Insightful Automatic Performance Modeling

TECHNISCHE UNIVERSITÄT DARMSTADT

ETH zürich

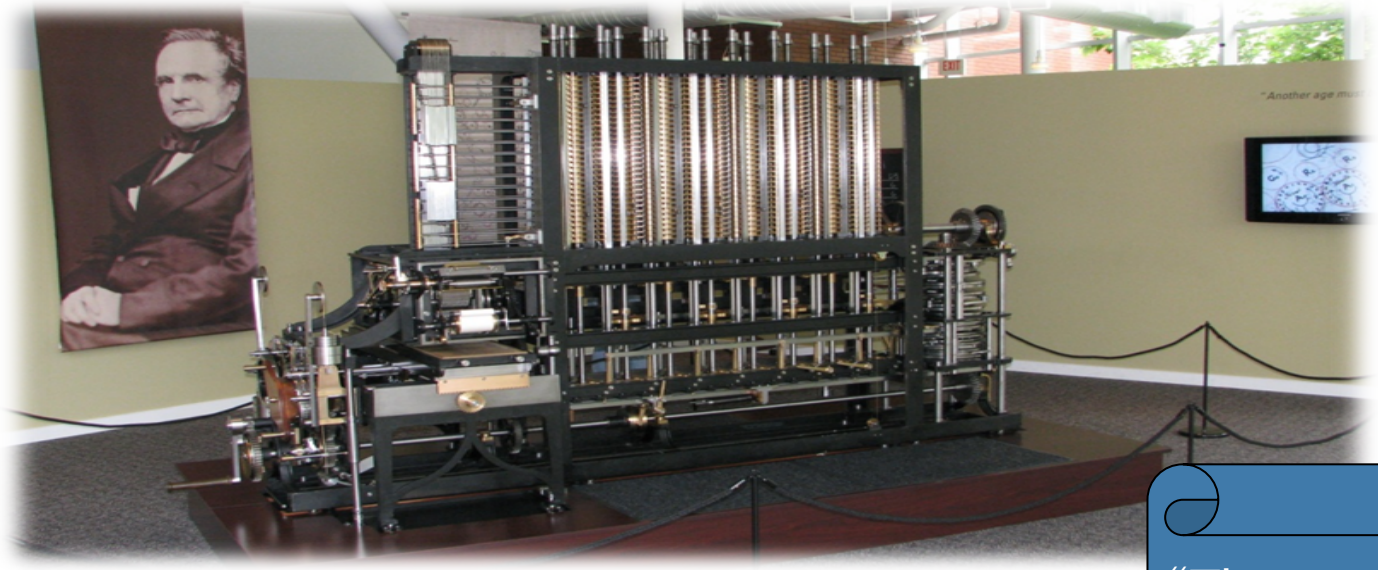
Alexandru Calotoiu[1]

Lawrence Livermore National Laboratory

[1] TU Darmstadt

Performance engineering

Performance: an old problem



"The most constant difficulty in contriving the engine has arisen from the desire to reduce the time in which the calculations were executed to the shortest which is possible."

Charles Babbage
1791 – 1871

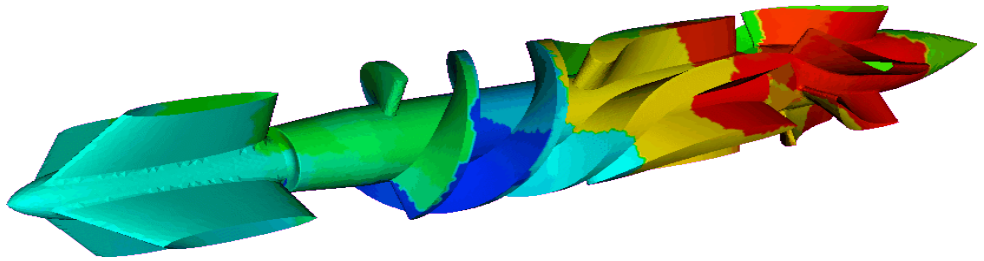
Today: the “free lunch” is over

- Moore's law is still in charge, but
  - Clock rates no longer increase
  - Performance gains only through increased parallelism
- Optimizations of applications more difficult
  - Increasing application complexity
    - Multi-physics
    - Multi-scale
  - Increasing machine complexity
    - Hierarchical networks / memory
    - More CPUs / multi-core

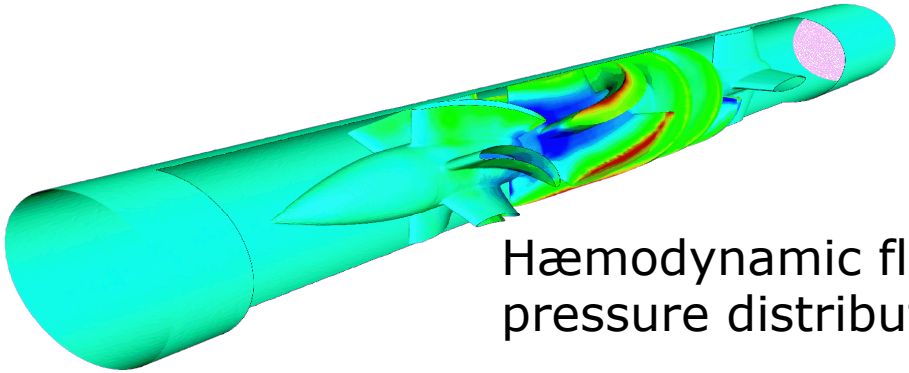
✦ Every doubling of scale reveals a new bottleneck!

Example: XNS

- CFD simulation of unsteady flows
  - Developed by CATS / RWTH Aachen
  - Exploits finite-element techniques, unstructured 3D meshes, iterative solution strategies
- MPI parallel version
  - >40,000 lines of Fortran & C
  - DeBakey blood-pump data set (3,714,611 elements)

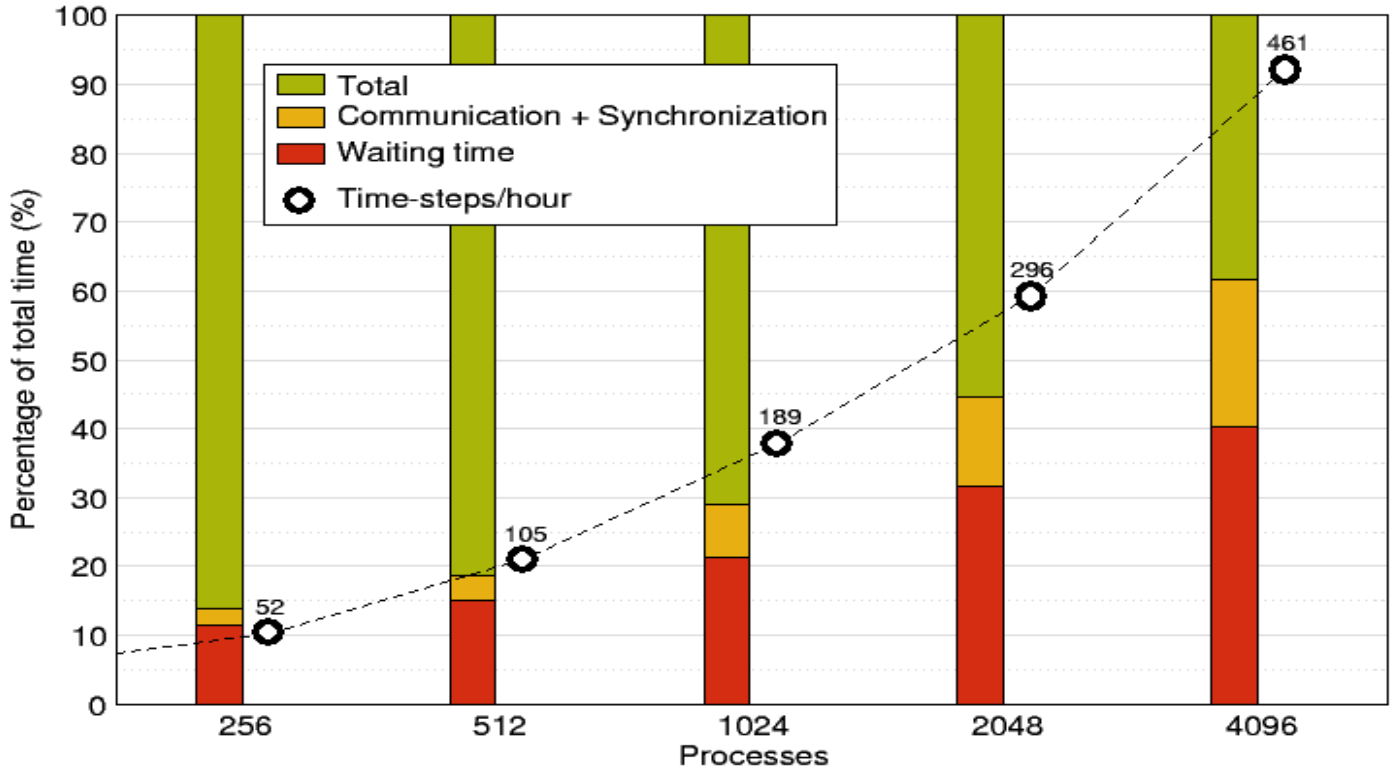


Partitioned finite-element mesh



Hæmodynamic flow pressure distribution

XNS wait-state analysis on BG/L (2007)



Performance factors of parallel applications

- "Sequential" factors
  - Computation
    - ✦ Choose right algorithm, use optimizing compiler
  - Cache and memory
    - ✦ Tough! Only limited tool support, hope compiler gets it right
  - Input / output
    - ✦ Often not given enough attention
- "Parallel" factors
  - Partitioning / decomposition
  - Communication (i.e., message passing)
  - Multithreading
  - Synchronization / locking
    - ✦ More or less understood, good tool support

Tuning basics

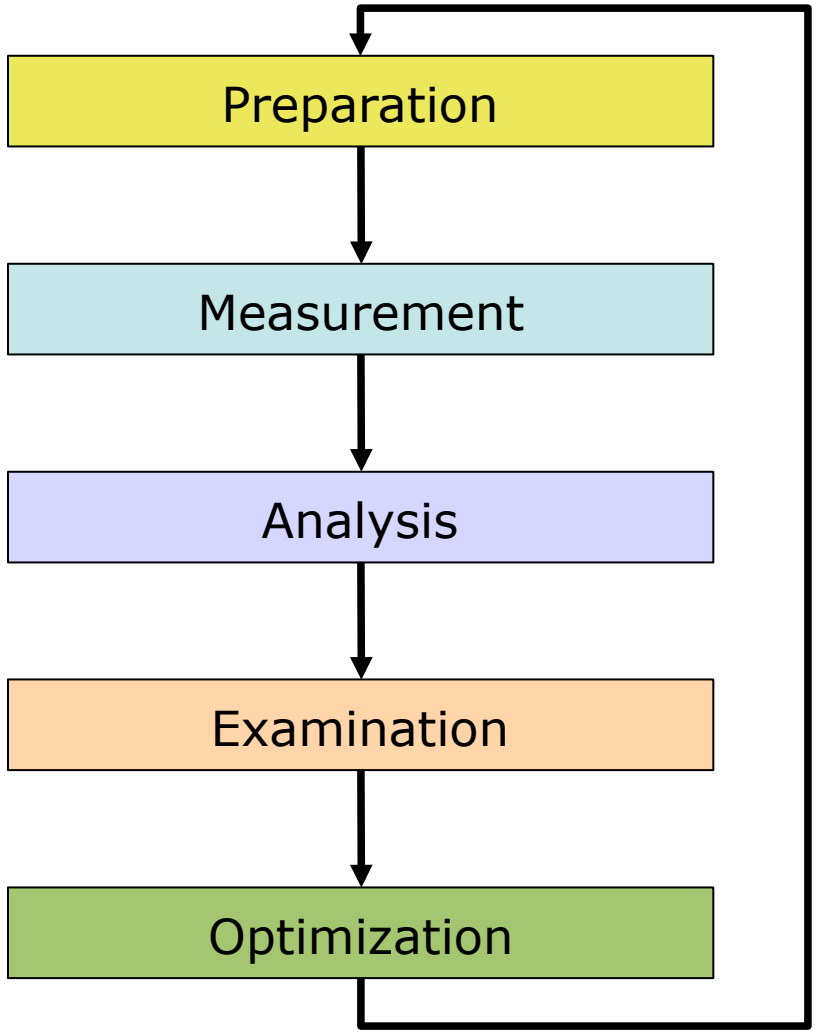
- Successful engineering is a combination of
  - The right algorithms and libraries
  - Compiler flags and directives
  - Thinking !!!
- Measurement is better than guessing
  - To determine performance bottlenecks
  - To compare alternatives
  - To validate tuning decisions and optimizations
    - ✦ After each step!

However…

> "We should forget about small efficiencies, say 97% of the time: premature optimization is the root of all evil."
>
> Charles A. R. Hoare

- It's easier to optimize a slow correct program than to debug a fast incorrect one
  - *Nobody cares how fast you can compute a wrong answer...*

Performance engineering workflow



- Prepare application (with symbols), insert extra code (probes/hooks)
- Collection of data relevant to execution performance analysis
- Calculation of metrics, identification of performance metrics
- Presentation of results in an intuitive/ understandable form
- Modifications intended to eliminate/reduce performance problems

The 80/20 rule

- Programs typically spend 80% of their time in 20% of the code
- Programmers typically spend 20% of their effort to get 80% of the total speedup possible for the application
  - *Know when to stop!*

- Don't optimize what does not matter
  - *Make the common case fast!*

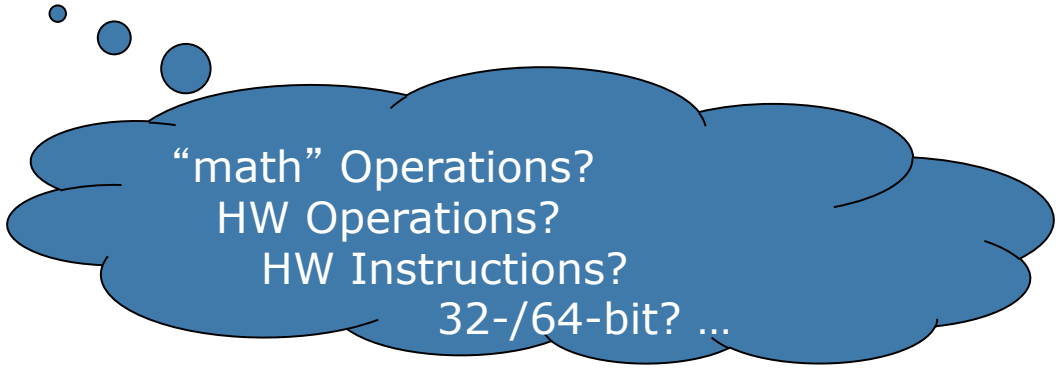
> "If you optimize everything, you will always be unhappy."
>
> Donald E. Knuth

Metrics of performance

- What can be measured?
  - A **count** of how often an event occurs
    - E.g., the number of MPI point-to-point messages sent
  - The **duration** of some interval
    - E.g., the time spent these send calls
  - The **size** of some parameter
    - E.g., the number of bytes transmitted by these calls
- Derived metrics
  - E.g., rates / throughput
  - Needed for normalization

Example metrics

- Execution time
- Number of function calls
- CPI
  - CPU cycles per instruction
- FLOPS
  - Floating-point operations executed per second

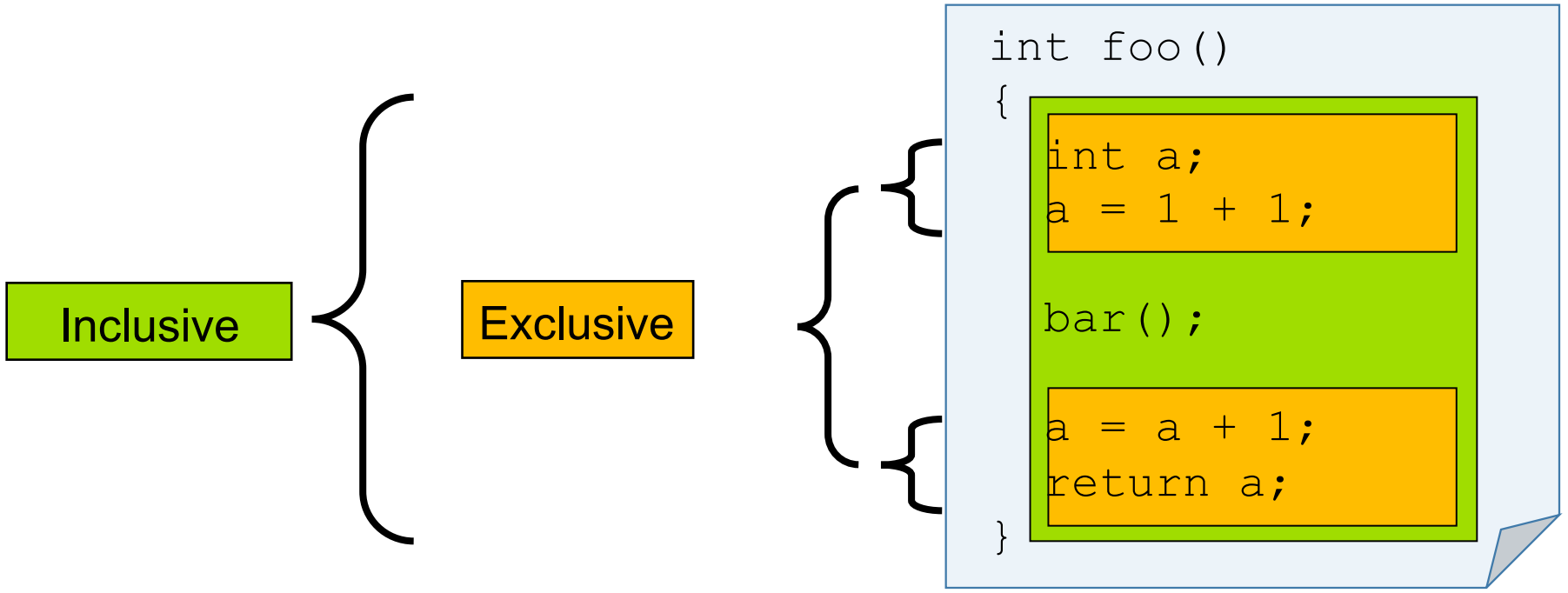


Execution time

- Wall-clock time
  - Includes waiting time: I/O, memory, other system activities
  - In time-sharing environments also the time consumed by other applications
- CPU time
  - Time spent by the CPU to execute the application
  - Does not include time the program was context-switched out
    - Problem: Does not include inherent waiting time (e.g., I/O)
    - Problem: Portability? What is user, what is system time?
- Problem: Execution time is non-deterministic
  - Use mean or minimum of several runs

Inclusive vs. Exclusive values

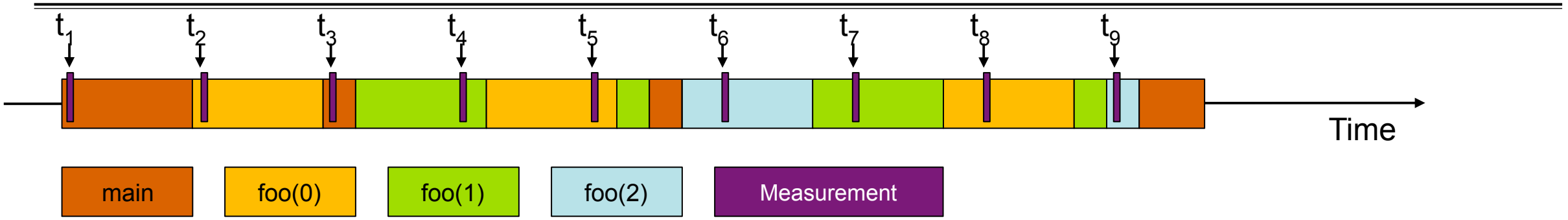
- Inclusive
  - Information of all sub-elements aggregated into single value
- Exclusive
  - Information cannot be subdivided further



Classification of measurement techniques

- How are performance measurements triggered?
  - Sampling
  - Code instrumentation
- How is performance data recorded?
  - Profiling / Runtime summarization
  - Tracing
- How is performance data analyzed?
  - Online
  - Post mortem

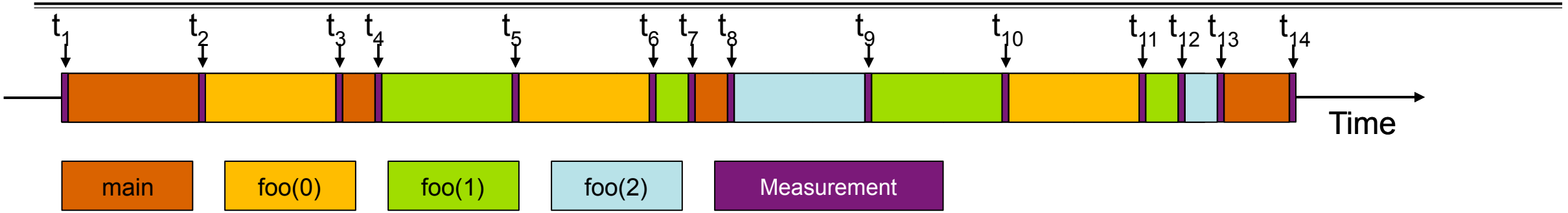
Sampling




```
int main()
{
  int i;
  for (i=0; i < 3; i++)
    foo(i);
  return 0;
}
void foo(int i)
{
  if (i > 0)
    foo(i - 1);
}
```

- Running program is periodically interrupted to take measurement
  - Timer interrupt, OS signal, or HWC overflow
  - Service routine examines return-address stack
  - Addresses are mapped to routines using symbol table information
- **Statistical** inference of program behavior
  - Not very detailed information on highly volatile metrics
  - Requires long-running applications
- Works with unmodified executables

Instrumentation




```
int main()
{  int i;
   Enter("main");
  for (i=0; i < 3; i++)
    foo(i);
   Leave("main");
  return 0;
}
void foo(int i)
{
   Enter("foo");
  if (i > 0)
    foo(i - 1);
   Leave("foo");
}
```

- Measurement code is inserted such that every event of interest is captured **directly**
  - Can be done in various ways
- Advantage:
  - Much more detailed information
- Disadvantage:
  - Processing of source-code / executable necessary
  - Large relative overheads for small functions

Instrumentation techniques

- Static instrumentation
  - Program is instrumented prior to execution
- Dynamic instrumentation
  - Program is instrumented at runtime
- Code is inserted
  - Manually
  - Automatically
    - By a preprocessor / source-to-source translation tool
    - By a compiler
    - By linking against a pre-instrumented library / runtime system
    - By binary-rewrite / dynamic instrumentation tool

Critical issues

- Accuracy
  - Intrusion overhead
    - Measurement itself needs time and thus lowers performance
  - Perturbation
    - Measurement alters program behaviour
    - E.g., memory access pattern
  - Accuracy of timers & counters
- Granularity
  - How many measurements?
  - How much information / processing during each measurement?

✦ *Tradeoff: Accuracy vs. Expressiveness of data*

Classification of measurement techniques

- How are performance measurements triggered?
  - Sampling
  - Code instrumentation
- How is performance data recorded?
  - Profiling / Runtime summarization
  - Tracing
- How is performance data analyzed?
  - Online
  - Post mortem

Profiling / Runtime summarization

- Recording of aggregated information
  - Total, maximum, minimum, …
- For measurements
  - Time
  - Counts
    - Function calls
    - Bytes transferred
    - Hardware counters
- Over program and system entities
  - Functions, call sites, basic blocks, loops, …
  - Processes, threads

✦ *Profile = summarization of events over execution interval*

Types of profiles

- Flat profile
  - Shows distribution of metrics per routine / instrumented region
  - Calling context is not taken into account
- Call-path profile
  - Shows distribution of metrics per executed call path
  - Sometimes only distinguished by partial calling context (e.g., two levels)
- Special-purpose profiles
  - Focus on specific aspects, e.g., MPI calls or OpenMP constructs
  - Comparing processes/threads

Tracing

- Recording information about significant points (events) during execution of the program
  - Enter / leave of a region (function, loop, …)
  - Send / receive a message, …
- Save information in event record
  - Timestamp, location, event type
  - Plus event-specific information (e.g., communicator, sender / receiver, …)
- Abstract execution model on level of defined events

✦ *Event trace = Chronologically ordered sequence of event records*

Tracing vs. Profiling

- Tracing advantages
  - Event traces preserve the **temporal** and **spatial** relationships among individual events (✦ context)
  - Allows reconstruction of **dynamic** application behaviour on any required level of abstraction
  - Most general measurement technique
    - Profile data can be reconstructed from event traces
- Disadvantages
  - Traces can very quickly become extremely large
  - Writing events to file at runtime causes perturbation
  - Writing tracing software is complicated
    - Event buffering, clock synchronization, ...

Classification of measurement techniques

- How are performance measurements triggered?
  - Sampling
  - Code instrumentation
- How is performance data recorded?
  - Profiling / Runtime summarization
  - Tracing
- How is performance data analyzed?
  - Online
  - Post mortem

Online analysis

- Performance data is processed during measurement run
  - Process-local profile aggregation
  - More sophisticated inter-process analysis using
    - “Piggyback” messages
    - Hierarchical network of analysis agents
- Inter-process analysis often involves application steering to interrupt and re-configure the measurement

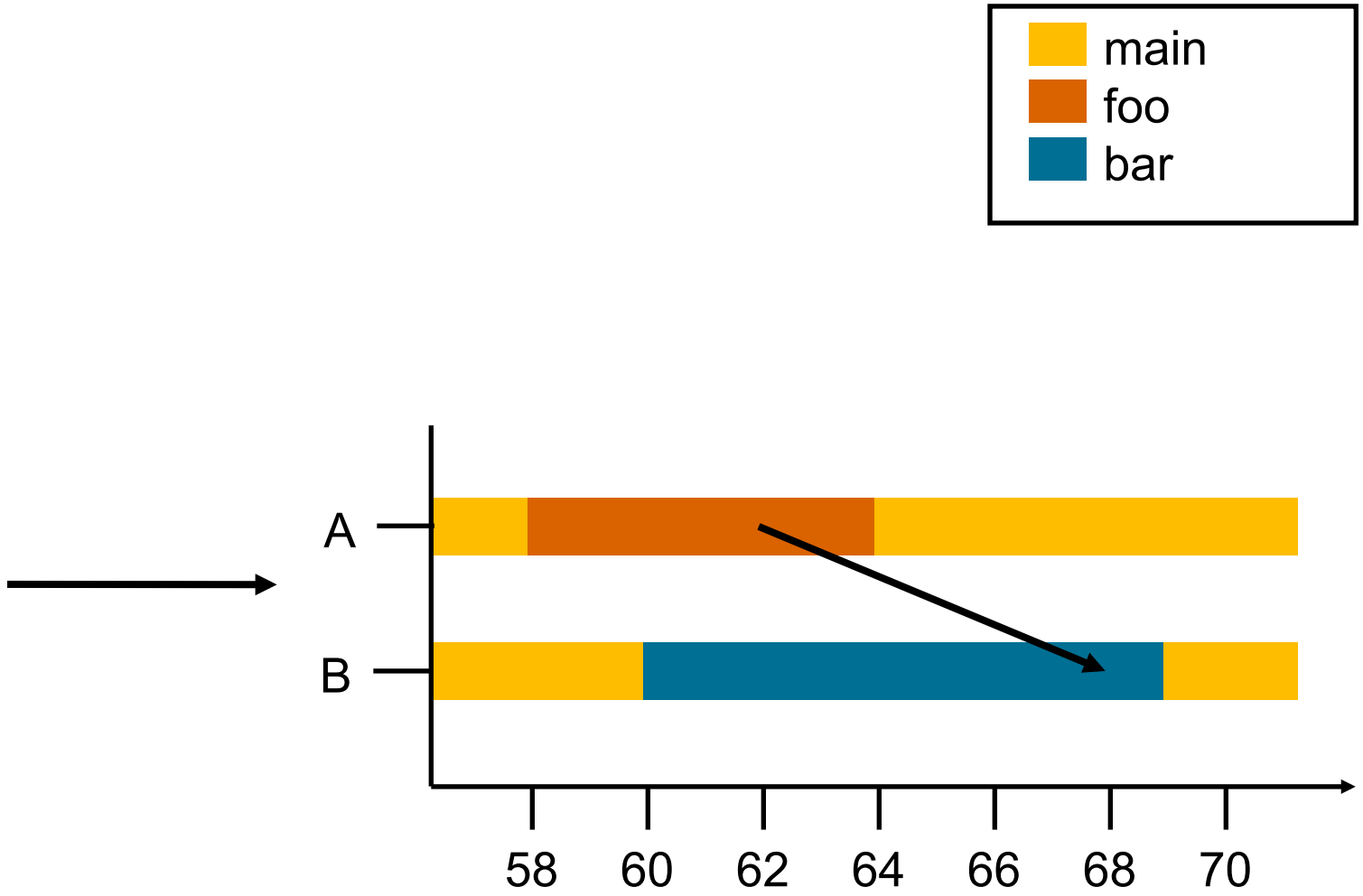
Post-mortem analysis

- Performance data is stored at end of measurement run
- Data analysis is performed afterwards
  - Automatic search for bottlenecks
  - Visual trace analysis
  - Calculation of statistics

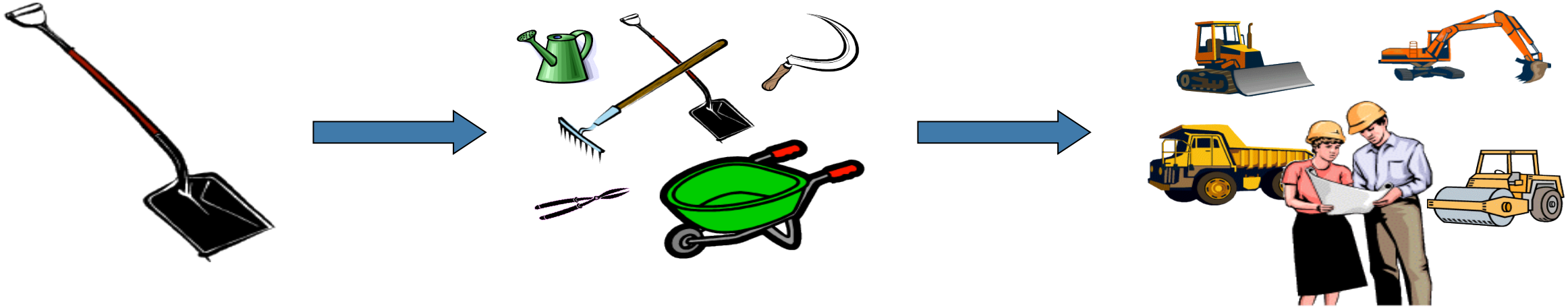
Example: Time-line visualization

1	foo
2	bar
3	...

...			
58	A	ENTER	1
60	B	ENTER	2
62	A	SEND	B
64	A	EXIT	1
68	B	RECV	A
69	B	EXIT	2
...			



No single solution is sufficient!



- *A combination of different methods, tools and techniques is typically needed!*
  - Analysis
    - Statistics, visualization, automatic analysis, data mining, ...
  - Measurement
    - Sampling / instrumentation, profiling / tracing, ...
  - Instrumentation
    - Source code / binary, manual / automatic, ...

Typical performance analysis procedure

- Do I have a performance problem at all?
  - Time / speedup / scalability measurements
- What is the key bottleneck (computation / communication)?
  - MPI / OpenMP / flat profiling
- Where is the key bottleneck?
  - Call-path profiling, detailed basic block profiling
- Why is it there?
  - Hardware counter analysis, trace selected parts to keep trace size manageable
- Does the code have scalability problems?
  - Load imbalance analysis, compare profiles at various sizes function-by-function

Score-P

Virtual Institute – High Productivity Supercomputing

Association of HPC programming tool builders

Mission

- Development of portable programming tools that assist programmers in diagnosing programming errors and optimizing the performance of their applications
- Integration of these tools
- Organization of training events designed to teach the application of these tools
- Organization of academic workshops to facilitate the exchange of ideas on tool development and to promote young scientists

www.vi-hps.org

Fragmentation of Tools Landscape

- Several performance tools co-exist
- Separate measurement systems and output formats
- Complementary features and overlapping functionality
- Redundant effort for development and maintenance
- Limited or expensive interoperability
- Complications for user experience, support, training

Vampir	Scalasca	TAU	Periscope
VampirTrace OTF	EPILOG / CUBE	TAU native formats	Online measurement

SILC Project Idea

- Start a community effort for a common infrastructure
  - Score-P instrumentation and measurement system
  - Common data formats OTF2 and CUBE4
- Developer perspective:
  - Save manpower by sharing development resources
  - Invest in new analysis functionality and scalability
  - Save efforts for maintenance, testing, porting, support, training
- User perspective:
  - Single learning curve
  - Single installation, fewer version updates
  - Interoperability and data exchange
- SILC project funded by BMBF
- Close collaboration PRIMA project funded by DOE

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung
DEPARTMENT OF ENERGY • UNITED STATES OF AMERICA •

Partners

- Forschungszentrum Jülich, Germany
- TU Darmstadt, Darmstadt, Germany
- Gesellschaft für numerische Simulation mbH Braunschweig, Germany
- RWTH Aachen, Germany
- Technische Universität Dresden, Germany
- Technische Universität München, Germany
- University of Oregon, Eugene, USA

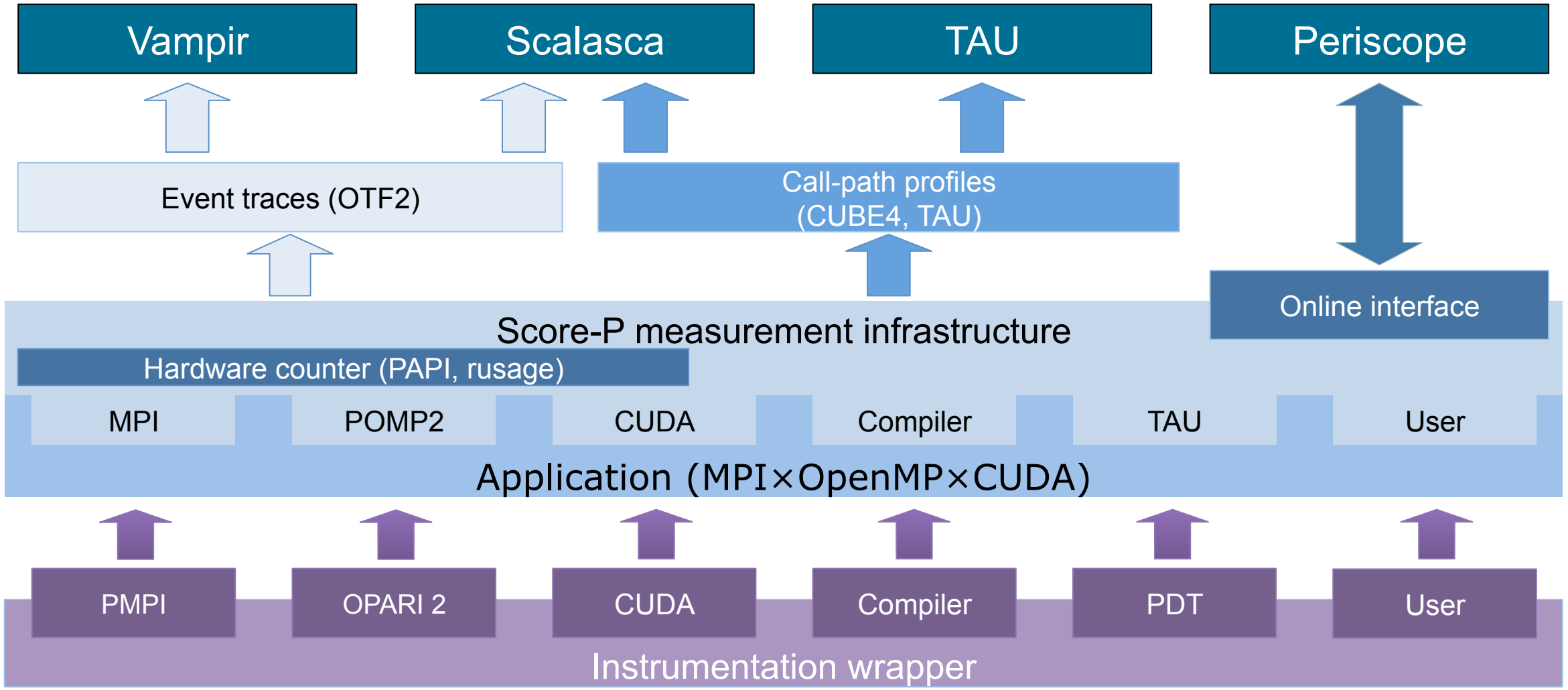
Score-P Functionality

- Provide typical functionality for HPC performance tools
- Support all fundamental concepts of partner's tools

- Instrumentation (various methods)
- Flexible measurement without re-compilation:
  - Basic and advanced profile generation
  - Event trace recording
  - Online access to profiling data

- MPI, OpenMP, and hybrid parallelism (and serial)
- Enhanced functionality (OpenMP 3.0, CUDA, highly scalable I/O)

Score-P Architecture



Future Features and Management

- Scalability to maximum available CPU core count
- Support for OpenCL, HMPP, PTHREAD
- Support for sampling, binary instrumentation
- Support for new programming models, e.g., PGAS
- Support for new architectures

- Ensure a single official release version at all times
  which will always work with the tools
- Allow experimental versions for new features or research

- Commitment to joint long-term cooperation

Score-P User Instrumentation API

- Can be used to mark initialization, solver & other phases
  - Annotation macros ignored by default
  - Enabled with [**--user**] flag
- Appear as additional regions in analyses
  - Distinguishes performance of important phase from rest
- Can be of various type
  - E.g., function, loop, phase
  - See user manual for details
- Available for Fortran / C / C++

Score-P User Instrumentation API (Fortran)

```
#include "scorep/SCOREP_User.inc"

subroutine foo(…)
  ! Declarations
  SCOREP_USER_REGION_DEFINE( solve )

  ! Some code…
  SCOREP_USER_REGION_BEGIN( solve, "<solver>", \
                            SCOREP_USER_REGION_TYPE_LOOP )
  do i=1,100
    [...]
  end do
  SCOREP_USER_REGION_END( solve )
  ! Some more code…
end subroutine
```

- Requires processing by the C preprocessor

Score-P User Instrumentation API (C/C++)

```
#include "scorep/SCOREP_User.h"

void foo()
{
  /* Declarations */
  SCOREP_USER_REGION_DEFINE( solve )

  /* Some code… */
  SCOREP_USER_REGION_BEGIN( solve, "<solver>", \
                            SCOREP_USER_REGION_TYPE_LOOP )
  for (i = 0; i < 100; i++)
  {
    [...]
  }
  SCOREP_USER_REGION_END( solve )
  /* Some more code… */
}
```

Score-P User Instrumentation API (C++)

```
#include "scorep/SCOREP_User.h"

void foo()
{
  // Declarations

  // Some code…
  {
    SCOREP_USER_REGION( "<solver>", SCOREP_USER_REGION_TYPE_LOOP )
    for (i = 0; i < 100; i++)
    {
      [...]
    }
  }
  // Some more code…
}
```

Score-P Measurement Control API

- Can be used to temporarily disable measurement for certain intervals
  - Annotation macros ignored by default
  - Enabled with [**--user**] flag

```
#include "scorep/SCOREP_User.inc"

subroutine foo(…)
  ! Some code…
  SCOREP_RECORDING_OFF()
  ! Loop will not be measured
  do i=1,100
    [...]
  end do
  SCOREP_RECORDING_ON()
  ! Some more code…
end subroutine
```

Fortran (requires C preprocessor)

```
#include "scorep/SCOREP_User.h"

void foo(…) {
  /* Some code… */
  SCOREP_RECORDING_OFF()
  /* Loop will not be measured */
  for (i = 0; i < 100; i++) {
    [...]
  }
  SCOREP_RECORDING_ON()
  /* Some more code… */
}
```

C / C++

Further Information

Score-P

- Community instrumentation & measurement infrastructure
  - Instrumentation (various methods)
  - Basic and advanced profile generation
  - Event trace recording
  - Online access to profiling data
- Available under New BSD open-source license
- Documentation & Sources:
  - http://www.score-p.org
- User guide also part of installation:
  - <prefix>/share/doc/scorep/{pdf,html}/
- Contact: info@score-p.org
- Bugs: scorep-bugs@groups.tu-dresden.de

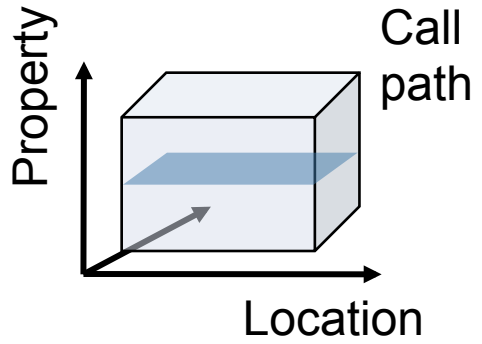
CUBE

CUBE

- Parallel program analysis report exploration tools
  - Libraries for XML report reading & writing
  - Algebra utilities for report processing
  - GUI for interactive analysis exploration
    - requires Qt4
- Originally developed as part of Scalasca toolset
- Now available as a separate component
  - Can be installed independently of Score-P, e.g., on laptop or desktop

Analysis presentation and exploration

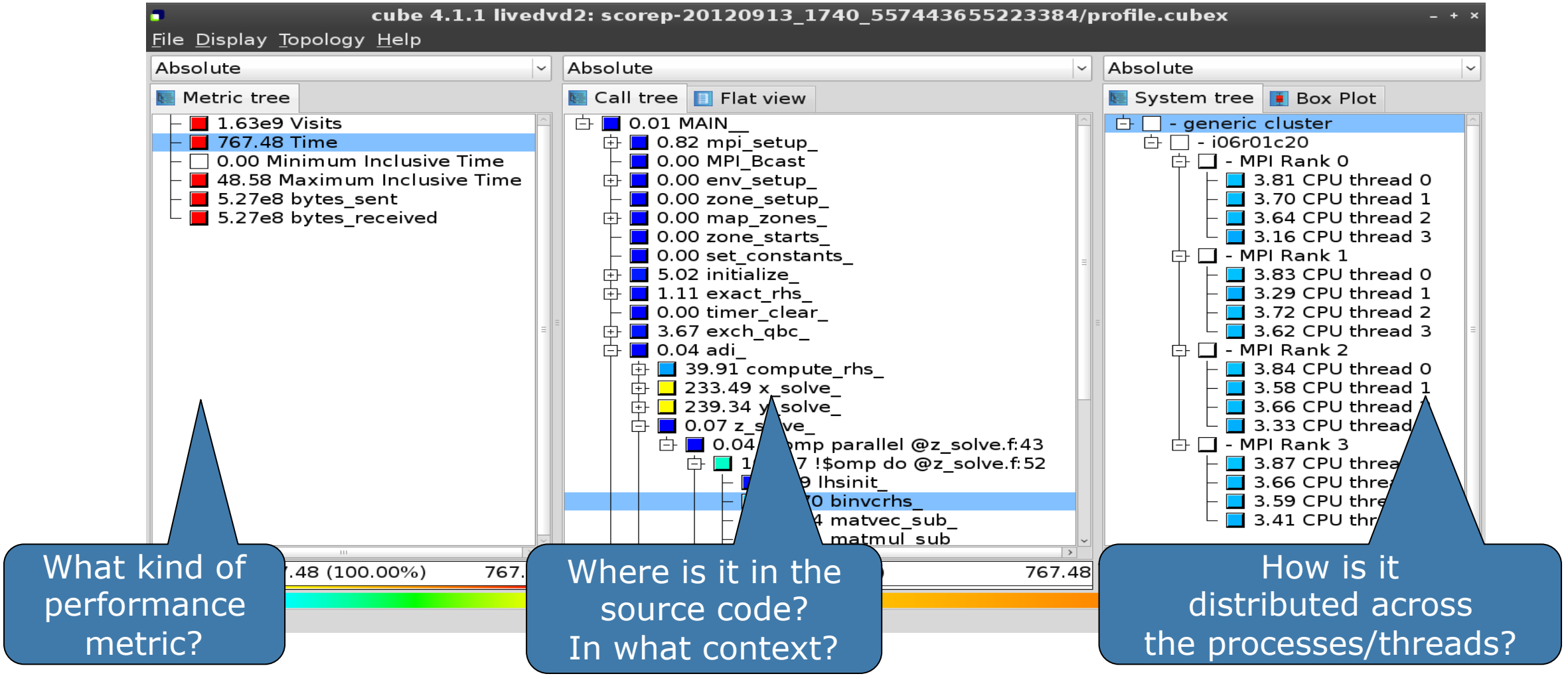
- Representation of values (severity matrix) on three hierarchical axes
  - Performance property (metric)
  - Call-tree path (program location)
  - System location (process/thread)



- Three coupled tree browsers

- CUBE displays severities
  - As value: for precise comparison
  - As colour: for easy identification of hotspots
  - Inclusive value when closed & exclusive value when expanded
  - Customizable via display mode

Analysis presentation



Further information

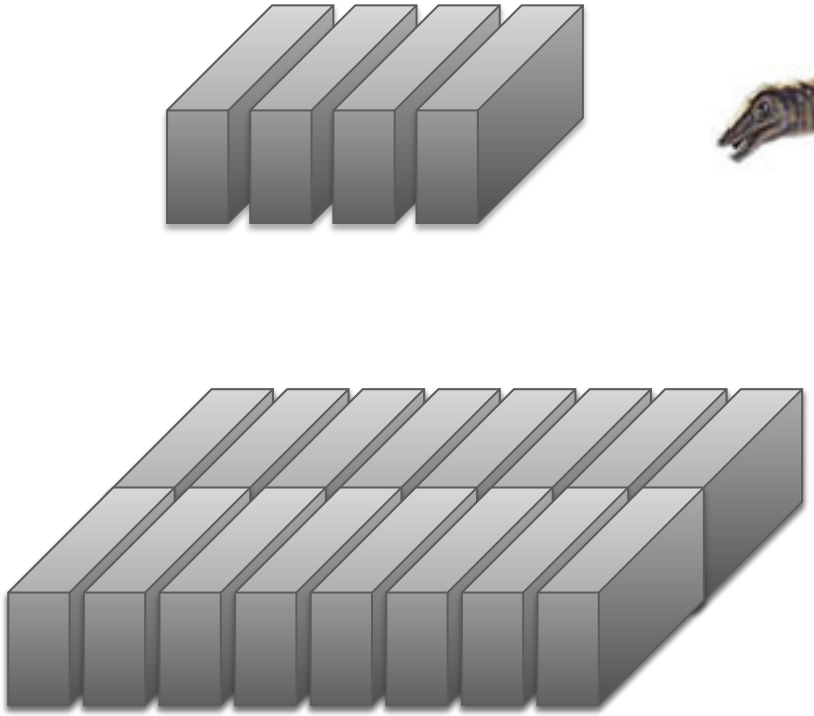
CUBE

- Parallel program analysis report exploration tools
  - Libraries for XML report reading & writing
  - Algebra utilities for report processing
  - GUI for interactive analysis exploration
- Available under New BSD open-source license
- Documentation & Sources:
  - http://www.score-p.org
- User guide also part of installation:
  - `cube-config --cube-dir`/share/doc/CubeGuide.pdf
- Contact:
  - mailto: scalasca@fz-juelich.de

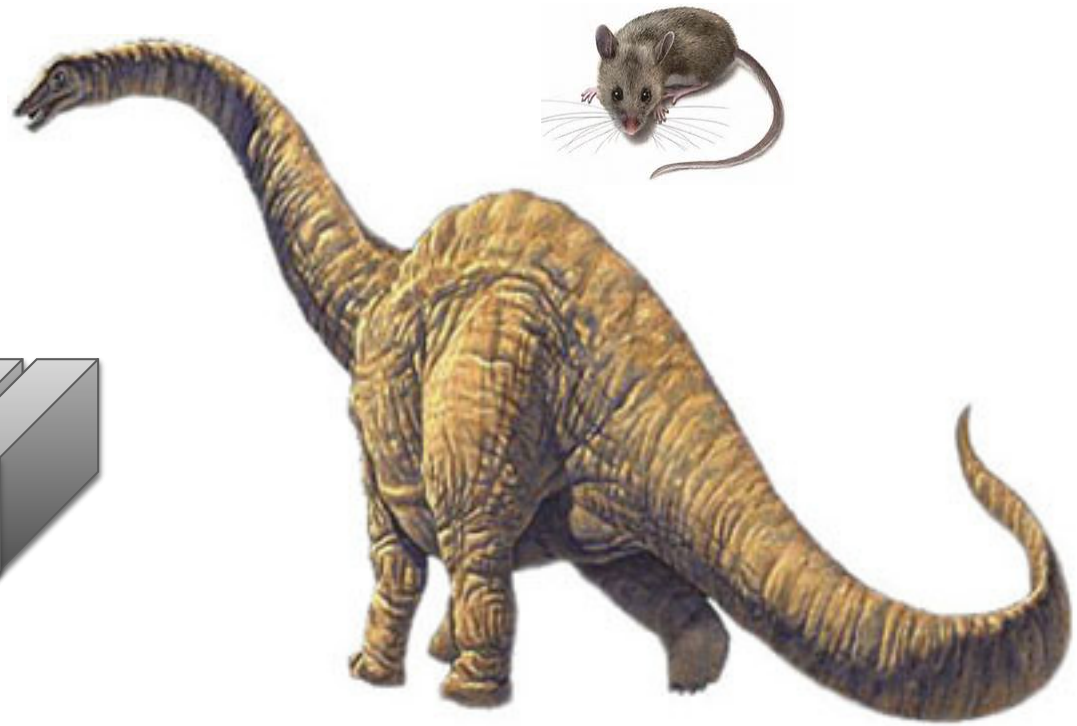
Extra-P

Motivation - latent scalability bugs

System size



Execution time

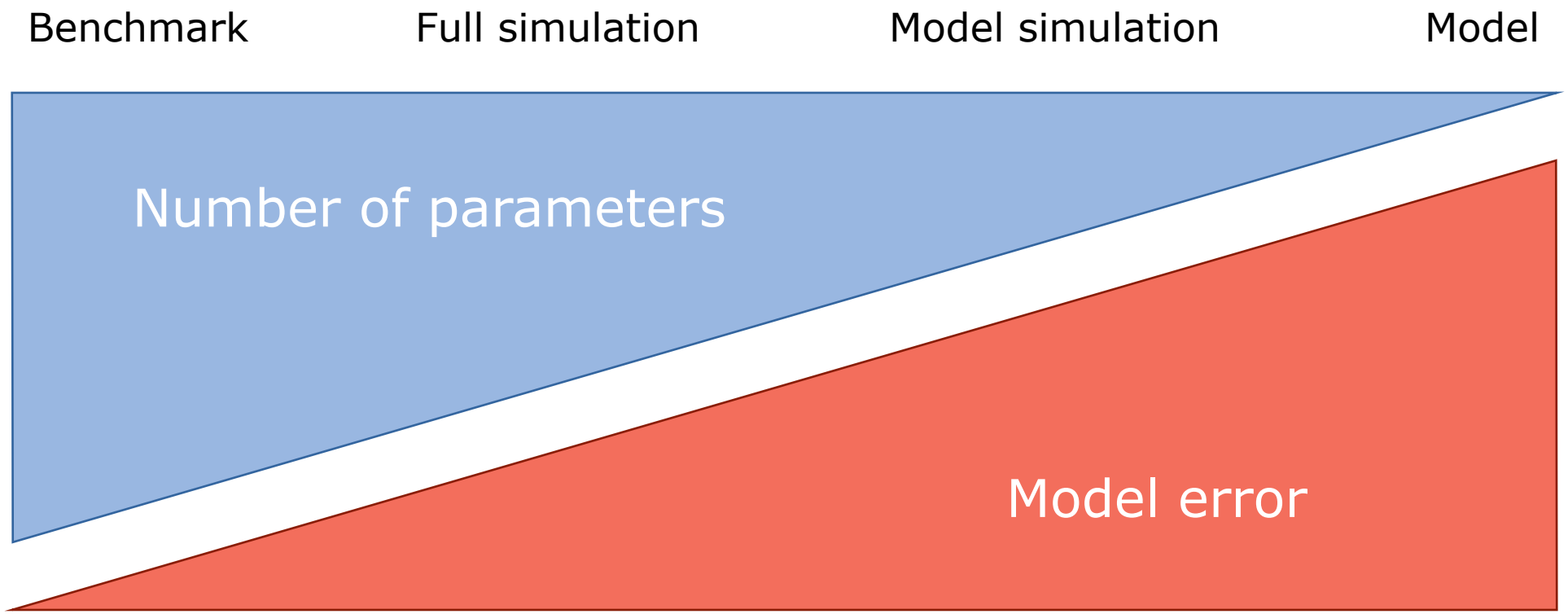


Introduction

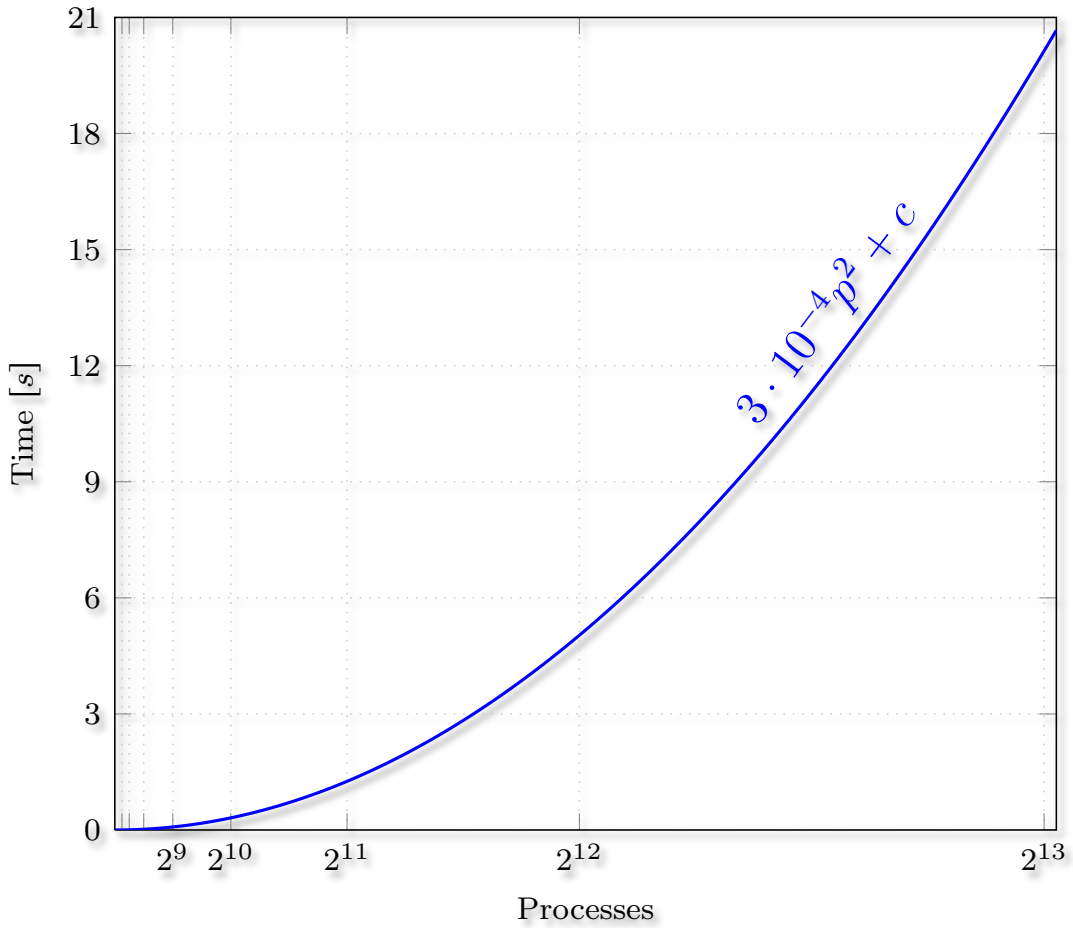
Outline

- Performance analysis methods
- Analytical performance modeling
- Automatic performance modeling
- Scalability validation framework

Spectrum of performance analysis methods



Scaling model



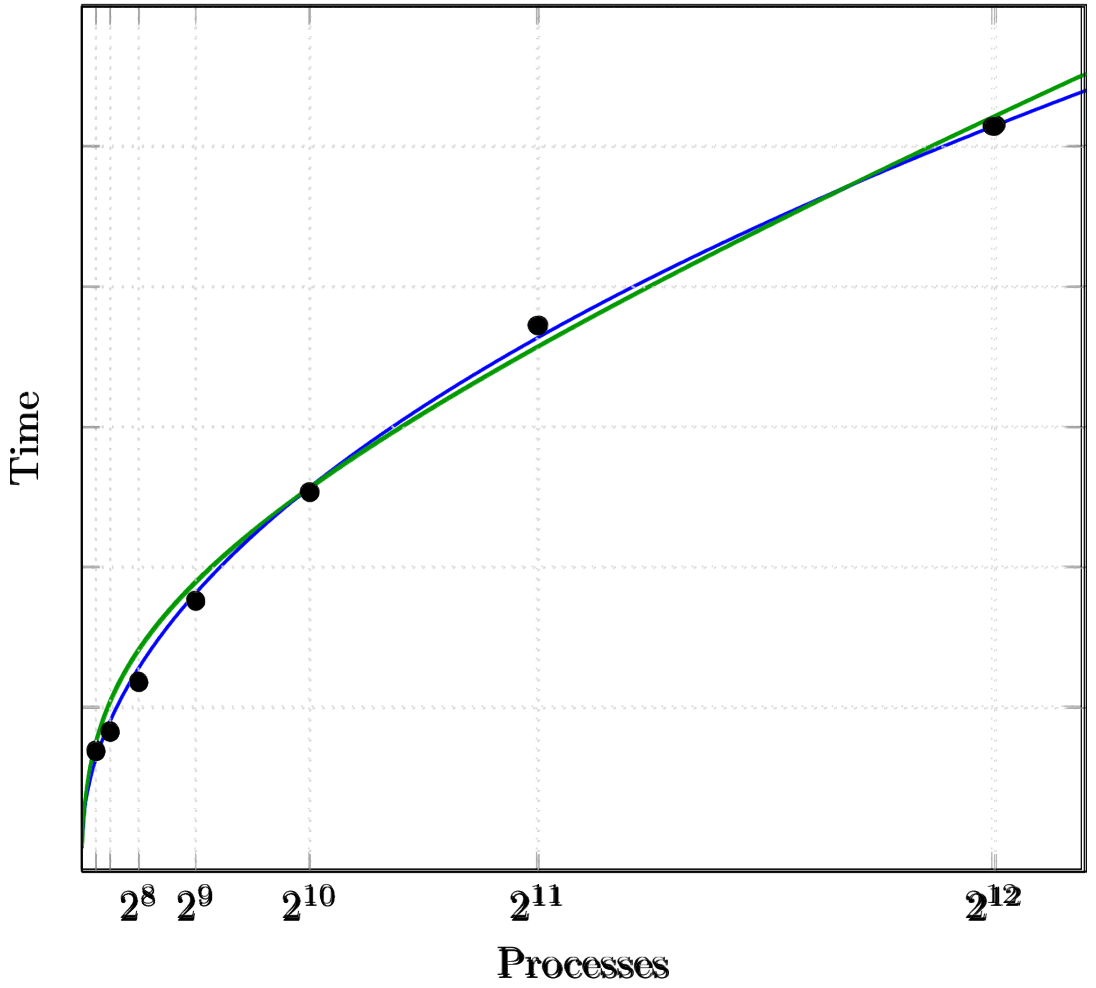
- Represents performance metric as a function of the number of processes
- Provides insight into the program behavior at scale

Pitfalls

Intuition is not enough

$$2.95 * \log^2 p + 0.0871 * p$$

$$12.06 * \sqrt{p}$$



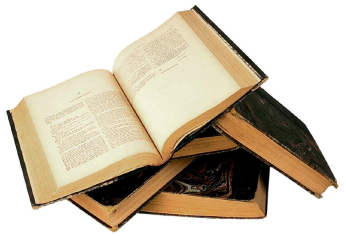
Analytical performance modeling

Identify kernels

- Parts of the program that dominate its performance at larger scales
- Identified via small-scale tests and intuition

Create models

- Laborious process
- Still confined to a small community of skilled experts



Examples:

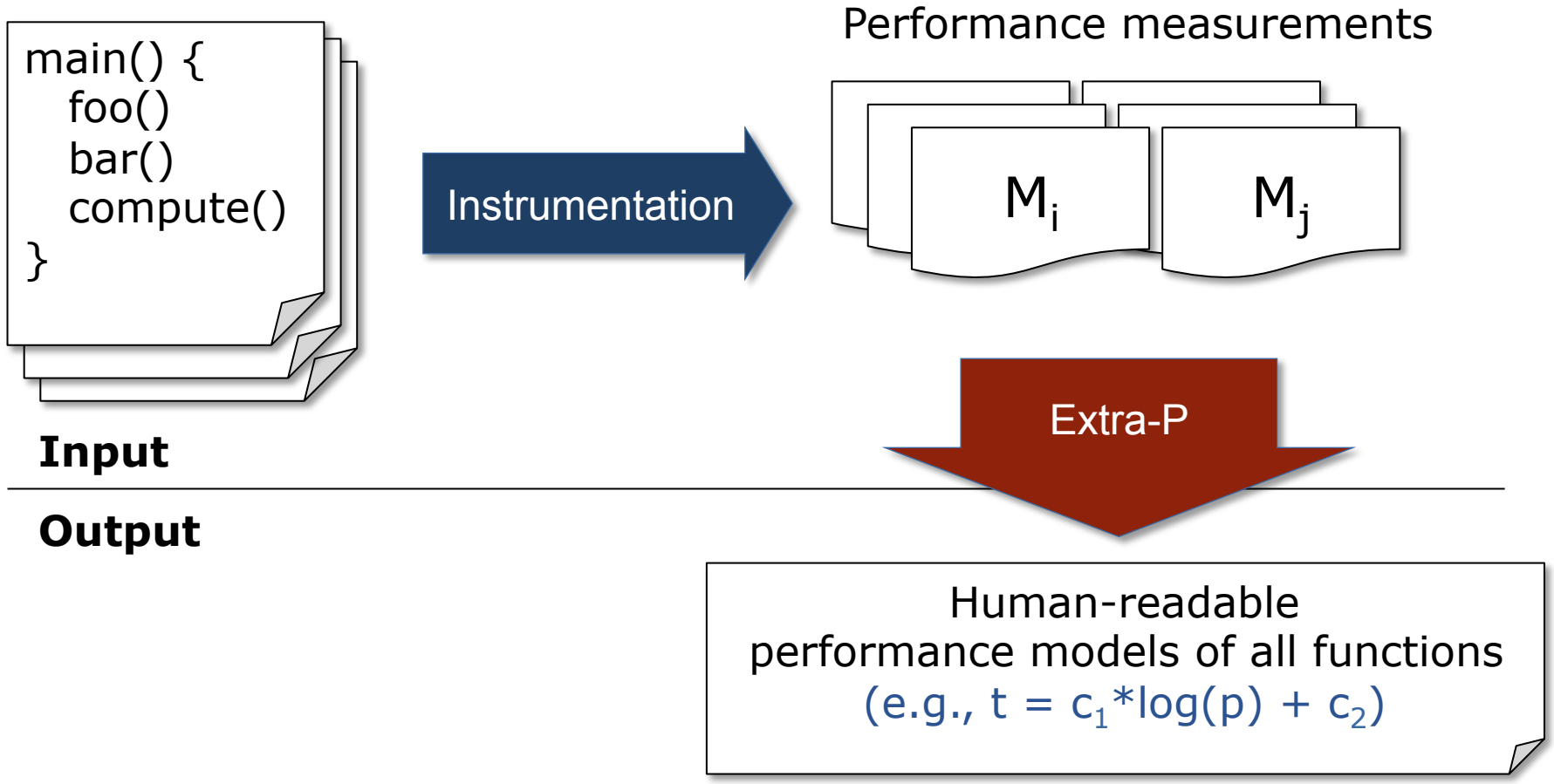
Hoisie et al.: *Performance and scalability analysis of teraflop-scale parallel architectures using multi-dimensional wavefront applications.* International Journal of High Performance Computing Applications, 2000

Bauer et al.: *Analysis of the MILC Lattice QCD Application su3_rmd*. CCGrid, 2012

Disadvantages:

- Time consuming
- Danger of overlooking unscalable code

Automatic performance modeling with Extra-P



Automatic performance modeling with Extra-P

```
main() {
   foo()
   bar()
   compute()
}
```

Input

Instrumentation

- All functions

Performance measurements (profiles)

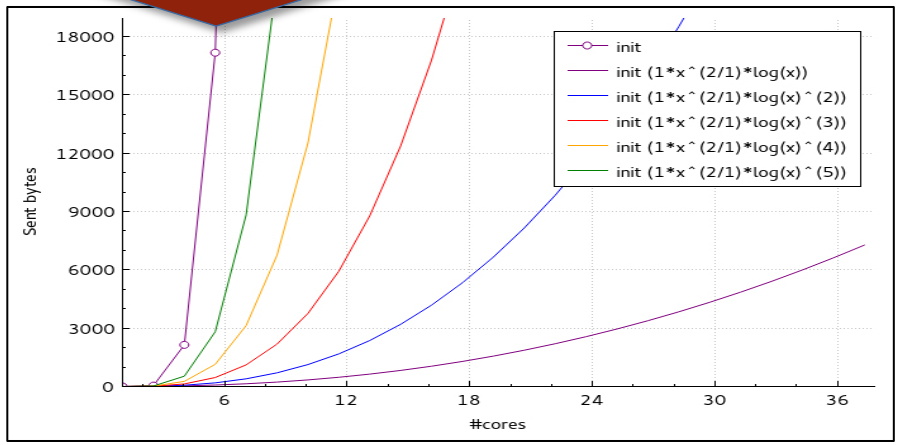
$p_1 = 128$ $p_4 = 1{,}024$
$p_2 = 256$ $p_5 = 2{,}048$
$p_3 = 512$ $p_6 = 4{,}096$

Extra-P

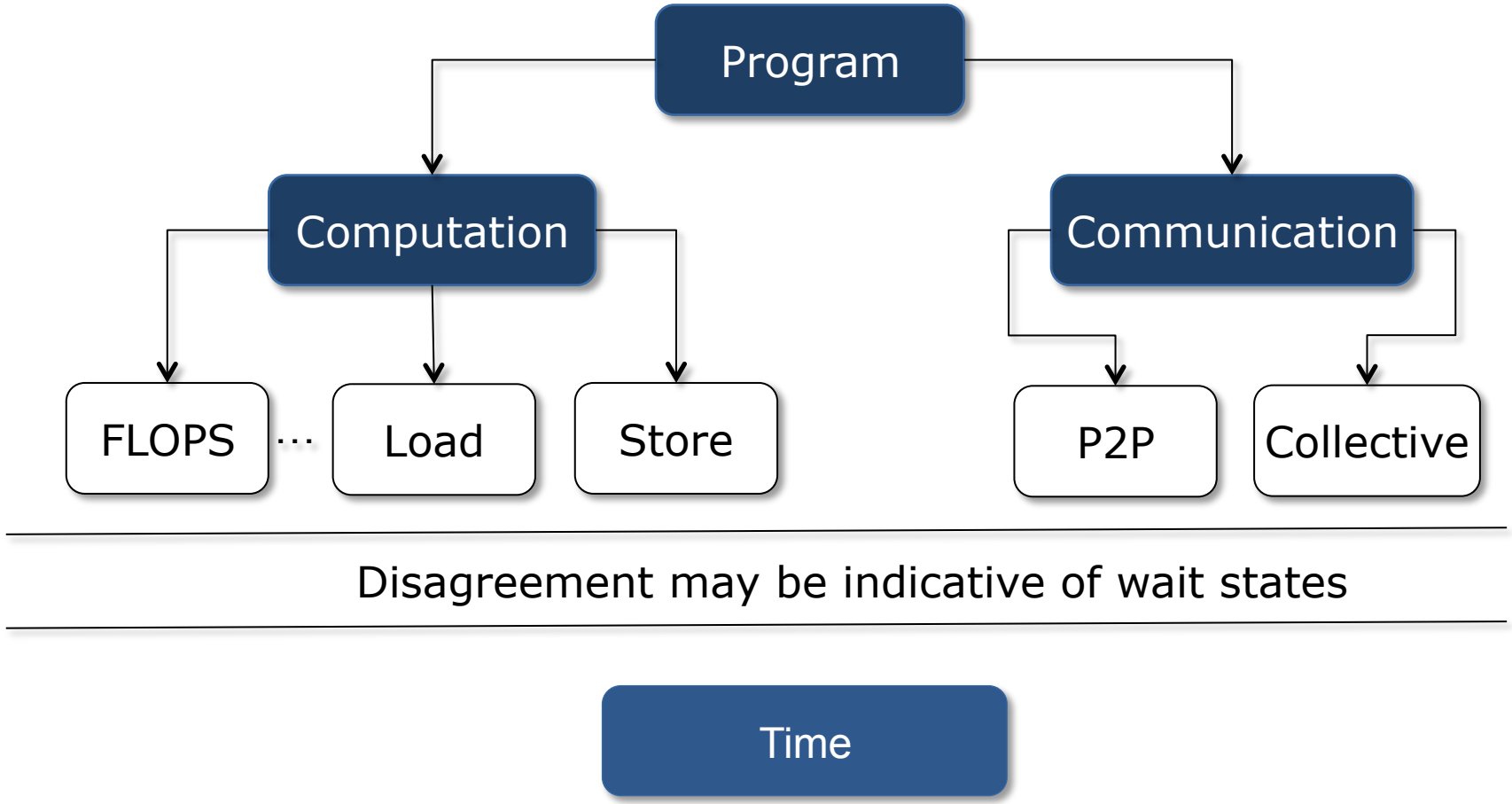
Output

Ranking:
- Target scale p_t
- Asymptotic

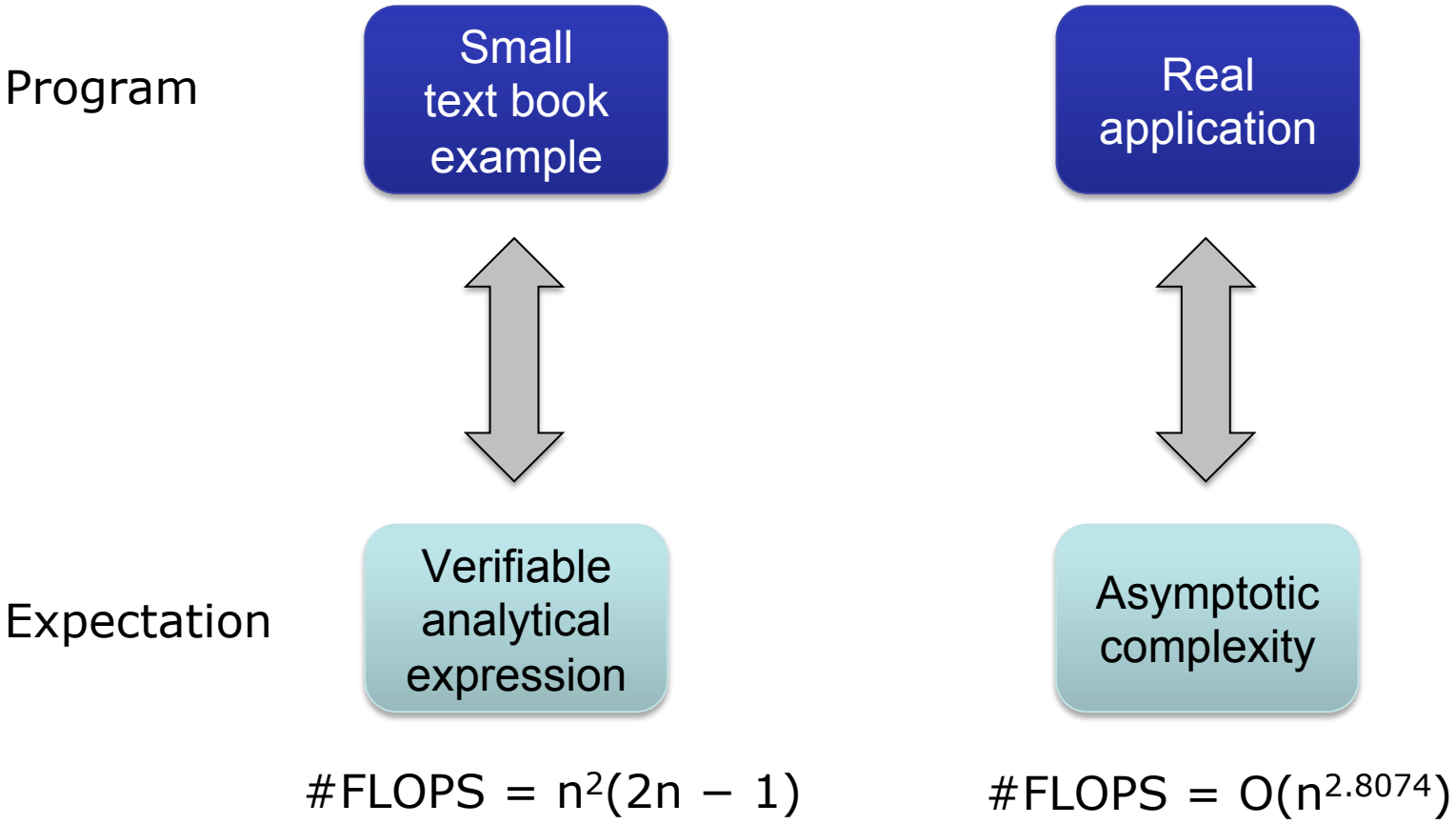
1. foo
2. compute
3. main
4. bar
[…]



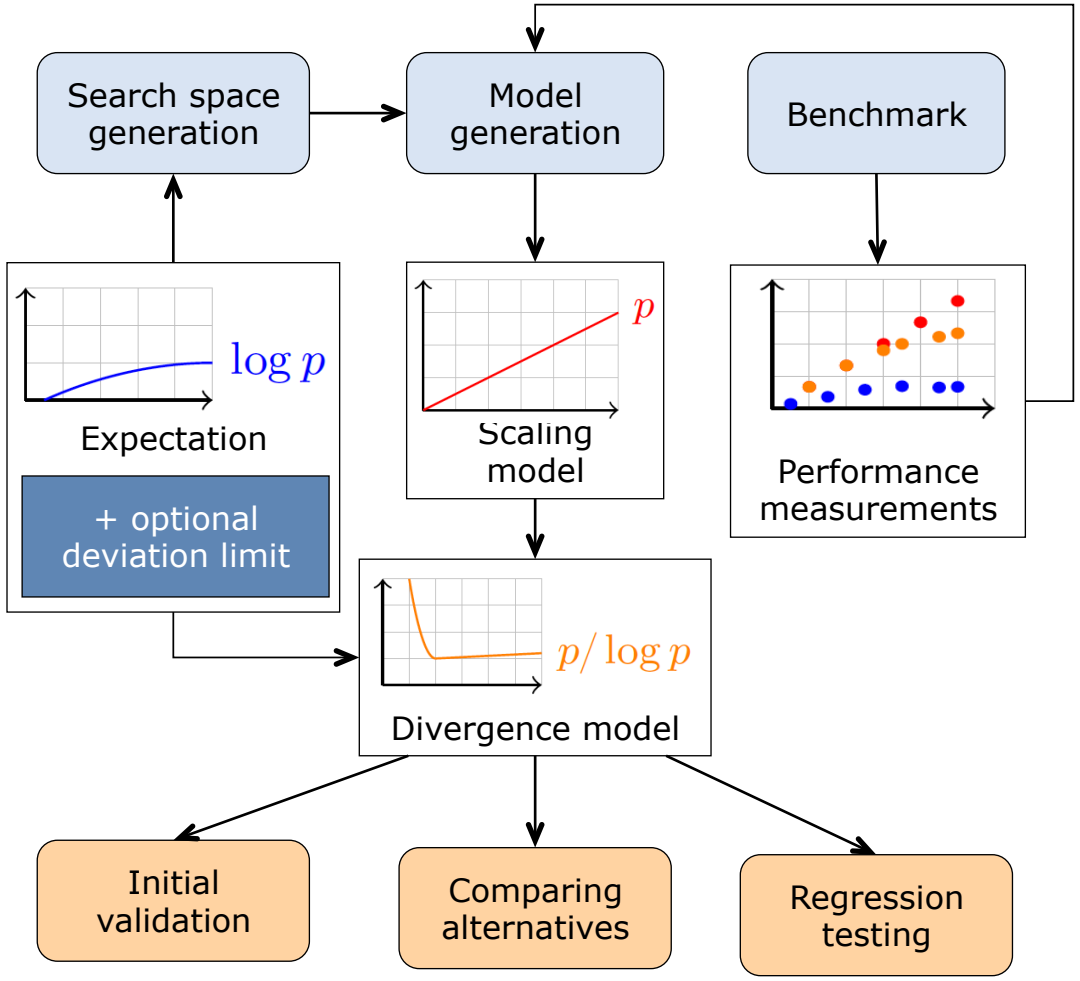
Requirements modeling



How to validate scalability in practice?



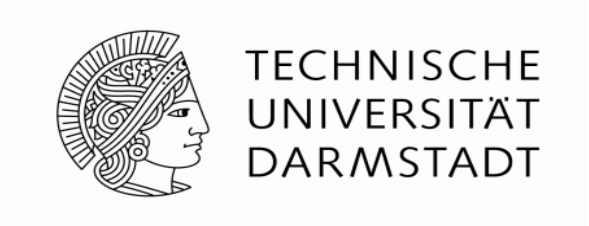
Scalability evaluation framework



Shudler et al: Exascaling Your Library: Will Your Implementation Meet Your Expectations?.

ICS 2015

Theory


TECHNISCHE
UNIVERSITÄT
DARMSTADT

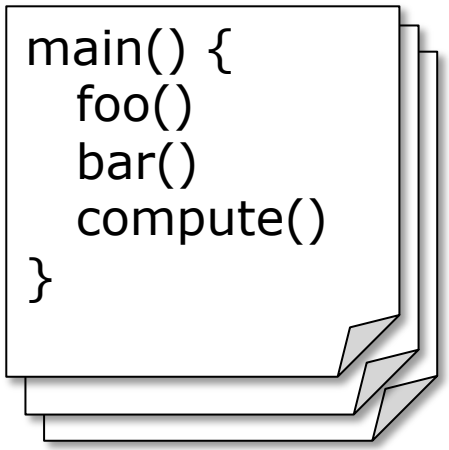

ETH zürich


Lawrence Livermore
National Laboratory

Outline

- Goal – scaling trends
- Model generation
- Performance Model Normal Form (PMNF)
- Statistical quality control & confidence intervals
- Assumptions & limitations of the method

Automatic performance modeling

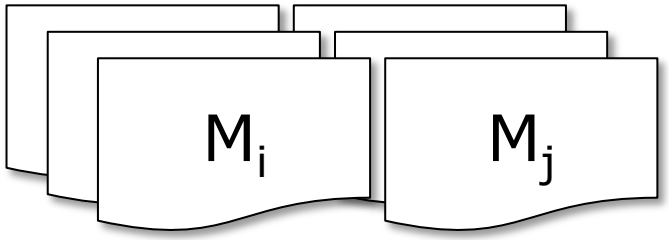


Input

Instrumentation

- All functions

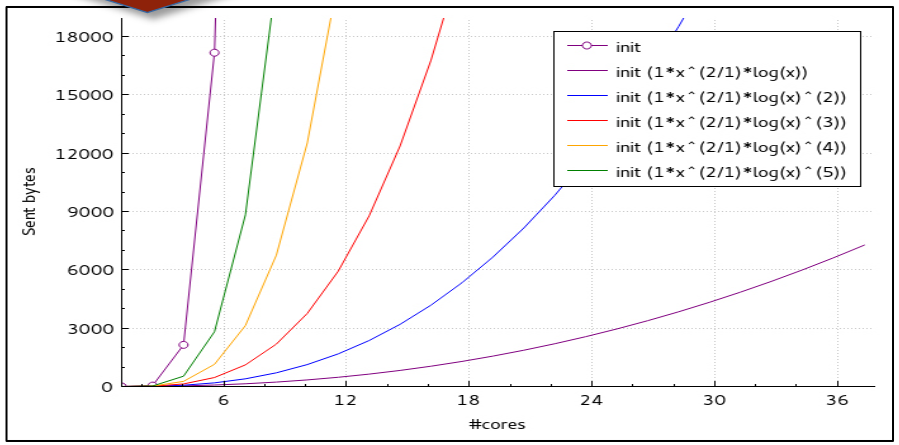
Performance measurements



Extra-P

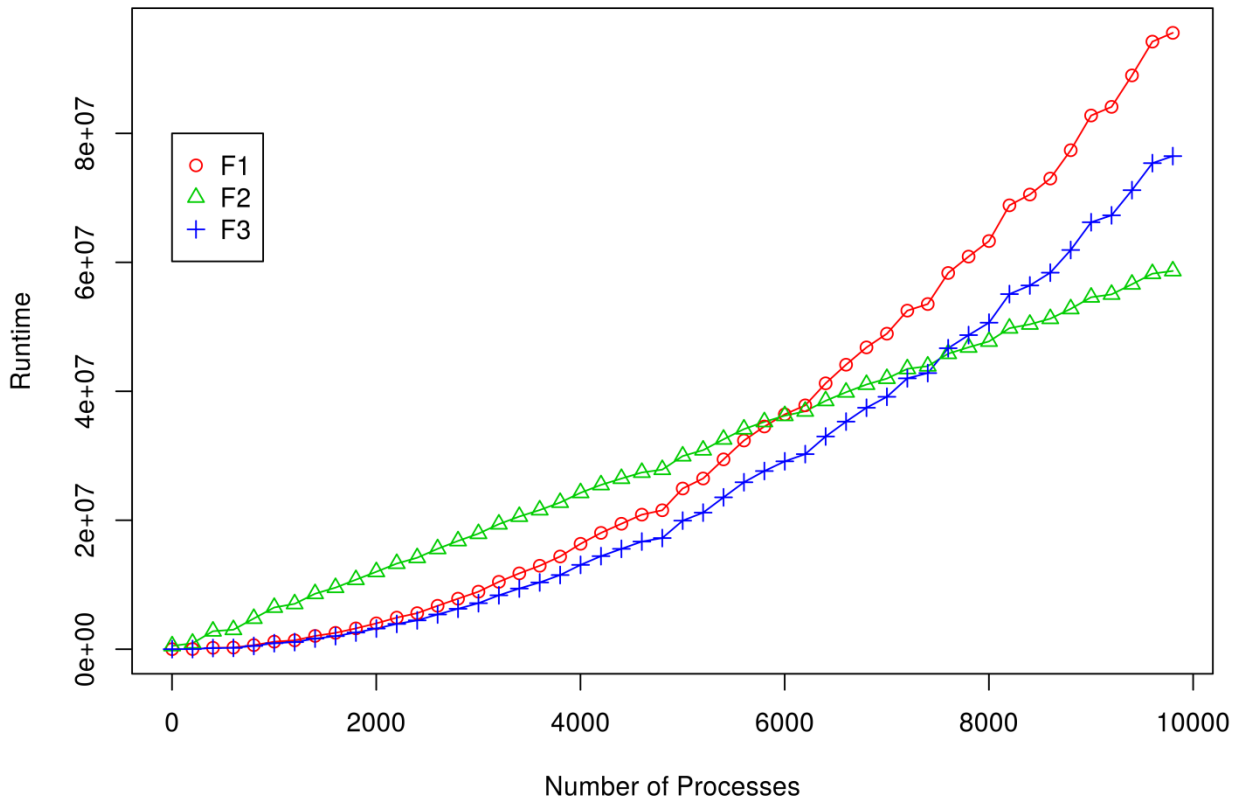
Output

Human-readable performance models of all functions (e.g., $t = c_1 * \log(p) + c_2$)



Primary focus on scaling trend

Common performance analysis chart in a paper

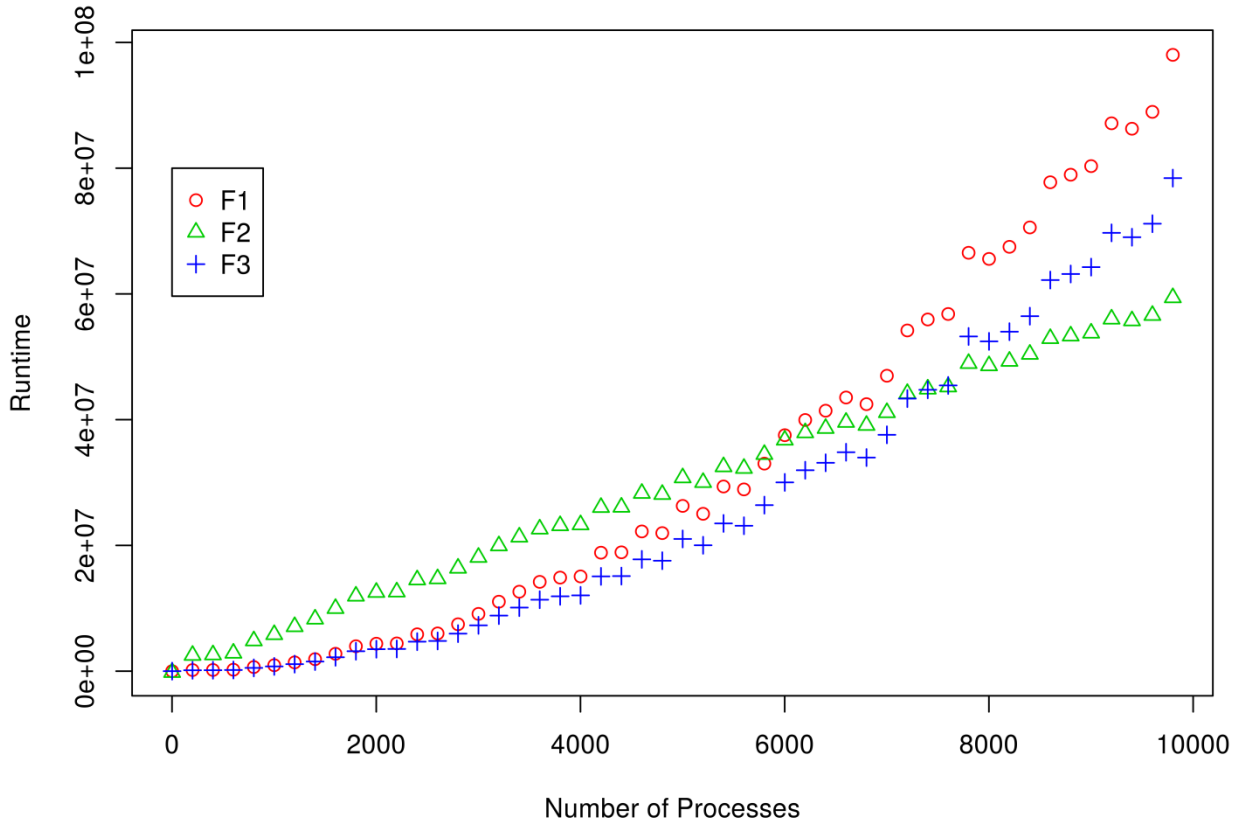


Ranking

1. F_2
2. F_1
3. F_3

Primary focus on scaling trend

Actual measurement in laboratory conditions

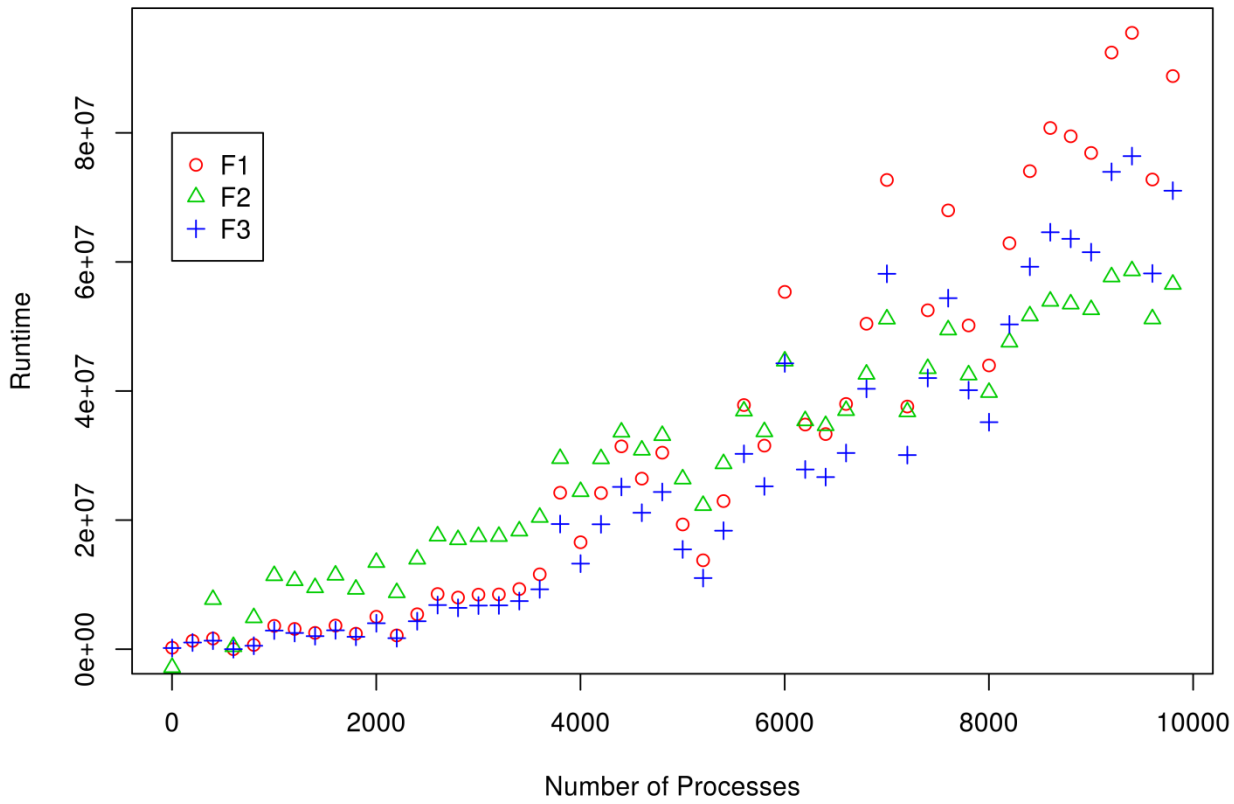


Ranking

1. F_2
2. F_1
3. F_3

Primary focus on scaling trend

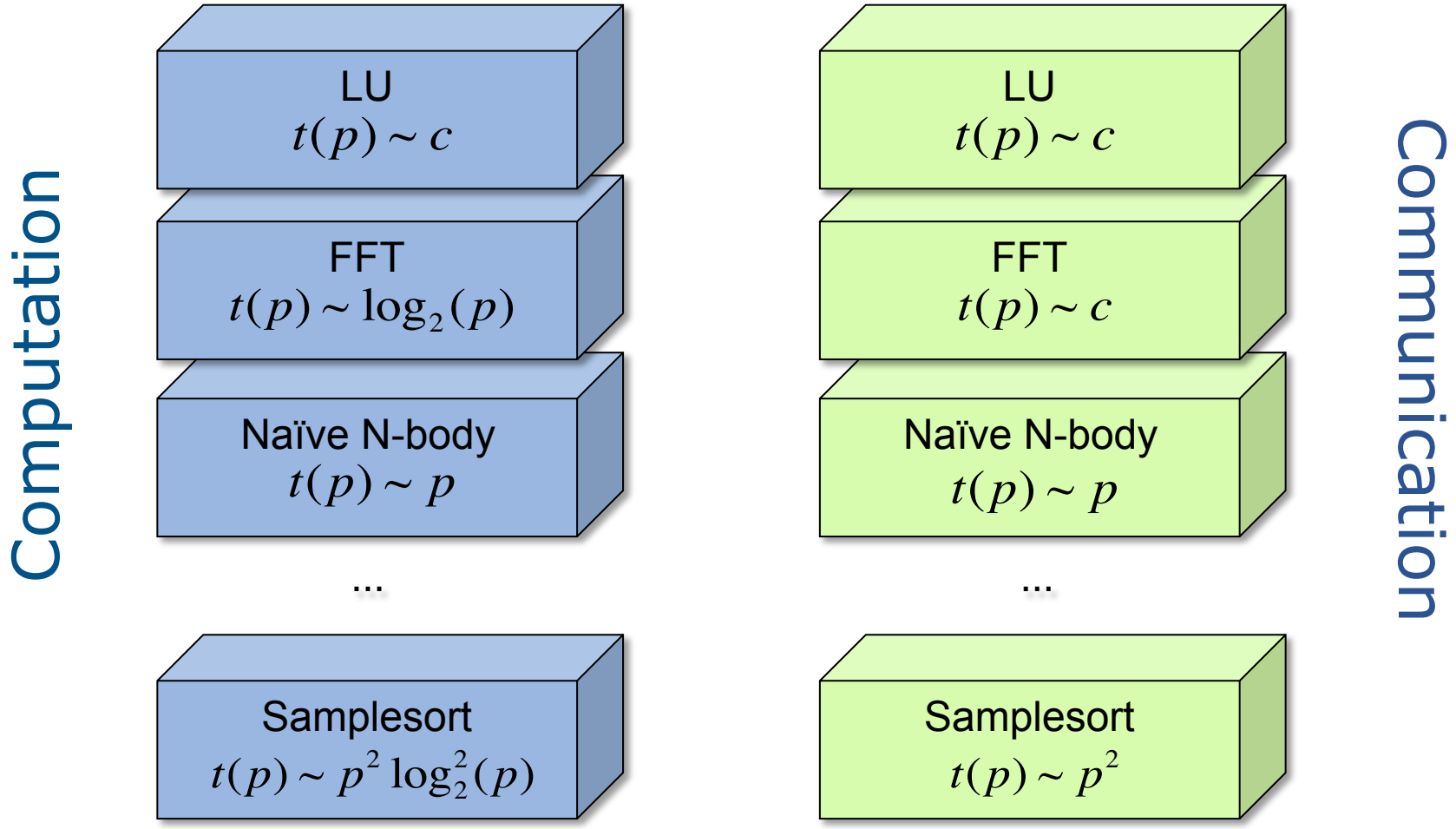
Production Reality



Ranking

1. F_2
2. F_1
3. F_3

Model building blocks

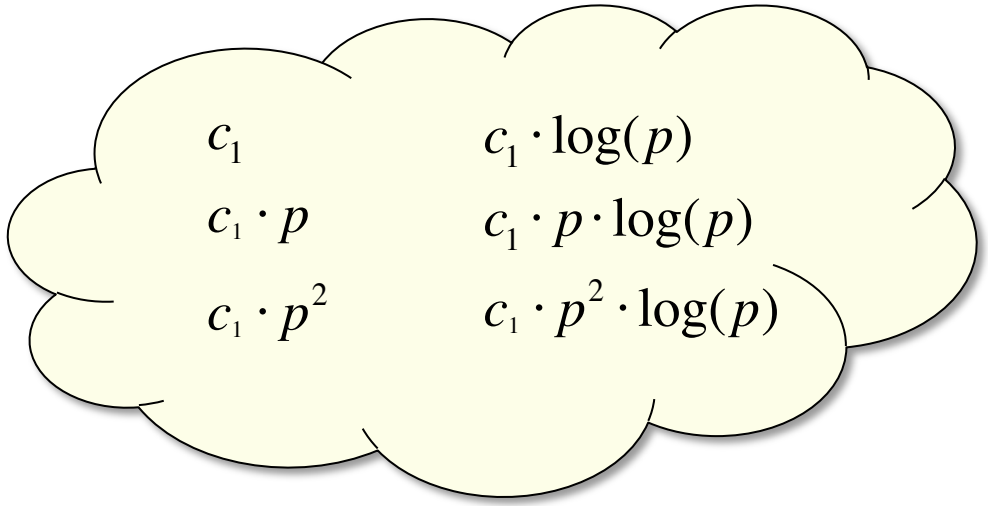


Performance model normal form

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

$$n \in \mathbb{N}$$
$$i_k \in I$$
$$j_k \in J$$
$$I, J \subset \mathbb{Q}$$

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



Performance model normal form

$$f(p) = \sum_{n} \; \cdot \log(p)^{n}$$

$n = 1$

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

$J = \{0,1\}$

$c_1 + c_2 \cdot p$

$c_1 + c_2 \cdot p^2$

$c_1 + c_2 \cdot \log(p)$

$c_1 + c_2 \cdot p \cdot \log(p)$

$c_1 + c_2 \cdot p^2 \cdot \log(p)$

$c_1 \cdot \log(p) + c_2 \cdot p$

$c_1 \cdot \log(p) + c_2 \cdot p \cdot \log(p)$

$c_1 \cdot \log(p) + c_2 \cdot p^2$

$c_1 \cdot \log(p) + c_2 \cdot p^2 \cdot \log(p)$

$c_1 \cdot p + c_2 \cdot p \cdot \log(p)$

$c_1 \cdot p + c_2 \cdot p^2$

$c_1 \cdot p + c_2 \cdot p^2 \cdot \log(p)$

$c_1 \cdot p \cdot \log(p) + c_2 \cdot p^2$

$c_1 \cdot p \cdot \log(p) + c_2 \cdot p^2 \cdot \log(p)$

$c_1 \cdot p^2 + c_2 \cdot p^2 \cdot \log(p)$

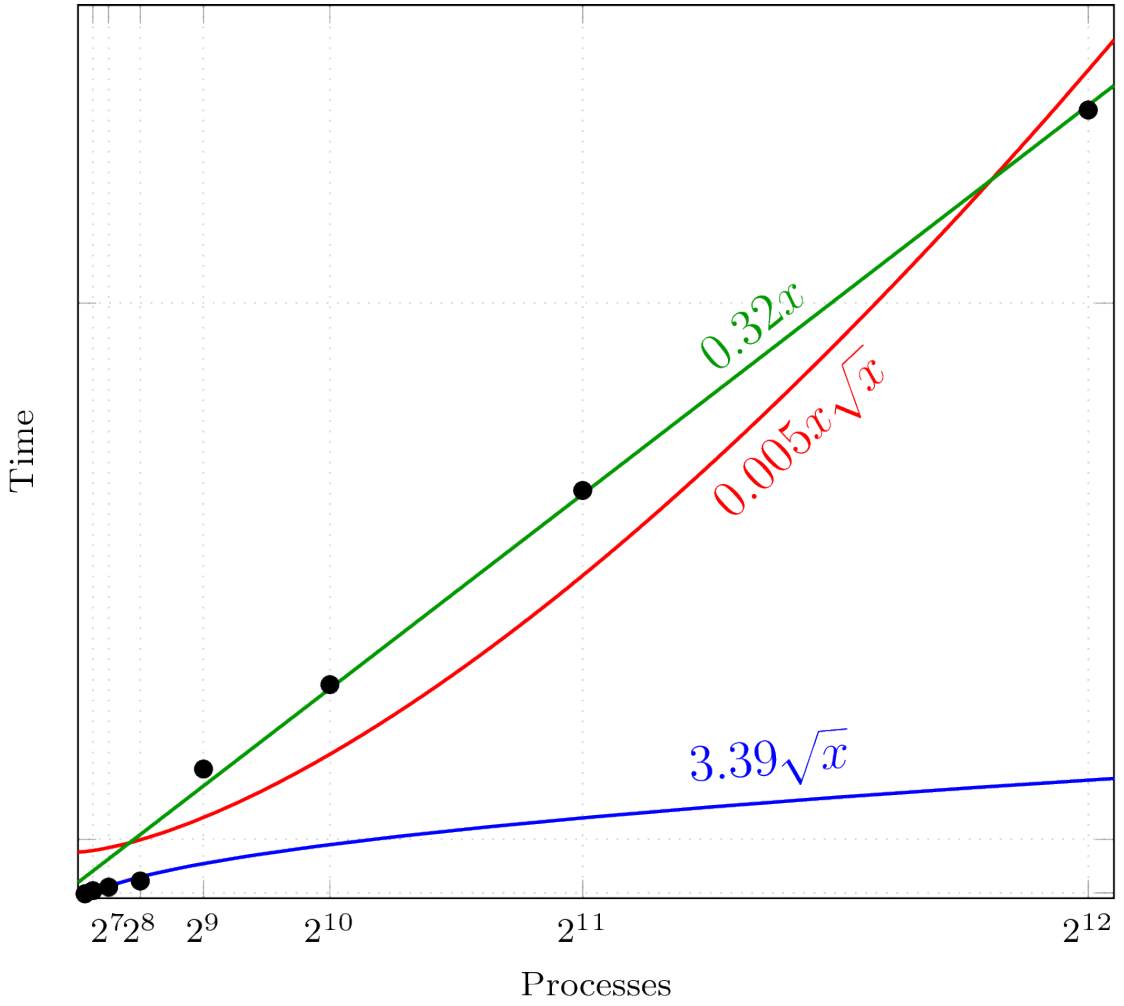
Weak vs. strong scaling

- Wall-clock time not necessarily monotonically increasing under strong scaling
  - Harder to capture model automatically
  - Different invariants require different reductions across processes

	Weak scaling	Strong scaling
Invariant	Problem size per process	Overall problem size
Model target	Wall-clock time	Accumulated time
Reduction	Maximum / average	Sum

Assumptions & limitations

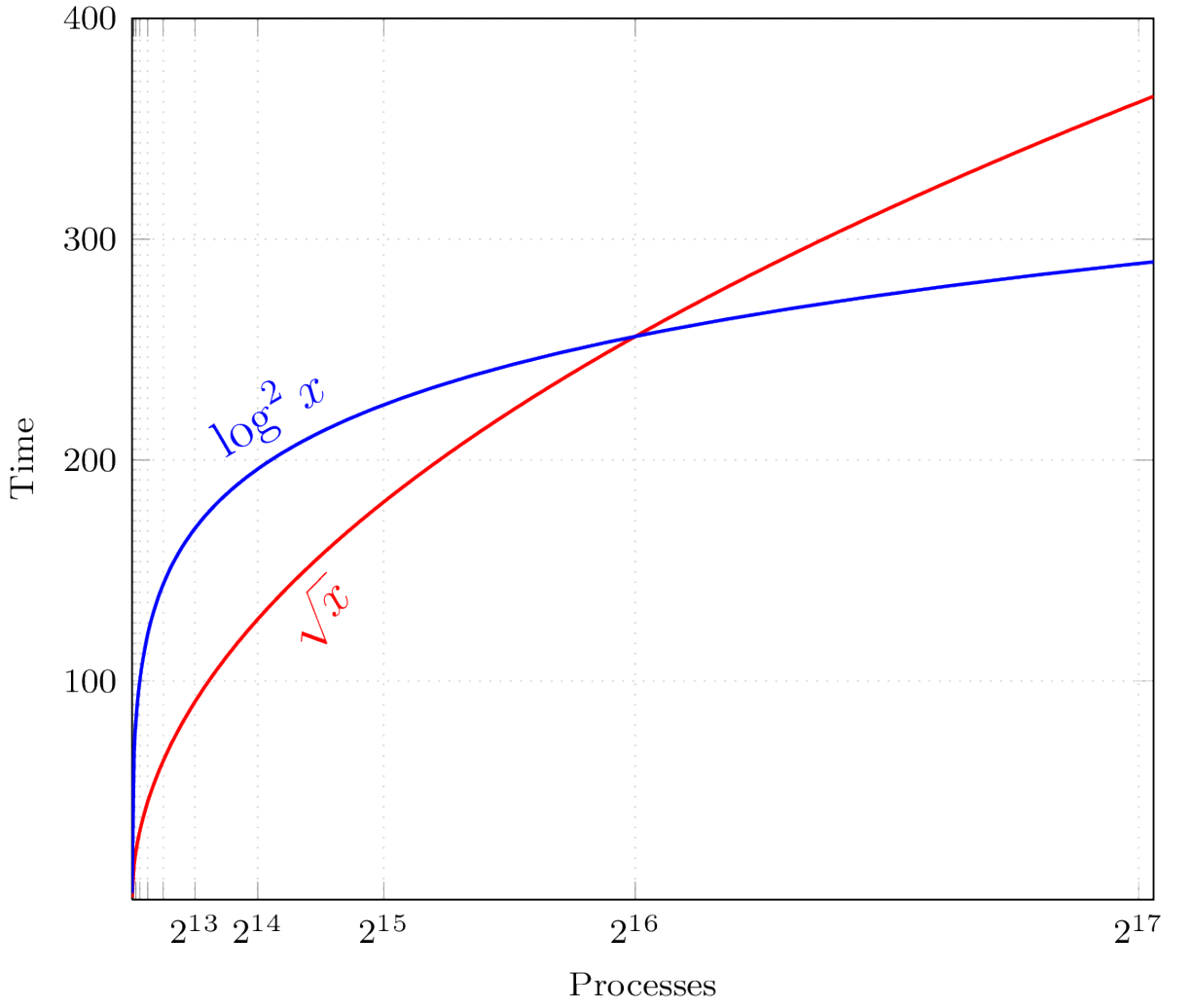
- Only one scaling behavior for all the measurements; no jumps
- Some MPI collective operations switch their algorithm – results in inaccurate models
- Example: red model tries to model measurements of different algorithms
  - First 4 points – one function
  - Last 4 points – another function (linear)
  - Adj. R2 = 0.95085 (!)



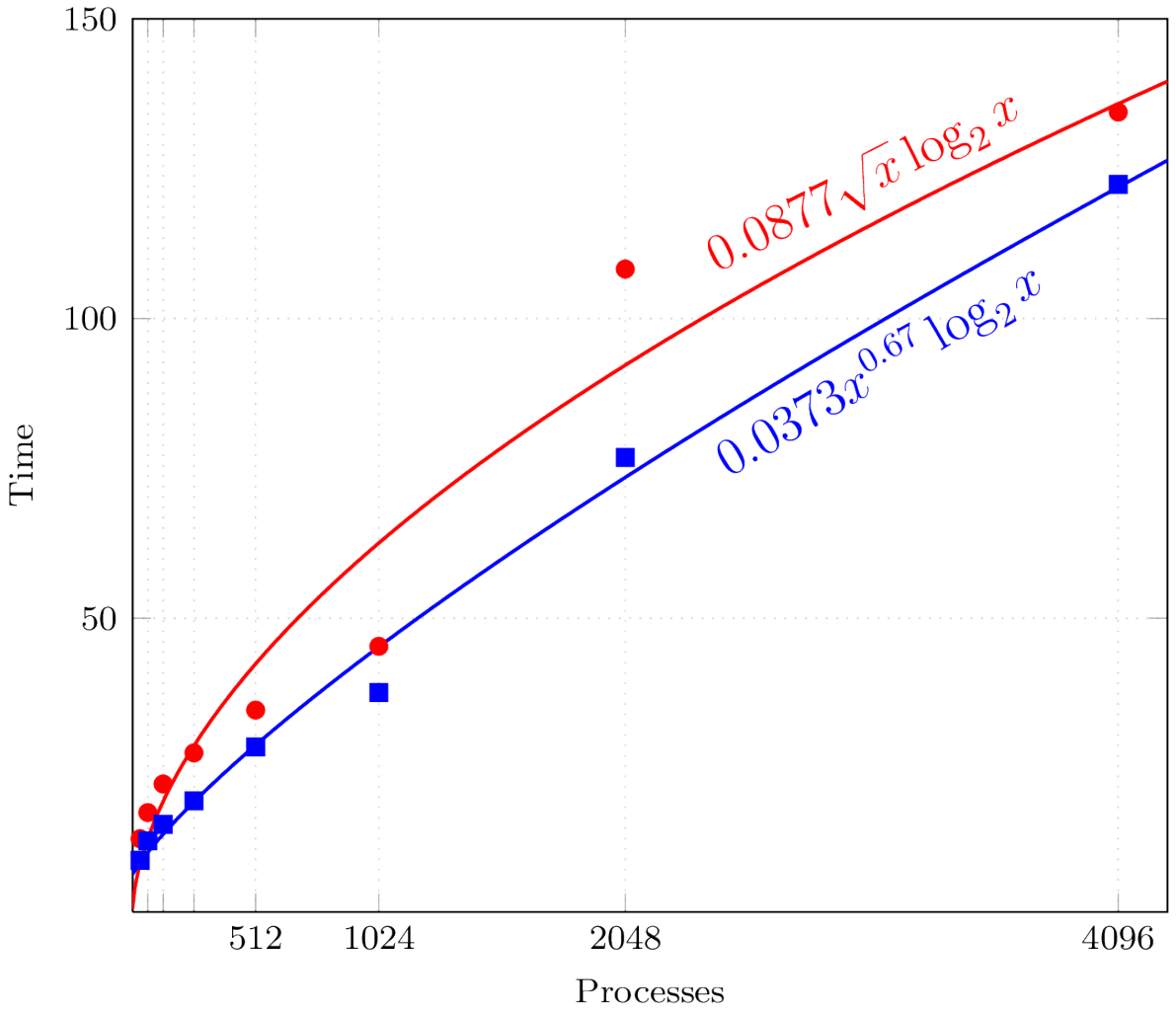
Changing growth trends

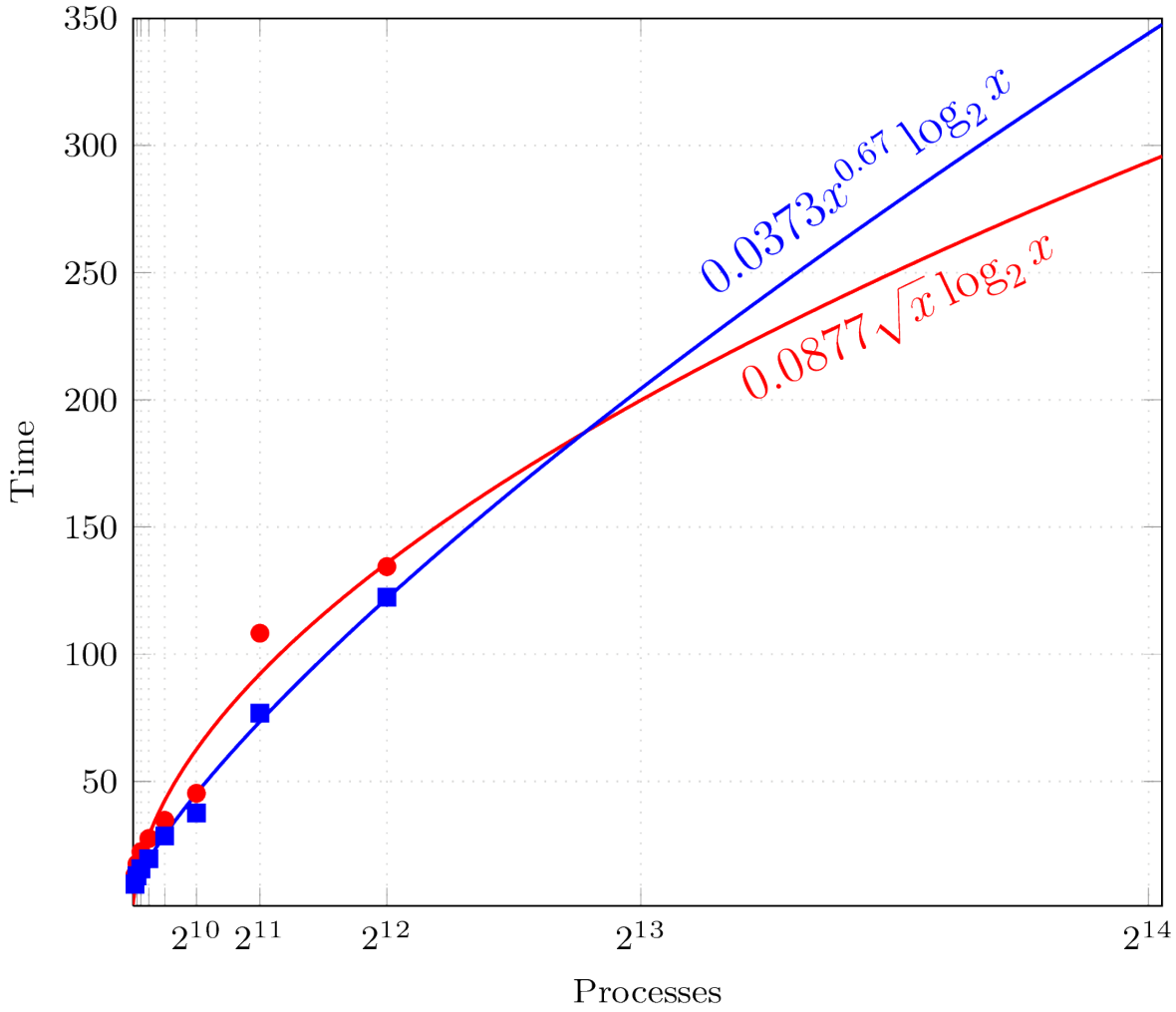
- Ranking according to growth rate difficult:

  $\log^2(p)$? $\sqrt{p}$



Changing growth trends (2)



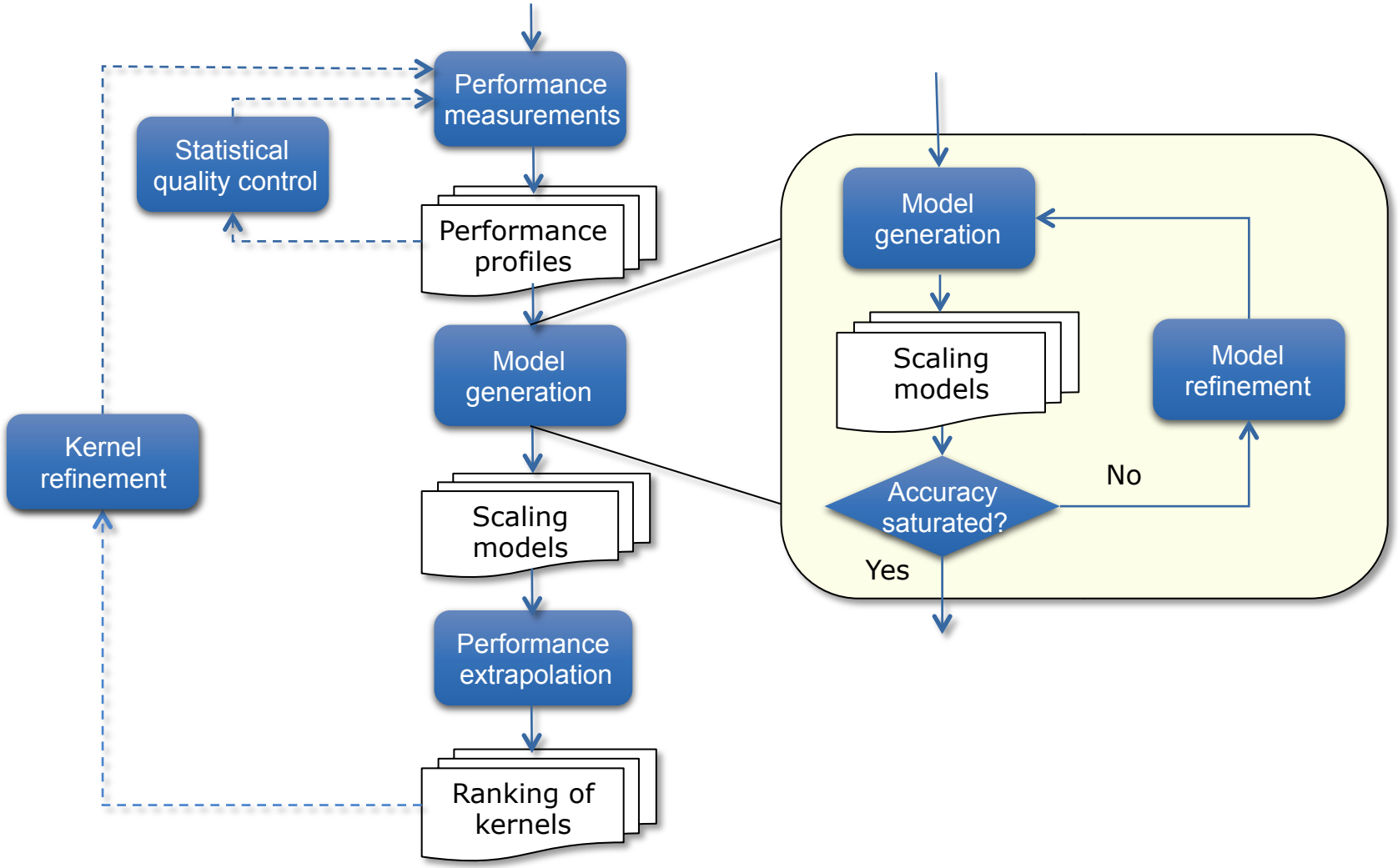


Practice

Outline

- Workflow
- Performance measurements
- Model refinement
- Adjusted R^2
- Kernel ranking
- Output representations

Workflow


Performance measurements
Statistical quality control
Performance profiles
Model generation
Scaling models
Performance extrapolation
Ranking of kernels
Kernel refinement
Model generation
Scaling models
Accuracy saturated?
No
Yes
Model refinement

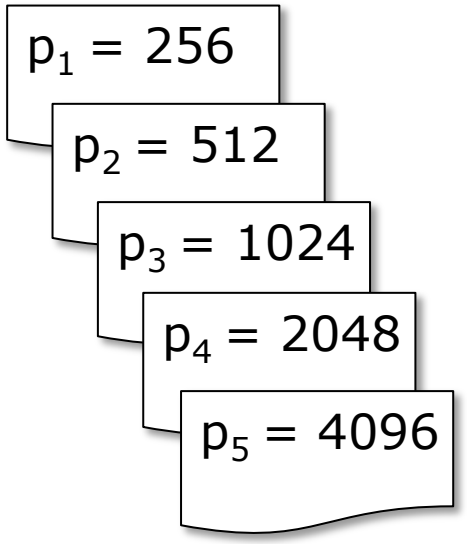
Performance measurements

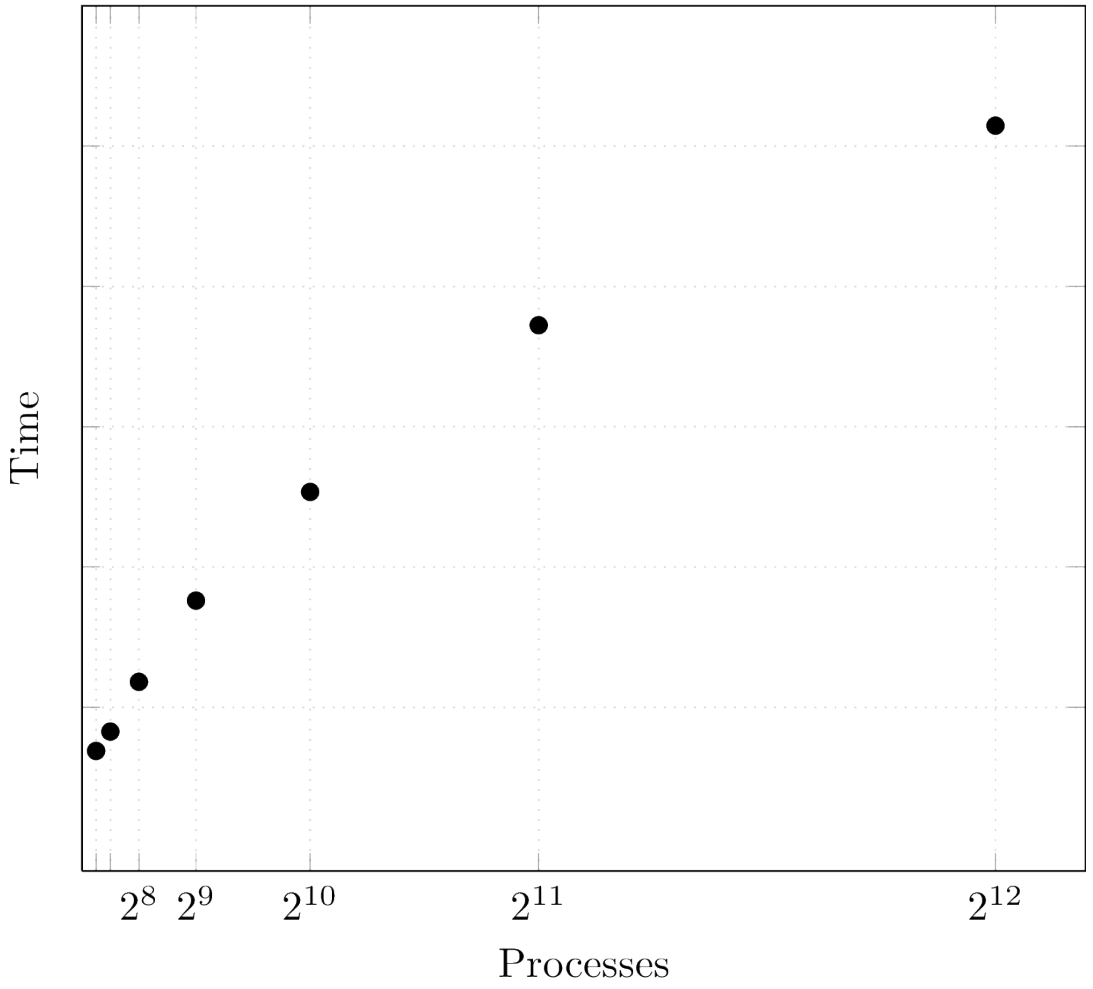
- Different ways of collecting measurements
- Score-P (http://www.vi-hps.org/projects/score-p/)
- Other profiling tools, e.g., HPCToolkit
- Manual ad-hoc measurements

Performance measurements (2)

- Our experience shows at least 5 different measurements required

Performance measurements (profiles)

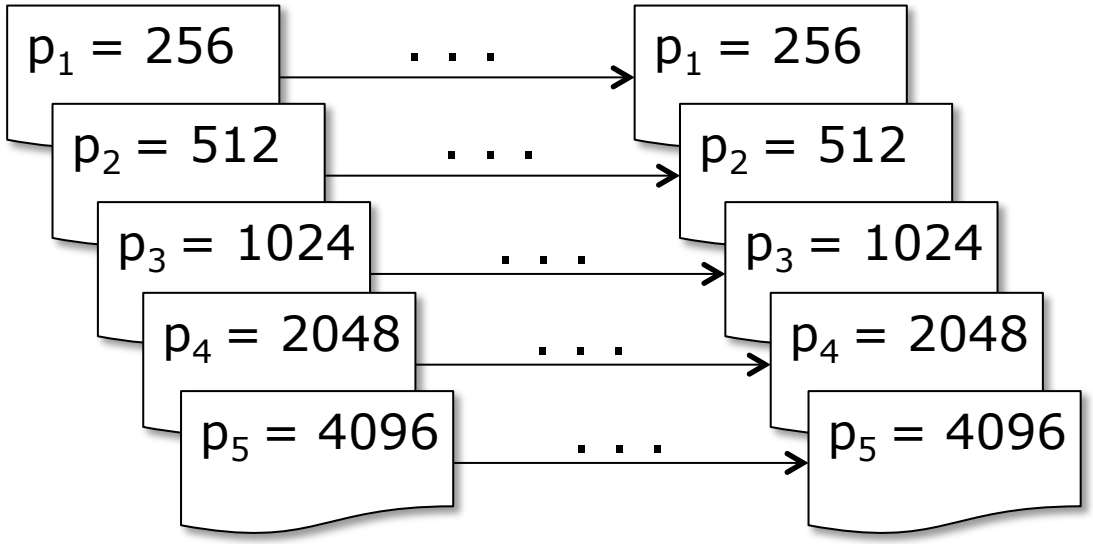


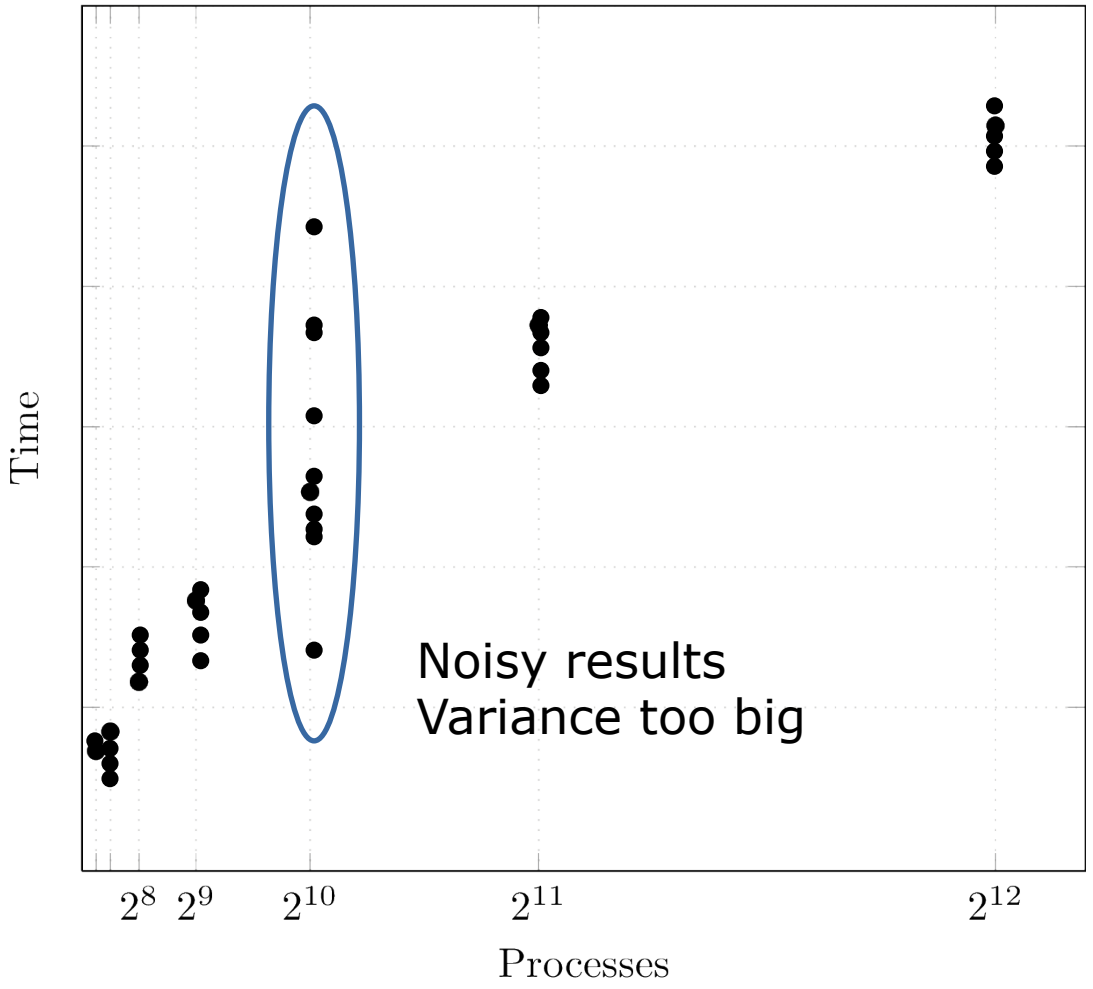


Performance measurements (3)

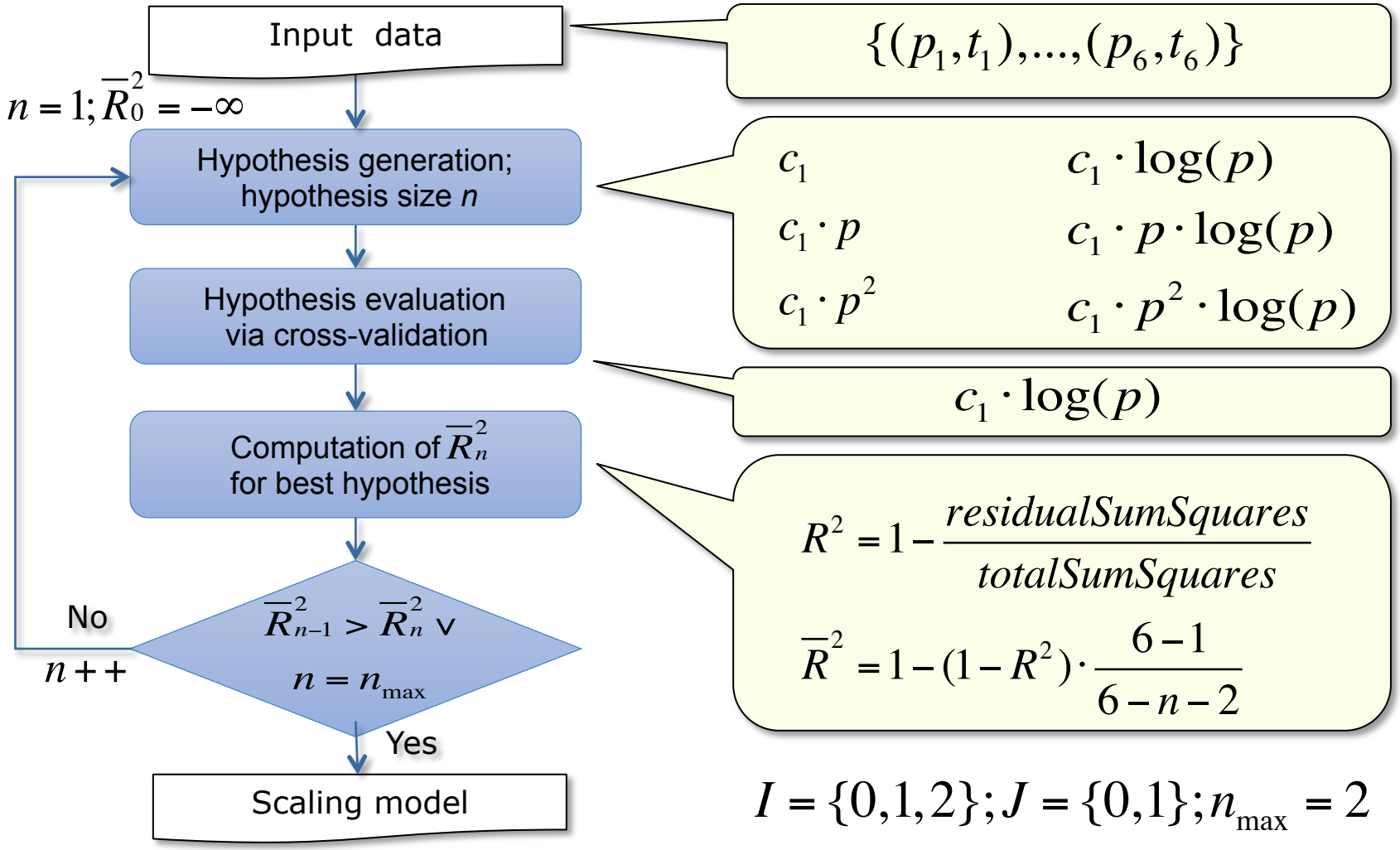
- Our experience shows at least 5 different measurements required
- Each measurement repeated multiple times

Performance measurements (profiles)





Model refinement



Adjusted R^2

- R^2 represents how well the determined function fits the M available measurements
- Adjusted R^2 adjusts for N, the number of terms used
  - Adj. R^2 decreases → more useless variables
  - Adj. R^2 increases → more useful variables
- Rule of thumb: adj. $R^2 > 0.95$

$$R^2 = 1 - \frac{residualSumSquares}{totalSumSquares}$$

$$\overline{R}^2 = 1 - (1 - R^2) \cdot \frac{M - 1}{M - N - 2}$$

Ranking of kernels

- Kernels are ranked according the leading-order terms in the models
- Leading-order term → big-O notation
- For example: $O(x)$ comes before $O(x^2)$

Output possibilities

- Textual
- Graphical with the use of Cube

Example of textual output

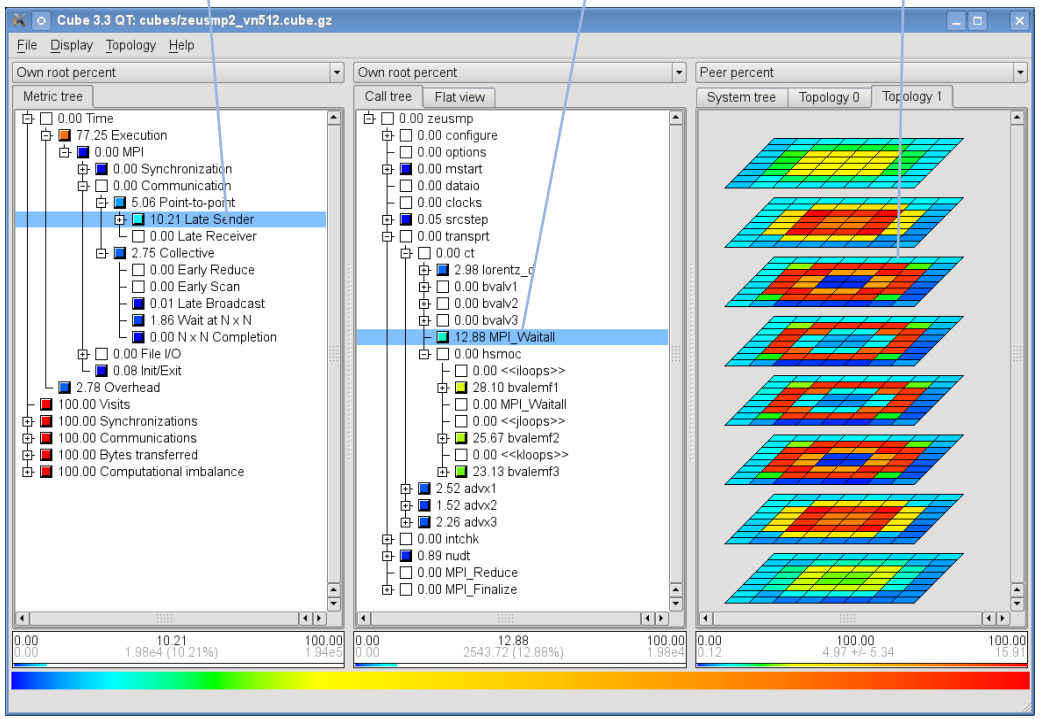
```
REGION: 0 reg1
Metric: Barrier
( 32,    8.1137 )+-(0.0, 0.0282888)
( 64,    12.7498 )+-(0.0, 0.0860876)
( 128,    19.1494 )+-(0.0, 0.151423)
( 256,    31.5625 )+-(0.0, 0.598003)
( 512,    65.8557 )+-(0.0, 3.47238)
( 1024,    98.912 )+-(0.0, 1.40606)
( 2048,    178.03 )+-(0.0, 6.134)
( 4096,    302.749 )+-(0.0, 3.01668)

AdjR2: 0.998942
RSS (fit error): 67.8154
Procentual relative error: 1.14834%
Model: (3.08757) + (0.0977357 * x^( 0.666667 )) * (log2(x))^( 1 ))

Metric: Bcast
( 32,    8.5346 )+-(0.0, 0.101737)
...

AdjR2: 0.983131
RSS (fit error): 490.025
Relative error: 3.99708%
Model: (-17.4593) + (3.38532 * x^( 0.5 ))

...
```

cube
scalasca

Which problem?

Where in the program?

Which process?

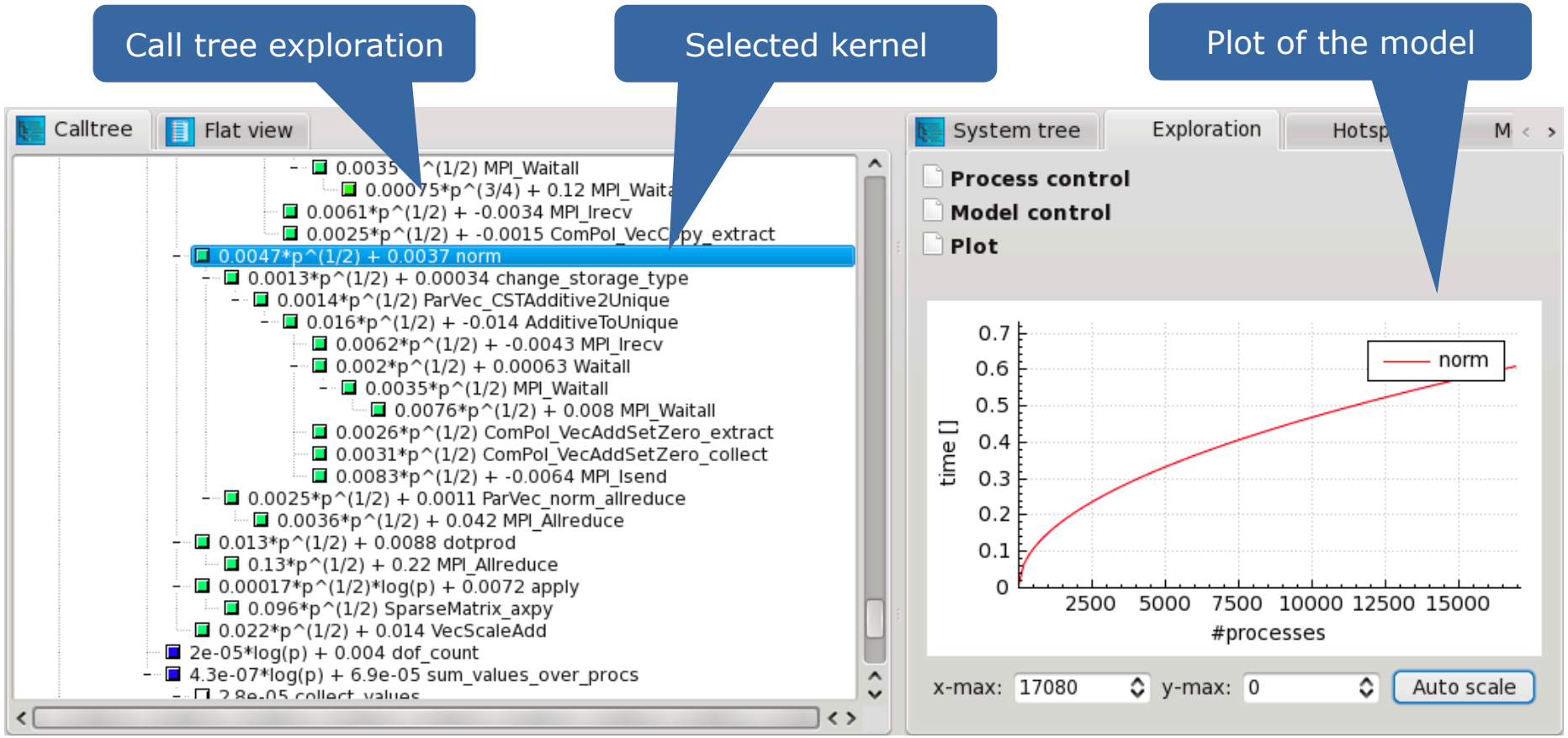


JÜLICH
FORSCHUNGSZENTRUM

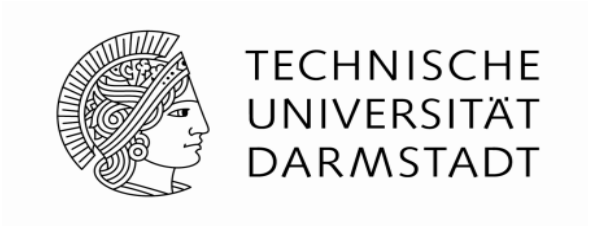
TECHNISCHE
UNIVERSITÄT
DARMSTADT

www.scalasca.org

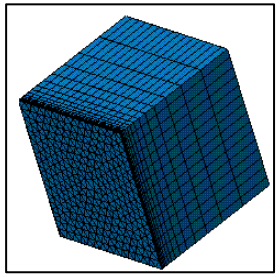
Visual performance exploration

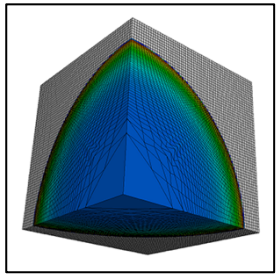


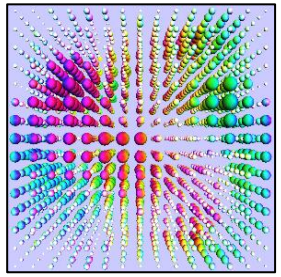
Case studies

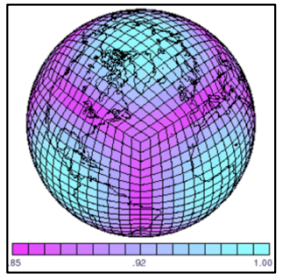


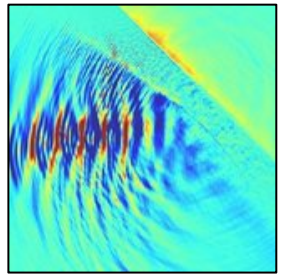
Case studies

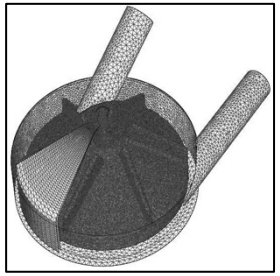

Sweep3d


Lulesh

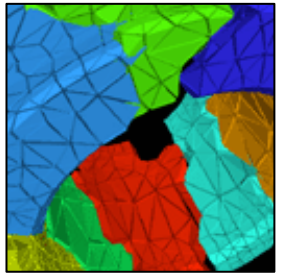

Milc

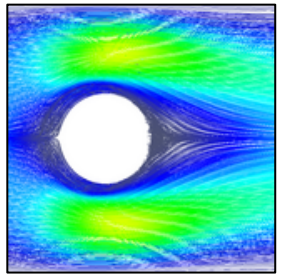

HOMME

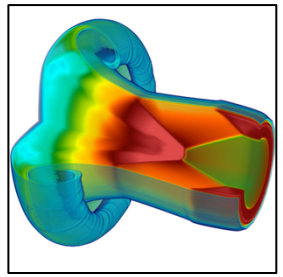

JUSPIC


XNS


NEST


UG4


MP2C

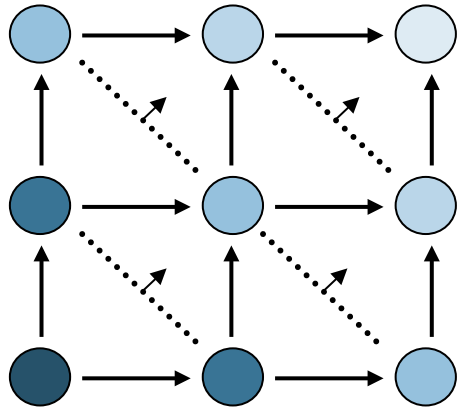

BLAST

Sweep3D – Neutron transport simulation

LogGP model for communication developed by Hoisie et al.

$$t^{comm} = [2(p_x + p_y - 2) + 4(n_{sweep} - 1)] \cdot t_{msg}$$

$$t^{comm} = c \cdot \sqrt{p}$$



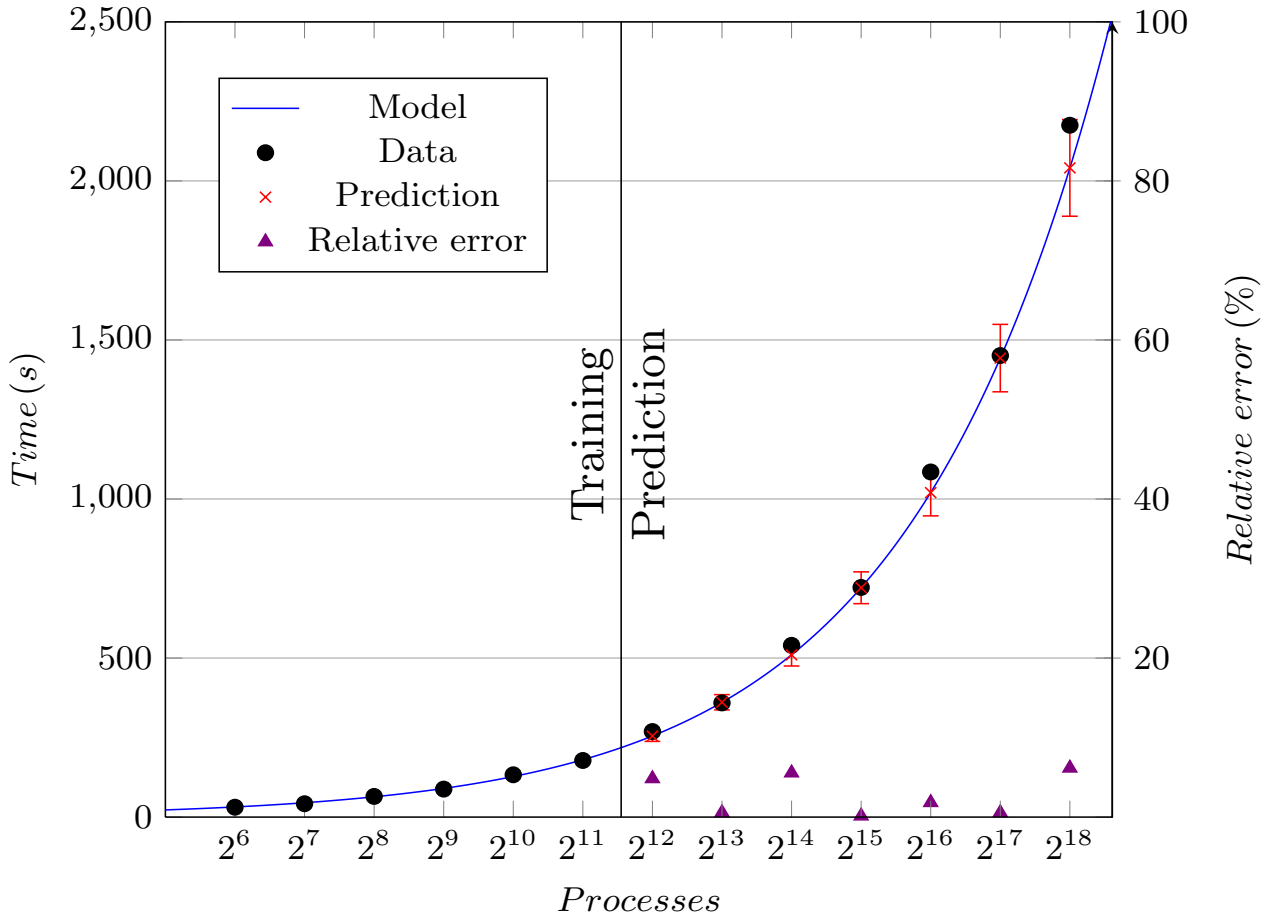
Kernel [2 of 40]	Model [s] t = f(p)	Predictive error [%] p_t=262k
sweep → MPI_Recv	$4.03\sqrt{p}$	5.10
sweep	582.19	0.01

$p_i \leq 8k$

#bytes ~ const.
#msg ~ const.

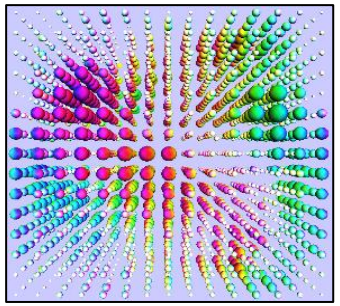
Sweep3D – Neutron transport simulation

Model: $4.03\sqrt{p}$



Milc

MILC/su3_rmd – from MILC suite of QCD codes
with performance model manually created by Hoefler et al.



- Time per process should remain constant except for a rather small logarithmic term caused by global convergence checks

Kernel [3 of 479]	Model [s] t=f(p)	Predictive Error [%] p_t=64k
compute_gen_staple_field	$2.40 \cdot 10^{-2}$	0.43
g_vecdoublesum → MPI_Allreduce	$6.30 \cdot 10^{-6} \cdot \log_2^2(p)$	0.01
mult_adj_su3_fieldlink_lathwec	$3.80 \cdot 10^{-3}$	0.04

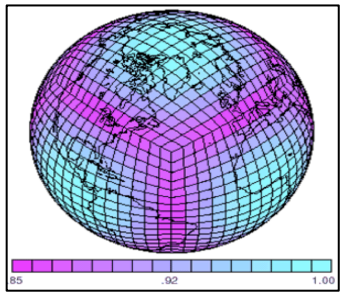
$p_i \leq 16k$

HOMME – Climate

NCAR NATIONAL CENTER FOR ATMOSPHERIC RESEARCH

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	$0.026 + 2.53 \cdot 10^{-6} p \cdot \sqrt{p} + 1.24 \cdot 10^{-12} p^3$	57.02
vlaplace_sphere_vk	49.53	99.32
compute_and_apply_rhs	48.68	1.65

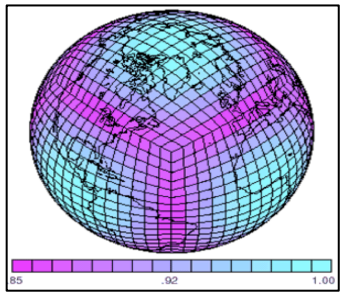
$$p_i \leq 15\text{k}$$

HOMME – Climate (2)

NCAR NATIONAL CENTER FOR ATMOSPHERIC RESEARCH

Core of the Community Atmospheric Model (CAM)

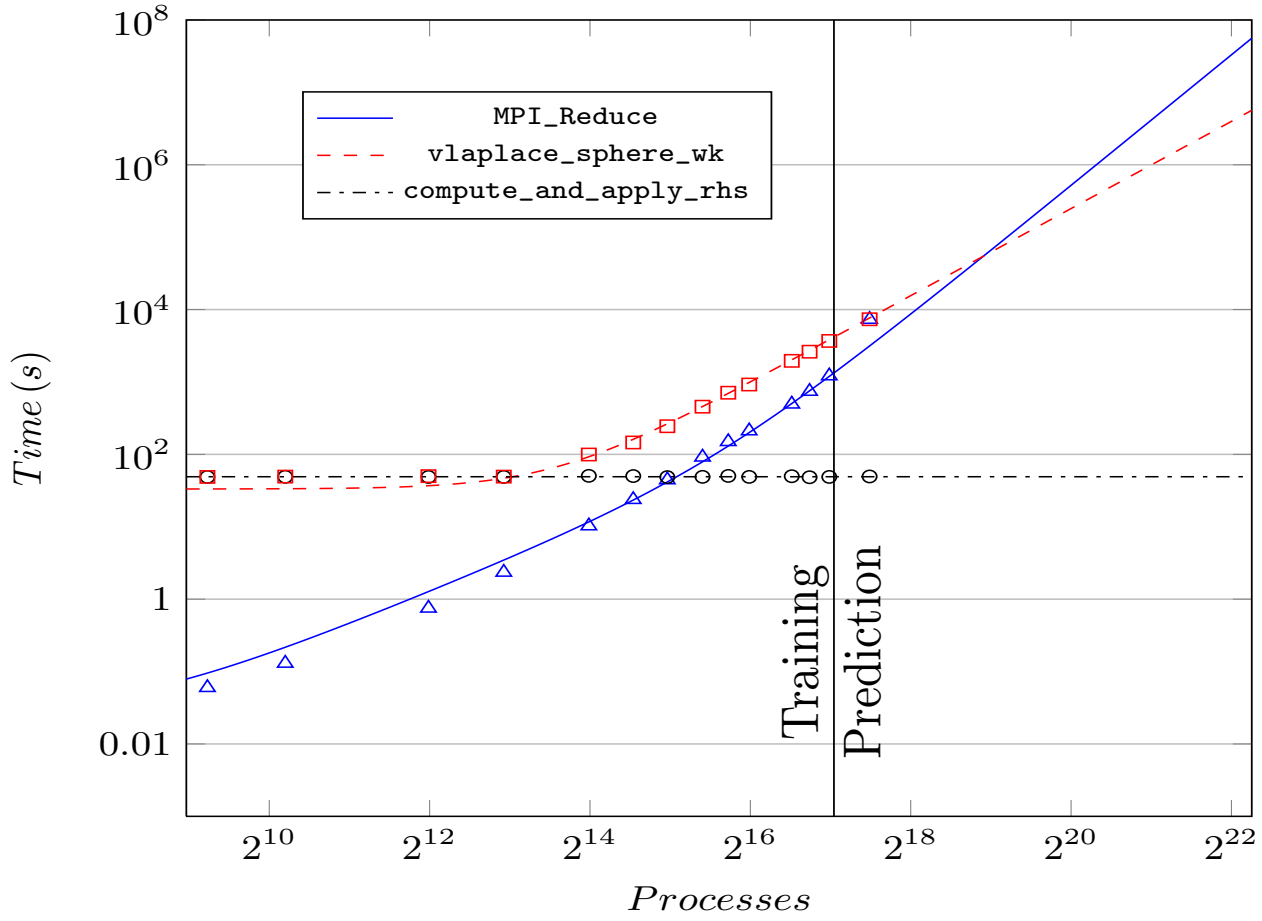
- Spectral element dynamical core on a cubed sphere grid



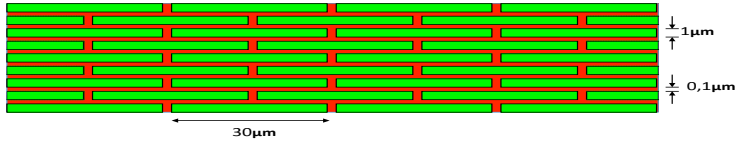
Kernel [3 of 194]	Model [s] t = f(p)	Predictive error [%] p_t = 130k
box_rearrange → MPI_Reduce	$3.63 \cdot 10^{-6} p \cdot \sqrt{p} + 7.21 \cdot 10^{-13} p^3$	30.34
vlaplace_sphere_vk	$24.44 + 2.26 \cdot 10^{-7} p^2$	4.28
compute_and_apply_rhs	49.09	0.83

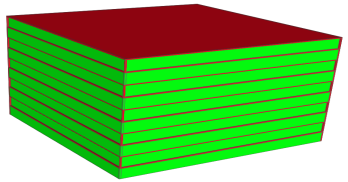
$$p_i \leq 43\text{k}$$

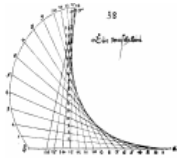
HOMME – Climate (3)



UG4







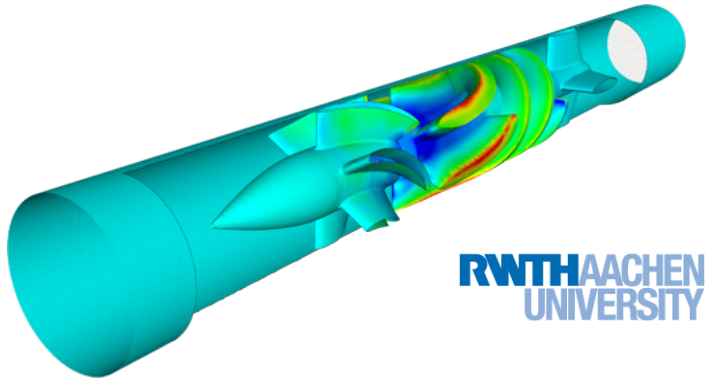
G-CSC

- Numerical framework for grid-based solution of partial differential equations (~500,000 lines of C++ code, 2,000 kernels)
  - Application: drug diffusion through the human skin
- In general, all kernels scale well
  - Multigrid solver kernel (MGM) scales logarithmically
  - Number of iterations needed by the unpreconditioned conjugate gradient (CG) method depends on the mesh size
    - Increases by factor of two with each refinement
    - Will therefore suffer from iteration count increase in weak scaling

Kernel	Model (time [s])
CG	$0.227 + 0.31 * p^{0.5}$
MGM	$0.219 + 0.0006 * \log^2(p)$

XNS

- Finite element flow simulation
- Strong scaling analysis using accumulated time across processes as metric



Kernel	Runtime [%] p=128	Runtime [%] p=4096	Model [s] t = f(p)
ewdgennprm->MPI_Recv	0.46	51.46	$0.029 \cdot p^2$
ewddot	44.78	5.04	$37406.80 \quad 13.29 \cdot \sqrt{p} \cdot \log(p)$

#bytes = ~p
#msg = ~p

MPI

Platform	Juqueen	Juropa	Piz Daint
Barrier [s]		**Expectation: *O* (log *p*)**	
Model	O (log p)	O ($p^{0.67}$ log p)	O ($p^{0.33}$)
R^2	0.99	0.99	0.99
Divergence	O (1)	O ($p^{0.67}$)	O ($p^{0.33}$/log p)
Match	✓	✗	~
Bcast [s]		**Expectation: *O* (log *p*)**	
Model	O (log p)	O ($p^{0.5}$)	O ($p^{0.5}$)
R^2	0.86	0.98	0.94
Divergence	O (1)	O ($p^{0.5}$/log p)	O ($p^{0.5}$/log p)
Match	✓	~	~
Reduce [s]		**Expectation: *O* (log *p*)**	
Model	O (log p)	O ($p^{0.5}$ log p)	O ($p^{0.5}$ log p)
R^2	0.93	0.99	0.94
Divergence	O (1)	O ($p^{0.5}$)	O ($p^{0.5}$)
Match	✓	~	~

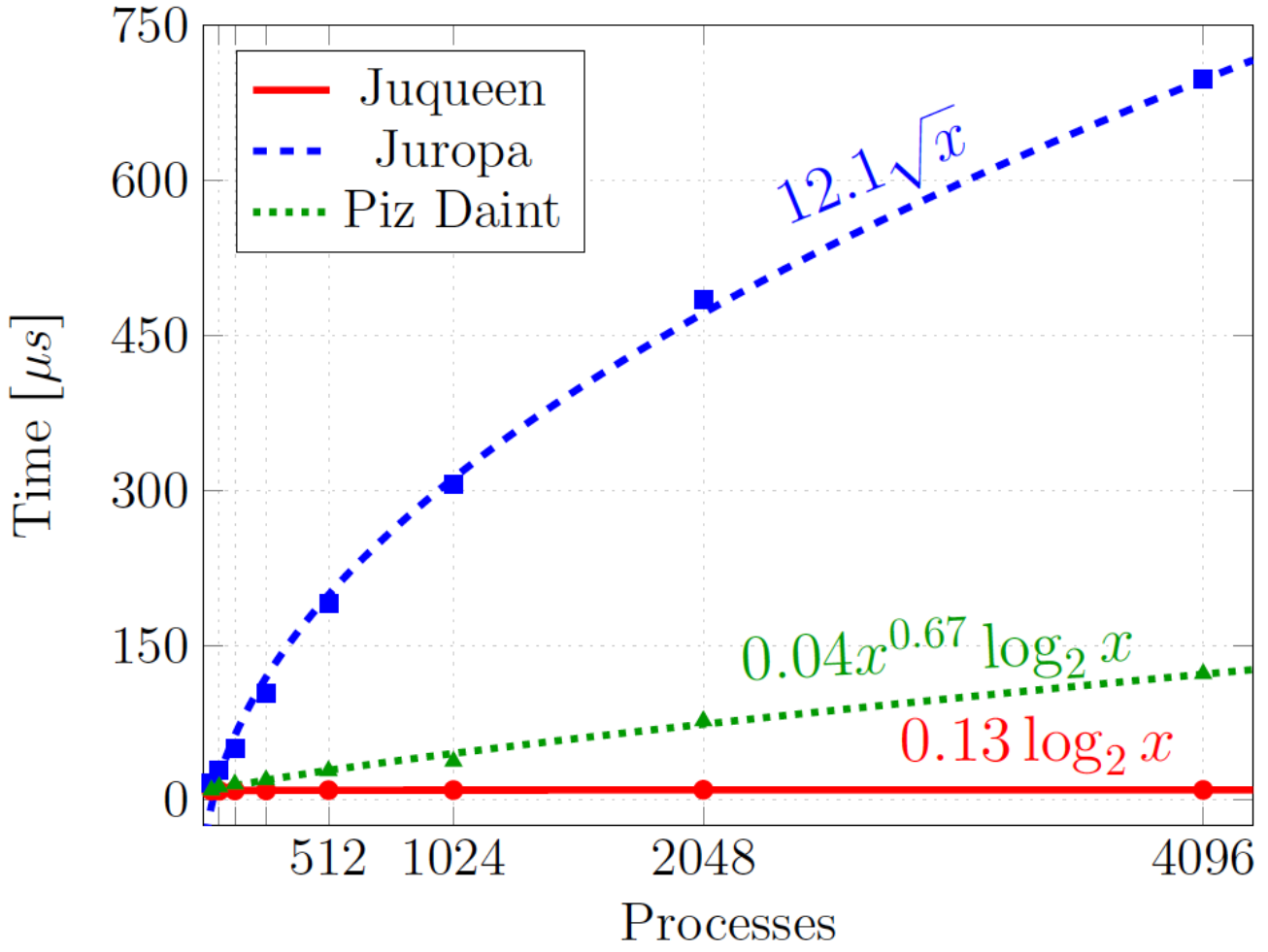
Platform	Juqueen	Juropa	Piz Daint
MPI memory [MB]		**Expectation: *O* (log *p*)**	
Model	O (log p)	O (p)	O (log p)
R^2	0.72	1	0.23
Divergence	O (1)	O (p / log p)	O (1)
Match	✓	✗	✓
Comm_create [B]		**Expectation: *O* (*p*)**	
Model	O (p)	O (p)	O (p)
R^2	1	1	0.99
Divergence	O (1)	O (1)	O (1)
Match	✓	✓	✓
Win_create [B]		**Expectation: *O* (*p*)**	
Model	O (p)	O (p)	O (p)
R^2	1	1	0.99
Divergence	O (1)	O (1)	O (1)
Match	✓	✓	✓

MPI (2)

Platform	Juqueen	Juropa	Piz Daint
Allreduce [s]		**Expectation: *O* (log *p*)**	
Model	*O* (log *p*)	*O* ($p^{0.5}$)	*O* ($p^{0.67}$ log *p*)
R^2	0.87	0.99	0.99
Divergence	0	*O* ($p^{0.5}$/log *p*)	*O* ($p^{0.67}$)
Match	✓	~	✗!
Comm_dup [B]		**Expectation: *O* (1)**	
Model	2.2e5	256	3770 + 18*p*
R^2	1	1	0.99
Divergence	*O* (1)	*O* (1)	*O* (*p*)
Match	✓	✓	✗

MPI (3)

- MPI_Allreduce
- 3 different machines – 3 different models



Mass-producing performance models

- Is feasible
- Offers insight
- Requires low effort
- Improves code coverage

References

[1] Alexandru Calotoiu, Torsten Hoefler, Marius Poke, Felix Wolf: Using Automated Performance Modeling to Find Scalability Bugs in Complex Codes. In Proc. of the ACM/IEEE Conference on Supercomputing (SC13), Denver, CO, USA, pages 1-12, ACM, November 2013.

[2] Sergei Shudler, Alexandru Calotoiu, Torsten Hoefler, Alexandre Strube, Felix Wolf: Exascaling Your Library: Will Your Implementation Meet Your Expectations?. In *Proc. of the International Conference on Supercomputing (ICS), Newport Beach, CA, USA, pages 1-11, ACM, June 2015*

[3] Andreas Vogel, Alexandru Calotoiu, Alexandre Strube, Sebastian Reiter, Arne Nägel, Felix Wolf, Gabriel Wittum: 10,000 Performance Models per Minute - Scalability of the UG4 Simulation Framework. In Proc. of the 21st Euro-Par Conference, Vienna, Austria of Lecture Notes in Computer Science, pages 519–531, Springer, August 2015.

[4] Christian Iwainsky, Sergei Shudler, Alexandru Calotoiu, Alexandre Strube, Michael Knobloch, Christian Bischof, Felix Wolf: How Many Threads will be too Many? On the Scalability of OpenMP Implementations. In Proc. of the 21st Euro-Par Conference, Vienna, Austria of Lecture Notes in Computer Science, pages 451–463, Springer, August 2015.

Thank You!

Get Extra-P at: http://www.scalasca.org/software/extra-p/download.html