ERLANGEN REGIONAL
COMPUTING CENTER

# Performance Engineering – Why and How?

Georg Hager
Erlangen Regional Computing Center (RRZE)
Friedrich-Alexander-Universität Erlangen-Nürnberg
Germany

PASC'18
July 2-4, 2018, Basel, Switzerland

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

# Performance Engineering in scientific computing

- A possible definition

> Performance Engineering is a process to study and possibly optimize computer programs in view of a target metric.

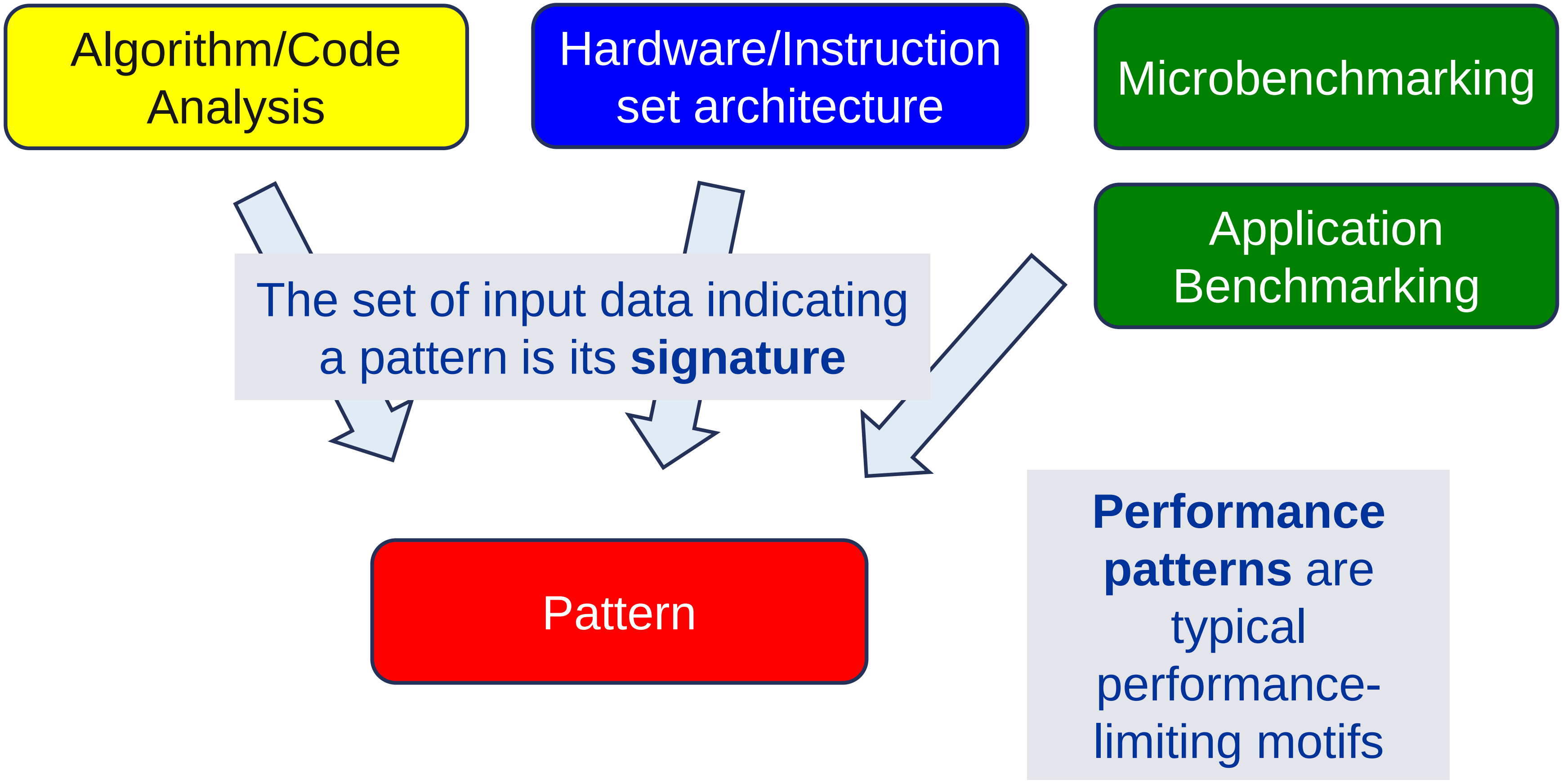
- Target metrics
  - Performance, runtime
  - Scalability
  - Power dissipation, energy consumption
  - Any resource utilization

# Performance Engineering as a Process



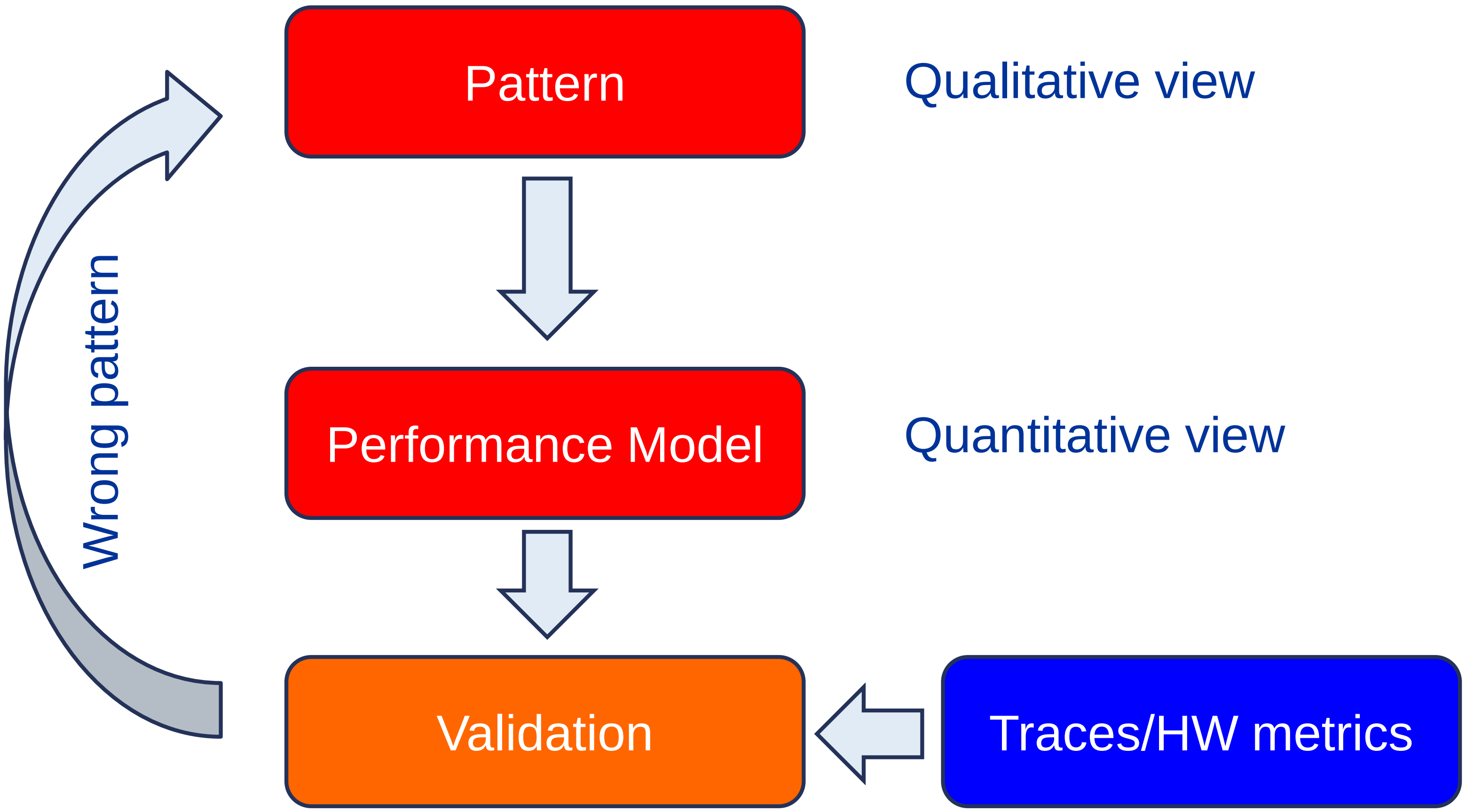
FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

# Performance Engineering Process: Analysis



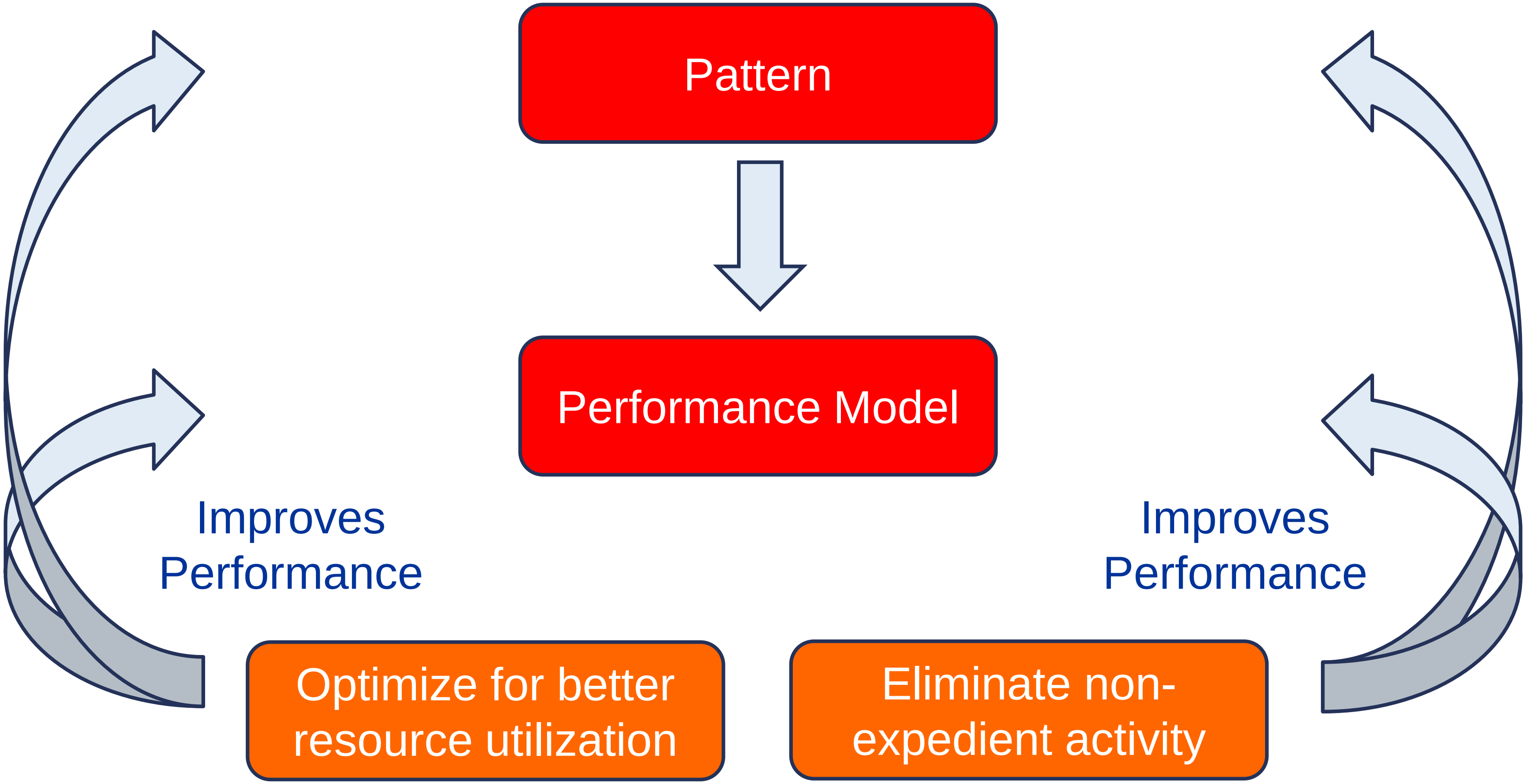

Step 1 **Analysis:** Understanding observed performance

# Performance Engineering Process: Modelling




Step 2 **Formulate Model:** Validate pattern and get quantitative insight.

# Performance Engineering Process: Optimization




Step 3 **Optimization:** Improve utilization of bottleneck resources.

# Performance Patterns



FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

# Performance pattern classification

1. Maximum resource utilization
   (computing at a bottleneck)

2. Hazards
   (something “goes wrong”)

3. Work related
   (too much work or too inefficiently done)

DOI: 10.1007/978-3-642-36949-0_50

# Patterns (I): Bottlenecks & hazards

| Pattern | | Performance behavior | Metric signature, LIKWID performance group(s) |
|---|---|---|---|
| **Bandwidth saturation** (2d-5pt) | | Saturating speedup across cores sharing a data path | Bandwidth meets BW of suitable streaming benchmark (MEM, L3) |
| **ALU saturation** (Kahan summation in L1 cache) | | Throughput at design limit(s) | Good (low) CPI, integral ratio of cycles to specific instruction count(s) (FLOPS_*, DATA, CPI) |
| **Inefficient data access** | **Excess data volume** | (spMVM RHS access) Simple bandwidth performance model much too optimistic | Low BW utilization / Low cache hit ratio, frequent CL evicts or replacements (CACHE, DATA, MEM) |
| | **Latency-bound access** | | |
| **Micro-architectural anomalies** (LD-after-ST aliasing conflict) | | Large discrepancy from simple performance model based on LD/ST and arithmetic throughput | Relevant events are very hardware-specific, e.g., memory aliasing stalls, conflict misses, unaligned LD/ST, requeue events |

# Patterns (II): Hazards

| Pattern | Performance behavior | Metric signature, LIKWID performance group(s) |
|---|---|---|
| **False sharing of cache lines** | Large discrepancy from performance model in parallel case, bad scalability | Frequent (remote) CL evicts (CACHE) |
| **Bad ccNUMA page placement** (No parallel initialization) | Bad or no scaling across NUMA domains, performance improves with interleaved page placement | Unbalanced bandwidth on memory interfaces / High remote traffic (MEM) |
| **Pipelining issues** (Loop-carried dependency) | In-core throughput far from design limit, performance insensitive to data set size | (Large) integral ratio of cycles to specific instruction count(s), bad (high) CPI (FLOPS_*, DATA, CPI) |
| **Control flow issues** (Random branching) | See above | High branch rate and branch miss ratio (BRANCH) |

# Patterns (III): Work-related

| Pattern | | Performance behavior | Metric signature, LIKWID performance group(s) |
|---|---|---|---|
| **Load imbalance / serial fraction**<br>triangular dMVM | | Saturating/sub-linear speedup | Different amount of "work" on the cores (FLOPS_*); note that instruction count is not reliable! |
| **Synchronization overhead**<br>Low-workload OMP loops | | Speedup going down as more cores are added / No speedup with small problem sizes / Cores busy but low FP performance | Large non-FP instruction count (growing with number of cores used) / Low CPI (FLOPS_*, CPI) |
| **Instruction overhead**<br>C++ abstractions gone awry | | [illegible]w application performance, good scaling across cores, performance insensitive to problem size | Low CPI near theoretical limit / Large non-FP instruction count (constant vs. number of cores) (FLOPS_*, DATA, CPI) |
| **Code composition** | **Expensive instructions** | DIV, SQRT in inner loop<br>Similar to instruction overhead | Many cycles per instruction (CPI) if the problem is large-latency arithmetic |
| | **Ineffective instructions** | C/C++ aliasing problem | Scalar instructions dominating in data-parallel loops (FLOPS_*, CPI) |

# Performance Models



FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

# Getting a little more specific

What data/knowledge can a model be based on?

1. Only documented hardware properties + hypotheses
   - Purely analytic model

   } white box

2. Hardware properties + (some) microbenchmark results + hypotheses
   - (Partly) phenomenological model

   } gray box

3. Measured performance/speedup data + hypotheses
   - Curve-fitting analytic model

   } black box

# Models and insights

**Purely predictive analytic model**

- Direct insight into bottlenecks from first principles
- Model failures challenge model assumptions or input data
- Refinements lead to better insights

**Phenomenological analytic model**

- Insight with some “uncharted territory”
- Model failure points to inaccurate or unsuitable measurements

**Curve-fitting model**

- Yields predictions
- Model failure may indicate short-comings of fitting approach
- Refinements by using more fit parameters

white box — gray box — black box

# White- and Grey-Box Models

# Examples for white-/gray-box models

program speedup

serial fraction

$$S(N) = \frac{1}{s + \frac{1-s}{N} + c(N)}$$

Amdahl's Law with communication

latency

msg. length

bandwidth

$$T_{PtP} = T_l + \frac{L}{B}$$

Hockney model for message transmission time

time for computation

time for data transfer

$$T_{\text{exec}} = \max(T_{\text{calc}}, T_{\text{data}})$$

Roofline model for loop code execution time

non-overlapping execution

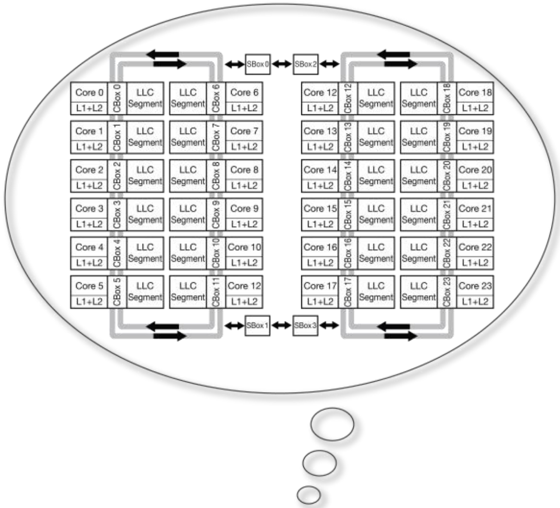
time for data transfer

$$T_{\text{exec}} = f(T_{nOL}, T_{\text{data}}, T_{OL})$$

overlapping execution

ECM model for serial loop code execution time

# Motivation for white-box analytic modeling



- Advantages of white-box models
  - Identification of universality
  - Identification of governing mechanisms
  - Insight via model nature
  - Insight via model failure

- White-box models
  - Determine bottlenecks and influencing factors
  - Design space exploration: What would happen if resource *X* were improved?

# Example: Refining the execution-cache-memory (ECM) performance model

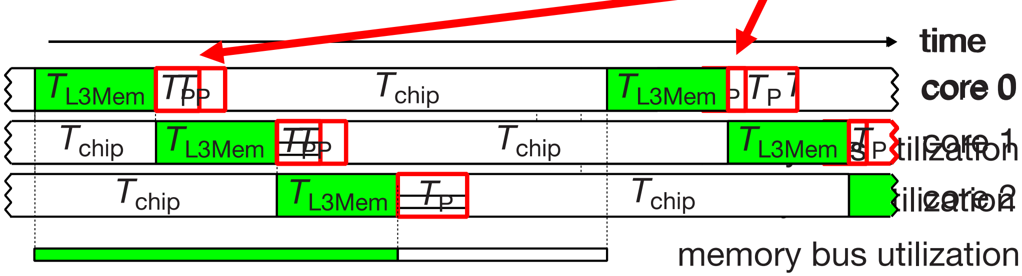
DOI: 10.1007/978-3-319-92040-5_2

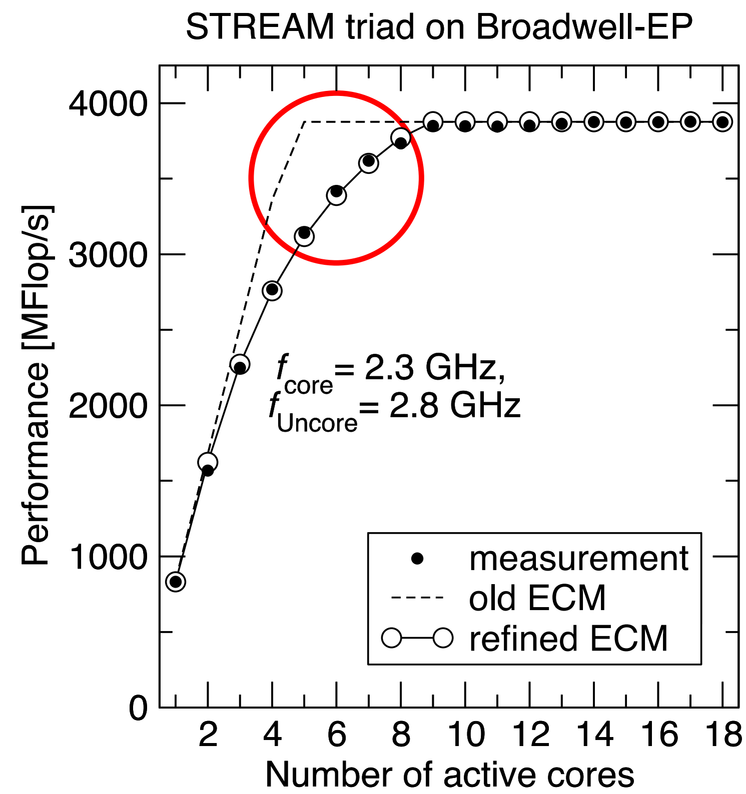
- Original ECM model: $P_{\mathrm{ECM}}(n) = \max\left(n \cdot P_{\mathrm{ECM}}, P_{\mathrm{sat}}\right)$

- Refined model of shared resources

$$u(1) = \frac{T_{\mathrm{L3Mem}}}{T_{\mathrm{L3Mem}} + T_{\mathrm{chip}}}$$

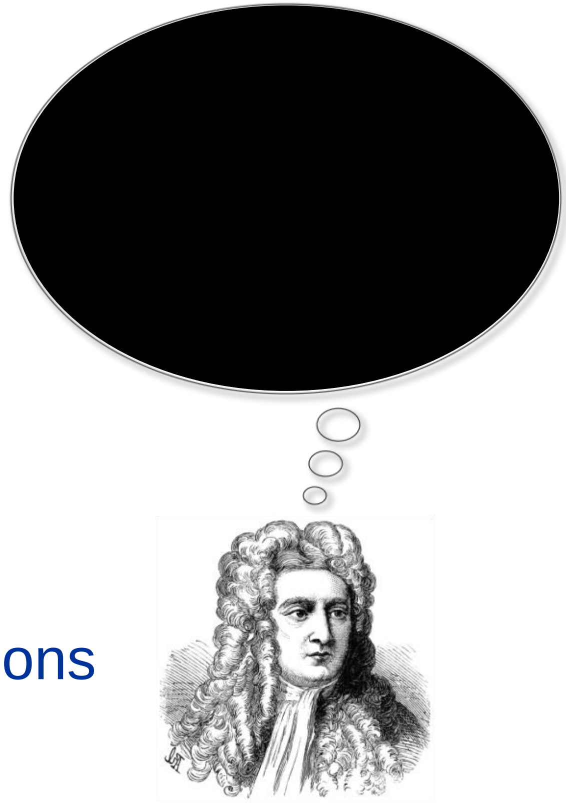
$$u(n) = \frac{nT_{\mathrm{L3Mem}}}{T_{\mathrm{L3Mem}} + T_{\mathrm{chip}} + \underbrace{(n-1)\,u(n-1)\,p_0}_{T_{\mathrm{p}}}}$$



# Black-Box Models

# Motivation for black-box analytic modeling



- White-box models are based on strict assumptions, e.g.:
  - Full overlap of execution & data transfer
  - Steady-state, i.e., ignore wind-up effects
  - Hardware simplifications
- Black-box models have much fewer restrictions
  - Anything that works is allowed
  - Still some assumptions possible

- Black-box performance models
  - Determine influencing factors
  - Deliver target metric predictions for analysis of inaccessible parameter intervals

# Performance model normal form

TECHNISCHE UNIVERSITÄT DARMSTADT

$$f(p) = \sum_{k=1}^{n} c_k \cdot p^{i_k} \cdot \log_2^{j_k}(p)$$

$n \in \mathbb{N}$

$i_k \in I$

$j_k \in J$

$I, J \subset \mathbb{Q}$

$n = 1$

$I = \{0,1,2\}$

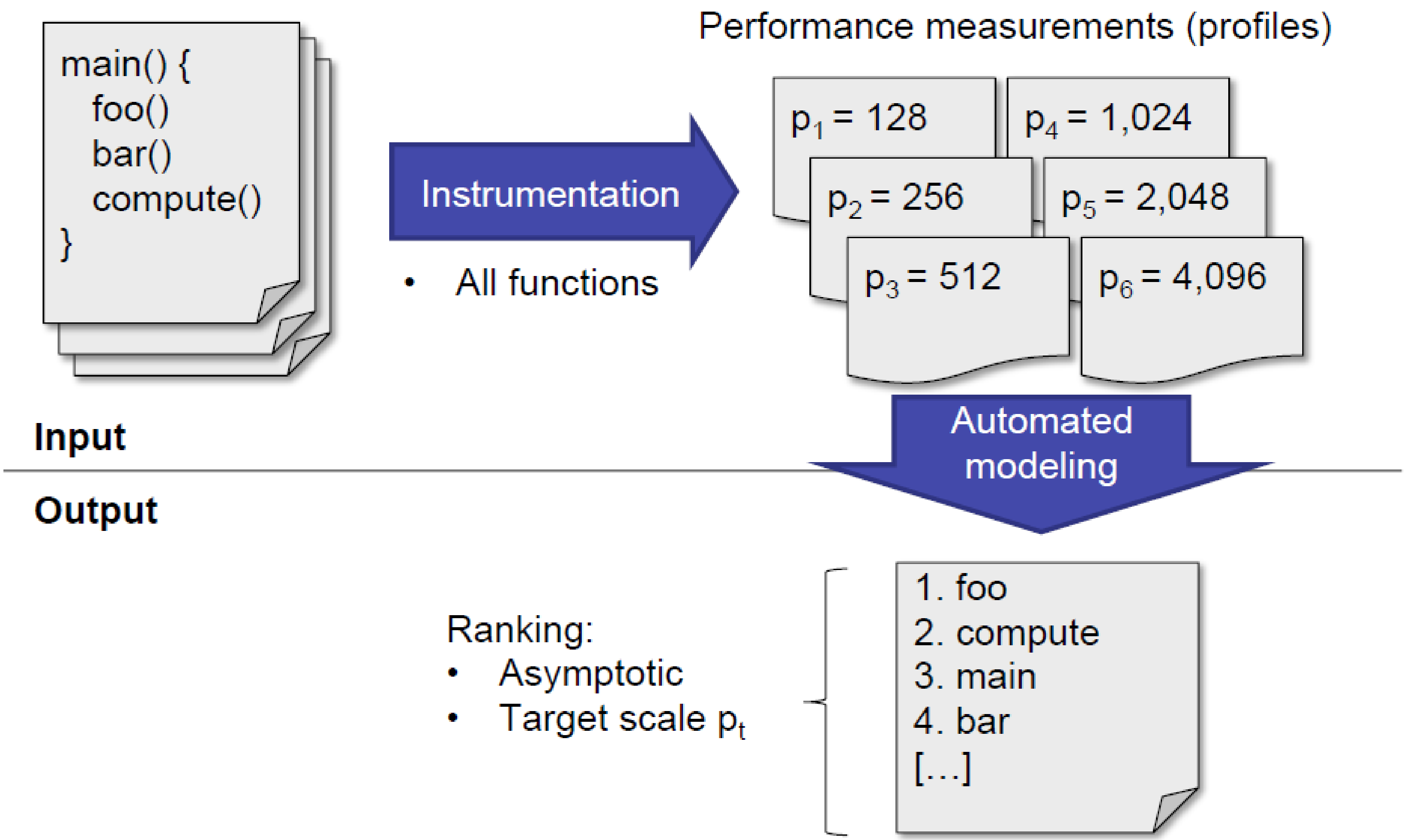
$J = \{0,1\}$

$c_1$ | $c_1 \cdot \log(p)$

$c_1 \cdot p$ | $c_1 \cdot p \cdot \log(p)$

$c_1 \cdot p^2$ | $c_1 \cdot p^2 \cdot \log(p)$

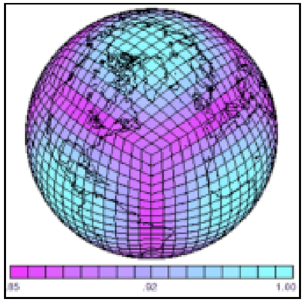
# Automated empirical modeling (2)



# HOMME – Climate

Core of the Community Atmospheric Model (CAM)

- Spectral element dynamical core on a cubed sphere grid



| Kernel [3 of 194] | Model [s] $t = f(p)$ | Predictive error [%] $p_t = 130k$ |
|---|---|---|
| box_rearrange → MPI_Reduce | $3.63 \cdot 10^{-6} p \cdot \sqrt{p} + 7.21 \cdot 10^{-13} p^3$ | 30.34 |
| vlaplace_sphere_vk | $24.44 + 2.26 \cdot 10^{-7} p^2$ | 4.28 |
| compute_and_apply_rhs | 49.09 | 0.83 |

$p_i \leq 43\text{k}$

# ERLANGEN REGIONAL COMPUTING CENTER

SPONSORED BY THE Federal Ministry of Education and Research

SPPEXA

KONWIHR III

Bavarian Network for HPC

**Thank You.**

Julian Hammer
Holger Stengel
Jan Eitzinger
Gerhard Wellein
Johannes Hofmann
Moritz Kreutzer