Developing high-performance software – from modeling to programming.

Torsten Hoefler
Department of Computer Science
ETH Zurich

Invited talk at Multicore@Siemens conference, Nuremberg, Feb. 2018

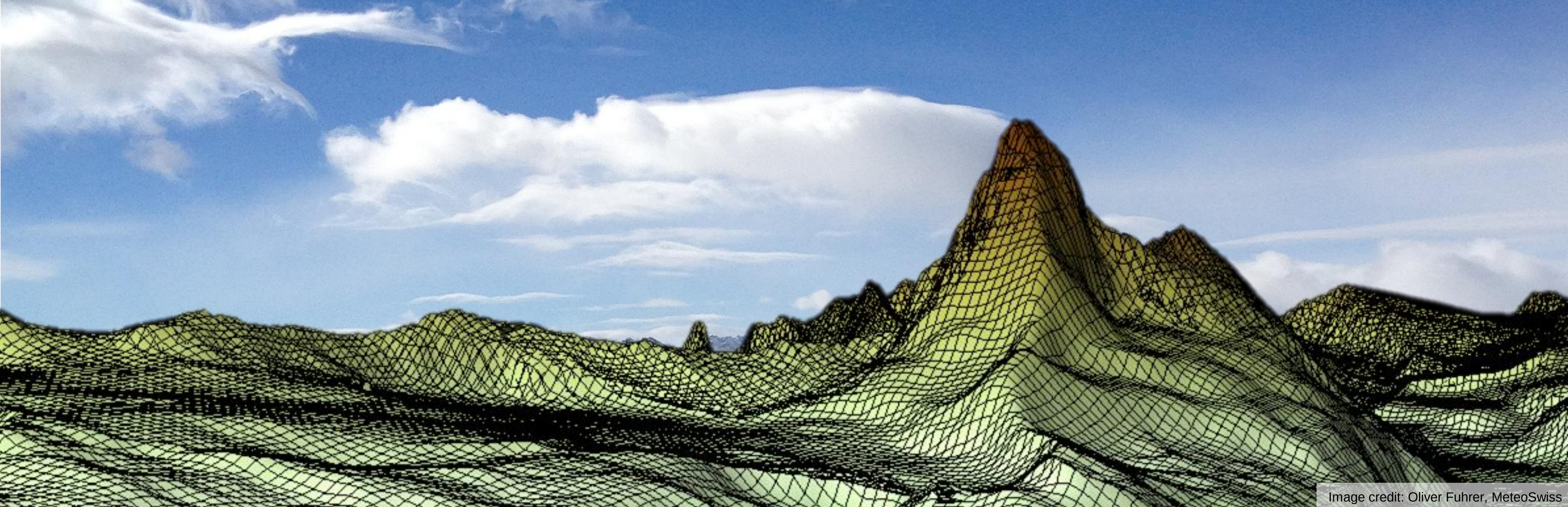


Image credit: Oliver Fuhrer, MeteoSwiss

A factor **2x** in resolution roughly corresponds to a factor **10x** compute

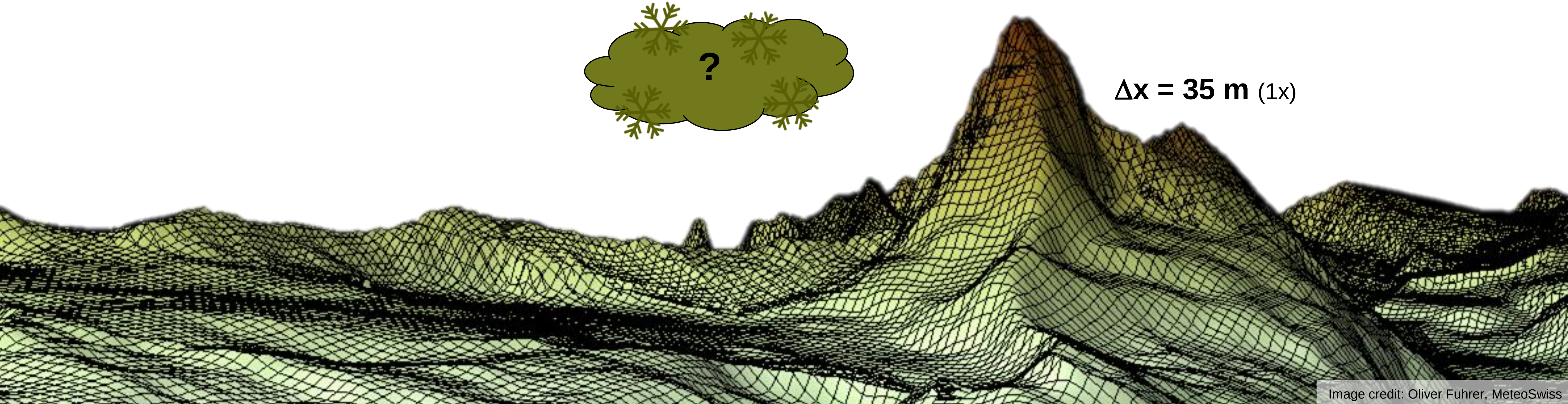
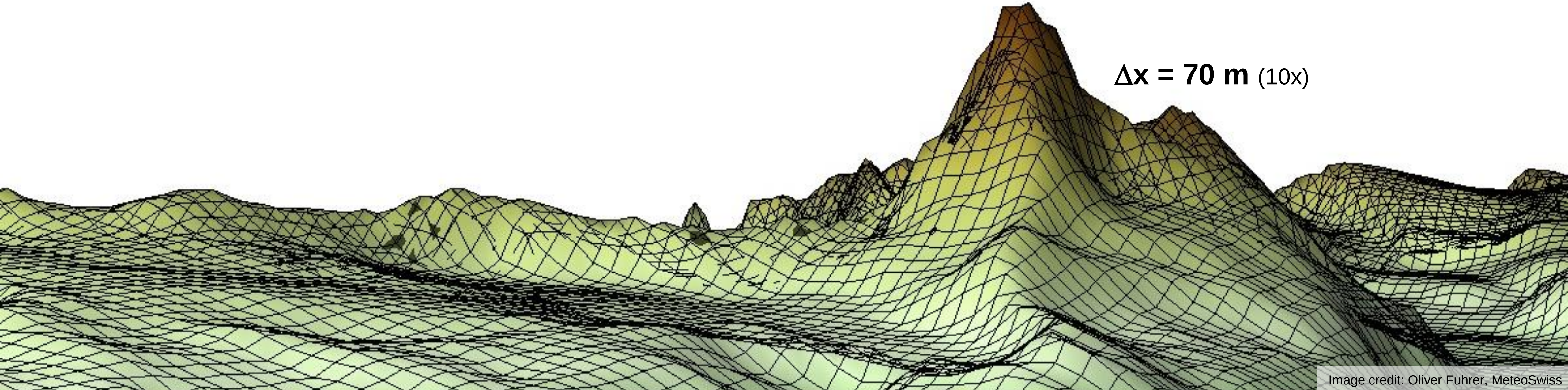
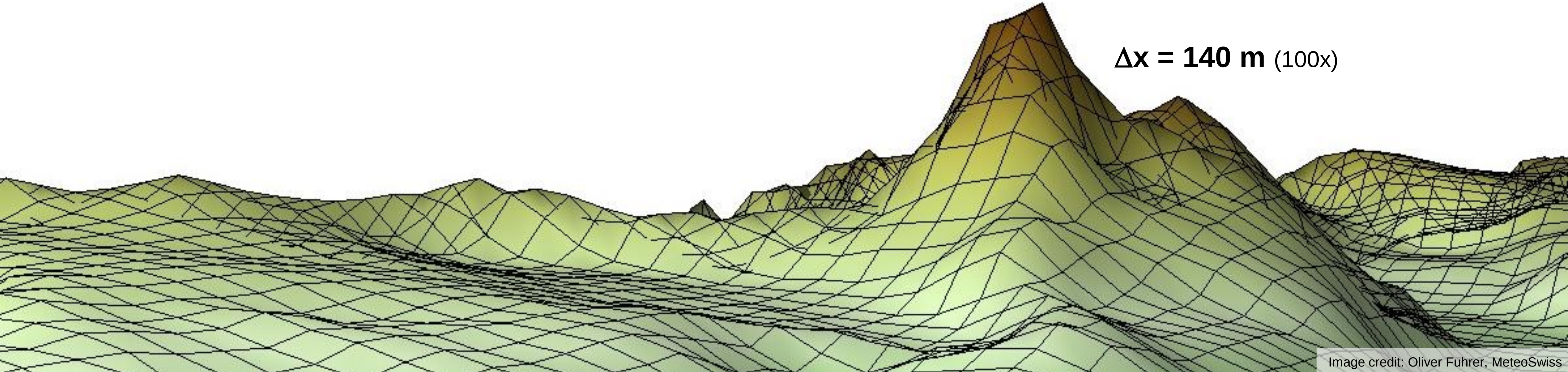


Image credit: Oliver Fuhrer, MeteoSwiss

A factor **2x** in resolution roughly corresponds to a factor **10x** compute



A factor **2x** in resolution roughly corresponds to a factor **10x** compute



Δx = 140 m (100x)

Image credit: Oliver Fuhrer, MeteoSwiss

A factor **2x** in resolution roughly corresponds to a factor **10x** compute

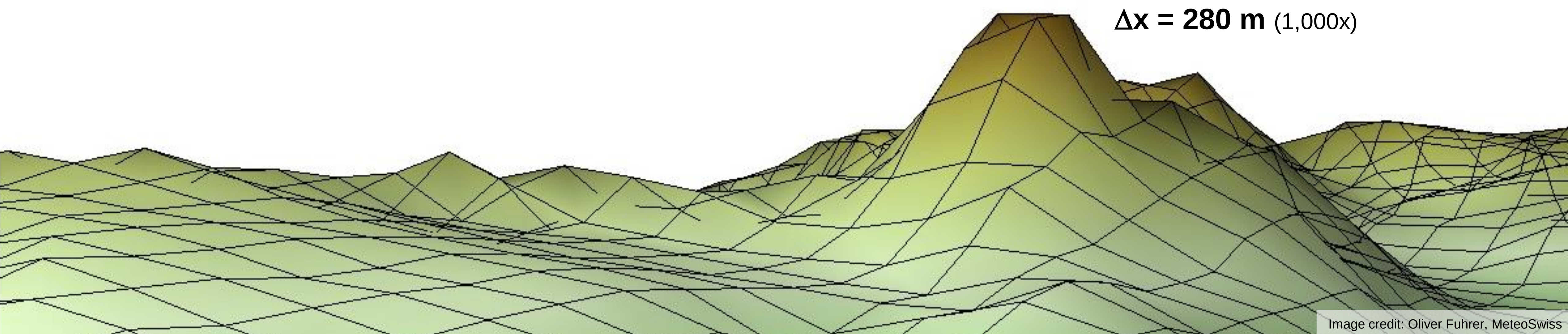


Image credit: Oliver Fuhrer, MeteoSwiss

A factor **2x** in resolution roughly corresponds to a factor **10x** compute

Δx = 550 m (10,000x)

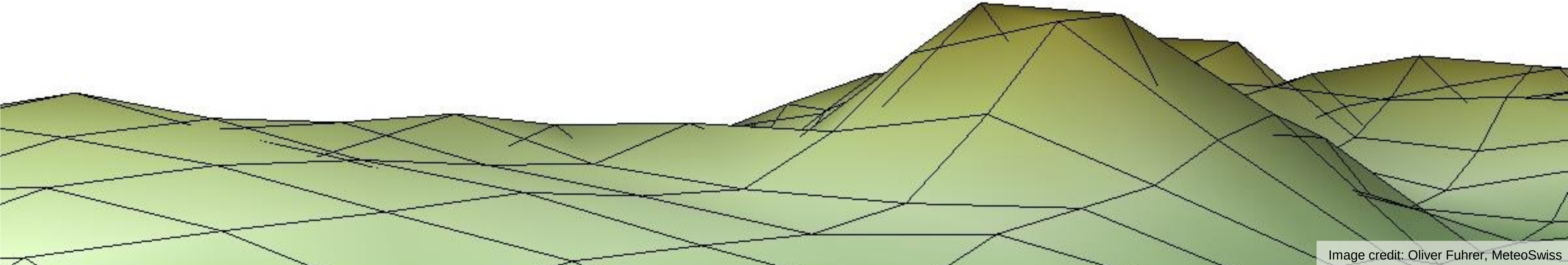


Image credit: Oliver Fuhrer, MeteoSwiss

A factor **2x** in resolution roughly corresponds to a factor **10x** compute

Operational model of MeteoSwiss today!

Δx = 1100 m (100,000x)

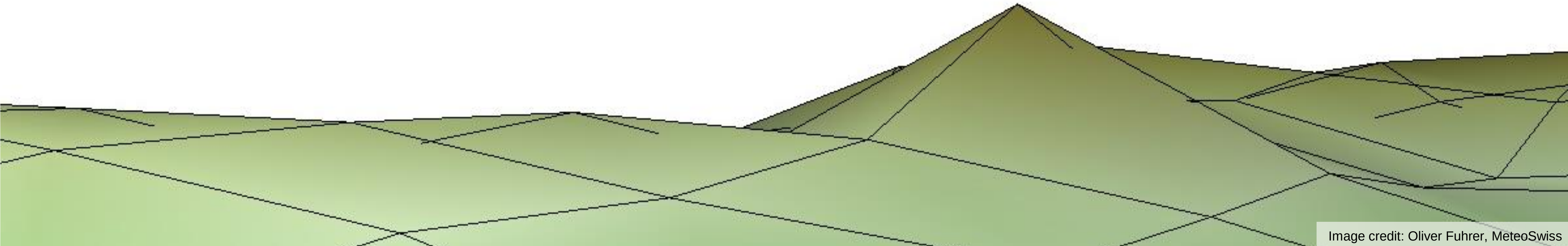
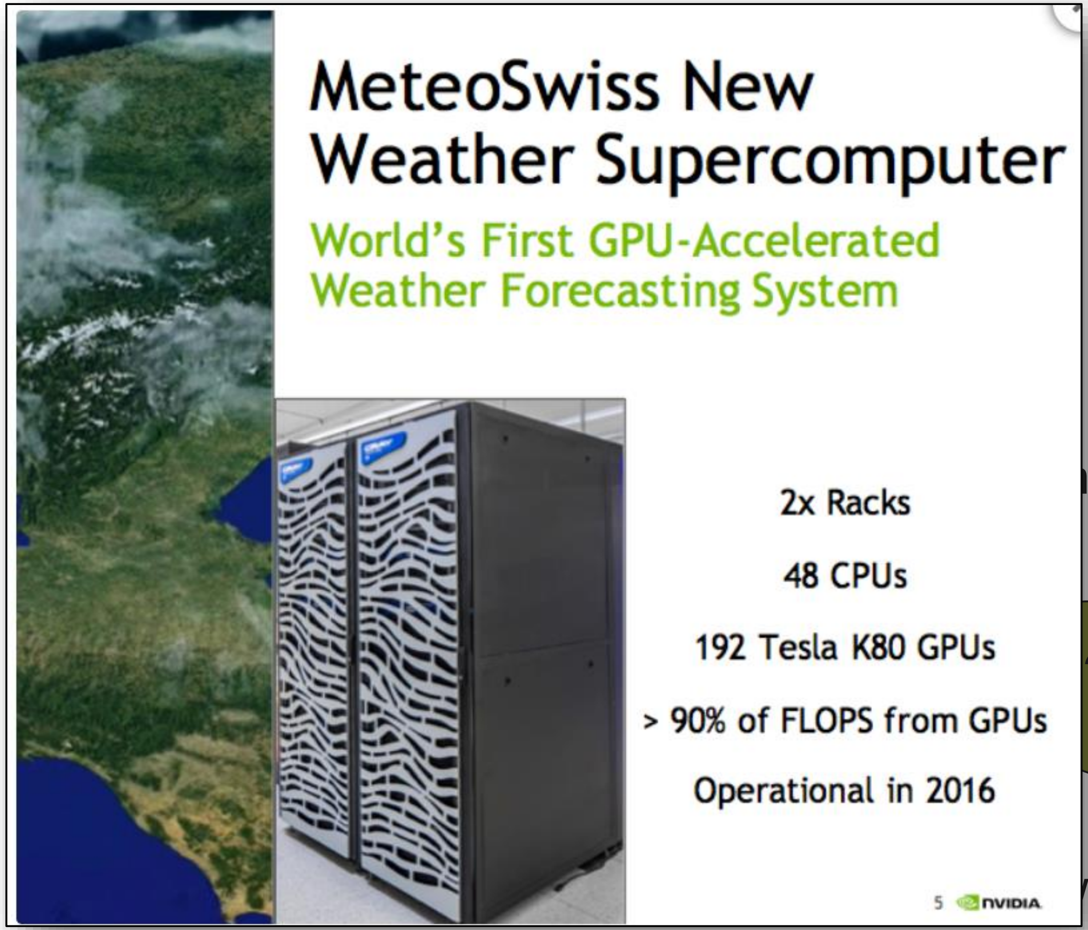
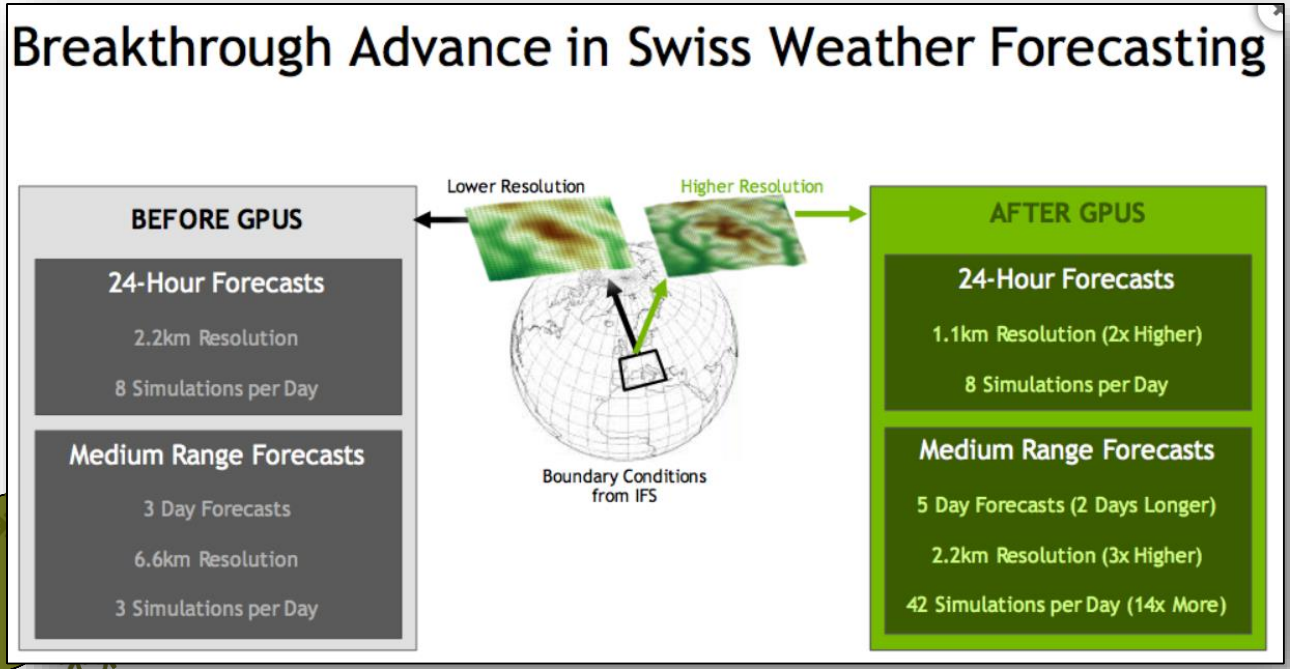


Image credit: Oliver Fuhrer, MeteoSwiss



ds

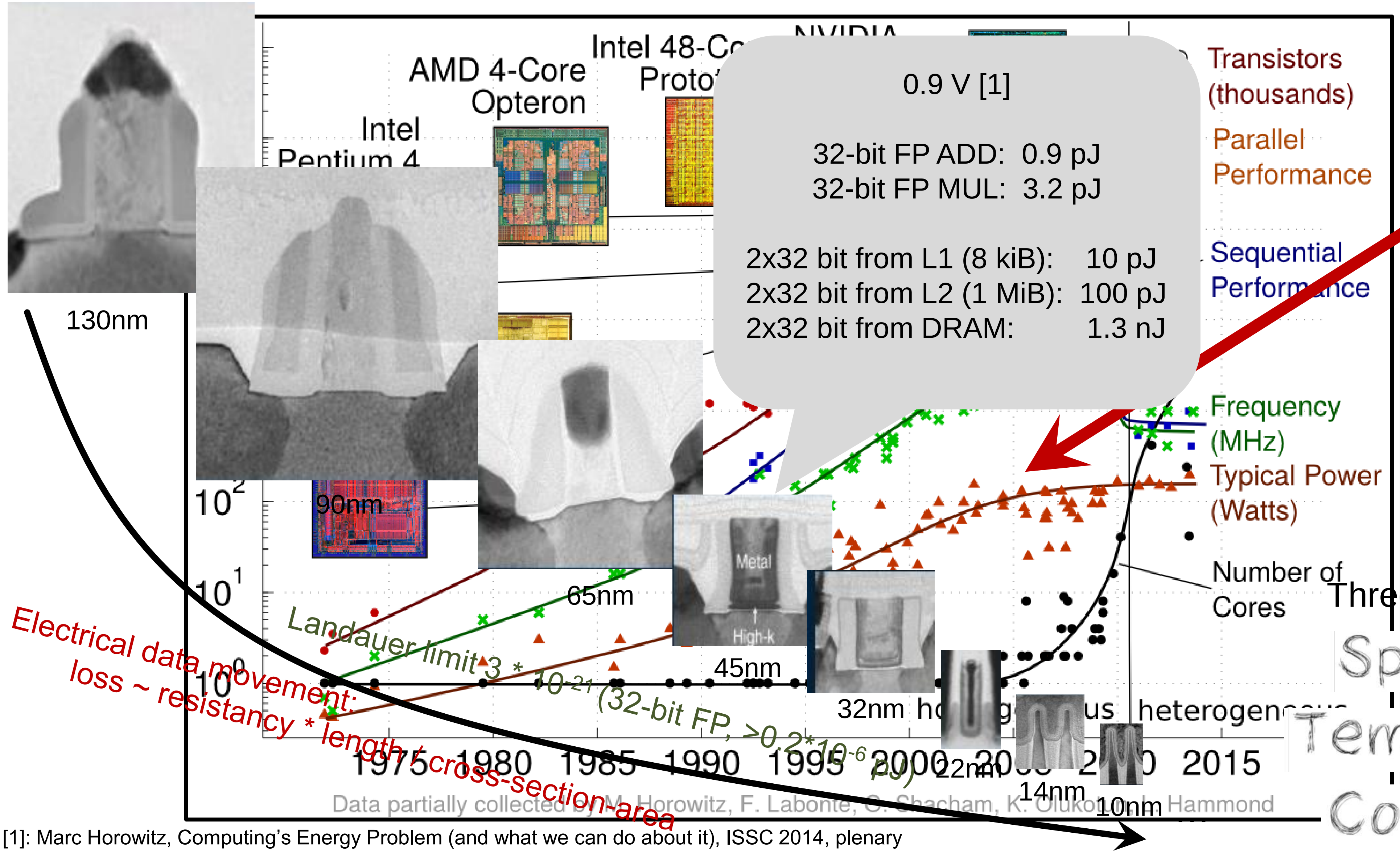
?



ay!

Image credit: Oliver Fuhrer, MeteoSwiss

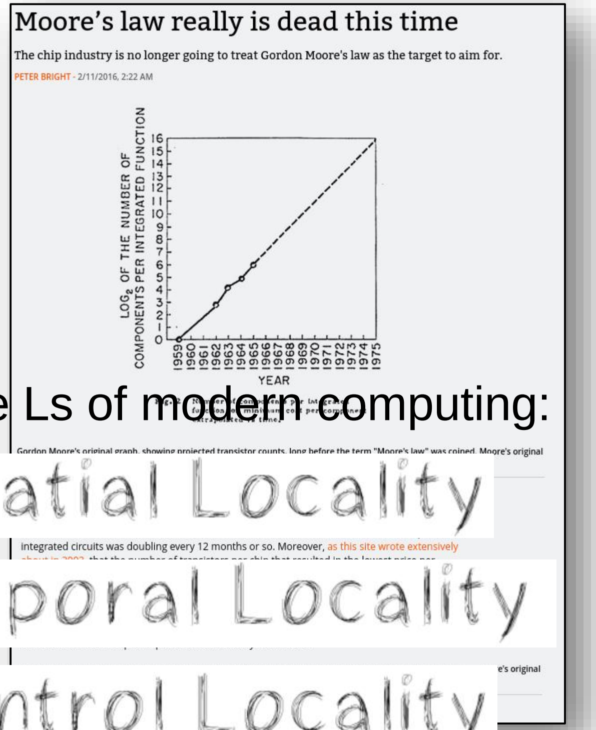
Changing hardware constraints and the physics of computing



0.9 V [1]

32-bit FP ADD: 0.9 pJ
32-bit FP MUL: 3.2 pJ

2x32 bit from L1 (8 kiB): 10 pJ
2x32 bit from L2 (1 MiB): 100 pJ
2x32 bit from DRAM: 1.3 nJ


Moore's law really is dead this time

The chip industry is no longer going to treat Gordon Moore's law as the target to aim for.

PETER BRIGHT - 2/11/2016, 2:22 AM

integrated circuits was doubling every 12 months or so. Moreover, as this site wrote extensively

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

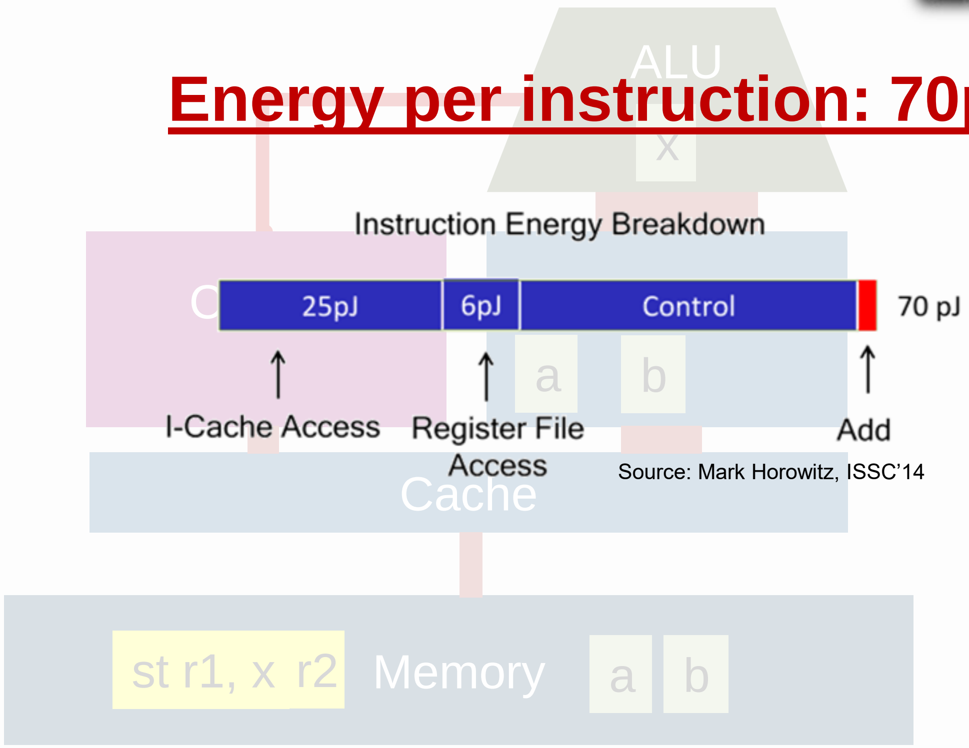
Load-store vs. Dataflow architectu

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

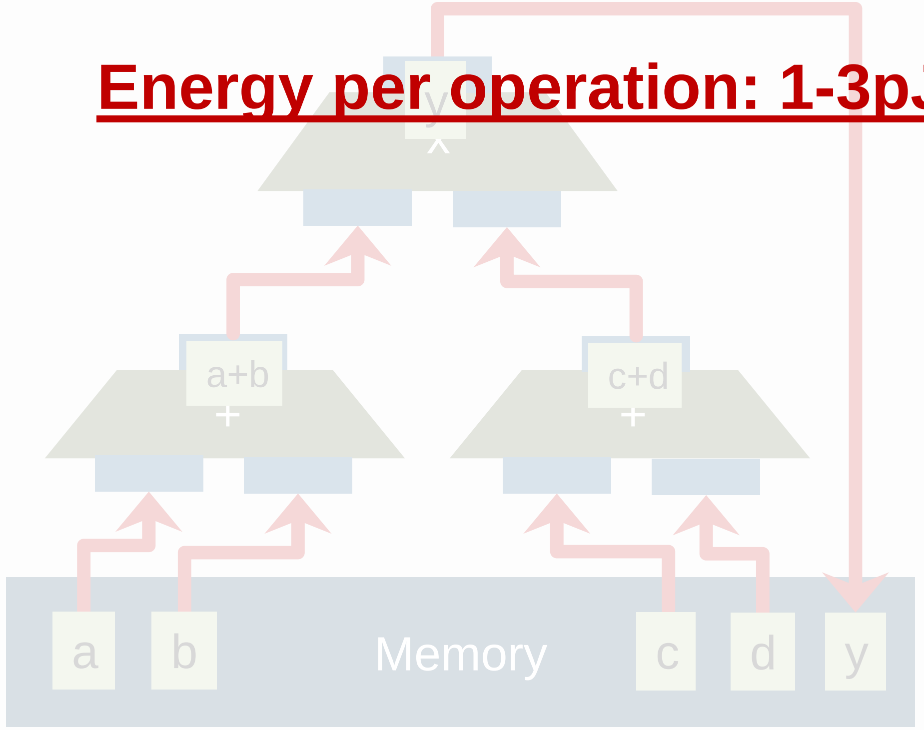
Energy per instruction: 70pJ



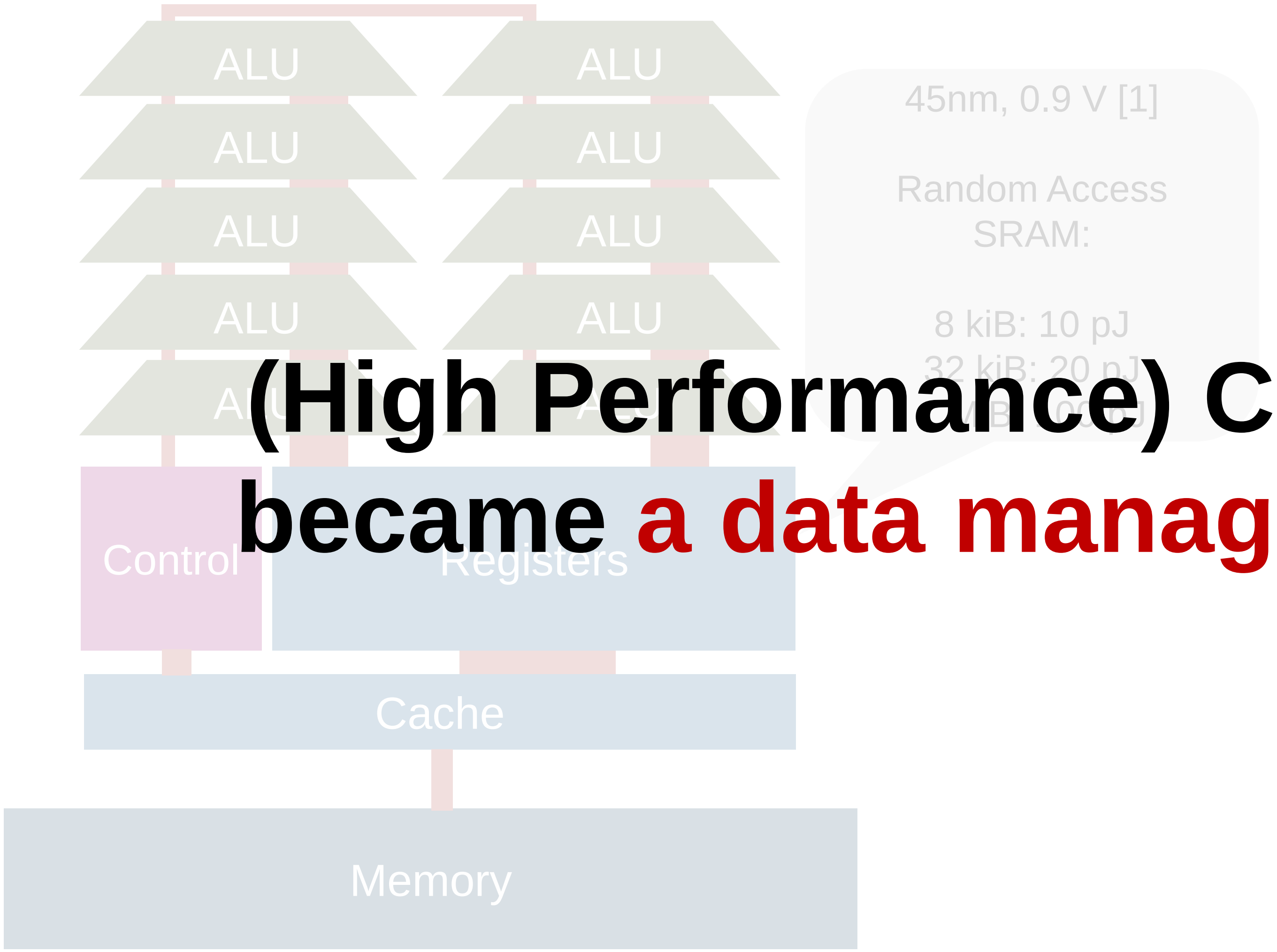
Static Dataflow ("non von Neumann")

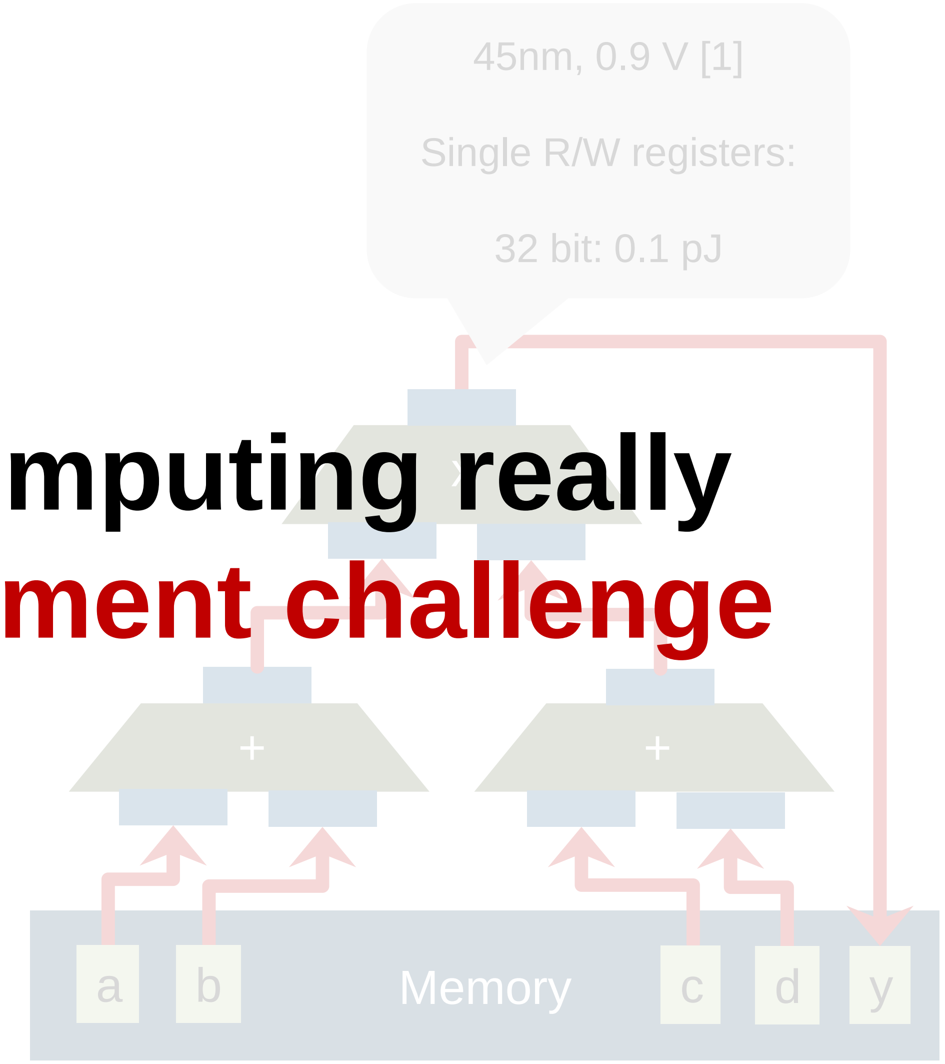
y=(a+b)*(c+d)

Energy per operation: 1-3pJ



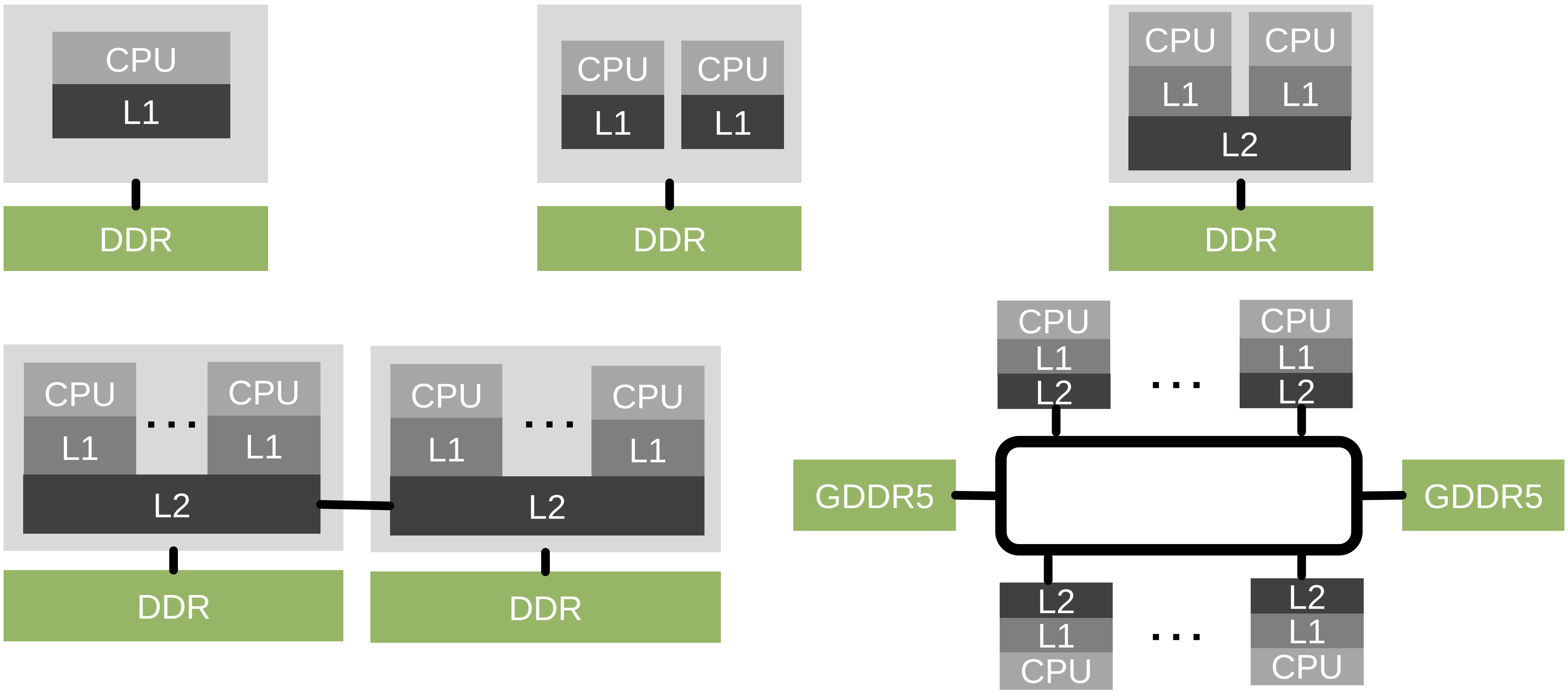
Single Instruction Multiple Data/Threads (SIMD - Vector CPU, SIMT - GPU)





(High Performance) Computing really became a data management challenge

But memory architectures are becoming more and more complex



Xeon Phi KNL: 3 memory models, 5 configuration modes each → 15 options for configuring the system!

How do we optimize codes for these complex architectures?

- **Performance engineering**: ***"encompasses the set of roles, skills, activities, practices, tools, and deliverables applied at every phase of the systems development life cycle which ensures that a solution will be designed, implemented, and operationally supported to meet the non-functional requirements for performance (such as throughput, latency, or memory usage)."***

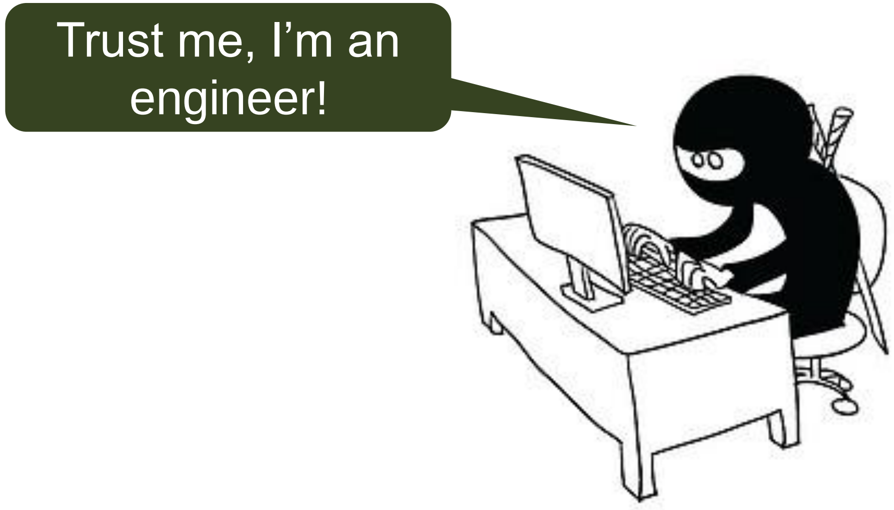
- **Manually profile codes and tune them to the given architecture**
  - Requires highly-skilled performance engineers
  - Need familiarity with

    NUMA (topology, bandwidths etc.)

    Caches (associativity, sizes etc.)

    Microarchitecture (number of outstanding loads etc.)

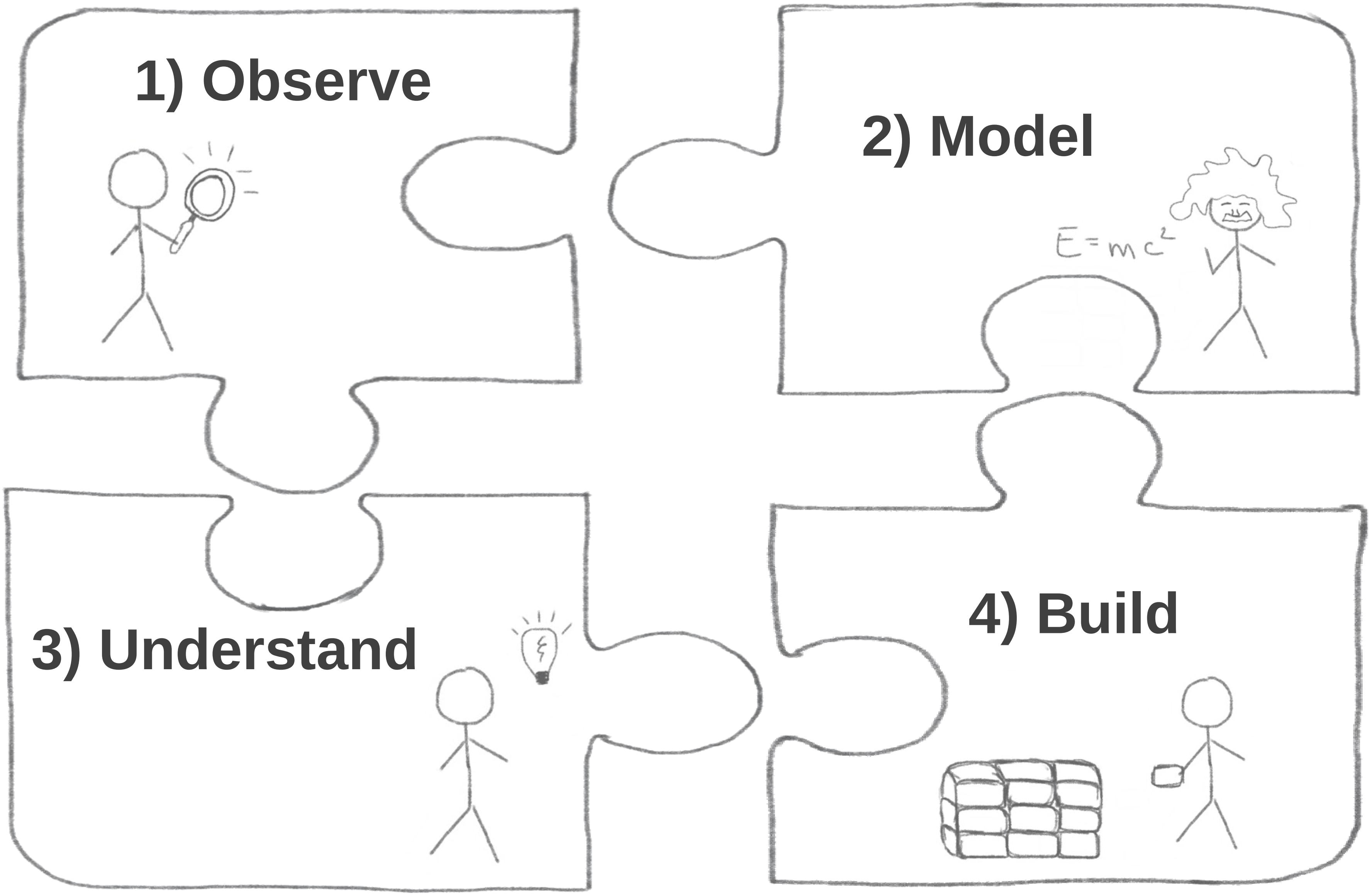
    …



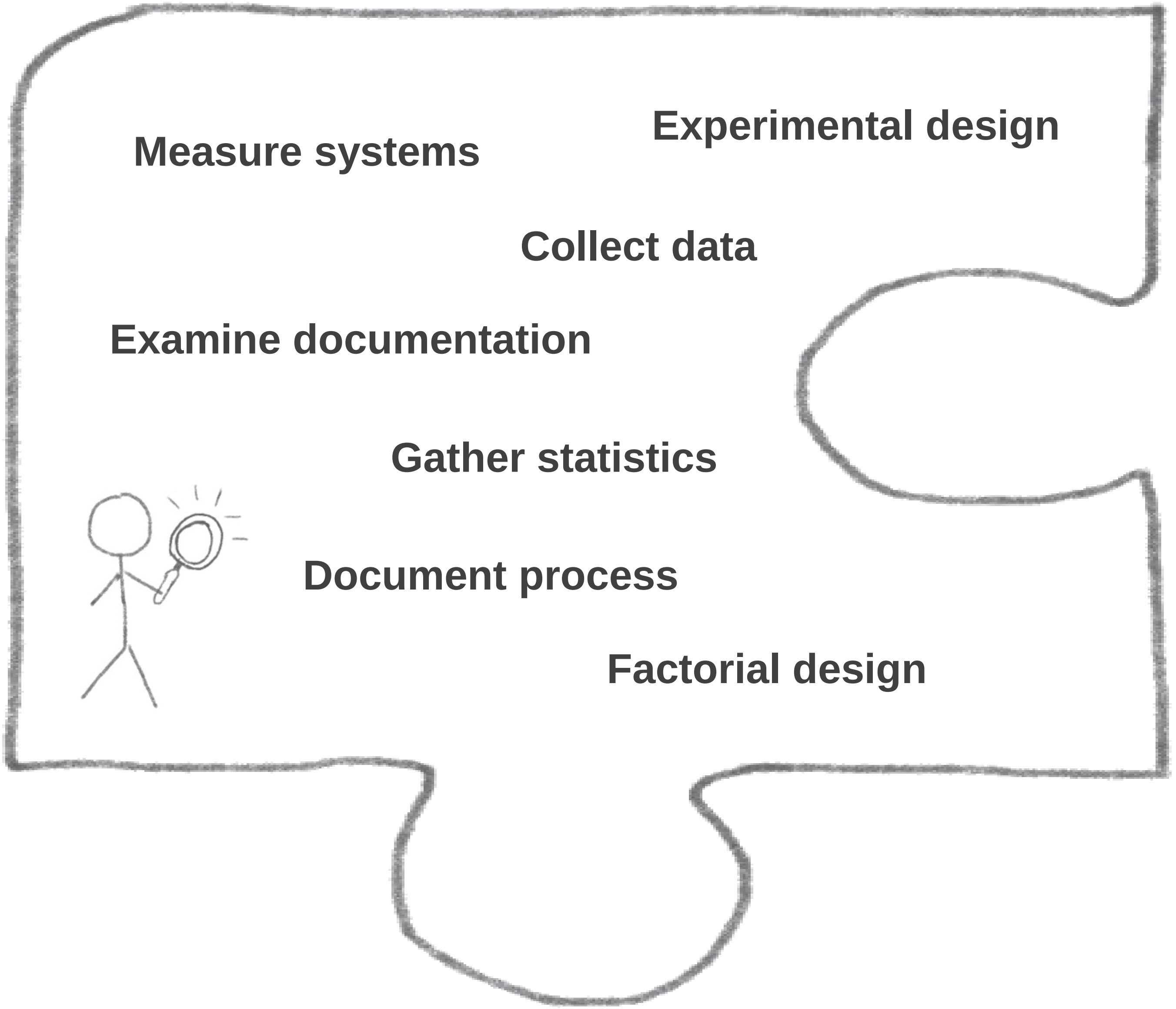
<title>code ninja</title>

An engineering example – Tacoma Narrows Bridge

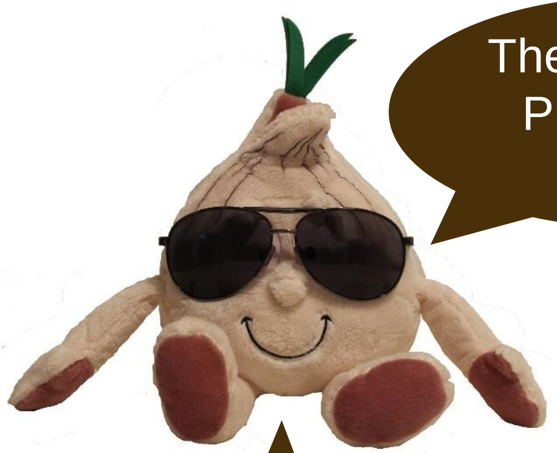
Scientific Performance Engineering



Part I: Observe

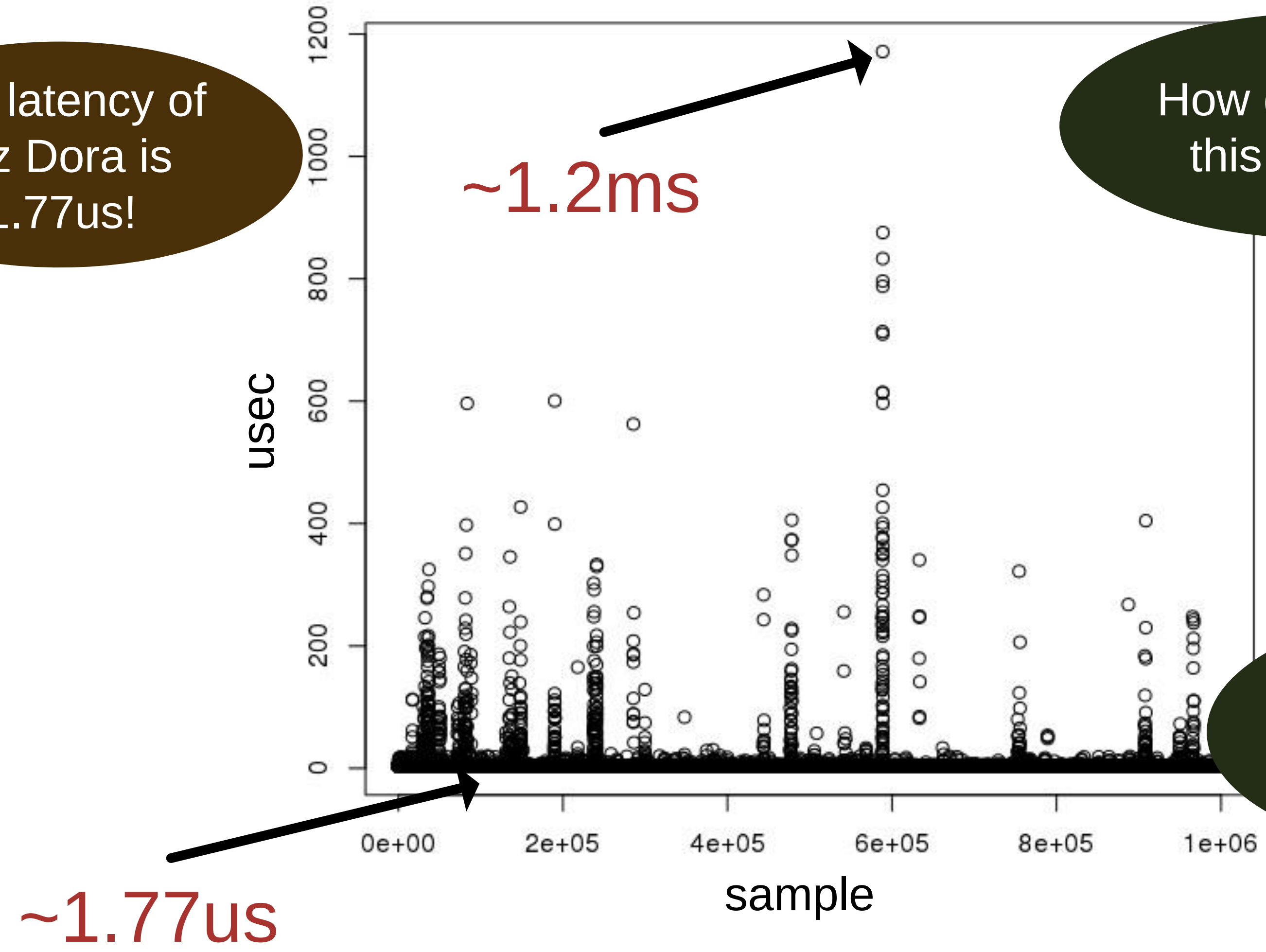


Example: Simple ping-pong latency benchmark



The latency of Piz Dora is 1.77us!

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



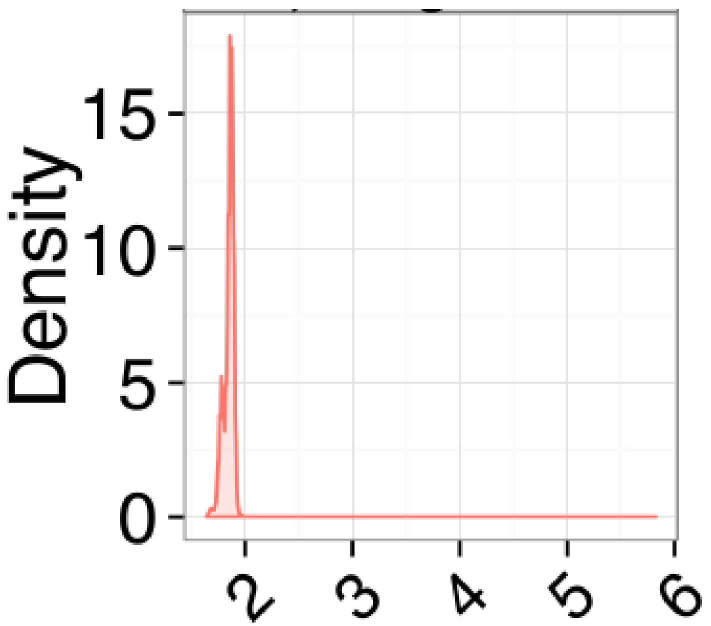
How did you get this number?

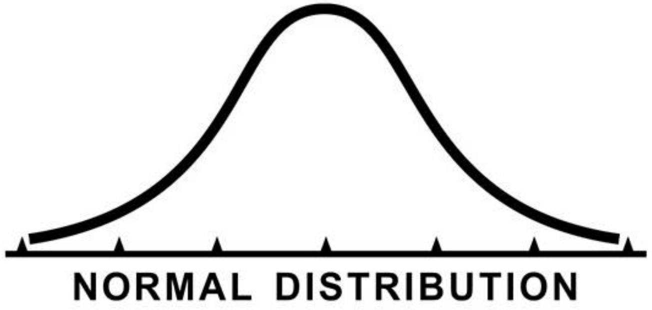
Why do you think so? Can I see the data?

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!





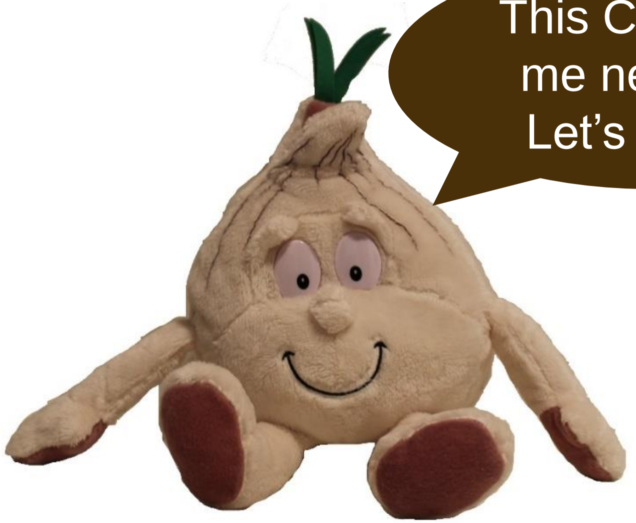
Did you assume normality?

Can we test for normality?

TH, Belli: Scientific Benchmarking of Parallel Computing Systems, IEEE/ACM SC15

Looking at the data in detail

This CI makes me nervous. Let's check!



Clearly, the mean/median are not sufficient!

Try quantile regression!

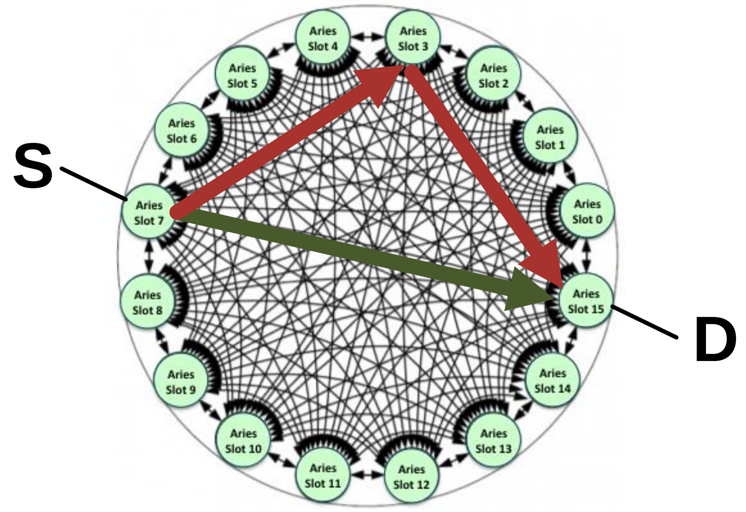
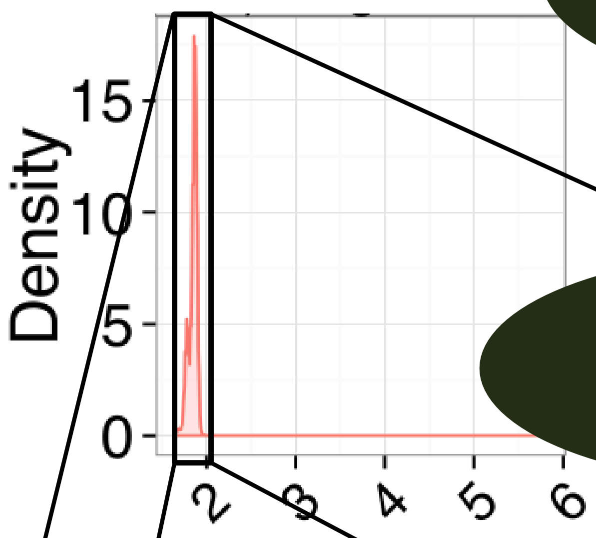
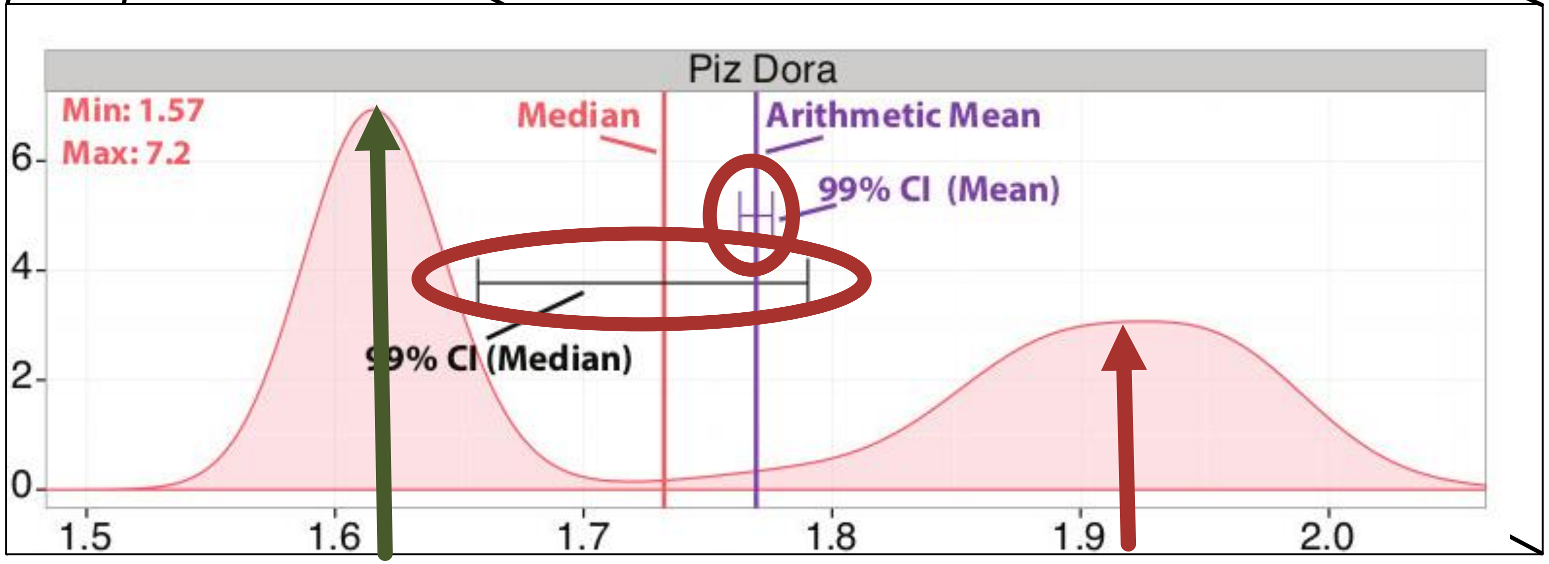


Image credit: nersc.gov





Scientific benchmarking of parallel computing systems

ACM/IEEE Supercomputing 2015 (SC15)

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.

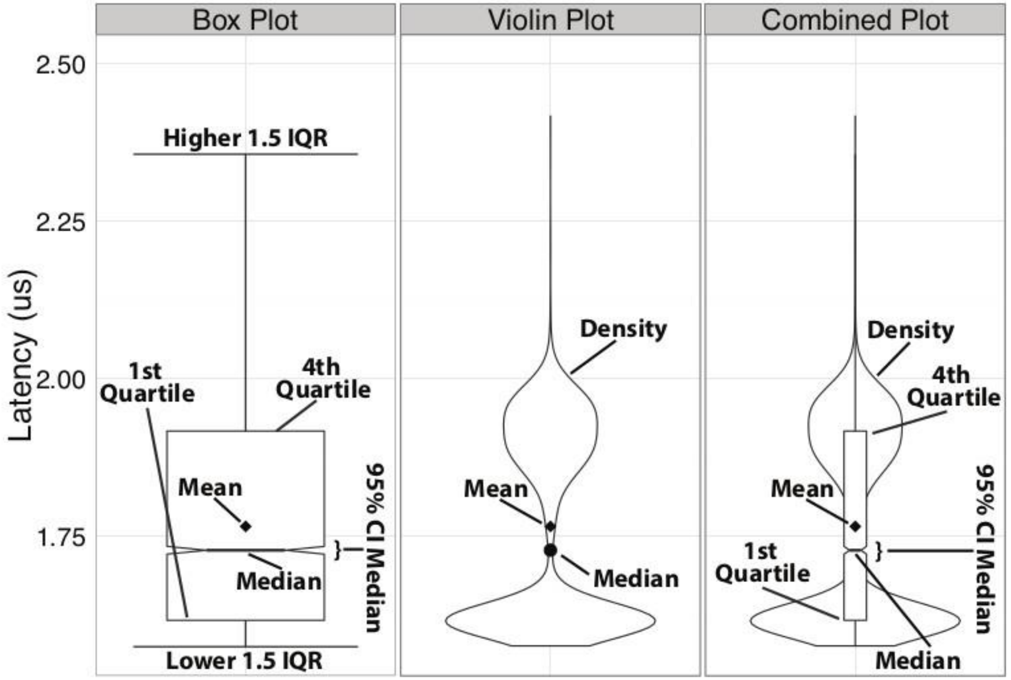
The complexity and uniqueness of many supercomputers makes reproducibility a hard task. For example, it is practically impossible to recreate most hero-runs that utilize the world's largest machines because these machines are often unique and their software configurations changes regularly. We introduce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

erpret the
by lines if
alid.

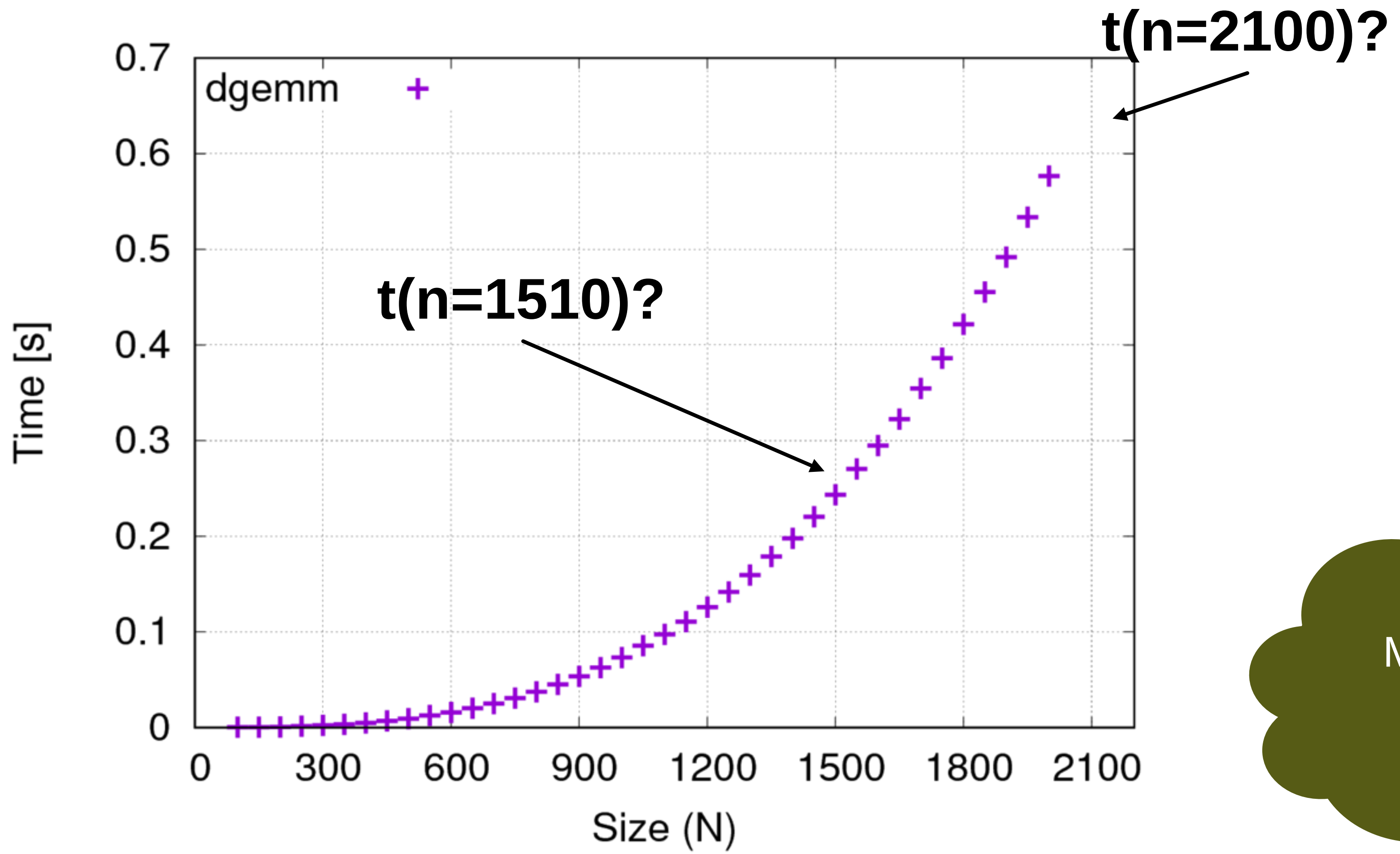
Simplifying Measuring and Reporting: LibSciBench

```c
#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



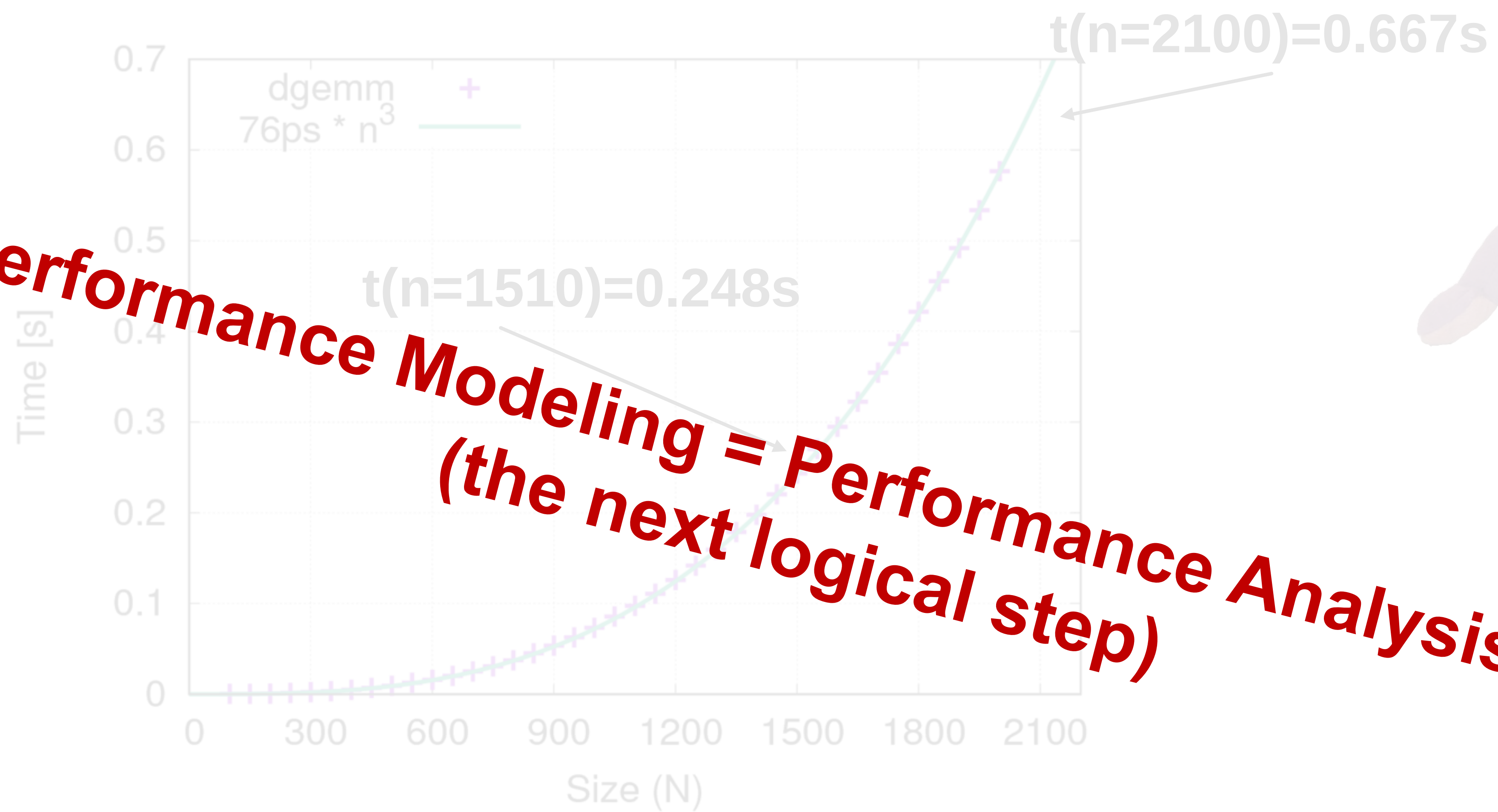
We have the (statistically sound) data, now what?



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

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

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

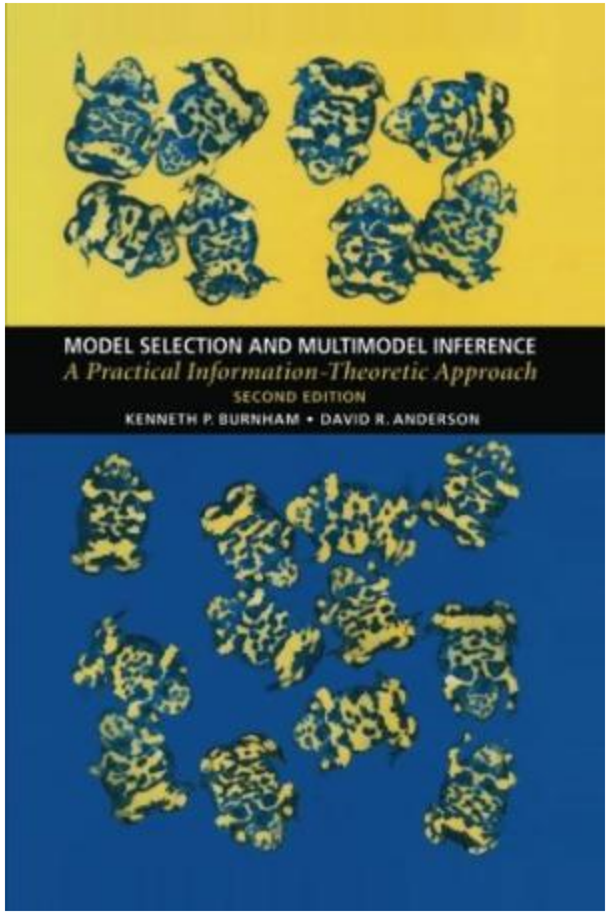


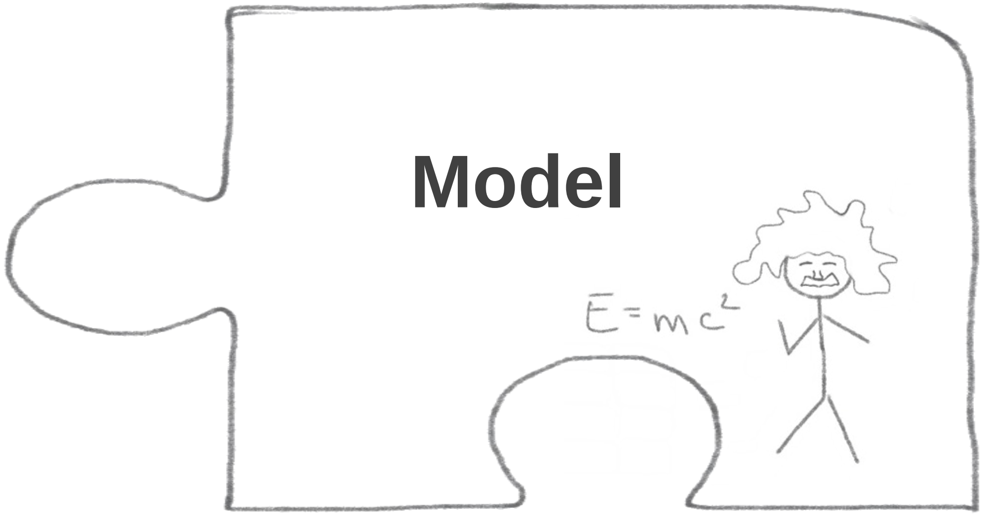
Performance Modeling = Performance Analysis v 2.0
(the next logical step)

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

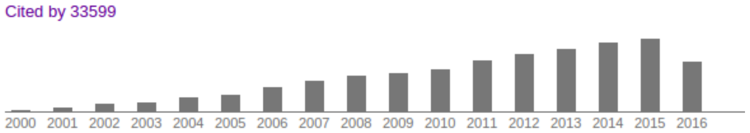
TH, W. Gropp, M. Snir, W. Kramer: Performance Modeling for Systematic Performance Tuning, IEEE/ACM SC11

Part II: Model



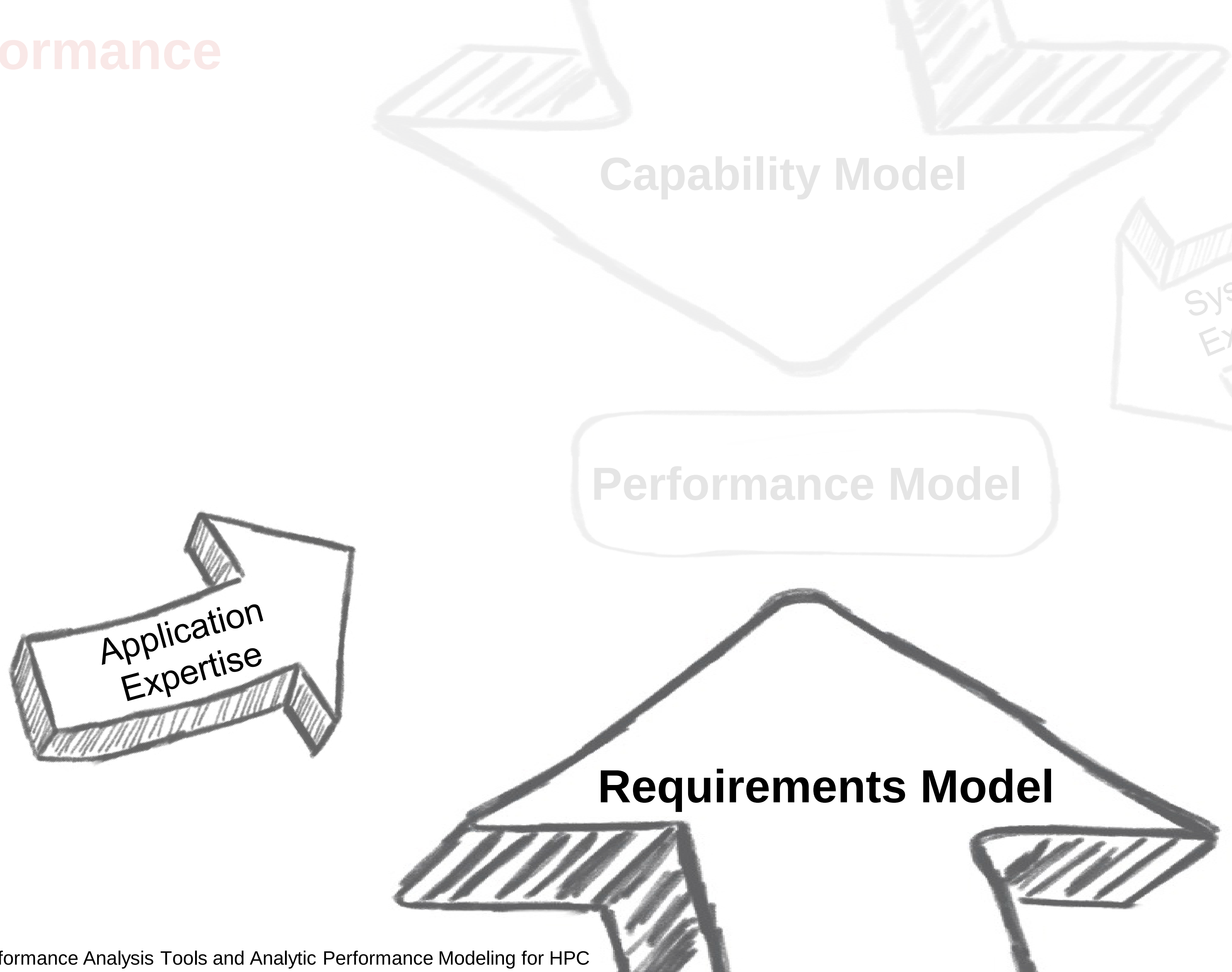


Burnham, Anderson: *"A model is a simplification or approximation of reality and hence will not reflect all of reality. ... Box noted that "all models are wrong, but some are useful." While a model can never be "truth," a model might be ranked from very useful, to useful, to somewhat useful to, finally,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

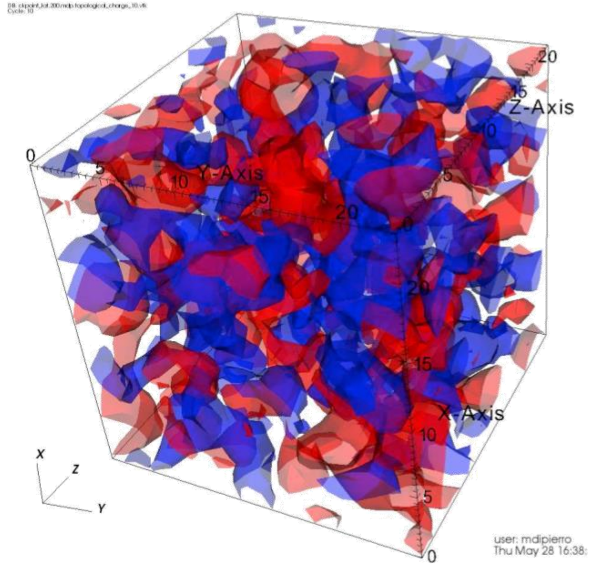
Describe application kernels

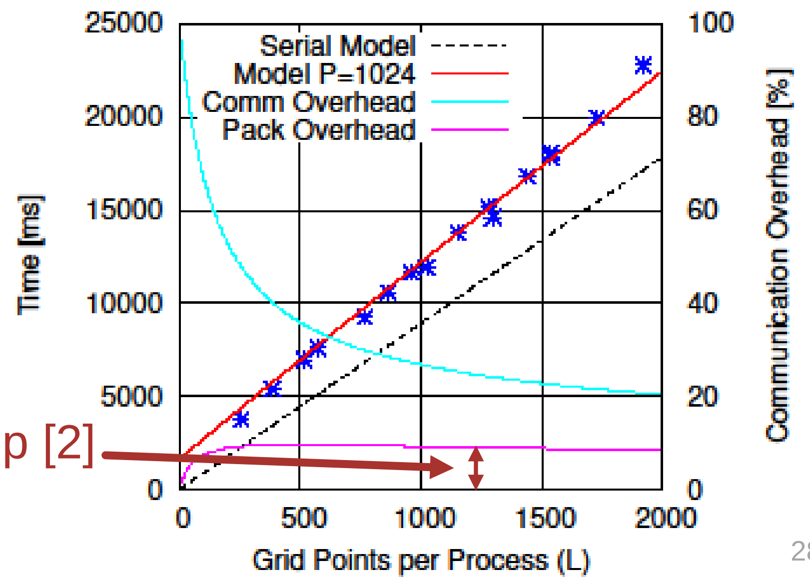
Communication pattern

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

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

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

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

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**

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

$n \in \mathbb{N}$

$i_k \in I$

$j_k \in J$

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

$n = 2$

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

$J = \{0, 1\}$

$c_1 + c_2 \cdot p$

$c_1 + c_2 \cdot p^2$

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

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

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

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

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

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

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

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

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

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

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

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

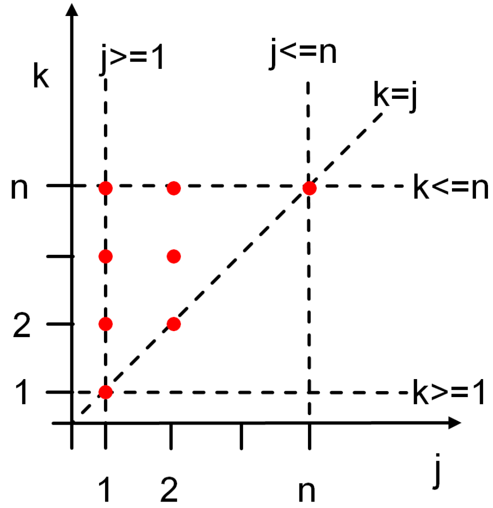
$c_1 \cdot p^2 + c_2 \cdot p^2 \cdot \log(p)$

[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);
```

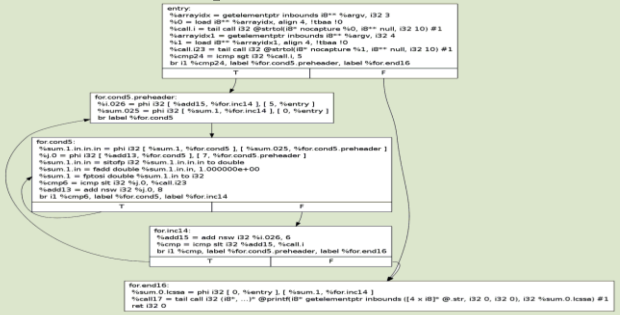


$$N = (n+1)\log_2 n - n + 2$$

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

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

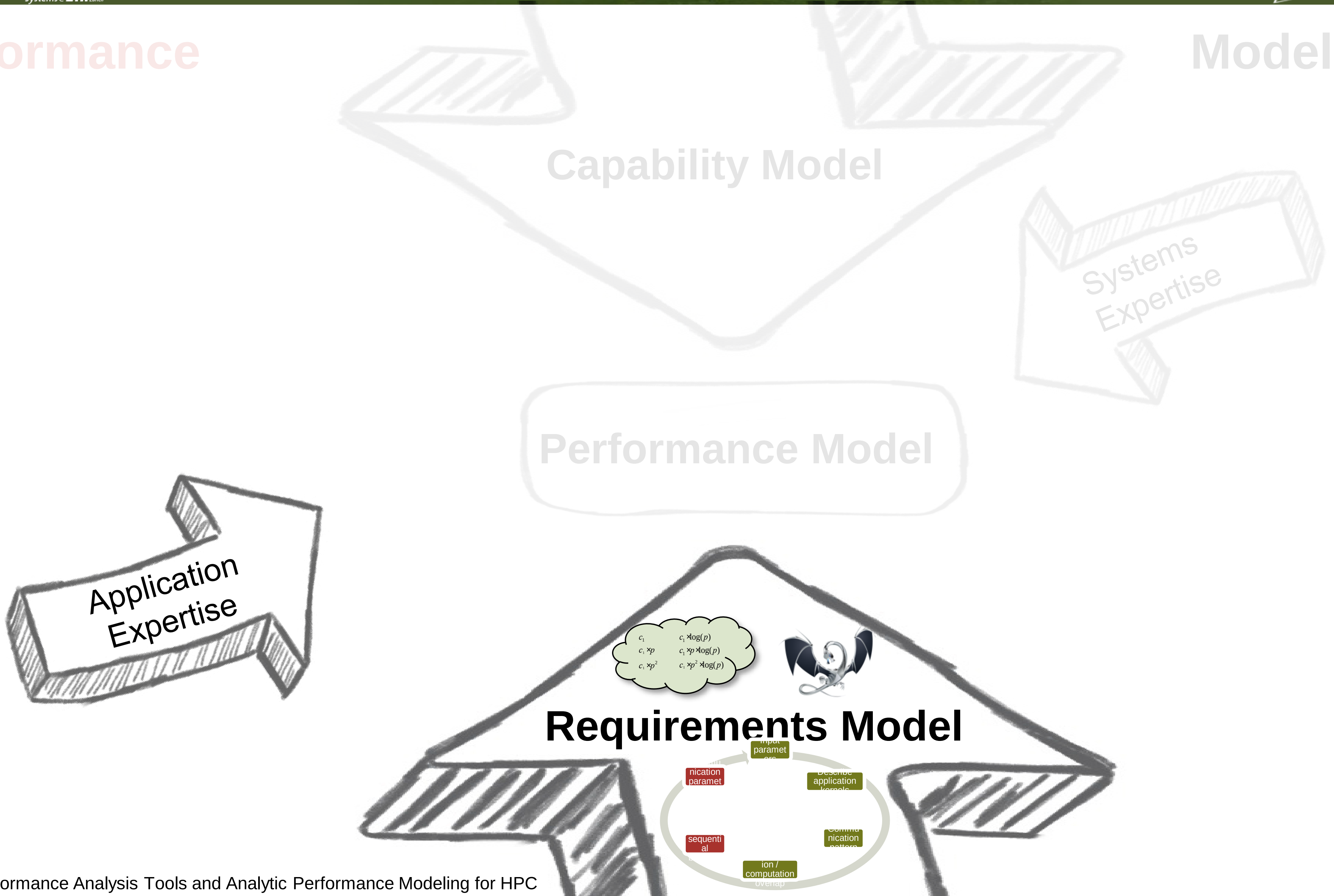
Requirements Models

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

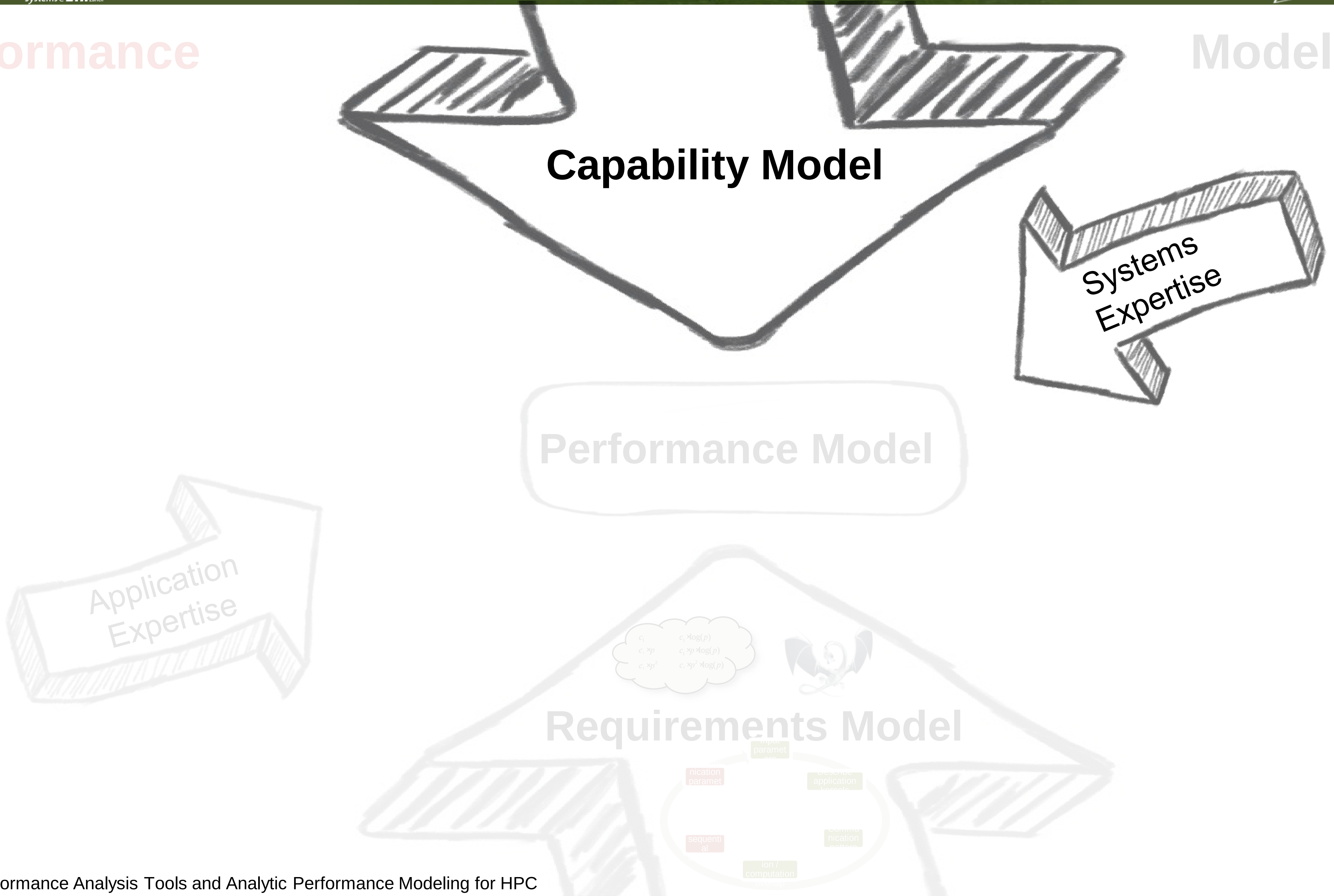
$$D = N\Big|_{p\to\infty}$$

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

Performance Modeling



Performance Modeling



Capability models for network communication

The LogP model family and the LogGOPS model [1]


A new parallel machine model reflects the critical technology trends underlying parallel computers

A PRACTICAL MODEL of PARALLEL COMPUTATION

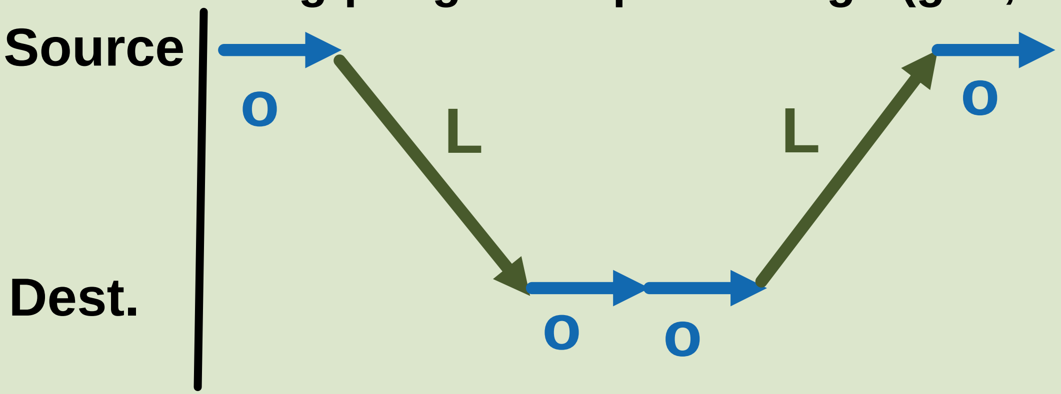
LogP

UR GOAL IS TO DEVELOP A MODEL OF PARALLEL COMPUTATION THAT WILL serve as a basis for the design and analysis of fast, portable parallel algorithms, such as algorithms that can be implemented effectively on a wide variety of current and future parallel machines. If we look at the body of parallel algorithms developed under current parallel models, many are impractical because they exploit artificial factors not present in any rea-

PRAM consists of a collection of processors which compute synchronously in parallel and communicate with a global random access

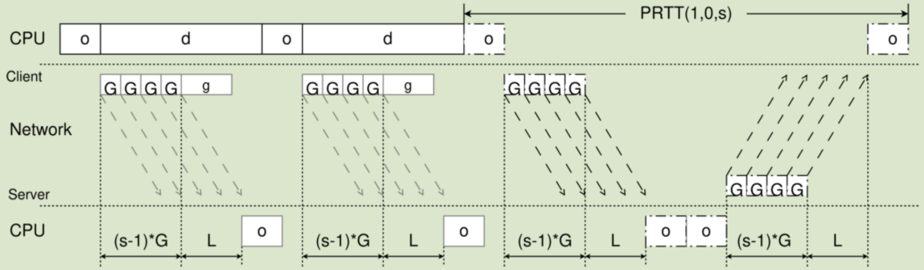
David E. Culler, Richard M. Karp, David Patterson, Abhijit Sahay, Eunice E. Santos, Klaus Erik Schauser, Ramesh Subramonian, and Thorsten von Eicken

Ping-pong in simplified LogP (g<o, P=2)



Finding LogGOPS parameters

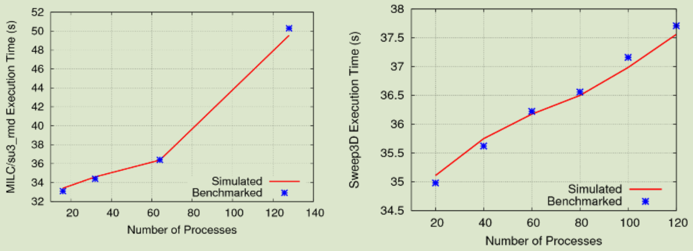
Netgauge [2], model from first principles, fit to data using special kernels



Large scale LogGOPS Simulation

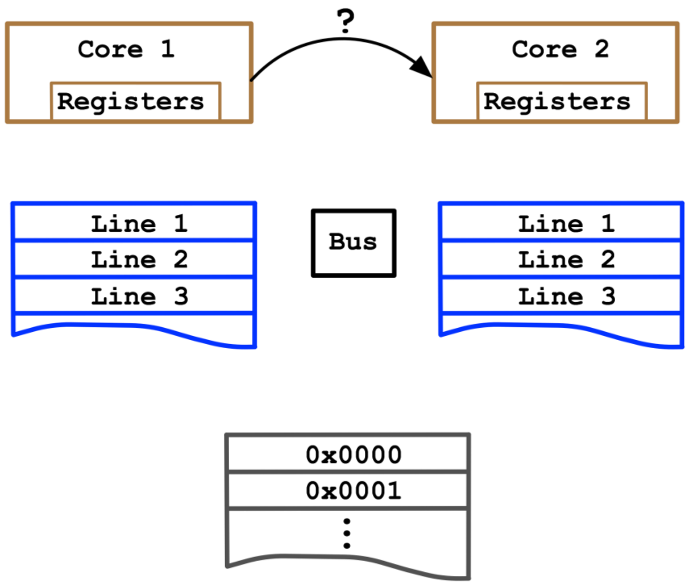
LogGOPSim [1], simulates LogGOPS with 10 million MPI ranks

<5% error

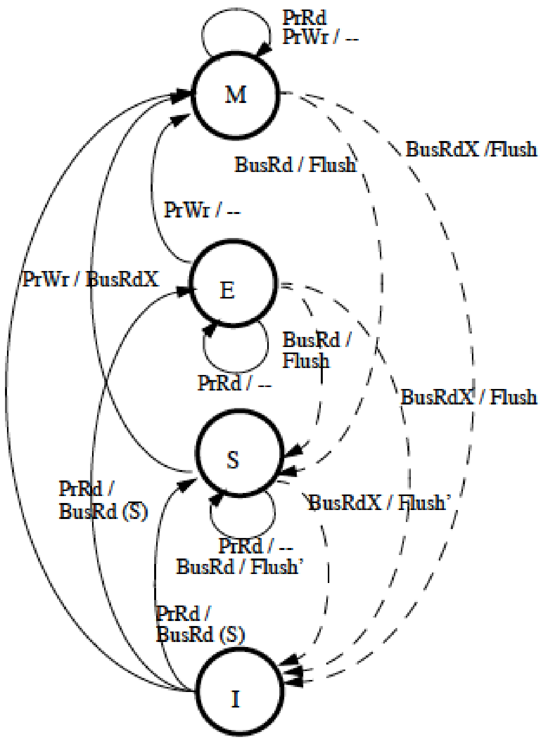


[1]: TH, T. Schneider and A. Lumsdaine: LogGOPSim - Simulating Large-Scale Applications in the LogGOPS Model, LSAP 2010, https://spcl.inf.ethz.ch/Research/Performance/LogGOPSim/
[2]: TH, T. Mehlan, A. Lumsdaine and W. Rehm: Netgauge: A Network Performance Measurement Framework, HPCC 2007, https://spcl.inf.ethz.ch/Research/Performance/Netgauge/

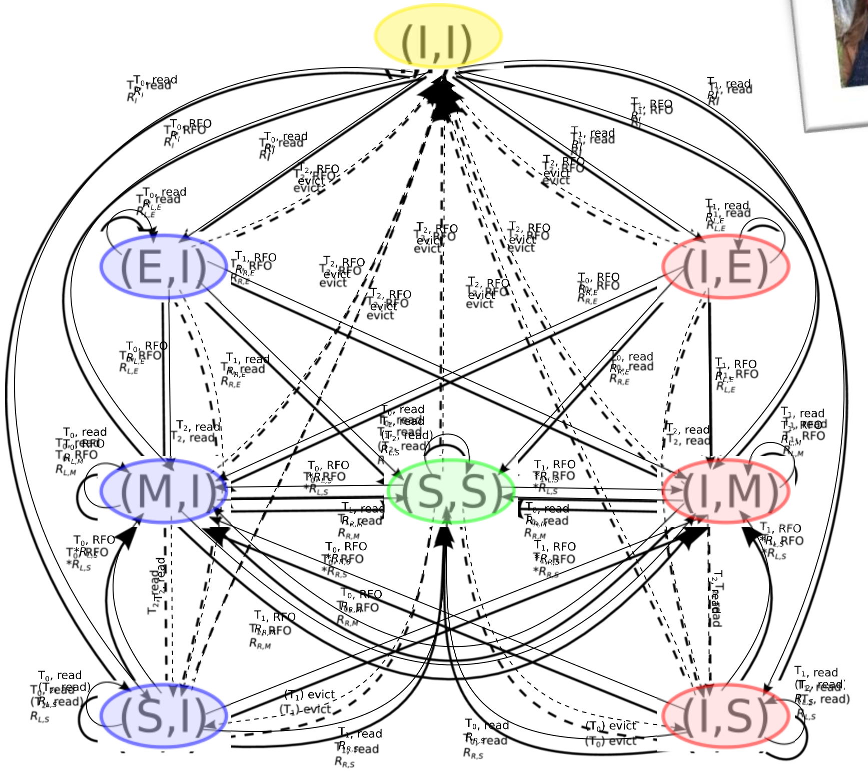
Capability models for cache-to-cache communication

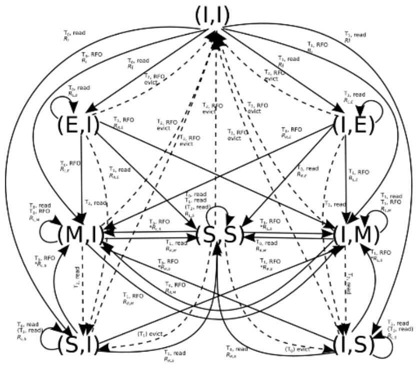


X

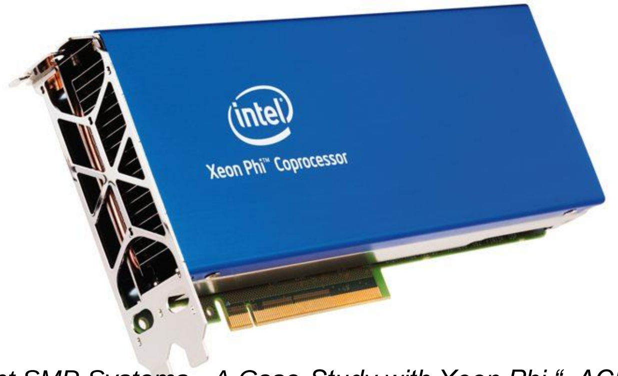


=

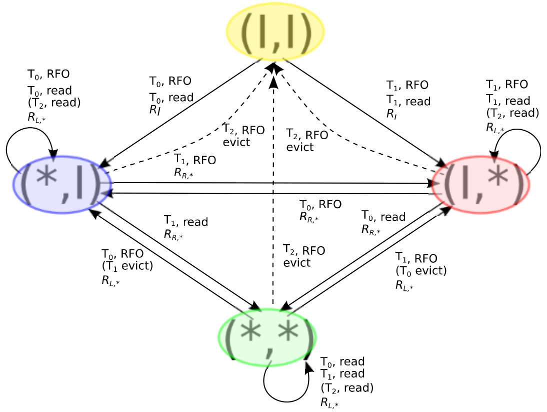




|



=



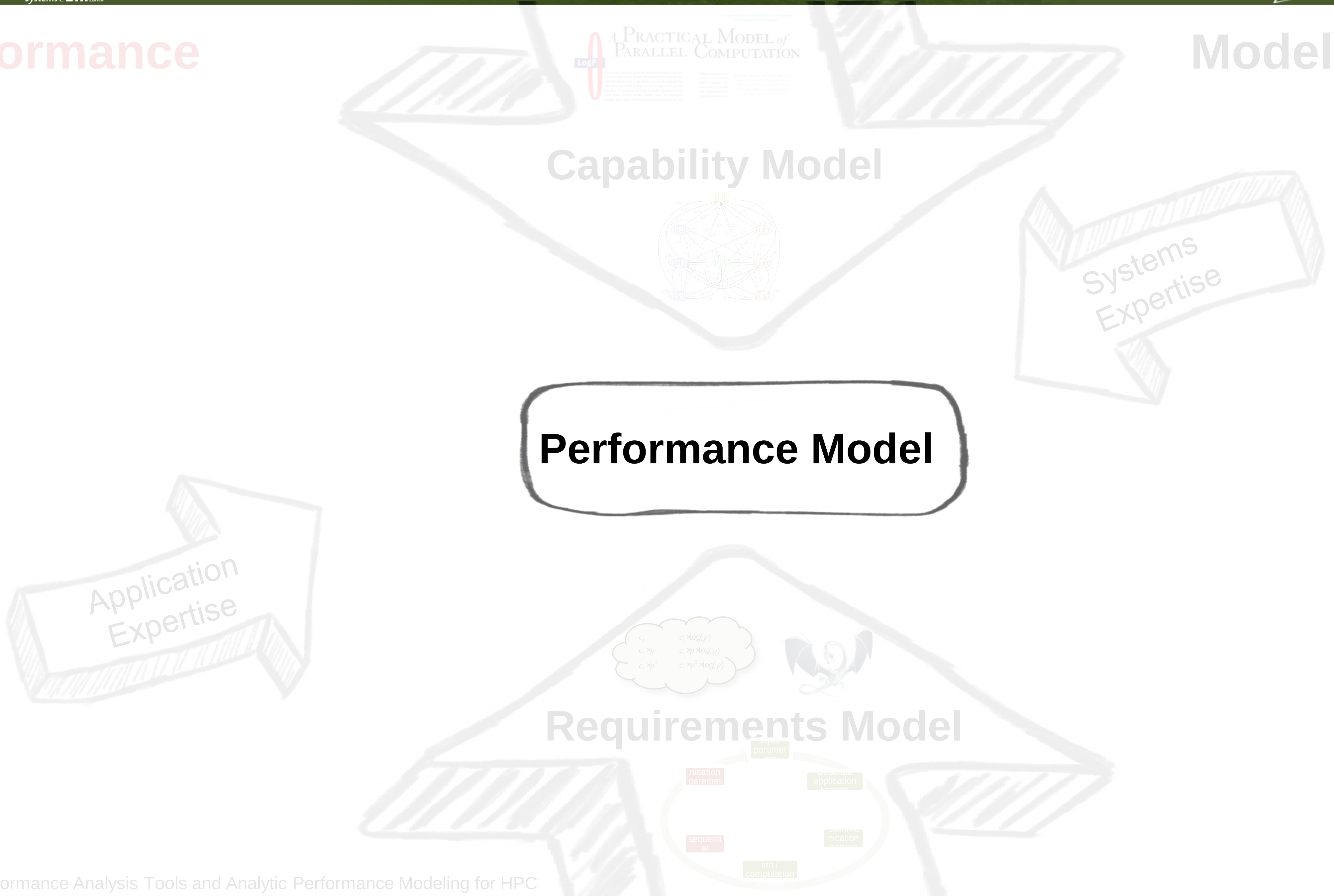
Invalid read R_I = 278 ns

Local read: R_L= 8.6 ns

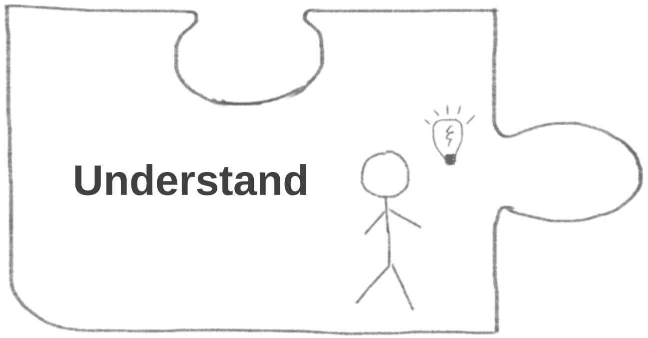
Remote read R_R = 235 ns

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

Performance Modeling



Part III: Understand

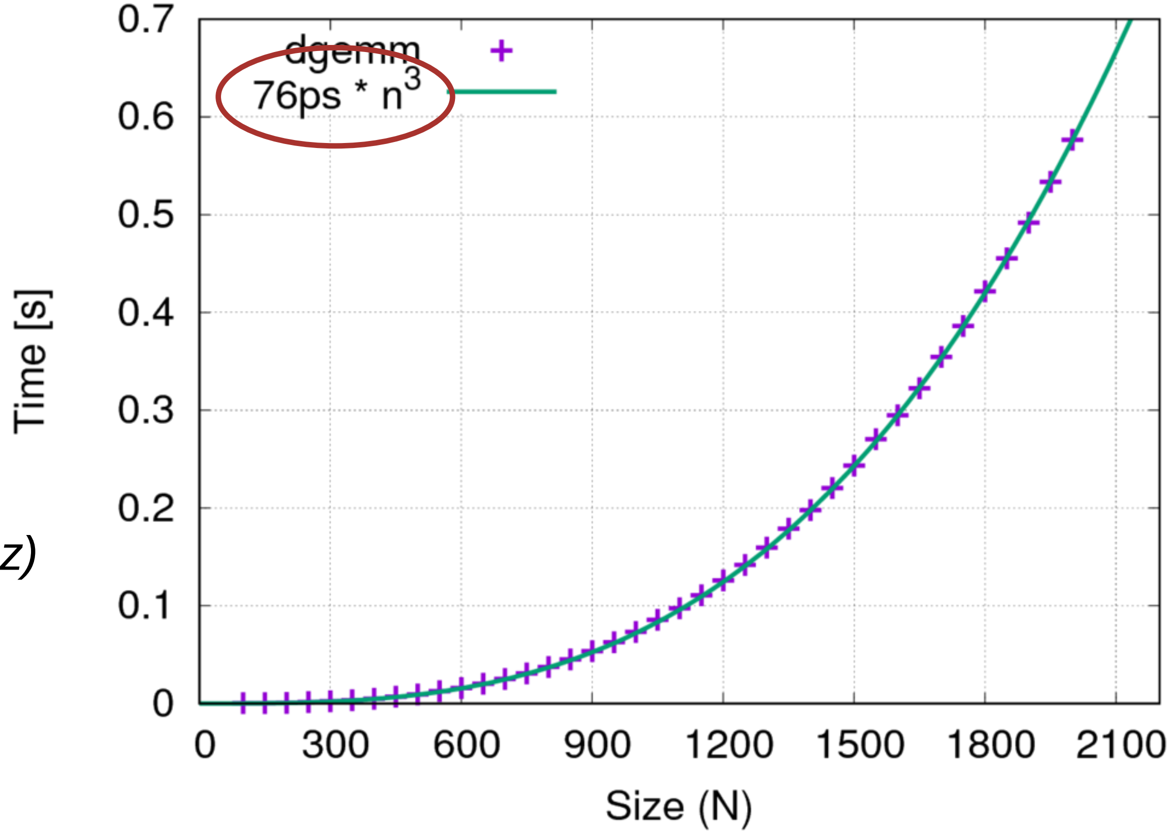


- **Use models to**
  1. Proof optimality of real implementations
     - *Stop optimizing, step back to algorithm level*
  2. Design optimal algorithms or systems in the model
     - *Can lead to non-intuitive designs*

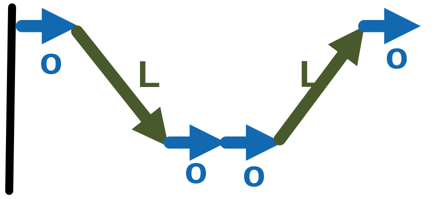
- **Proof optimality of matrix multiplication**
  - Intuition: flop rate is the bottleneck
  - $t(n) = 76ps * n^3$
  - **Flop rate:** $R = 2flop * n^3/(76ps * n^3)$ = **27.78 Gflop/s**
  - **Flop peak**: 3.864 GHz * 8 flops = **30.912 Gflop/s**

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

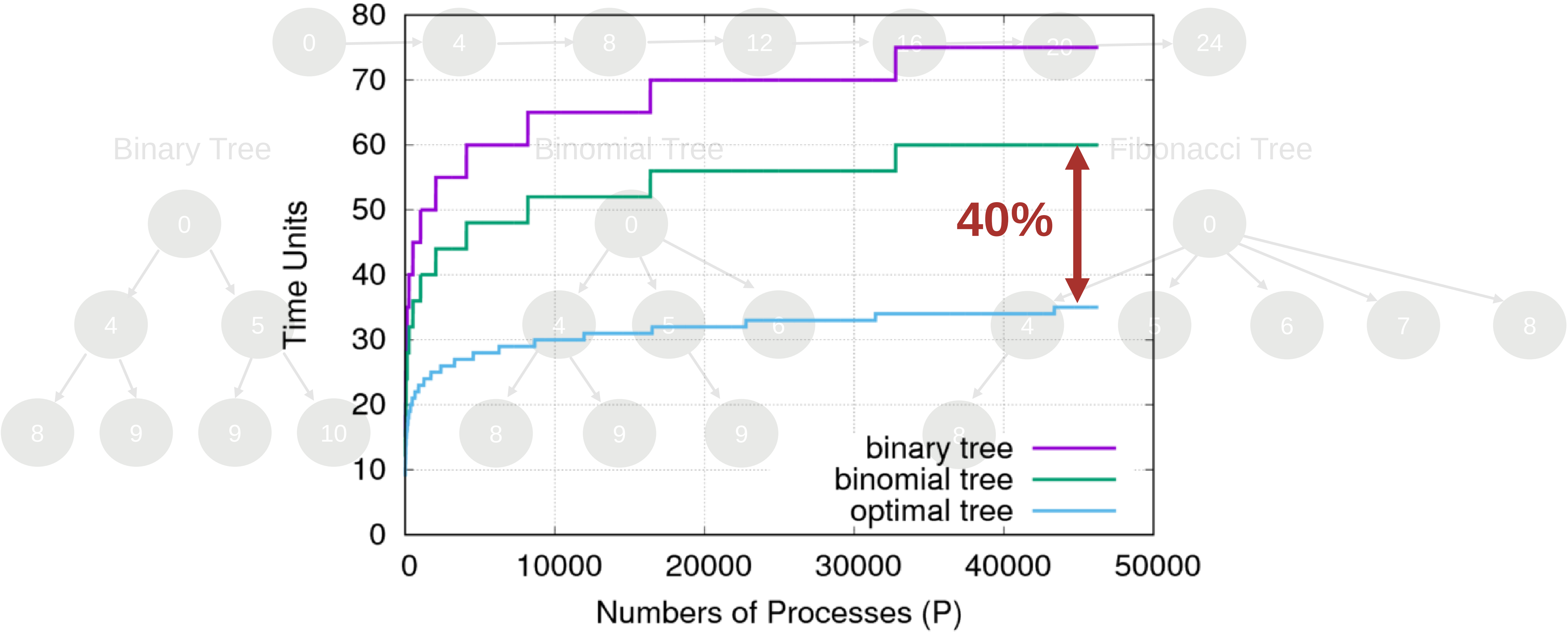
- **Gets more complex quickly**
  - Imagine sparse matrix-vector



2) Design optimal algorithms – small broadcast in LogP

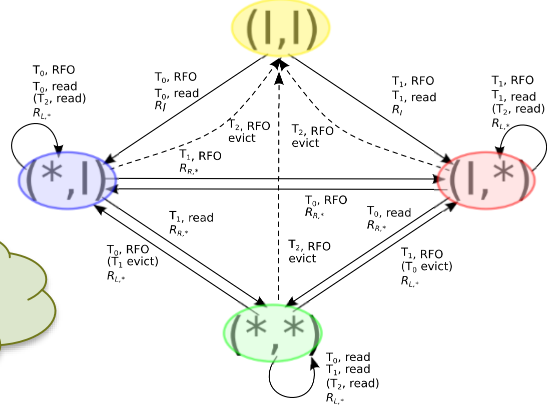


L=2, o=1, P=7

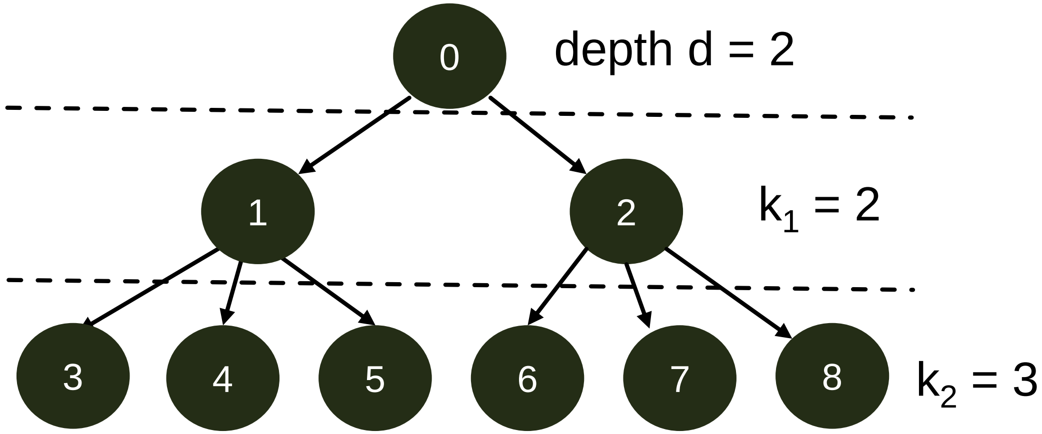


TH, D. Moor: Energy, Memory, and Runtime Tradeoffs for Implementing Collective Communication Operations, JSFI 2015

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

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

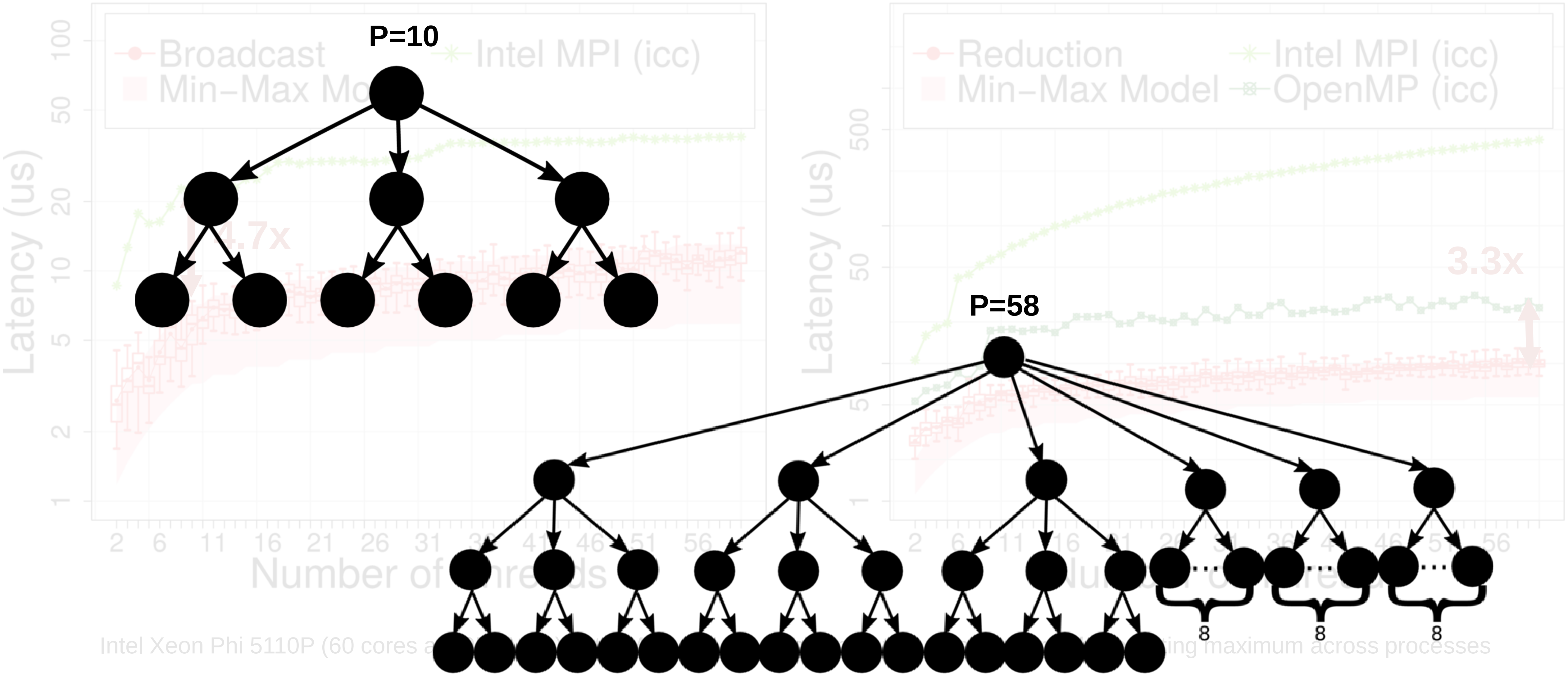
$$\mathcal{T}_{sbcast} = \min_{d,k_i} \left(\mathcal{T}_{fw} + \sum_{i=1}^{d} (c \cdot k_i + b) + \sum_{i=1}^{d} \mathcal{T}_{nb}(k_i + 1) \right)$$

Reached threads

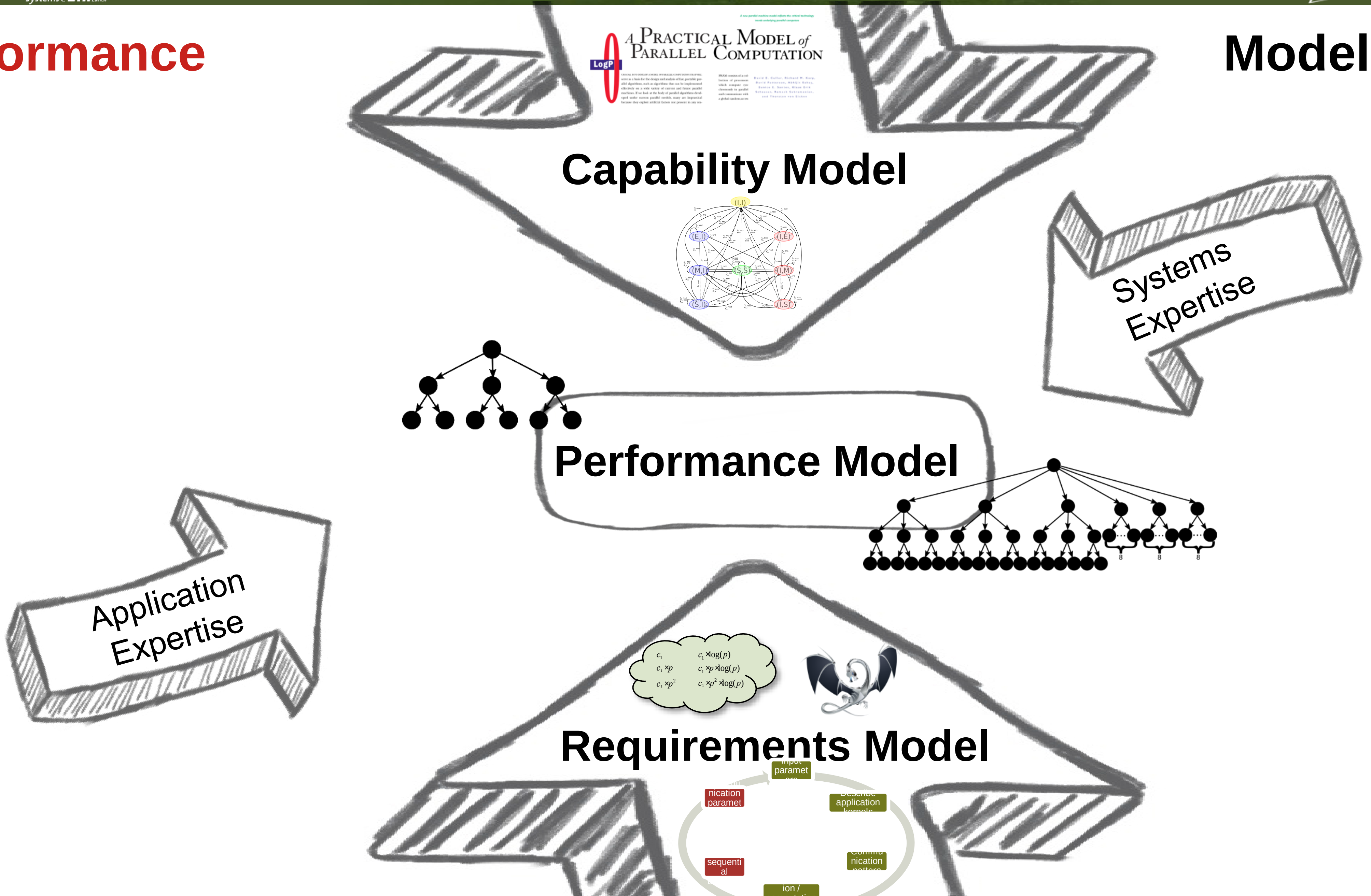
$$N \leq 1 + \sum_{i=1}^{d} \prod_{j=1}^{i} k_j, \quad \forall i < j, k_i \leq k_j$$

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

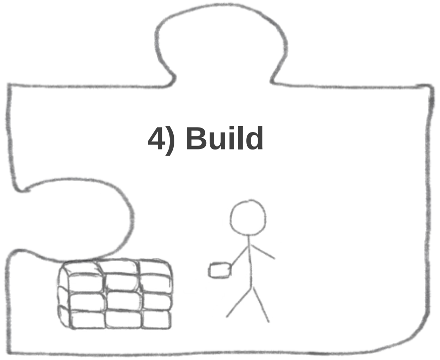
Measured results – small broadcast and reduction



Performance Modeling

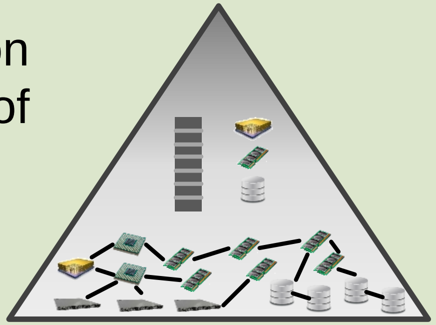


Part IV: Build



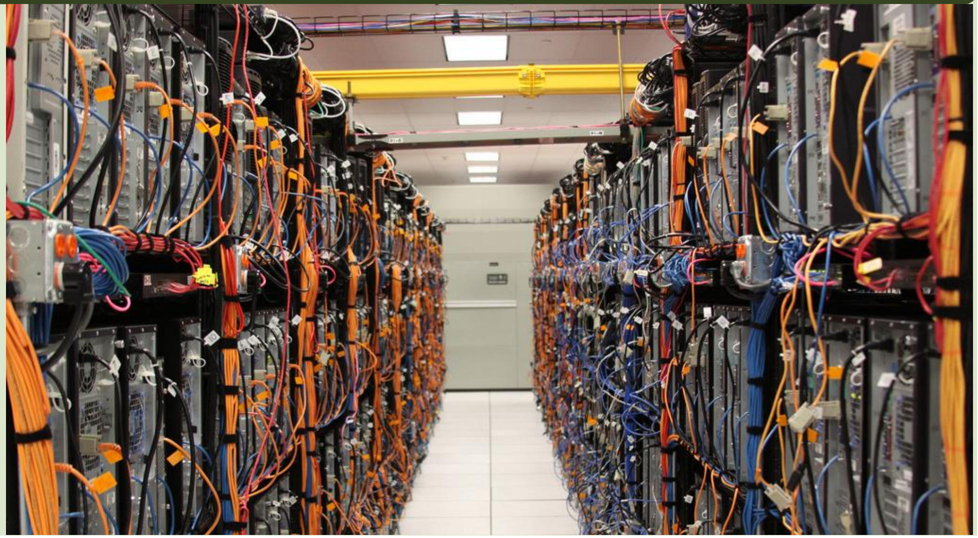
Abstraction is Key

- Enables to focus on essential aspects of a system



Case study: Network Topologies

- **Observe**: optimize for cost, maintain performance:
  - router radix, number of cables, number of routers → cost
  - number of endpoints, latency, global bandwidth → capabilities
- **Model**: system as graph
- **Understand**: degree-diameter graphs
- **Build**: Slim Fly topology
- Result: non-trivial topology that is 1/3rd cheaper than all existing



M. Besta, TH: Slim Fly: A Cost Effective Low-Diameter Network Topology, ACM/IEEE Supercomputing 2014, SC14

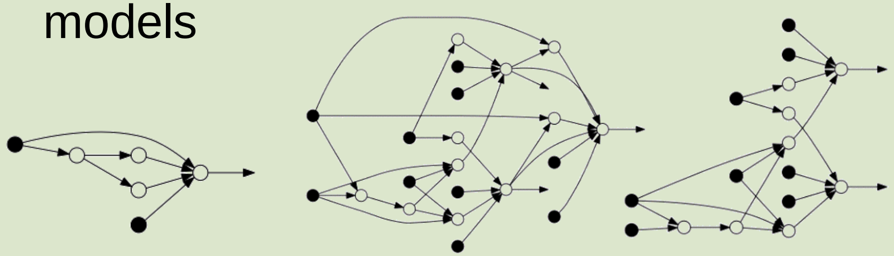
How to continue from here?

Transformation System

- User-supported, compile- and run-time

Parallel Language

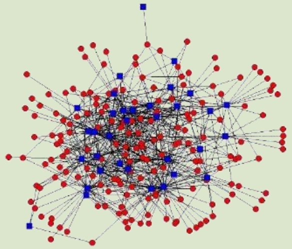
- Data-centric, explicit requirements models



DAPP

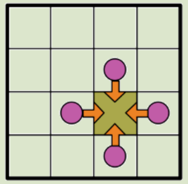
European Research Council
Established by the European Commission
Supporting top researchers from anywhere in the world

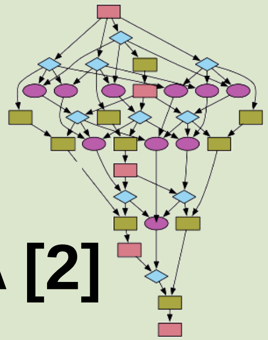
Performance-transparent Platforms



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

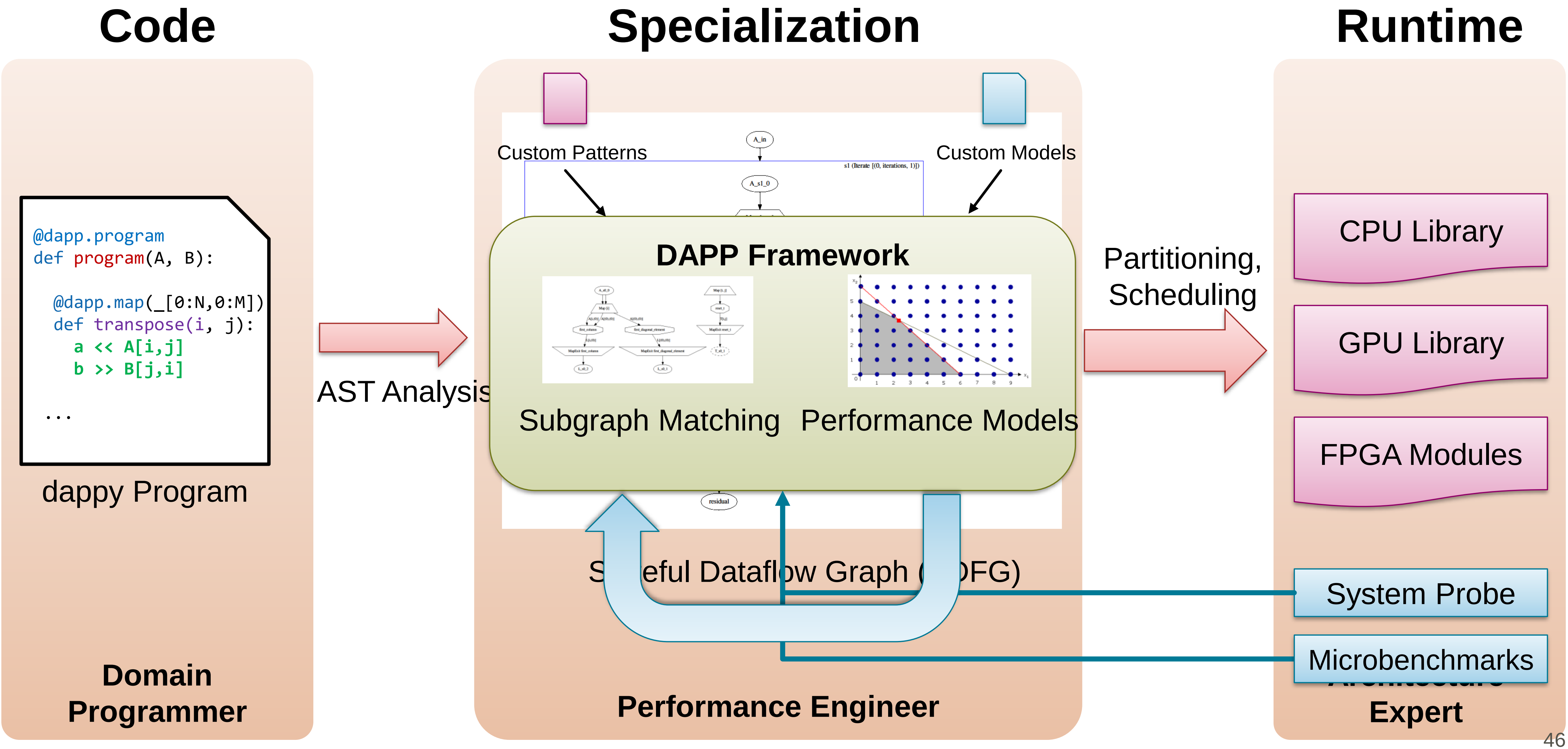
DAPPy – Data-centric Parallel Programming for Python

DAPP

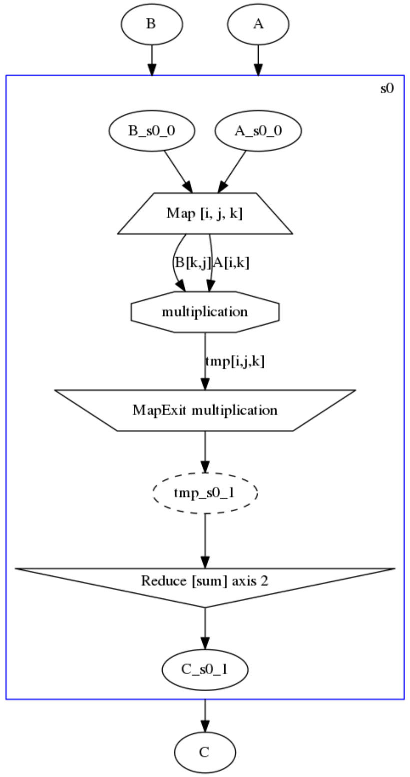
- **Memory access decoupled from computation**
- **Programs are composed of Tasklets and Memlets**
  - Tasklets wrapped by simple primitives: Map, Iterate, Reduce
  - Hide communication, caching and data-movement
- **Easy-to-integrate Python programming interface**
- **Graph-based compilation pipeline**

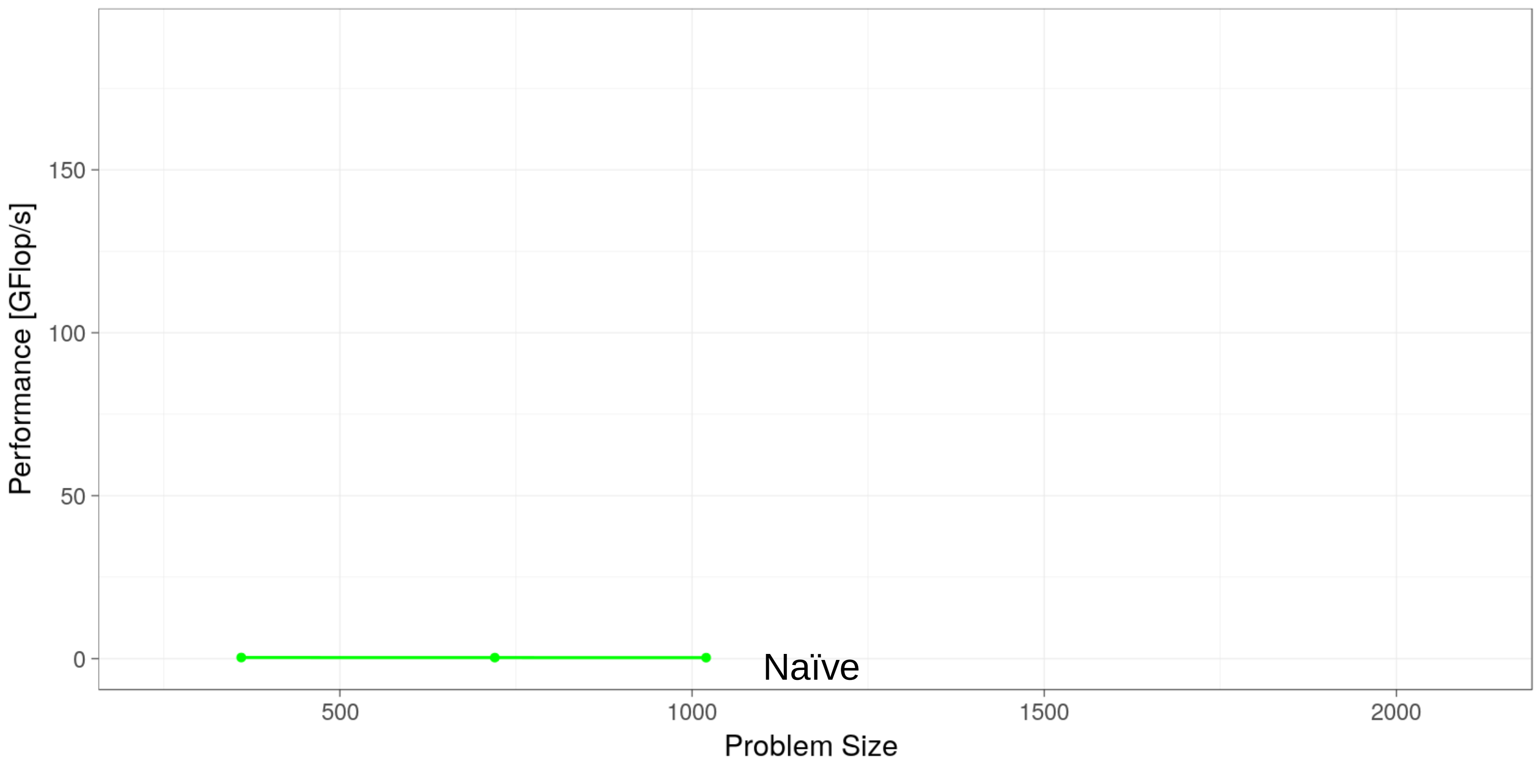
```
@dapp.program
def gemm(A, B, C):
    # local definitions

    @dapp.map(_[0:M, 0:K, 0:N])
    def multiplication(i, j, k):
        in_A << A[i,k]
        in_B << B[k,j]
        out  >> tmp[i,j,k]

        out = in_A * in_B

    @dapp.reduce(tmp, C, axis=2)
    def sum(a,b):
        return a+b
```

DAPPy Compilation Infrastructure



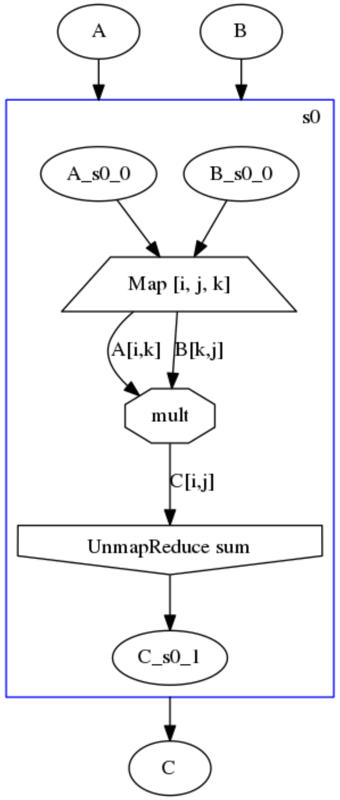
Performance

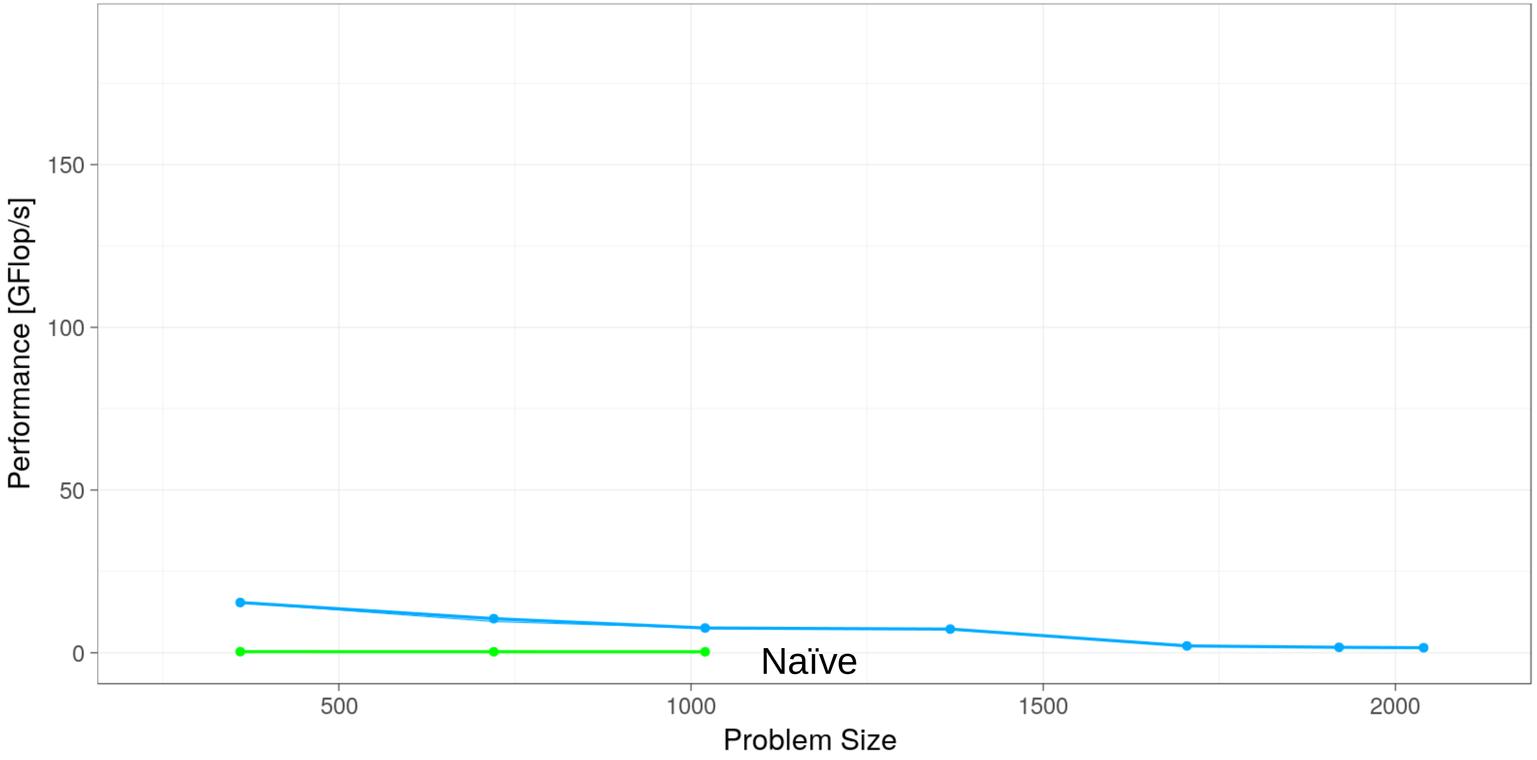




SDFG

Performance

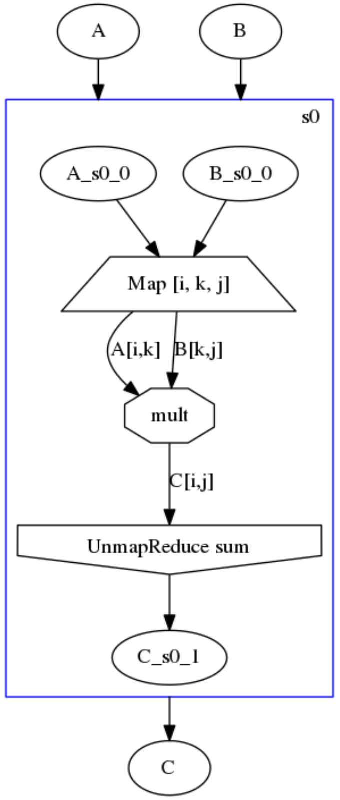


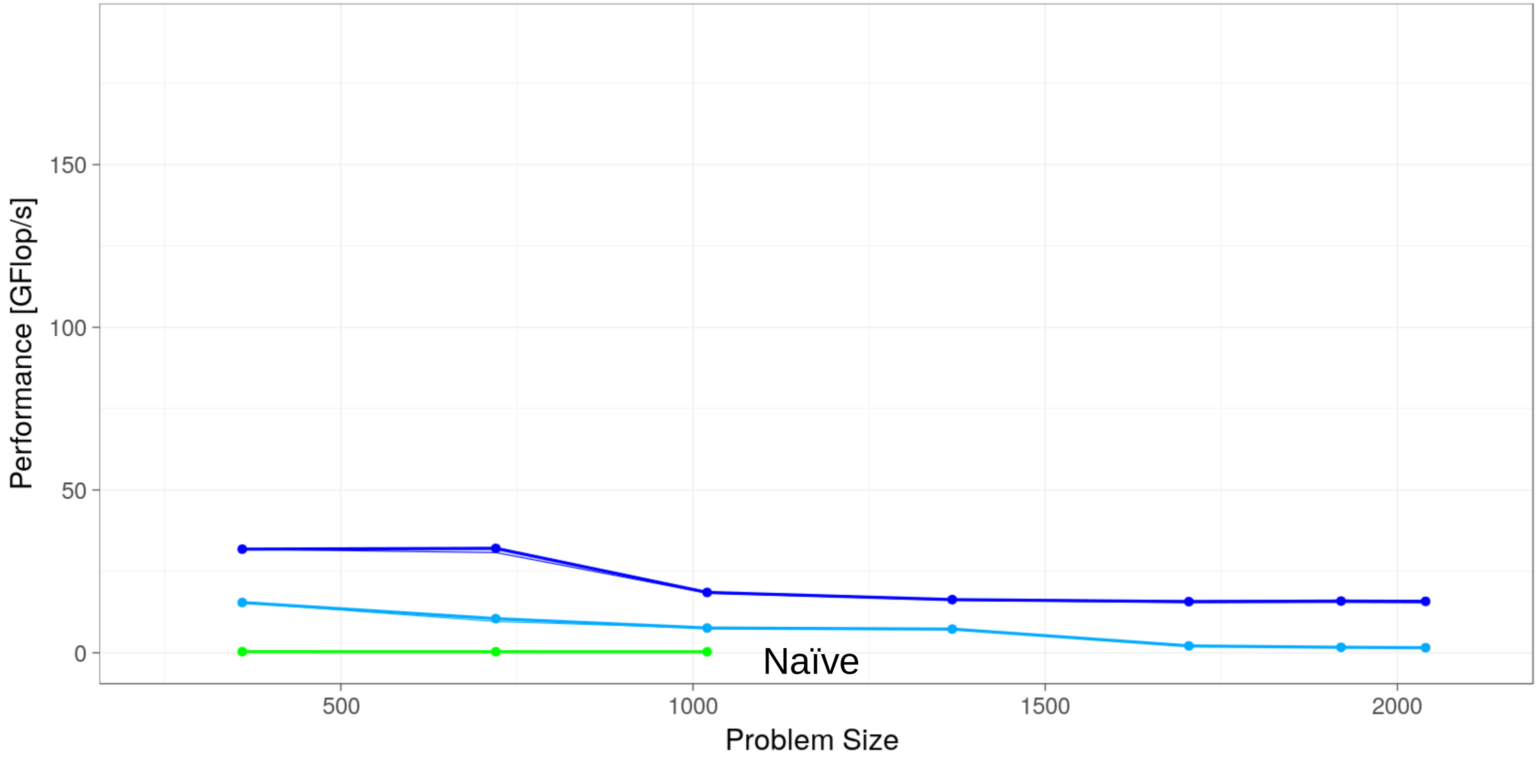


MapReduceFusion

SDFG

Performance

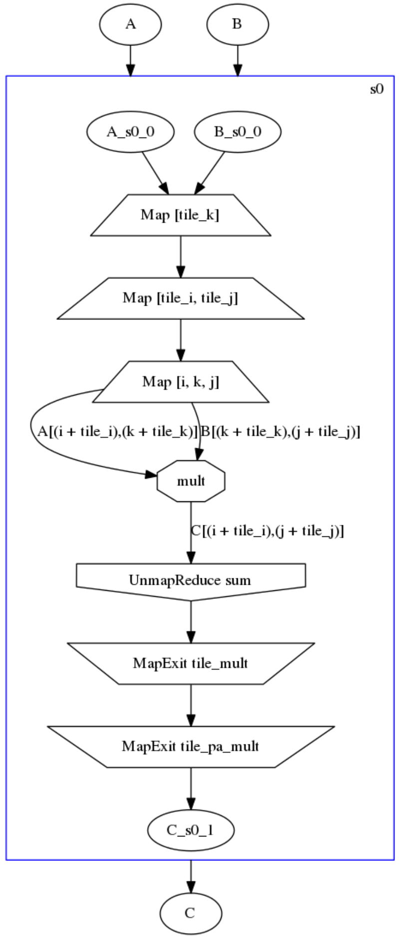


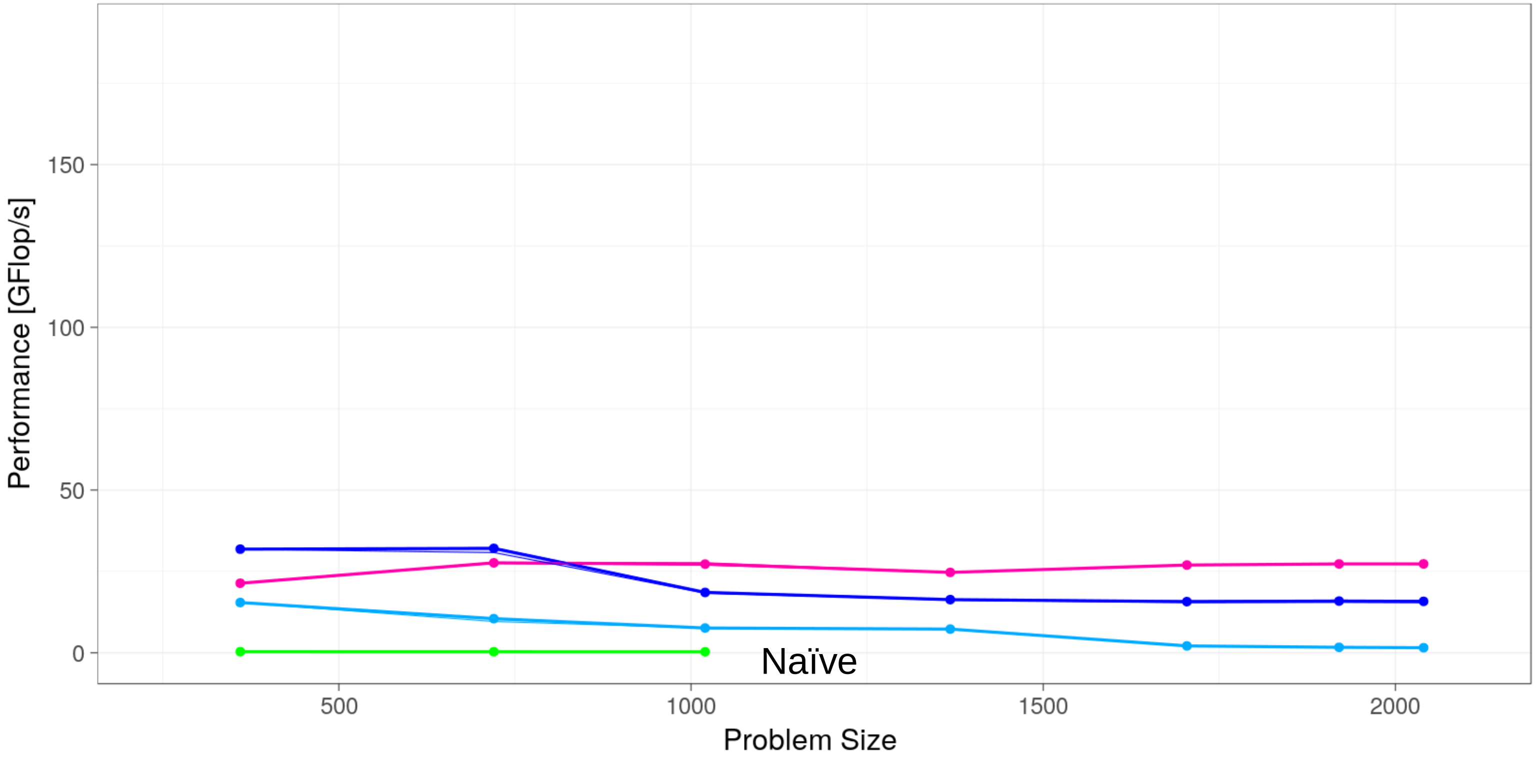


LoopReorder
MapReduceFusion

SDFG

Performance

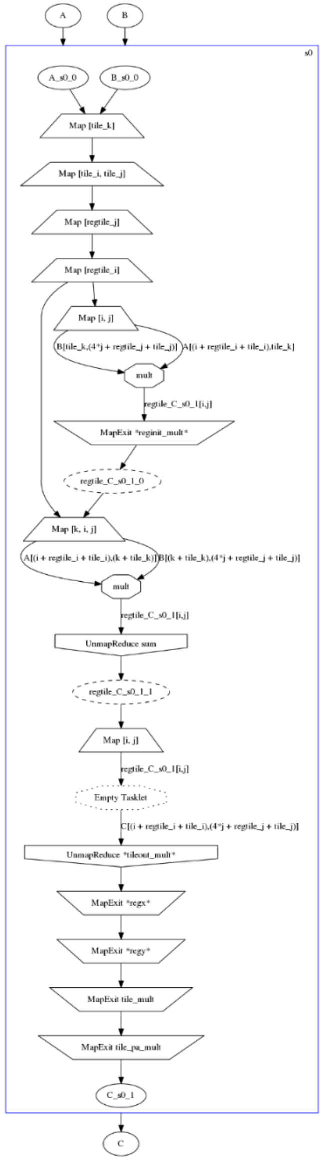




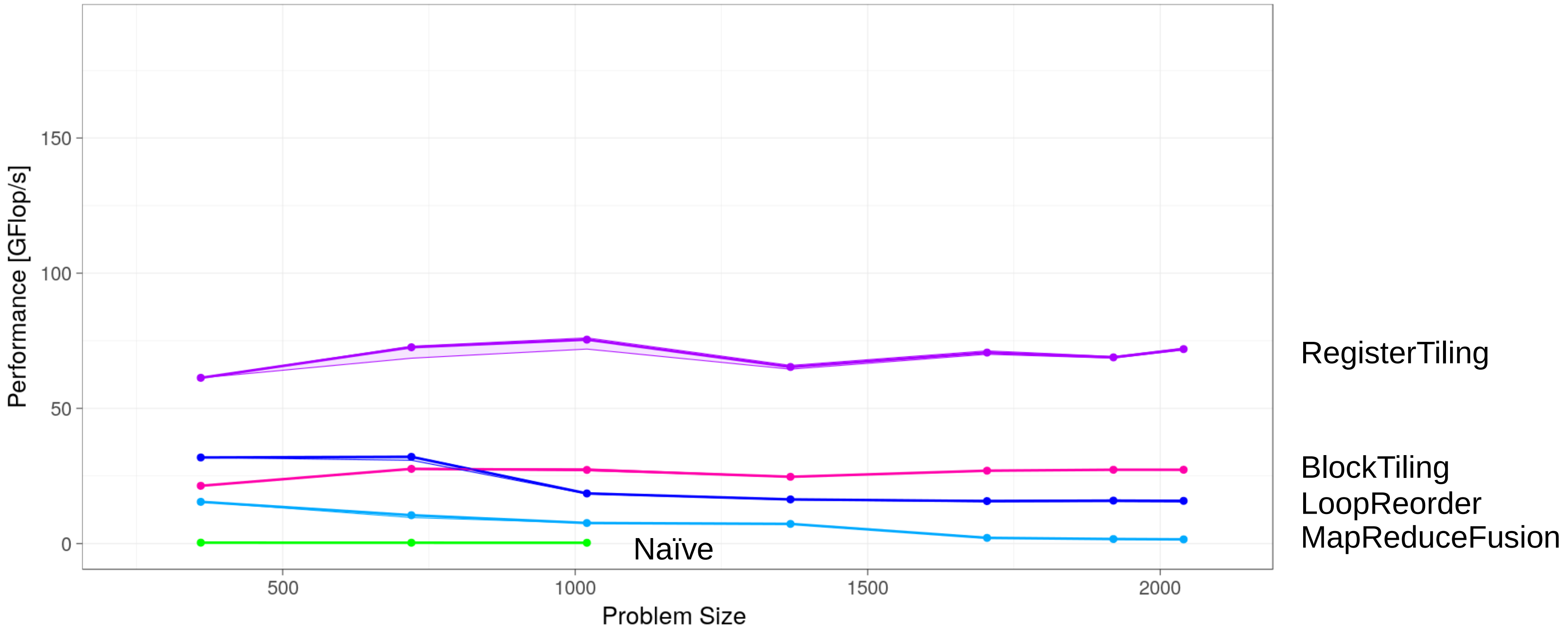
BlockTiling
LoopReorder
MapReduceFusion

SDFG

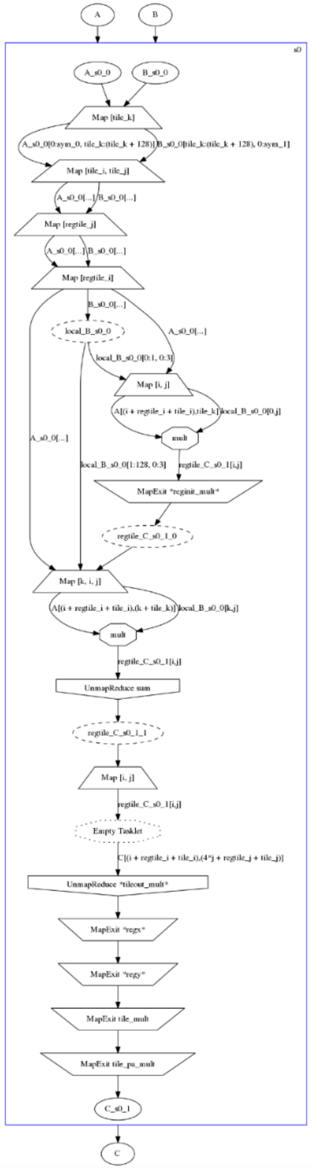
Performance



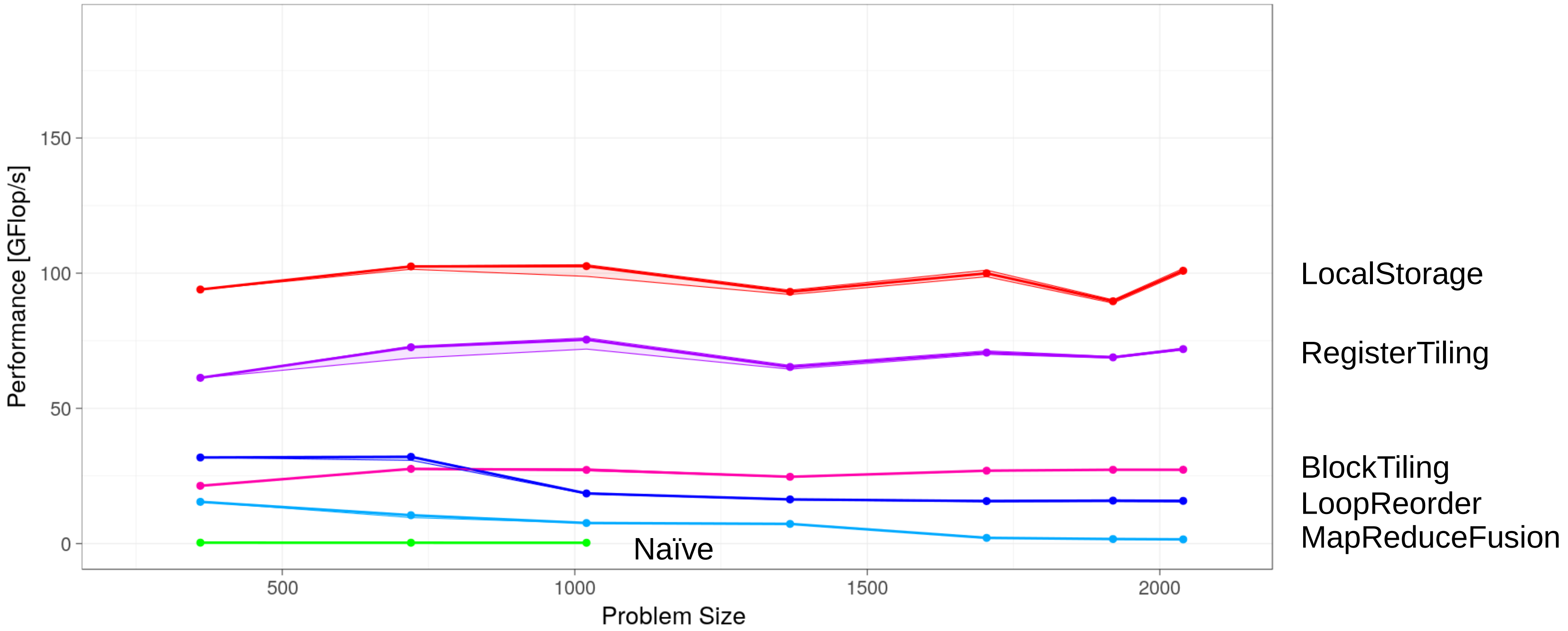
SDFG



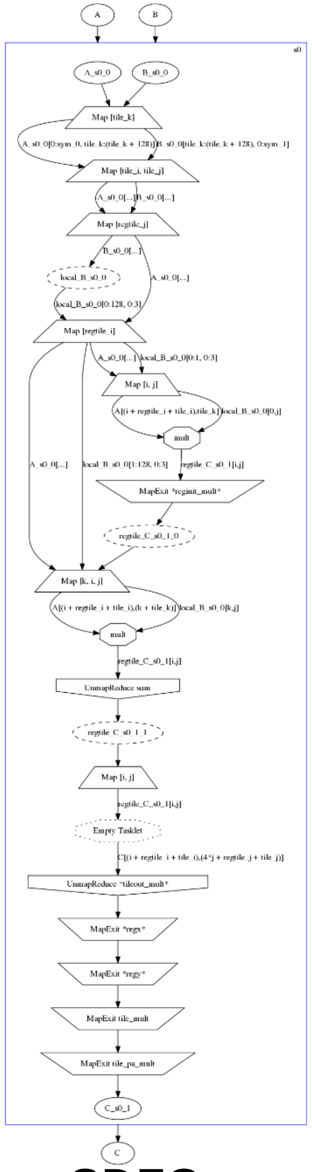
Performance



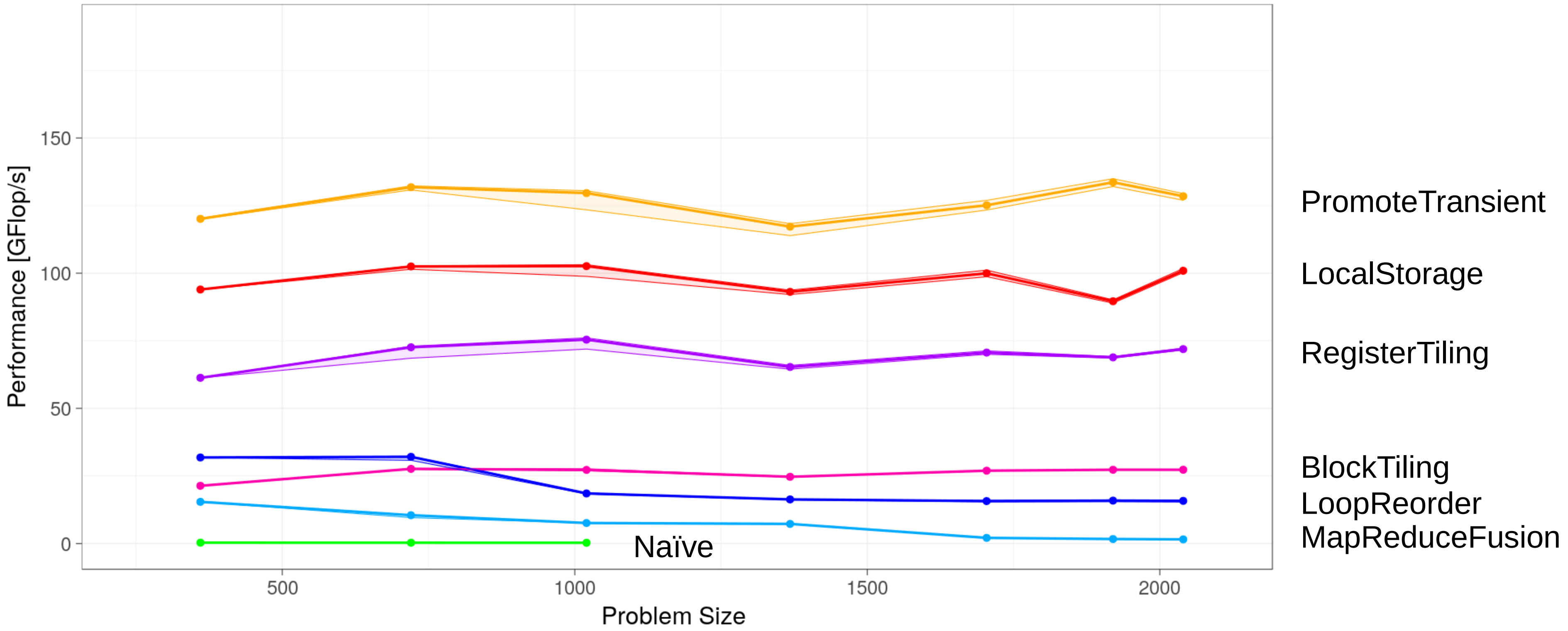
SDFG



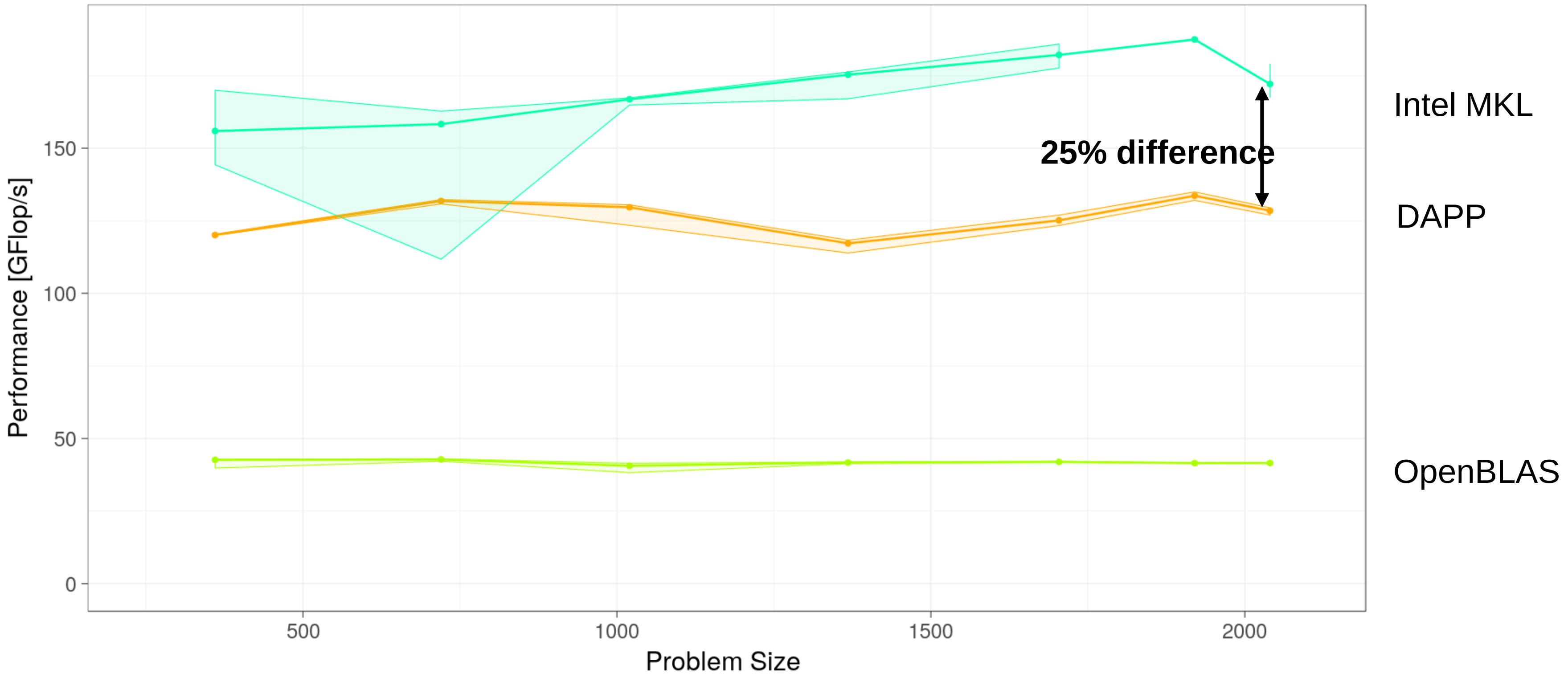
Performance



SDFG



Performance



Generated DAPP/C++ Code (Excerpt)

```
void _program_gemm(int sym_0, int sym_1, int sym_2, double * __restrict__ A, double * __restrict__ B, double * __restrict__ C) {
    // State s0
    for (int tile_k = 0; tile_k < sym_2; tile_k += 128) {
        #pragma omp parallel for
        for (int tile_i = 0; tile_i < sym_0; tile_i += 64) {
            for (int tile_j = 0; tile_j < sym_1; tile_j += 240) {
                for (int regtile_j = 0; regtile_j < (min(240, sym_1 - tile_j)); regtile_j += 12) {

                    vec<double, 4> local_B_s0_0[128 * 3];
                    Global2Stack_2D_FixedWidth<double, 4, 3>(&B[tile_k*sym_1 + (regtile_j + tile_j)], sym_1,
                                                             local_B_s0_0, min(sym_2 - tile_k, 128));

                    for (int regtile_i = 0; regtile_i < (min(64, sym_0 - tile_i)); regtile_i += 4) {
                        vec<double, 4> regtile_C_s0_1[4 * 3];
                        for (int i = 0; i < 4; i += 1) {
                            for (int j = 0; j < 3; j += 1) {
                                double in_A = A[(i + regtile_i + tile_i)*sym_2 + tile_k];
                                vec<double, 4> in_B = local_B_s0_0[0*3 + j];
                                // Tasklet code (mult)
                                auto out = (in_A * in_B);
                                regtile_C_s0_1[i*3 + j] = out;
                            }
                        }
                        for (int k = 1; k < (min(128, sym_2 - tile_k)); k += 1) {
                        // ...
```

Backup