

# Performance modeling for domain scientists

*with applications to CFD*

Raphaël PONCET  
CMLA, ENS Cachan

*Ecole thématique Maths-Info-HPC*  
*May 12, 2016 – St Germain au Mont d'Or*

# The team

- Joint work with
  - Marie Bechereau (PhD candidate, ENS Cachan)
  - Florian De Vuyst (ENS Cachan)
  - Thibault Gasc (PhD candidate, MDLS, CEA, ENS Cachan)
  - Renaud Motte (CEA DAM)
  - Mathieu Peybernes (CEA DEN)

# Acknowledgements

- Thanks to
  - Thomas Guillet & Philippe Thierry (Intel)
  - Jean-Michel Ghidaglia & Amine Mrabet (ENS Cachan)
  - Daniel Bouche (CEA DAM)
  - Guillaume Colin de Verdiere (CEA DAM)

# Introduction

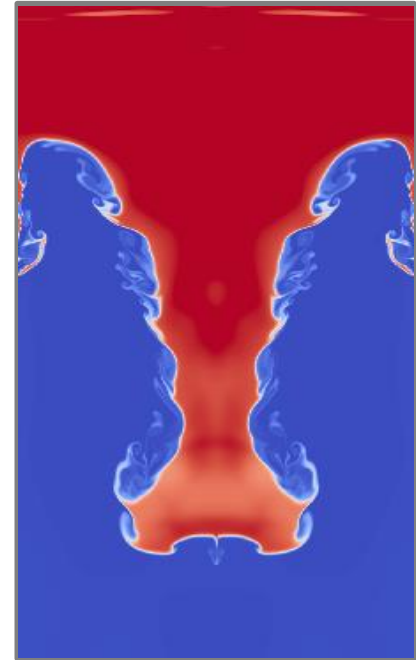
# Our background

- We are domain scientists
  - Applied mathematicians: invent algorithms...
  - ... **and** HPC engineers: optimize algorithms

# Our domain: CFD

$$\frac{\partial}{\partial t} \begin{bmatrix} \rho \\ \rho u_x \\ \rho u_y \\ E \end{bmatrix} + \frac{\partial}{\partial x} \begin{bmatrix} \rho u_x \\ \rho u_x^2 + p \\ \rho u_x u_y \\ (E + p)u_x \end{bmatrix} + \frac{\partial}{\partial y} \begin{bmatrix} \rho u_y \\ \rho u_y u_x \\ \rho u_y^2 + p \\ (E + p)u_y \end{bmatrix} = 0$$

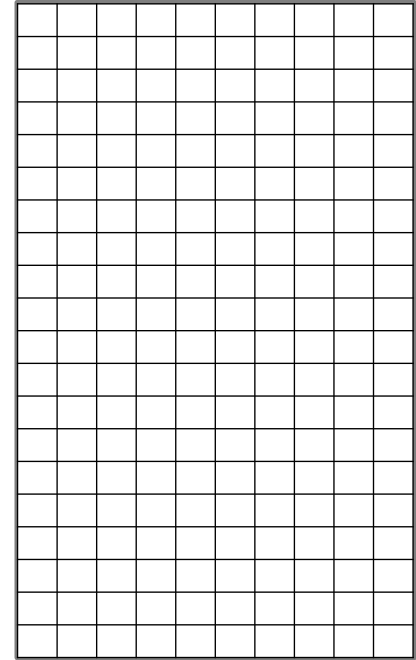
Compressible Euler equations



# Our domain: CFD

$$\frac{\partial}{\partial t} \begin{bmatrix} \rho \\ \rho u_x \\ \rho u_y \\ E \end{bmatrix} + \frac{\partial}{\partial x} \begin{bmatrix} \rho u_x \\ \rho u_x^2 + p \\ \rho u_x u_y \\ (E + p)u_x \end{bmatrix} + \frac{\partial}{\partial y} \begin{bmatrix} \rho u_y \\ \rho u_y u_x \\ \rho u_y^2 + p \\ (E + p)u_y \end{bmatrix} = 0$$

Compressible Euler equations on cartesian meshes



# Some of our daily problems

- Algorithm A is faster than algorithm B
  - Is it due to the algorithm, or its implementation ?
  - Will it be true with CPUs 2 years from now ?
- Algorithm A runs in 2 minutes on processor 1
  - What about on processor 2 ?



# Some of our daily problems

- What limits algorithm A performance ?
  - When do I stop optimizing it ?
  - How can I improve it ?
- Can I explain *in simple terms* why my algorithm runs slow/fast ?

# What is performance modeling ?

For us, it is a tool to:

1. *predict*

2. *understand*

3. *explain*

algorithm **runtime** (not FLOPs)

# Why do we care ?

## 1. *Predict* algorithm performance

- **WHAT:** quantitative performance blueprint
- **WHY:** hardware extrapolation
- **WHY:** fair comparison between algorithms

# Why do we care ?

## 2. *Understand* algorithm performance

- **WHAT:** identifying bottlenecks
- **WHY:** getting ideas for improvement

# Why do we care ?

## 3. *Explain* algorithm performance

- **WHAT:** separate concepts from technicalities
- **WHY:** HPC expertise is scarce

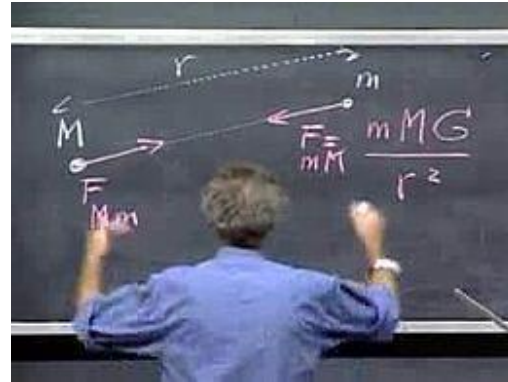
# Our goal

Apply existing models to gain insight  
(not invent new ones)

# A digression on models

In physics, a model is

- A simplified version of reality...
- ...that helps predict/understand it



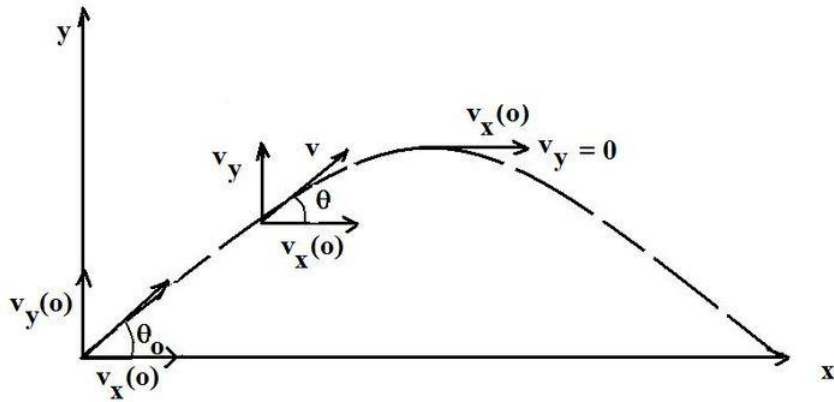
# A digression on models

- There is no good/bad model
- All models are false
- It is all about the match between
  - The model
  - The phenomenon to be studied



# A fundamental rule

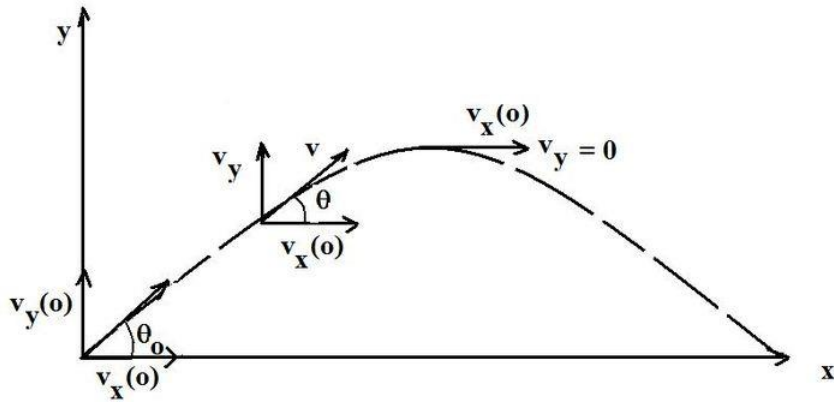
“use the **simplest** model that predicts and/or explains your data”



$$\left[ \frac{-\hbar^2}{2m} \nabla^2 + V \right] \Psi = i \hbar \frac{\partial}{\partial t} \Psi$$

# A fundamental rule

“use the **simplest** model that predicts and/or explains your data” (but not simpler)

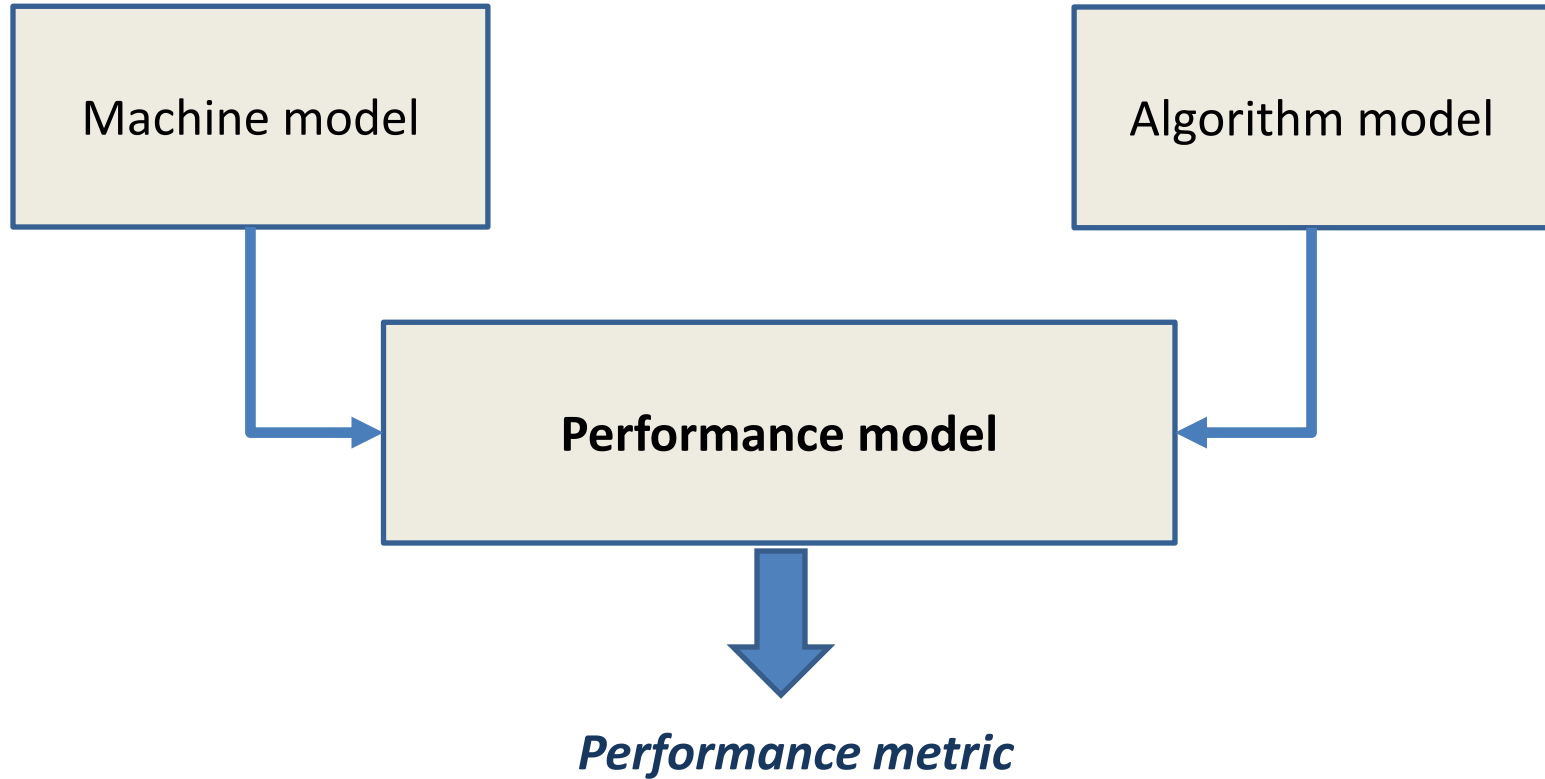


$$\left[ \frac{-\hbar^2}{2m} \nabla^2 + V \right] \Psi = i \hbar \frac{\partial}{\partial t} \Psi$$

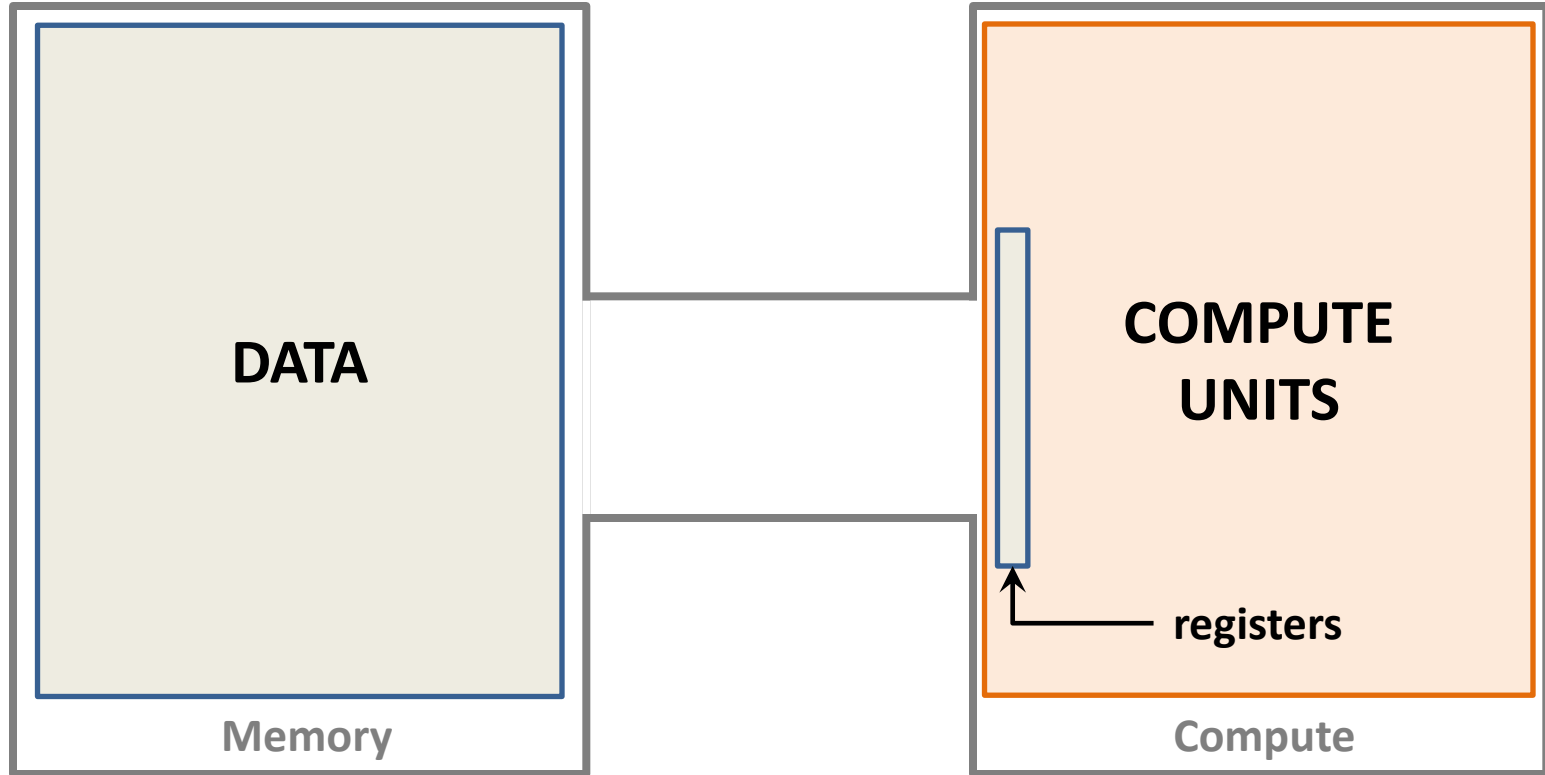
# Performance models

*Rooflines and ECM*

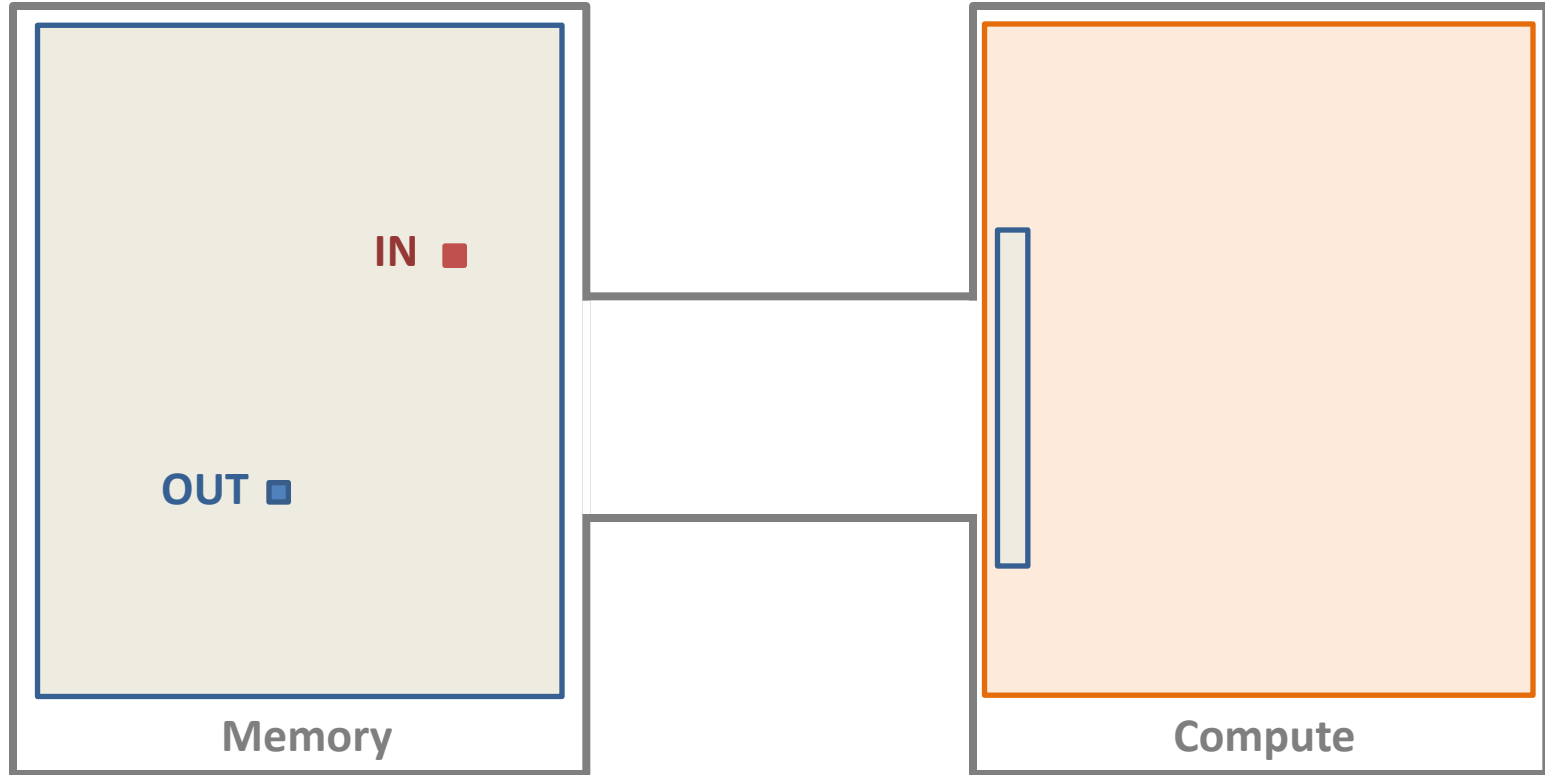
# Computational performance models



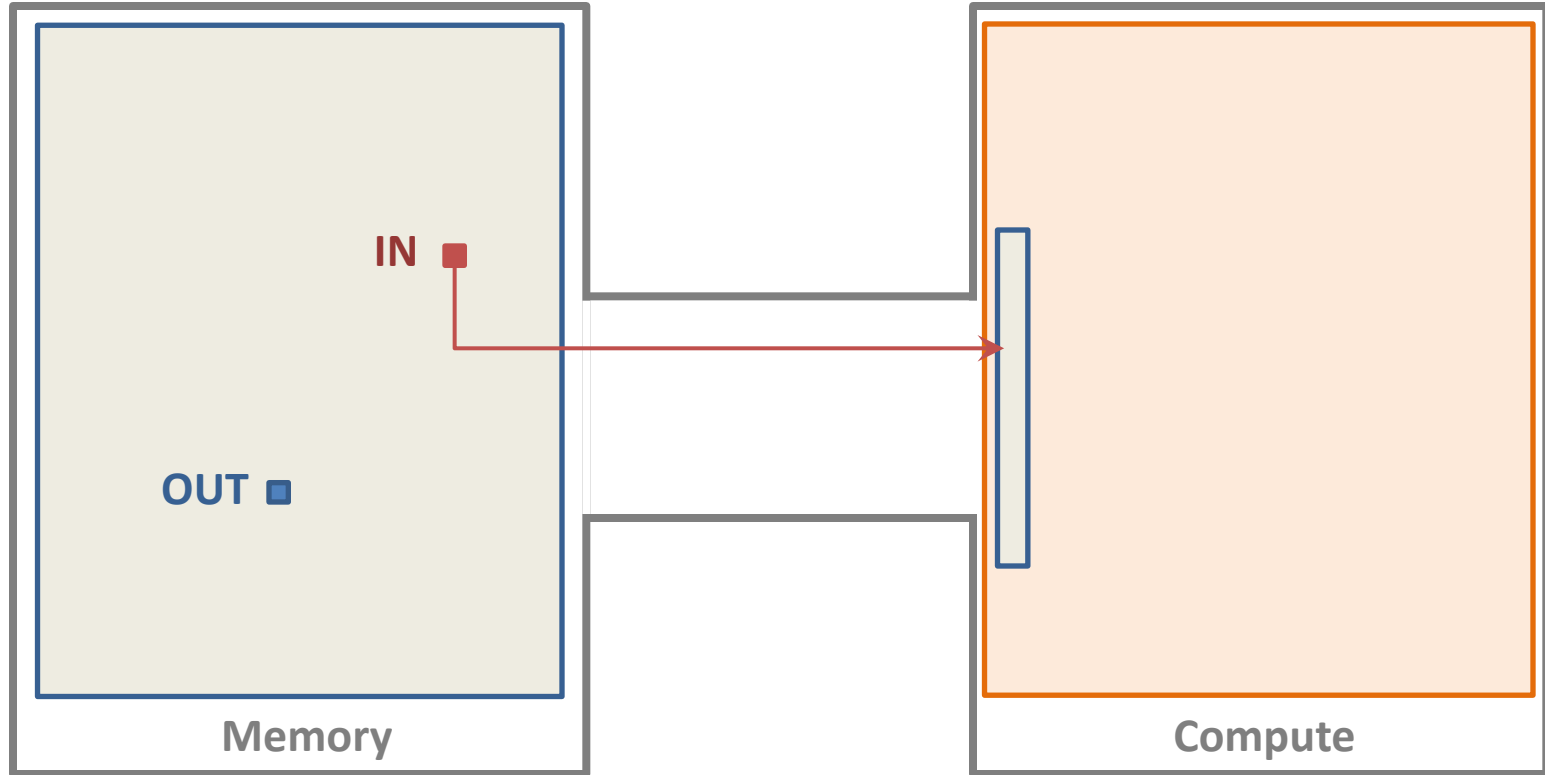
# Machine description



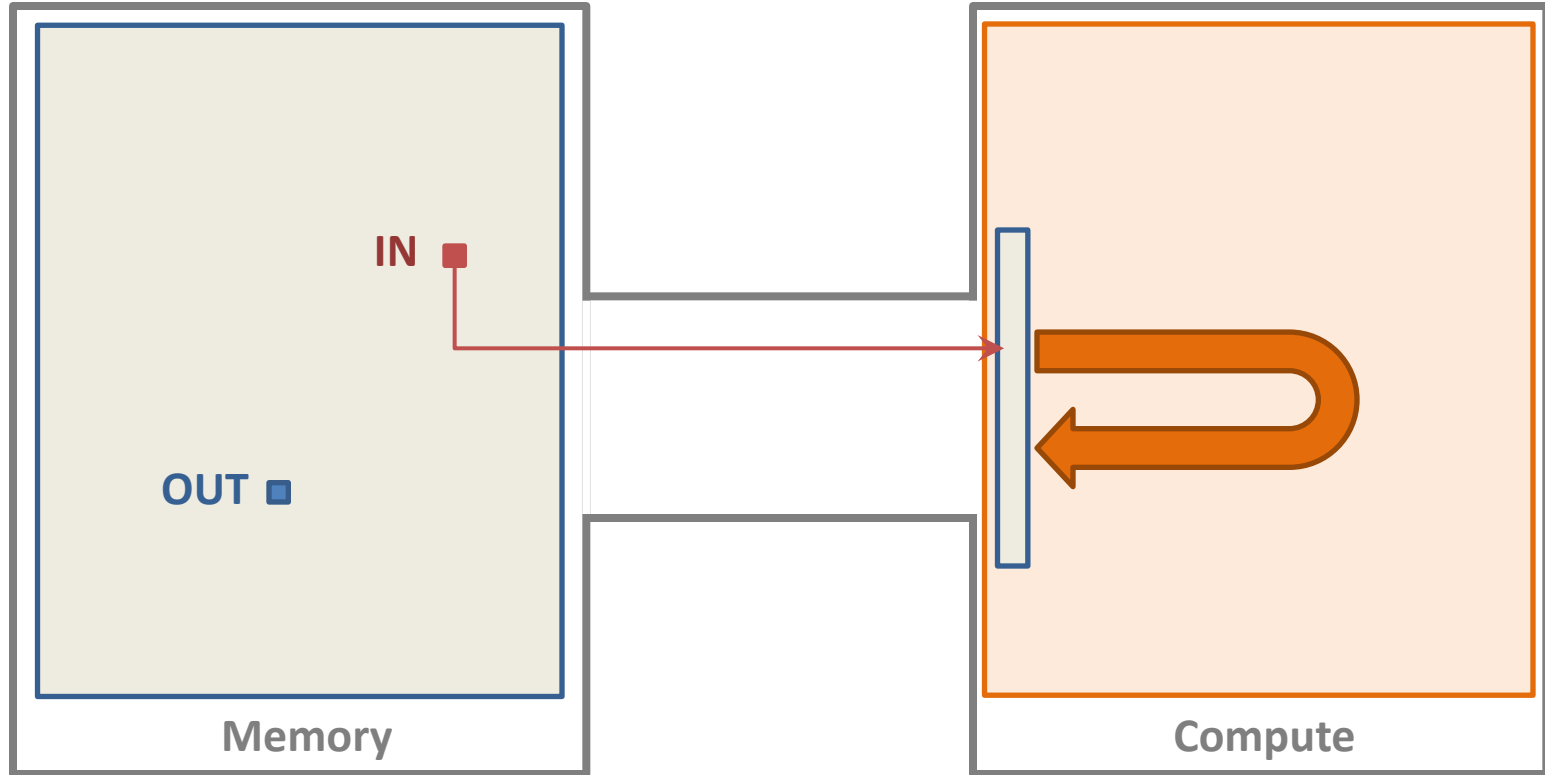
# Machine description



# Machine description

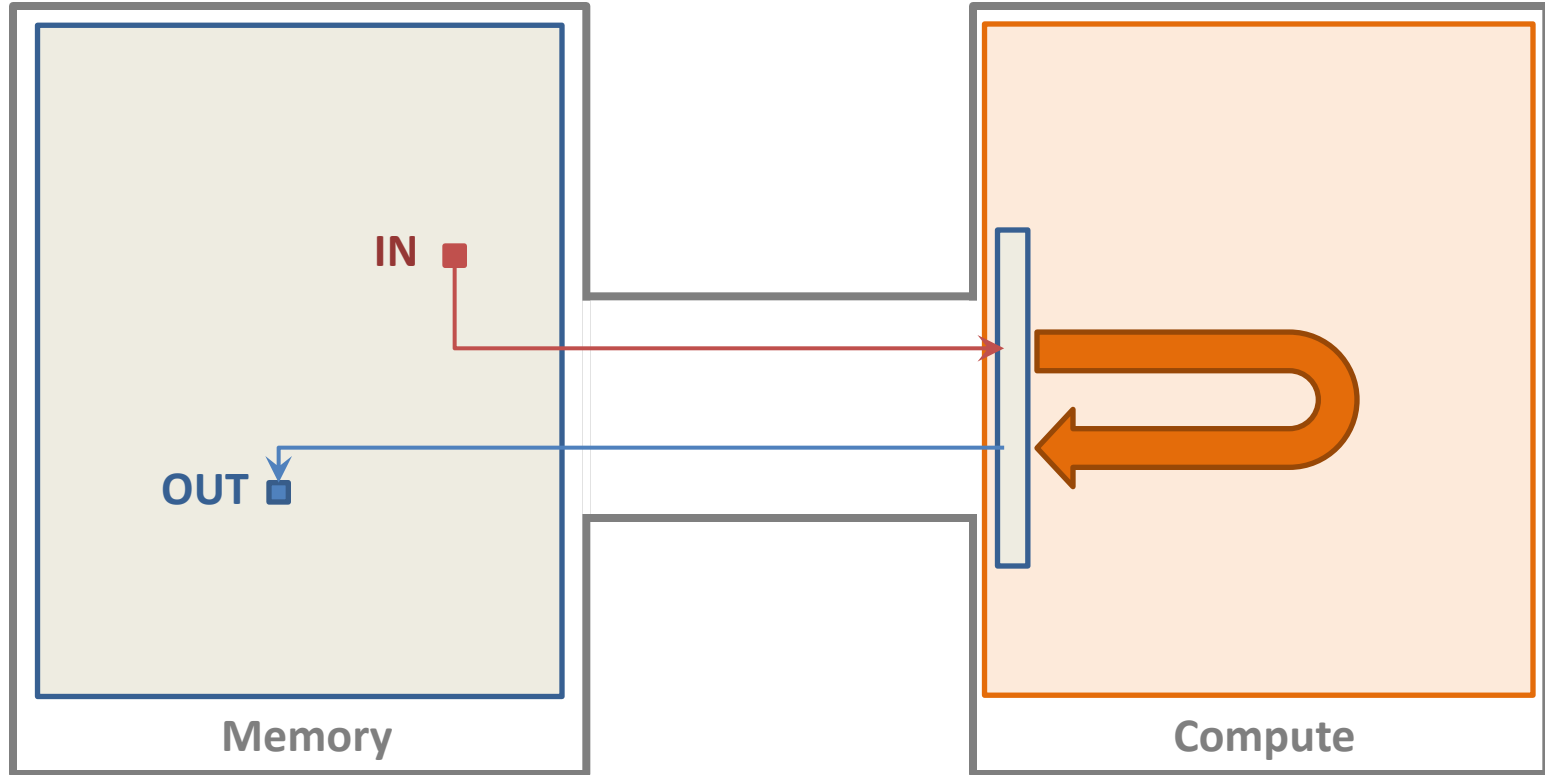


# Machine description





# Machine description



# The simplest model

## Count arithmetic operations

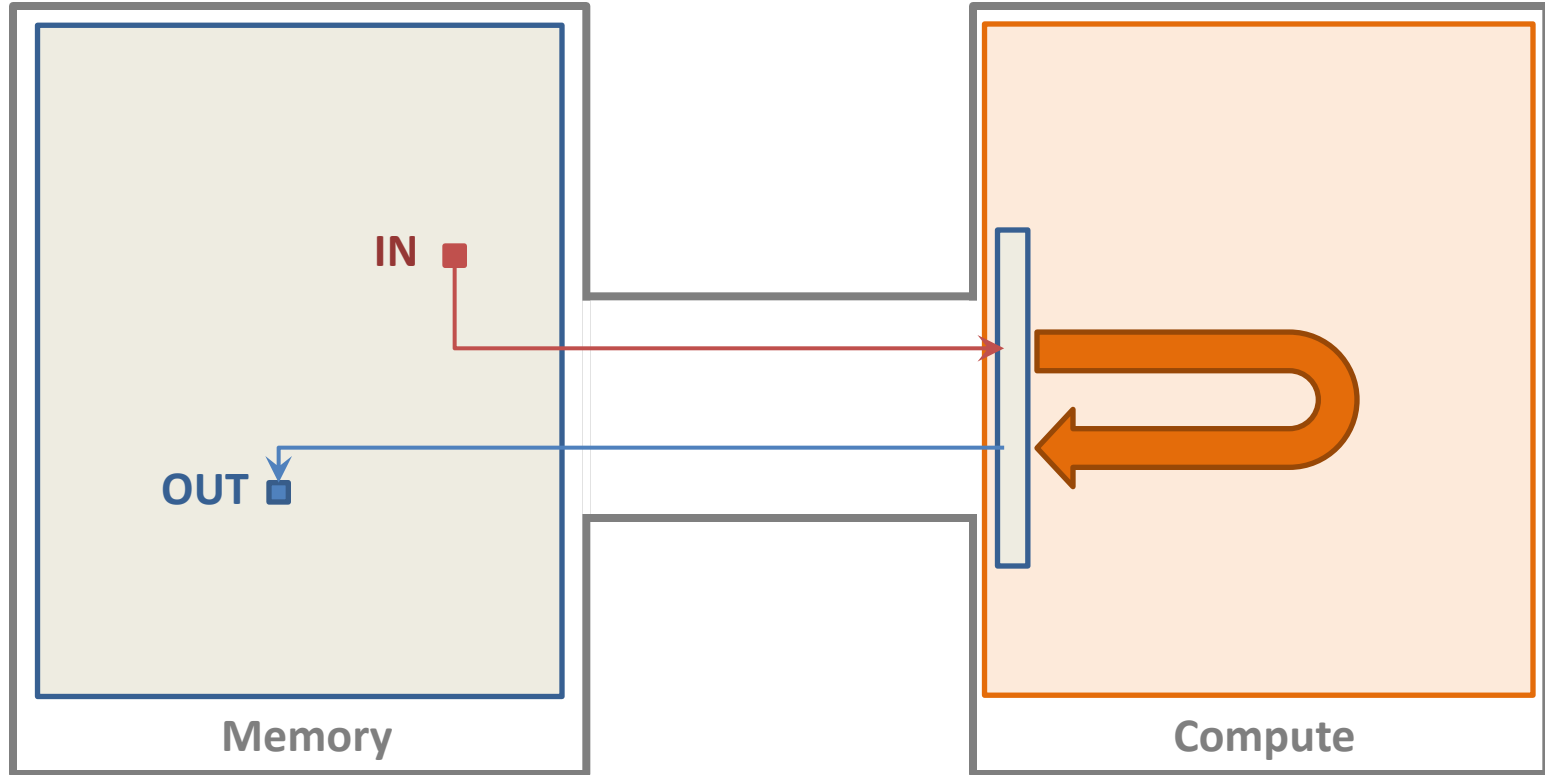
Ex. For (i=0; i < N; ++i) {  
    a[i] = b[i] + c[i] \* d[i]; }

$$T = \frac{2N \text{ \#cores}}{16 \text{ freq}}$$

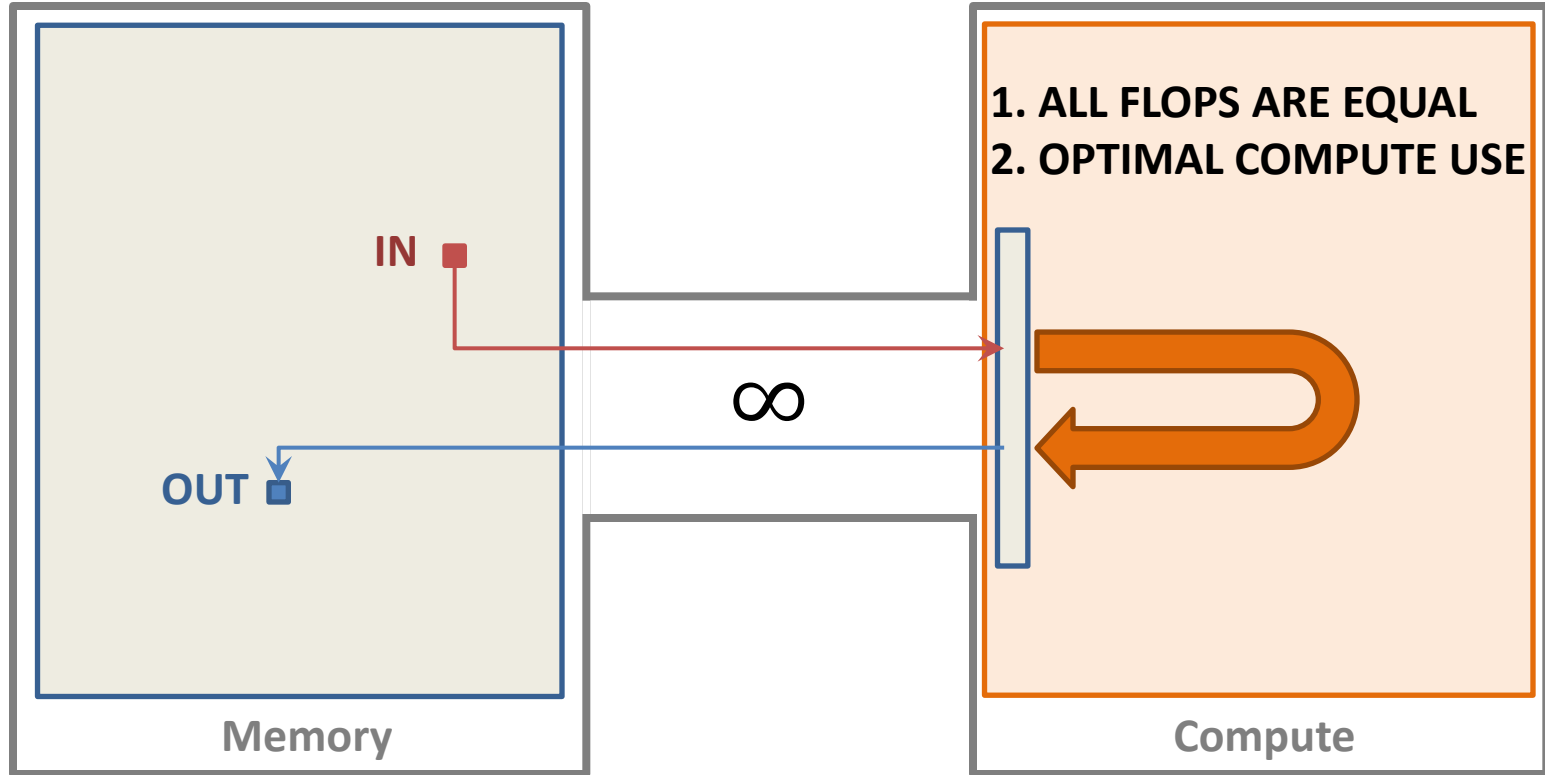


In double precision,  
on a Haswell/Broadwell/Skylake CPU

# Assumptions of this model



# Assumptions of this model



# The simplest model

- It rarely works

Intel Core i5 5200U @2.2GHz

Ex. For (i=0; i < N; ++i) {  
    a[i] = b[i] + c[i] \* d[i]; }

Predicted	Measured
17.6 10 <sup>9</sup> points/s	400 10 <sup>8</sup> points/s


- Our experience in compressible CFD:
  - Most of the time, data transfer is the bottleneck

# Another simple model

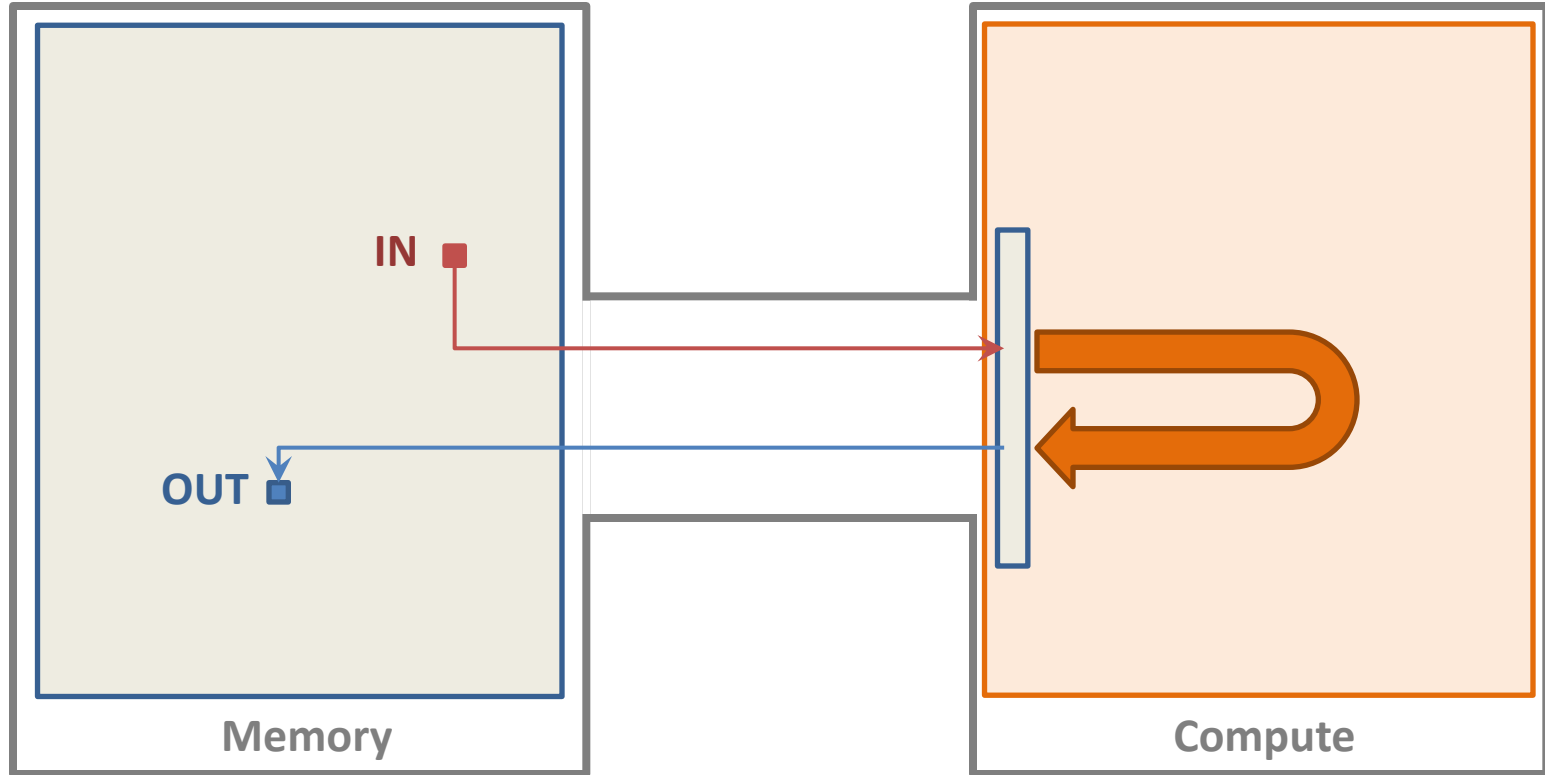
## Count data transfers

Ex. For (i=0; i < N; ++i) {  
    a[i] = b[i] + c[i] \* d[i]; }

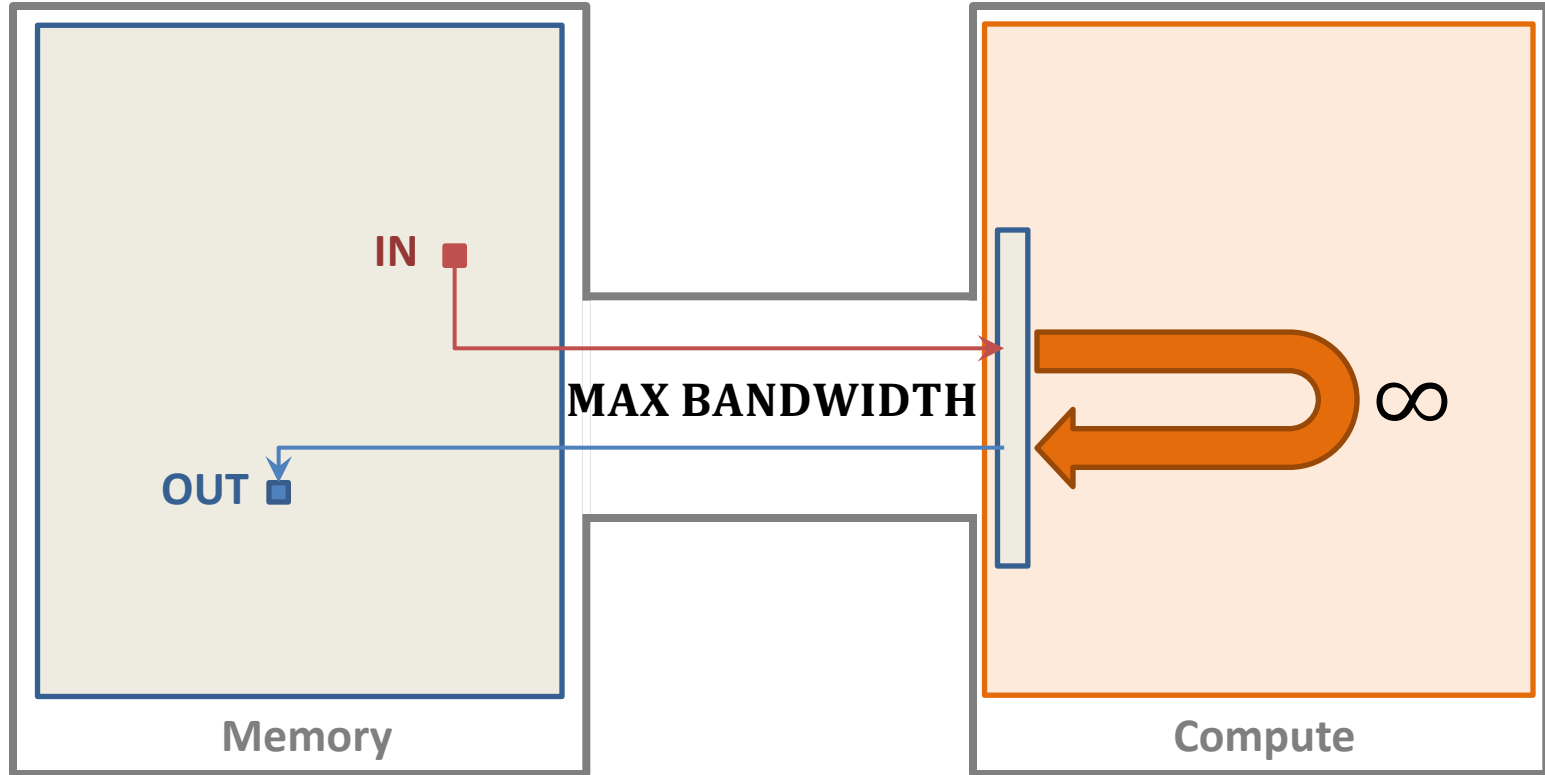
*Double precision*  $8 \times (4+1)$  *4 reads, 1 write*


$$T = \frac{40N}{\text{bandwidth}}$$

# Assumptions of this model



# Assumptions of this model





# Another simple model

- Works better for this example

Ex. For (i=0; i < N; ++i) {  
    a[i] = b[i] + c[i] \* d[i]; }

Intel Core i5 5200U @2.2GHz

Predicted	Measured
640 10 <sup>8</sup> points/s	532 10 <sup>8</sup> points/s



With **theoretical** bandwidth

# Towards roofline

$$T \geq T_{compute}, \text{ and } T \geq T_{transfer}$$

Thus,  $T \geq \max( T_{compute} , T_{transfer} )$   
**always true**

# Predictive roofline

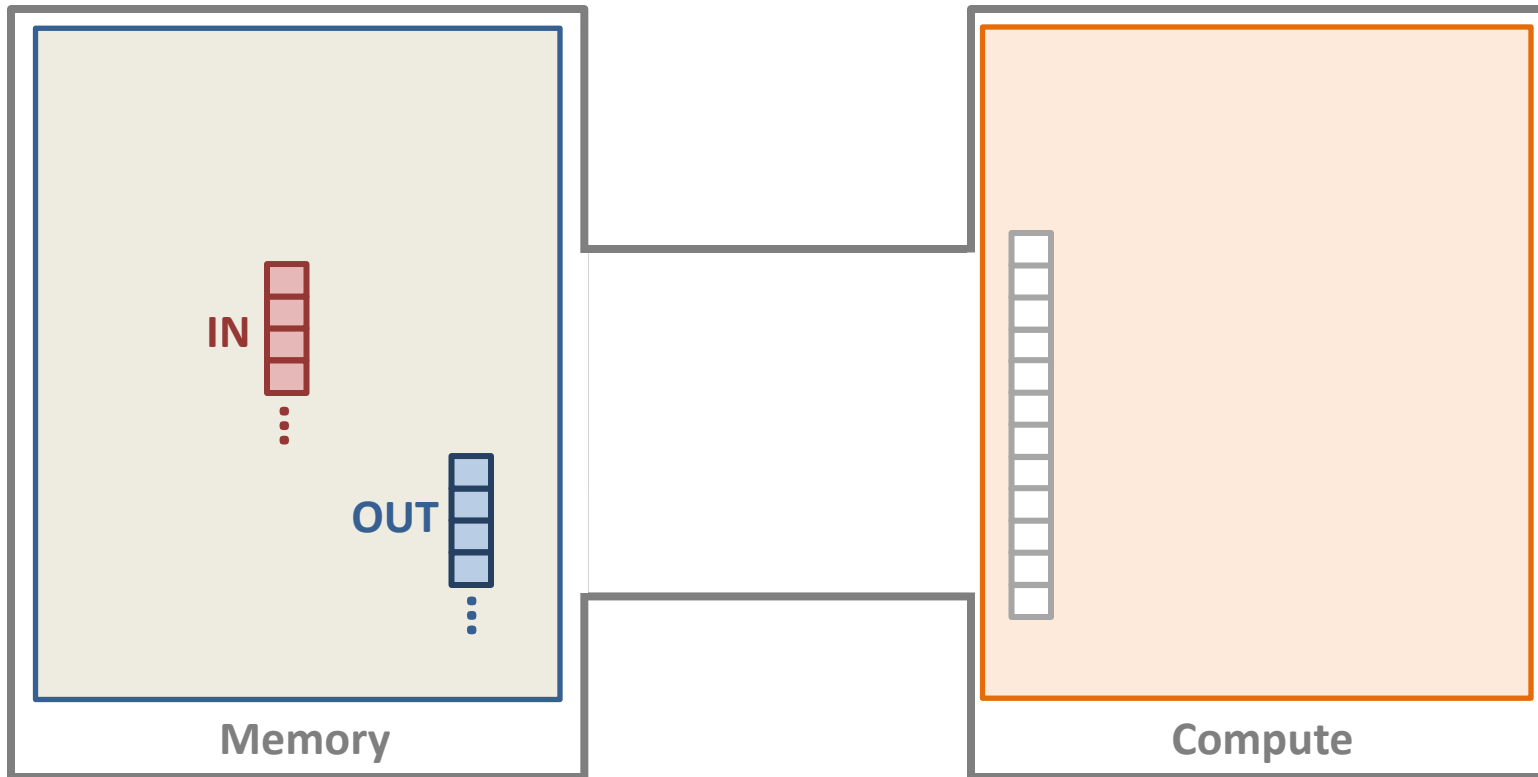
$$T \geq T_{compute}, \text{ and } T \geq T_{transfer}$$

Thus,  $T = \max( T_{compute} , T_{transfer} )$

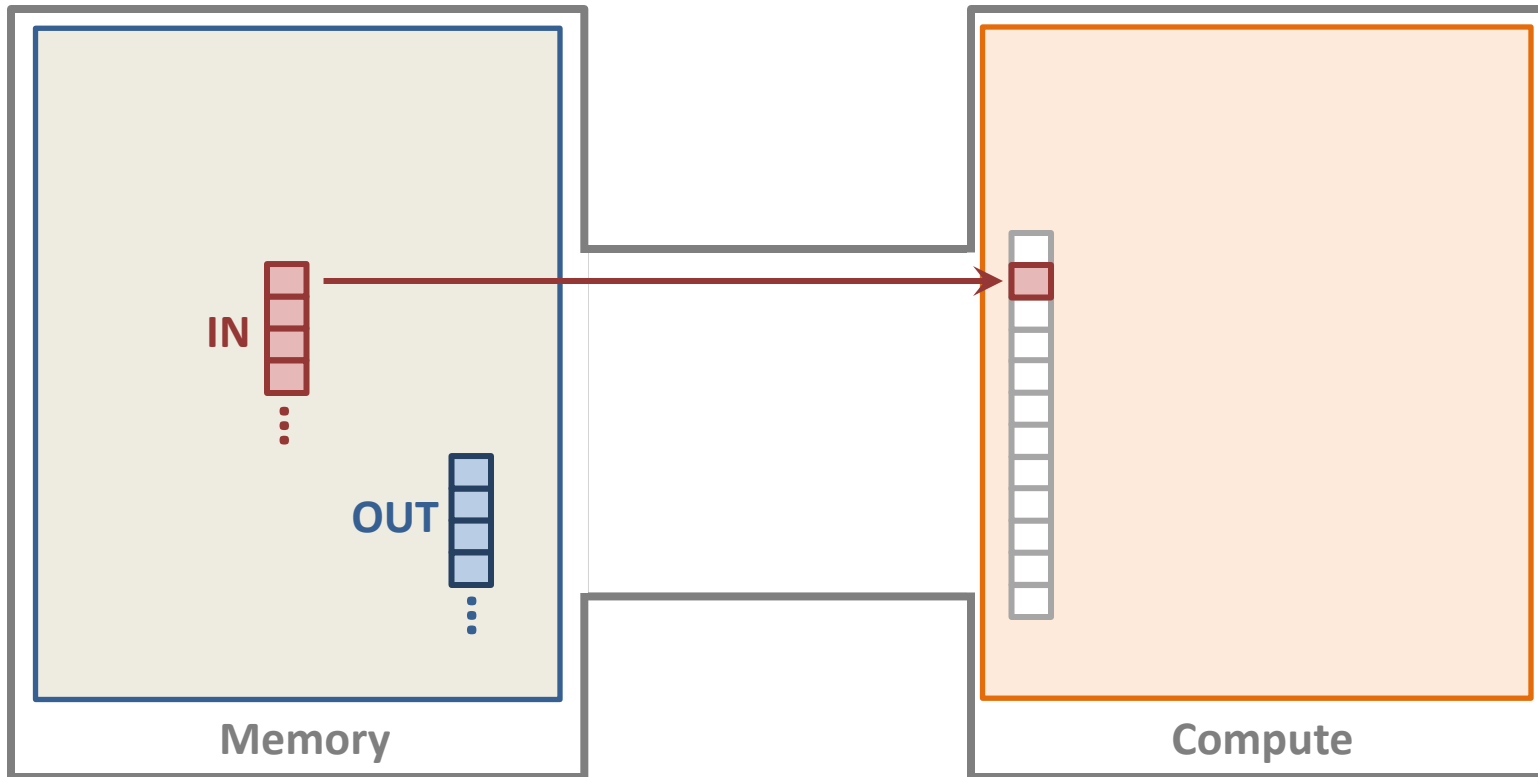
**Predictive roofline**

[Williams, Waterman & Patterson 2009] + (many) successors

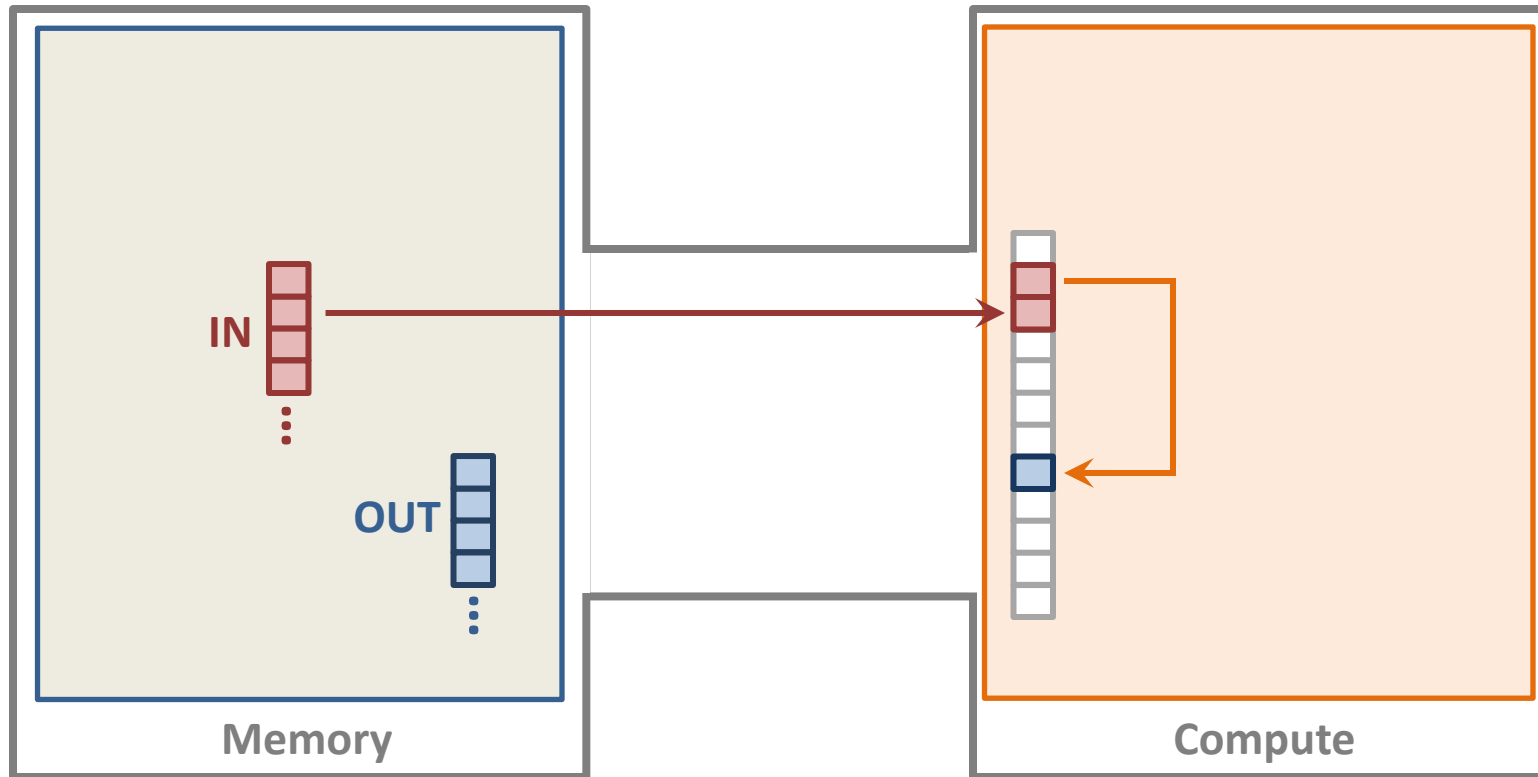
# Roofline = perfect overlap



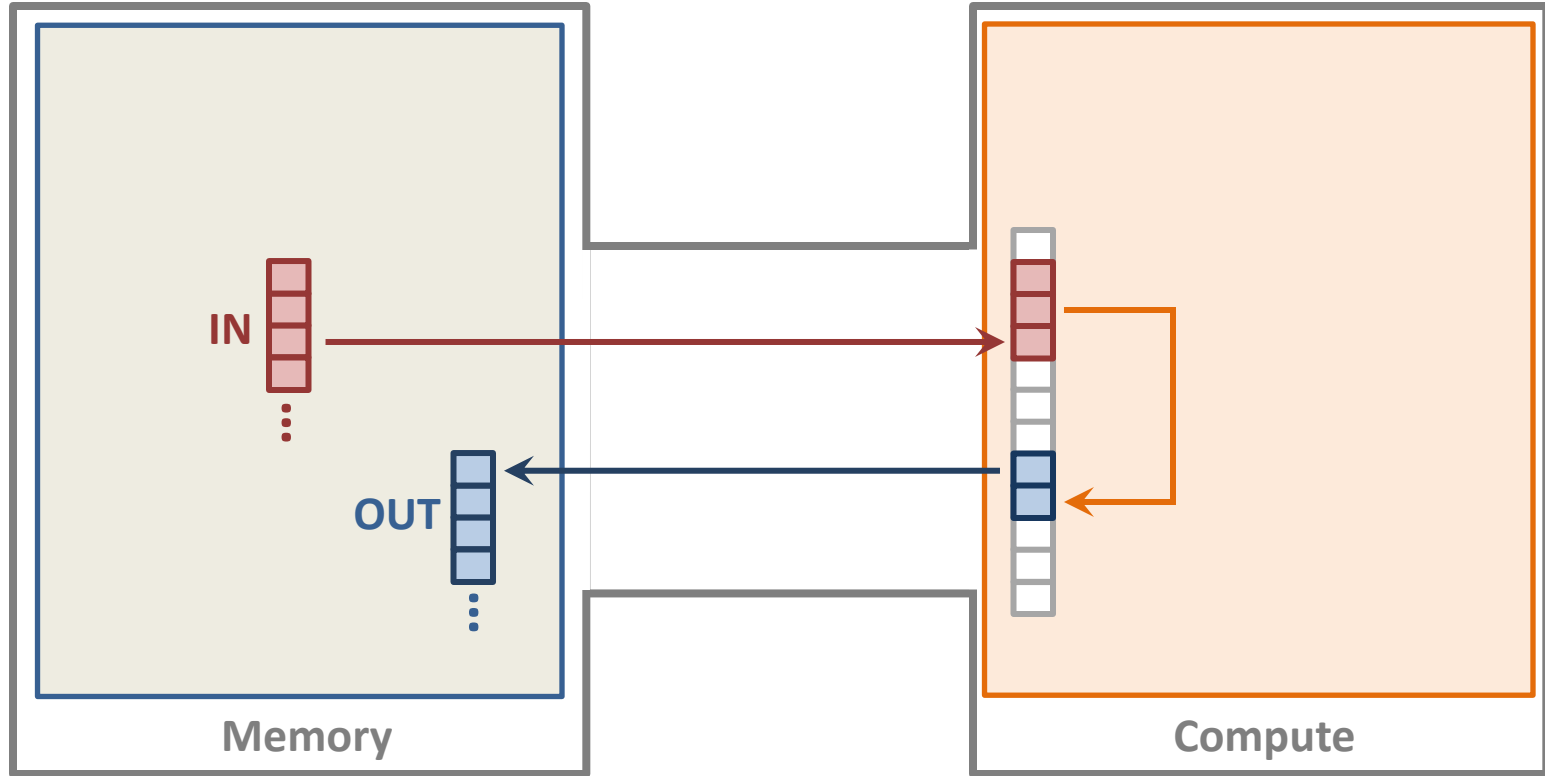
# Roofline = perfect overlap



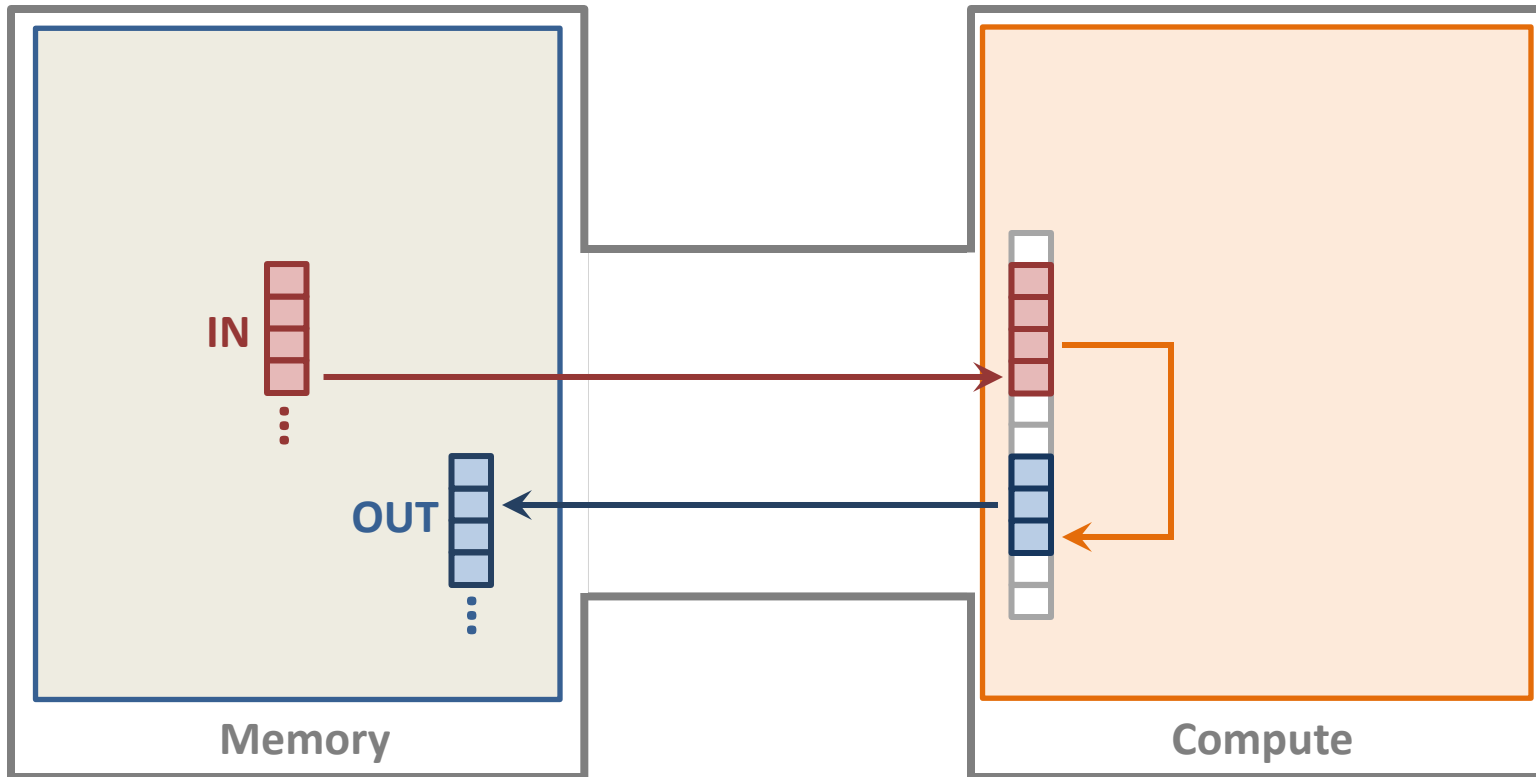
# Roofline = perfect overlap



# Roofline = perfect overlap



# Roofline = perfect overlap





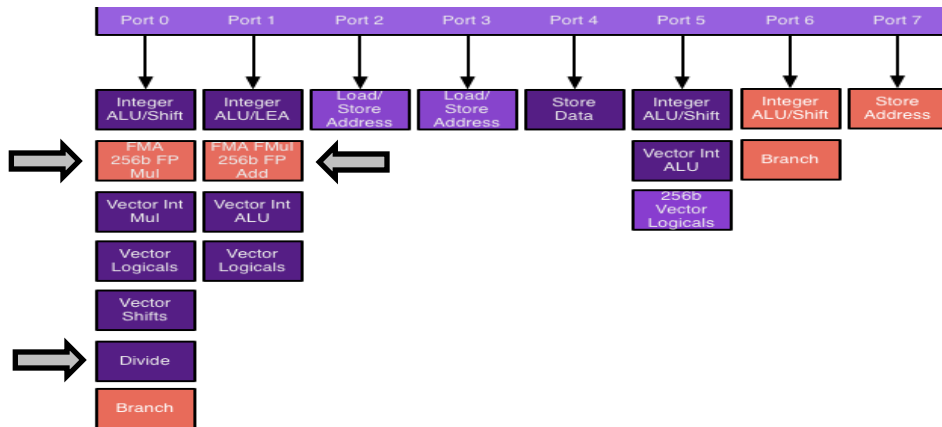
# Roofline assumptions

- Perfect overlap between computation and memory transfers
  - Need “predictive” memory access patterns
- If  $T_{compute}$  and  $T_{transfer}$  estimated with peak FLOPs and bandwidth: **ideal roofline**

# Better $T_{compute}$ estimate

- All FLOP are not equal
  - Ex.  $\sqrt{\quad}$  or  $\div \approx 10^1 \text{ cycles}$ , + or  $\times \approx 1 \text{ cycle}$
- 1 core process FLOPs in parallel (ILP)

Ex. Intel Haswell



# Better $T_{compute}$ estimate

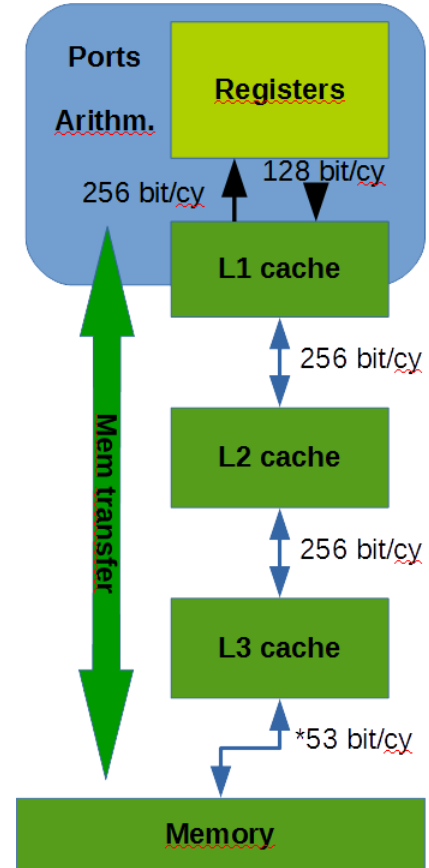
- Map the algorithm graph to the microarchitecture and estimate the critical path
- Leverage existing tools: Intel IACA
  - See for instance [Treibig et. al 2013]
  - Static analysis
  - Gory detail: AFAIK, you need to vectorize by hand for Intel IACA to work...

# Better $T_{transfer}$ estimate

- Use an effective bandwidth – not peak
- E.g. the result of a STREAM benchmark (memcpy)

# Better $T_{transfer}$ estimate

- Take caches into account
- We use the ECM model
  - [Treibig & Hager 2010] + successors
  - “Roofline + caches”
  - (+ counting memory instructions)
  - Is able to predict multicore scalability



# Our final model

- We chose to use the ECM model
  - Methodology similar to [Stengel et. al 2015]
- Not too complicated (YMMV)
- Main ingredients:
  - Roofline + cache hierarchy + effective bandwidth
  - (+ minor things)
  - Compute time estimated using static analysis to map algorithm graph to microarchitecture

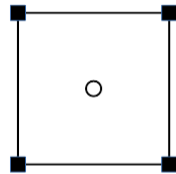
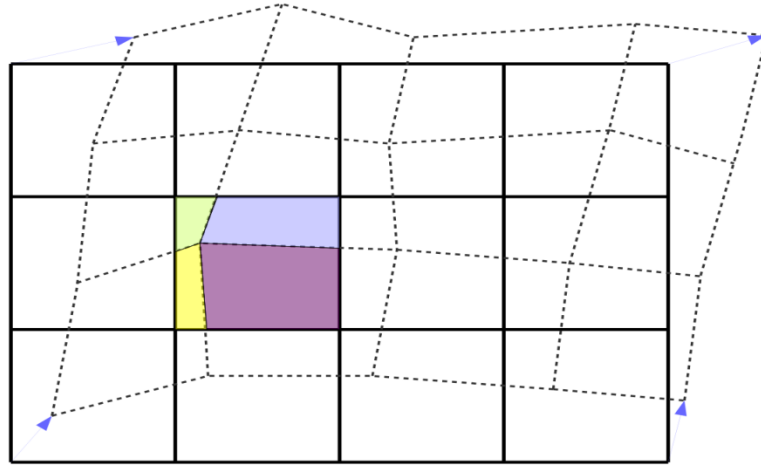
# **Application to CFD**

# Starting point

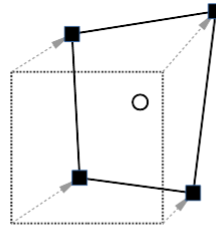
- Baseline algorithm: Lagrange-remap solver
- Legacy algorithm (Von Neumann & Richtmyer 1950)
- Robust, used in the industry
  - Hydrocodes, crash simulations,...



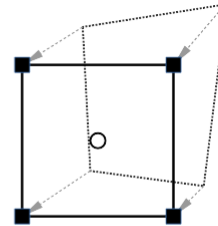
# Lagrangian remap hydrodynamics



0) Eulerian cell

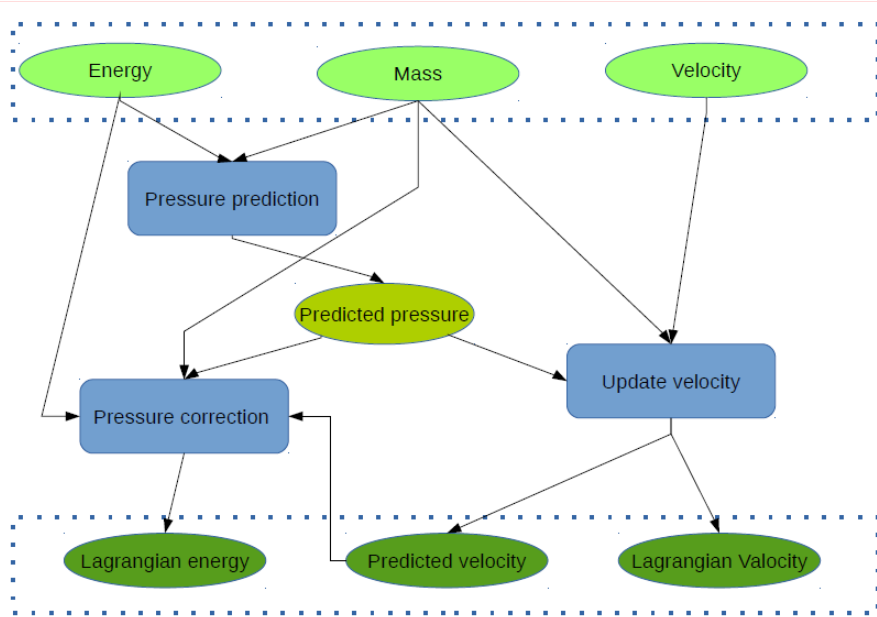


i) Lagrange step

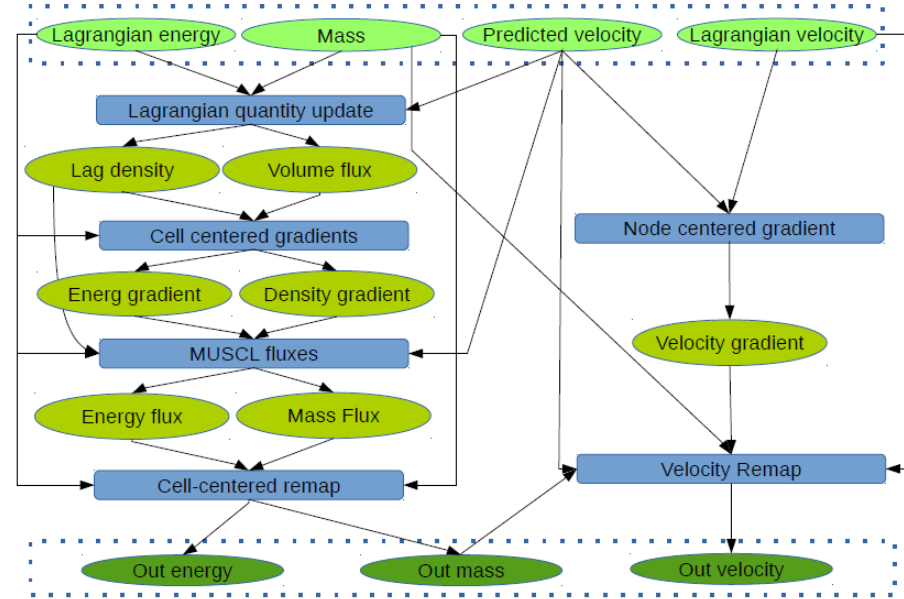


ii) Remapping step

# Lagrangian remap hydrodynamics



LAGRANGE



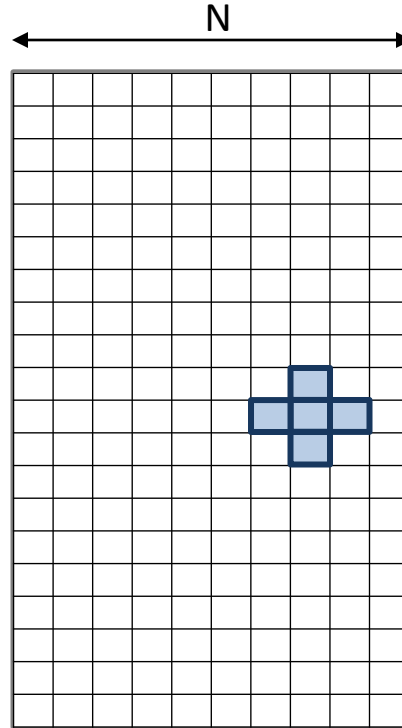
REMAP

Input/Output data, kernels

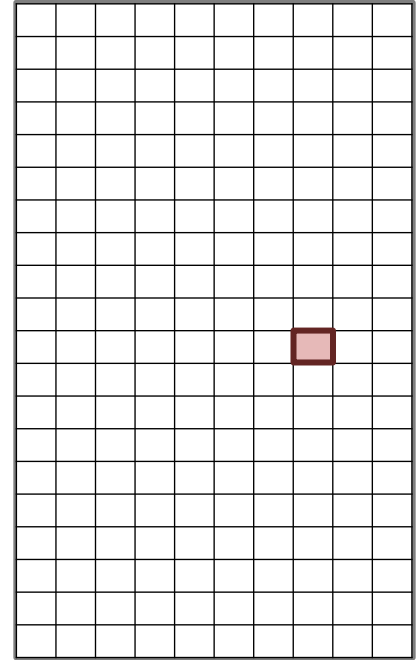
# Physical access pattern

Stencil pattern

Example: pressure gradient



IN

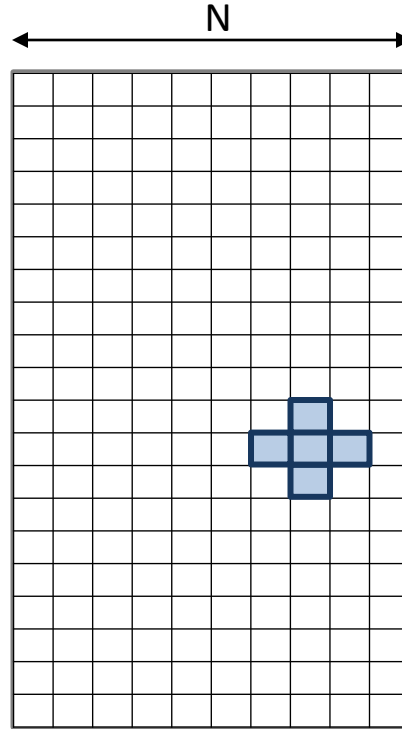


OUT

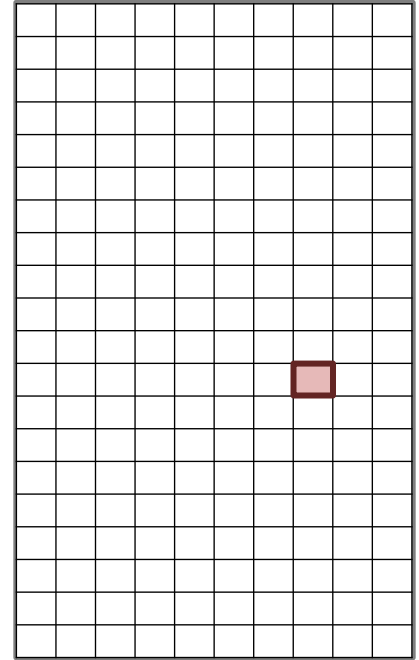
# Physical access pattern

Stencil pattern

Example: pressure gradient

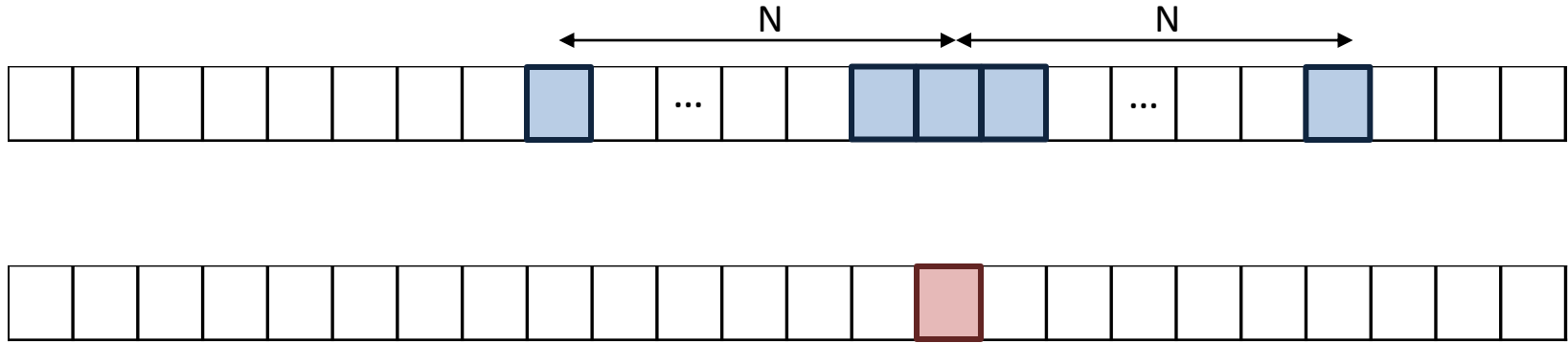


IN

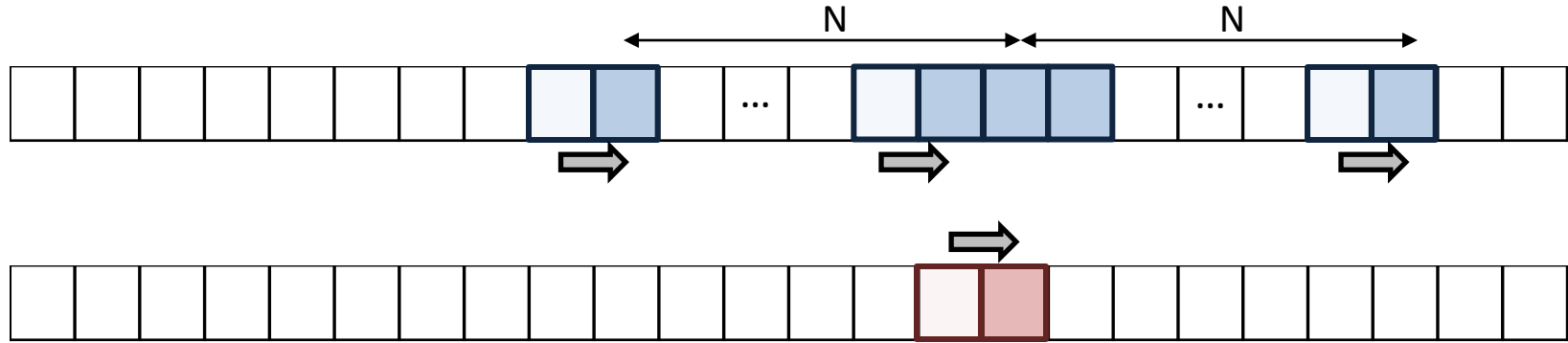


OUT

# Memory access pattern



# Memory access pattern

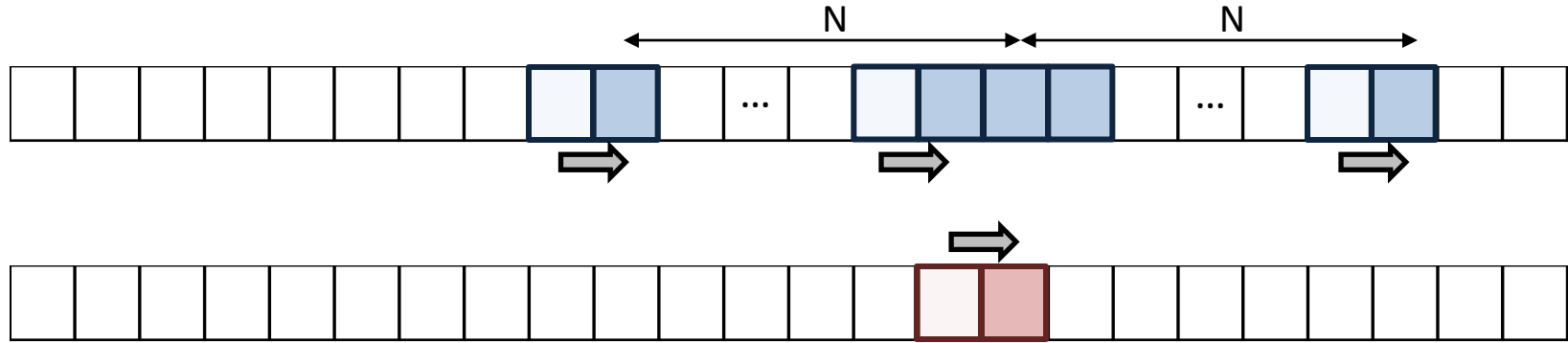


Predictive access pattern (at compile time): *stride 1*



**Q. Good fit for ECM ?**

# Memory access pattern



Predictive access pattern (at compile time): *stride 1*



**Q. Good fit for ECM ? A. Yes**

# What we did (1)

## Phase 1: predict and validate

- Applied ECM model to all kernels
  - on intrinsics AVX multithreaded version of the code
- Used Intel IACA to estimate computation
- Use L1/L2/L3/RAM description to estimate data transfers
- Got predictions for single core and multi-core



# Performance prediction

Kernel name	data in L3				data in memory			
	type	prediction	measure	error	type	prediction	measure	error
<i>Lagrange kernels</i>								
Pressure prediction	CB	168 cy/CL	173 cy/CL	3%	CB	168 cy/CL	173 cy/CL	3%
Update velocity	CB	56 cy/CL	59 cy/CL	5%	MB	80 cy/CL	78 cy/CL	3%
Pressure correction	CB	56 cy/CL	58 cy/CL	3%	MB	71 cy/CL	65 cy/CL	8%
<i>Remap kernels</i>								
Lagrangian q. update	CB	56 cy/CL	58 cy/CL	3%	MB	57 cy/CL	58 cy/CL	2%
Cell centered gradient	CB	168 cy/CL	170 cy/CL	1%	CB	168 cy/CL	170 cy/CL	1%
MUSCL fluxes	MB	21 cy/CL	25 cy/CL	16 %	MB	44 cy/CL	42 cy/CL	5%
Cell centered remap	CB	56 cy/CL	57 cy/CL	2%	MB	76 cy/CL	65 cy/CL	17%
Node centered gradient	CB	168 cy/CL	170 cy/CL	1%	CB	168 cy/CL	170 cy/CL	1%
Velocity remap	MB	13 cy/CL	14 cy/CL	7%	MB	30 cy/CL	25 cy/CL	17%

Single core mean/median error in [3%, 8%]

# Performance prediction

Kernel name	Speedup on 8-core Intel Sandy Bridge			Speedup on 4-core Intel Haswell		
	type	predicted	measured	type	predicted	measured
<i>Lagrange kernels</i>						
Pressure prediction	CB	8	7.45	CB	4	3.5
Update velocity	MB	2.6	3.0	MB	1.4	1.6
Pressure correction	MB	2.8	3.0	MB	1.5	1.5
<i>Remap kernels</i>						
Lagrangian q. update	MB	3.2	3.4	MB	1.6	1.6
Cell centered gradient	CB	8	7.9	CB	4	3.9
MUSCL fluxes	MB	2.3	2.6	MB	1.3	1.4
Cell centered remap	MB	2.5	2.7	MB	1.2	1.6
Node centered gradient	CB	8	7.9	CB	4	3.9
Velocity remap	MB	2.3	2.7	MB	1.5	1.5

Multicore scalability also predicted

# Predict and understand

Kernel name	Predicted speedup (1 core)	Measured speedup (1 core)
PEAK	1.53	1.53
Pressure prediction	1.16	1.15

Hardware extrapolation

(SandyBridge 2.6GHz to Haswell 2.0GHz)

# Predict and understand

<b>Kernel name</b>	Predicted speedup	Measured speedup
Pressure correction	1.25	1.24
Update velocity	1.16	1.13

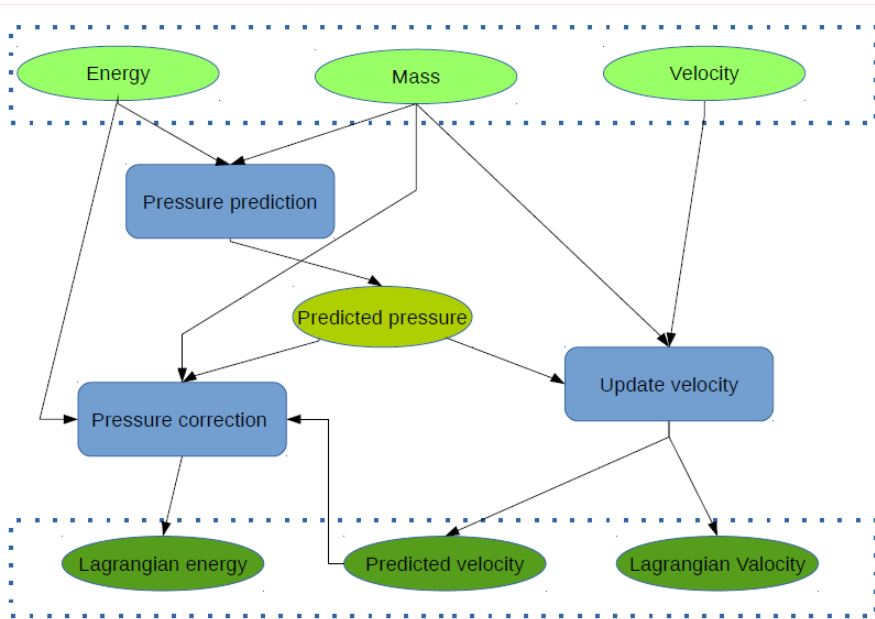
Cache blocking influence

# What we did (2)

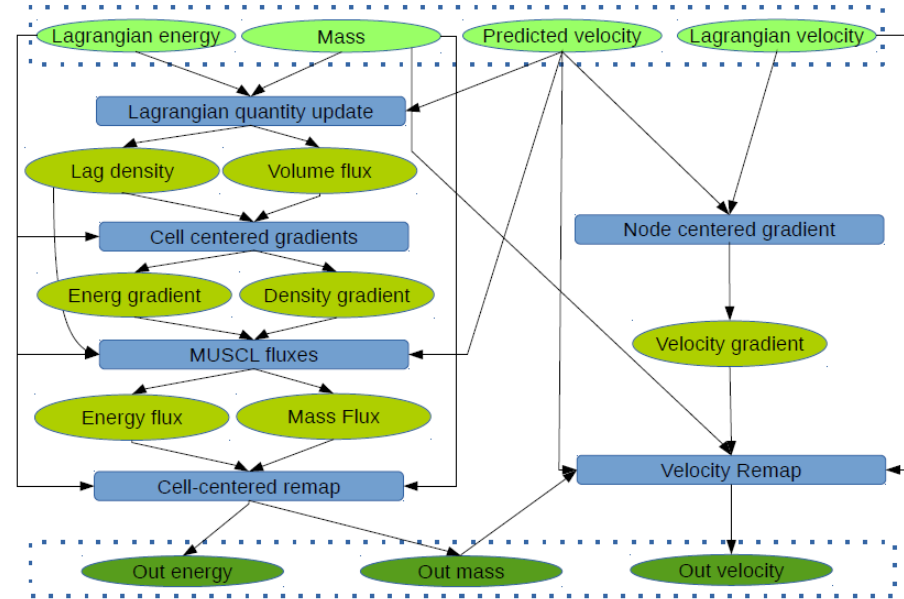
## Phase 2: understand and redesign

- Identify bottlenecks

# What's the bottleneck ?



LAGRANGE



REMAP

Input/Output data, kernels

# What we did (2)

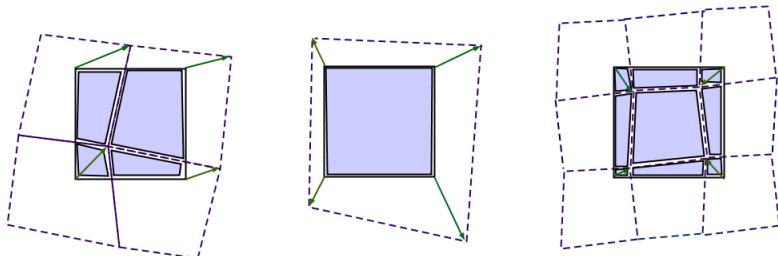
## Phase 2: understand and redesign

- Bottleneck 1: lots of kernels
  - **WHAT:** Data transfers are the bottleneck
  - **WHY:** Variables live on several grids (staggered)
  - **WHY:** Several phases (Lagrange + remaps)
  - Kernel fusion not straightforward

# What we did (2)

## Phase 2: understand and redesign

- Bottleneck 2: multimaterial remap is not SIMD friendly
  - **WHY:** geometric remapping = lots of different cases

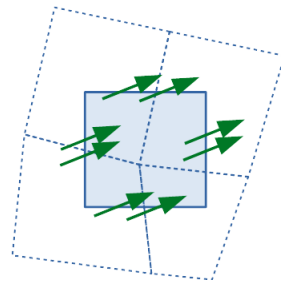




# What we did (2)

## Phase 2: understand and redesign

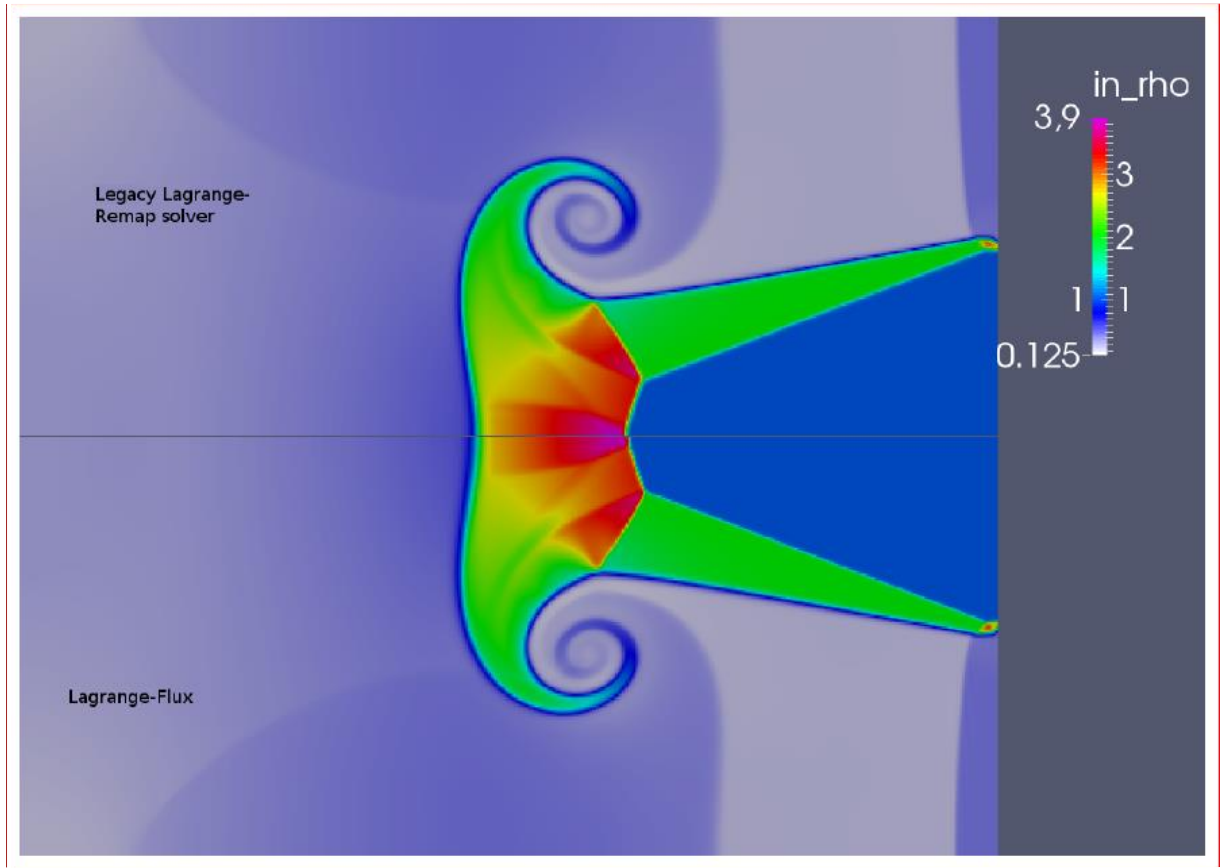
- Solution: Lagrange-Flux schemes
- geometric-free reformulation by balance of advection fluxes
  - Only one grid, only two kernels
  - Remapping is SIMD friendly



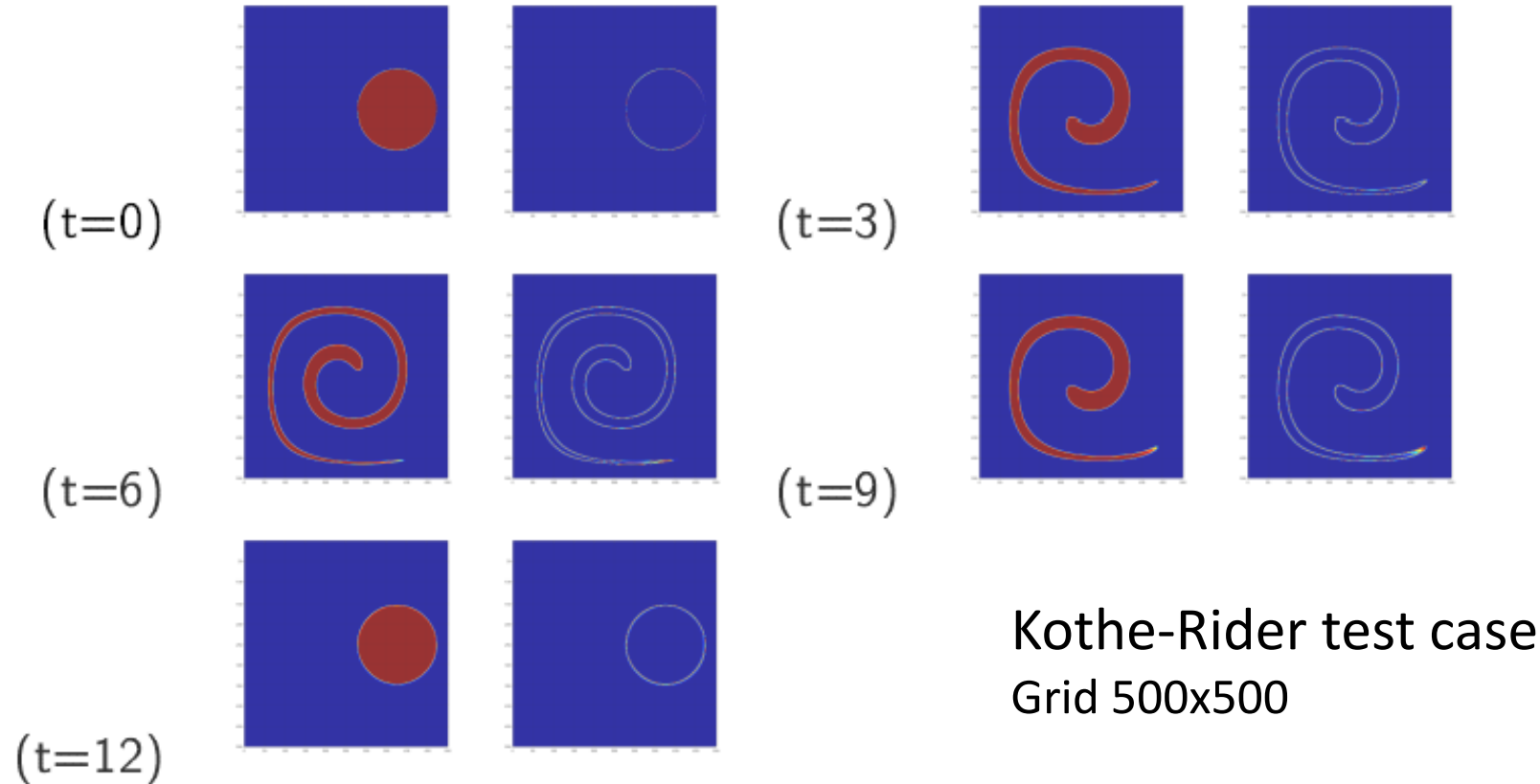
# Validation

Lagrange-remap

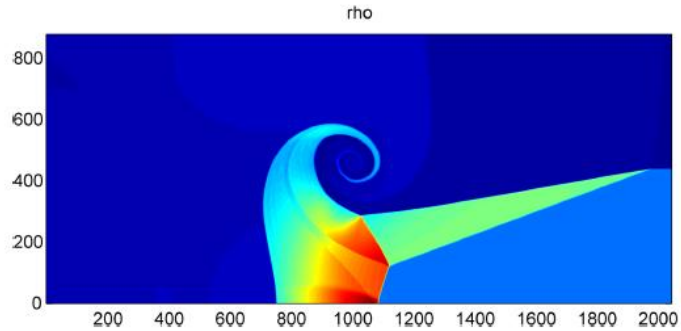
Lagrange-flux



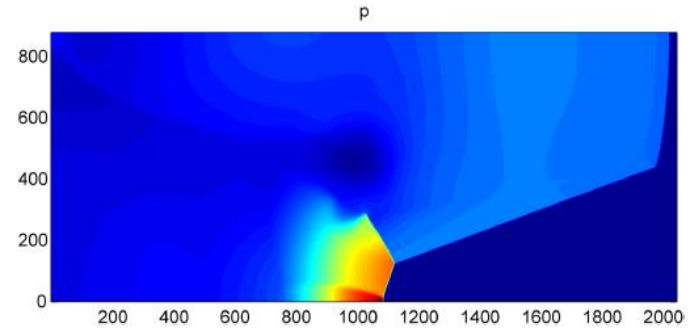
# Multimaterial validation



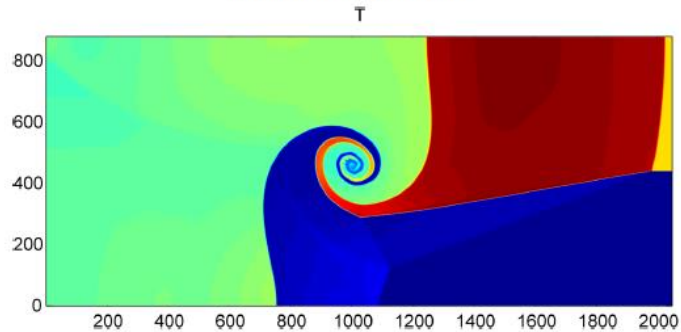
# Multimaterial validation



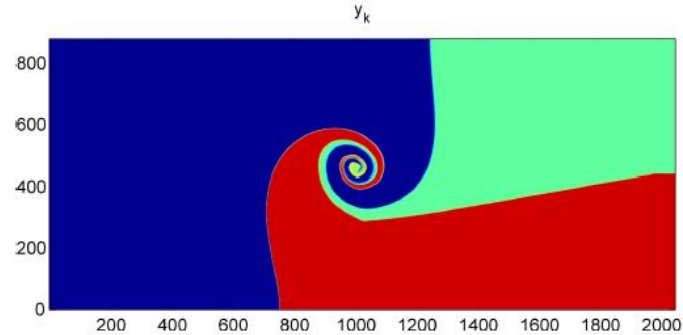
(a) Density field



(b) Pressure field

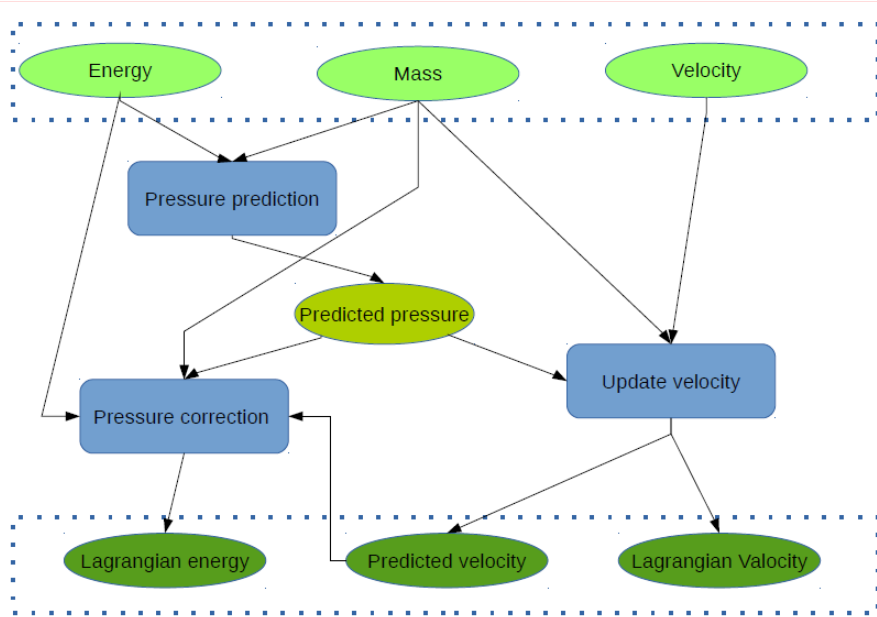


(c) Temperature field

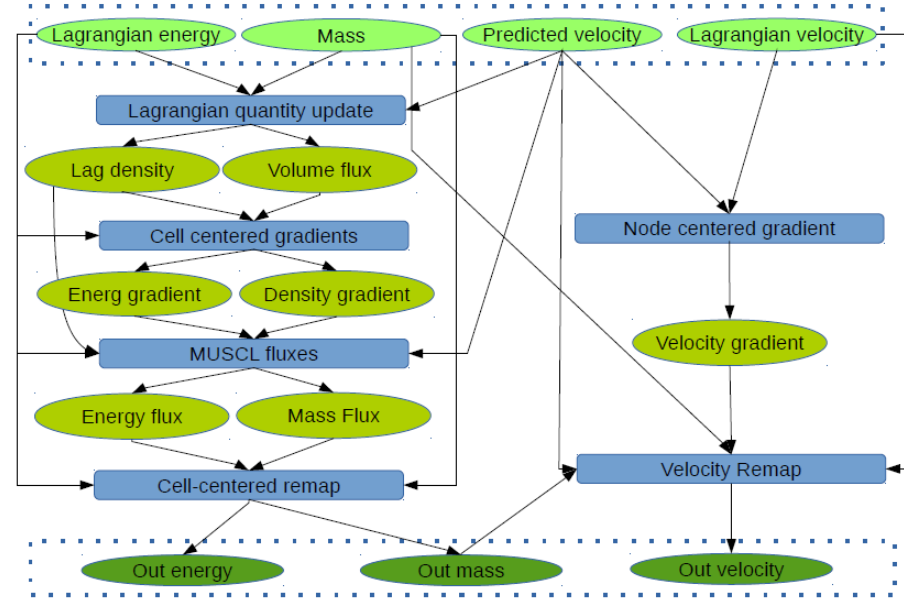


(d) Colored representation of material indicators

# Lagrangian remap vs Lagrange Flux



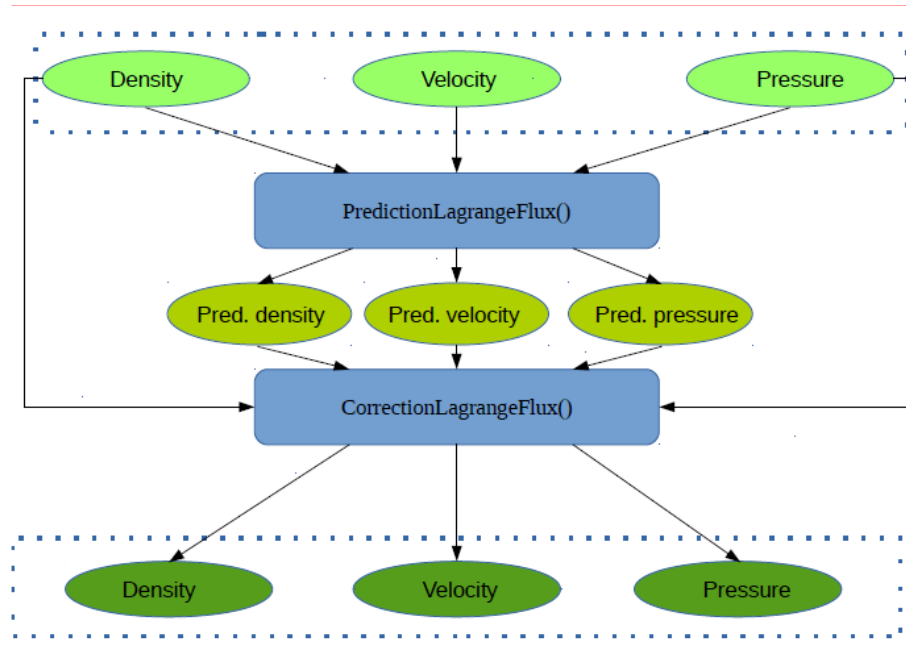
LAGRANGE



REMAP

Input/Output data, kernels

# Lagrangian remap vs Lagrange Flux



LAGRANGE FLUX

Input/Output data, kernels

# What we did (3)

## Phase 3: validate the performance our new scheme

Scheme	1 core	1 core AVX	16 cores AVX	scalability
Lagrange-Flux	1.9	3.9	52.0	27.1
Lagrange-Remap	2.4	3.7	36.5	14.6

Absolute performance in millions of cell updates / seconds

- Lagrange-flux is faster
  - Scalar Lagrange-flux is slower, but more scalable
  - **WHY:** because it is compute bound

# Conclusions

Performance modeling:

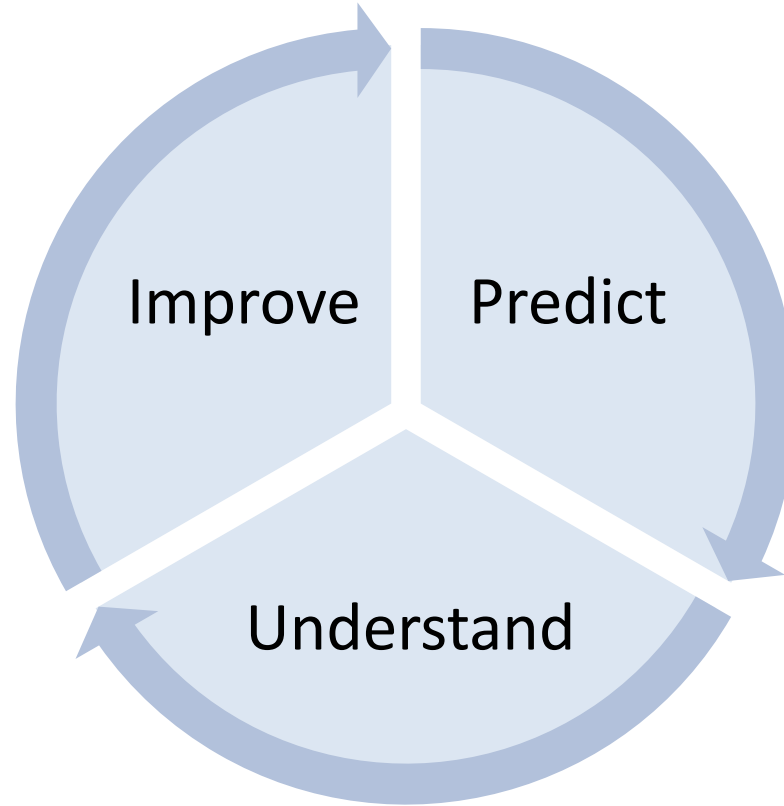
- makes HPC more quantitative
- is useful for algorithm optimization *and* design
  - for HPC engineers *and* applied mathematicians



# Conclusions

For our problems, ECM works very well

# Virtuous cycle



# Perspectives

- Extension to other machines (e.g. GPUs)
  - Leverage existing work
- Extension to other algorithms
  - CFD on unstructured grids ?

# References

- S. Williams, A. Waterman, and D. Patterson. "Roofline: an insightful visual performance model for multicore architectures." *Communications of the ACM* 52.4 (2009): 65-76.
- J. Treibig and G. Hager: *Introducing a Performance Model for Bandwidth-Limited Loop Kernels*. Proceedings of the Workshop "Memory issues on Multi- and Manycore Platforms" at [PPAM 2009](#), the 8th International Conference on Parallel Processing and Applied Mathematics, Wroclaw, Poland, September 13-16, 2009. [Lecture Notes in Computer Science](#) Volume 6067, 2010, pp 615-624. [DOI: 10.1007/978-3-642-14390-8\\_64](#). [arXiv:0905.0792](#)
- J. Treibig, G. Hager, H. G. Hofmann, J. Hornegger, and G. Wellein: Pushing the limits for medical image reconstruction on recent standard multicore processors. *International Journal of High Performance Computing Applications* 27(2), 162–177
- H. Stengel, J. Treibig, G. Hager, and G. Wellein: *Quantifying performance bottlenecks of stencil computations using the Execution-Cache-Memory model*. Proc. [ICS15](#), the 29th International Conference on Supercomputing, June 8-11, 2015, Newport Beach, CA. [DOI: 10.1145/2751205.2751240](#). Preprint: [arXiv:1410.5010](#)

# References

- T. Gasc, F. De Vuyst, M. Peybernes, R. Poncet, R. Motte, Building a more efficient Lagrange-remap scheme thanks to performance modeling, ECCOMAS 2016, Paper P12210, Proc. of the Conference ECCOMAS 2016, Minisymposium “MS 414 - New trends in numerical methods for multi-material compressible fluid flows”, accepted (2016)
- F. De Vuyst, T. Gasc, R. Motte, M. Peybernes, R. Poncet, Lagrange-Flux Eulerian schemes for compressible multimaterial flows, ECCOMAS 2016, Paper E8851, Proc. of the Conference ECCOMAS 2016, Minisymposium “MS 414 - New trends in numerical methods for multi-material compressible fluid flows”, accepted (2016)
- F. De Vuyst, M. Béchereau, T. Gasc, R. Motte, M. Peybernes, R. Poncet, Stable and accurate low-diffusive interface capturing advection schemes, submitted to IJNMF, Special issue for the MULTIMAT 2015 Conf., accepted (2016)
- R. Poncet, M. Peybernes, T. Gasc and F. De Vuyst, Performance modeling of a compressible hydrodynamics solver on multicore CPUs, Proc. of the PARCO 2015 Conference, Edinburgh, to appear (2016)

Thank you