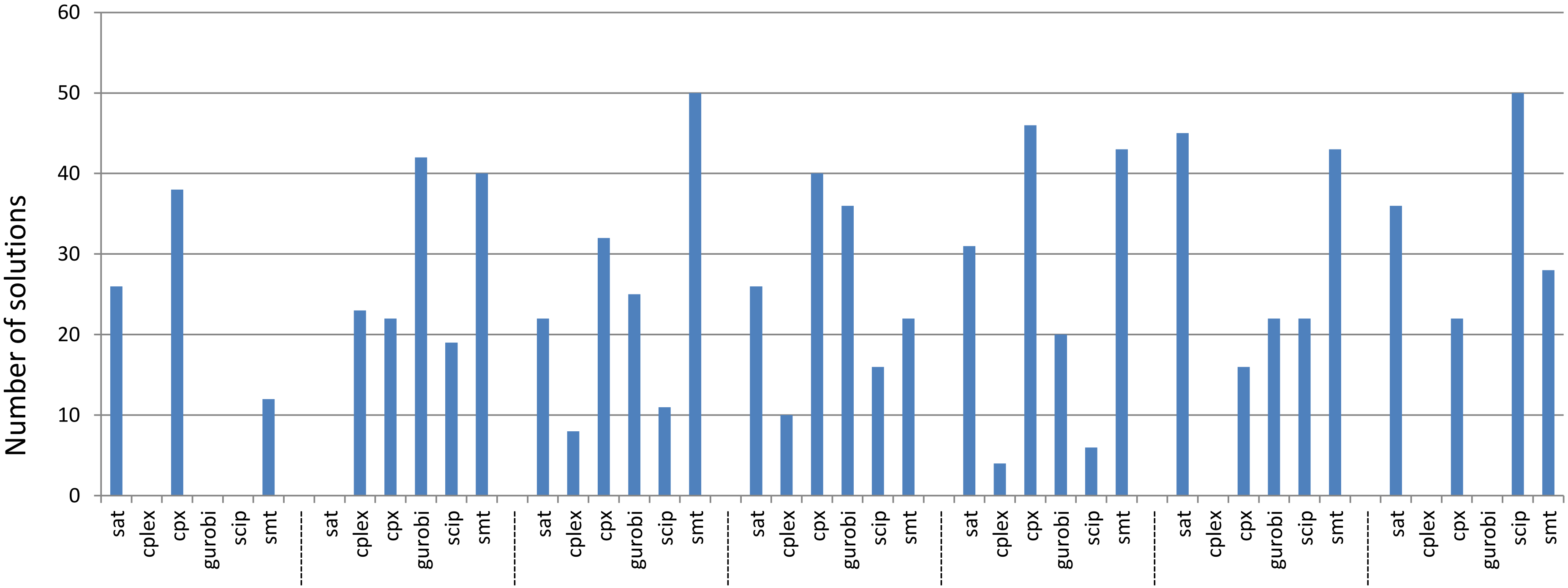
a) Pipeline of traditional portfolio solver.
b) Solvetime for problem instance showing improved performance.
c) Pipeline of The SolveEngine™ selects pre-processing and encoding procedures for each run to automatically select the most favorable region of the solution space.

SAT-Solving Comparison

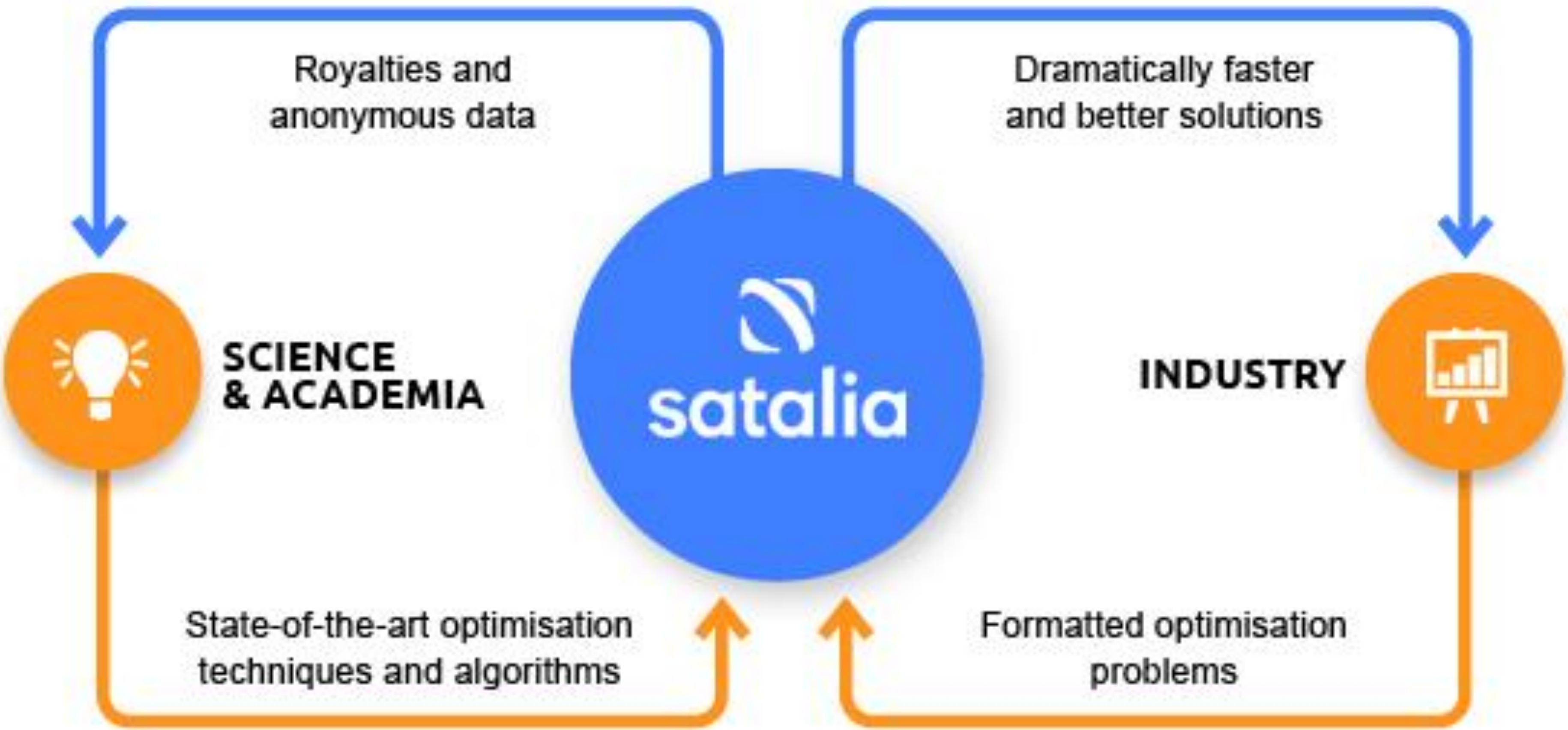


Generic-Solving Comparison

- Algorithm Family vs. Application



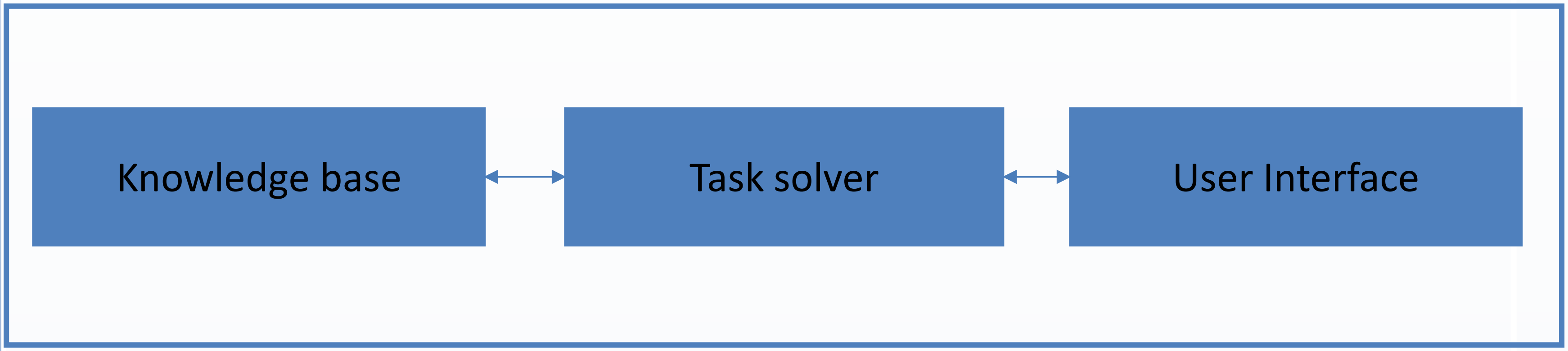
Open-Innovation Model



How to simplify development and maintenance of intelligent software?

Valeriya Gribova
Russian Academy of Sciences

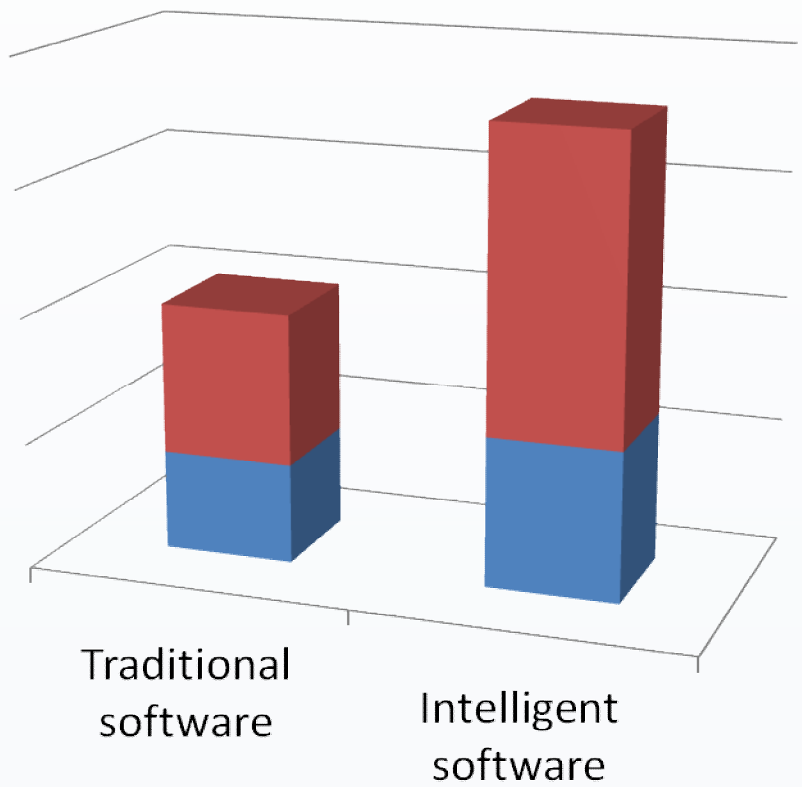
Intelligent system

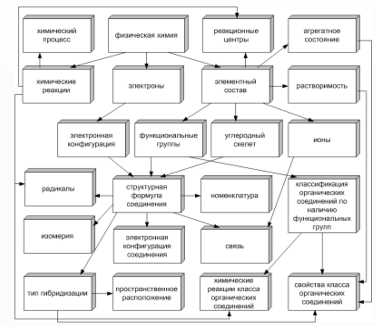


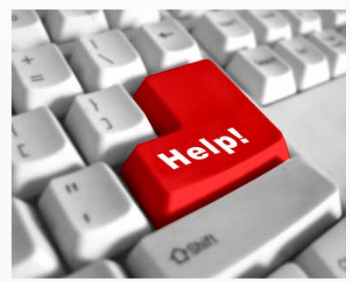
Intelligent system is a system that emulates the decision making ability of a human expert

- High complexity of knowledge formalization
- Complexity of implementation and maintenance
- Rapid obsolescence of knowledge

Software development and maintenance







Complications of program development and maintenance

- **Program development**

Developer

- To understand a set of computation processes (extension of a task) to obtain results for various possible input data
- To specify this set in a programming language (to write a program)

- **Maintenance**

Maintainer

- To recover extension of the task and comprehend why the computation processes result in exactly these output data
- To understand how to change these processes in order to obtain new output data and then modify the program

Imperative paradigm

The basis:	Computational models
The process of obtaining results:	Sequence of states; every follow-up state is generated from the previous one using the assignment operator
The state of a computation process:	A set of variable values
The next state:	A modification of a variable value
The terminal state:	**The computation result**

All the states of the computation process, except for the terminal state, are only **indirectly connected** to the computation result.

Functional paradigm

The basis:	Lambda calculus
The process of obtaining results:	An oriented marked network of a function call
The label of every terminal vertex:	Input data
The label of every non-terminal vertex:	A function value
Arguments of this function:	Labels of arcs outgoing from this vertex
The label of the network root	**The computation result**

All temporary values (labels of non-terminal vertexes), except for the root label, are **indirectly connected** to the computation result.

Logical paradigm

The basis:	First order predicate calculus
The process of obtaining results:	An oriented marked network of result inference
The label of every terminal vertex:	Input data (a relationship tuple)
The label of every non-terminal vertex:	a relationship tuple representing the result of applying a rule to premises
Labels of arcs outgoing from this vertex:	Premises
The label of the network root	**The computation result**

All temporary values (labels of non-terminal vertexes), except for the root label, are **indirectly connected** to the computation result.

The computation result is obtained at the last step of the computation process

To simplify development , understanding, and modification a program

- To suggest a programming paradigm where processes of obtaining result are **direct**
- A fragment of a result is formed **at the every step** of the computation process
- **A program is an executable specification of a set of results of computation** (but not a set of indirect processes of obtaining them)

Specification of a set of results of computation ≡ an **ontology** of computation results

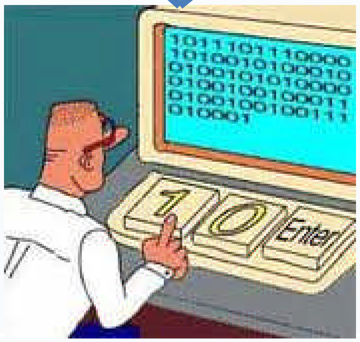
Ontological programming paradigm

The main idea is to suggest a programming paradigm where processes of obtaining result are direct. It means that a fragment of a result is formed at the every step of the computation process.

Ontological programming paradigm

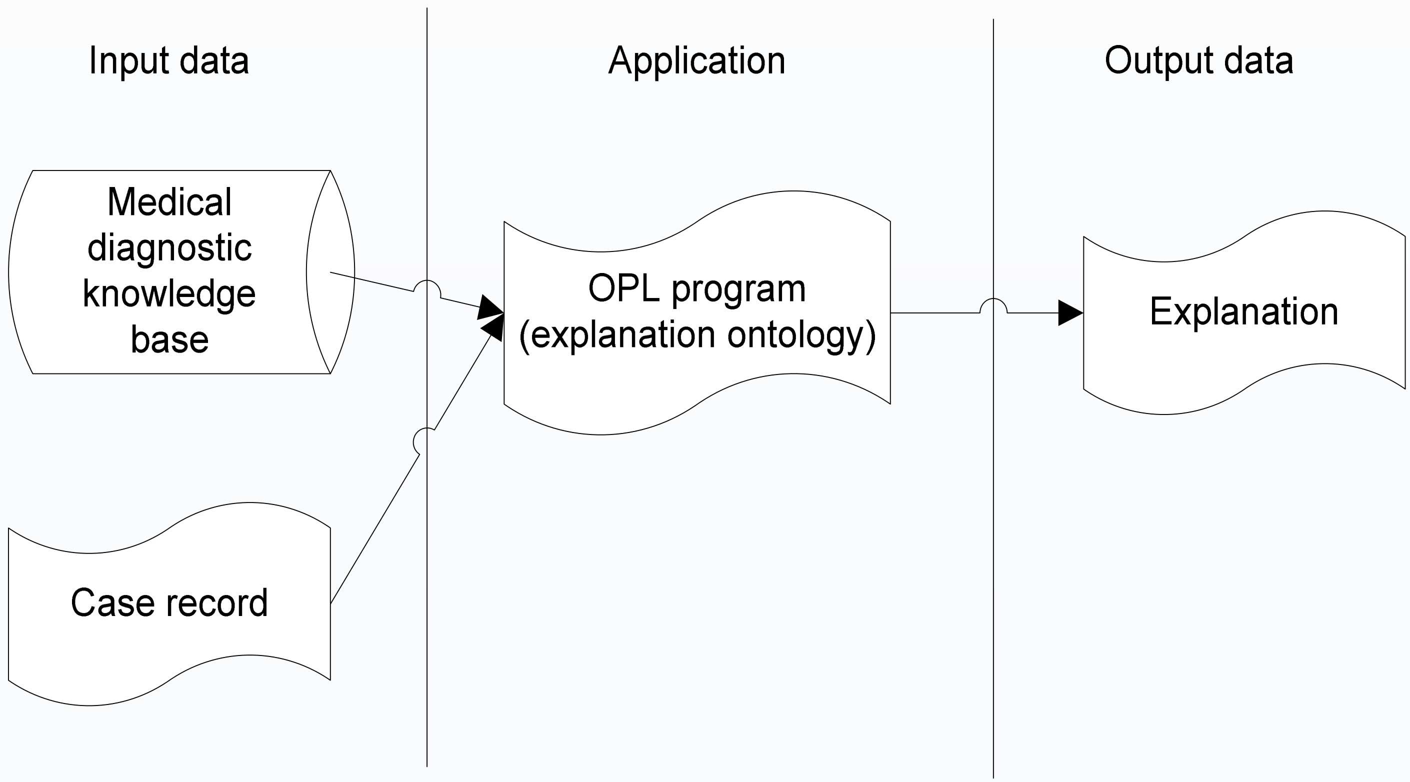
~~Program≡ a process of obtaining results~~

Program≡ an ontology of results

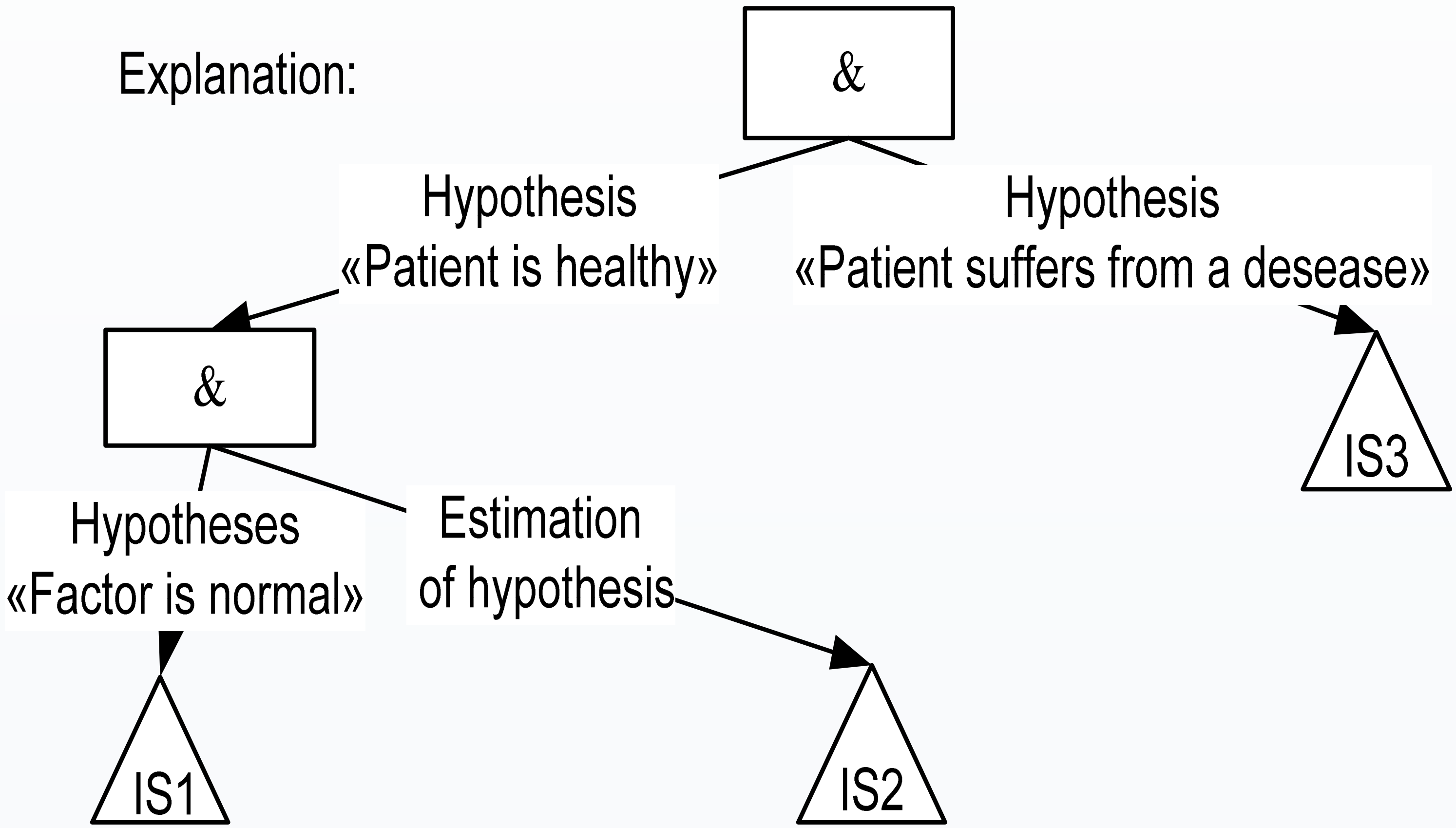


Programmer

Example: expert system of medical diagnosis

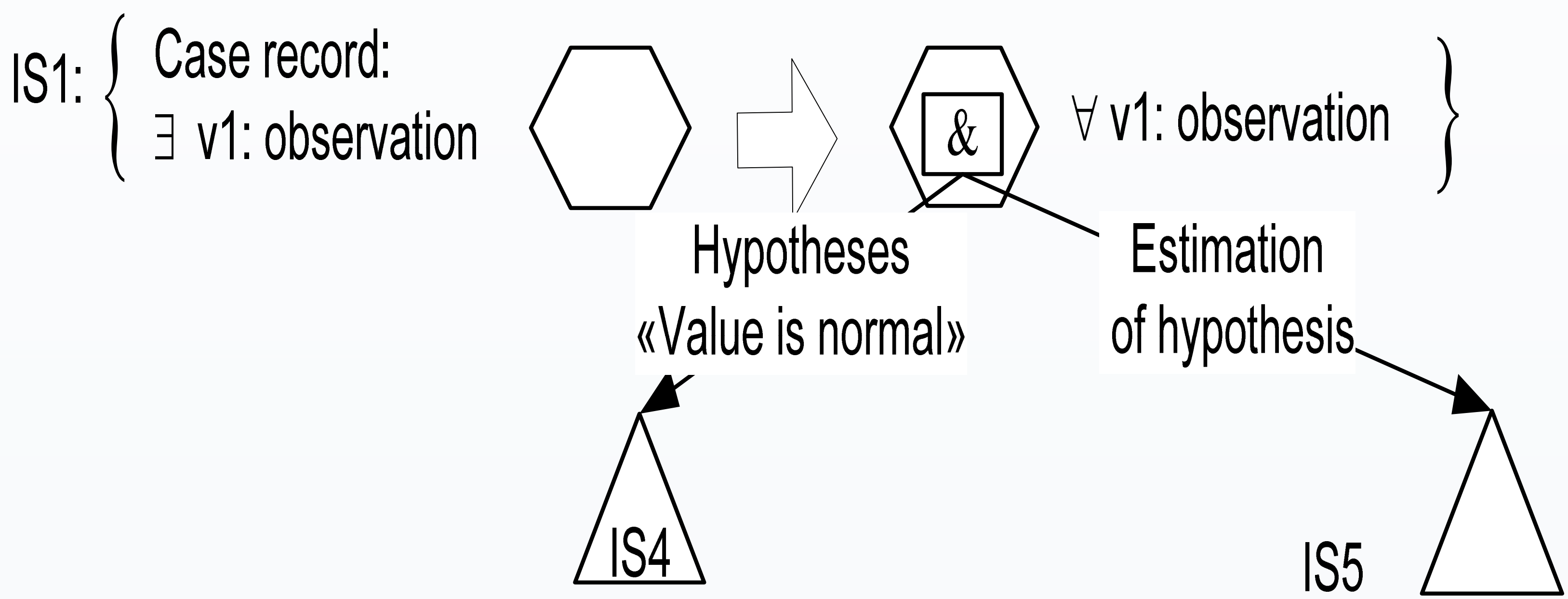


Expert system of medical diagnosis



Expert system of medical diagnosis

Hypotheses “Factor is normal”



Emerging Computing Paradigms & Their Theoretical and Practical Support Tools

Future Trends and Challenges of Compiler Construction and Programming Languages

Torsten Ullrich

Fraunhofer Austria, Visual Computing
and Technische Universität Graz

July 26th, 2012

Torsten Ullrich

torsten.ullrich@fraunhofer.at

http://www.fraunhofer.at

- Senior Researcher at Fraunhofer Austria Research GmbH, Graz, Austria
- Areas of Specialization: Geometry and Modeling Languages
- PhD in Computer Science (Dr. techn.), Technische Universität Graz, Austria
- MSc in Mathematics (Dipl. Math.), Universität Karlsruhe (TH), Germany

CPU and GPU

GPUs are faster than CPUs

- factor 2.5 according to Lee, V. [LKC+10] (Intel Corporation)
- factor 300 according to Fang, Q. [FB09], Keane, A. [Kea10] (NVIDIA Corporation)

Productivity

GPUs are programmed

- using dedicated languages (CUDA, OpenCL) or
- via GPU libraries.

Fraunhofer AUSTRIA

TU Graz

CPU and GPU

GPUs are faster than CPUs

- factor 2.5 according to Lee, V. [LKC$^+$10]
  (Intel Corporation)
- factor 300 according to Fang, Q. [FB09], Keane, A. [Kea10]
  (NVIDIA Corporation)

Productivity

GPUs are programmed

- using dedicated languages (CUDA, OpenCL) or
- via GPU libraries.

GPUs and Productivity

Productivity

Productivity depends on [Pre00]

- programming language,
- available libraries,
- tool chain, and
- societal / educational settings.

Problem

- How to match high-level languages and design with low-level hardware?
- How to translate “Array of Structures” into “Structure of Arrays”?

GPUs and Productivity

Productivity

Productivity depends on [Pre00]

- programming language,
- available libraries,
- tool chain, and
- societal / educational settings.

Problem

- How to match high-level languages and design with low-level hardware?
- How to translate "Array of Structures" into "Structure of Arrays"?

“Array of Structures” vs. “Structure of Arrays”

Array of Structures

```
Vector data1, data2;

Vector calculate() {
    Vector result = ...;
    for(...)
      result.set(...,
        values1.get(...)
          * values2.get(...));
    return result;
}
```

Structure of Arrays

```
class {
    float[] values1;
    float[] values2;

    float[] calculate() {
      float[] result = ...;
      for(...)
        result[...] = values1[...]
              * values2[...];
      return result;
} }
```

Fraunhofer AUSTRIA

TU Graz

Future Challenges / Open Issues

With new hardware platforms (CPU $\rightarrow$ GPU) and new hardware paradigms (single-core/multi-core $\rightarrow$ massively parallel multi-core)

- Do we need new programming languages?
- Do we need new programming paradigms?
- Can existing languages / code be translated to GPU-platforms automatically and take advantage of their computational power (e.g. a Java JIT-compiler with GPU-backend)?

References

Qianqian Fang and David A. Boas.
Monte Carlo simulation of photon migration in 3D turbid media accelerated by graphics processing units.
Optics Express, 17:20178–20190, 2009.

Andy Keane.
"GPUs are only up to 14 times faster then CPUs" says INTEL.
blogs.nvidia.com, 20100623:1, 2010.

Victor W. Lee, Changkyu Kim, Jatin Chhugani, Michael Deisher, Daehyun Kim, Anthony D. Nguyen, Nadathur Satish, Mikhail Smelyanskiy, Srinivas Chennupaty, Per Hammarlund, Ronak Singhal, and Pradeep Dubey.
Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU.

Proceedings of the Annual International Symposium on Computer Architecture, 37:451–460, 2010.

Lutz Prechelt.
An empirical comparison of seven programming languages.
Computer, 33:23–29, 2000.