# Supercomputers, real and imagined

hpcgarage.org/groppfest


**Rich Vuduc**, Georgia Tech


For Bill Gropp, in memory of **Hormozd Gahvari** (1981–2016)

PRESENTED TO:

ICERM — September 13 & 14, 2025

doi:10.1016/B978-044482851-4.50030-X

Parallel Computational Fluid Dynamics
Towards Teraflops, Optimization and Novel Formulations
D. Keyes, A. Ecer, J. Periaux, N. Satofuka and P. Fox (Editors)



# Towards Realistic Performance Bounds for Implicit CFD Codes

W. D. Gropp,[a*] D. K. Kaushik,[b†] D. E. Keyes,[c‡] and B. F. Smith[d§]

[a]Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, gropp@mcs.anl.gov.

[b]Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439 and Computer Science Department, Old Dominion University, Norfolk, VA 23529, kaushik@cs.odu.edu.

[c]Mathematics & Statistics Department, Old Dominion University, Norfolk, VA 23529, ISCR, Lawrence Livermore National Laboratory, Livermore, CA 94551, and ICASE,

Finally, we store the $N$ output vector elements. This leads to the following estin the data volume:

$$\begin{aligned}\text{Total Bytes Transferred} &= m * \text{sizeof\_int} + 2 * m * N * \text{sizeof\_double} \\ &\quad + nz * (\text{sizeof\_int} + \text{sizeof\_double}) \\ &= 4 * (m + nz) + 8 * (2 * m * N + nz).\end{aligned}$$

This gives us an estimate of the bandwidth required in order for the processo: $2 * nz * N$ flops at the peak speed:

$$\text{Bytes Transferred/fmadd} = \left(16 + \frac{4}{N}\right)\frac{m}{nz} + \frac{12}{N}.$$

Alternatively, given a memory performance, we can predict the maximum ach performance. This results in

$$M_{BW} = \frac{2}{(16 + \frac{4}{N})\frac{m}{nz} + \frac{12}{N}} \times BW,$$

where $M_{BW}$ is measured in Mflops/sec and $BW$ stands for the available memory width in Mbytes/s, as measured by STREAM [11] benchmark. (The raw bandwidtl

What **should** we build?

What **will** we build?

What **could** we build?

# What **should** we build?

# Hormozd Gahvari (1981–2016)

Beginning around 2010—the predawn of exascale—Hormozd, Bill, and colleagues wrote a series speculating on the "speeds and feeds" necessary to build machines that could achieve 1 EF/s on "classical" scientific kernels, like FFTs, FMMs, and AMG, among others.

PRESENTED TO:

ICERM — September 13-14, 2025


With Hormozd Gahvari at SC'15

# IPDPS'10

## An Introductory Exascale Feasibility Study for FFTs and Multigrid

Hormozd Gahvari and William Gropp
*Computer Science Department*
*University of Illinois at Urbana-Champaign*
*{gahvari,wgropp}@illinois.edu*

**Abstract**

*he coming decade is going to see a push towards scale computing. Assuming gigahertz cores, this ans exascale systems will have between 100 million 1 billion of them to achieve this level of performance. At this scale, some important questions need e answered on the applications end. What applications are feasible at this scale? What needs to be e to make them scalable? How does the hardware e to adapt to meet application needs? In this paper, introduce a new feasibility-based approach to answering these questions. Our approach involves finding er and lower bounds on problem size and machine ameters to determine a feasibility region for the lication in question. As the underlying architecture future exascale machine is currently unknown, we LogP-based performance models and vary machine ameters to give architecture-indepenent hardware straints. We consider both strong-scaling and weak-ling scenarios, and present results for two applications, the Fast Fourier Transform and basic geometric ltigrid. The results show substantial constraints that d to be satisfied to enable exascale performance.*

## Introduction

With the recent realization of petascale computing, ntion has now turned towards the next step, the scale. An exascale machine is one that will be able of performing $10^{18}$ operations per second. It expected that an exascale computer will require dreds of millions to billions of processor cores, make use of new technologies and perhaps novel itectures [1]. This is a huge jump from the mahes of today, so the question of which algorithms applications would scale to an exascale machine is ertinent one. Scientists and engineers need to know ich algorithms they should use on these machines. plication programmers need to know on which algorithms they should focus. Hardware and syst software designers need to know for which applicati to optimize. In this paper, we look at the FFT multigrid because they are popular algorithms which there are differing issues that cause scalabi concerns that we wish to highlight.

The rest of the paper is organized as follows. Sect 2 explains the overall approach to our feasibility stu Sections 3 and 4 examine the exascale feasibility of FFT and basic geometric multigrid, respectively. S tion 5 summarizes the results and lays out directi for future work.

## 2. Approach to Studying Feasibility

The main challenge in studying the feasibility applications at the exascale is that the specific des and machine parameters of a future exascale syst are far from known. There have been studies but no specific designs as of yet. So we can straightforwardly develop performance models, plug machine parameters, and make an easy determinat of feasibility.

What we can do, though, is treat the mach parameters as variables and see what is the range values they can take such that exascale performa is possible. Specifically, we adjust communicati related parameters such as latency and bandwi assuming the application is running on a hypothet exascale machine with between 100 million anc billion gigahertz cores. Our hypothetical machine have $2^{28}$ (almost 268.5 million) cores with each c having a compute time per floating-point operation $t_c = 0.1$ ns. This translates to a peak performa of 2.68 ExaFLOPS. We also vary the problem size consider both strong-scaling (smaller problems that being solved today) and weak-scaling (larger proble that will be solvable with the increased proces count) scenarios.

244-6443-2/10/$26.(

doi:10.1109/IPDPS.2010.5470417



# IPDPS'11

2011 IEEE International Parallel & Distributed Processing Symposium

## Architectural constraints to attain 1 Exaflop/s for three scientific application classes

Abhinav Bhatele, Pritish Jetley, Hormozd Gahvari, Lukasz Wesolowski, William D. Gropp, Laxmikant V. Kalé
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
E-mail: {bhatele, pjetley2, gahvari, wesolwsk, wgropp, kale}@illinois.edu

***Abstract*— The first Teraflop/s computer, the ASCI Red, came operational in 1997, and it took more than 11 years a Petaflop/s performance machine, the IBM Roadrunner, appear on the Top500 list. Efforts have begun to study hardware and software challenges for building an exascale chine. It is important to understand and meet these challenges order to attain Exaflop/s performance. This paper presents feasibility study of three important application classes to mulate the constraints that these classes will impose on the chine architecture for achieving a sustained performance of 1 aflop/s.**

**The application classes being considered in this paper are – ssical molecular dynamics, cosmological simulations and unuctured grid computations (finite element solvers). We analyze problem sizes required for representative algorithms in each ss to achieve 1 Exaflop/s and the hardware requirements in ms of the network and memory. Based on the analysis for ieving an Exaflop/s, we also discuss the performance of these orithms for much smaller problem sizes.**

***Keywords*-application scalability; exascale; performance anal-s; molecular dynamics; cosmology; finite element methods**

### I. Introduction

Parallel supercomputers have kept up the pace of peak rformance improvement: The first peak Petaflop/s machine, adrunner, appeared on the Top500 [1] list in June 2008, d multiple systems beyond that performance level have been nned for near future. The community has set a goal of ilding an Exaflop/s machine by 2018. There are several hardre challenges to be overcome before we break the Exaflop/s rrier - power/energy costs, memory costs, communication d others. The continuous frequency increase that we enjoyed the past has come to an end. In part due to this, it has been ear that a co-design approach, where machines are designed conjunction with exascale applications will be needed to hieve the goal of an Exaflop/s by 2018 [2].

Assuming that we can overcome the hardware challenges d an Exaflop/s machine is built, scientists will have to odify/develop algorithms and applications that scale to excale. To this end, we analyze three prevalent application sses that currently occupy a significant portion of compute cles on various supercomputers (supported by INCITE and AC allocation awards) – classical molecular dynamics, smological simulations and unstructured mesh computations nite element solvers).

Goals arising from the science involved suggest that the sc entific communities using these applications will need exasca performance, so it is important to project the performance these applications on an exascale machine.

These three application classes encompass some of the mo common parallel data structures, including structured gric unstructured grids and particles ($N$-body). Between the thr chosen classes, a range of computational and communicati patterns are covered which should provide insight into t scaling challenges we will face on the road to exascale. V consider weak scaling of these applications to the full size the machine. At exascale, scientifically important objectiv may also involve studying problems smaller than what wea scaling suggests (i.e. 1000 times larger problems compar with those on petascale). Therefore, we also study performan issues for smaller problem instances.

The first class of applications chosen for the study a molecular dynamics (MD) applications that focus on t simulation of biomolecular systems. Several highly scali MD codes are used today on supercomputers – NAMD [ AMBER [4], Gromacs [5], Desmond [6] and Blue Matter [ MD simulations involve calculation of forces on a syste of $N$ atoms. We discuss different parallelization strategi for the force calculation and select the one with the lowe computation to communication ratio. For the purposes of tl study, we consider only short-range calculations (also referr to as Lennard-Jones dynamics).

The second class of applications are cosmological simul tions. These applications constitute another important catego with a unique communication pattern. Gravitational solve for the $N$-body problem use one of many different methoc direct sum, tree-methods, particle-mesh methods and hybr codes. Some examples of cosmology codes are PkdGRAV [ ChaNGa [9], Enzo [10] and FLASH [11]. We consider tr methods for solving the $N$-body problem for our analysis a set aside hydrodynamics for a later study.

Unstructured grid problems, the third class under conside ation, arise frequently in science and engineering. Many pro lem domains have complex shapes that do not lend themselv well to a simple finite difference discretization. Setting t problem as an unstructured grid, which involves breaking t domain into triangles (in 2D) and tetrahedra (in 3D), allov for complicated domains to be discretized in a straightforwa

-2075/11 $26.00 © 2011 IEEE
10.1109/IPDPS.2011.18





doi:10.1109/IPDPS.2011.18

# JPDC'19

Journal of Parallel and Distributed Computing 136 (2020) 63–74

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

## FFT, FMM, and multigrid on the road to exascale: Performance challenges and opportunities

Huda Ibeid *, Luke Olson, William Gropp
*University of Illinois at Urbana-Champaign, Urbana, IL 61801, United States of America*

ARTICLE INFO

*Article history:*
Received 31 July 2018
Received in revised form 28 May 2019
Accepted 23 September 2019
Available online 9 October 2019

*Keywords:*
Fast Fourier transform
Fast multipole method
Multigrid
Exascale
Performance modeling

ABSTRACT

FFT, FMM, and multigrid methods are widely used fast and highly scalable so However, emerging large-scale computing systems are introducing challenges in petascale computers. Recent efforts (Dongarra et al. 2011) have identified seve design of exascale software that include massive concurrency, resilience manag high performance of heterogeneous systems, energy efficiency, and utilizing complex memory hierarchy expected at exascale. In this paper, we perform a mo of the FFT, FMM, and multigrid methods in the context of these projected cons use performance models to offer predictions about the expected performance system configurations based on current technology trends.



### 1. Introduction

Elliptic PDEs arise in many applications in computational science and engineering. Classic examples are found in computational astrophysics, fluid dynamics, molecular dynamics, plasma physics, and many other areas. The rapid solution of elliptic PDEs remains of wide interest and often represents a significant portion of simulation time.

The fast Fourier transform (FFT), the fast multipole method (FMM), and multigrid methods (MG) are widely used fast and highly scalable solvers for elliptic PDEs. The FFT, FMM, and MG methods have been used in a wide variety of scientific computing applications such as particle-in-cell methods, the calculation of long-range (electrostatic) interactions in many-particle systems, such as molecular dynamics and Monte Carlo sampling [3], and in signal analysis. The performance expectations of these methods helps guide algorithmic changes and optimizations to enable migration to exascale systems, as well as to help identify potential bottlenecks in exascale architectures. In addition, modeling helps assess the trade-offs at extreme scales, which can assist in choosing optimal methods and parameters for a given application and specific machine architecture.

Each method has advantages and disadvantages, and all have their place as PDE solvers. Generally, the FFT is used for uniform discretizations, FMM and geometric MG are efficient solvers on irregular grids with local features or discontinuities, and algebraic MG can handle arbitrary geometries, variab general boundary conditions. The focus of t FFT, FMM, and *geometric* MG, although several ob an algebraic setting as well [7].

One aim of the International Exascale Sof is to enable the development of applicatio full performance of exascale computing platf these exascale platforms are not yet fully sp believed that they will require significant ch hardware architecture relative to the curren The IESP roadmap reports that technology tr constraints on the design of an exascale softw expected to affect system software and app are summarized as

**Concurrency:** Future supercomputing perfo mainly on increases in system scale. one million or more for current system cale systems are likely to incorporate o cores, assuming GHz technology. As a increase in concurrency necessitates computing for large-scale scientific ap extrapolated scalability.

**Resiliency:** The exponential increase in core exascale will lead to increases in the switches, interconnects, and r Consequently, resilience will be a chal cations on future exascale systems.

**Heterogeneity:** As accelerators advance in and energy efficiency, heterogeneity h

* Corresponding author.
*E-mail addresses:* hibeid@illinois.edu (H. Ibeid), lukeo@illinois.edu (L. Olson), wgropp@illinois.edu (W. Gropp).

https://doi.org/10.1016/j.jpdc.2019.09.014





doi:10.1016/j.jpdc.2019.09.014

doi:10.1109/IPDPS.2011.18

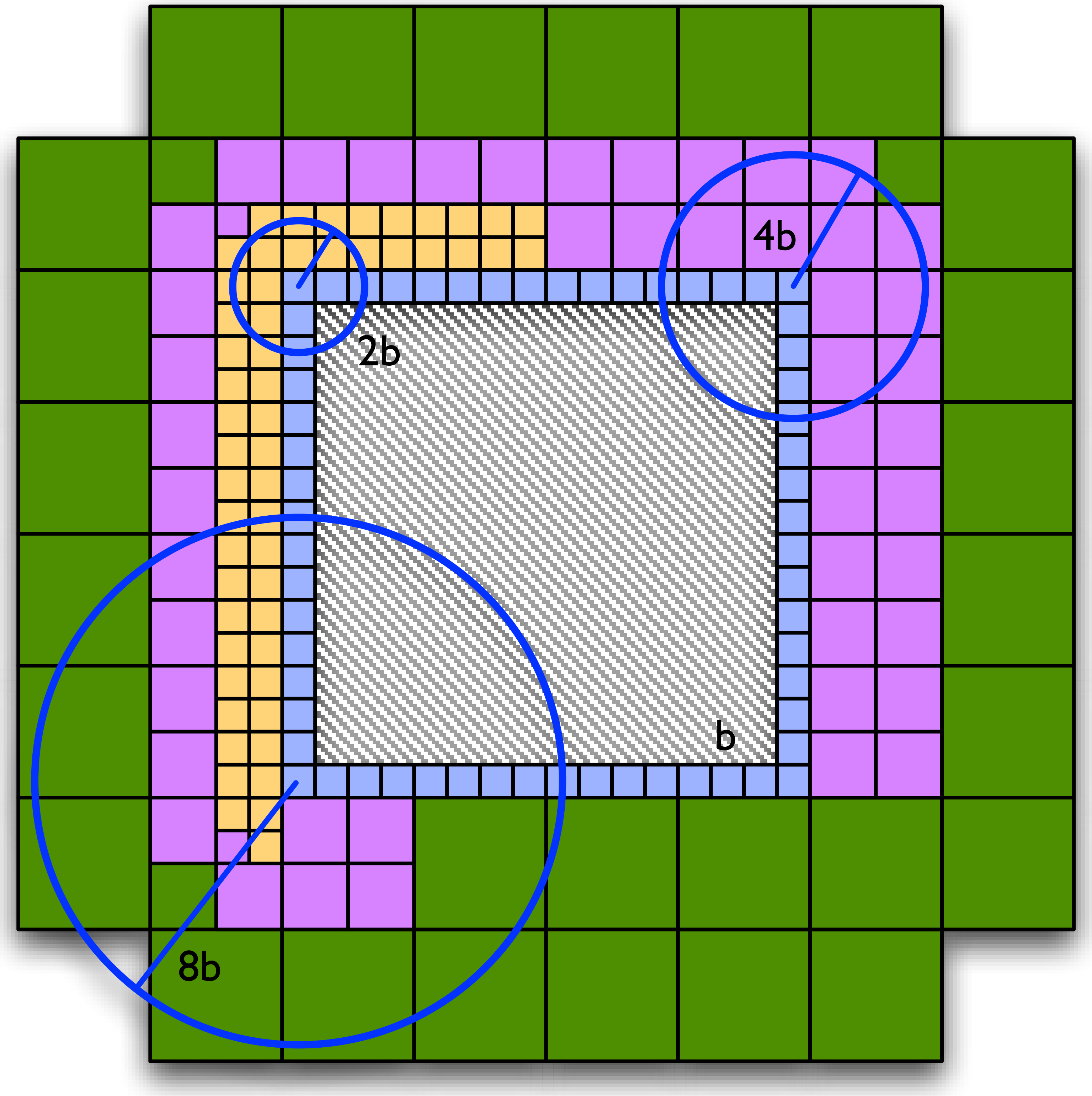



Fig. 4. Communication pattern of a single node at the bottom three de
in the Barnes-Hut tree. The striped region in the center represents the c
subvolume of particles assigned to the node, and the immediate squ
surrounding it represent the buckets along its faces. Progressively la
squares represent remote cells at different depths that are requested by
node for $\theta_T = 0.5$. Circles of radii $2b$, $4b$ and $8b$ described arounc
centers of corner buckets determine which cells are requested.

the amount of communication generated per processor by
expansion of higher-level cells as follows:

$$C_2^{\text{cell}} = 31\left(\frac{\lg P_n}{3} - 1\right) \quad \text{cells}$$

doi:10.1109/IPDPS.2011.18

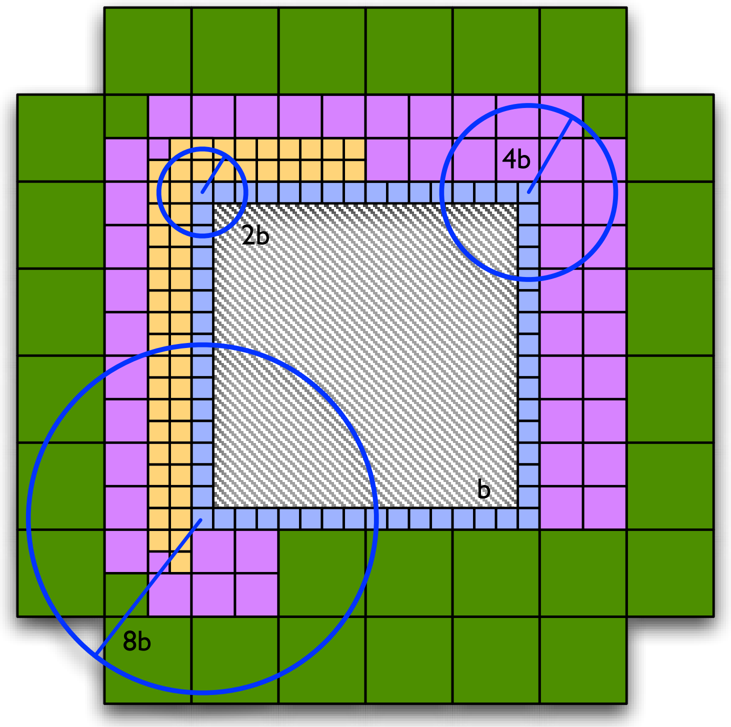



Fig. 4. Communication pattern of a single node at the bottom three de
in the Barnes-Hut tree. The striped region in the center represents the c
subvolume of particles assigned to the node, and the immediate squ
surrounding it represent the buckets along its faces. Progressively la
squares represent remote cells at different depths that are requested by
node for $\theta_T = 0.5$. Circles of radii $2b$, $4b$ and $8b$ described arounc
centers of corner buckets determine which cells are requested.

the amount of communication generated per processor by
expansion of higher-level cells as follows:

$$C_2^{\text{cell}} = 31 \left( \frac{\lg P_n}{3} - 1 \right) \quad \text{cells}$$

The expansion of each cell yields eight children. We assu
that for each expanded cell, a single message is genera
which contains all its children. This model may be exten

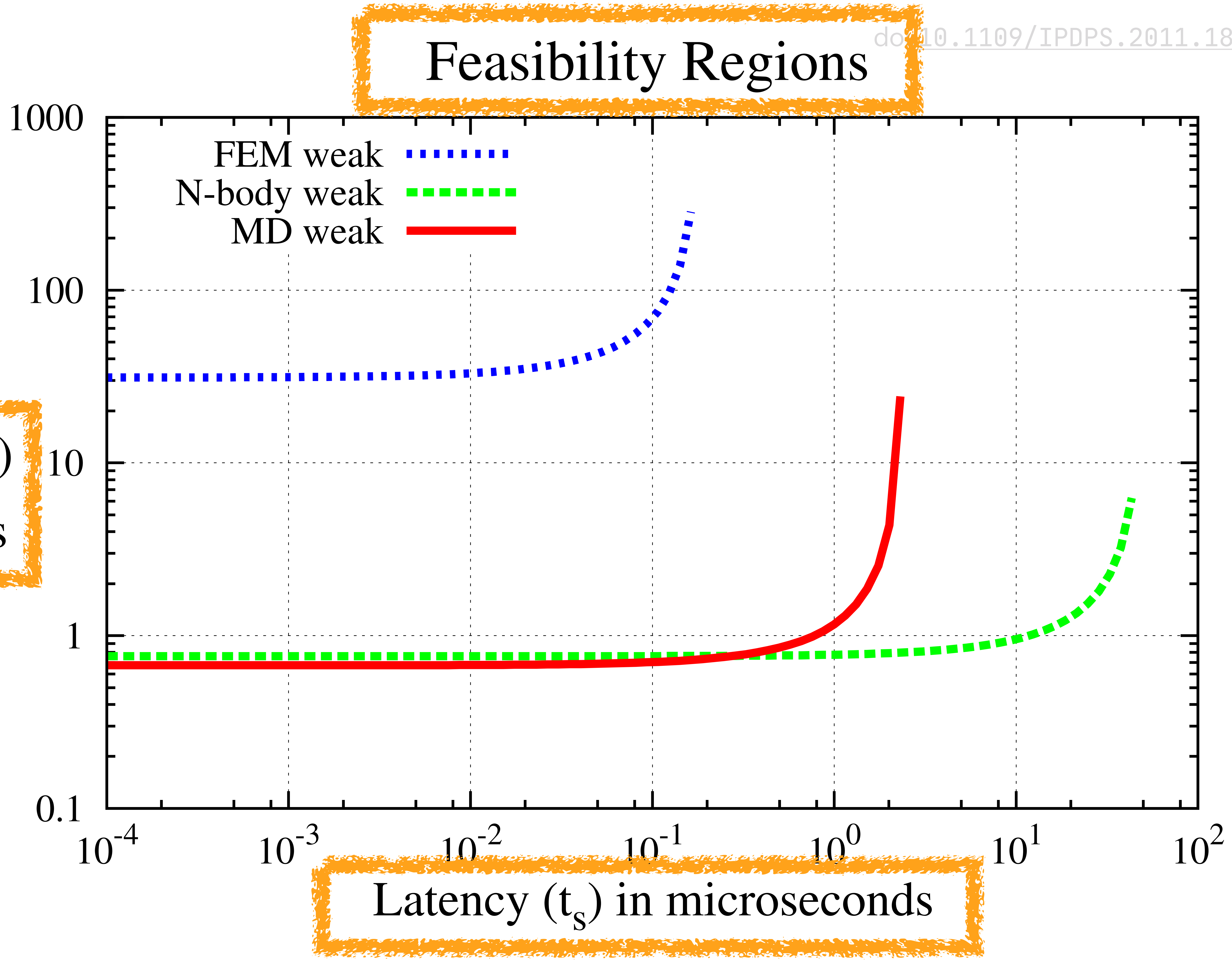



Fig. 11. Feasibility regions for molecular dynamics (MD), cosmology (N-body) and finite element solvers (FEM) for weak scaling to achieve 1 Exaflop/s

Figure 3. Feasibility curves in $L$ and $g$ for the 3D FFT.

doi:10.1109/IPDPS.2010.5470417

communication and computation, and a more complicated model in which there is substantial overlap of communication and computation.

**3.2.1. No Overlap Model.** If we do not consider overlap of communication, we get that each processor computes its portion of the data, and during each communication round has to communicate with $p$ other processors. The corresponding expression for the runtime of the 3D FFT using the LogP performance model is

$$T = t_c \frac{N}{P} \log_2 N + 2(p-1)(L+o) + 2(p-2)g$$

Note that we do not do any latency-hiding, because we treat the latency here as the cost to send the entire message, not just the first word.

**3.2.2. Overlap Model.** Now allowing overlap of communication and computation, we set up another performance model, using instead of LogP the LogGP model [10] which extends it by adding a bandwidth term $G$ that represents a per-unit cost of transferring data over the network. The model assumes that one $n \times \frac{n}{p}$ sheet is computed at a time, with communication of each sheet occurring after its computation,

in a
comp
of the
the cc
comn
with
the cc
expre

$T =$

In the
It cor
the re
It i
one r
some
For si

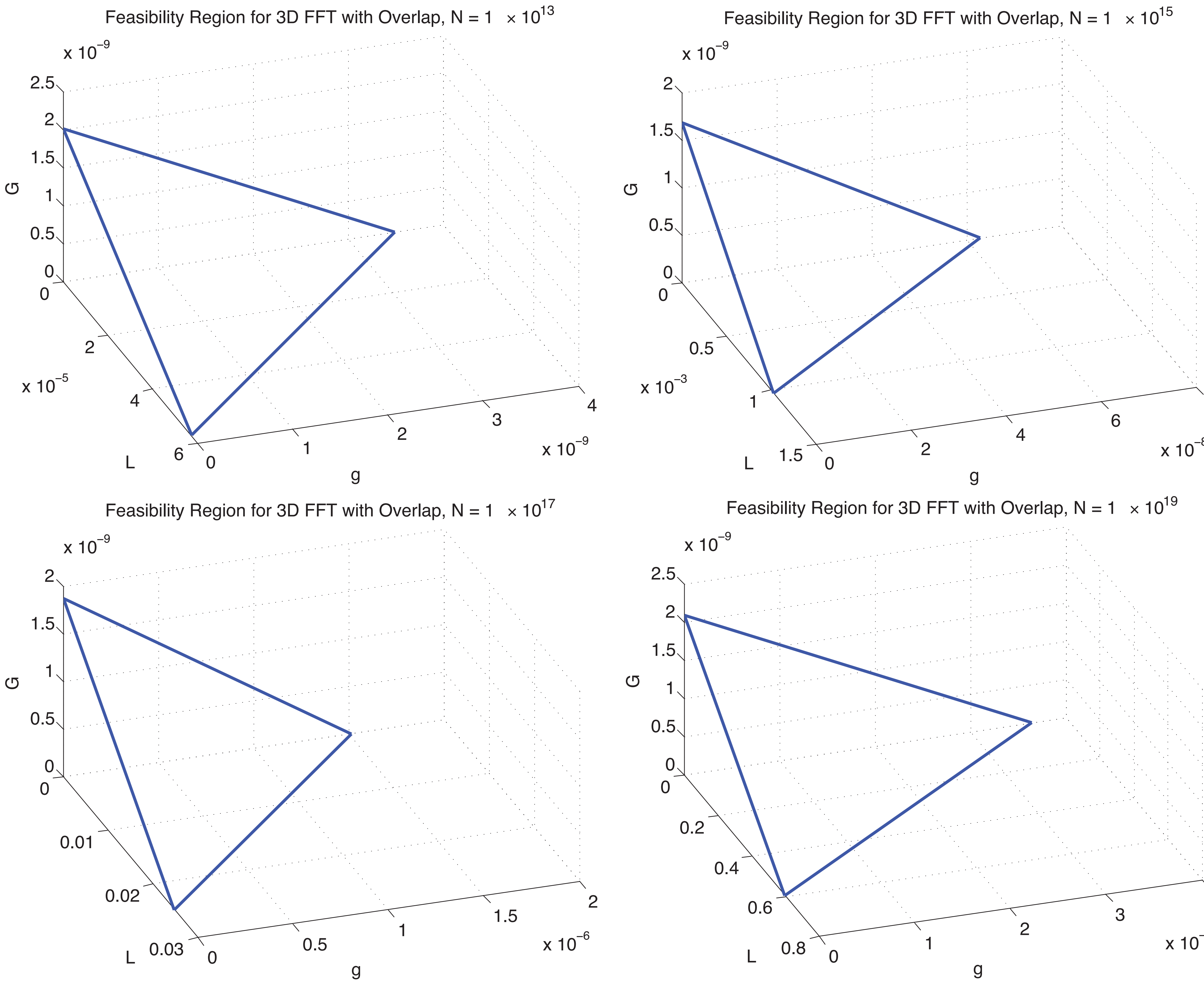



Figure 4. Feasibility regions for FFT with overlap of communication and computatior

communication and computation, and a more complicated model in which there is substantial overlap of communication and computation.

**3.2.1. No Overlap Model.** If we do not consider overlap of communication, we get that each processor computes its portion of the data, and during each communication round has to communicate with $p$ other processors. The corresponding expression for the runtime of the 3D FFT using the LogP performance model is

$$T = t_c \frac{N}{P} \log_2 N + 2(p-1)(L+o) + 2(p-2)g$$

Note that we do not do any latency-hiding, because we treat the latency here as the cost to send the entire message, not just the first word.

**3.2.2. Overlap Model.** Now allowing overlap of communication and computation, we set up another performance model, using instead of LogP the LogGP model [10] which extends it by adding a bandwidth term $G$ that represents a per-unit cost of transferring data over the network. The model assumes that one $n \times \frac{n}{p}$ sheet is computed at a time, with communication of each sheet occurring after its computation,

Figure 3. Feasibility curves in $L$ and $g$ for the 3D FFT.

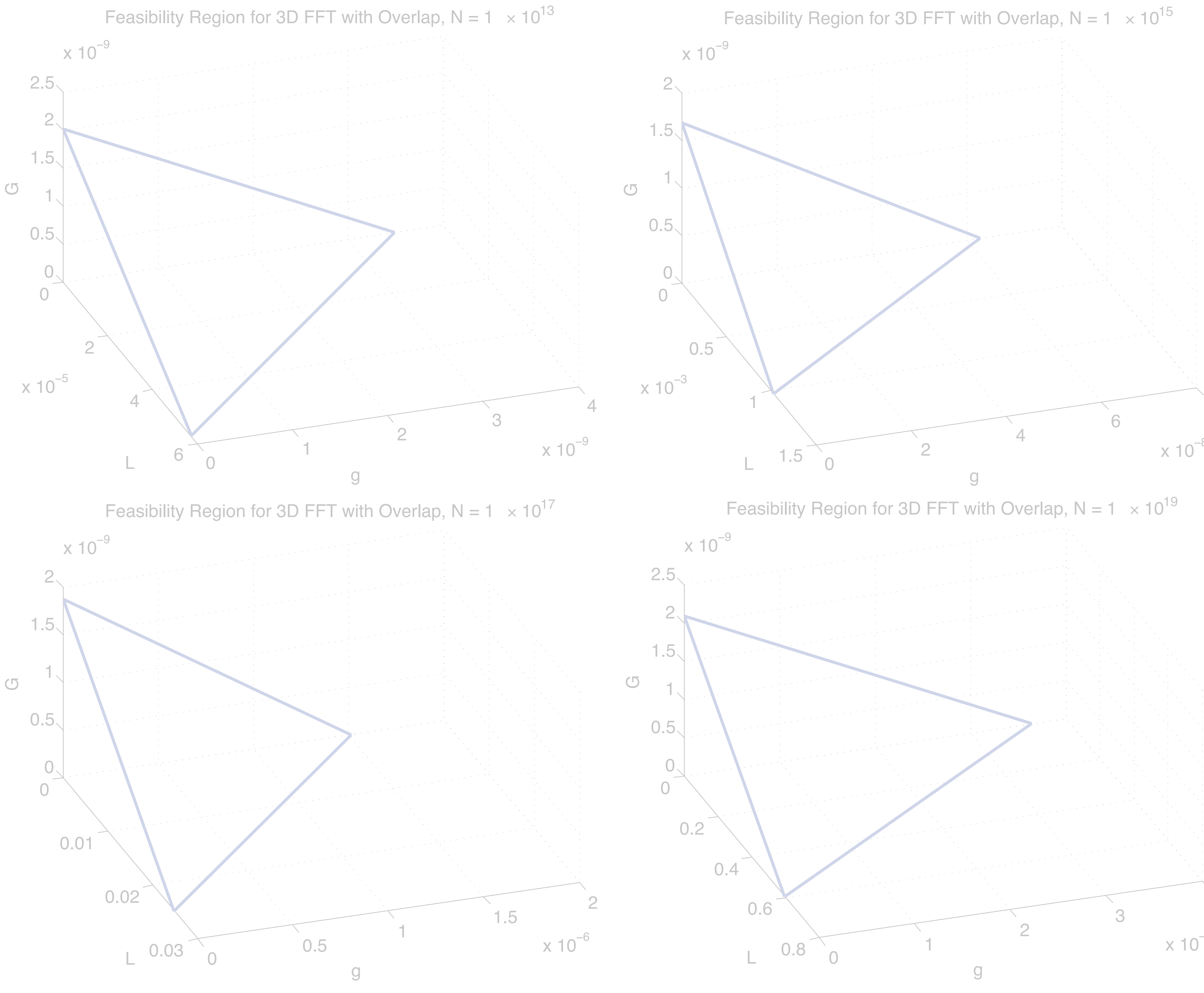



Figure 4. Feasibility regions for FFT with overlap of communication and computation

communication and computation, and a more complicated model in which there is substantial overlap of communication and computation.

**3.2.1. No Overlap Model.** If we do not consider overlap of communication, we get that each processor computes its portion of the data, and during each communication round has to communicate with $p$ other processors. The corresponding expression for the runtime of the 3D FFT using the LogP performance model is

$$T = t_c \frac{N}{P} \log_2 N + 2(p-1)(L+o) + 2(p-2)g$$

Note that we do not do any latency-hiding, because we treat the latency here as the cost to send the entire message, not just the first word.

**3.2.2. Overlap Model.** Now allowing overlap of communication and computation, we set up another performance model, using instead of LogP the LogGP model [10] which extends it by adding a bandwidth term $G$ that represents a per-unit cost of transferring data over the network. The model assumes that one $n \times \frac{n}{p}$ sheet is computed at a time, with communication of each sheet occurring after its computation,

$$\frac{\mathbf{a}+\mathbf{b}}{\max(\mathbf{a},\mathbf{b})} \leq \mathbf{2}$$

**(So who cares? Let's revisit later...)**

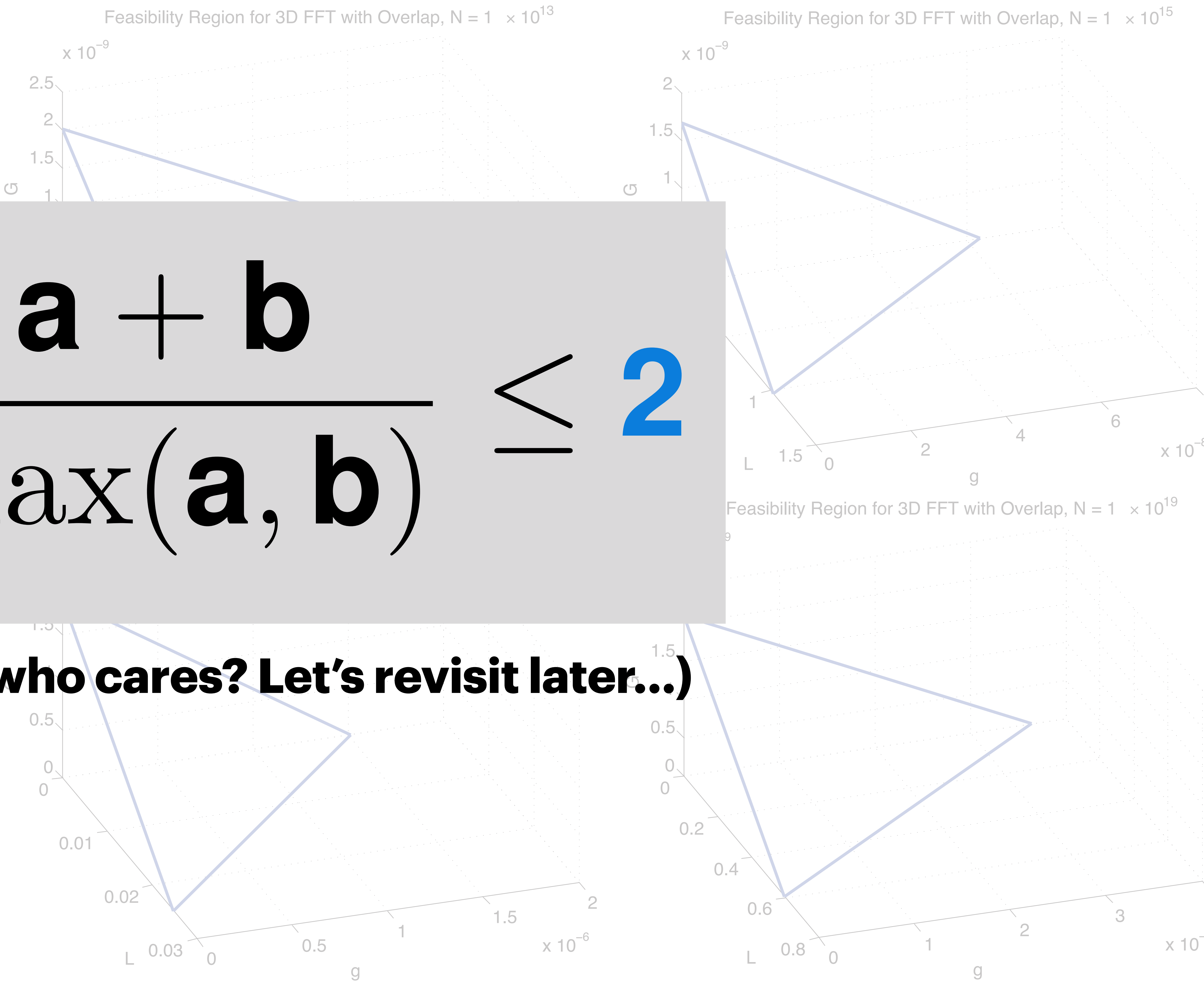



Figure 4. Feasibility regions for FFT with overlap of communication and computation

doi:10.1016/j.jpdc.2019.09.014

# Projection

(ca. 2018)

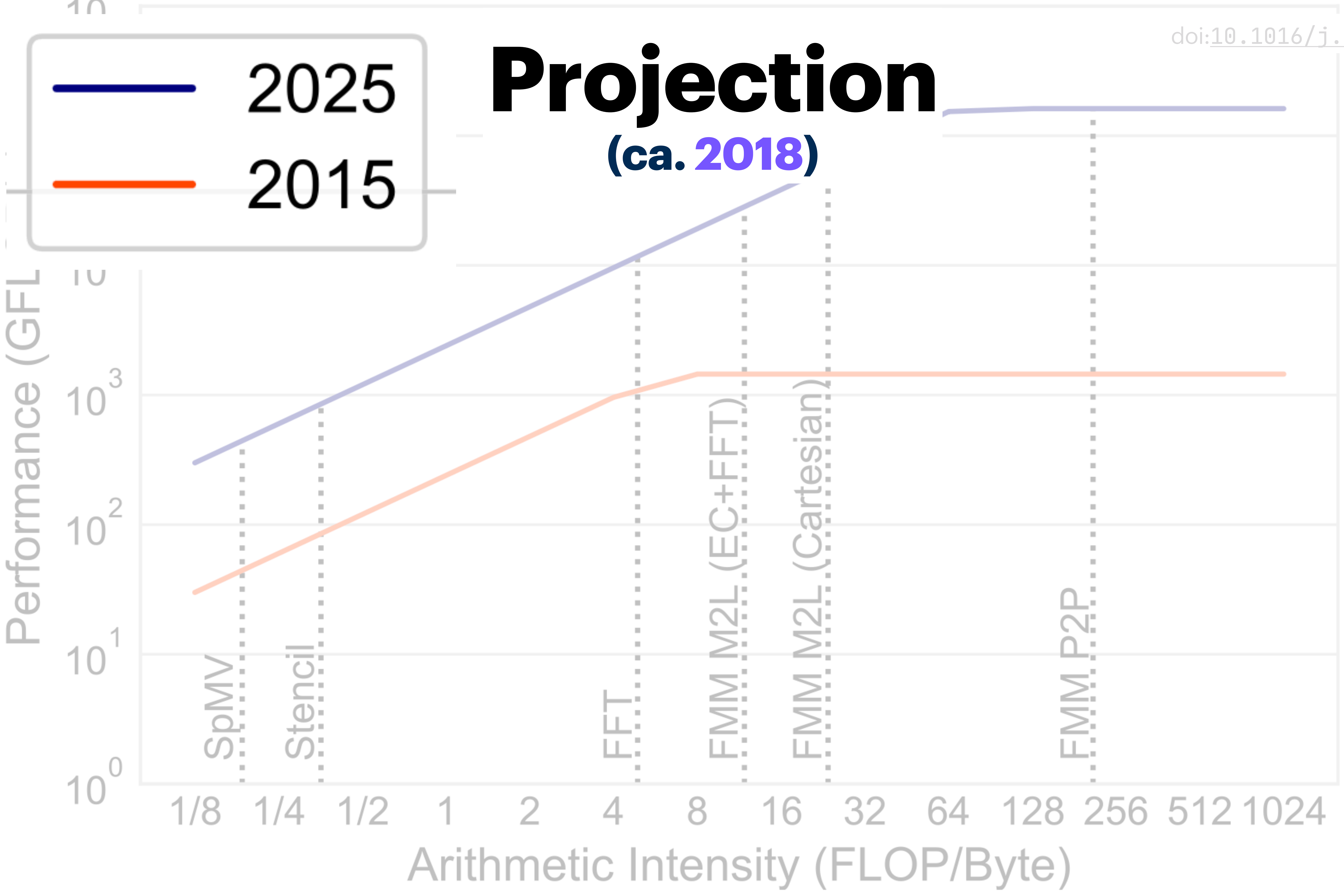
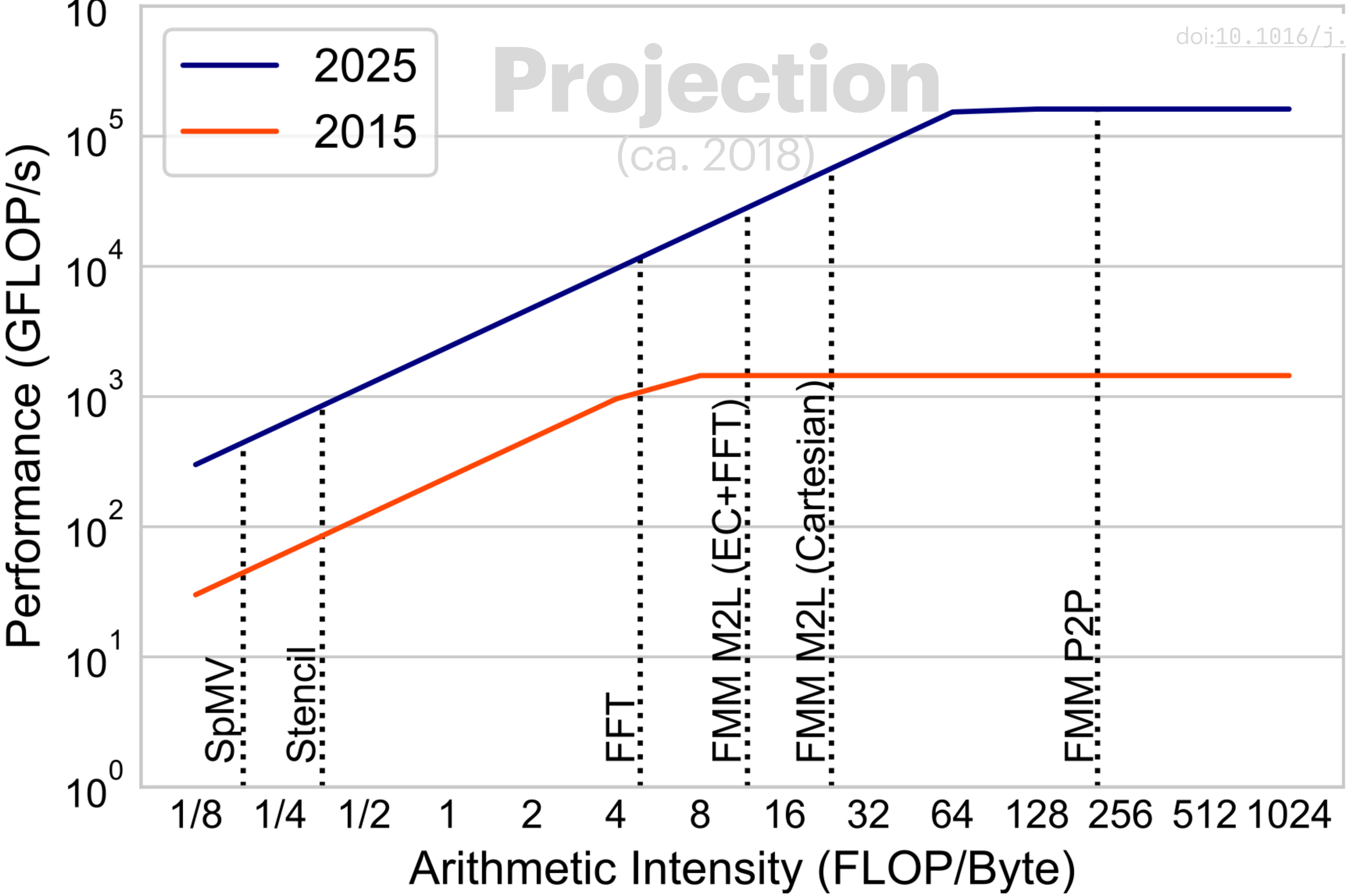



**Fig. 9.** Roofline model of NVIDIA Tesla GPU and computation intensity of various




doi:10.1016/j.jpdc.2019.09.014

**Fig. 9.** Roofline model of NVIDIA Tesla GPU and computation intensity of various

doi:10.1016/j.jpdc.2019.09.014

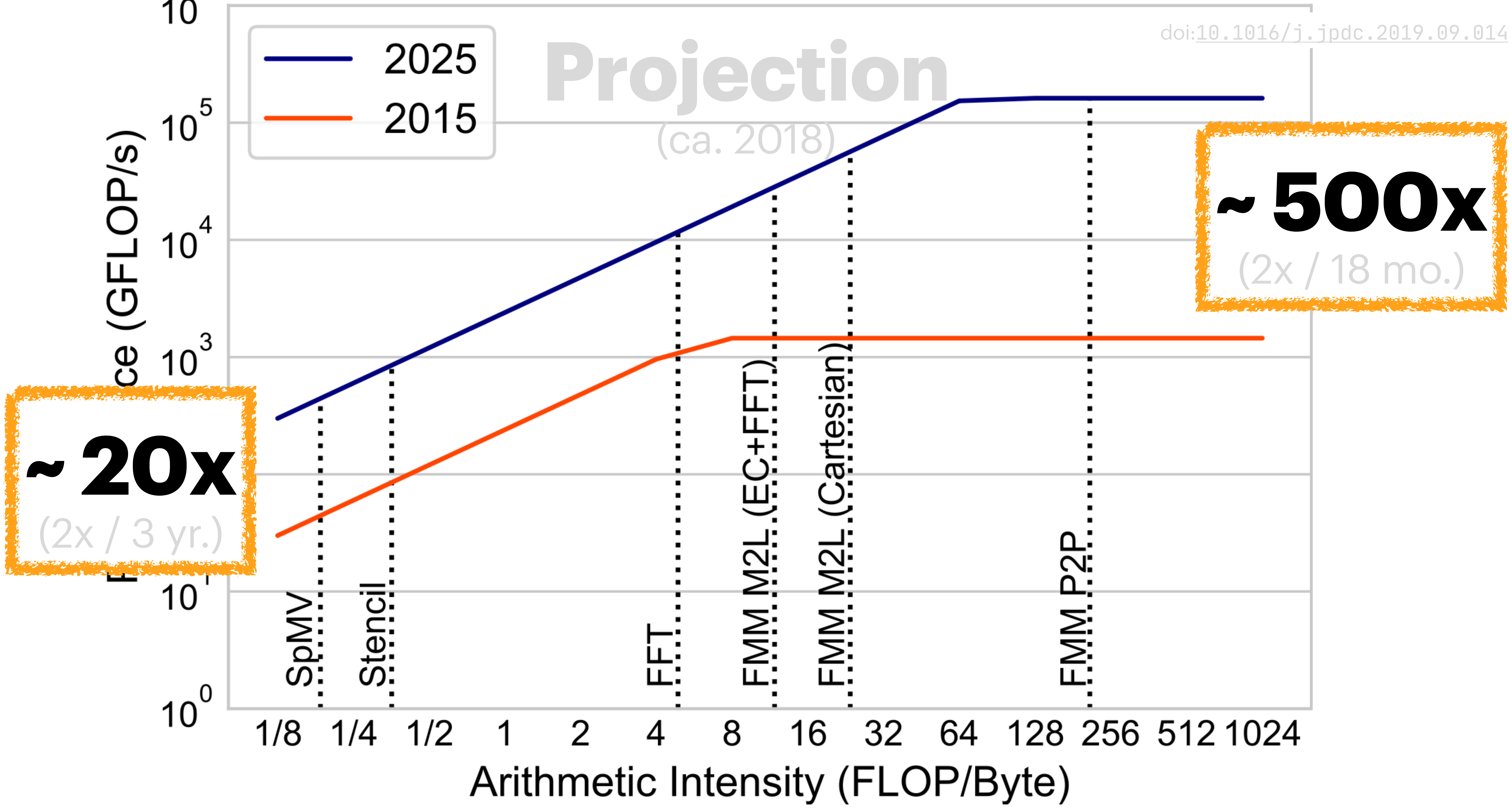



**Fig. 9.** Roofline model of NVIDIA Tesla GPU and computation intensity of various

# What **will** we build?

(That last roofline exemplifying a way to answer this question.)

# HPC "trickles down" to commerce

Gregory Abowd (GT). "Beyond Weiser: From ubiquitous computing to collective computing." DOI: 10.1109/MC.2016.22

## OUTLOOK

**TABLE 1.** A framework for comparing computing generations, inspired by Mark Weiser.

| Generation | Time frame | Human–computer ratio | Canonical device | Application | |
|---|---|---|---|---|---|
| | | | | Initial | Follow-on |
| 1 | Mid-1930s | Many–1 | Mainframe | Scientific calculation | Data processing |
| 2 | Late 1960s | 1–1 | PC | Spreadsheet | Database management, document processing |
| 3 | Late 1980s | 1–many | Inch/foot/yard | Calendar and contact management, human–human communication | Location-based services, social media, app ecosystem, education |
| 4 | Mid-2000s | Many–many | Cloud/crowd/shroud | Personal navigation and entertainment | Health advisors, educational assistants, supply chain logistics |

# HPC "trickles down" to commerce

Gregory Abowd (GT). "Beyond Weiser: From ubiquitous computing to collective computing." DOI: 10.1109/MC.2016.22

## OUTLOOK

**TABLE 1.** A framework for comparing computing generations, inspired by Mark Weiser.

| Generation | Time frame | Human–computer ratio | Canonical device | Application | |
|---|---|---|---|---|---|
| | | | | Initial | Follow-on |
| 1 | Mid-1930s | Many–1 | Mainframe | Scientific calculation | Data processing |
| 2 | Late 1960s | 1–1 | PC | Spreadsheet | Database management, document processing |
| 3 | Late 1980s | 1–many | Inch/foot/yard | Calendar and contact management, human–human communication | Location-based services, social media, app ecosystem, education |
| 4 | Mid-2000s | Many–many | Cloud/crowd/shroud | Personal navigation and entertainment | Health advisors, educational assistants, supply chain logistics |

# HPC "trickles down" to commerce

Gregory Abowd (GT). "Beyond Weiser: From ubiquitous computing to collective computing." DOI: 10.1109/MC.2016.22

## OUTLOOK

**TABLE 1.** A framework for comparing computing generations, inspired by Mark Weiser.

| Generation | Time frame | Human–computer ratio | Canonical device | Application: Initial | Application: Follow-on |
|---|---|---|---|---|---|
| 1 | Mid-1930s | Many–1 | Mainframe | Scientific calculation | Data processing |
| 2 | Late 1960s | 1–1 | PC | Spreadsheet | Database management, document processing |
| 3 | Late 1980s | 1–many | Inch/foot/yard | Calendar and contact management, human–human communication | Location-based services, social media, app ecosystem, education |
| 4 | Mid-2000s | Many–many | Cloud/crowd/shroud | Personal navigation and entertainment | Health advisors, educational assistants, supply chain logistics |

# HPC "trickles down" to commerce

Gregory Abowd (GT). "Beyond Weiser: From ubiquitous computing to collective computing." DOI: 10.1109/MC.2016.22

| n | Time frame | Human–computer ratio | Car |
|---|---|---|---|
| | Mid-1930s | Many–1 | Mai |
| | Late 1960s | 1–1 | PC |
| | Late 1980s | 1–many | Inch |
| | Mid-2000s | Many–many | Clo |

OUTLO

tions, inspired by Mark Weiser.

| Generati | Application: Initial | Application: Follow-on |
|---|---|---|
| 1 | c calculation | Data processing |
| 2 | heet | Database management, document processing |
| 3 | r and contact ment, human– communication | Location-based services, social media, app ecosystem, education |
| 4 | l navigation and nment | Health advisors, educational assistants, supply chain logistics |

$$\lim_{t \to \infty} (\text{ratio}) = \mathbf{0} \text{ humans : machines}$$

OUTLO

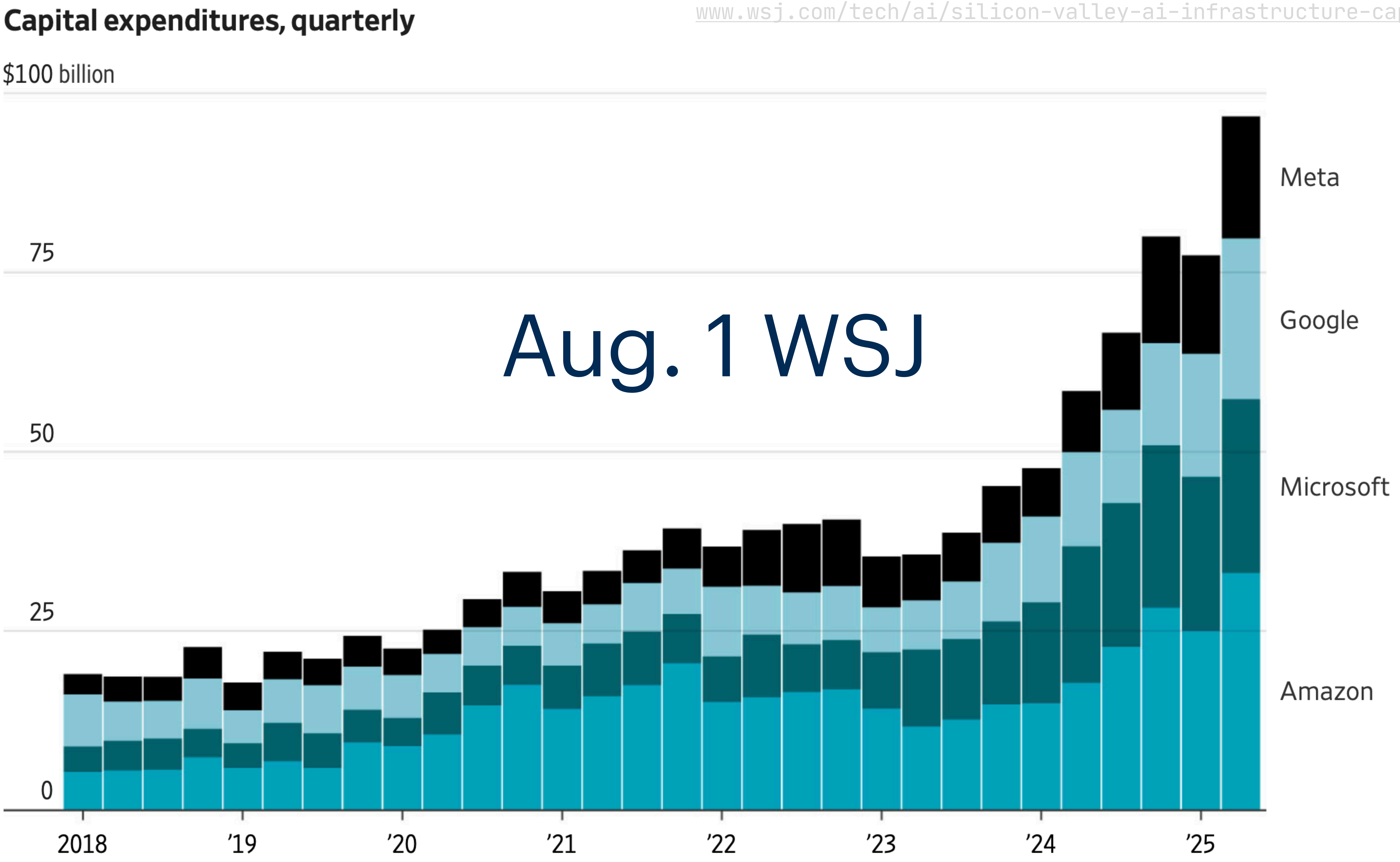
| n | Time frame | Human–computer ratio | Car |
|---|---|---|---|
| | Mid-1930s | Many–1 | Mai |
| | Late 1960s | 1–1 | PC |
| | Late 1980s | 1–many | Inch |
| | Mid-2000s | Many–many | Clou |

tions, inspired by Mark Weiser.

| Generati | Application | |
|---|---|---|
| | Initial | Follow-on |
| 1 | c calculation | Data processing |
| 2 | heet | Database management, document processing |
| 3 | r and contact ment, human– ommunication | Location-based services, social media, app ecosystem, education |
| 4 | l navigation and nment | Health advisors, educational assistants, supply chain logistics |

# Follow the **money**.

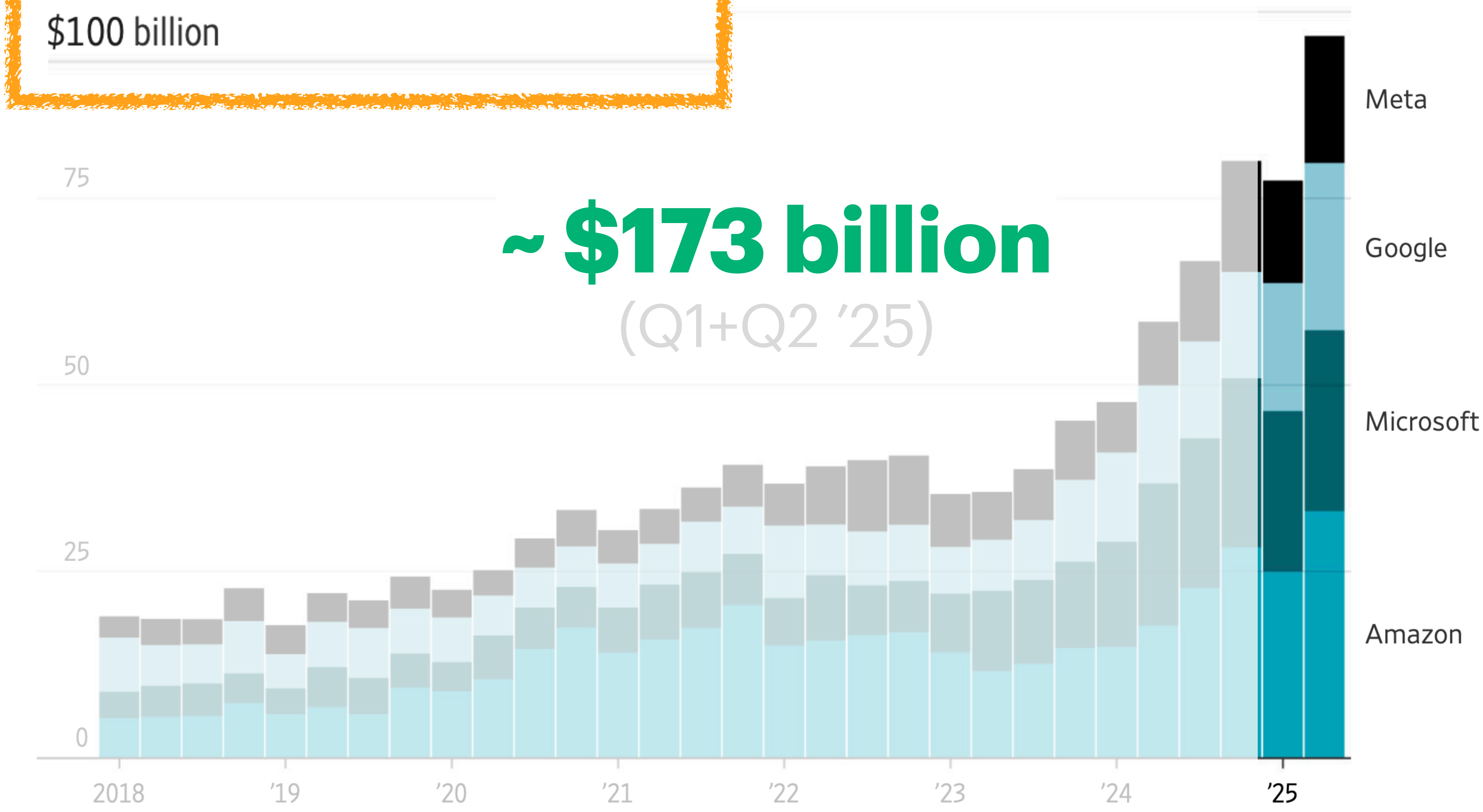
www.wsj.com/tech/ai/silicon-valley-ai-infrastructure-capex-cffe0431




# Aug. 1 WSJ

Note: Data are for calendar quarters and include finance leases.
Source: the companies

# Capital expenditures, quarterly




Note: Data are for calendar quarters and include finance leases.
Source: the companies

Investor and tech pundit Paul Kedrosky says that, as a percentage of gross domestic product, spending on AI infrastructure has already exceeded spending on telecom and internet infrastructure from the dot-com boom—and it's still growing. He also argues that one explanation for the U.S. economy's ongoing strength, despite tariffs, is that spending on IT infrastructure is so big that it's acting as a sort of private-sector stimulus program.

Capex spending for AI contributed more to growth in the U.S. economy in the past two quarters than *all of consumer spending*, says Neil Dutta, head of economic research at Renaissance Macro Research, citing data from the Bureau of Economic Analysis.

A global accounting of this infrastructure spending would be even bigger, as it would include capex from these companies' most important partners. Foxconn has recently spent big building out factories for Apple in India, which just supplanted China as the

| Rank | System | Cores | Rmax (PFlop/s) | Rpeak (PFlop/s) | Power (kW) |
|---|---|---|---|---|---|
| 1 | **El Capitan** - HPE Cray EX255a, AMD 4th Gen EPYC 24C 1.8GHz, AMD Instinct MI300A, Slingshot-11, TOSS, HPE<br>DOE/NNSA/LLNL<br>United States | 11,039,616 | 1,742.00 | 2,746.38 | 29,581 |
| 2 | **Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE Cray OS, HPE<br>DOE/SC/Oak Ridge National Laboratory<br>United States | 9,066,176 | 1,353.00 | 2,055.72 | 24,607 |
| 3 | **Aurora** - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel<br>DOE/SC/Argonne National Laboratory<br>United States | 9,264,128 | 1,012.00 | 1,980.01 | 38,698 |
| 4 | **JUPITER Booster** - BullSequana XH3000, GH Superchip 72C 3GHz, NVIDIA GH200 Superchip, Quad-Rail NVIDIA InfiniBand NDR200, RedHat Enterprise Linux, EVIDEN<br>EuroHPC/FZJ<br>Germany | 4,801,344 | 793.40 | 930.00 | 13,088 |
| 5 | **Eagle** - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft Azure<br>Microsoft Azure<br>United States | 2,073,600 | 561.20 | 846.84 | |

| Rank | System | Cores | Rmax (PFlop/s) | Rpeak (PFlop/s) | Power (kW) |
|---|---|---|---|---|---|
| 1 | **El Capitan** - HPE Cray EX255a, AMD 4th Gen EPYC 24C 1.8GHz, AMD Instinct MI300A, Slingshot-11, TOSS, HPE<br>DOE/NNSA/LLNL<br>United States | 11,039,616 | 1,742.00 | 2,746.38 | 29,581 |
| 2 | **Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE Cray OS, HPE<br>DOE/SC/Oak Ridge National Laboratory<br>United States | 9,066,176 | 1,353.00 | 2,055.72 | 24,607 |
| 3 | **Aurora** - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel<br>DOE/SC/Argonne National Laboratory<br>United States | 9,264,128 | 1,012.00 | 1,980.01 | 38,698 |
| 4 | **JUPITER Booster** - BullSequana XH3000, GH Superchip 72C 3GHz, NVIDIA GH200 Superchip, Quad-Rail NVIDIA InfiniBand NDR200, RedHat Enterprise Linux, EVIDEN | 4,801,344 | 793.40 | 930.00 | 13,088 |
| 5 | **Eagle** - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft Azure<br>Microsoft Azure<br>United States | 2,073,600 | 561.20 | 846.84 | |

# These are real supercomputers...

| Rank | System | Cores | Rmax (PFlop/s) | Rpeak (PFlop/s) | Power (kW) |
|---|---|---|---|---|---|
| 1 | **El Capitan** - HPE Cray EX255a, AMD 4th Gen EPYC 24C 1.8GHz, AMD Instinct MI300A, Slingshot-11, TOSS, HPE<br>DOE/NNSA/LLNL<br>United States | 11,039,616 | 1,742.00 | 2,746.38 | 29,581 |
| 2 | **Frontier** - HPE Cray EX235a EPYC 64C 2GHz, AMD Instir Cray OS, HPE<br>DOE/SC/Oak Ridge Nationa<br>United States | | | | |
| 3 | **Aurora** - HPE Cray EX - Inte CPU Max 9470 52C 2.4GHz, Slingshot-11, Intel<br>DOE/SC/Argonne National Laboratory<br>United States | | | | |
| 4 | **JUPITER Booster** - BullSequana XH3000, GH Superchip 72C 3GHz, NVIDIA GH200 Superchip, Quad-Rail NVIDIA InfiniBand NDR200, RedHat Enterprise Linux, EVIDEN | 4,801,344 | 793.40 | 930.00 | 13,088 |
| 5 | **Eagle** - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft Azure<br>Microsoft Azure<br>United States | 2,073,600 | 561.20 | 846.84 | |

🦅☁️

# Inside an Azure NDv5 compute node

| | |
|---|---|
| 2x 56c Intel Sapphire Rapids | Host CPUs |
| 2.0 TB DDR5-4800 | Host DRAM |
| 8x NVIDIA H100 / 80 GB HBM<br>8x AMD MI300X / 192 GB HBM | GPU options |
| 8x 3.84 TB E1.S NVMe<br>1x 960 GB M.2 NVMe<br>2x 1.92 TB M.2 NVMe | Local scratch<br>Boot disk<br>Service cache |
| 8x400G NDR InfiniBand<br>Microsoft 100G SmartNIC | Backend NICs<br>Frontend NIC |

Source: **Glenn Lockwood** (Microsoft) at the Smoky Mountains Conference in 2024

| | |
|---|---|
| 2x 56c Intel Sapphire Rapids | Host CPUs |
| 2.0 TB DDR5-4800 | Host DRAM |
| 8x NVIDIA H100 / 80 GB HBM<br>8x AMD MI300X / 192 GB HBM | GPU options |
| 8x 3.84 TB E1.S NVMe<br>1x 960 GB M.2 NVMe<br>2x 1.92 TB M.2 NVMe | Local scratch<br>Boot disk<br>Service cache |
| 8x400G NDR InfiniBand<br>Microsoft 100G SmartNIC | Backend NICs<br>Frontend NIC |

# Future supercomputers will be tuned for (cloud-based) LLM workloads.

# What will those look like?



# Calculon

## Co-designing supercomputers & LLMs



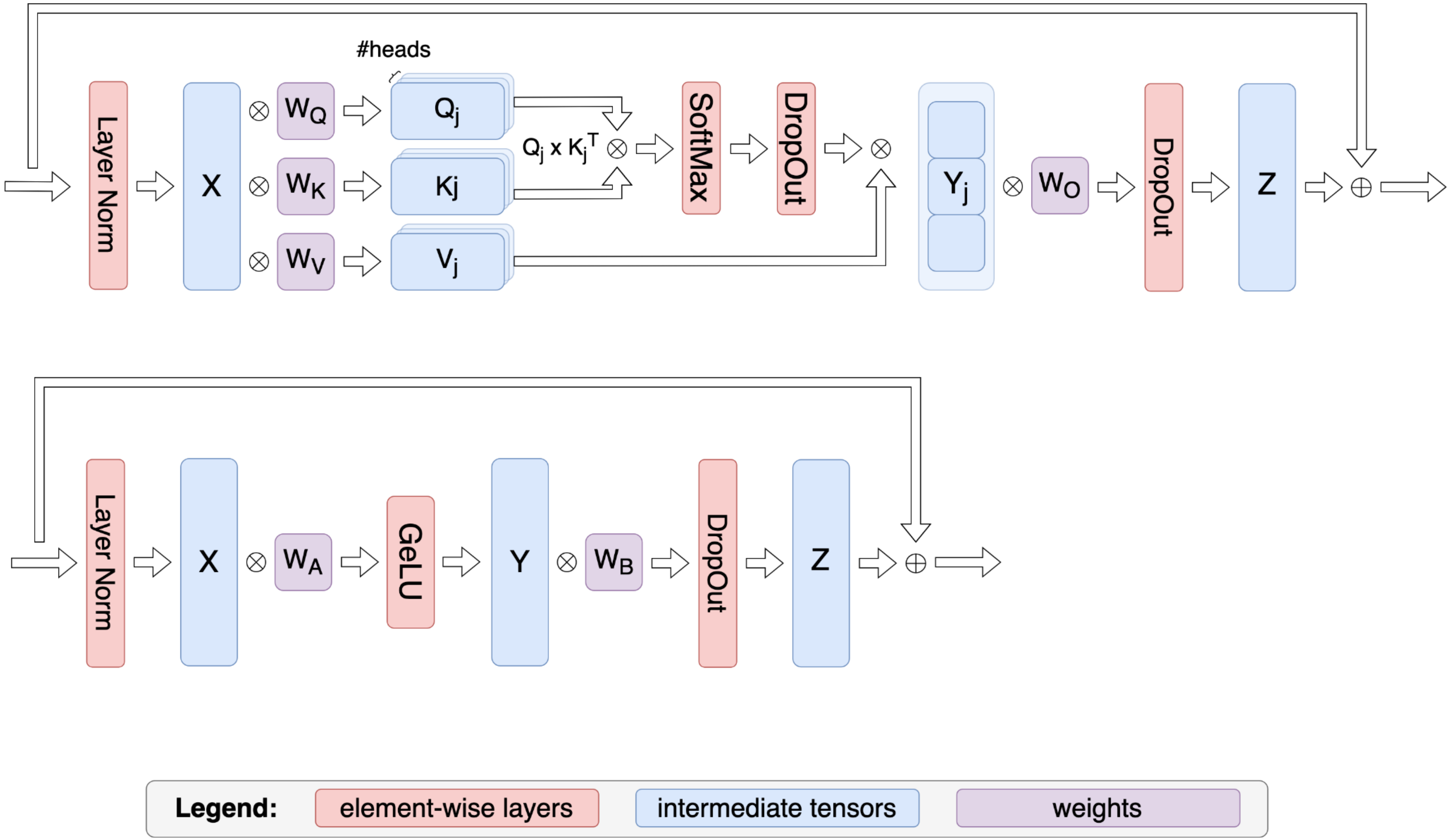
**Mikhail Isaev** (NVIDIA / GT), Nic McDonald (NVIDIA), Larry Dennison (NVIDIA), R.V. (GT) — **SC’23**

# Estimating future LLM performance on future hardware

Inputs

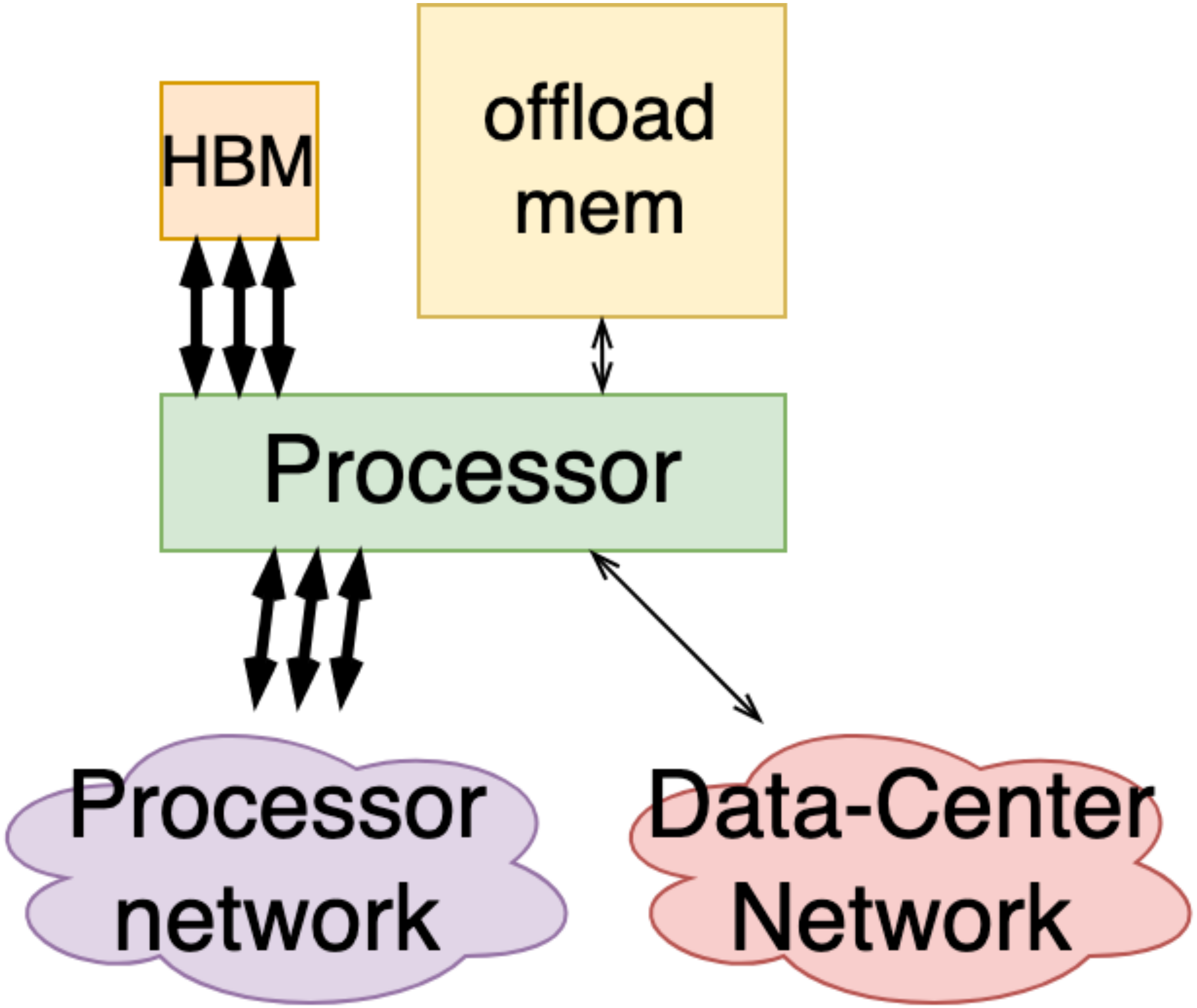
# Estimating future LLM performance on future hardware

Input 1: LLM structure and shapes

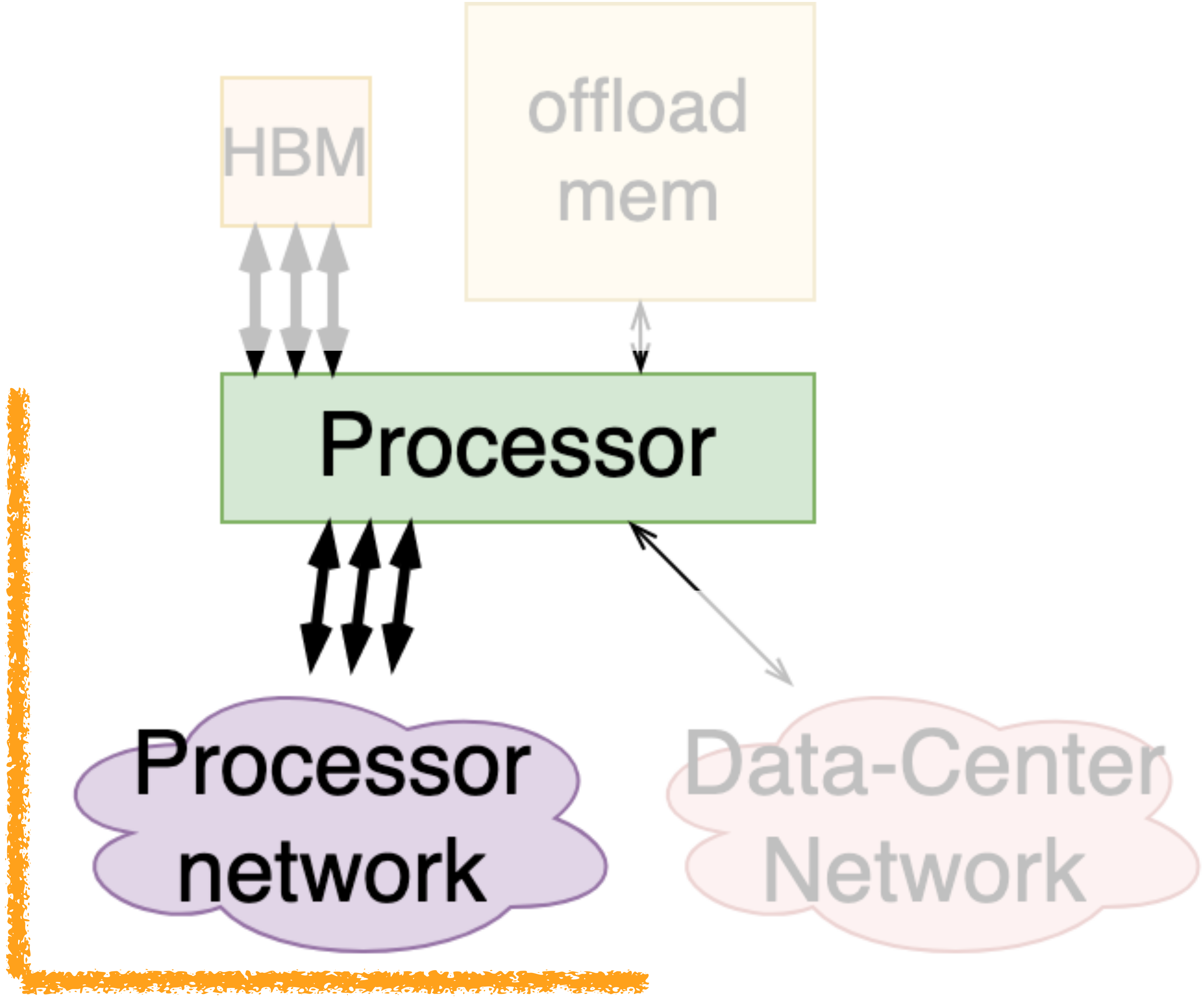
# Estimating future LLM performance on future hardware

Input 2: Hardware speeds and feeds

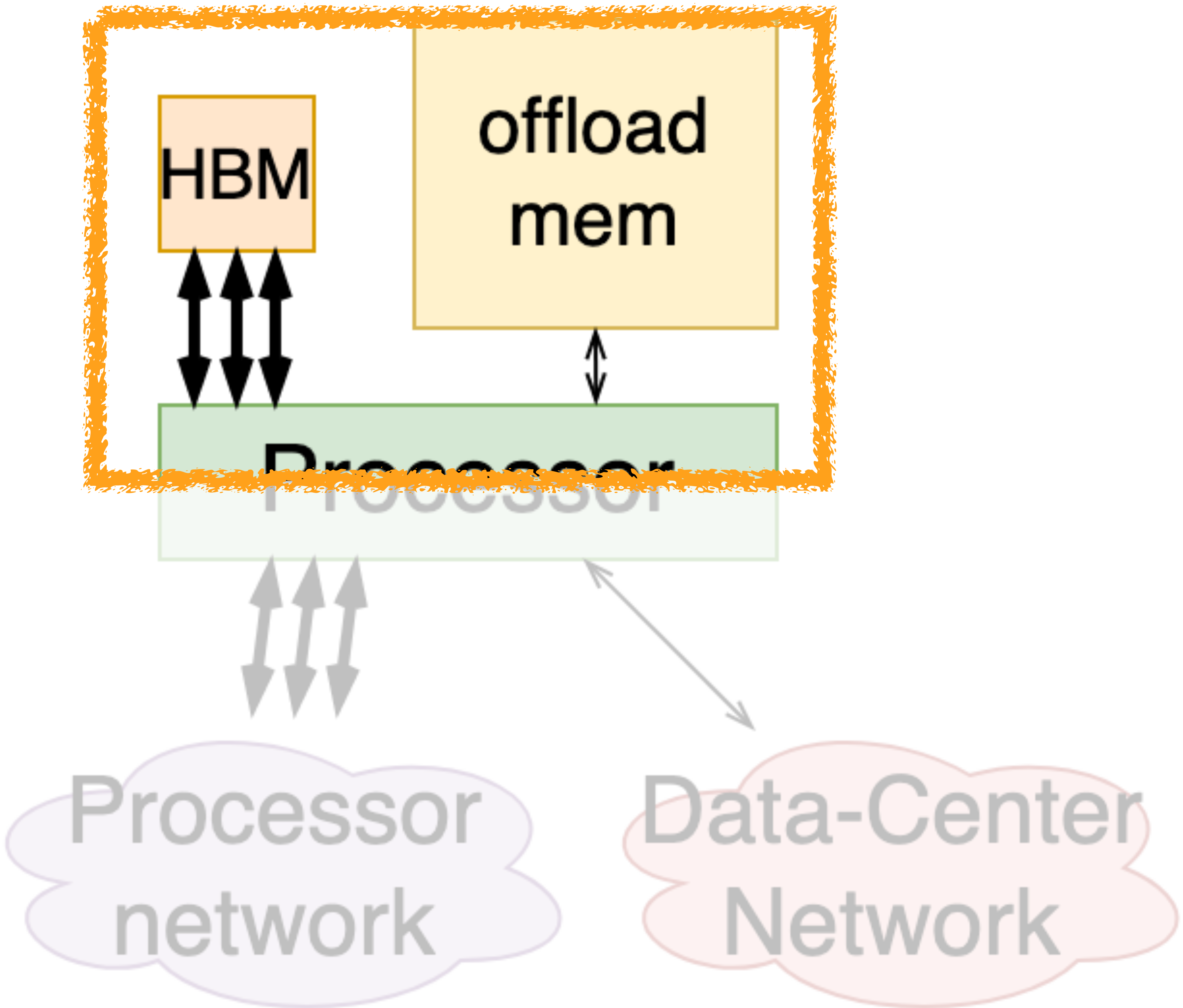
# Estimating future LLM performance on future hardware

Input 2: Hardware speeds and feeds

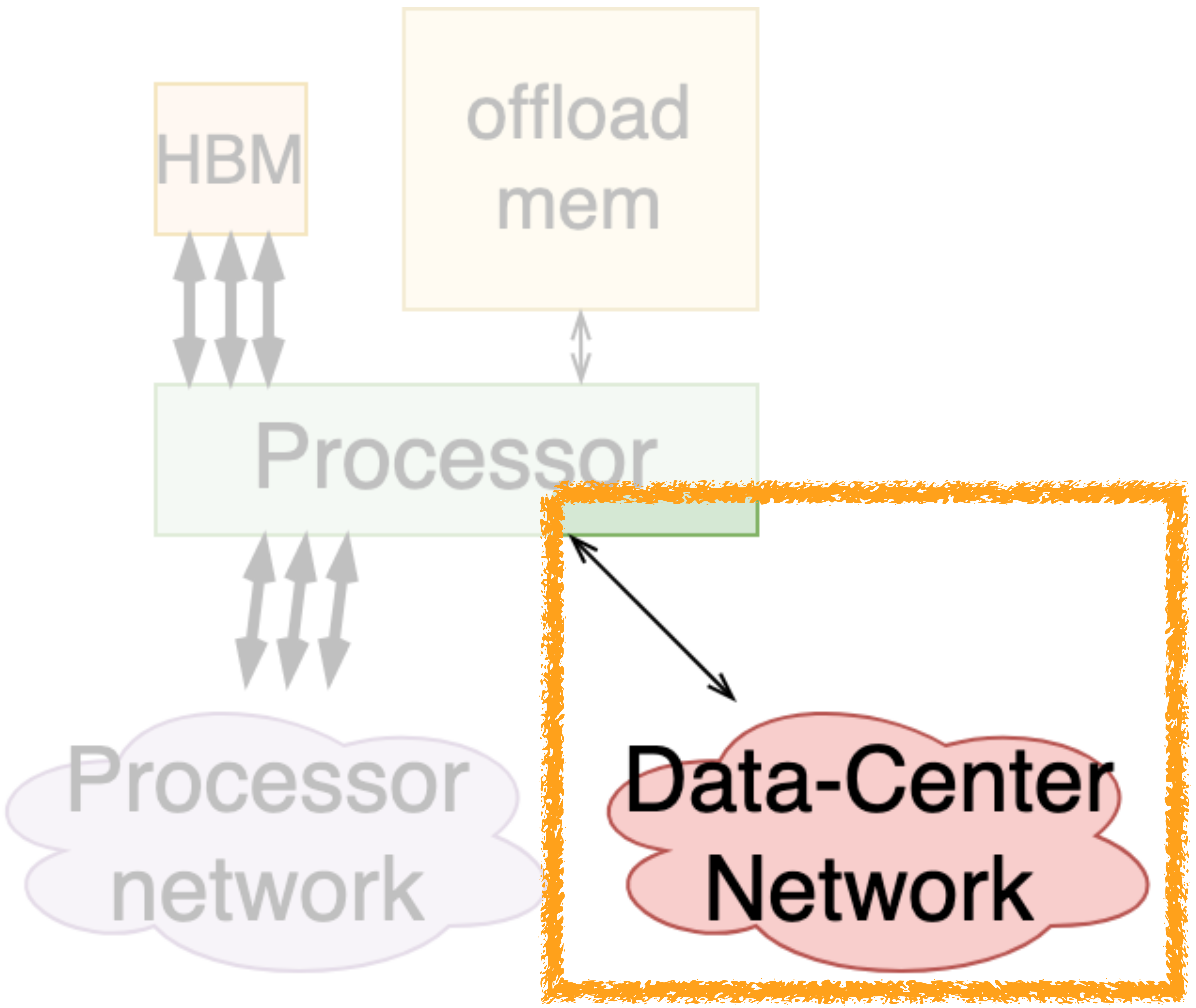
# Estimating future LLM performance on future hardware

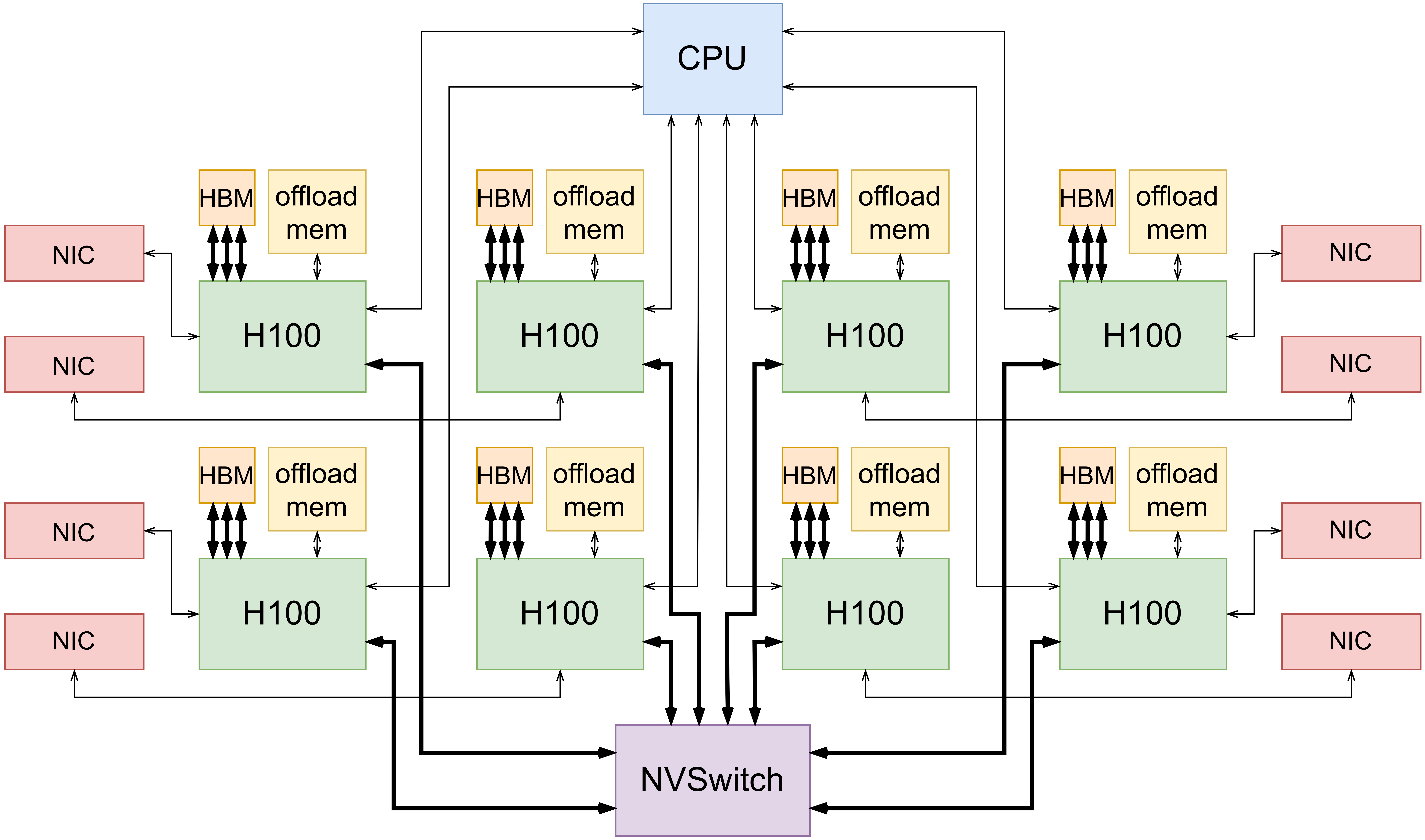
Input 2: Hardware speeds and feeds

# Estimating future LLM performance on future hardware

Input 2: Hardware speeds and feeds

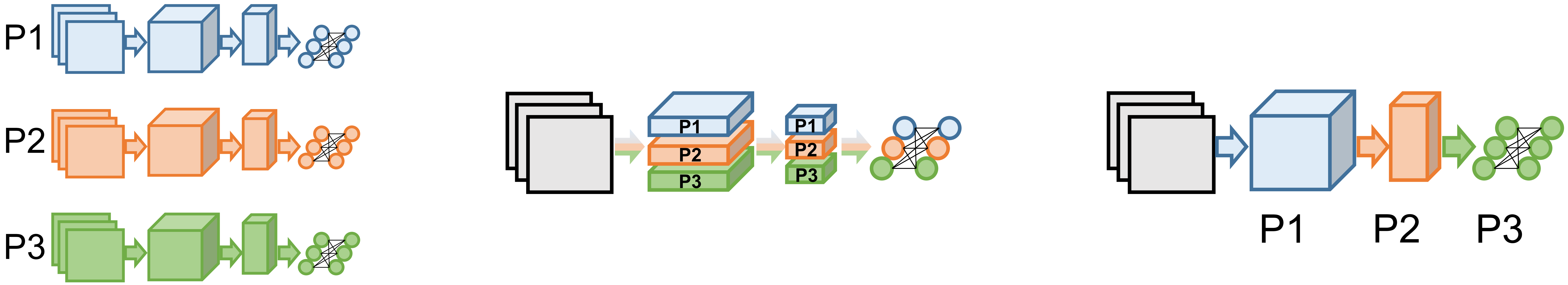

CPU
HBM
offload mem
HBM
offload mem
HBM
offload mem
HBM
offload mem
NIC
H100
H100
H100
H100
NIC
NIC
NIC
HBM
offload mem
HBM
offload mem
HBM
offload mem
HBM
offload mem
NIC
H100
H100
H100
H100
NIC
NIC
NIC
NVSwitch

# Estimating future LLM performance on future hardware

Input 3: Execution strategy — how will the computation be mapped to the machine?

# Estimating future LLM performance on future hardware

Input 3: Execution strategy — how will the computation be mapped to the machine?


(a) Data Parallelism (b) Model Parallelism (c) Layer Pipelining

(Tensor)

**Ben-Nun & Hoefler (2019)**. "Demystifying parallel and distributed deep learning: an in-depth concurrency analysis." doi:10.1145/3320060

# Estimating future LLM performance on future hardware

Input 3: Execution strategy — how will the computation be mapped to the machine?


| Optimization | Year | Related system | Comp time | Comp util | Mem time | Mem cap | Mem BW | Net time | Net BW | range |
|---|---|---|---|---|---|---|---|---|---|---|
| Data parallelism (DP) [61] | 1989 | network | – | ↑ | – | ↑↑↑ | – | ↑ | ↑ | 1 .. batch |
| DP overlap [25] | 2017 | network | ↑ | ↓ | – | – | – | ↓↓↓ | – | true/false |
| Optimizer sharding [24] | 2019 | network | ↓ | – | – | ↓↓ | – | – | – | true/false |
| Recompute [5, 10] | 2000 | compute | ↑↑ | – | – | ↓↓↓ | – | – | – | full/attn/none |
| Fused layers [28] | 2018 | compute | – | ↑↑ | ↓↓ | ↓↓ | ↓ | – | – | true/false |
| Microbatch training [13] | 2019 | compute | – | ↑↑ | – | ↑↑↑ | – | – | – | 1 .. batch/DP |
| Pipeline parallelism (PP) [7, 13] | 2012 | network | ↑ | ↓↓ | – | ↓↓ | – | ↑ | ↑ | 1 .. blocks |
| PP 1F1B schedule [7, 32] | 2012 | network | – | – | – | ↓↓ | – | – | – | true/false |
| PP interleaving [33] | 2021 | network | ↓ | ↑↑ | – | ↑ | – | ↑ | ↑↑ | 1 .. blocks/PP |
| PP RS + AG [21] | 2022 | network | – | – | – | – | – | ↓ | ↓↓ | true/false |
| Tensor parallelism (TP) [7, 22, 49] | 2012 | network | ↓↓ | ↓ | – | ↓↓ | ↓↓ | ↑↑↑ | ↑↑↑ | 1 .. attn |
| TP RS + AG instead AR [33] | 2021 | network | – | – | ↑ | ↑ | – | ↓ | ↓ | true/false |
| Sequence parallelism (SP) [21] | 2022 | network | ↓ | – | ↓ | ↓↓ | ↓ | ↑ | ↑ | true/false |
| TP redo for SP [21] | 2022 | network | – | – | – | ↓ | – | ↑ | ↑ | true/false |
| TP overlap [58] | 2022 | network | ↑ | ↓ | – | – | – | ↓↓ | – | true/false |
| Weight offload [48] | 2021 | memory | – | – | ↑ | ↓↓↓ | ↑ | – | – | true/false |
| Activation offload [48] | 2021 | memory | – | – | ↑ | ↓↓↓ | ↑ | – | – | true/false |
| Optimizer offload [48] | 2021 | memory | – | – | ↑ | ↓ | ↑ | – | – | true/false |

# Estimating future LLM performance on future hardware

Input 3: Execution strategy — how will the computation be mapped to the machine?

LLM

exe

HW

| Optimization | Year | Related system | Comp time | Comp util | Mem time | Mem cap | Mem BW | Net time | Net BW | range |
|---|---|---|---|---|---|---|---|---|---|---|
| Data parallelism (DP) [61] | 1989 | network | – | ↑ | – | ↑↑↑ | – | ↑ | ↑ | 1 .. batch |
| DP overlap [25] | 2017 | network | ↑ | ↓ | – | – | – | ↓↓↓ | – | true/false |
| Optimizer sharding [24] | 2019 | network | ↓ | – | – | ↓↓ | – | – | – | true/false |
| Recompute [5, 10] | 2000 | compute | ↑↑ | – | – | ↓↓↓ | – | – | – | full/attn/none |
| Fused layers [28] | 2018 | compute | – | ↑↑ | ↓↓ | ↓↓ | ↓ | – | – | true/false |
| Microbatch training [13] | 2019 | compute | – | ↑↑ | – | ↑↑↑ | – | – | – | 1 .. batch/DP |
| Pipeline parallelism (PP) [7, 13] | 2012 | network | ↑ | ↓↓ | – | ↓↓ | – | ↑ | ↑ | 1 .. blocks |
| PP 1F1B schedule [7, 32] | 2012 | network | – | – | – | ↓↓ | – | – | – | true/false |
| PP interleaving [33] | 2021 | network | ↓ | ↑↑ | – | ↑ | – | ↑ | ↑↑ | 1 .. blocks/PP |
| PP RS + AG [21] | 2022 | network | – | – | – | – | – | ↓ | ↓↓ | true/false |
| Tensor parallelism (TP) [7, 22, 49] | 2012 | network | ↓↓ | ↓ | – | ↓↓ | ↓↓ | ↑↑↑ | ↑↑↑ | 1 .. attn |
| TP RS + AG instead AR [33] | 2021 | network | – | – | ↑ | ↑ | – | ↓ | ↓ | true/false |
| Sequence parallelism (SP) [21] | 2022 | network | ↓ | – | ↓ | ↓↓ | ↓ | ↑ | ↑ | true/false |
| TP redo for SP [21] | 2022 | network | – | – | – | ↓ | – | ↑ | ↑ | true/false |
| TP overlap [58] | 2022 | network | ↑ | ↓ | – | – | – | ↓↓ | – | true/false |
| Weight offload [48] | 2021 | memory | – | – | ↑ | ↓↓↓ | ↑ | – | – | true/false |
| Activation offload [48] | 2021 | memory | – | – | ↑ | ↓↓↓ | ↑ | – | – | true/false |
| Optimizer offload [48] | 2021 | memory | – | – | ↑ | ↓ | ↑ | – | – | true/false |

# Estimating future LLM performance on future hardware

Input 3: Execution strategy — how will the computation be mapped to the machine?


| Optimization | Year | Related system | Comp time | Comp util | Mem time | Mem cap | Mem BW | Net time | Net BW | range |
|---|---|---|---|---|---|---|---|---|---|---|
| Data parallelism (DP) [61] | 1989 | network | – | ↑ | – | ↑↑↑ | – | ↑ | ↑ | 1 .. batch |
| DP overlap [25] | 2017 | network | ↑ | ↓ | – | – | – | ↓↓↓ | – | true/false |
| Optimizer sharding [24] | 2019 | network | ↓ | – | – | ↓↓ | – | – | – | true/false |
| Recompute [5, 10] | 2000 | compute | ↑↑ | – | – | ↓↓↓ | – | – | – | full/attn/none |
| Fused layers [28] | 2018 | compute | – | ↑↑ | ↓↓ | ↓↓ | ↓ | – | – | true/false |
| Microbatch training [13] | 2019 | compute | – | ↑↑ | – | ↑↑↑ | – | – | – | 1 .. batch/DP |
| Pipeline parallelism (PP) [7, 13] | 2012 | network | ↑ | ↓↓ | – | ↓↓ | – | ↑ | ↑ | 1 .. blocks |
| PP 1F1B schedule [7, 32] | 2012 | network | – | – | – | ↓↓ | – | – | – | true/false |
| PP interleaving [33] | 2021 | network | ↓ | ↑↑ | – | ↑ | – | ↑ | ↑↑ | 1 .. blocks/PP |
| PP RS + AG [21] | 2022 | network | – | – | – | – | – | ↓ | ↓↓ | true/false |
| Tensor parallelism (TP) [7, 22, 49] | 2012 | network | ↓↓ | ↓ | – | ↓↓ | ↓↓ | ↑↑↑ | ↑↑↑ | 1 .. attn |
| TP RS + AG instead AR [33] | 2021 | network | – | – | ↑ | ↑ | – | ↓ | ↓ | true/false |
| Sequence parallelism (SP) [21] | 2022 | network | ↓ | – | ↓ | ↓↓ | ↓ | ↑ | ↑ | true/false |
| TP redo for SP [21] | 2022 | network | – | – | – | ↓ | – | ↑ | ↑ | true/false |
| TP overlap [58] | 2022 | network | ↑ | ↓ | – | – | – | ↓↓ | – | true/false |
| Weight offload [48] | 2021 | memory | – | – | ↑ | ↓↓↓ | ↑ | – | – | true/false |
| Activation offload [48] | 2021 | memory | – | – | ↑ | ↓↓↓ | ↑ | – | – | true/false |
| Optimizer offload [48] | 2021 | memory | – | – | ↑ | ↓ | ↑ | – | – | true/false |

# Estimating future LLM performance on future hardware

Input 3: Execution strategy — how will the computation be mapped to the machine?

LLM

exe

HW

| Optimization | Year | Related system | Comp time | Comp util | Mem time | Mem cap | Mem BW | Net time | Net BW | Range |
|---|---|---|---|---|---|---|---|---|---|---|
| Data parallelism (DP) [61] | 1989 | network | [illegible] | ↑ | [illegible] | ↑↑↑ | [illegible] | ↑ | ↑ | .. batch |
| DP overlap [25] | 2017 | network | ↑ | ↓ | – | – | – | ↓↓↓ | – | true/false |
| Optimizer sharding [24] | 2019 | network | ↓ | – | – | ↓↓ | – | – | – | true/false |
| Recompute [5, 10] | 2000 | compute | ↑↑ | – | – | ↓↓↓ | – | – | – | full/attn/none |
| Fused layers [28] | 2018 | compute | – | ↑↑ | ↓↓ | ↓↓ | ↓ | – | – | true/false |
| Microbatch training [13] | 2019 | compute | – | ↑↑ | – | ↑↑↑ | – | – | – | 1 .. batch/DP |
| Pipeline parallelism (PP) [7, 13] | 2012 | network | ↑ | ↓↓ | – | ↓↓ | – | ↑ | ↑ | 1 .. blocks |
| PP 1F1B schedule [7, 32] | 2012 | network | – | – | – | ↓↓ | – | – | – | true/false |
| PP interleaving [33] | 2021 | network | ↓ | ↑↑ | – | ↑ | – | ↑ | ↑↑ | 1 .. blocks/PP |
| PP RS + AG [21] | 2022 | network | – | – | – | – | – | ↓ | ↓↓ | true/false |
| Tensor parallelism (TP) [7, 22, 49] | 2012 | network | ↓↓ | ↓ | – | ↓↓ | ↓↓ | ↑↑↑ | ↑↑↑ | 1 .. attn |
| TP RS + AG instead AR [33] | 2021 | network | – | – | ↑ | ↑ | – | ↓ | ↓ | true/false |
| Sequence parallelism (SP) [21] | 2022 | network | ↓ | – | ↓ | ↓↓ | ↓ | ↑ | ↑ | true/false |
| TP redo for SP [21] | 2022 | network | – | – | – | ↓ | – | ↑ | ↑ | true/false |
| TP overlap [58] | 2022 | network | ↑ | ↓ | – | – | – | ↓↓ | – | true/false |
| Weight offload [48] | 2021 | memory | – | – | ↑ | ↓↓↓ | ↑ | – | – | true/false |
| Activation offload [48] | 2021 | memory | – | – | ↑ | ↓↓↓ | ↑ | – | – | true/false |
| Optimizer offload [48] | 2021 | memory | – | – | ↑ | ↓ | ↑ | – | – | true/false |

# Estimating future LLM performance on future hardware

Input 3: Execution strategy — how will the computation be mapped to the machine?


| Optimization | Year | Related system | Comp time | Comp util | Mem time | Mem cap | Mem BW | Net time | Net BW | range |
|---|---|---|---|---|---|---|---|---|---|---|
| Data parallelism (DP) [61] | 1989 | network | – | ↑ | – | ↑↑↑ | – | ↑ | ↑ | 1 .. batch |
| DP overlap [25] | 2017 | network | ↑ | ↓ | – | – | – | ↓↓↓ | – | true/false |
| Optimizer sharding [24] | 2019 | network | ↓ | – | – | ↓↓ | – | – | – | true/false |
| Recompute [5, 10] | 2000 | compute | ↑↑ | – | – | ↓↓↓ | – | – | – | full/attn/none |
| Fused layers [28] | 2018 | compute | – | ↑↑ | ↓↓ | ↓↓ | ↓ | – | – | true/false |
| Microbatch training [13] | 2019 | compute | – | ↑↑ | – | ↑↑↑ | – | – | – | 1 .. batch/DP |
| Pipeline parallelism (PP) [7, 13] | 2012 | network | ↑ | ↓↓ | – | ↓↓ | – | ↑ | ↑ | 1 .. blocks |
| PP 1F1B schedule [7, 32] | 2012 | network | – | – | – | ↓↓ | – | – | – | true/false |
| PP interleaving [33] | 2021 | network | ↓ | ↑↑ | – | ↑ | – | ↑ | ↑↑ | 1 .. blocks/PP |
| PP RS + AG [21] | 2022 | network | – | – | – | – | – | ↓ | ↓↓ | true/false |
| Tensor parallelism (TP) [7, 22, 49] | 2012 | network | ↓↓ | ↓ | – | ↓↓ | ↓↓ | ↑↑↑ | ↑↑↑ | 1 .. attn |
| TP RS + AG instead AR [33] | 2021 | | | | | | | | | |
| Sequence parallelism (SP) [21] | 2022 | | | | | | | | | |
| TP redo for SP [21] | 2022 | | | | | | | | | |
| TP overlap [58] | 2022 | | | | | | | | | |
| Weight offload [48] | 2021 | memory | – | – | ↑ | ↓↓↓ | ↑ | – | – | |
| Activation offload [48] | 2021 | memory | – | – | ↑ | ↓↓↓ | ↑ | – | – | |
| Optimizer offload [48] | 2021 | memory | – | – | ↑ | ↓ | ↑ | – | – | |

# Estimating future LLM performance on future hardware

Input 3: Execution strategy — how will the computation be mapped to the machine?


a factor of TP except for the activation space:

$$\mathtt{Flops}^{\mathrm{BLK}}_{\mathrm{fw}} \approx \mathtt{Flops}^{\mathrm{BLK}}_{\mathrm{recomp}} \approx \mathtt{Flops}^{\mathrm{BLK}}_{\mathrm{agrad}} \approx \mathtt{Flops}^{\mathrm{BLK}}_{\mathrm{wgrad}} \approx \frac{12\,\mathtt{batch} \times \mathtt{seq} \times \mathtt{hidden}^2}{TP}$$

$$\frac{M^{\mathrm{BLK}}_{\mathrm{weight}} \approx 12\,\mathtt{hidden}^2 B_{\mathrm{weight}}}{TP}$$

$$M^{\mathrm{BLK}}_{\mathrm{act}} \approx \mathtt{batch} \times \mathtt{seq} \times \mathtt{hidden} \times \left(4B_{\mathrm{act}} + 2 + \frac{12B_{\mathrm{act}}}{TP} + (2B_{\mathrm{act}} + 1)\frac{\mathtt{attn} \times \mathtt{seq}}{TP \times \mathtt{hidden}}\right) B_{\mathrm{act}}$$

$$M^{\mathrm{BLK}}_{\mathrm{act\_checkpoint}} = \frac{\mathtt{batch} \times \mathtt{seq} \times \mathtt{hidden}\; B_{\mathrm{act}}}{TP}. \tag{2.15}$$

For the case of partial activation recomputation [91], attention-related activations are not stored, and activation space becomes:

**(See Isaev's thesis: https://hdl.handle.net/1853/75228)**

# Estimating hardware

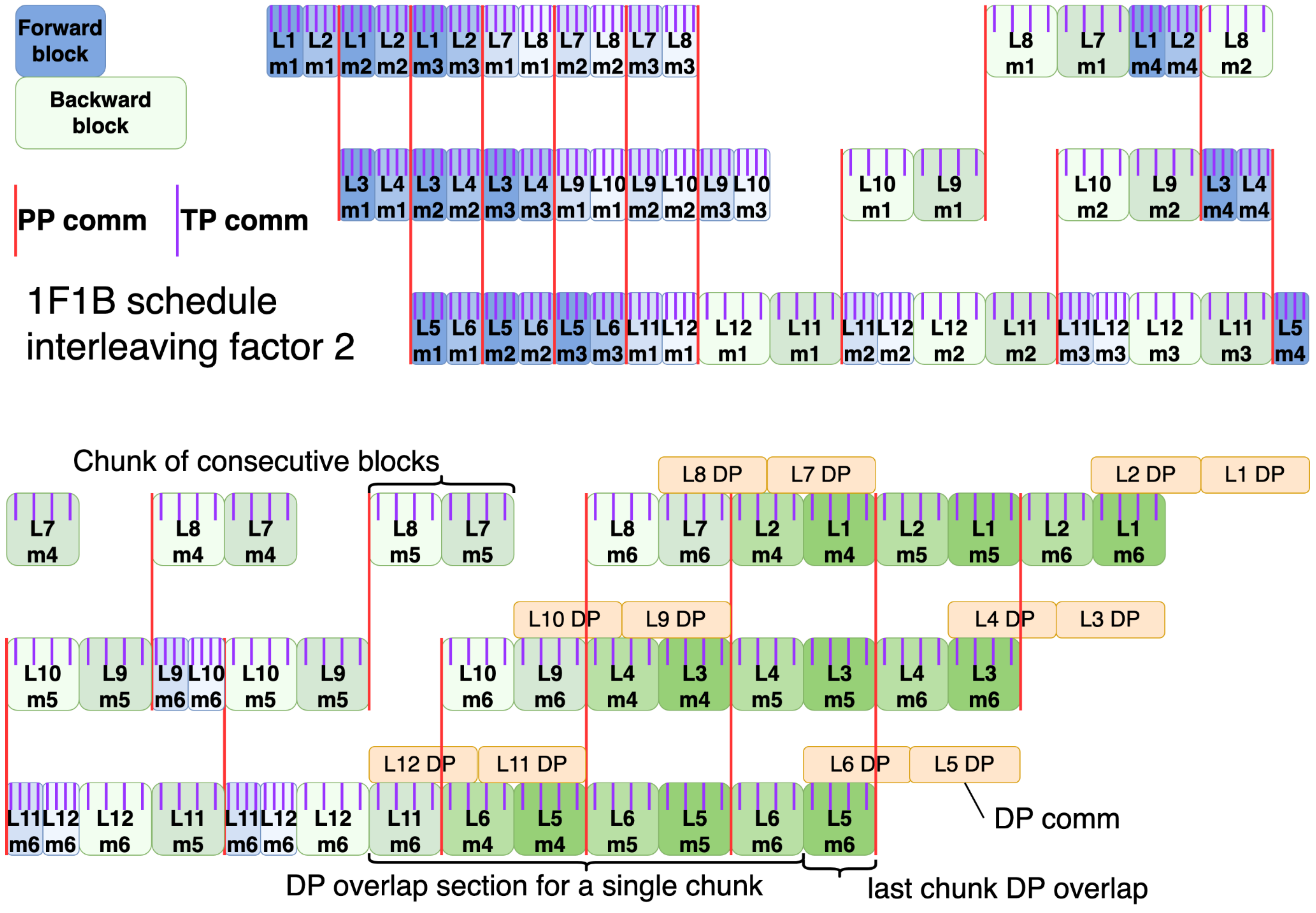

llm.py

calculon > llm > llm.py

```python
class Llm:
  def _compute_batch_stats(self):
    if self.exe.data_par > 1 and self.exe.training:
      if self.exe.data_par_overlap:
        # We can overlap DP with BW pass, overlap[ing AR for previous layer
        # with BW for current, except when optimizer sharded. We can't overlap
        # during optimizer step as we RS grads before step and AG weights after
        # Overlappable chunks have overlap size equal to
        # blocks_per_chunk * num_microbatches
        # In case of 1F1B schedule, num_microbatches == pipeline_par
        overlap_window = self.exe.pipeline_par * chunk_dp_overlap_time
        overlap_compute = self.exe.pipeline_par * chunk_dp_compute_time
        chunk_dp_time = self._blocks_per_chunk * self._block_dp_time
        # We may have PP and DP comm colliding if DP comm takes longer than
        # a single chunk BW time. We can't collide more PP than microbatches
        if self._dp_net == self._pp_net:
          if self.exe._num_microbatches % self.exe.pipeline_par != 0:
            num_overlapped_pp = min(
              chunk_dp_time // chunk_bw_time,
              self.exe._num_microbatches % self.exe.pipeline_par)
          else:
            num_overlapped_pp = min(
              chunk_dp_time // chunk_bw_time,
              self.exe.pipeline_par)
        else:
          # if PP and DP on different networks, overlapping is fine
          num_overlapped_pp = 0
        # we add DP/PP collision time and compute slowdown due to overlap
        overlap_inflection = chunk_dp_time - (overlap_window -
          num_overlapped_pp * chunk_bw_pp_time) + overlap_compute * \
          self._dp_net.processor_usage
        if overlap_inflection > 0:
          # Tcomm is larger than compute, excess is exposed
          overlappable_chunks_exposed_time = num_overlappable_chunks * \
            overlap_inflection
        else:
          # Tcomm is smaller than compute and hidden, but it contributes to
```

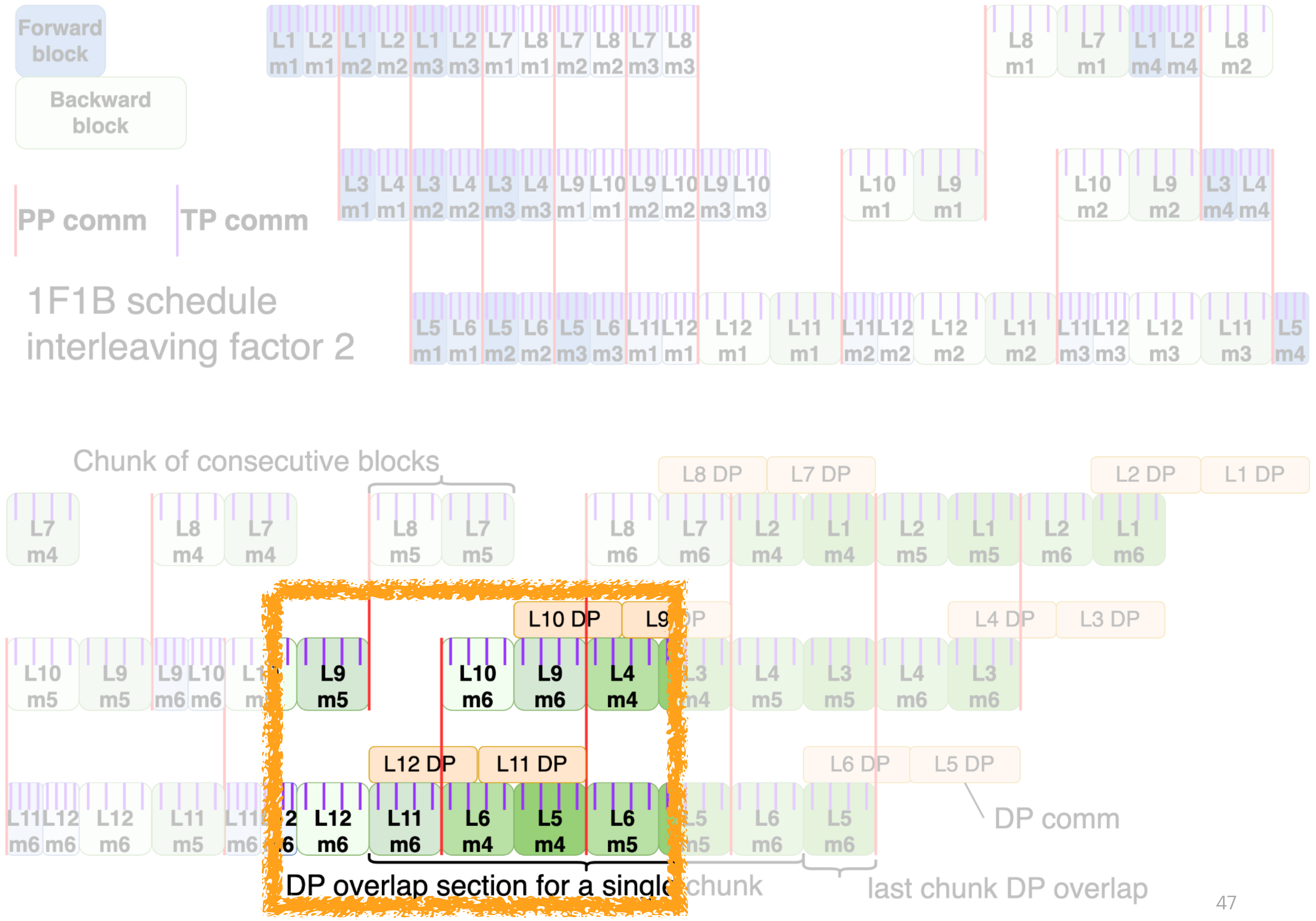
# Estimating future LLM performance on future hardware

Input 3: Execution strategy — how will the computation be mapped to the machine?

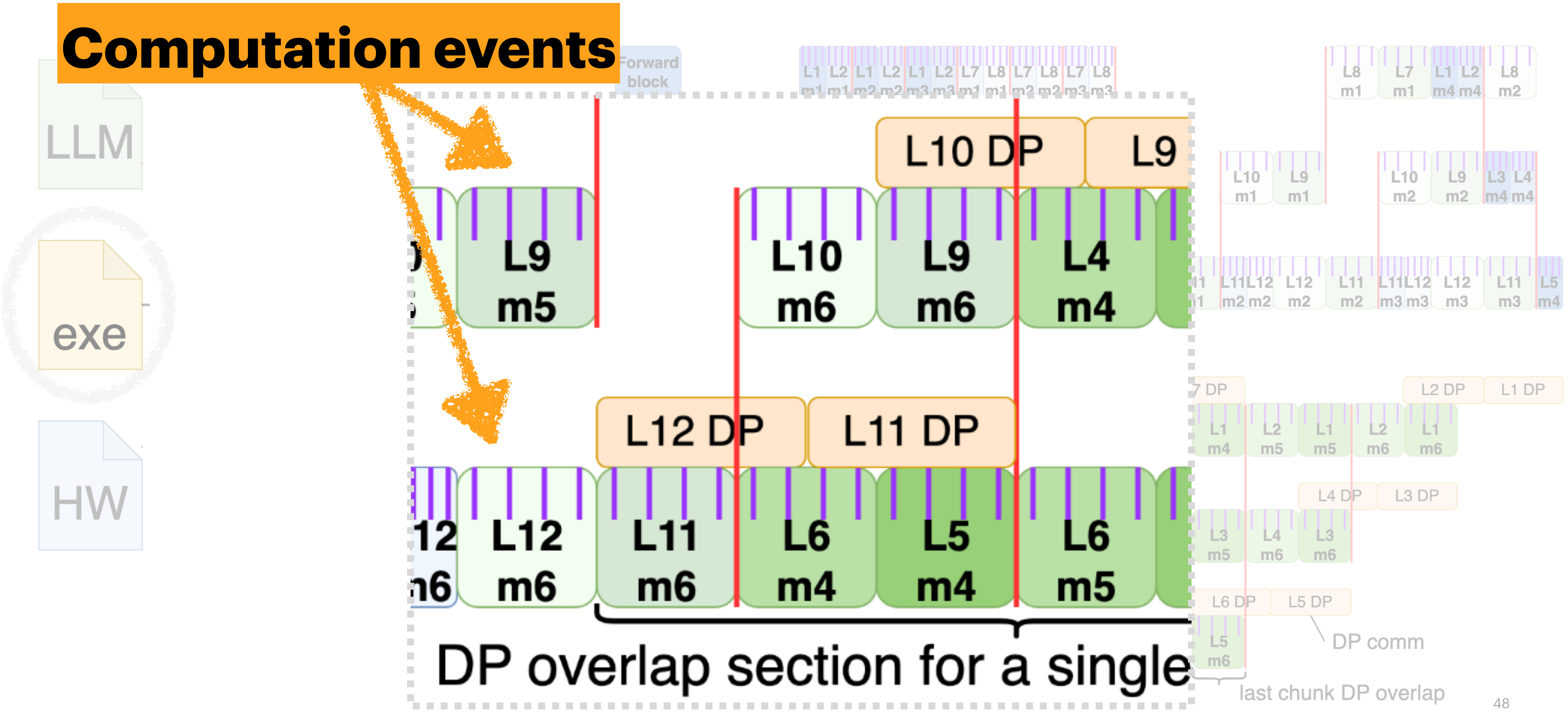



# Estimating future LLM performance on future hardware

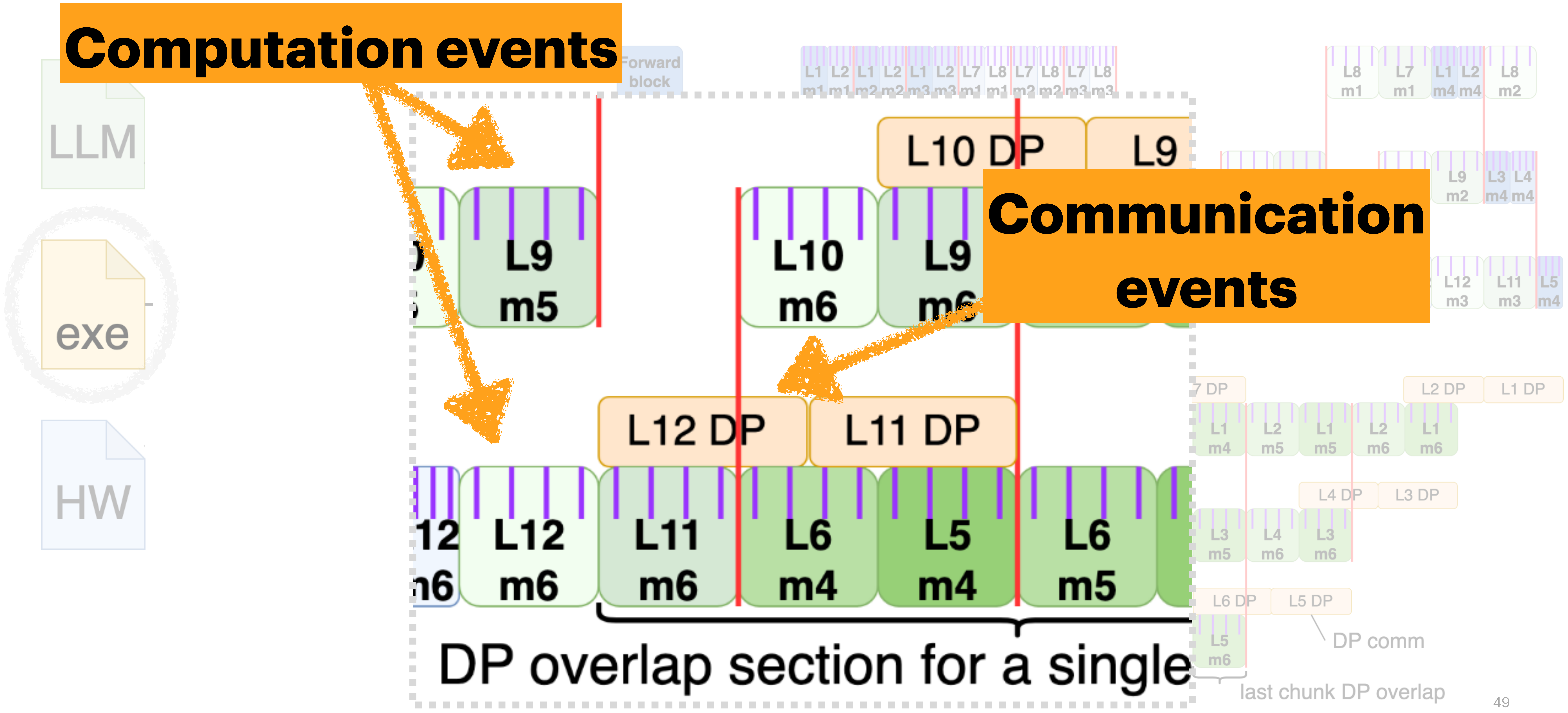
Input 3: Execution strategy — how will the computation be mapped to the machine?

Input 3: Execution strategy — how will the computation be mapped to the machine?

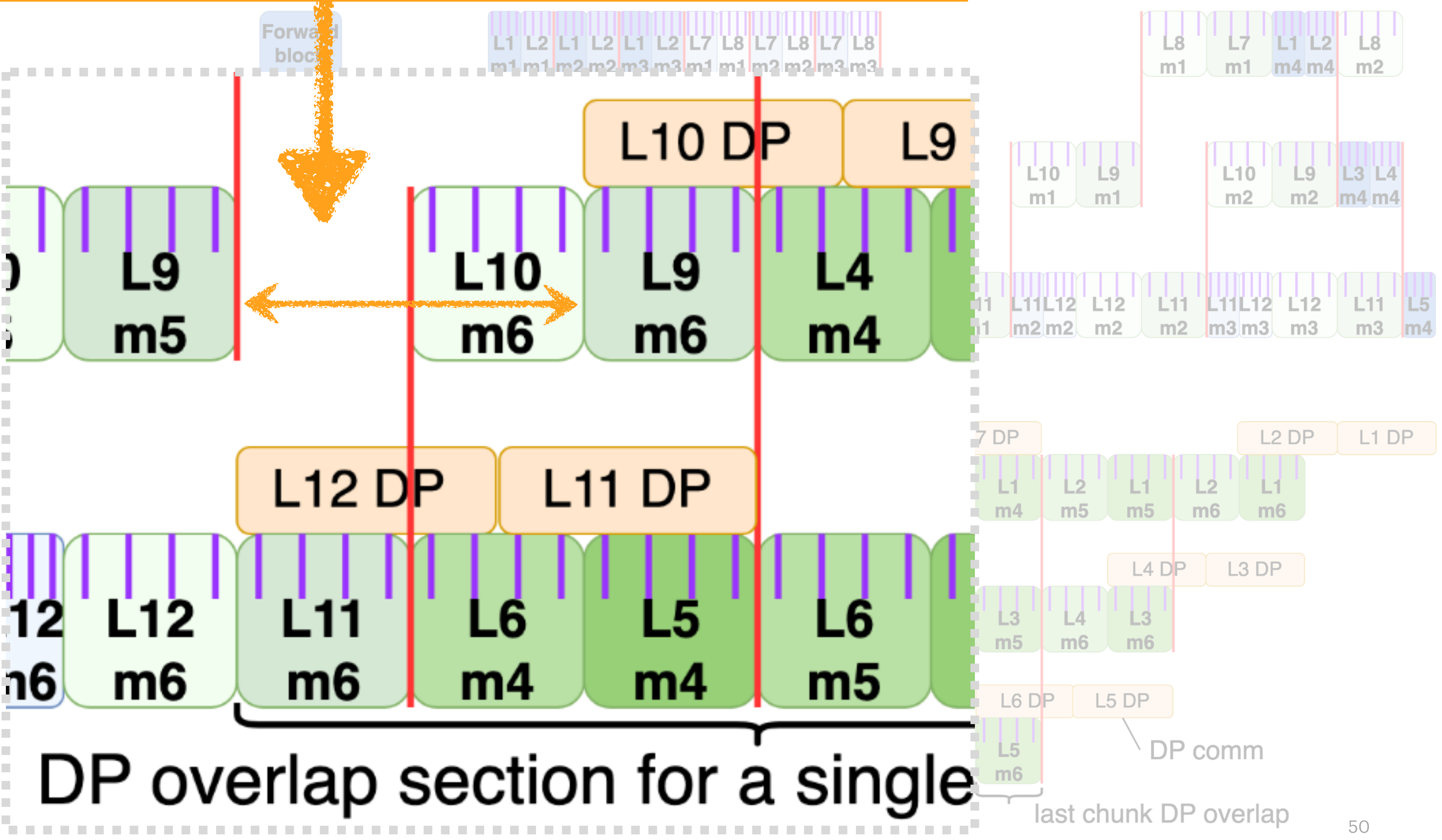


Input 3: Execution strategy — how will the computation be mapped to the machine?



# Inefficiency gaps

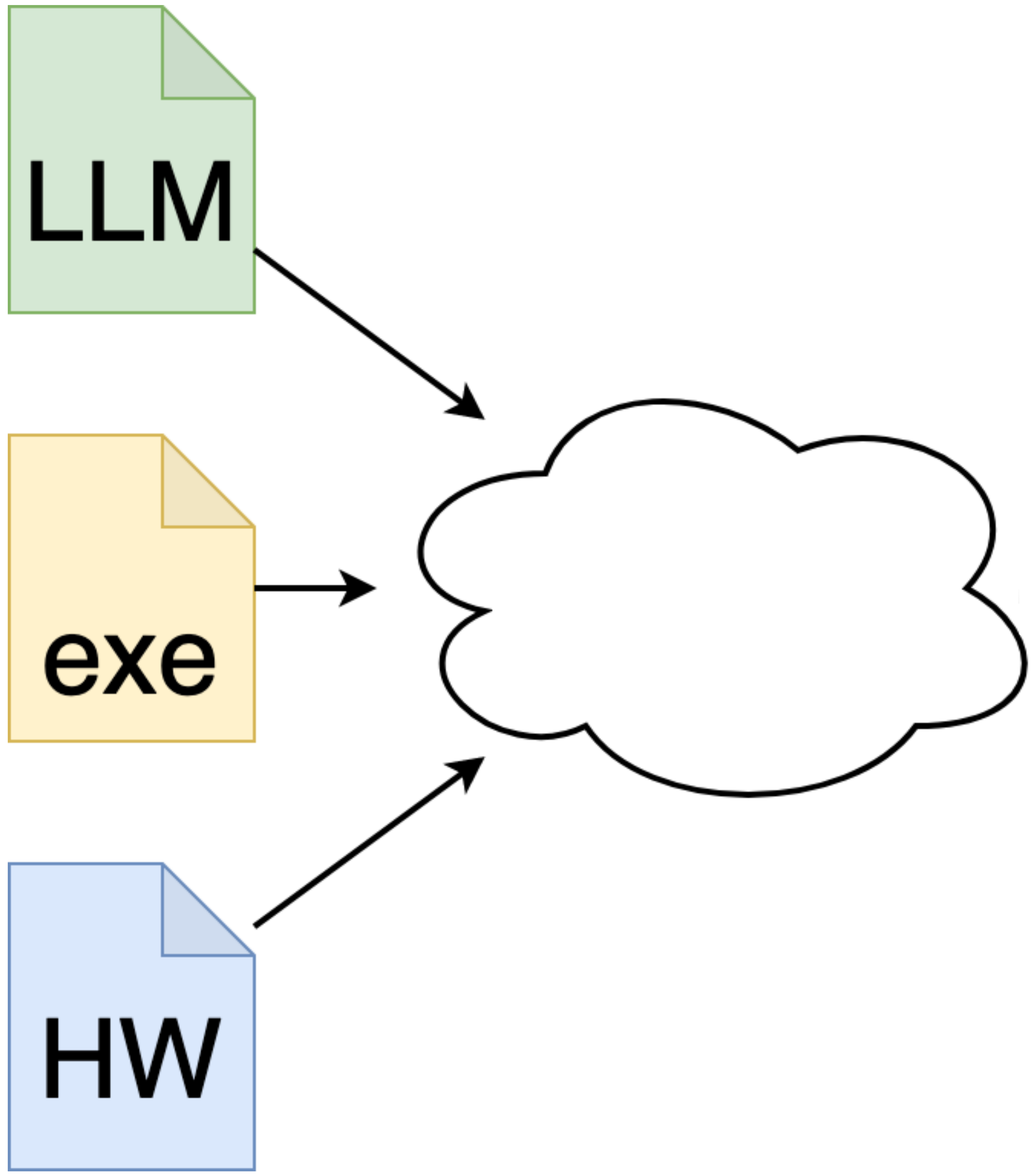
(Dependencies that inhibit full utilization)




Input: Execution strategy — how will the computation be mapped to the machine?

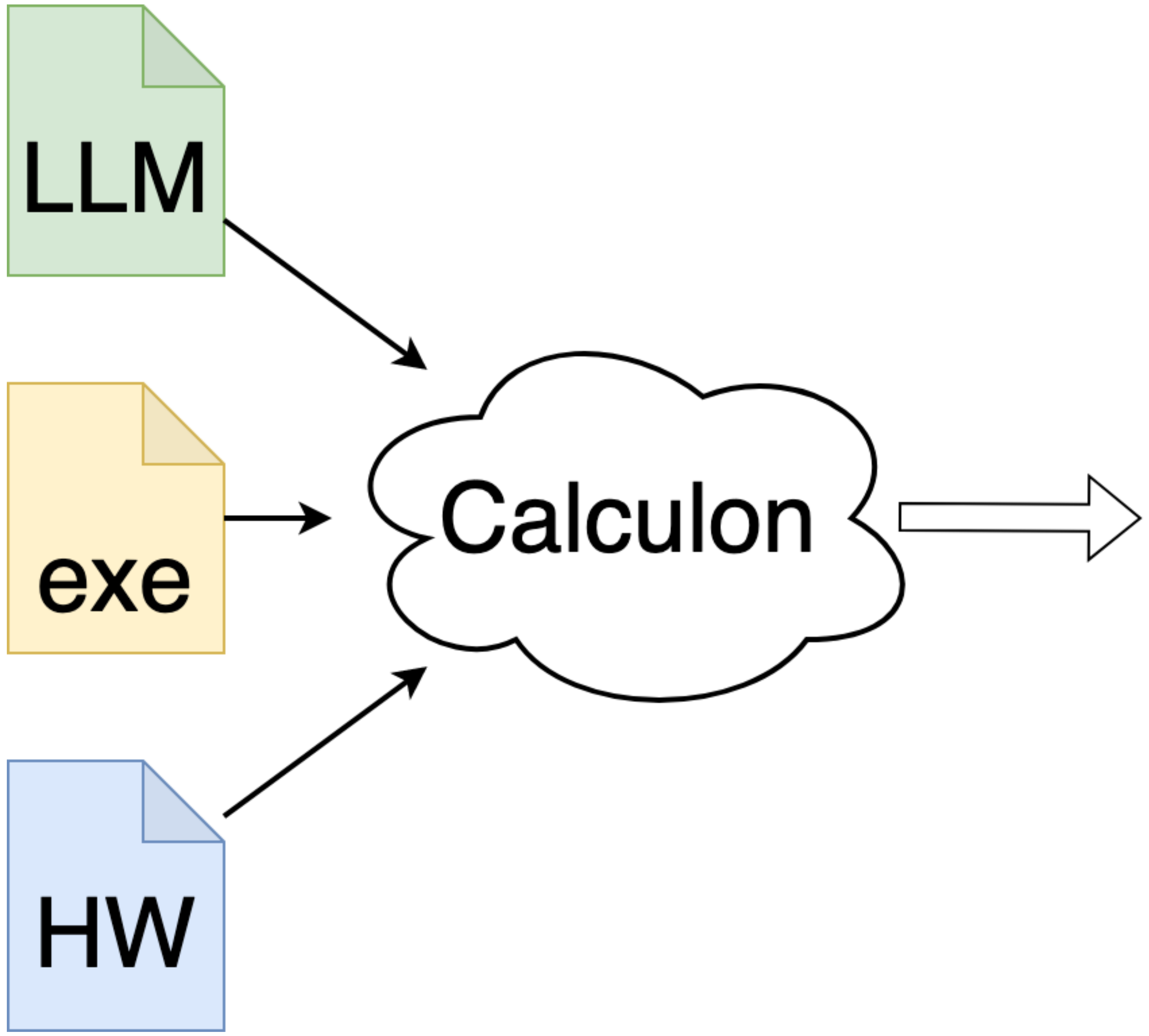
# Estimating future LLM performance on future hardware

Putting it all together...



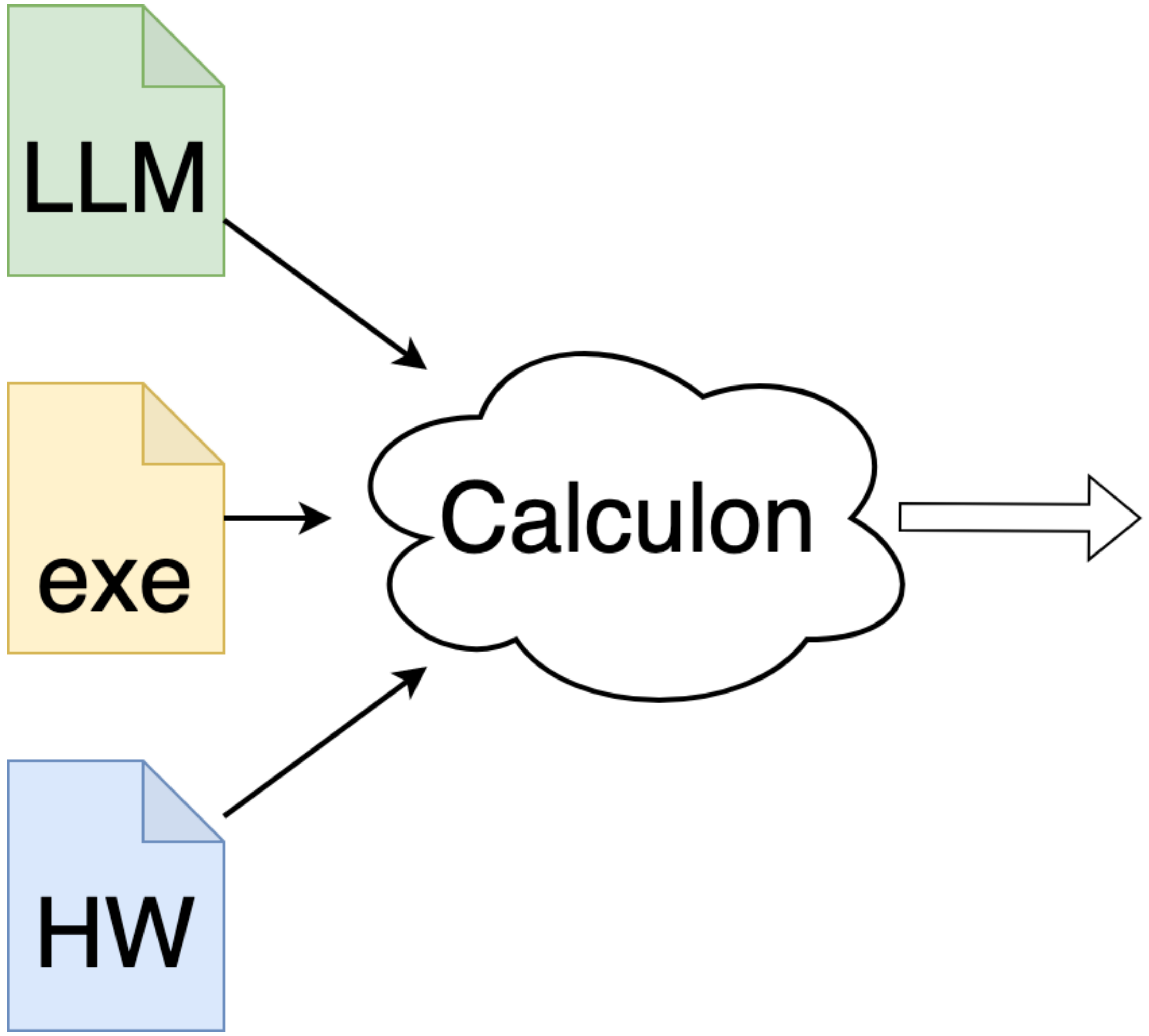
# Estimating future LLM performance on future hardware

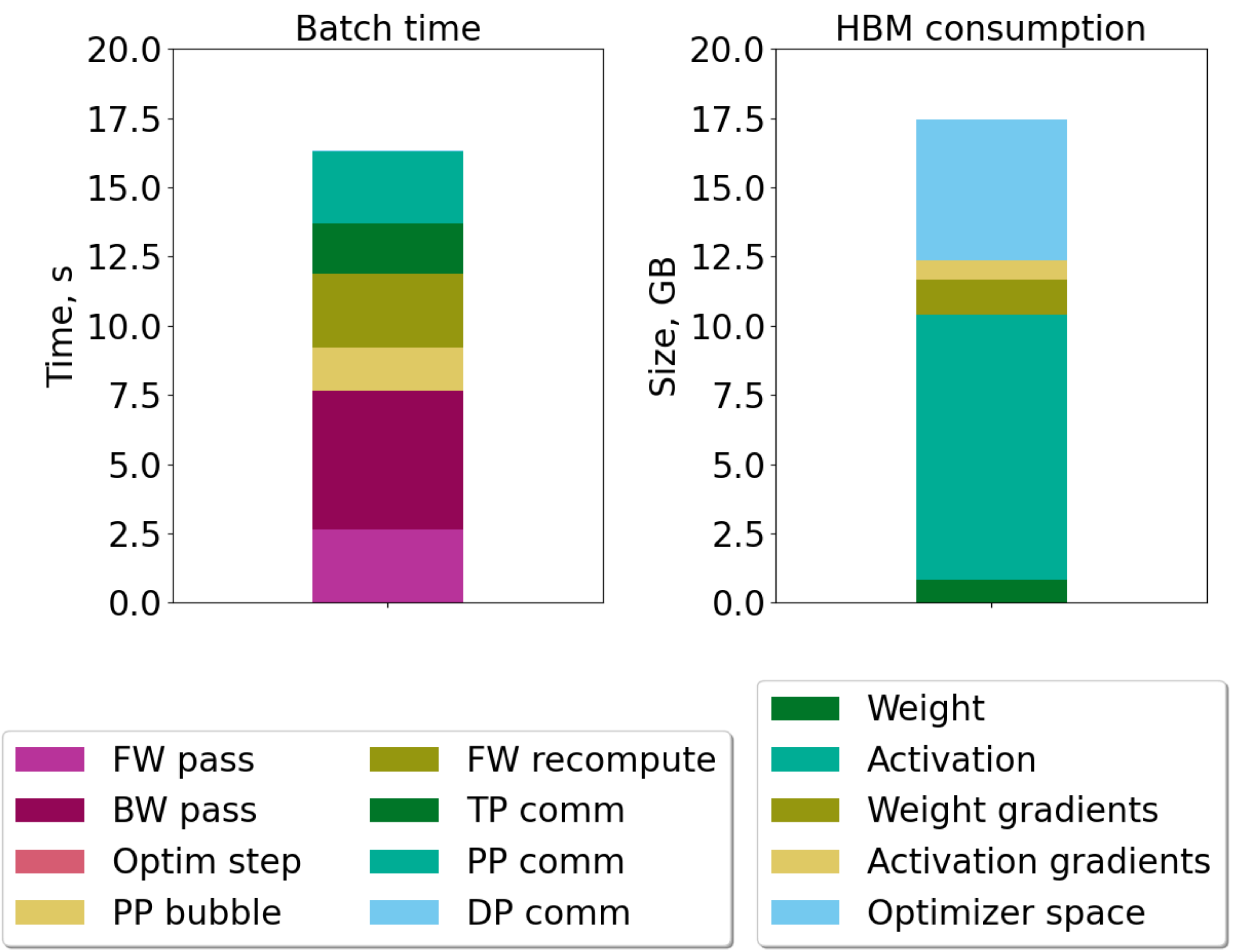
Putting it all together...



# Estimating future LLM performance on future hardware

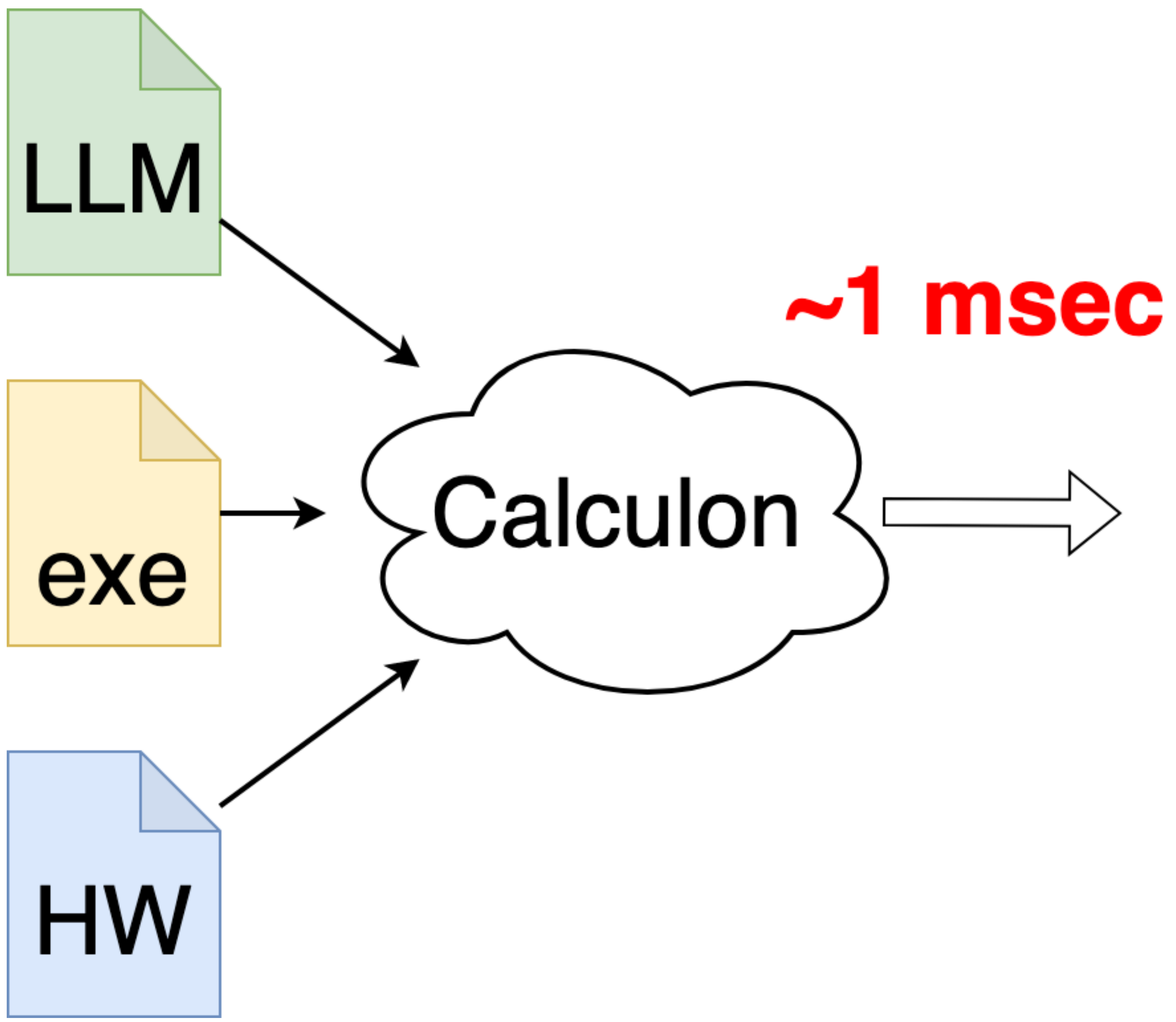
Putting it all together...

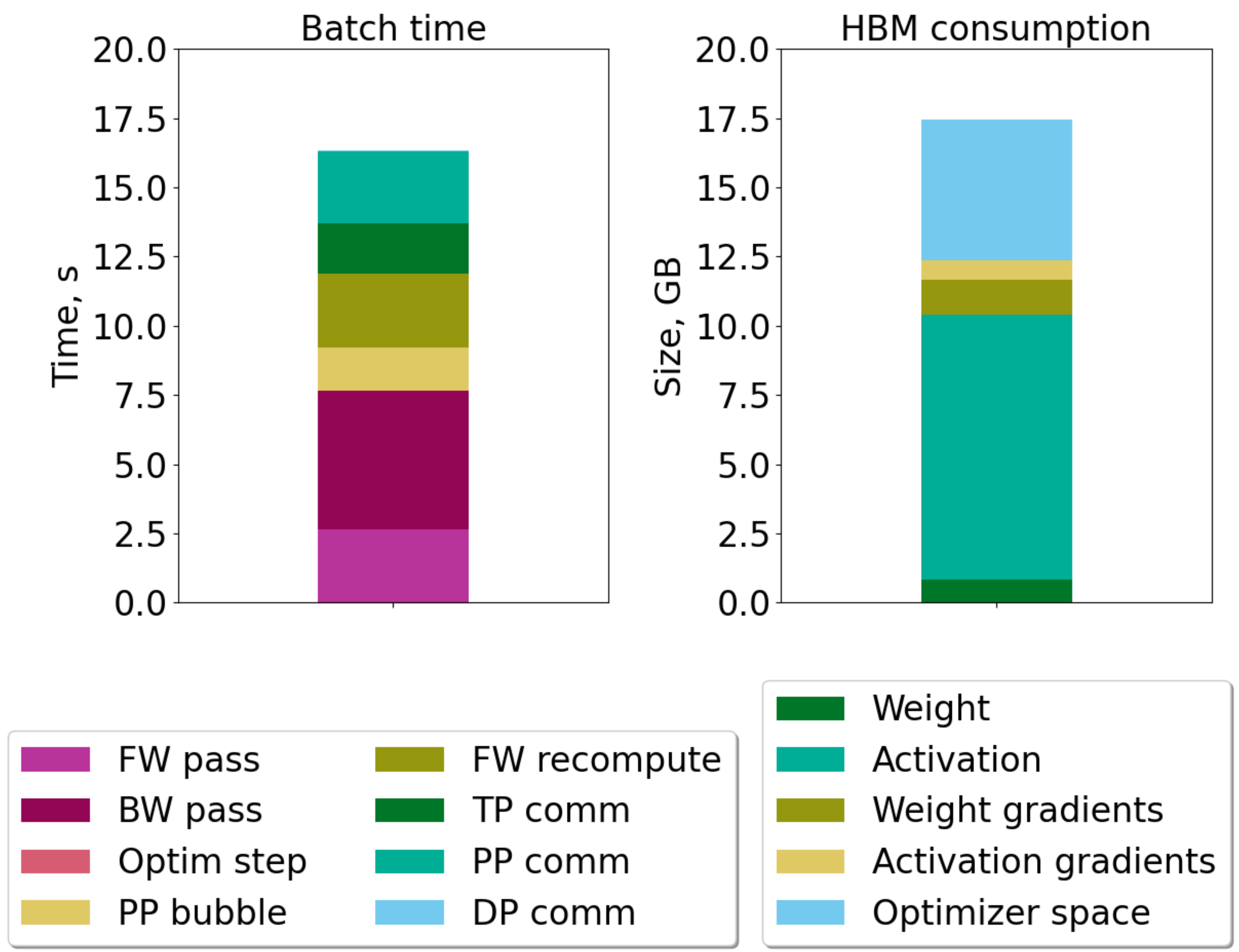


Putting it all together...

# Estimating future LLM performance on future hardware



# Six case studies

Applying Calculon and interpreting its results.

1. **Comparing data vs. tensor vs. pipeline parallelism**: Can combine and tune to manage time-space tradeoffs
2. **Characterizing the "speed distribution:"** An optimal configuration can be a "needle in a haystack"
3. **Strong scaling analysis**: Speed "cliffs" and "plateaus" exist due to "awkward mappings"
4. **Offload memory**: Slow, "low" bandwidth memory can dramatically reduce fast memory capacity requirements
5. **Price-performance analysis**: How should you set up your next $100 million system?
6. **Next-gen models**: What happens at **100 trillion parameters** (and beyond)?

# Six case studies

Applying Calculon and interpreting its results.

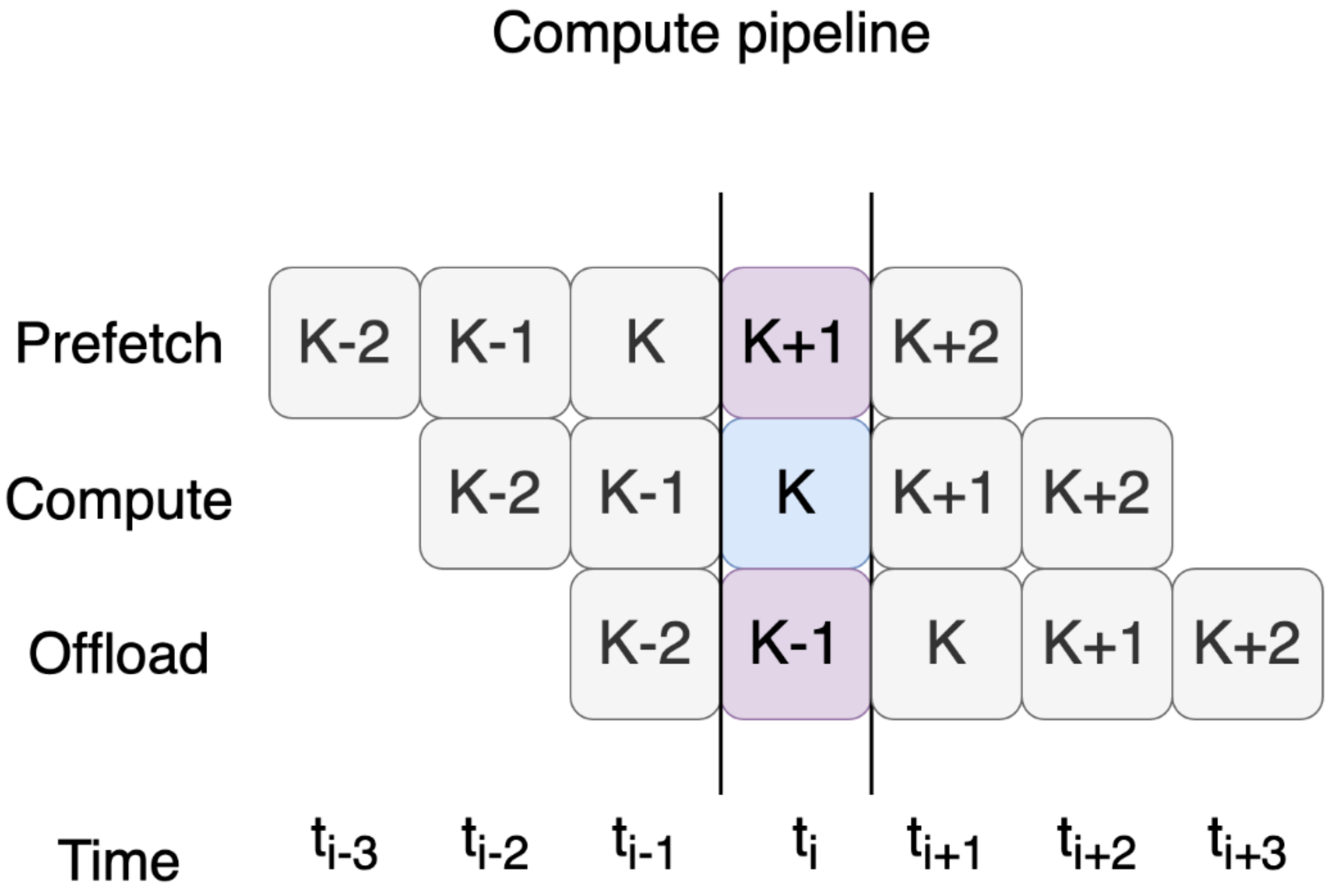
1. **Comparing data vs. tensor vs. pipeline parallelism**: Can combine and tune to manage time-space tradeoffs
2. **Characterizing the "speed distribution:"** An optimal configuration can be a "needle in a haystack"
3. **Strong scaling analysis**: Speed "cliffs" and "plateaus" exist due to "awkward mappings"
4. **Offload memory: Slow, "low" bandwidth memory can dramatically reduce fast memory capacity requirements**
5. **Price-performance analysis**: How should you set up your next $100 million system?
6. **Next-gen models**: What happens at **100 trillion parameters** (and beyond)?

# Study 4: tensor offloading

Analyzing an algorithmic design choice that balances parallelization with memory use.

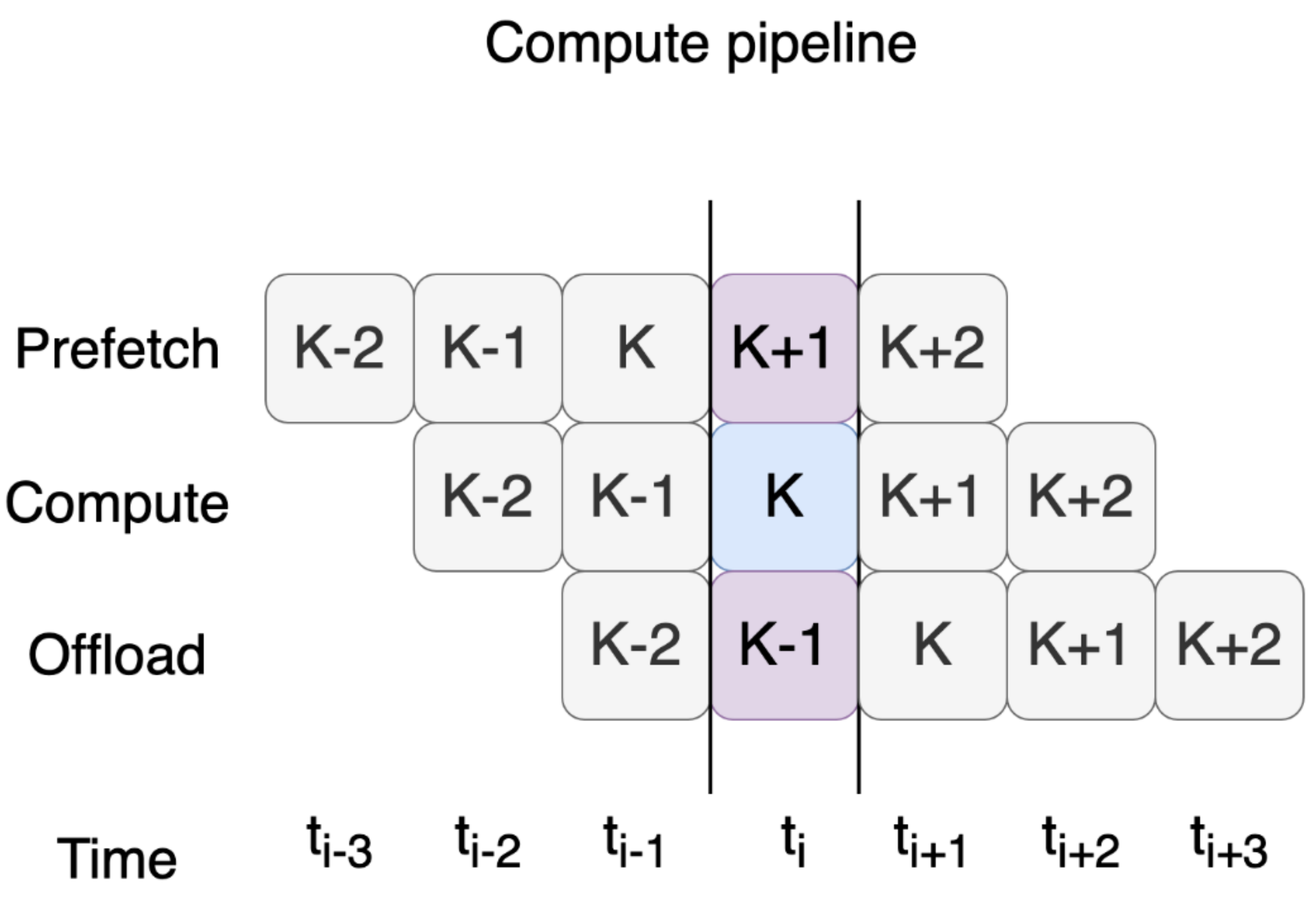
# Study 4: tensor offloading

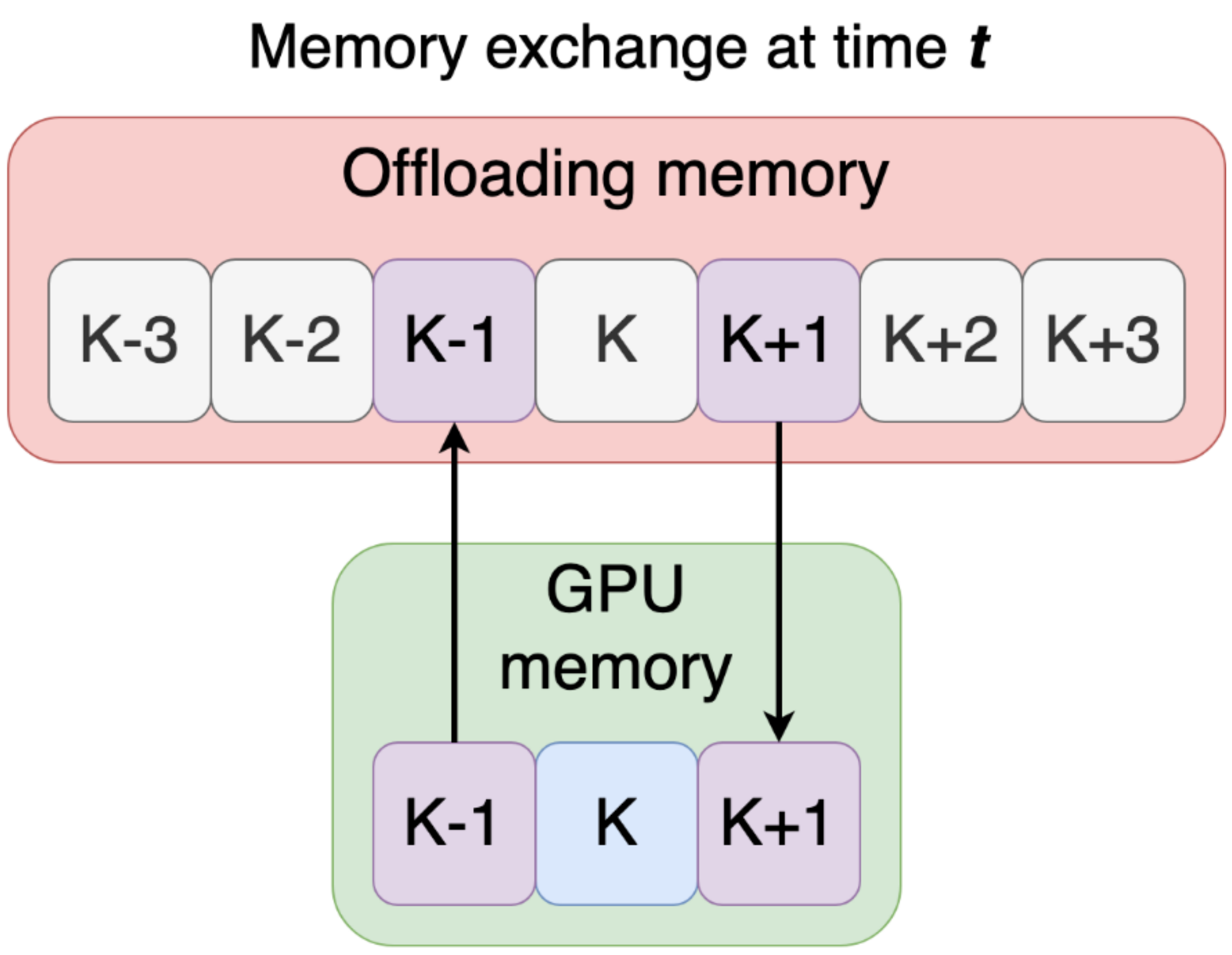
Analyzing an algorithmic design choice that balances parallelization with memory use.



# Study 4: tensor offloading

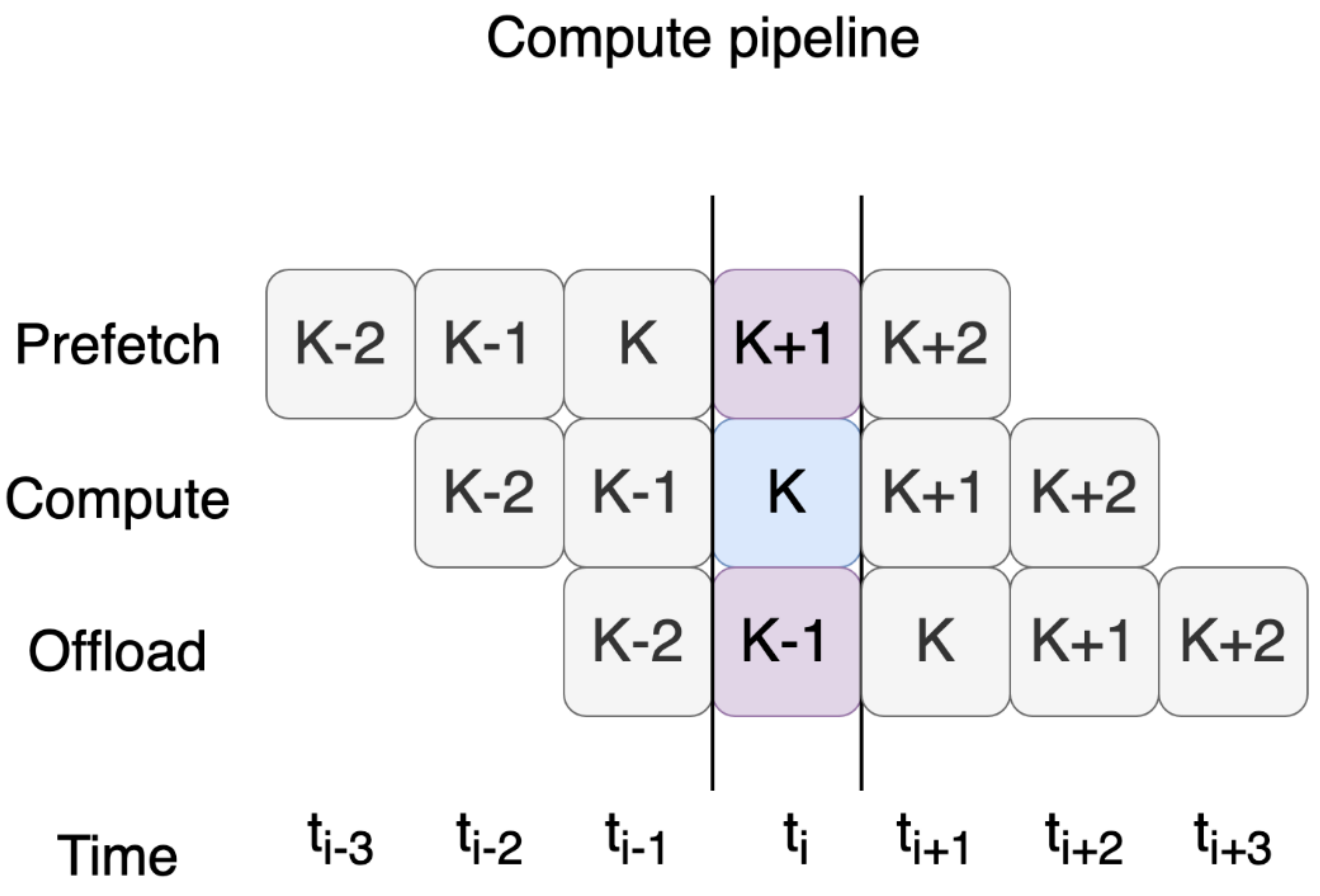
Analyzing an algorithmic design choice that balances parallelization with memory use.

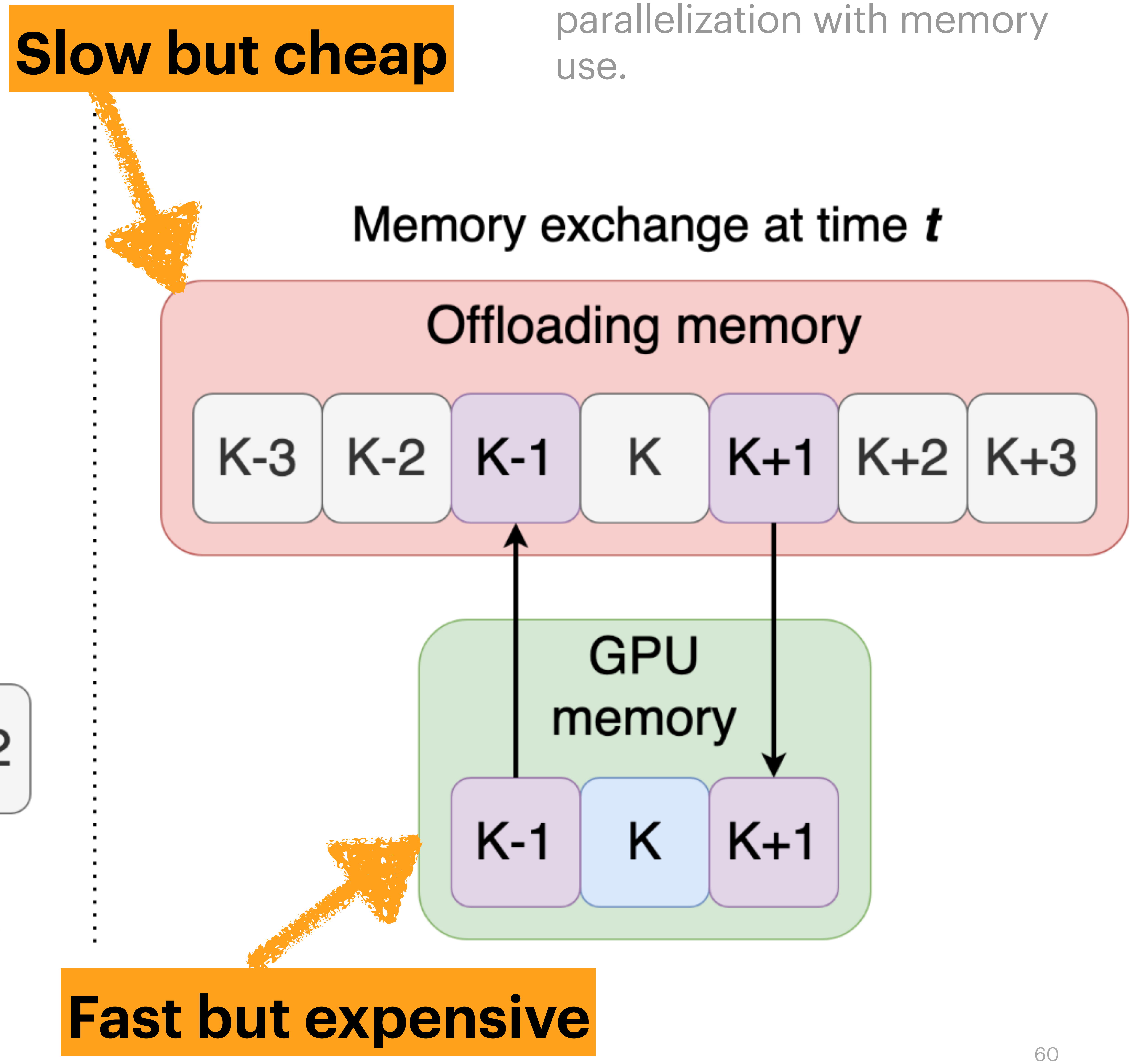


# Study 4: tensor offloading

Analyzing an algorithmic design choice that balances parallelization with memory use.



# Study 4: tensor offloading

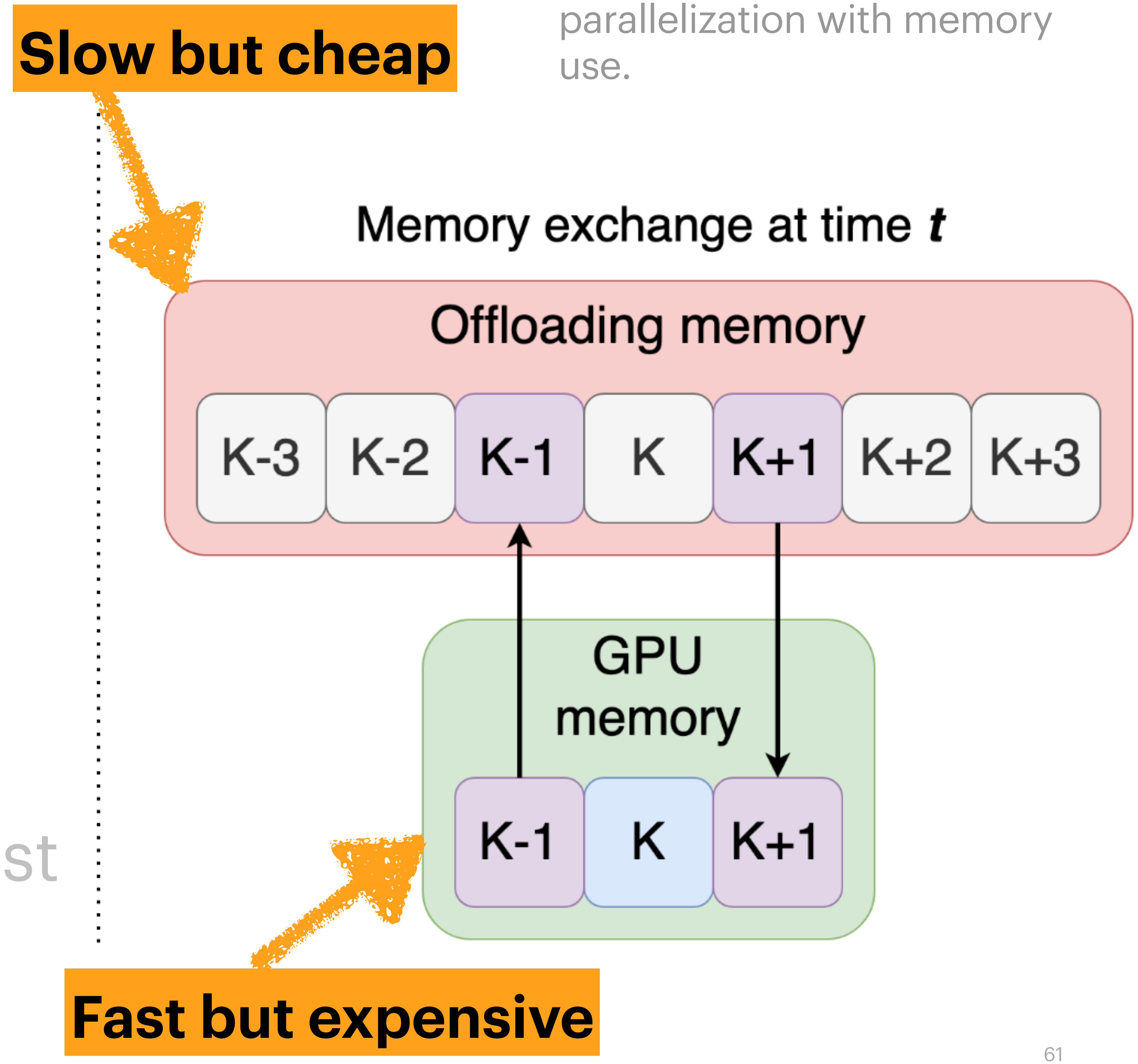
Analyzing an algorithmic design choice that balances parallelization with memory use.

Problem:

**Given a model, a specific GPU configuration, and a maximum GPU memory,**

pick the parallelization strategy, size & speed of slow memory

so that communication time is just hidden.



# Study 4: tensor offloading

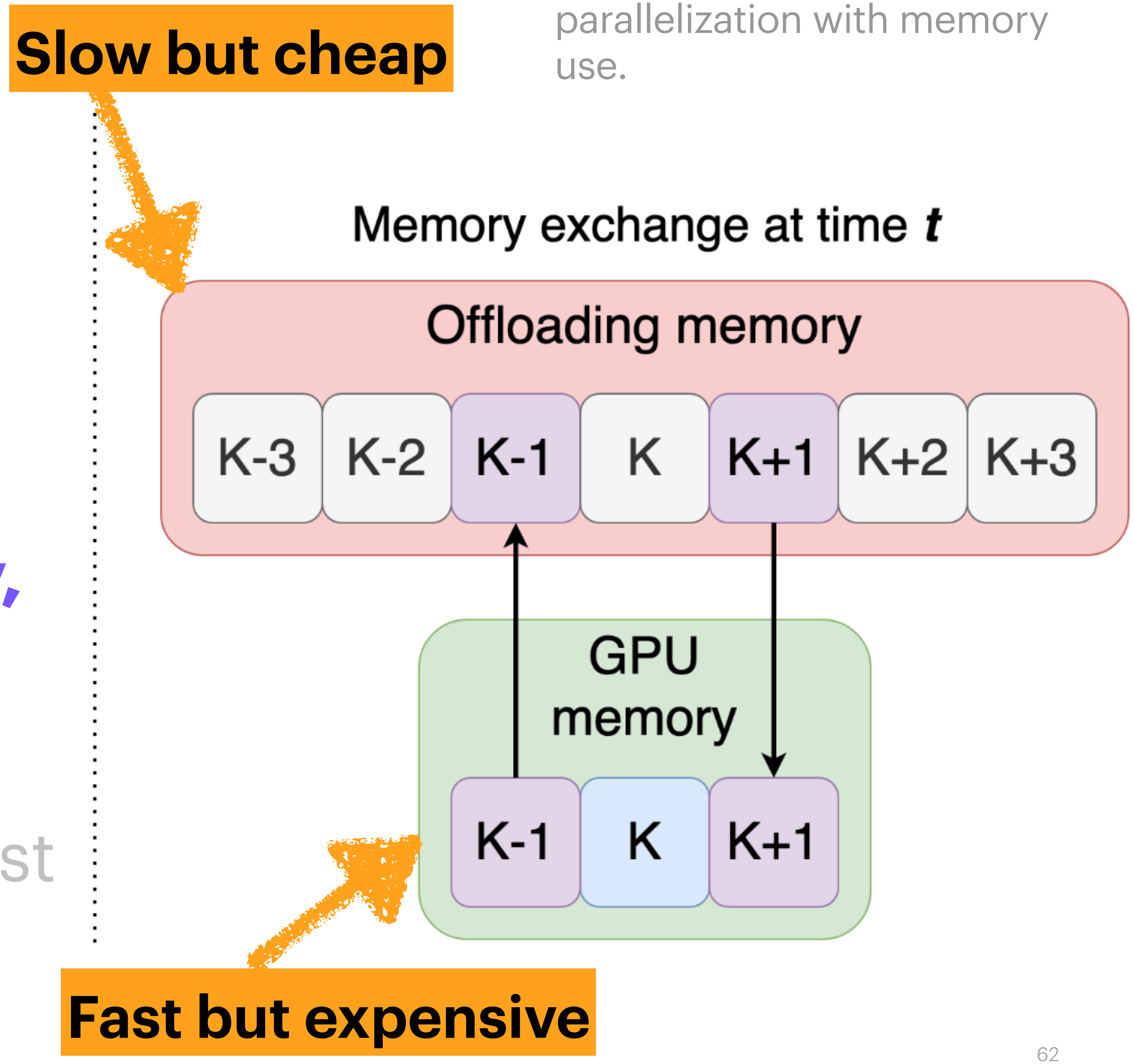
Analyzing an algorithmic design choice that balances parallelization with memory use.

Problem:

Given a model, a specific GPU configuration, and a maximum GPU memory,

**pick the parallelization strategy, size & speed of slow memory**

so that communication time is just hidden.



# Study 4: tensor offloading

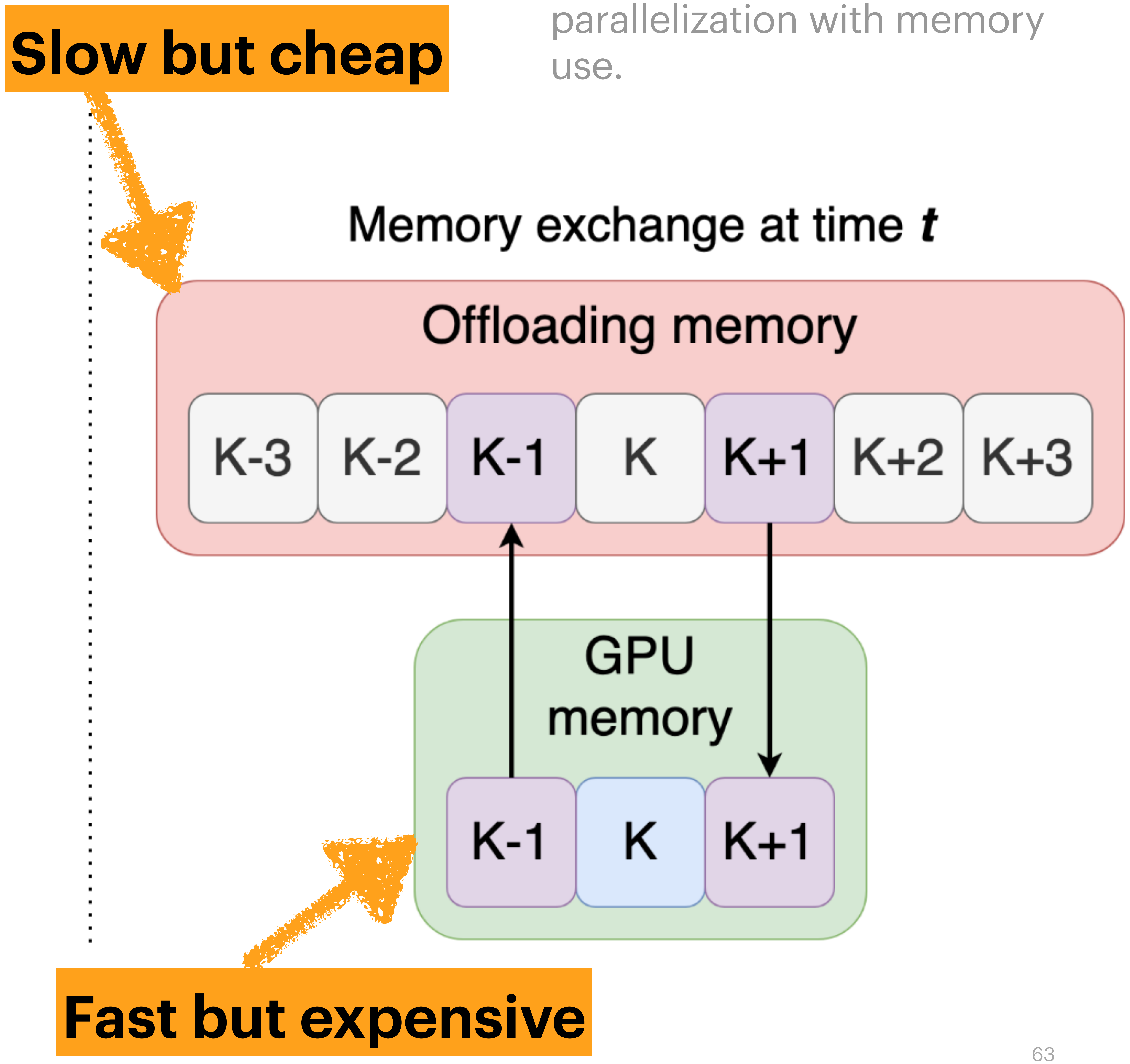
Analyzing an algorithmic design choice that balances parallelization with memory use.

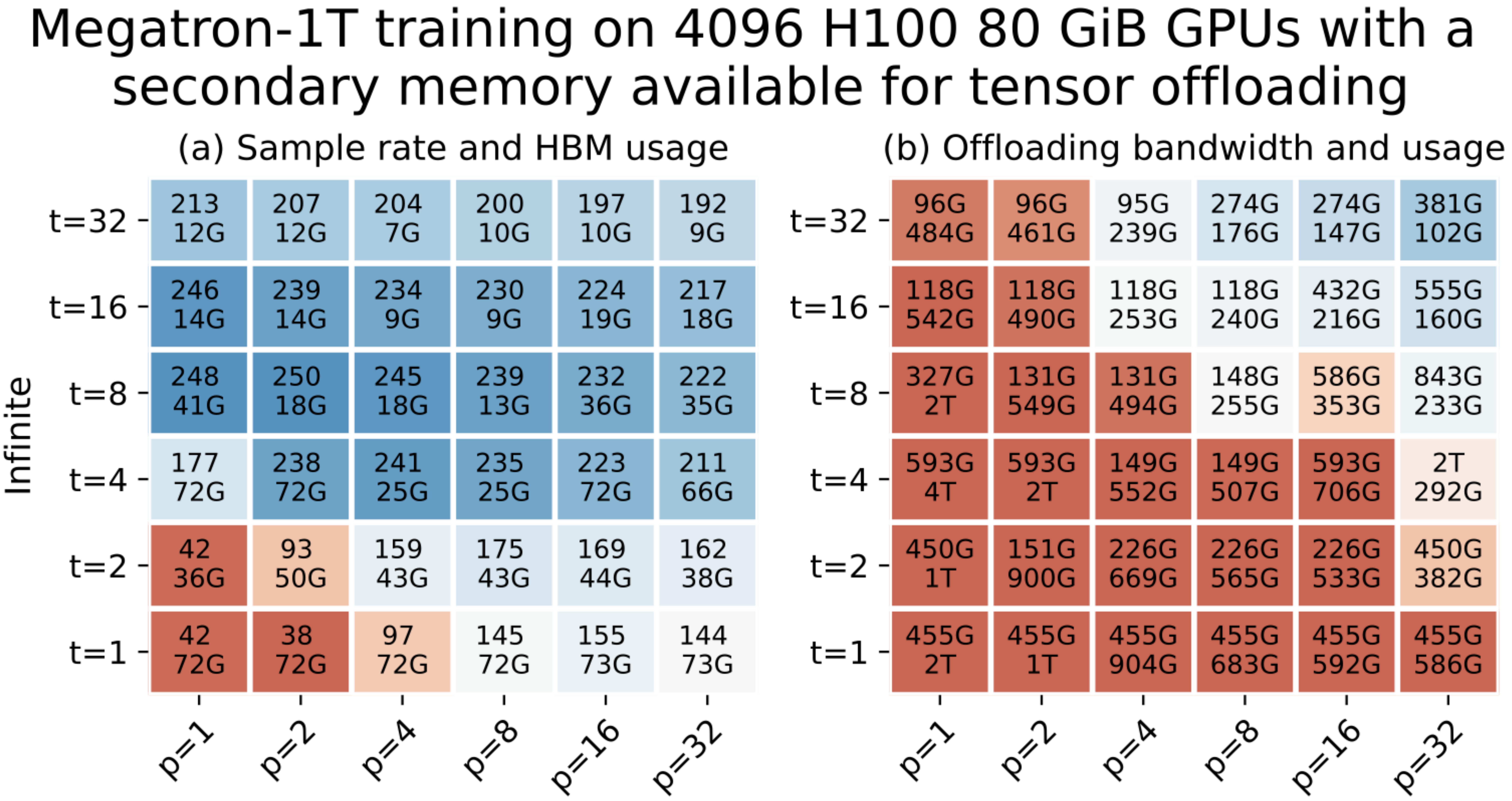
Problem:

Given a model, a specific GPU configuration, and a maximum GPU memory,

pick the parallelization strategy, size & speed of slow memory

**so that communication time is just hidden.**

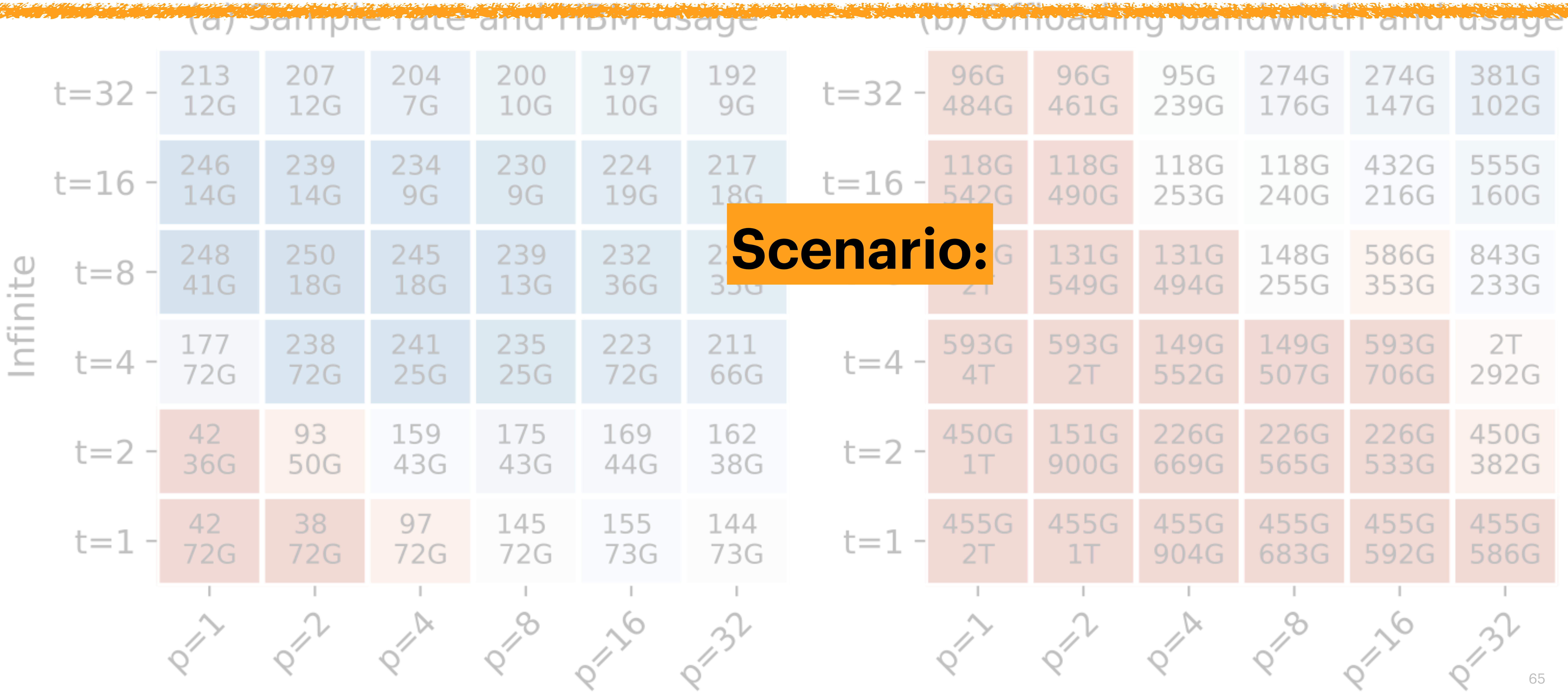




Megatron-1T training on 4096 H100 80 GiB GPUs with a secondary memory available for tensor offloading
(a) Sample rate and HBM usage
(b) Offloading bandwidth and usage
Infinite
t=32
213 12G
207 12G
204 7G
200 10G
197 10G
192 9G
96G 484G
96G 461G
95G 239G
274G 176G
274G 147G
381G 102G
t=16
246 14G
239 14G
234 9G
230 9G
224 19G
217 18G
118G 542G
118G 490G
118G 253G
118G 240G
432G 216G
555G 160G
t=8
248 41G
250 18G
245 18G
239 13G
232 36G
222 35G
327G 2T
131G 549G
131G 494G
148G 255G
586G 353G
843G 233G
t=4
177 72G
238 72G
241 25G
235 25G
223 72G
211 66G
593G 4T
593G 2T
149G 552G
149G 507G
593G 706G
2T 292G
t=2
42 36G
93 50G
159 43G
175 43G
169 44G
162 38G
450G 1T
151G 900G
226G 669G
226G 565G
226G 533G
450G 382G
t=1
42 72G
38 72G
97 72G
145 72G
155 73G
144 73G
455G 2T
455G 1T
455G 904G
455G 683G
455G 592G
455G 586G
p=1
p=2
p=4
p=8
p=16
p=32

# Megatron-1T training on 4096 H100 80 GiB GPUs with a secondary memory available for tensor offloading

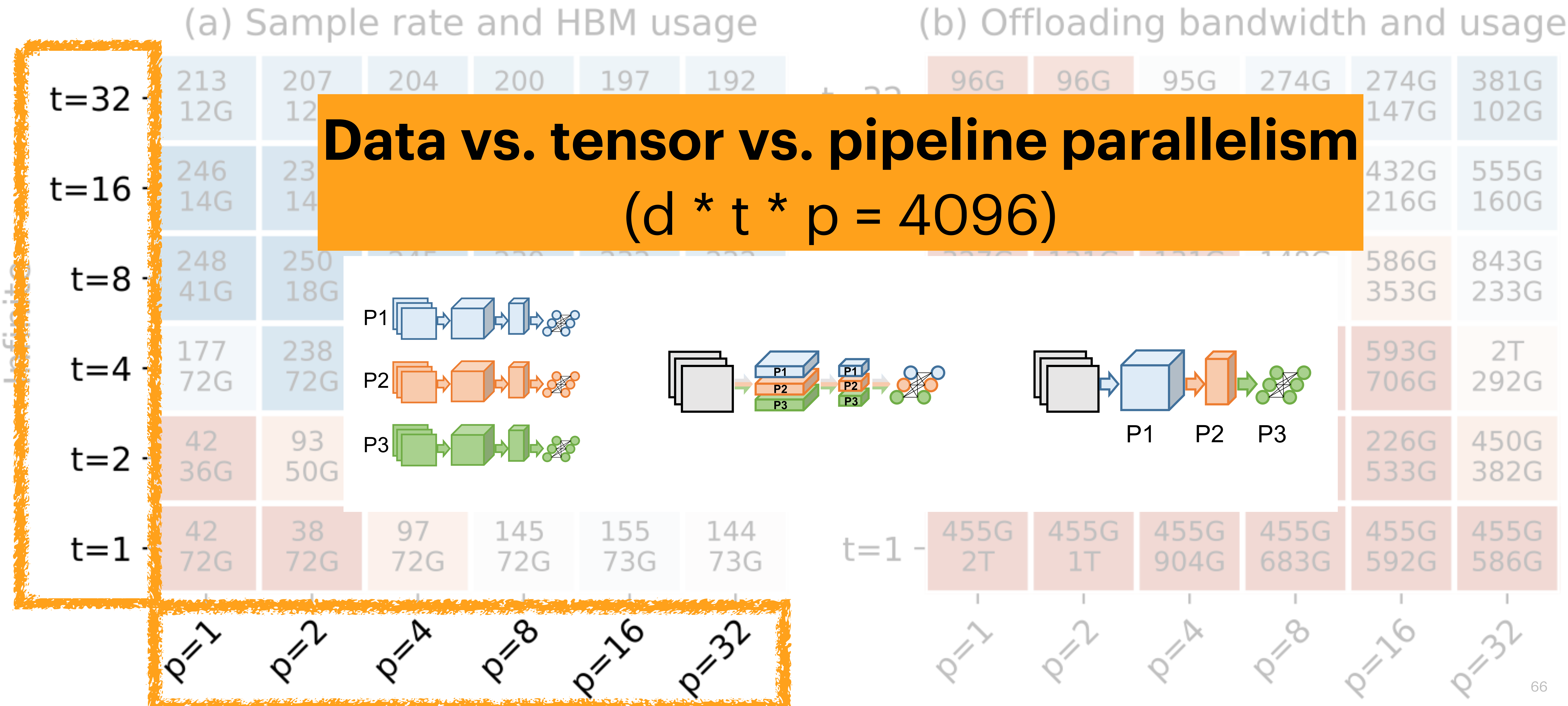
**Scenario:**



Megatron-1T training on **4096 H100 80 GiB GPUs** with a secondary memory available for tensor offloading

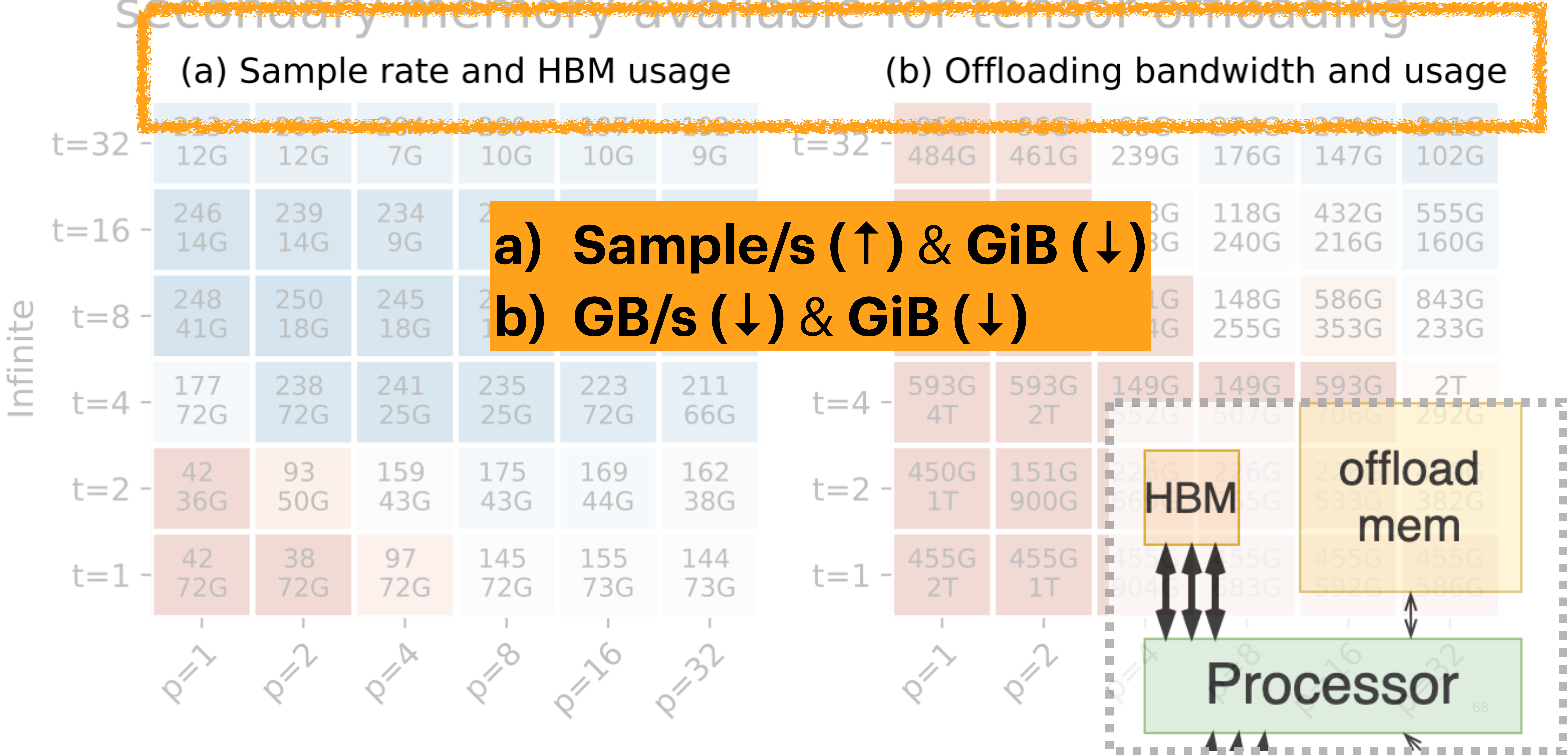
**Data vs. tensor vs. pipeline parallelism**

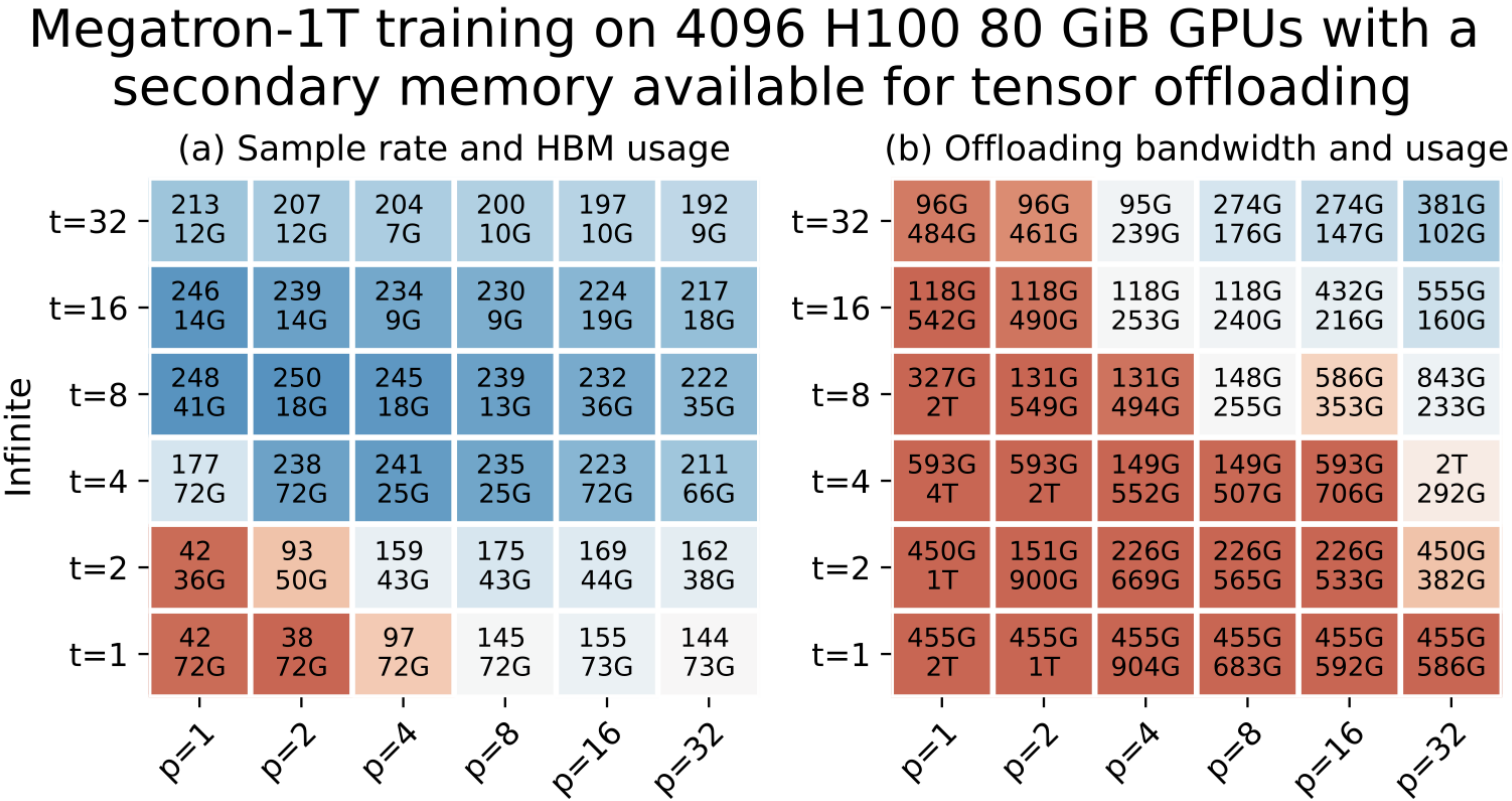
(d * t * p = 4096)




Megatron-1T training on 4096 H100 80 GiB GPUs with a secondary memory available for tensor offloading
(a) Sample rate and HBM usage
(b) Offloading bandwidth and usage
Infinite
Unlimited slow memory
t=32
t=16
t=4
t=2
t=1
p=1
p=2
p=4
p=8
p=16
p=32
213 12G
207 12G
204 7G
200 10G
197 10G
192 9G
246 14G
239 14G
234 9G
230 9G
224 19G
217 18G
248 41G
250 18G
245 18G
177 72G
238 72G
241 25G
235 25G
223 72G
211 66G
42 36G
93 50G
159 43G
175 43G
169 44G
162 38G
42 72G
38 72G
97 72G
145 72G
155 73G
144 73G
96G 484G
96G 461G
95G 239G
274G 176G
274G 147G
381G 102G
118G 542G
118G 490G
118G 253G
118G 240G
432G 216G
555G 160G
148G 255G
586G 353G
843G 233G
593G 4T
593G 2T
450G 1T
151G 900G
455G 2T
455G 1T
HBM
offload mem
Processor

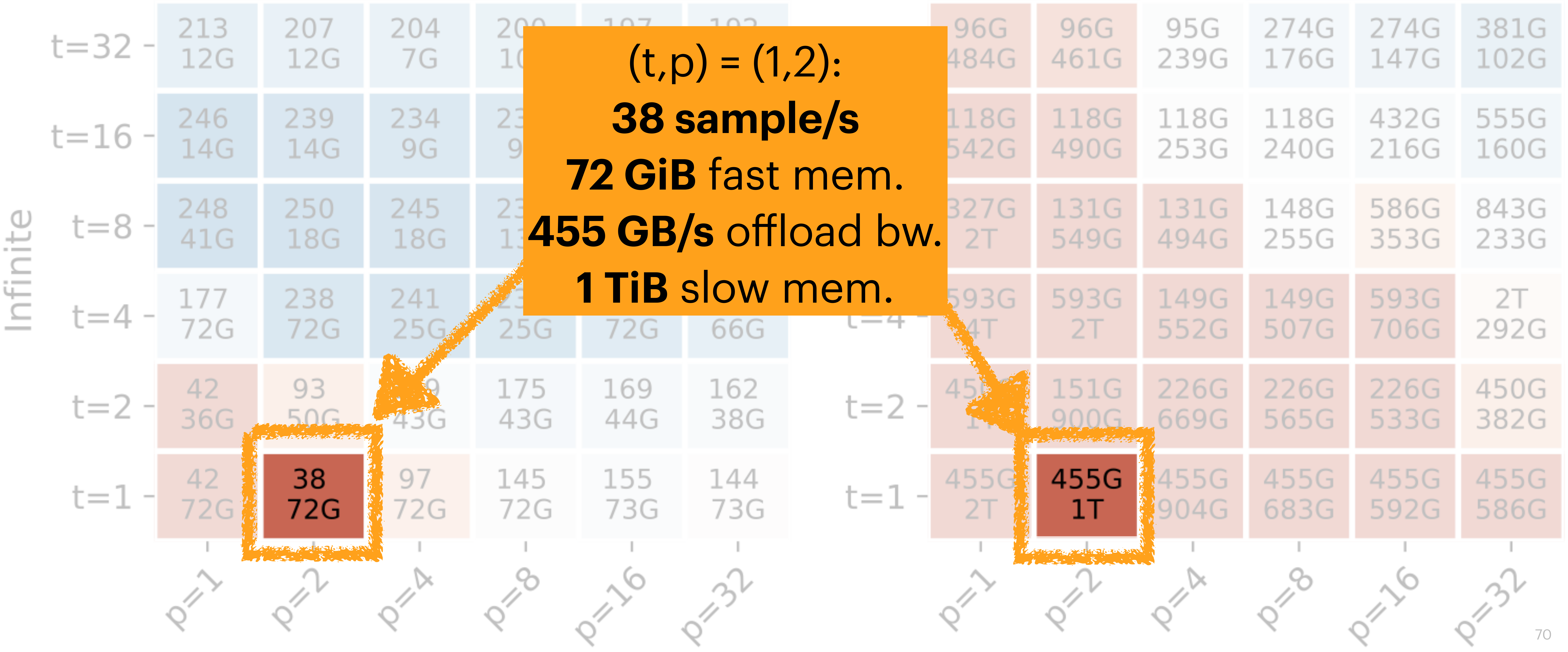
# Megatron-1T training on 4096 H100 80 GiB GPUs with a secondary memory available for tensor offloading




Megatron-1T training on 4096 H100 80 GiB GPUs with a secondary memory available for tensor offloading
(a) Sample rate and HBM usage
(b) Offloading bandwidth and usage
Infinite
t=32
t=16
t=8
t=4
t=2
t=1
p=1
p=2
p=4
p=8
p=16
p=32
213 12G
207 12G
204 7G
200 10G
197 10G
192 9G
246 14G
239 14G
234 9G
230 9G
224 19G
217 18G
248 41G
250 18G
245 18G
239 13G
232 36G
222 35G
177 72G
238 72G
241 25G
235 25G
223 72G
211 66G
42 36G
93 50G
159 43G
175 43G
169 44G
162 38G
42 72G
38 72G
97 72G
145 72G
155 73G
144 73G
96G 484G
96G 461G
95G 239G
274G 176G
274G 147G
381G 102G
118G 542G
118G 490G
118G 253G
118G 240G
432G 216G
555G 160G
327G 2T
131G 549G
131G 494G
148G 255G
586G 353G
843G 233G
593G 4T
593G 2T
149G 552G
149G 507G
593G 706G
2T 292G
450G 1T
151G 900G
226G 669G
226G 565G
226G 533G
450G 382G
455G 2T
455G 1T
455G 904G
455G 683G
455G 592G
455G 586G

# Megatron-1T training on 4096 H100 80 GiB GPUs with a

**Consider the slowest configuration in this space...**



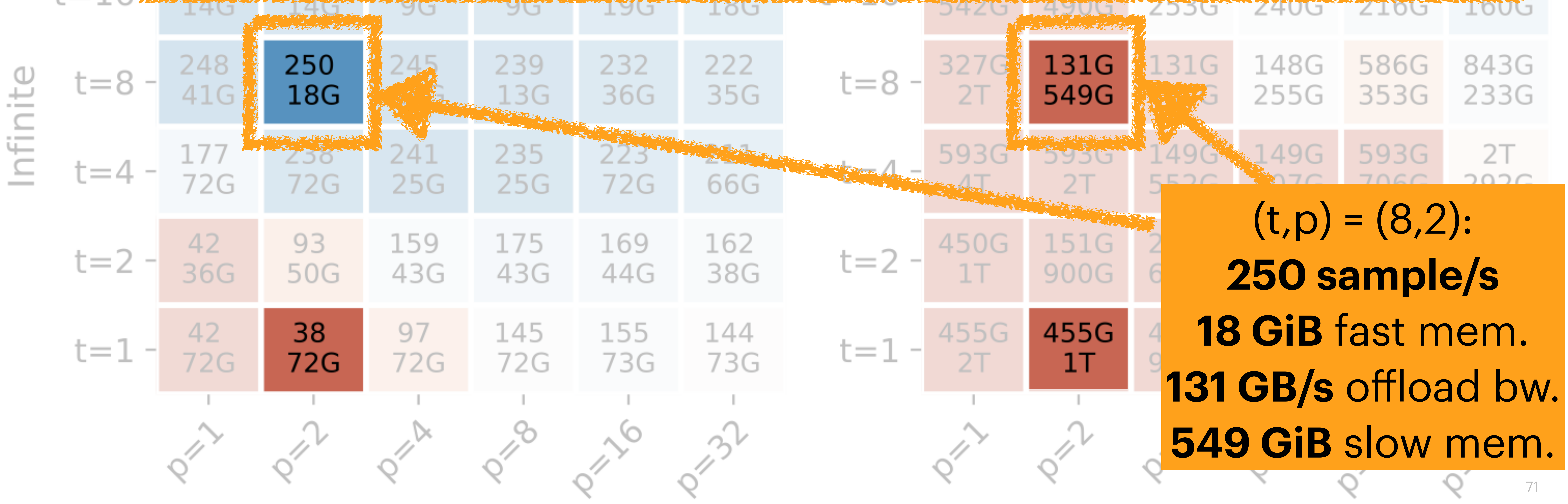
The fastest, by change in parallelization strategy, is:

**6.5x faster**, using

**75% less** fast memory,

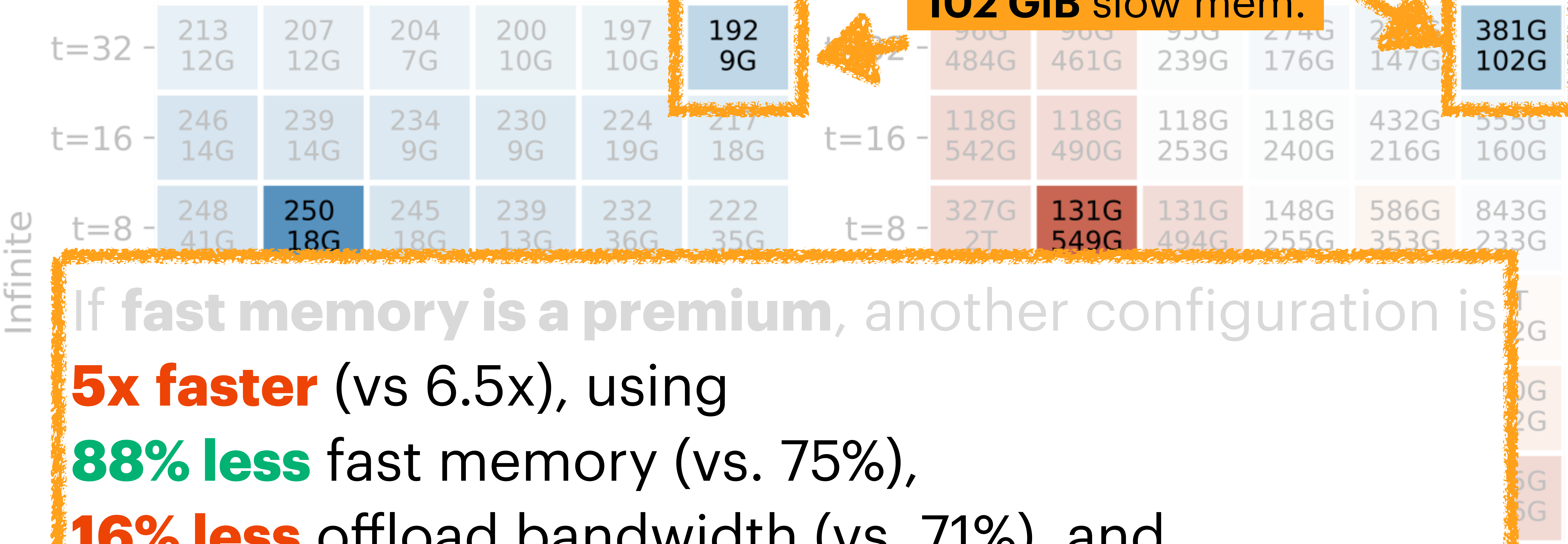
**71% less** offload bandwidth, and

**45% less** slow memory.



# Megatron-1T training on 4096 H… s with a secondary memory available … ading

(a) Sample rate and HBM usage … th and usage




(t,p) = (32,32):
**192 sample/s**
**9 GiB** fast mem.
**381 GB/s** offload bw.
**102 GiB** slow mem.

If **fast memory is a premium**, another configuration is

**5x faster** (vs 6.5x), using

**88% less** fast memory (vs. 75%),

**16% less** offload bandwidth (vs. 71%), and

**29% less** slow memory (vs. 45%).

# Six case studies (recap)

Applying Calculon and interpreting its results.

1. **Comparing data vs. tensor vs. pipeline parallelism**: Can combine and tune to manage time-space tradeoffs
2. **Characterizing the "speed distribution:"** An optimal configuration can be a "needle in a haystack"
3. **Strong scaling analysis**: Speed "cliffs" and "plateaus" exist due to "awkward mappings"
4. **Offload memory**: Slow, "low" bandwidth memory can dramatically reduce fast memory capacity requirements
5. **Price-performance analysis**: How should you set up your next $100 million system?
6. **Next-gen models**: What happens at **100 trillion parameters** (and beyond)?

doi:10.1109/IPDPS.2010.5470417



# Aside:

# Algorithmic aspects of overlapping computation & communication

communication steps will be needed to rearrange the data to make it contiguous in the other two dimensions so that 1D FFTs can be performed in those directions as well. In this case, with $2^{28}$ processors, there must be at least $2^{14} \times 2^{14} \times 2^{14} = 2^{42} \approx 4.4 \times 10^{12}$ elements total, which is a much more manageable problem size. The time required for just the computation on our hypothetical exascale machine would be 68.8 $\mu$s. Therefore, we only consider the pencils approach.

For performance models, we consider a simple LogP-based model in which we assume no overlap of communication and computation, and a more complicated model in which there is substantial overlap of communication and computation.

**3.2.1. No Overlap Model.** If we do not consider overlap of communication, we get that each processor computes its portion of the data, and during each communication round has to communicate with $p$ other processors. The corresponding expression for the runtime of the 3D FFT using the LogP performance model is

$$T = t_c \frac{N}{P} \log_2 N + 2(p-1)(L+o) + 2(p-2)g$$

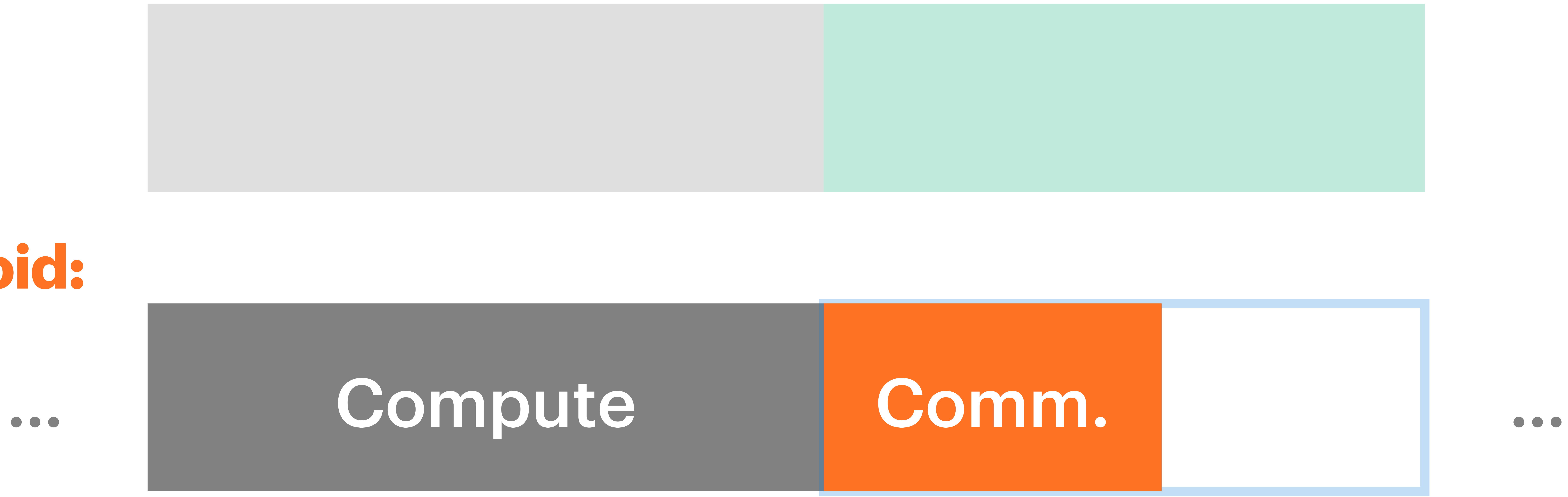
Note that we do not do any latency-hiding, because we treat the latency here as the cost to send the entire message, not just the first word.

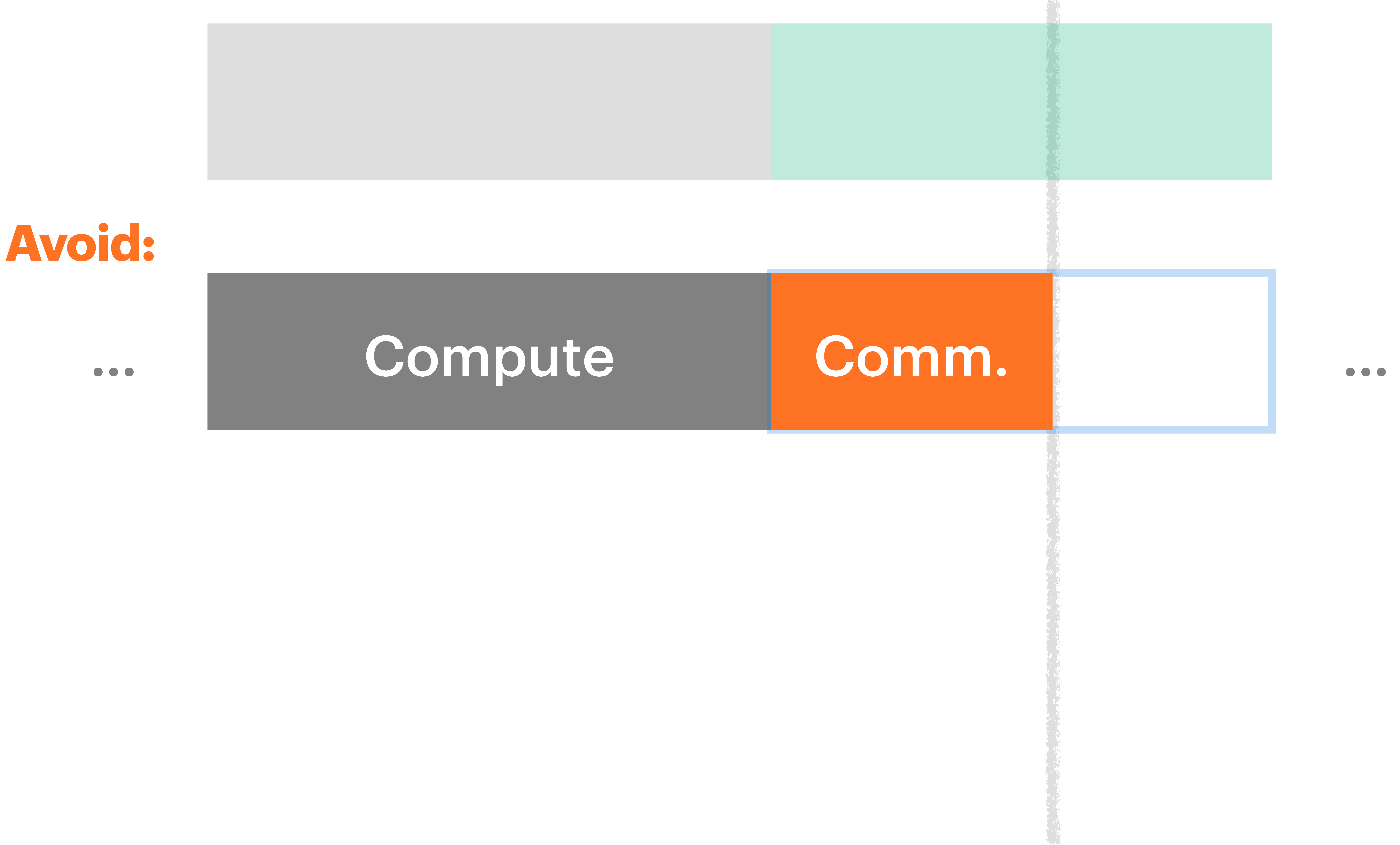
**3.2.2. Overlap Model.** Now allowing overlap of communication and computation, we set up another performance model, using instead of LogP the LogGP model [10] which extends it by adding a bandwidth term $G$ that represents a per-unit cost of transferring data over the network. The model assumes that one $n \times \frac{n}{p}$ sheet is computed at a time, with communication of each sheet occurring after its computation,

See: Isaev, Eswar, V. (**SPAA'25**): "Brief announcement: Optimality conditions for parallel communication-avoiding matrix multiplication with overlapped communication."

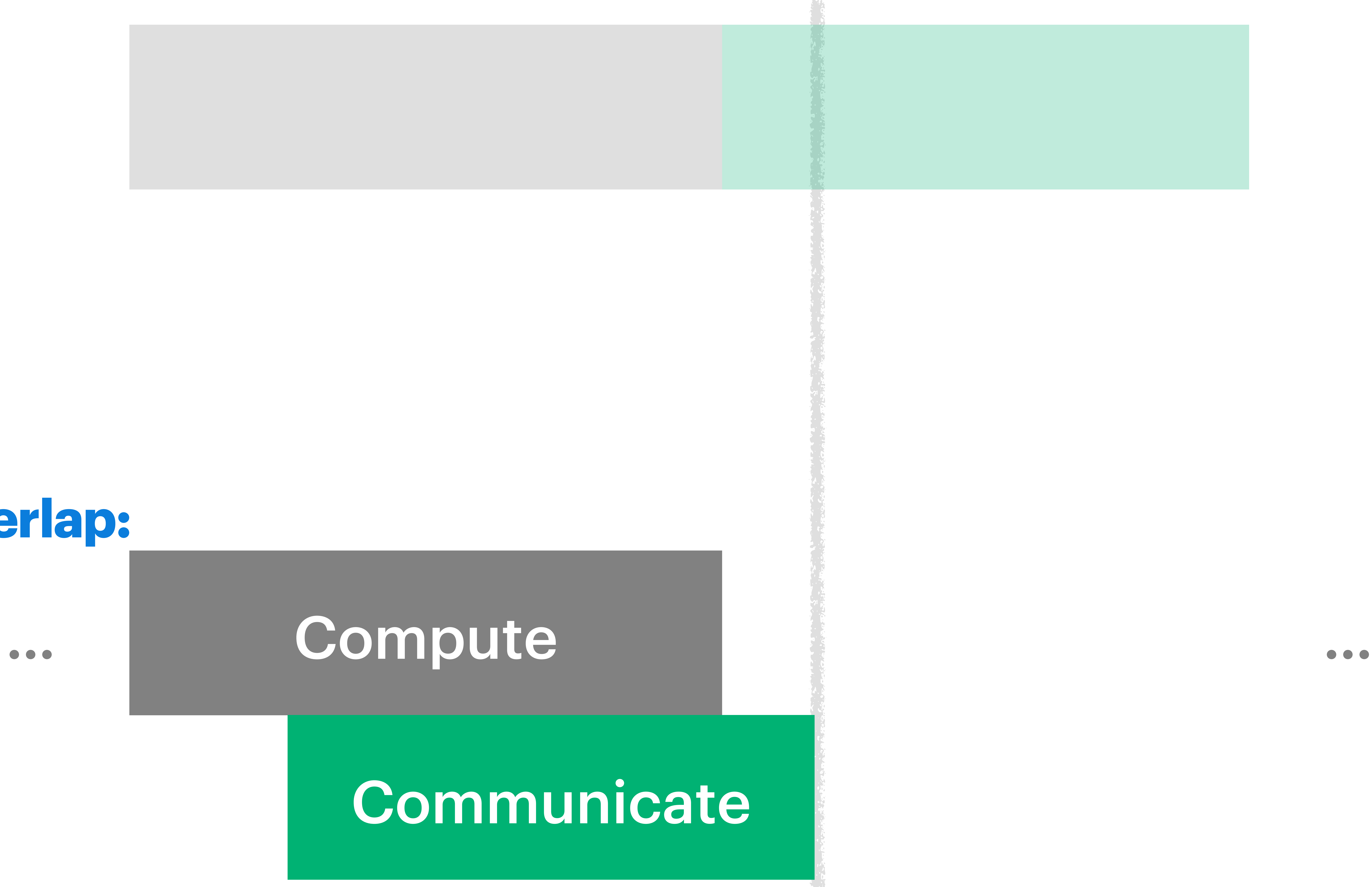
… Compute | Communicate …

**Avoid:**

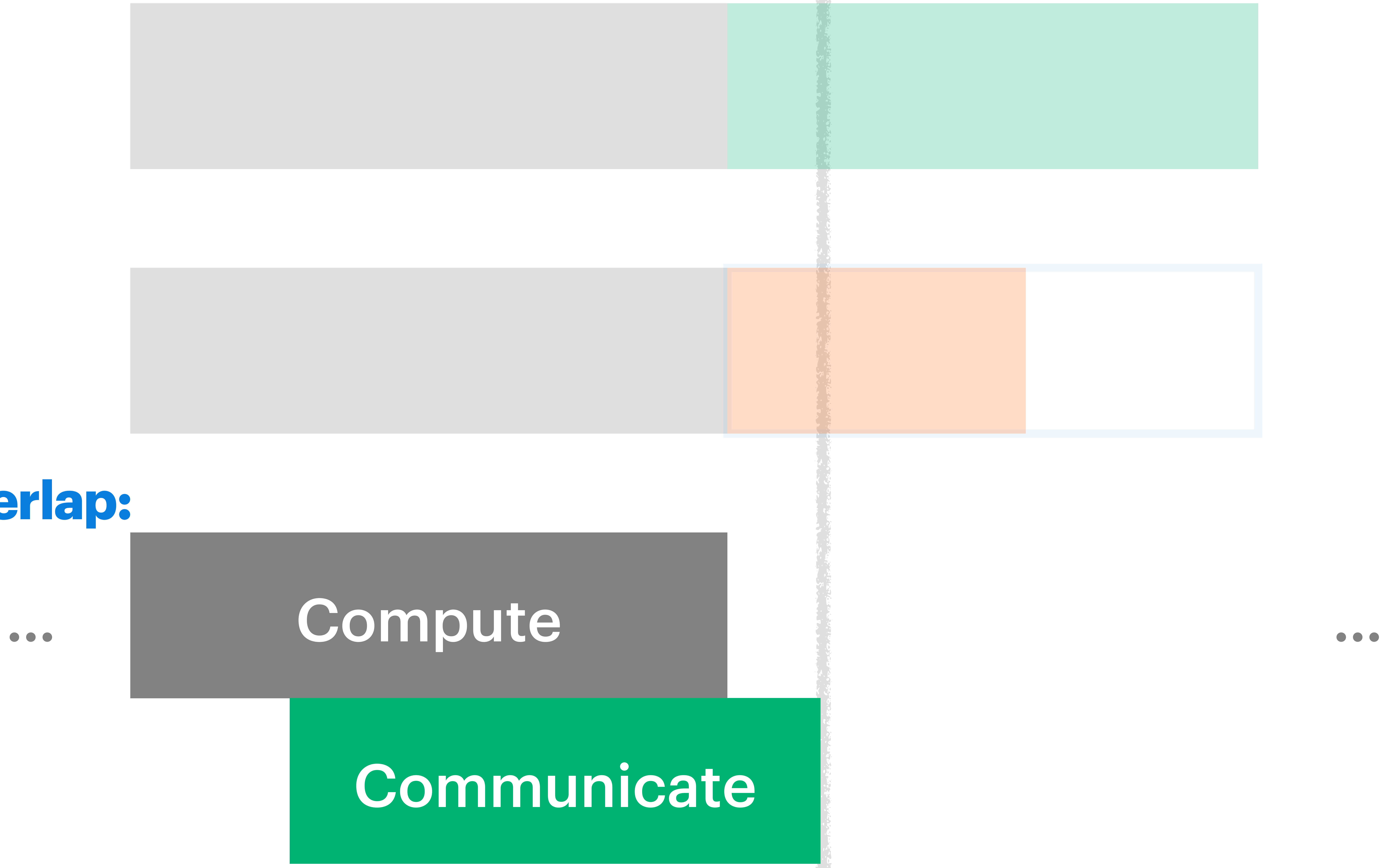



Avoid:
...
Compute
Comm.
...

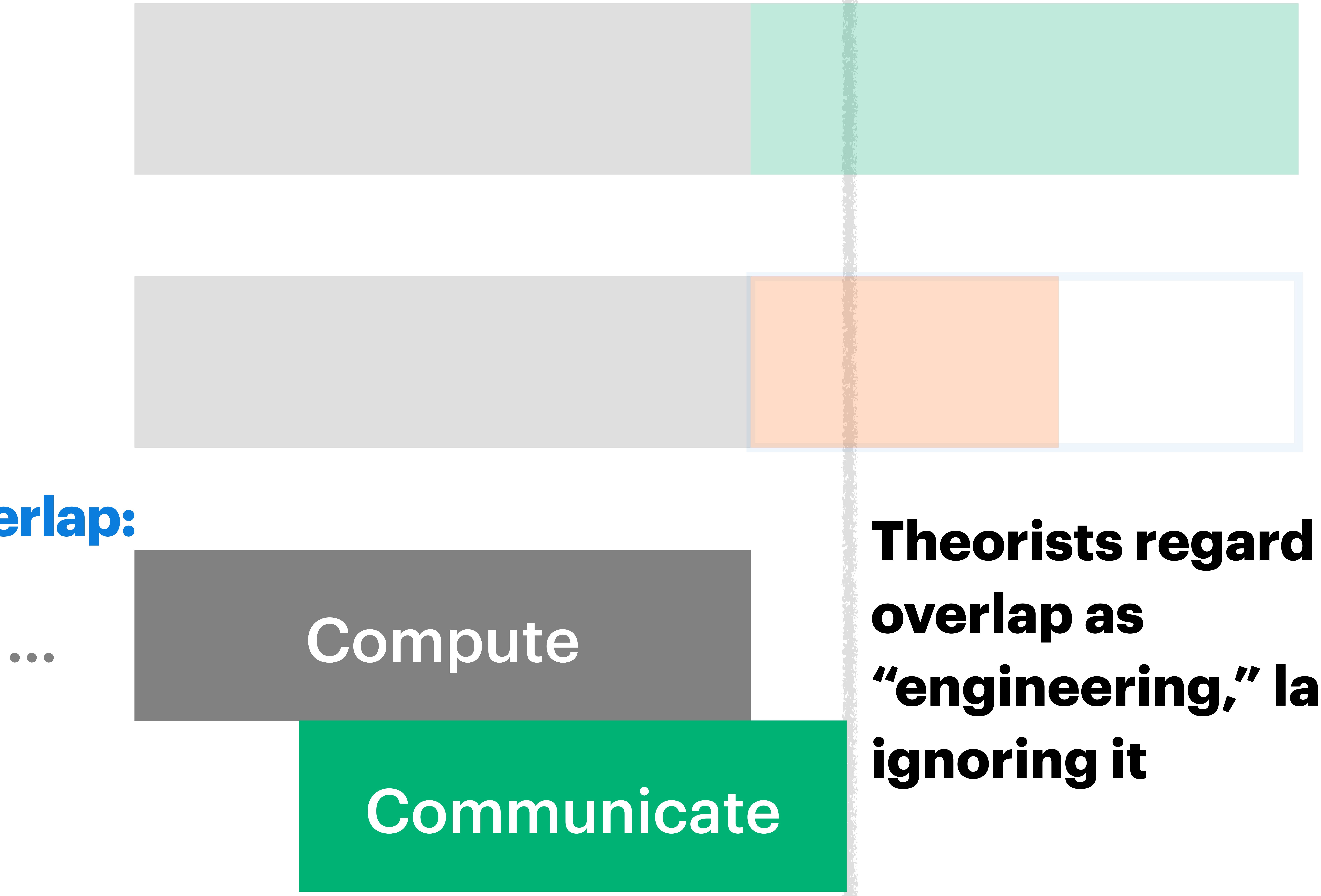
**Overlap:**



**Overlap:**



**Overlap:**




**Theorists regard overlap as "engineering," largely ignoring it**

# **Matrix multiply: 1D vs. 2D vs. 3D** ("comm. avoiding") **algorithms**

Comm volume

# Matrix multiply: 1D vs. 2D vs. 3D ("comm. avoiding") algorithms

# Matrix multiply: 1D vs. 2D vs. 3D ("comm. avoiding") algorithms


**At this many processes,**
**3D is faster than 2D**

# Now consider overlap and execution time:

# Minimum bandwidths

needed for **perfect** overlap

**GEMM end-to-end time**

$$\beta_{\mathbf{1d}} = \gamma \frac{\mathbf{p}}{\mathbf{d_3}}$$

$$\beta_{\mathbf{2d}} = \gamma \frac{\mathbf{2\sqrt{p}}}{\sqrt{\mathbf{d_2 d_3}}}$$

$$\beta_{\mathbf{3d}} = \gamma \frac{\mathbf{2\sqrt[3]{p}}}{\sqrt[3]{\mathbf{d_1 d_2 d_3}}}$$

system size

1

Comm volume

1D better $\frac{d_3}{d_2}$ 2D better $\frac{d_3 d_2}{d_1^2}$ 3D better

$p$

**GEMM end-to-end time**

**1D better (full comm hiding)**

**2D better (full comm hiding)**

**3D better (*partial* comm hiding)**

$\beta_{\mathbf{1d}} = \gamma \frac{\mathbf{p}}{\mathbf{d_3}}$

$\beta_{\mathbf{2d}} = \gamma \frac{\mathbf{2}\sqrt{\mathbf{p}}}{\sqrt{\mathbf{d_2 d_3}}}$

$\left(\frac{\mathbf{3}}{\mathbf{2}}\right)^{\mathbf{6}} \frac{\mathbf{d_3 d_2}}{\mathbf{d_1^2}}$

$\beta_{\mathbf{3d}} = \gamma \frac{\mathbf{2}\sqrt[3]{\mathbf{p}}}{\sqrt[3]{\mathbf{d_1 d_2 d_3}}}$

system size

1

$p$

Comm volume

1D better $\frac{d_3}{d_2}$ 2D better $\frac{d_3 d_2}{d_1^2}$ 3D better

New crossover ~ **11x larger!**

# Minimum bandwidths

needed for **perfect** overlap

$$\beta_{\mathbf{1d}} = \gamma \frac{\mathbf{p}}{\mathbf{d_3}}$$

$$\beta_{\mathbf{2d}} = \gamma \frac{\mathbf{2\sqrt{p}}}{\sqrt{\mathbf{d_2 d_3}}}$$

$$\beta_{\mathbf{3d}} = \gamma \frac{\mathbf{2\sqrt[3]{p}}}{\sqrt[3]{\mathbf{d_1 d_2 d_3}}}$$

# Matrix multiply: 2D vs. 3D ("comm. avoiding") algorithms

## What’s next?

# Modeling **physical** characteristics of data center

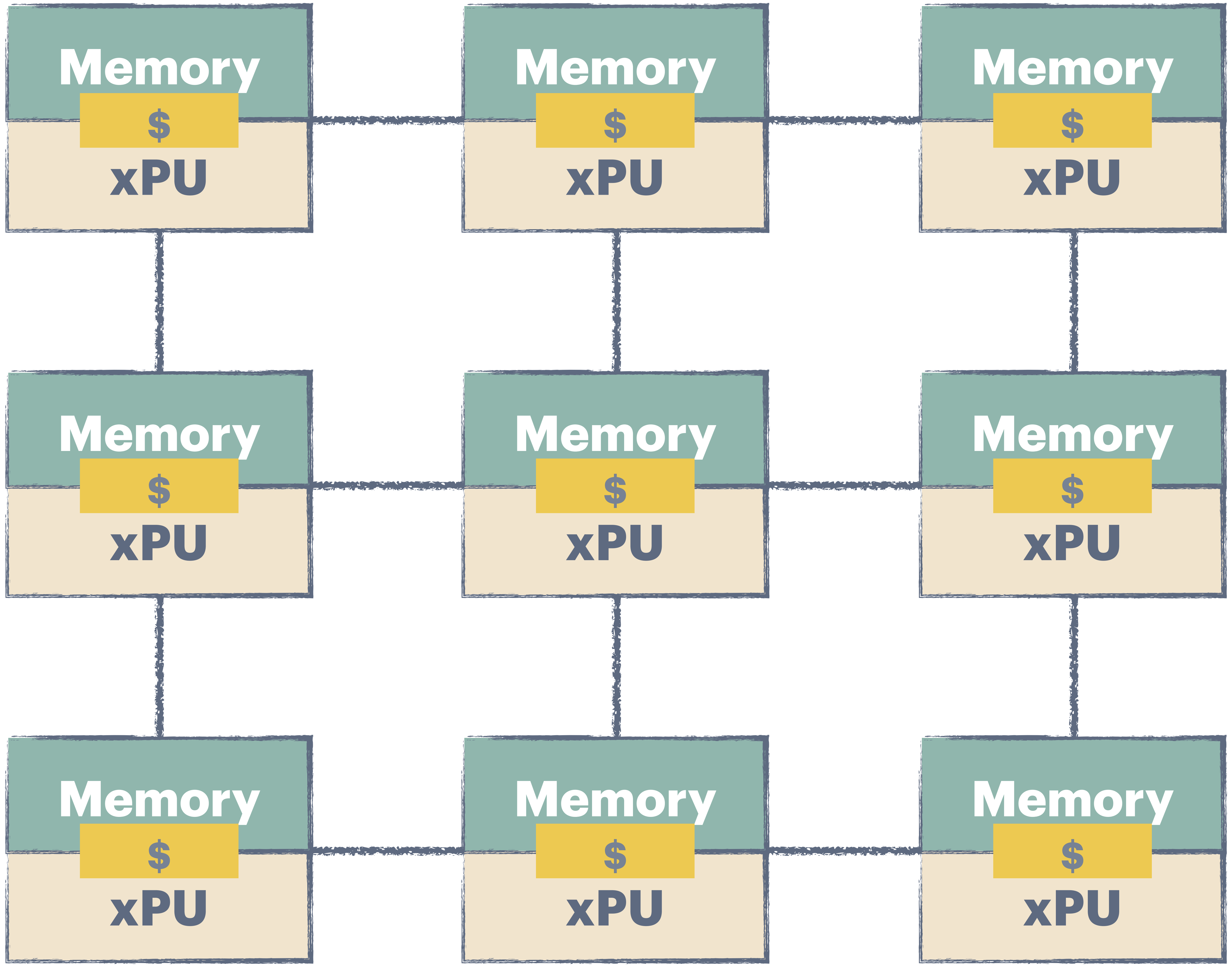
See: Upcoming **HPEC’25** paper on Calculon + **ExaDigiT** (a “data center digital twin” by ORNL)

# What **could** we build?

SUPERCOMPUTERS, REAL & IMAGINED @ ICERM

# What if you **start from scratch?**

# What is "**starting from scratch?**"


Memory
$
xPU
Memory
$
xPU
Memory
$
xPU
Memory
$
xPU
Memory
$
xPU
Memory
$
xPU
Memory
$
xPU
Memory
$
xPU
Memory
$
xPU

Problem:

# Given an algorithm

and a fixed power

& transistor budget,

pick the cores, caches, topology,

& all speeds and feeds

to minimize execution time.

Problem:

Given an algorithm

**and a fixed power**

**& transistor budget,**

**(Constraints!)**

pick the cores, caches, topology,

& all speeds and feeds

to minimize execution time.

Problem:

Given an algorithm
and a fixed power
& transistor budget,
**pick the cores, caches, topology,**
**& all speeds and feeds**
to minimize execution time.

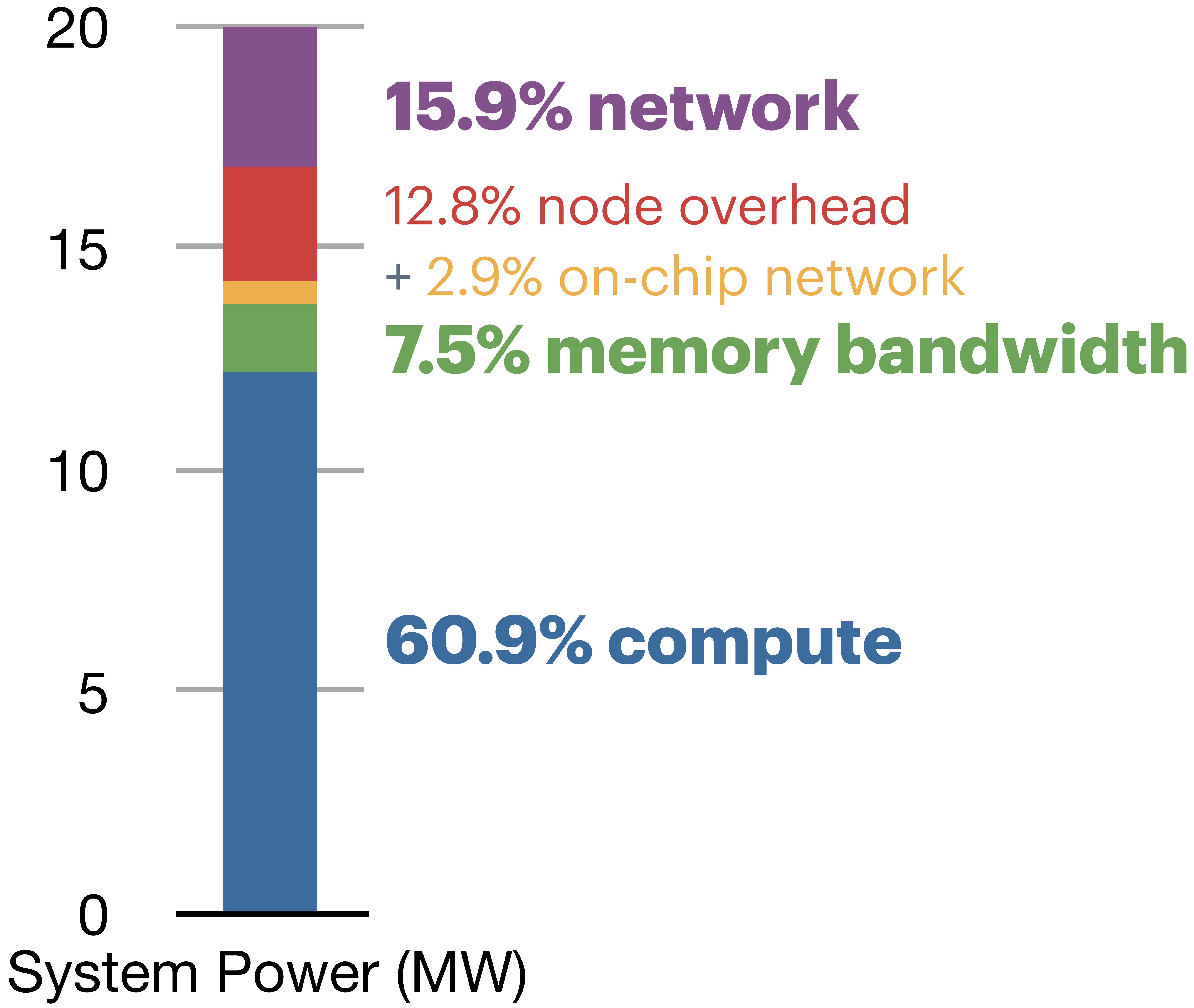
Problem:



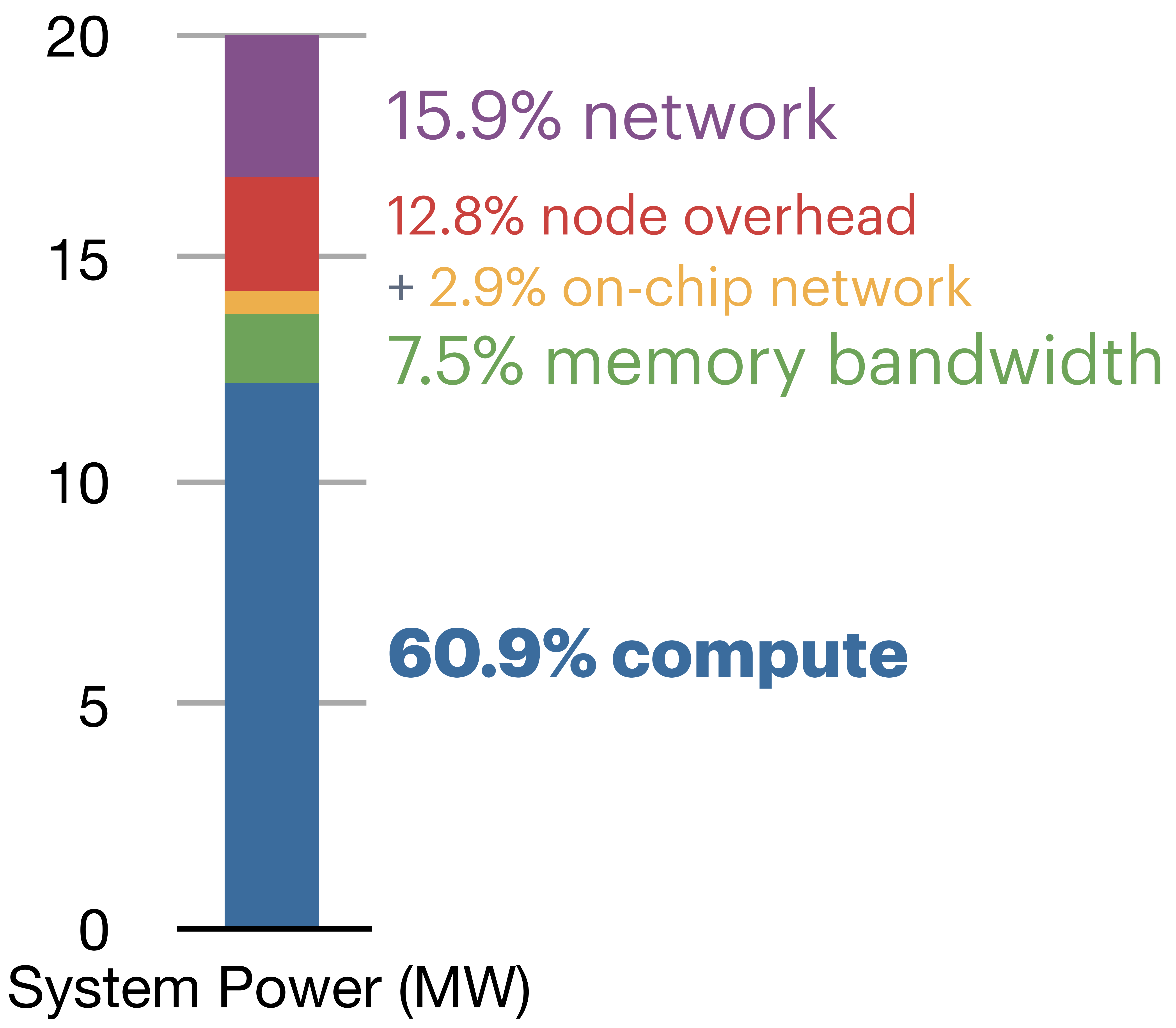
Given an algorithm
and a fixed power
& transistor budget,
pick the cores, caches, topology,
& all speeds and feeds
**to minimize execution time.**

# Power allocation for an “optimal” matrix multiply machine?

# Power allocation for an "optimal" matrix multiply machine



# Power allocation for an "optimal" matrix multiply machine




**ORNL Summit** (13-14 MW):

**67.0% GPU compute**

14.9% CPU compute

4.8% memory

5.3% network + disk

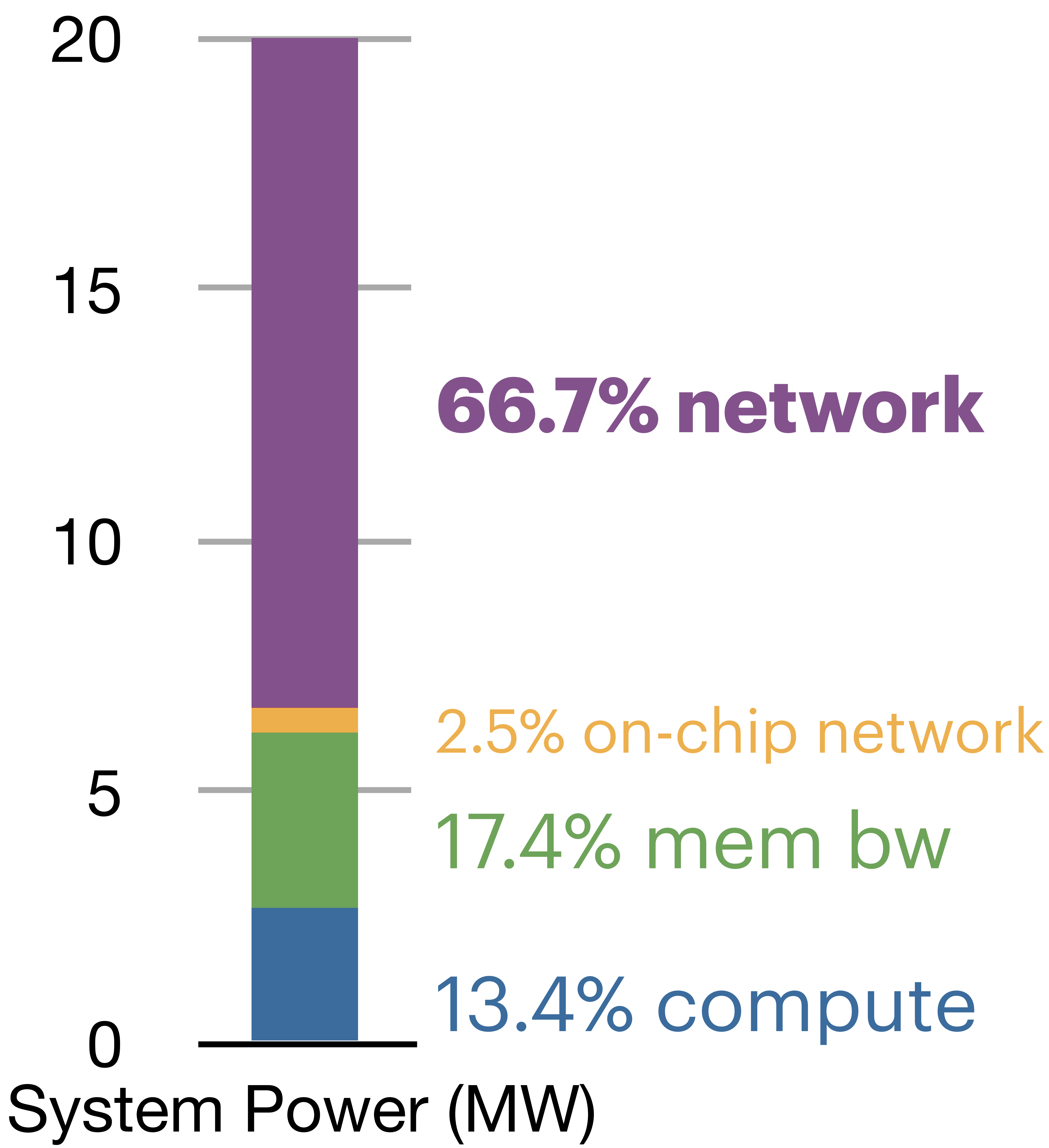
8% node overhead

*P.S.: $R_{max}$ / $R_{peak}$ ~ 75%*

# Power allocation for an “optimal” 3D FFT machine?

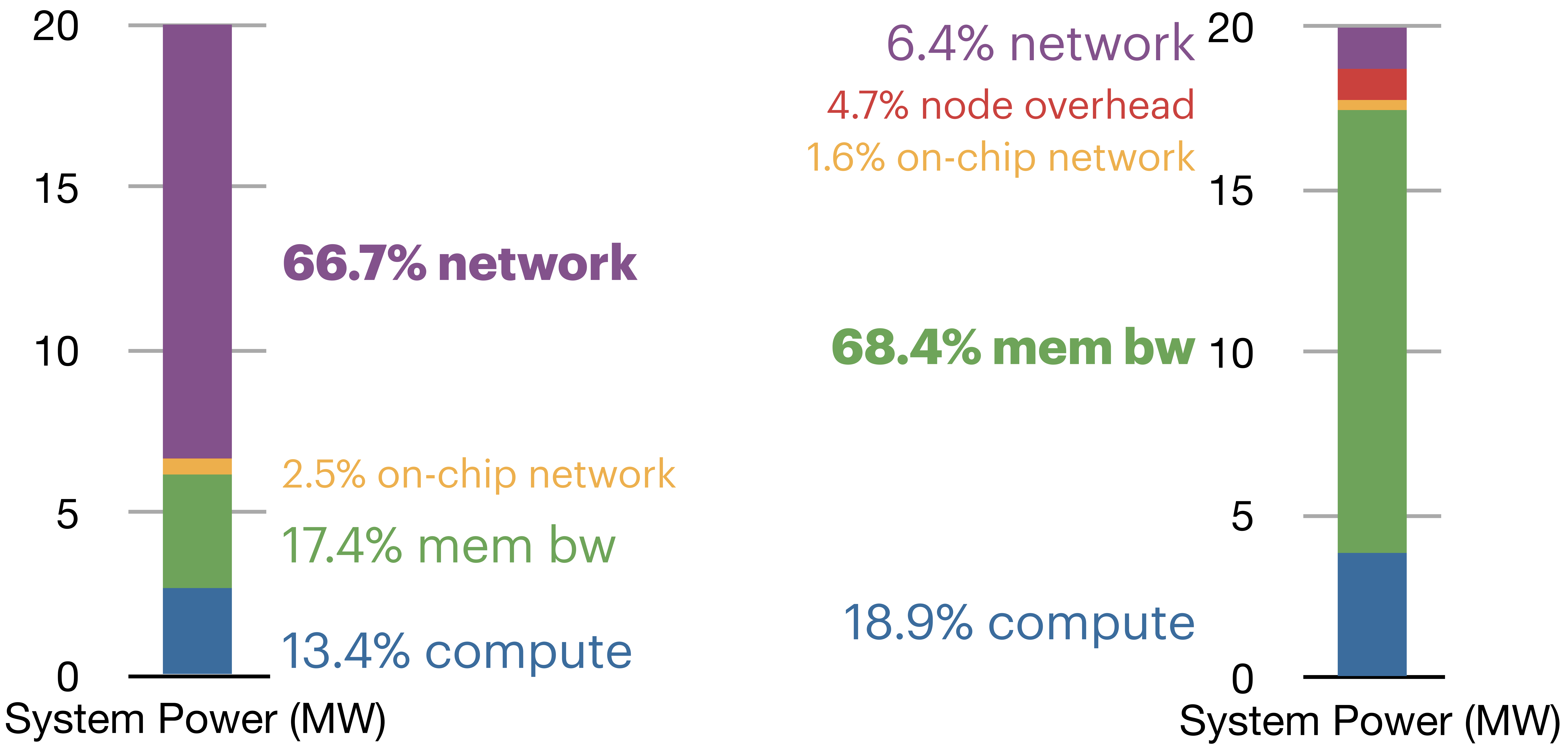
# Power allocation for an "optimal" 3D FFT machine



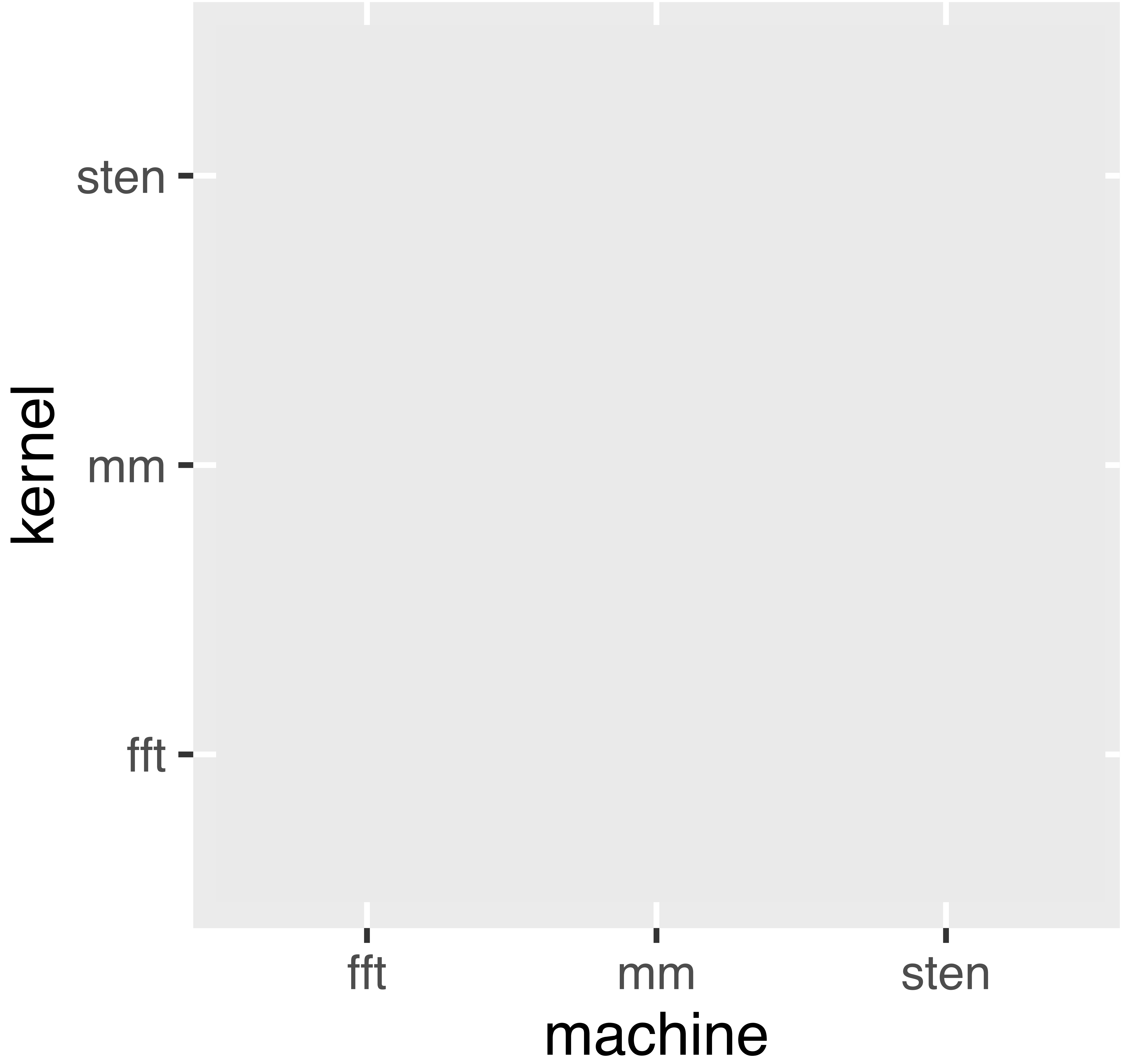


# 3D FFT vs. "Stencil" machines



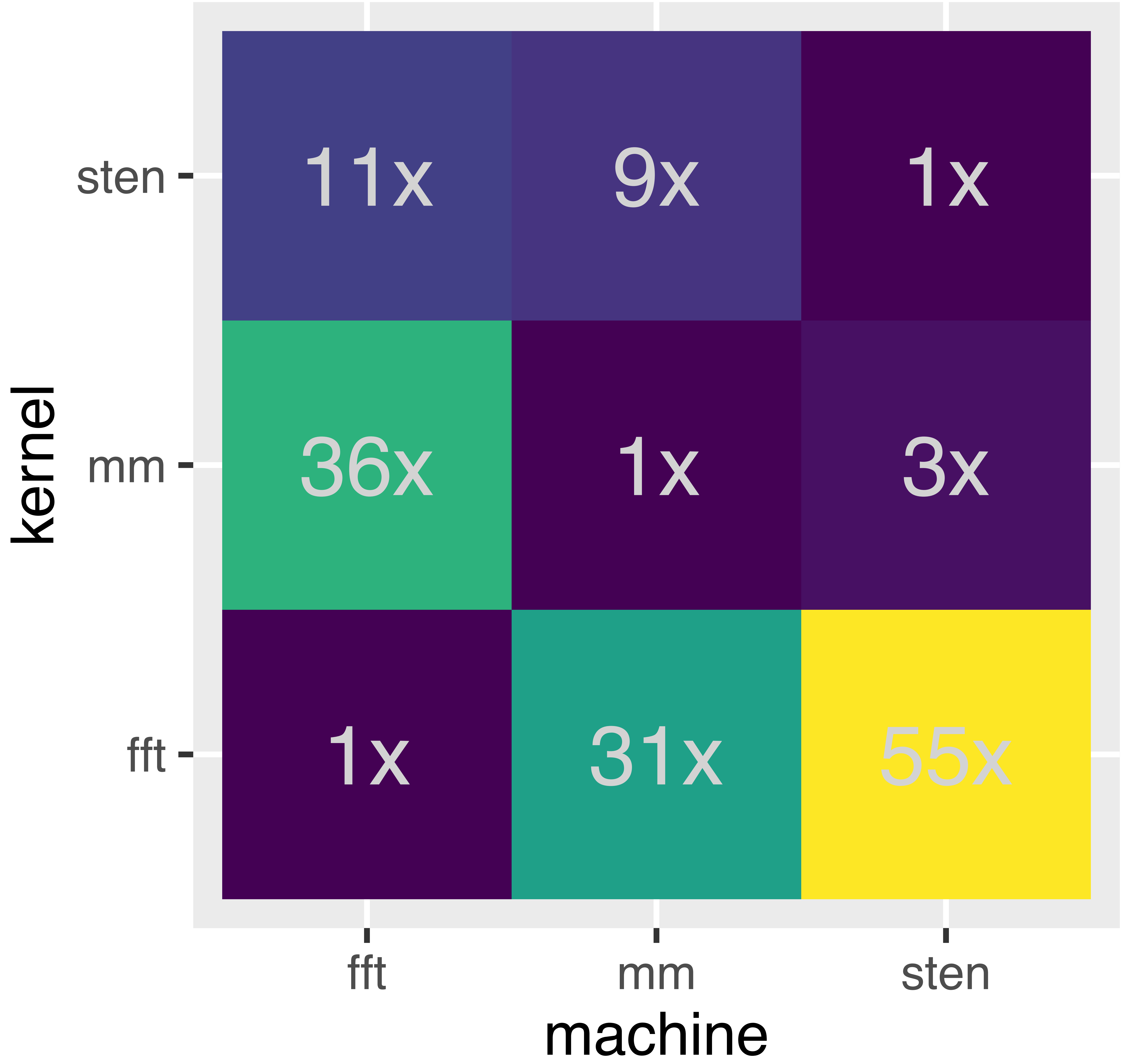


# Relative time (slowdown)



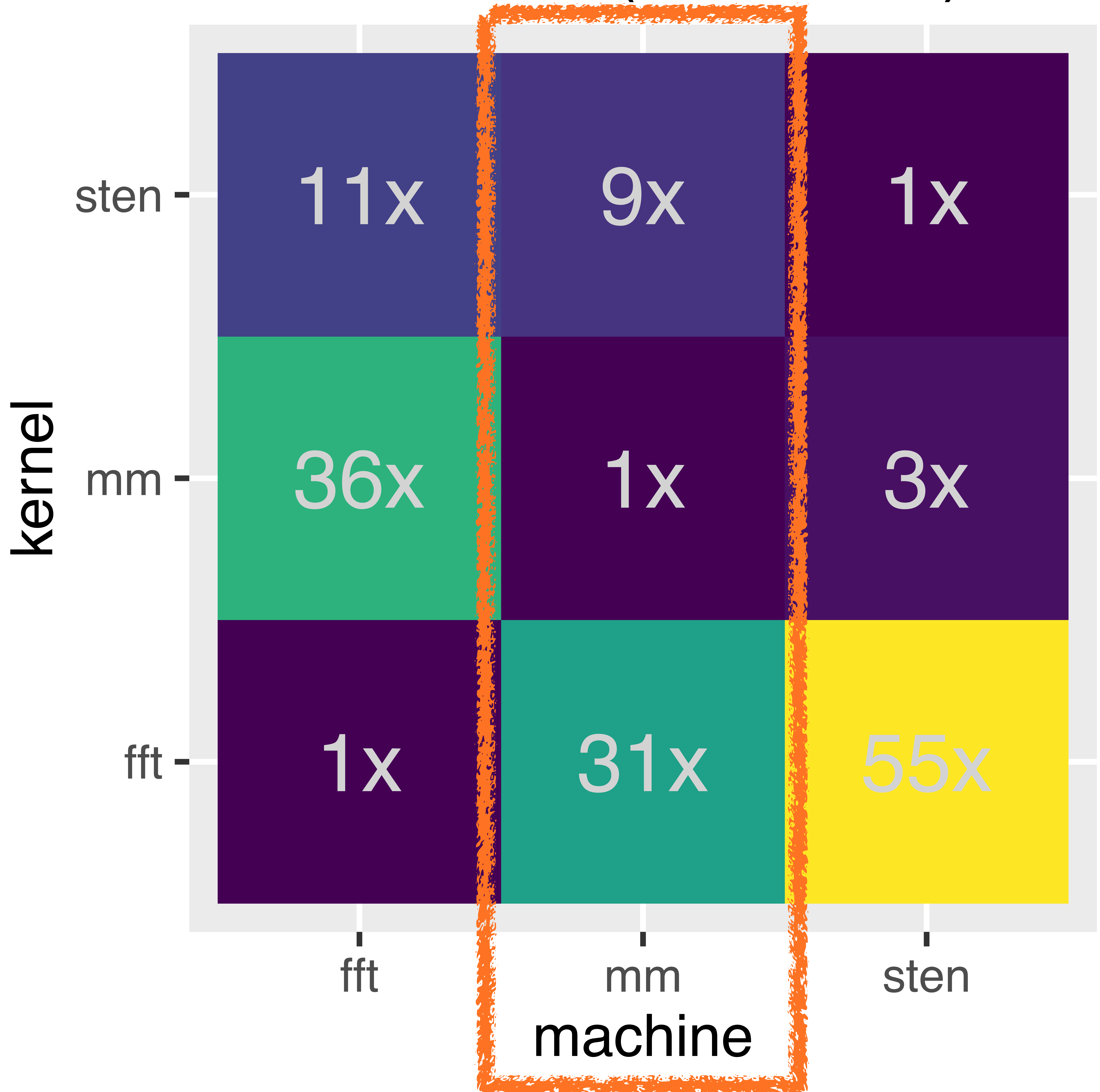
# Relative time (slowdown)





# Relative time (slowdown)





# Demand, supply, … and new demand?

Follow the **money**: Our supercomputers will be machines tuned for AI.

What could such machines look like? A **performance model** might tell you.

What **else** could and should we build???