TORSTEN HOEFLER

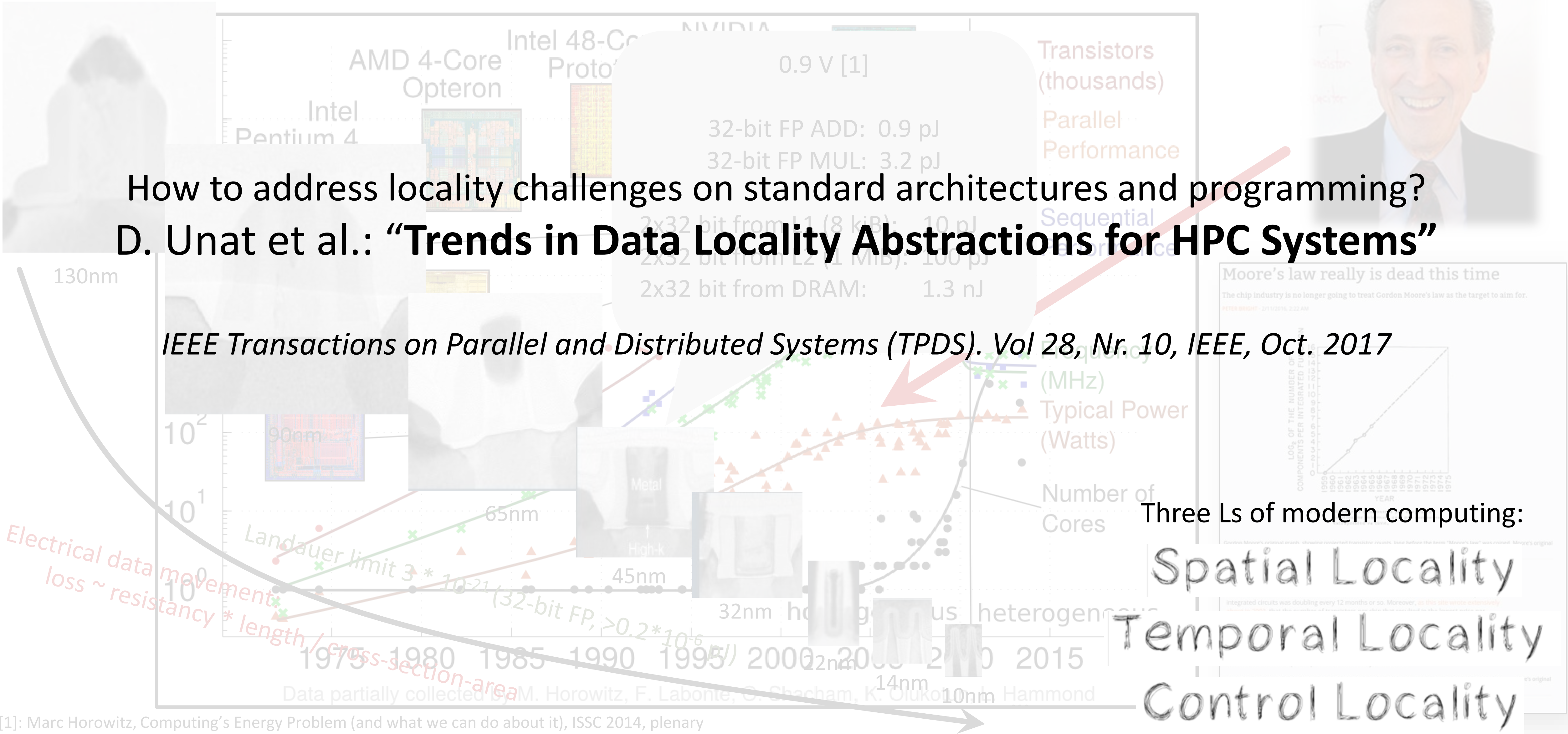
Performance Modeling for Future Computing Technologies

Presentation at Tsinghua University, Beijing, China

Part of the 60th anniversary celebration of the Computer Science Department (#9)

SPCL

Changing hardware constraints and the physics of computing



How to address locality challenges on standard architectures and programming?

D. Unat et al.: “**Trends in Data Locality Abstractions for HPC Systems**”

IEEE Transactions on Parallel and Distributed Systems (TPDS). Vol 28, Nr. 10, IEEE, Oct. 2017

Three Ls of modern computing:

Spatial Locality

Temporal Locality

Control Locality

[1]: Marc Horowitz, Computing’s Energy Problem (and what we can do about it), ISSC 2014, plenary

[2]: Moore: Landauer Limit Demonstrated, IEEE Spectrum 2012

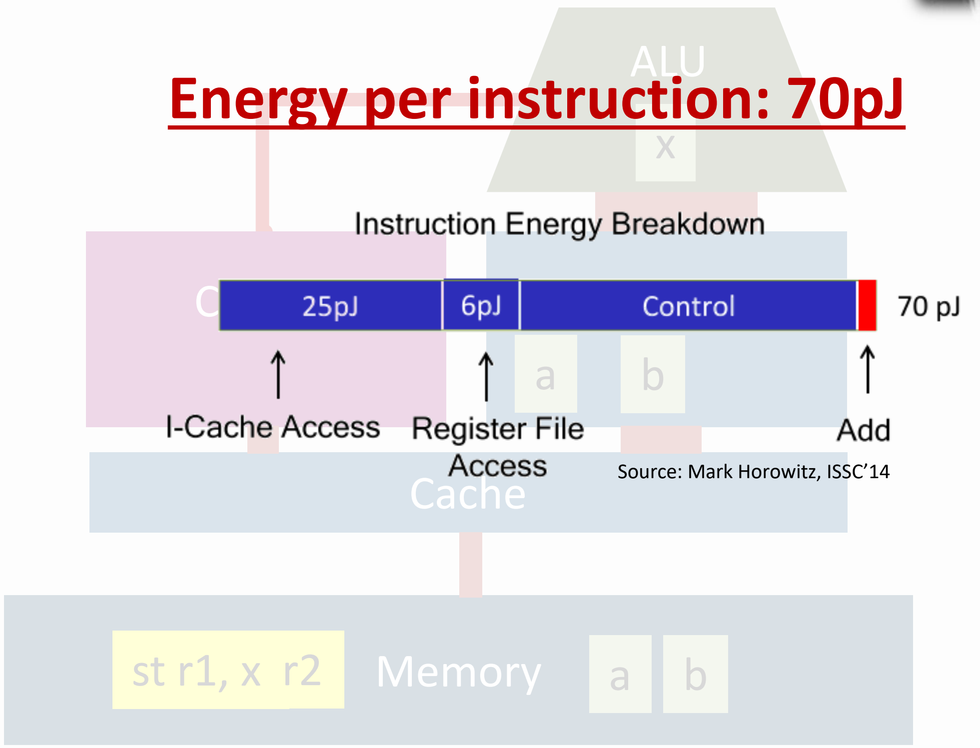
Control Locality? Load-store vs. Datafl

Turing Award 1977 (Backus): "Surely there must be a less primitive way of making big changes in the store than pushing vast numbers of words back and forth through the von Neumann bottleneck."

Load-store ("von Neumann")

x=a+b

Energy per instruction: 70pJ

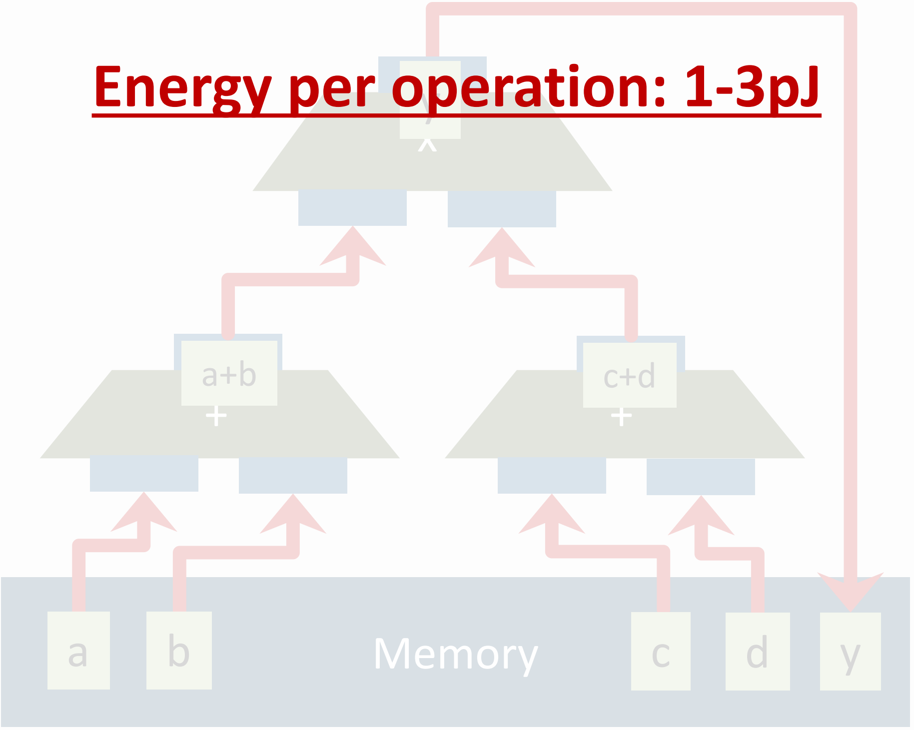


Source: Mark Horowitz, ISSC'14

Static Dataflow ("non von Neumann")

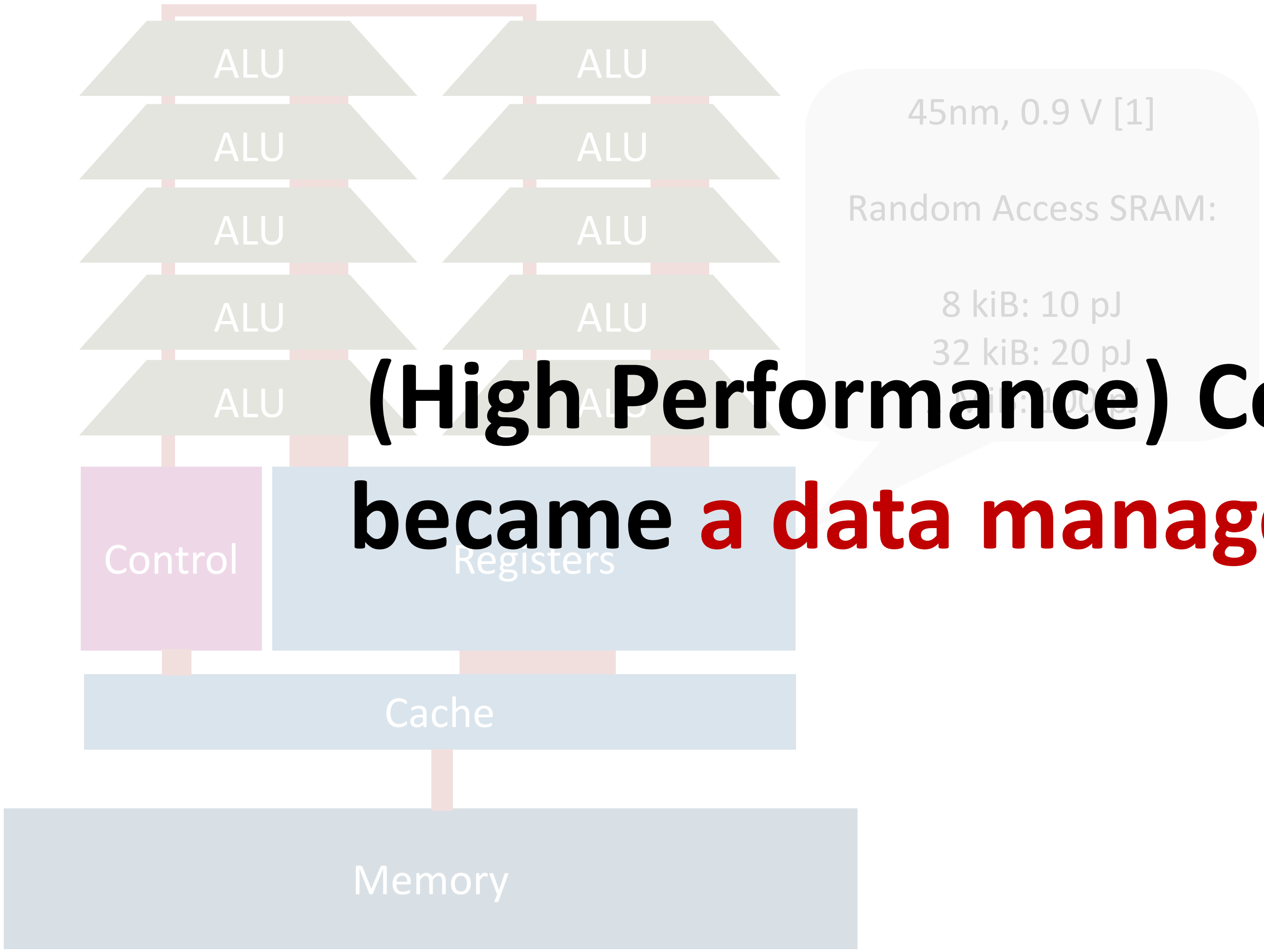
y=(a+b)*(c+d)

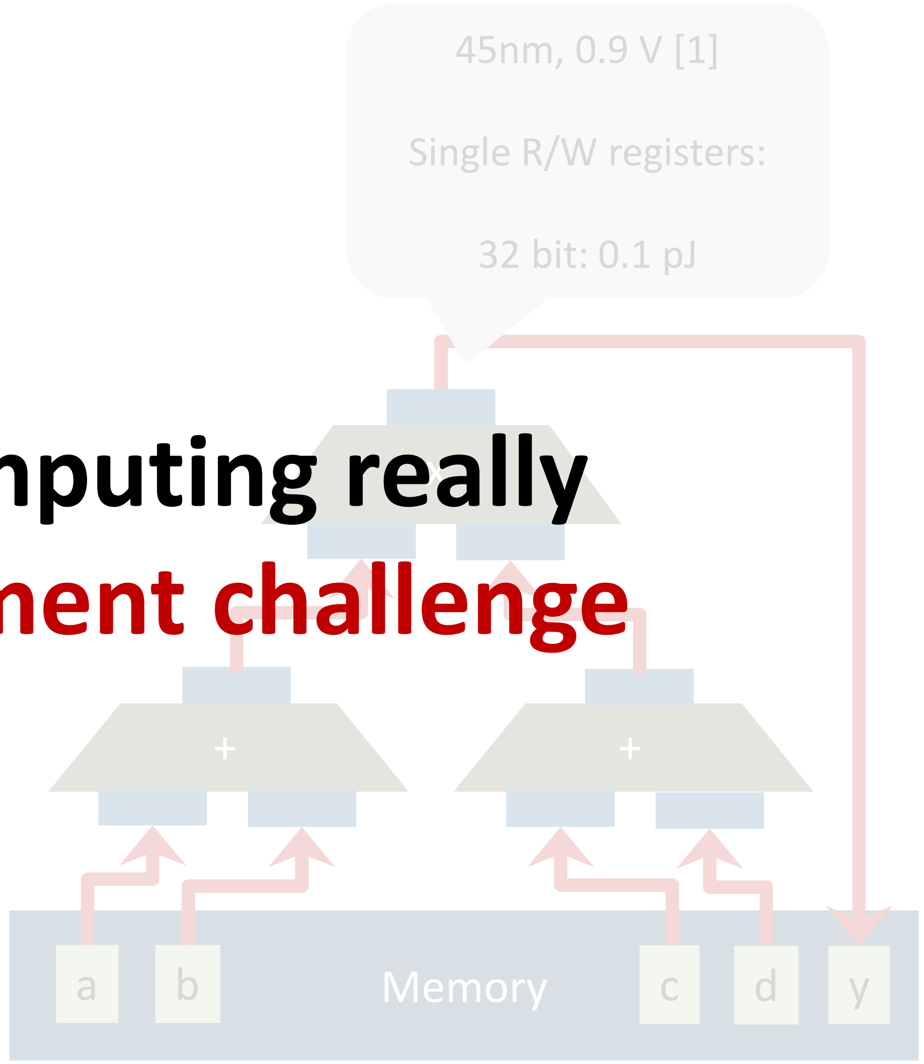
Energy per operation: 1-3pJ



Control Locality

Single Instruction Multiple Data/Threads (SIMD - Vector CPU, SIMT - GPU)





(High Performance) Computing really became a data management challenge

[1]: Marc Horowitz, Computing's Energy Problem (and what we can do about it), ISSC 2014, plenary

Crystal Ball into the Post-Moore Future (maybe already today?)

Future architectures will force us to manage accelerated heterogeneity

Rest of this talk: how do we understand which parts of programs to accelerate on which device?

Obvious answer: the slow ones!
So simply observe their performance? Not so fast.

Neuromorphic	Quantum	Efficient CMOS

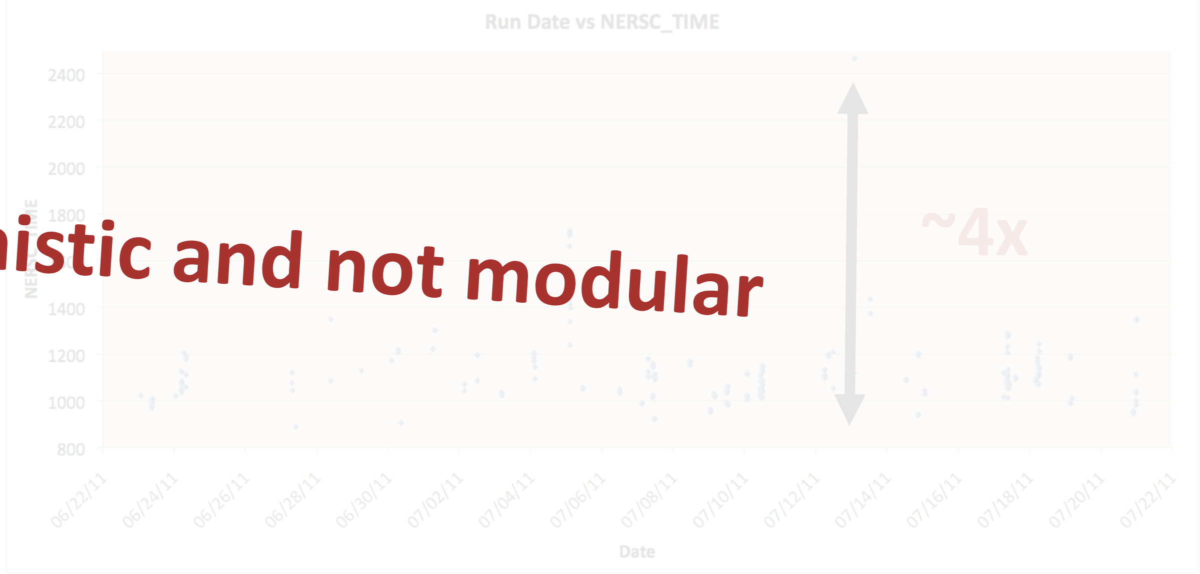
Neuromorphic
- Asynchronous CMOS circuits
  - 1000x energy benefit
- Integrates compute (neurons) and memo…
- Very specialized
  - …
  - Phrase your problem as inference!
- Even learning is hard
  - Comparatively little work
  - Suddenly much lower ene… benefits …

Quantum
- Completely different paradigm
  - Concept of qubits
  - Bases on quantum mechanics
- Many different ideas how to build
  - Optical
  - Spin-based
  - Superconducting
  - Majorana qubits
- Needs new algorithms to be useful
  - Algorithms are limited

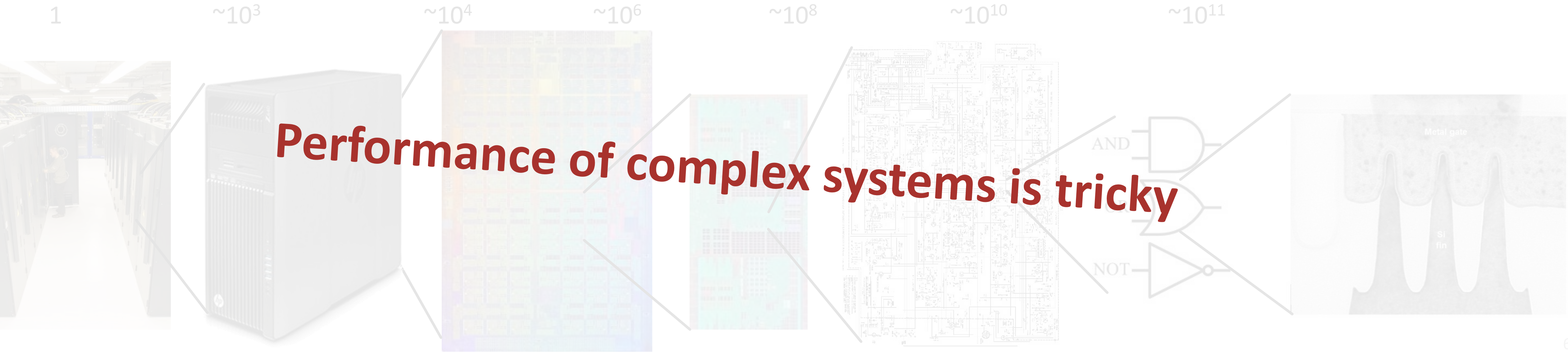
Efficient CMOS
- FPGAs or CGRAs or GPUs
  - Have been around for a while
- Use transistors more efficiently
  - Accelerators
  - Custom architectures
  - Reconfigurable datapaths
- …
  - Dataflow + Control Flow
- Main challenge
  - Programmability!
  - See our SC18 tutorial *"Productive Parallel Programming for FPGA*

image source: 21stcentury.com
image source: intel.com
image source: intel.com

What can we learn from High Performance Computing

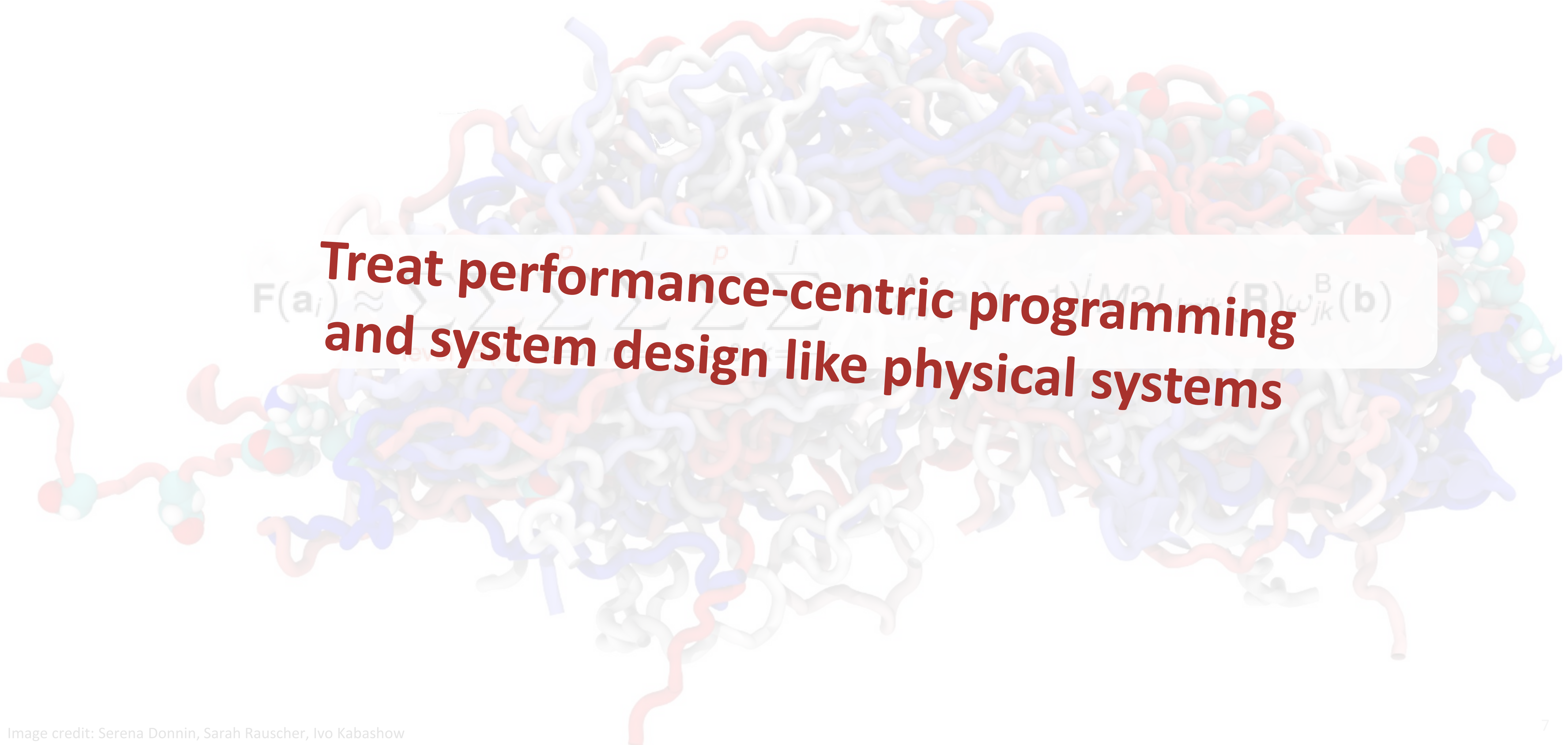


Performance is nondeterministic and not modular



Performance of complex systems is tricky

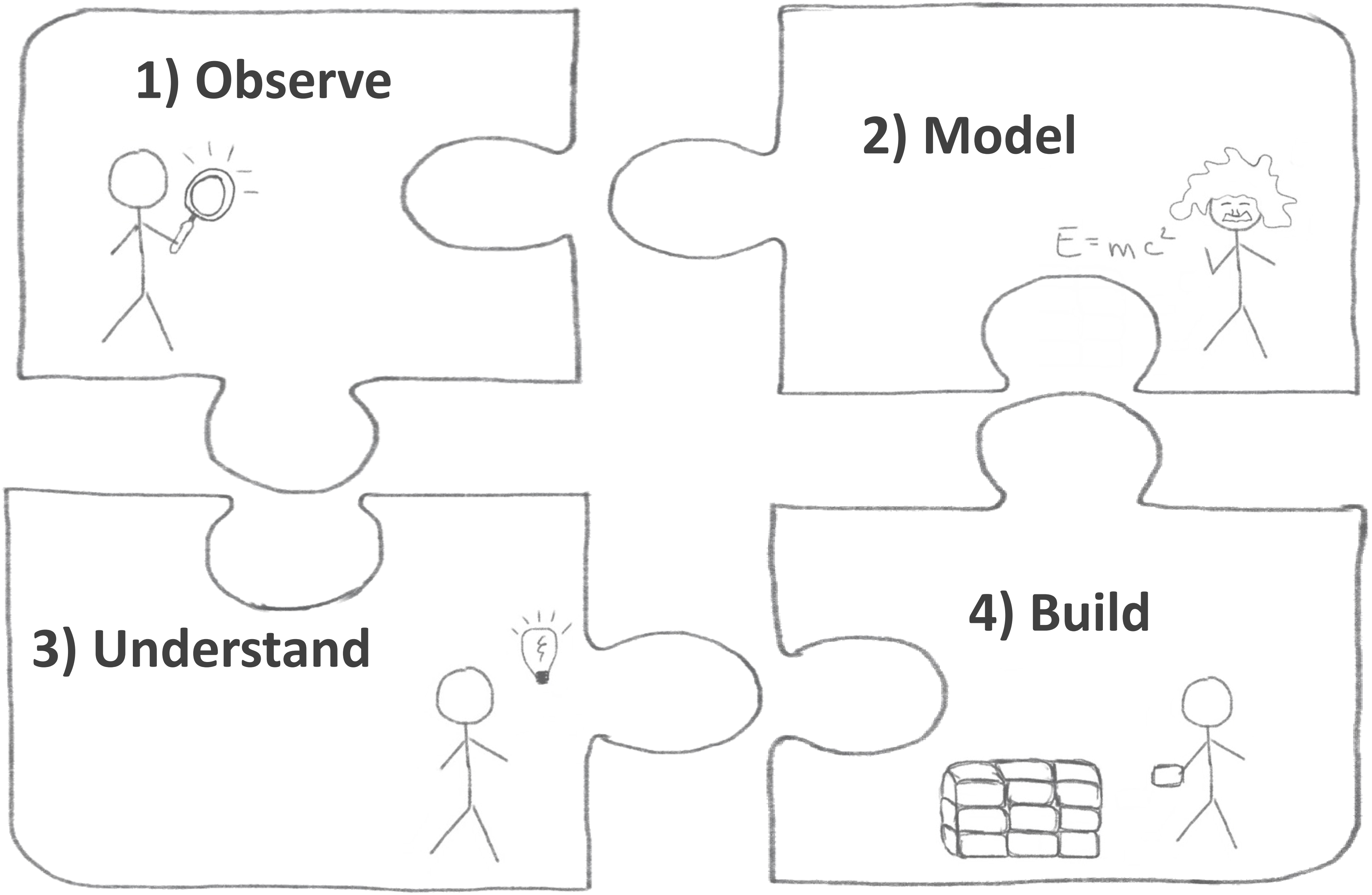
HPC is used to solve complex problems!



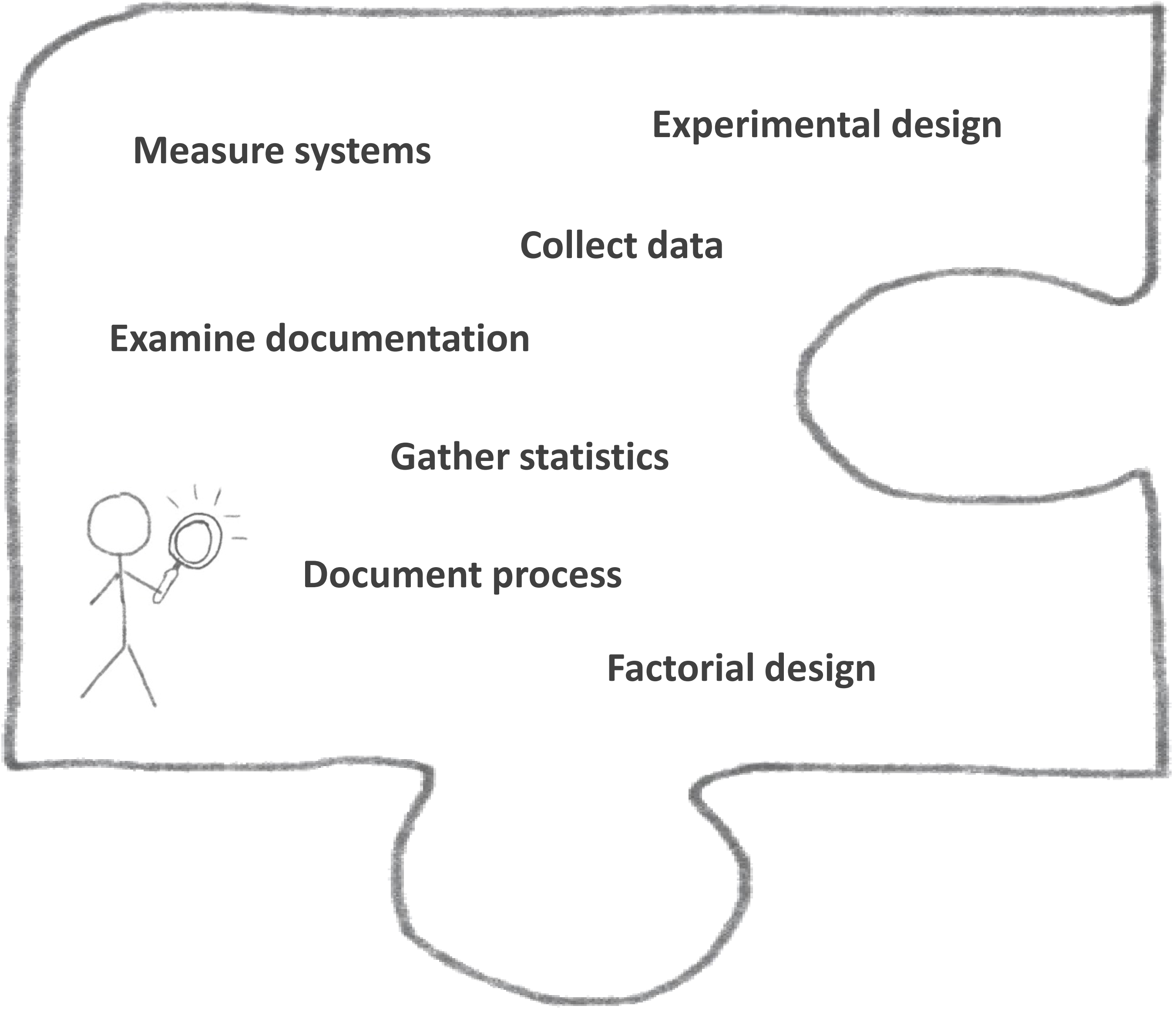
Treat performance-centric programming and system design like physical systems

Image credit: Serena Donnin, Sarah Rauscher, Ivo Kabashow

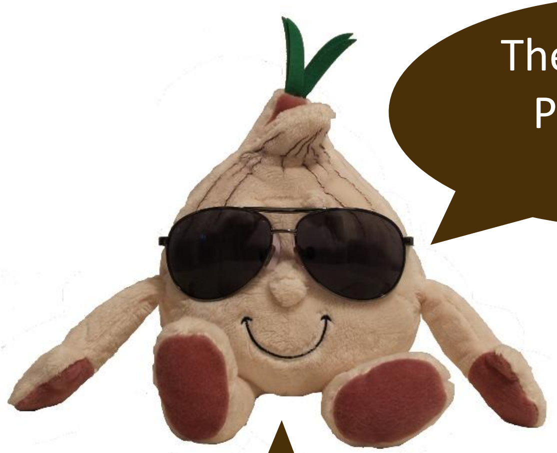
Scientific Performance Engineering



Part I: Observe

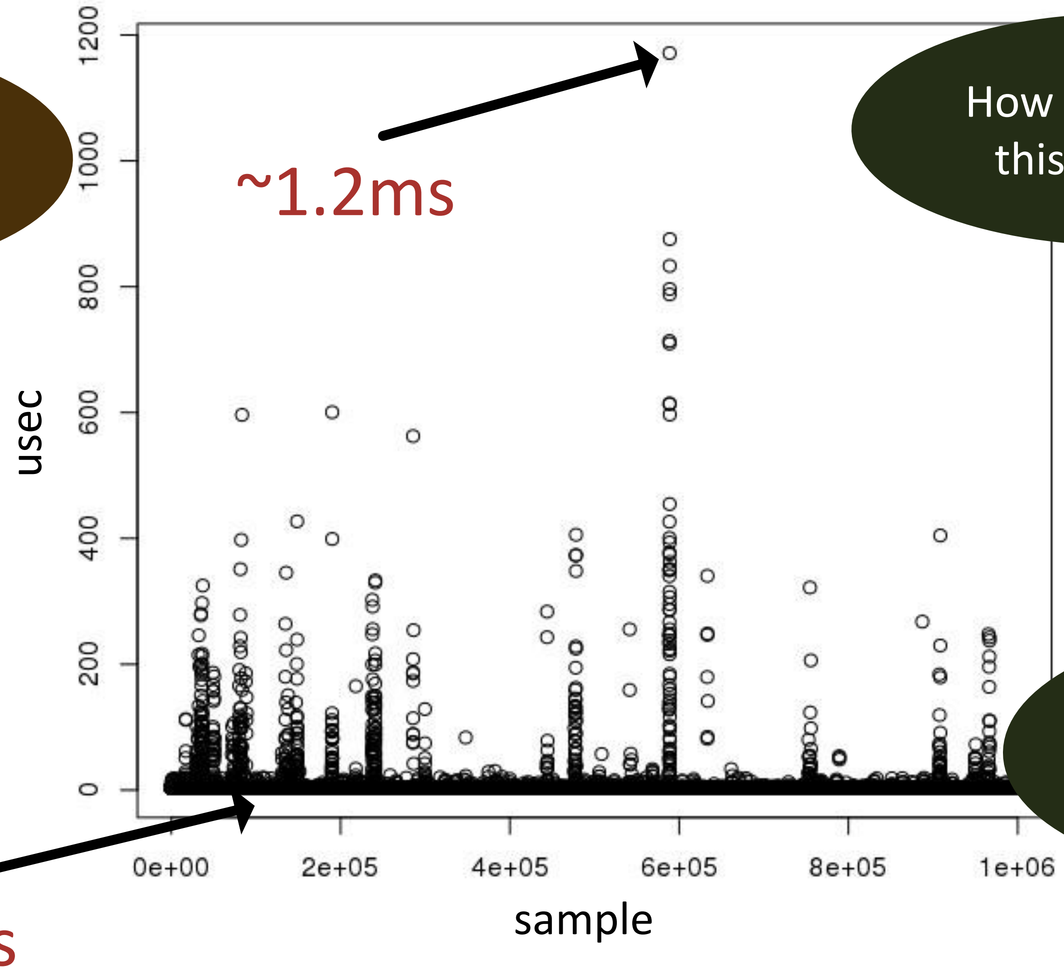


Trivial Example: Simple ping-pong latency benchmark



The latency of Piz Daint is 1.77us!

I averaged 10^6 experiments, it must be right!



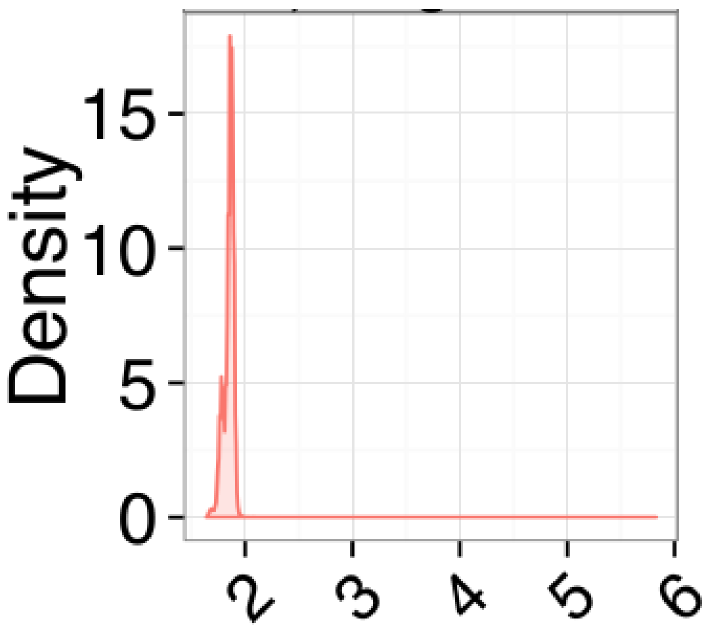
How did you get this number?

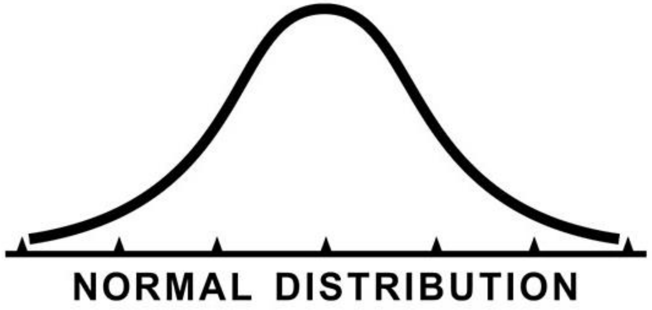
Why do you think so? Can I see the data?

Dealing with variation

The 99.9% confidence interval is 1.765us to 1.775us

Ugs, the data is not normal at all. The **nonparametric** 99.9% CI is much wider: 1.6us to 1.9us!



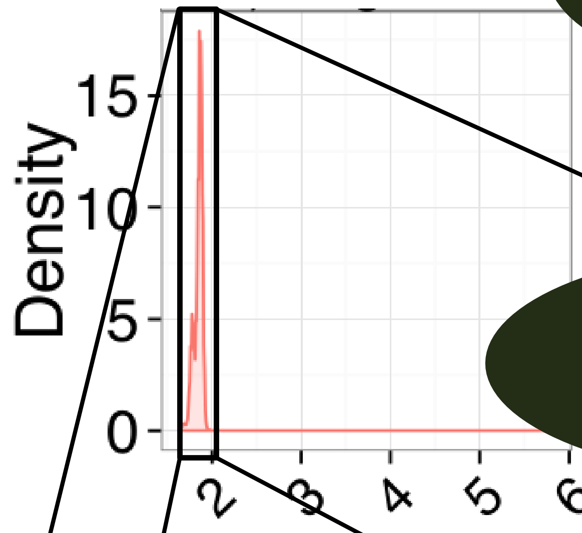


Did you assume normality?

Can we test for normality?

Looking at the data in detail

This CI makes me nervous. Let's zoom!



Clearly, the mean/median are not sufficient!

Try quantile regression!

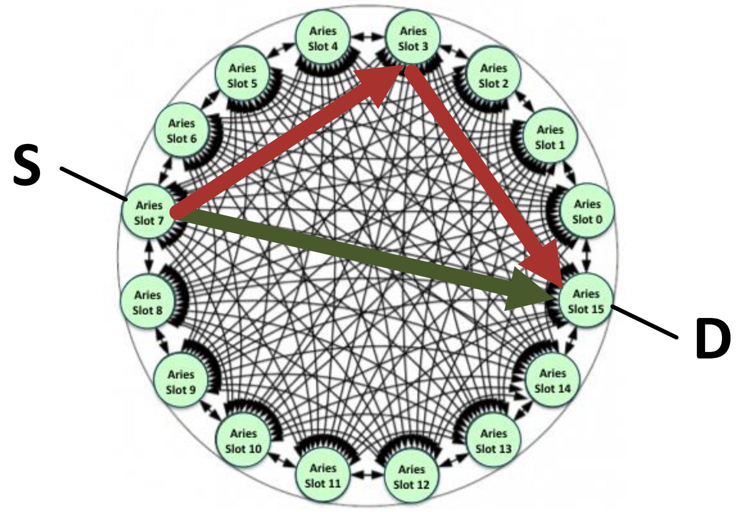
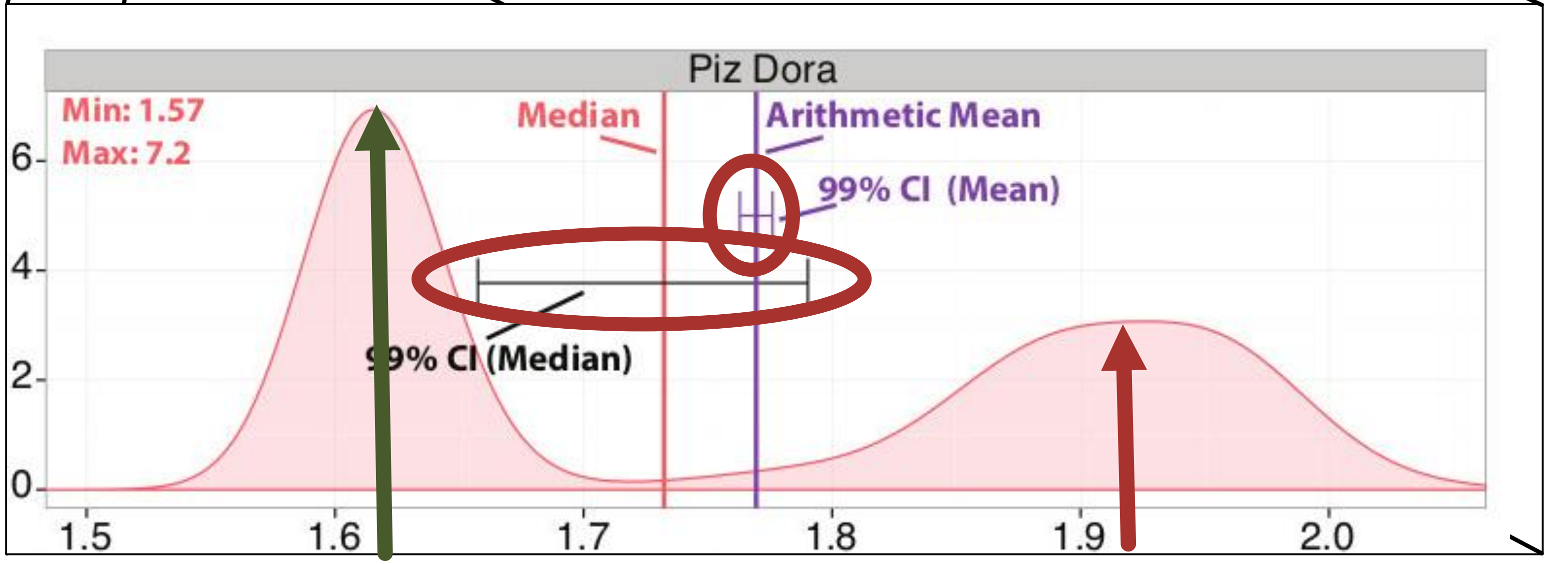


Image credit: nersc.gov



Scientific benchmarking of parallel computing systems

ACM/IEEE Supercomputing 2015 (SC15) + talk online on youtube!

Scientific Benchmarking of Parallel Computing Systems

Twelve ways to tell the masses when reporting performance results

Torsten Hoefler
Dept. of Computer Science
ETH Zurich
Zurich, Switzerland
htor@inf.ethz.ch

Roberto Belli
Dept. of Computer Science
ETH Zurich
Zurich, Switzerland
bellir@inf.ethz.ch

ABSTRACT

Measuring and reporting performance of parallel computers constitutes the basis for scientific advancement of high-performance computing (HPC). Most scientific reports show performance improvements of new techniques and are thus obliged to ensure reproducibility or at least interpretability. Our investigation of a stratified sample of 120 papers across three top conferences in the field shows that the state of the practice is lacking. For example, it is often unclear if reported improvements are deterministic or observed by chance. In addition to distilling best practices from existing work, we propose statistically sound analysis and reporting techniques and simple guidelines for experimental design in parallel computing and codify them in a portable benchmarking library. We

Reproducing experiments is one of the main principles of the scientific method. It is well known that the performance of a computer program depends on the application, the input, the compiler, the runtime environment, the machine, and the measurement methodology [20,43]. If a single one of these aspects of *experimental design* is not appropriately motivated and described, presented results can hardly be reproduced and may even be misleading or incorrect.

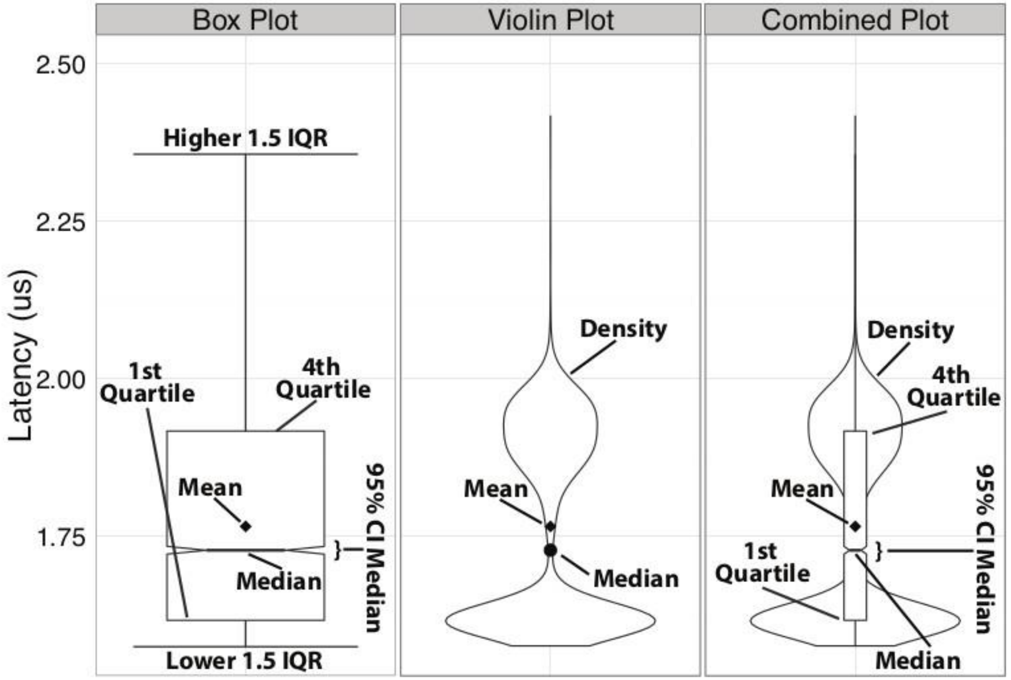
The complexity and uniqueness of many supercomputers makes reproducibility a hard task. For example, it is practically impossible to recreate most hero-runs that utilize the world's largest machines because these machines are often unique and their software configurations changes regularly. We introduce the notion of *interpretability*, which is weaker than reproducibility. *We call an ex-*

pret the
es if they

Simplifying Measuring and Reporting: LibSciBench

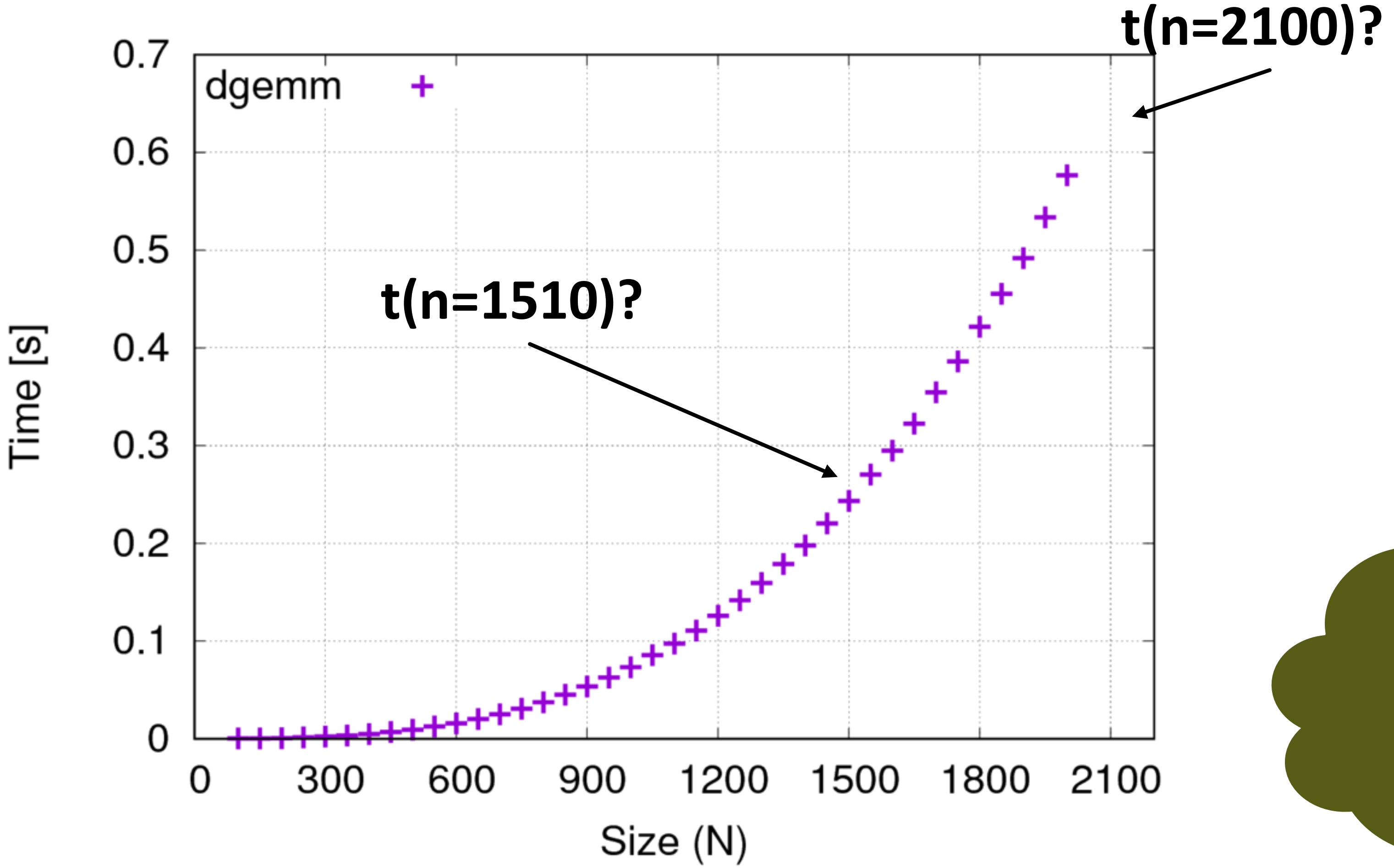
```
#include <mpi.h>
#include <liblsb.h>
#include <stdlib.h>

#define N 1024
#define RUNS 10

int main(int argc, char *argv[]){
    int i, j, rank, buffer[N];

    MPI_Init(&argc, &argv);
    LSB_Init("test_bcast", 0);

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    /* Output the info (i.e., rank, runs) in the results file */
    LSB_Set_Rparam_int("rank", rank);
    LSB_Set_Rparam_int("runs", RUNS);

    for (sz=1; sz<=N; sz*=2){
      for (j=0; j<RUNS; j++){
        /* Reset the counters */
        LSB_Res();

        /* Perform the operation */
        MPI_Bcast(buffer, sz, MPI_INT, 0, MPI_COMM_WORLD);

        /* Register the j-th measurement of size sz */
        LSB_Rec(sz);
      }
    }

    LSB_Finalize();
    MPI_Finalize();
    return 0;
}
```

- Simple MPI-like C/C+ interface
- High-resolution timers
- Flexible data collection
- Controlled by environment variables
- Tested up to 512k ranks
- Parallel timer synchronization
- R scripts for data analysis and visualization



S. Di Girolamo, TH: http://spcl.inf.ethz.ch/Research/Performance/LibLSB/

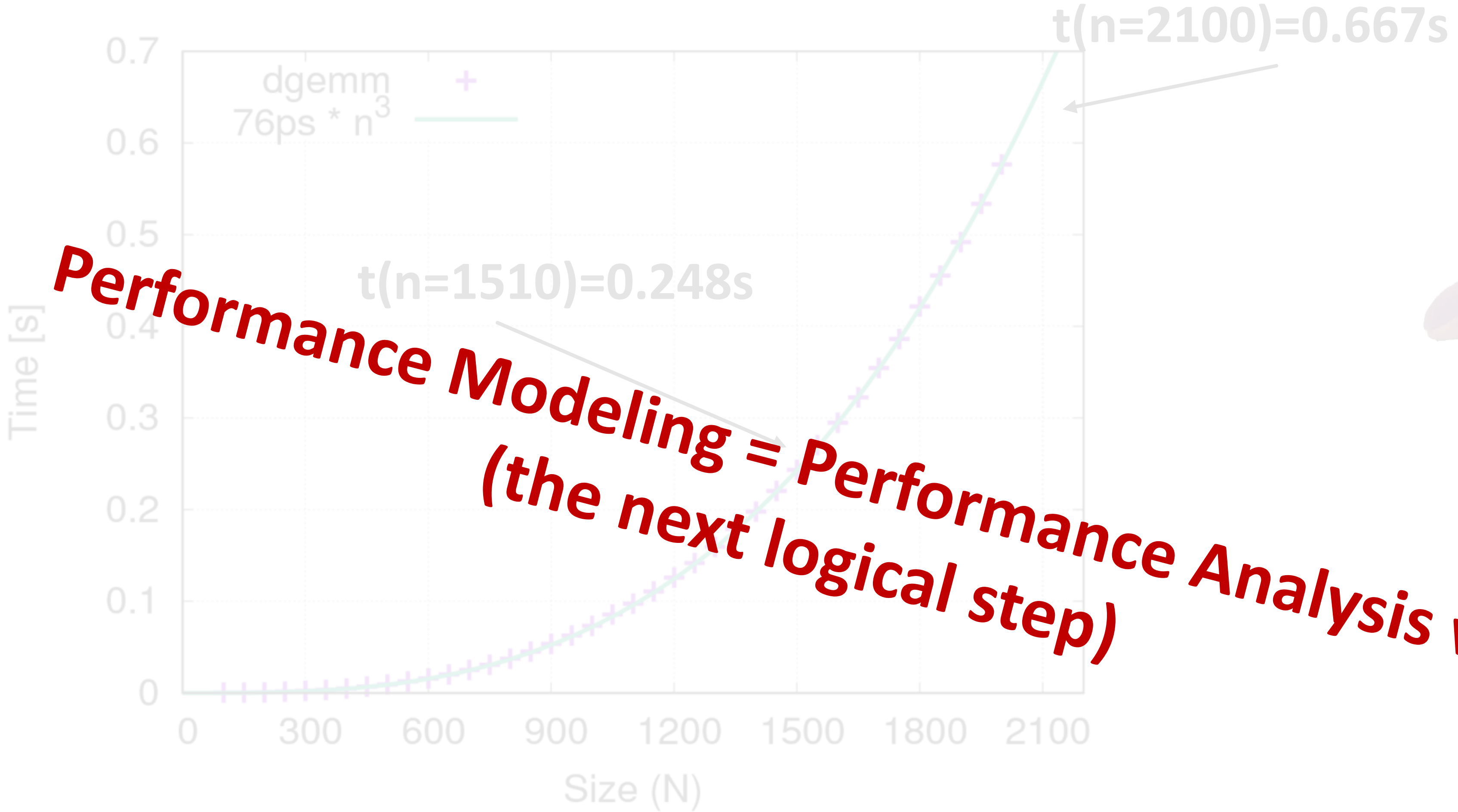
We have the (statistically sound) data, now what?



The 99% confidence interval is within 1% of the reported median.

Matrix Multiply
$t(n) = a*n^3$

We have the (statistically sound) data, now what?

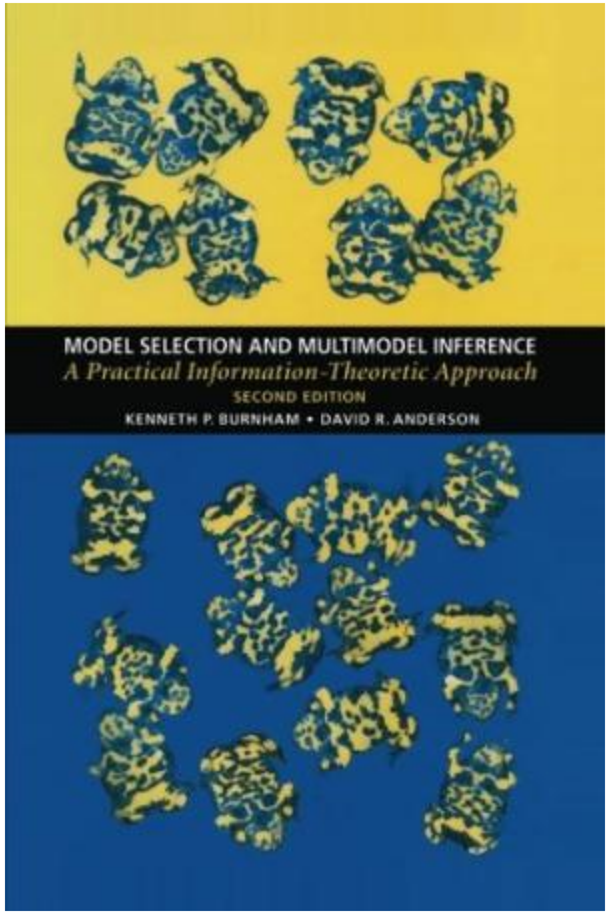


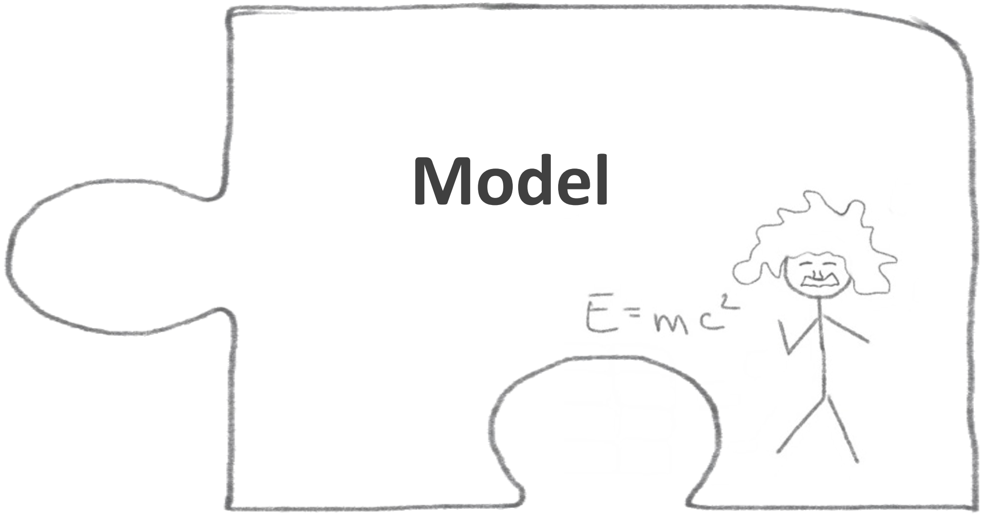
Performance Modeling = Performance Analysis v 2.0
(the next logical step)

The 99% confidence interval is within 1% of the reported median.
The adjusted R^2 of the model fit is 0.99

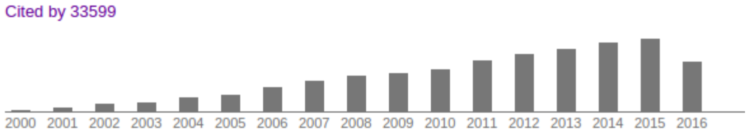
TH, W. Gropp, M. Snir, W. Kramer: Performance Modeling for Systematic Performance Tuning, IEEE/ACM SC11

Part II: Model



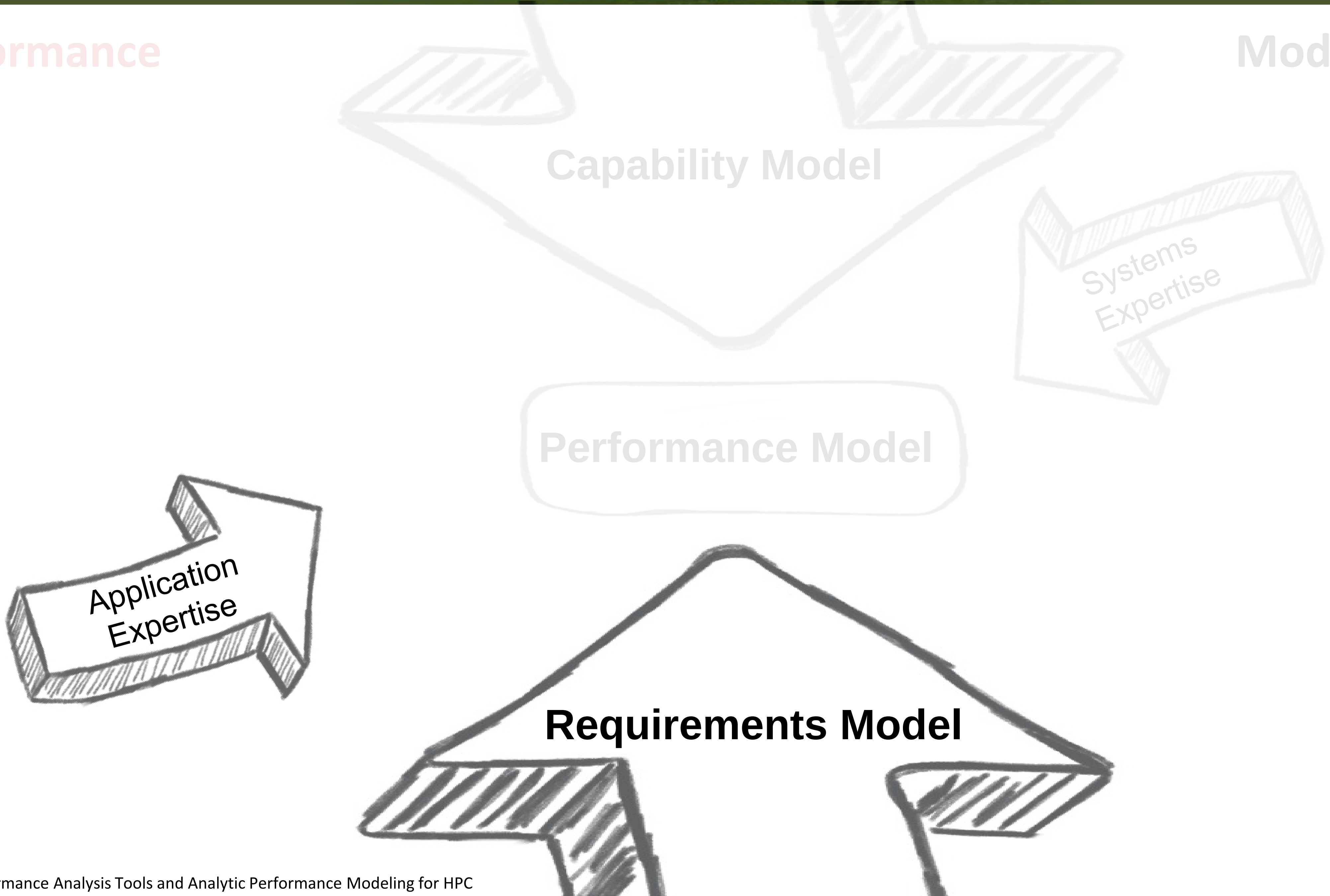


Burnham, Anderson: *"A model is a simplification or approximation of reality and hence will not reflect all of reality. ... Box noted that "all models are wrong, but some are useful." While a model can never be "truth," a model might be ranked from very useful, to useful, to somewhat useful to, finally, essentially useless."*



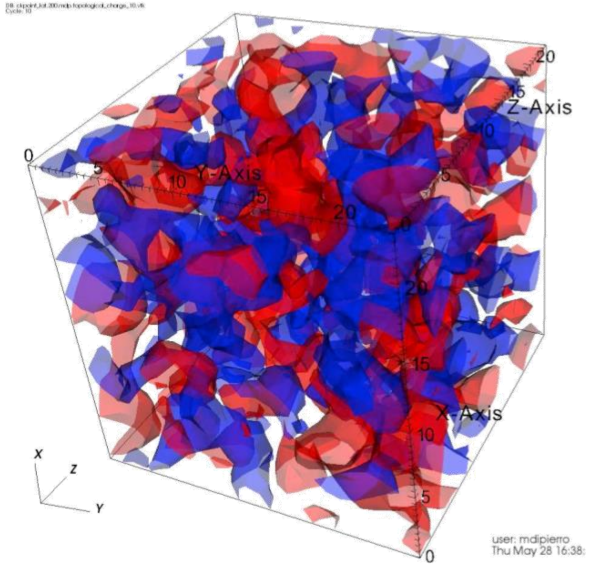
This is generally true for all kinds of modeling.
We focus on **performance modeling** in the following!

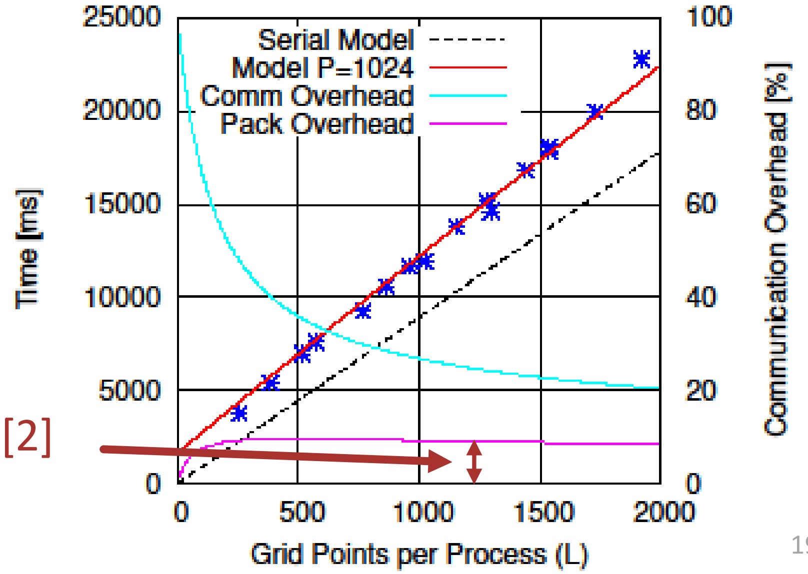
Performance Modeling



Requirements modeling I: Six-step performance modeling

- Input parameters
- Describe application kernels
- Communication pattern
- Communication / computation overlap
- Fit sequential baseline
- Communication parameters





10-20% speedup [2]

[1] TH, W. Gropp, M. Snir and W. Kramer: Performance Modeling for Systematic Performance Tuning, SC11
[2] TH and S. Gottlieb: Parallel Zero-Copy Algorithms for Fast Fourier Transform and Conjugate Gradient using MPI Datatypes, EuroMPI'10

Requirements modeling II: Automated best-fit modeling

- **Manual kernel selection and hypothesis generation is time consuming (boring and tricky)**
- **Idea: Automatically select best (scalability) model from predefined search space**

e.g., number of processes

number of terms

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

(model) constant

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

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

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

[1]: A. Calotoiu, TH, M. Poke, F. Wolf: Using Automated Performance Modeling to Find Scalability Bugs in Complex Codes, IEEE/ACM SC13

Requirements modeling II: Automated best-fit modeling

- **Manual kernel selection and hypothesis generation is time consuming (and boring)**
- **Idea: Automatically select best model from predefined space**

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

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

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

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

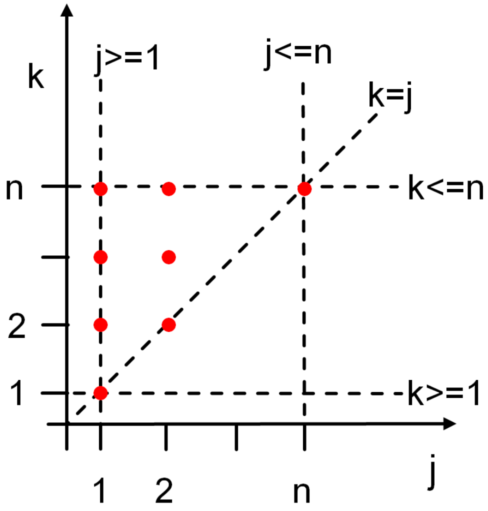
$c_1 \cdot \log(p) + c_2 \cdot p$
$c_1 \cdot \log(p) + c_2 \cdot p \cdot \log(p)$
$c_1 \cdot \log(p) + c_2 \cdot p^2$
$c_1 \cdot \log(p) + c_2 \cdot p^2 \cdot \log(p)$
$c_1 \cdot p + c_2 \cdot p \cdot \log(p)$
$c_1 \cdot p + c_2 \cdot p^2$
$c_1 \cdot p + c_2 \cdot p^2 \cdot \log(p)$
$c_1 \cdot p \cdot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

[1]: A. Calotoiu, T. Hoefler, M. Poke, F. Wolf: Using Automated Performance Modeling to Find Scalability Bugs in Complex Codes, IEEE/ACM SC13

Requirements modeling III: Source-code analysis [1]

- **Extra-P selects model based on best fit to the data**
  - What if the data is not sufficient or too noisy?
- **Back to first principles**
  - The source code describes all possible executions
  - Describing all possibilities is too expensive, focus on counting loop iterations symbolically

```
for (j = 1; j <= n; j = j*2)
  for (k = j; k <= n; k = k++)
    OperationInBody(j,k);
```

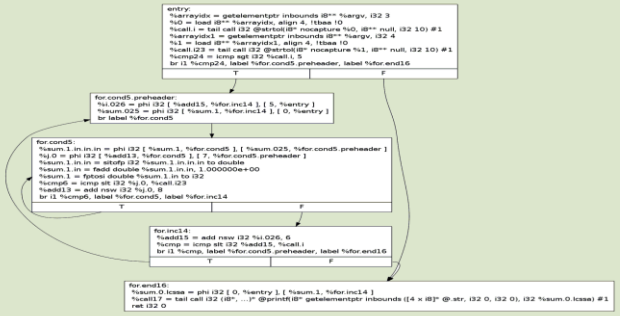


$$N = (n+1)\log_2 n - n + 2$$

Parallel program

```
    do i = 1, procCols
       call mpi_irecv( buff, 2, dp_type, reduce_exch_proc(i),
>                      i, mpi_comm_world, request, ierr )
       call mpi_send( buff2, 2, dp_type, reduce_exch_proc(i),
>                     i, mpi_comm_world, ierr )
       call mpi_wait( request, status, ierr )
    enddo

    do i = id *n/p, ( id +1)* n/p
       do j = 1, nSize
          call compute
```

Loop extraction



Number of iterations

$$N = \sum_{i_1=0}^{n_1(x_{0,1})} \sum_{i_2=0}^{n_2(x_{0,2})} \cdots \sum_{i_{r-1}=0}^{n_{r-1}(x_{0,r-1})} n_r(x_{0,r}).$$

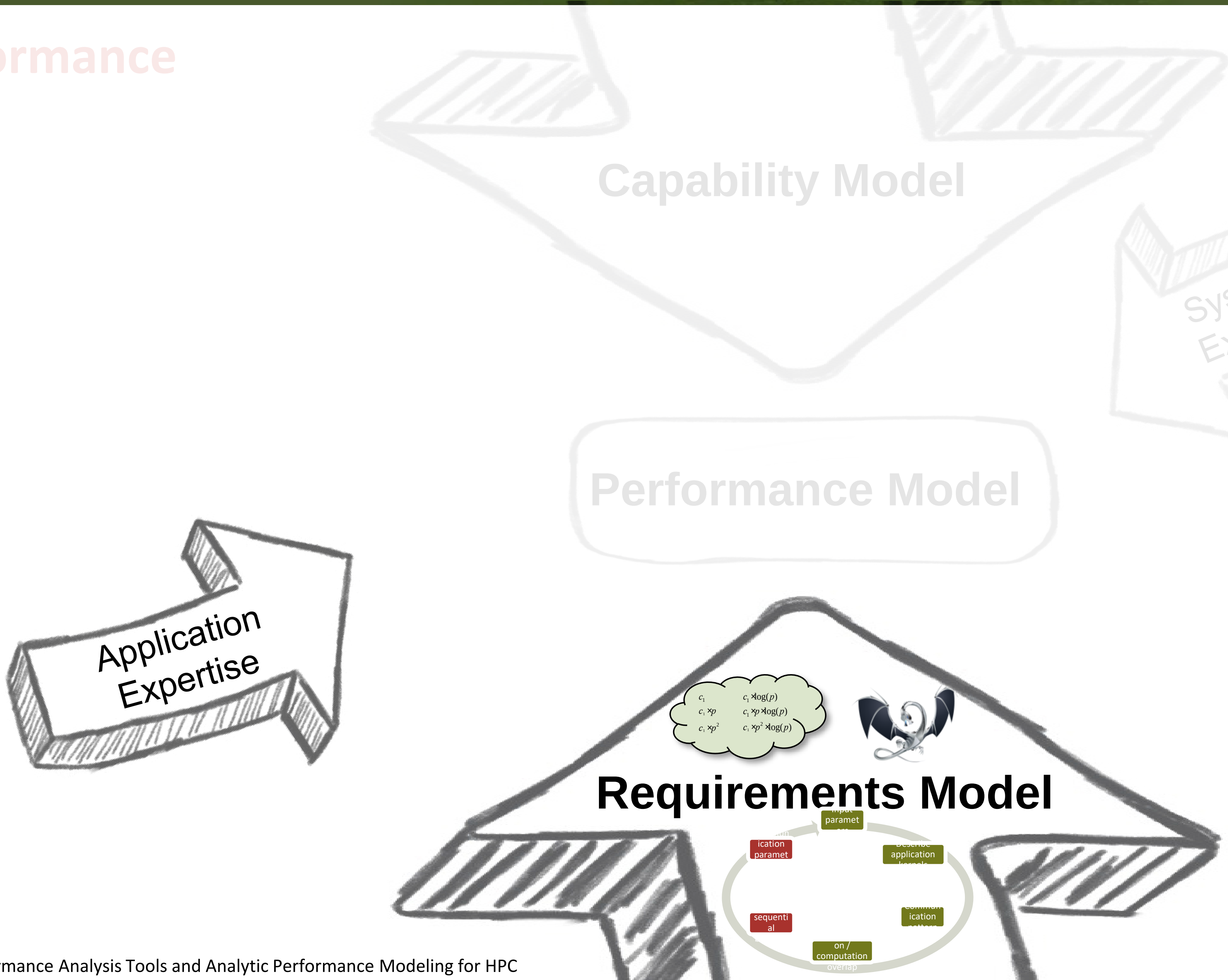
Requirements Models

$$W = N\big|_{p=1}$$

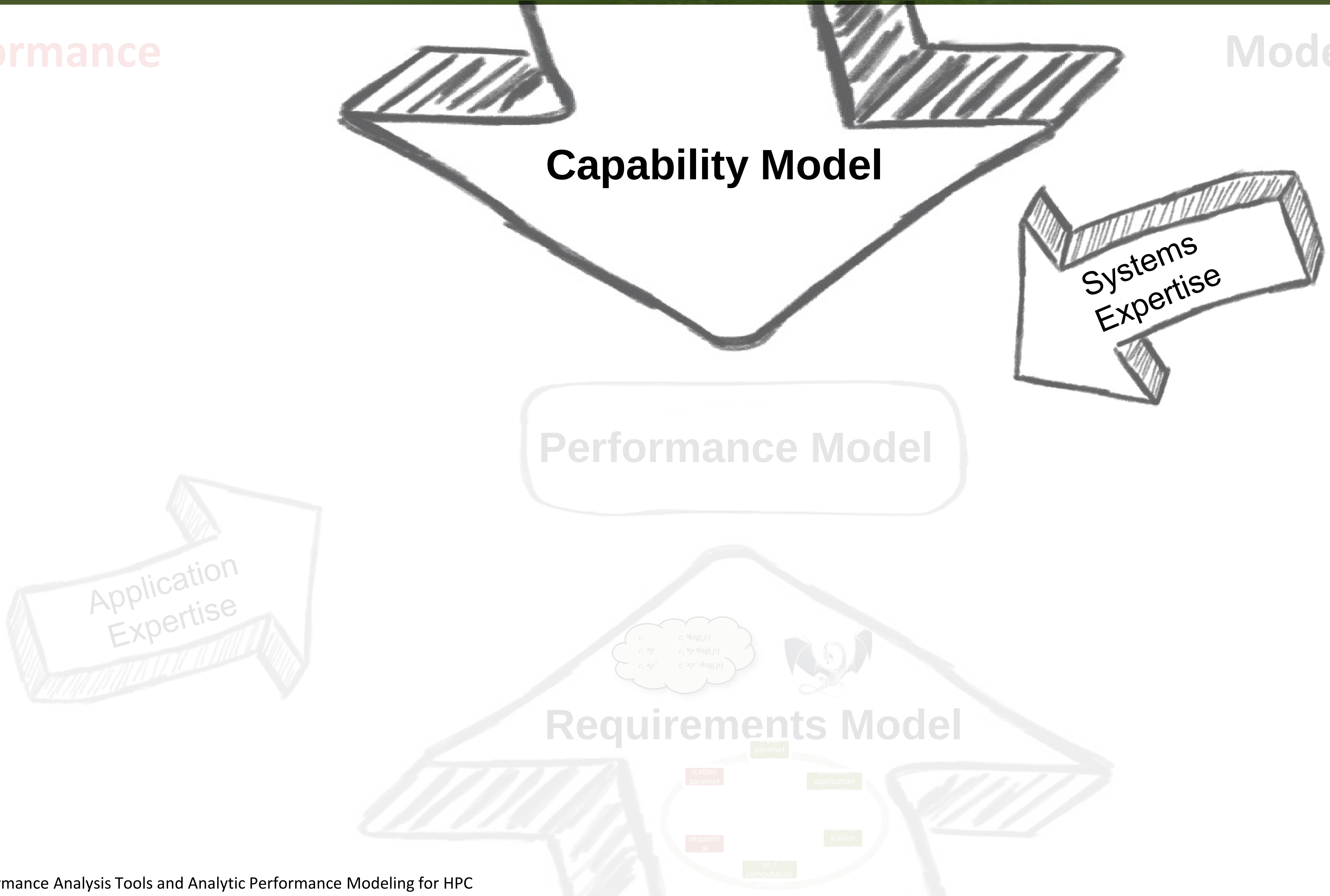
$$D = N\big|_{p\to\infty}$$

[1]: TH, G. Kwasniewski: Automatic Complexity Analysis of Explicitly Parallel Programs, ACM SPAA'14

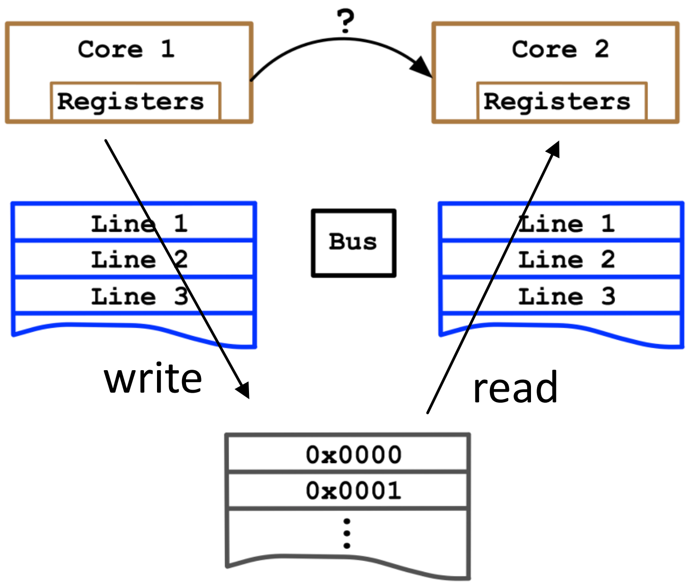
Performance Modeling



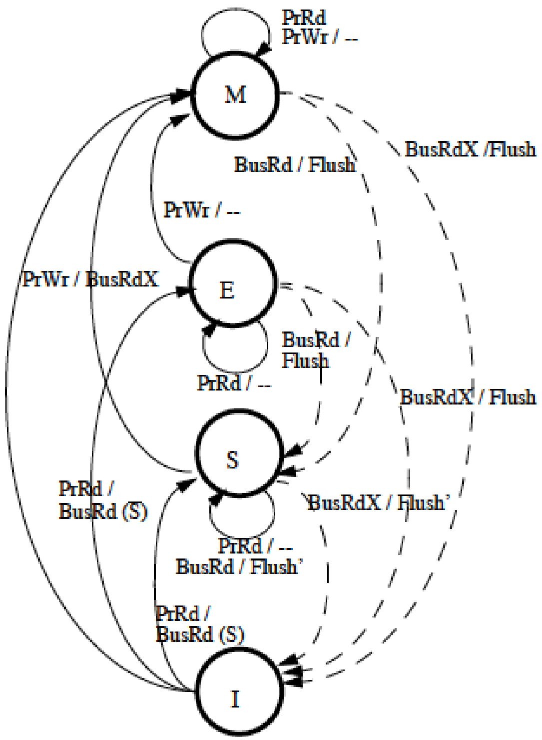
Performance Modeling



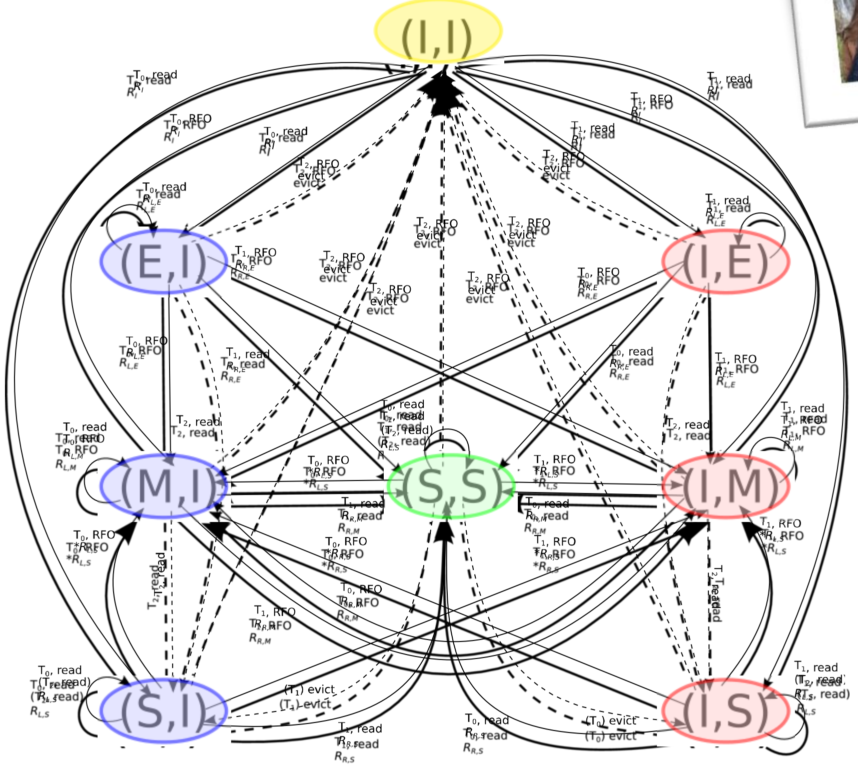
Capability models for cache-to-cache communication

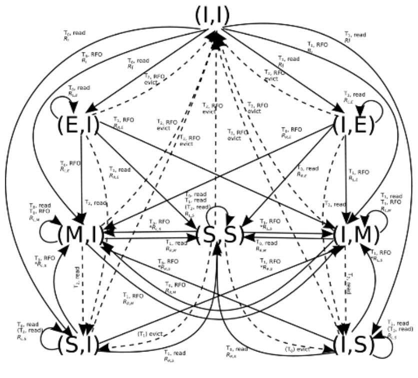


X

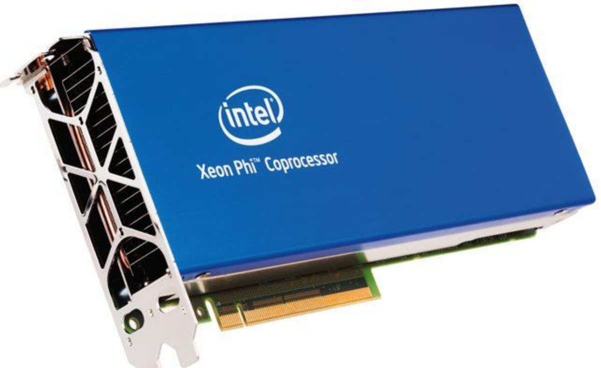


=

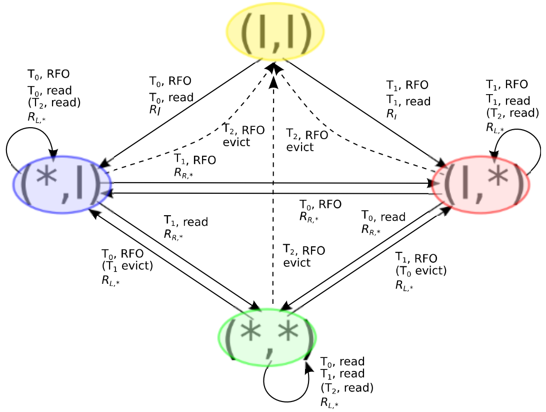




|



=



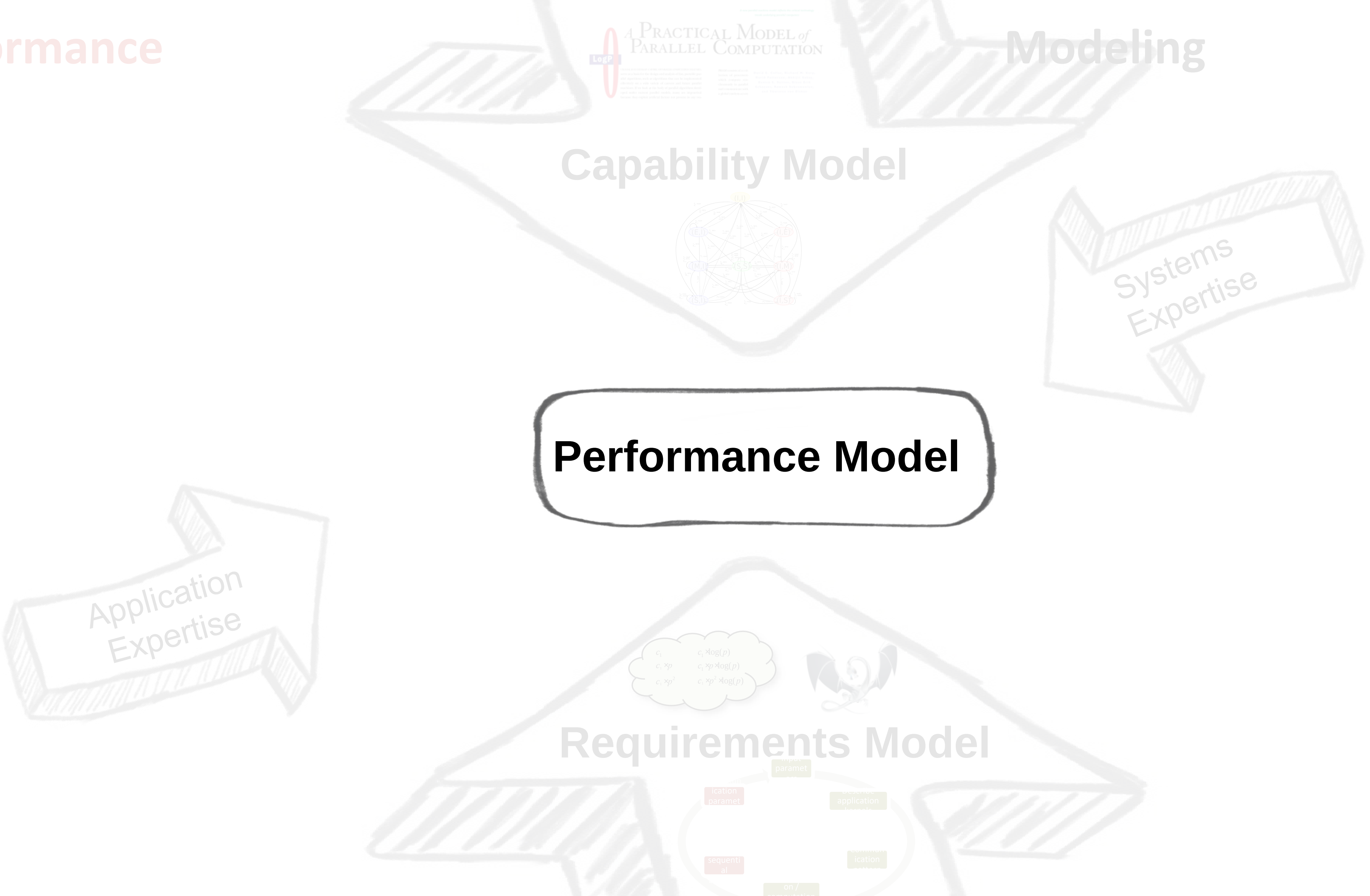
Invalid read R_I = 278 ns

Local read: R_L = 8.6 ns

Remote read R_R = 235 ns

S. Ramos, TH: *"Modeling Communication in Cache-Coherent SMP Systems - A Case-Study with Xeon Phi ", ACM HPDC'13*

Performance Modeling



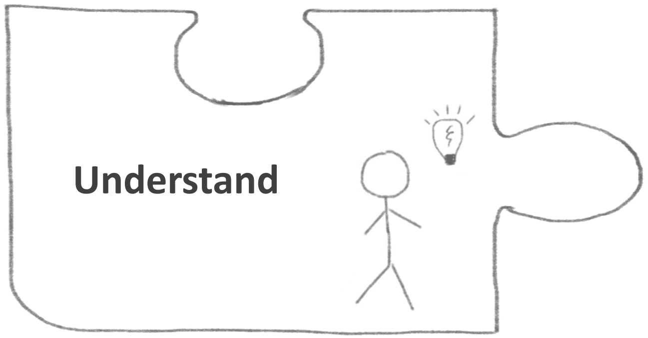
Part III: Understand

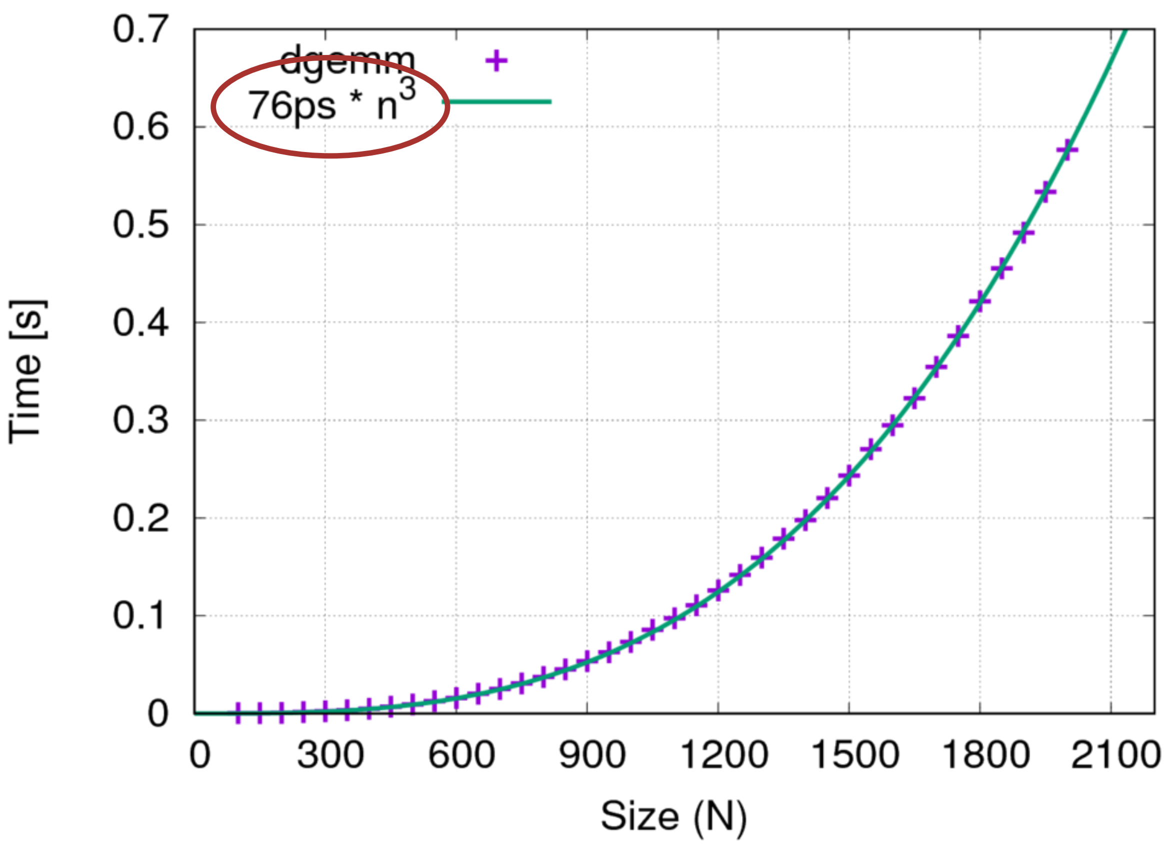
- **Use models to**
  1. Proof optimality of real implementations
     - *Stop optimizing, step back to algorithm level*
  2. Design optimal algorithms or systems in the model
     - *Can lead to non-intuitive designs*

- **Proof optimality of matrix multiplication**
  - Intuition: flop rate is the bottleneck
  - $t(n) = 76ps * n^3$
  - **Flop rate** $R = 2flop * n^3/(76ps * n^3)$ = **27.78 Gflop/s**
  - **Flop peak**: 3.864 GHz * 8 flops = **30.912 Gflop/s**

    Achieved ~90% of peak (IBM Power 7 IH @3.864GHz)

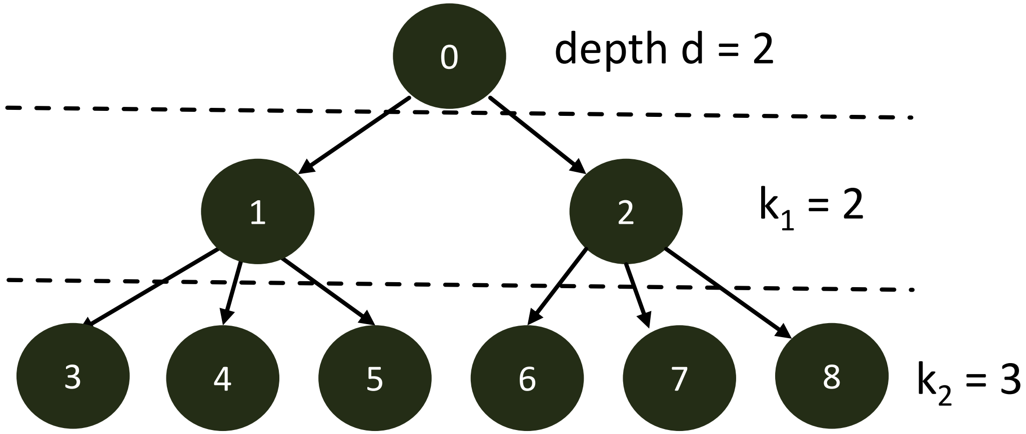
- **Gets more complex quickly**
  - Imagine sparse matrix-vector

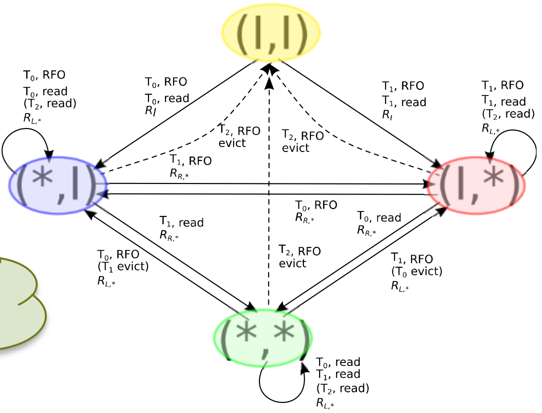




Design algorithms – bcast in cache-to-cache model

Multi-ary tree example





Tree depth

Level size

$$\mathcal{T}_{tree} = \sum_{i=1}^{d} \mathcal{T}_C(k_i) = \sum_{i=1}^{d} (c \cdot k_i + b)$$

Tree cost

$$= \sum_{i=1}^{d} (R_R + R_L + c \cdot (k_i - 1))$$

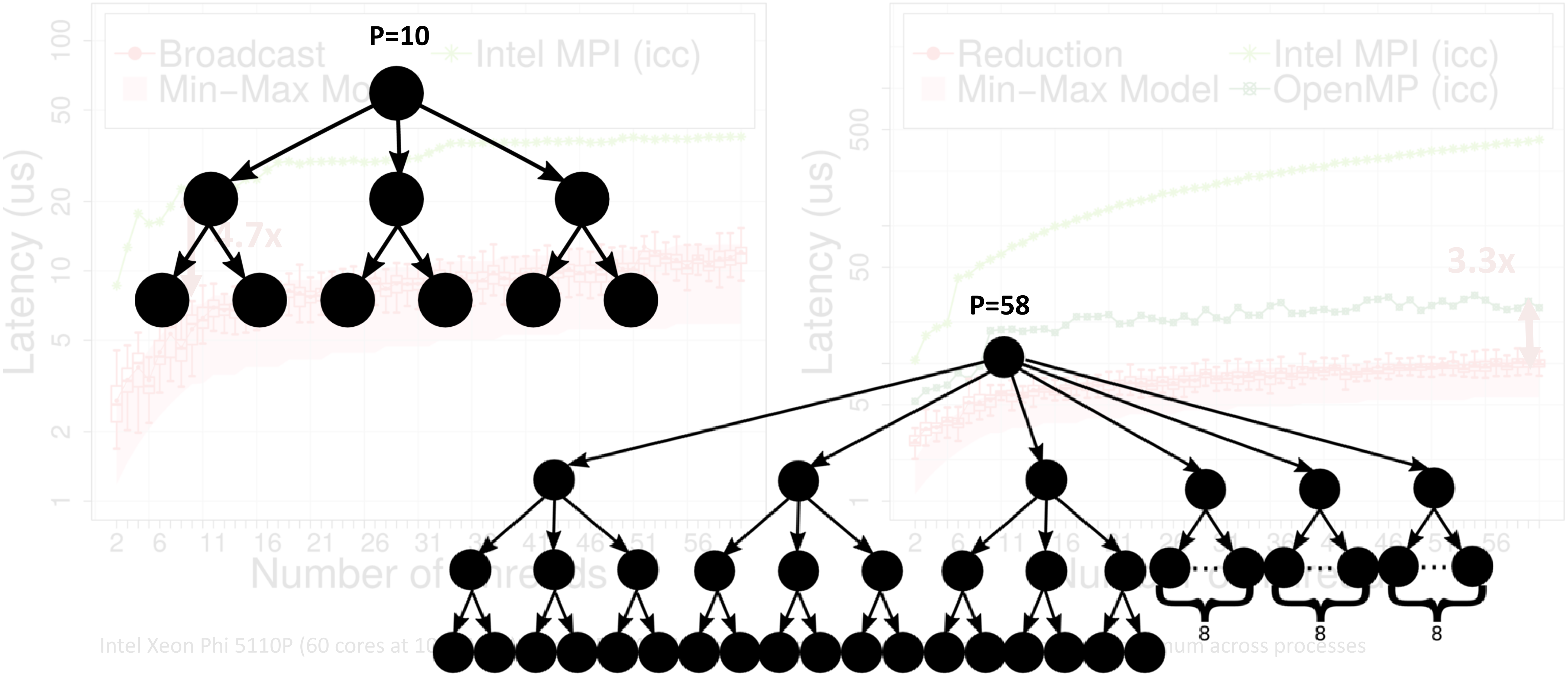
$$\mathcal{T}_{sbcast} = \min_{d,k_i} \left(\mathcal{T}_{fw} + \sum_{i=1}^{d} (c \cdot k_i + b) + \sum_{i=1}^{d} \mathcal{T}_{nb}(k_i + 1) \right)$$

Reached threads

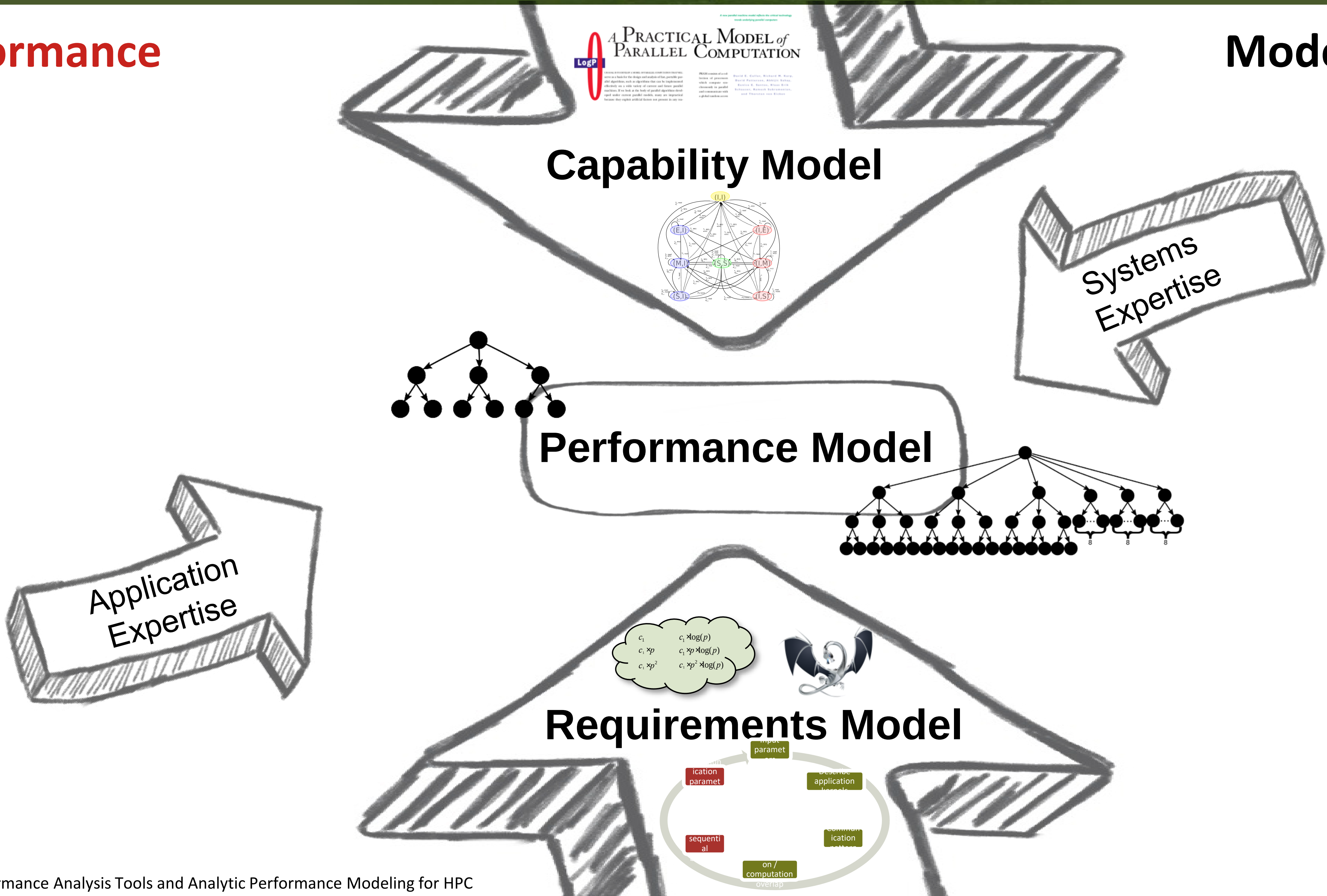
$$N \leq 1 + \sum_{i=1}^{d} \prod_{j=1}^{i} k_j, \quad \forall i < j, k_i \leq k_j$$

S. Ramos, TH: *"Modeling Communication in Cache-Coherent SMP Systems - A Case-Study with Xeon Phi ", ACM HPDC'13*

Measured results – small broadcast and reduction



Performance Modeling



How to continue from here?

DAPP Transformation System

- User-supported, compile- and run-time

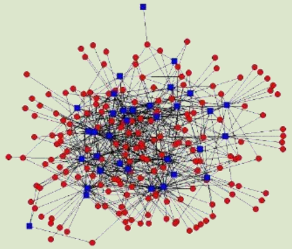
DAPP Parallel Language

- Data-centric, explicit requirements models

DAPP

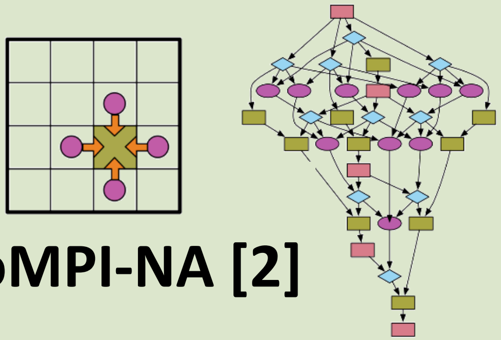
European Research Council
Established by the European Commission
Supporting top researchers from anywhere in the world

Performance-transparent Platforms



HTM [1]

MPI RMA



foMPI-NA [2]

NISA [3]

portals

[1]: M. Besta, TH: Accelerating Irregular Computations with Hardware Transactional Memory and Active Messages, ACM HPDC’15
[2]: R. Belli, TH: Notified Access: Extending Remote Memory Access Programming Models for Producer-Consumer Synchronization, IPDPS’15
[3]: S. Di Girolamo, P. Jolivet, K. D. Underwood, TH: Exploiting Offload Enabled Network Interfaces, IEEE Micro’16