# The Power (Savings) of Optimized Multi-Grid Solvers

Evan Weinberg, Senior Developer Technology Compute Engineer

NGT Algorithm Workshop, December 9-11, 2024

# Credit and Appreciation

- Thank you to the organizing committee for having me!
- Slides and Content:
  - Kate Clark
  - Balint Joo
  - Vishal Mehta
  - Jiqun Tu
  - Mathias Wagner
- Infinite Conversations and Collaborations
  - See above, and…
  - Peter Boyle
  - Rich Brower
  - Dean Howarth

# Agenda

- Overview
- Multigrid in Theory
- Multigrid as Engineered
- Multigrid on Modern Systems
- Discussion

“

Multigrid: A class of algorithms that mitigate critical slowing down

”

“

Reality: Time (and energy) to solution is the only thing that matters

”

"Energy Efficiency: Move fewer electrons a shorter distance (and accomplish the same goal)"

# Multigrid in Theory

# Why Focus on Multigrid

- LQCD is more than just solving the Dirac linear system over and over again
  - The diversity of topics at this workshop drives this home

# Why Focus on Multigrid

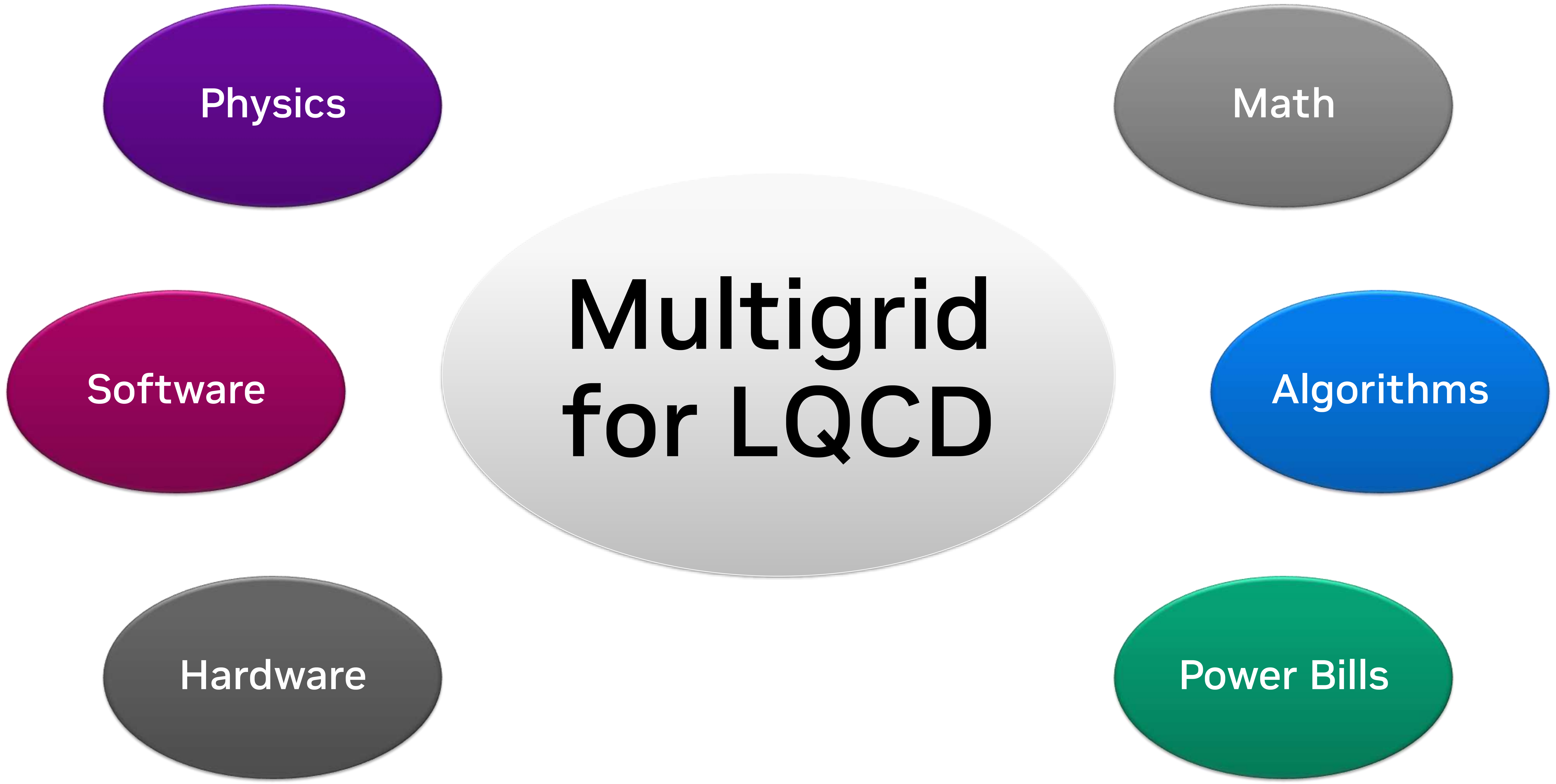
- LQCD is more than just solving the Dirac linear system over and over again
  - The diversity of topics at this workshop drives this home
- But solving the Dirac matrix linear system is still “the” key workflow
  - The diversity of topics just within linear system solvers at this workshop also drives this home

# Why Focus on Multigrid

- LQCD is more than just solving the Dirac linear system over and over again
  - The diversity of topics at this workshop drives this home
- But solving the Dirac matrix linear system is still “the” key workflow
  - The diversity of topics just within linear system solvers at this workshop also drives this home
- And multi-grid solvers are a critical accelerator thereof
  - …The diversity of topics within multi-grid solvers also-also drives this home

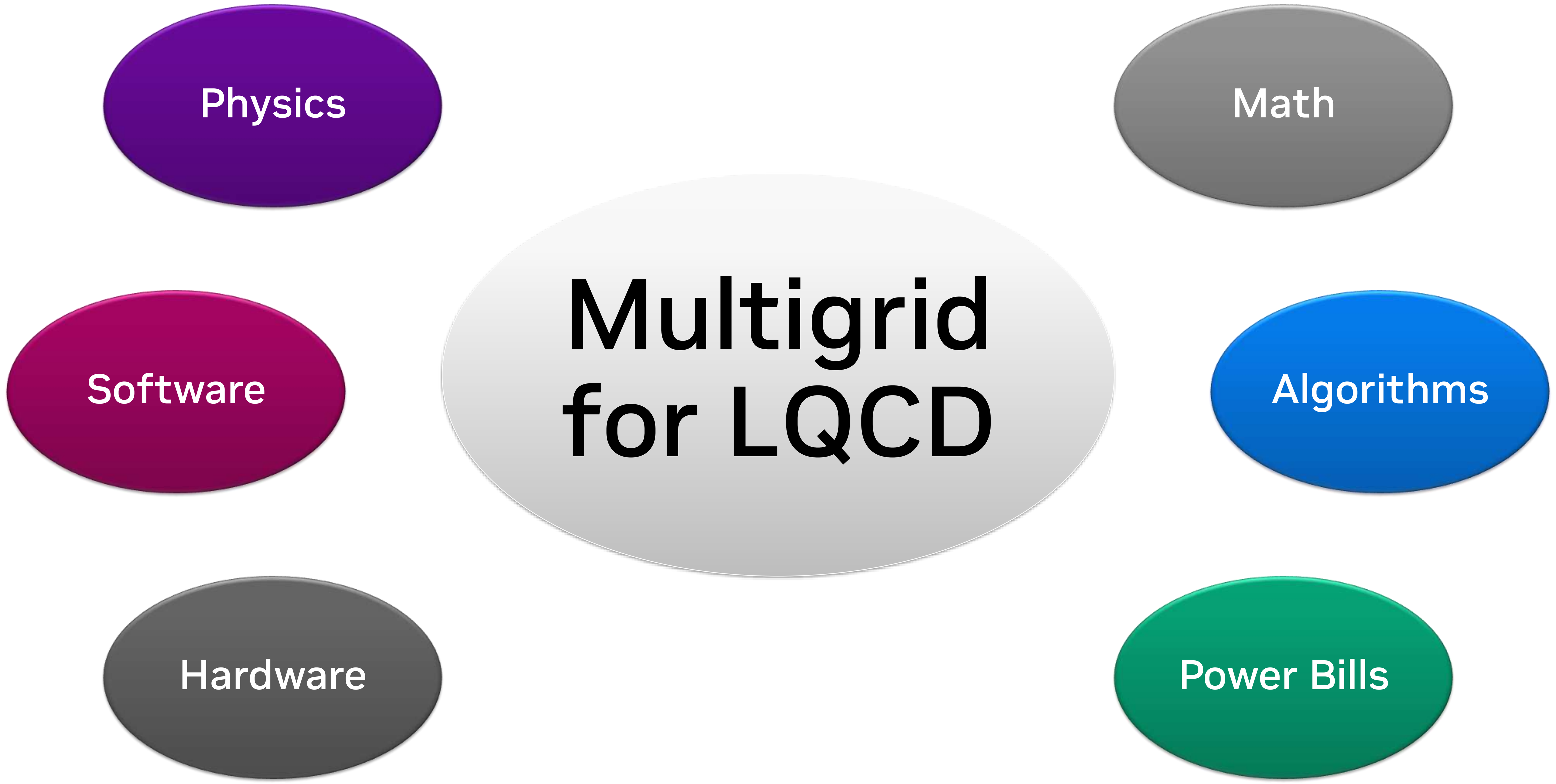
# Why Focus on Multigrid

- LQCD is more than just solving the Dirac linear system over and over again
  - The diversity of topics at this workshop drives this home
- But solving the Dirac matrix linear system is still “the” key workflow
  - The diversity of topics just within linear system solvers at this workshop also drives this home
- And multi-grid solvers are a critical accelerator thereof
  - …The diversity of topics within multi-grid solvers also-also drives this home
- Multi-grid solvers in LQCD are the ideal place to discuss the ~~future~~ present challenges and opportunities in high performance computing



# Why Focus on Multigrid

- LQCD is more than just solving the Dirac linear system over and over again
  - The diversity of topics at this workshop drives this home
- But solving the Dirac matrix linear system is still "the" key workflow
  - The diversity of topics just within linear system solvers at this workshop also drives this home
- And multi-grid solvers are a critical accelerator thereof
  - …The diversity of topics within multi-grid solvers also-also drives this home
- Multi-grid solvers in LQCD are the ideal place to discuss the ~~future~~ present challenges and opportunities in high performance computing
- MG will be the vehicle of my talk, but I want to prompt thinking and discussions outside this one class of algorithms



# Why Multigrid in Lattice QCD?

- As we take the continuum limit at constant physics, the cost of solving the Dirac linear system increases super-linearly in the lattice spacing $a$.
  - This is *critical slowing down*

# Why Multigrid in Lattice QCD?

- As we take the continuum limit at constant physics, the cost of solving the Dirac linear system increases super-linearly in the lattice spacing $a$.
  - This is *critical slowing down*
- Methods such as deflation mitigate this issue, but…
  - …deflation has quadratic scaling
  - *…memory/storage* is a killer as each eigenvector takes O(V) space
  - …the required number of eigenvectors for constant “benefit” also scales with the volume

# Why Multigrid in Lattice QCD?

- As we take the continuum limit at constant physics, the cost of solving the Dirac linear system increases super-linearly in the lattice spacing $a$.
  - This is *critical slowing down*
- Methods such as deflation mitigate this issue, but…
  - …deflation has quadratic scaling
  - *…memory/storage* is a killer as each eigenvector takes O(V) space
  - …the required number of eigenvectors for constant “benefit” also scales with the volume
- Multi-grid methods are a class of algorithms that also mitigate or eliminate critical slowing down
  - …ideally with the naïve cost scaling: O(V)
  - …and *if* you want to store the setup state, only O(V) storage

# What Operator Should We Look At?

Alternative title: we can’t talk about everything

- We could take this discussion in a few directions…

# What Operator Should We Look At?

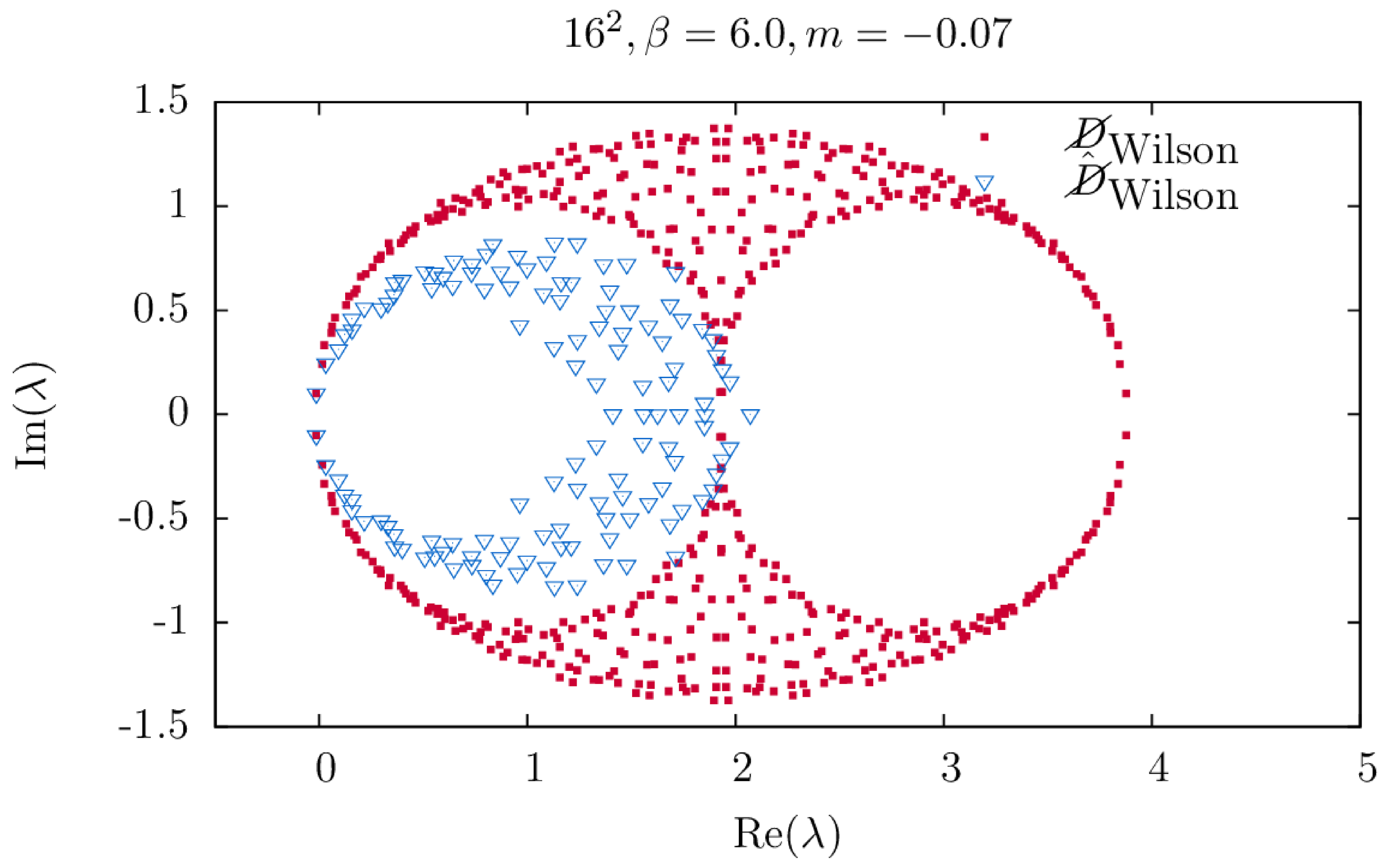
Alternative title: we can't talk about everything

- We could take this discussion in a few directions…
- MG on the normal, Hermitian positive-definite (HPD) operator
  - Strong theoretical justifications, but has its own conditioning and engineering challenges
- Instead, we're going to discuss the "direct" operators (with all of their spectral challenges)

# What Operator Should We Look At?

Alternative title: we can't talk about everything

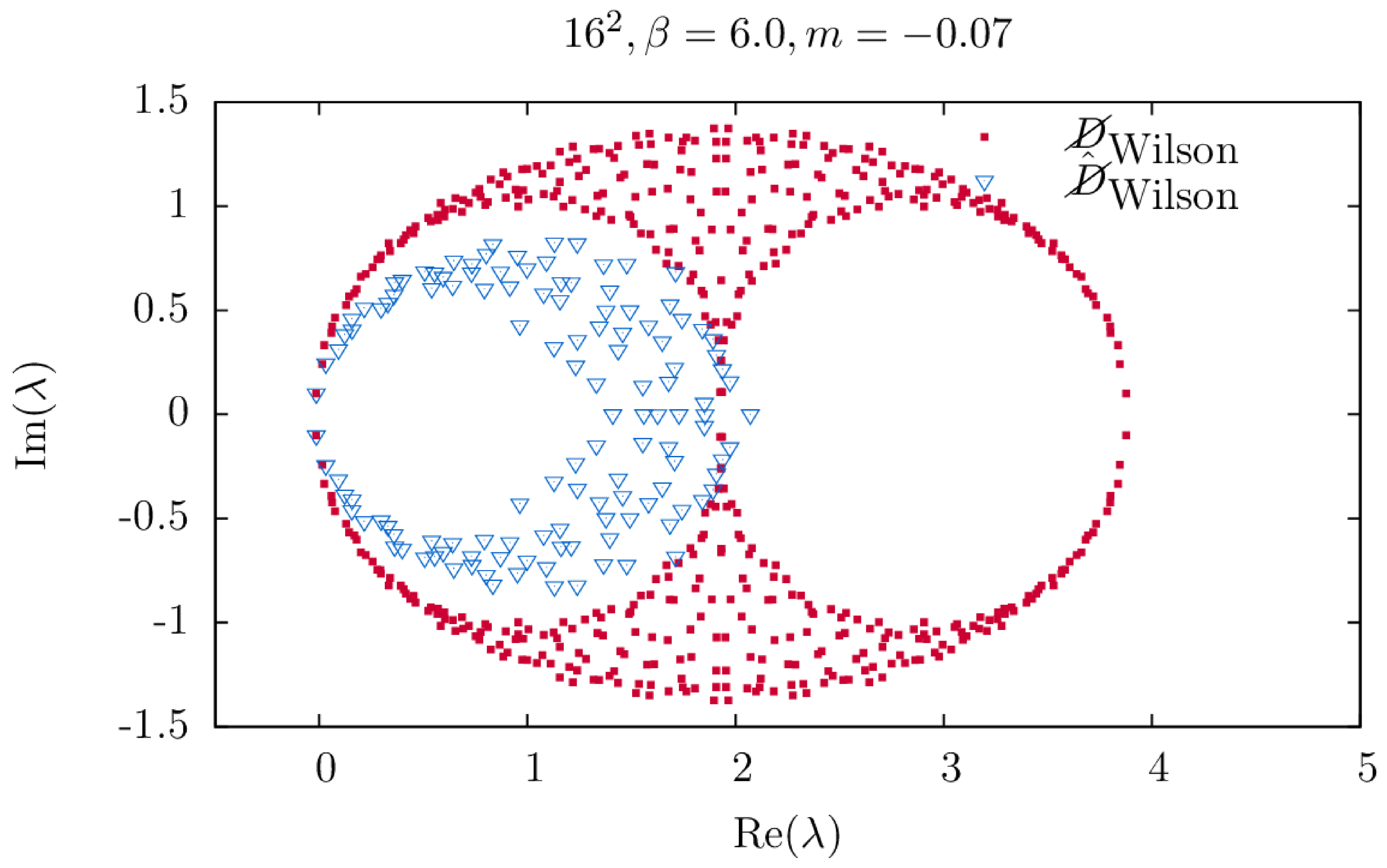
- We could take this discussion in a few directions…
- MG on the normal, Hermitian positive-definite (HPD) operator
  - Strong theoretical justifications, but has its own conditioning and engineering challenges
- Instead, we're going to discuss the "direct" operators (with all of their spectral challenges)

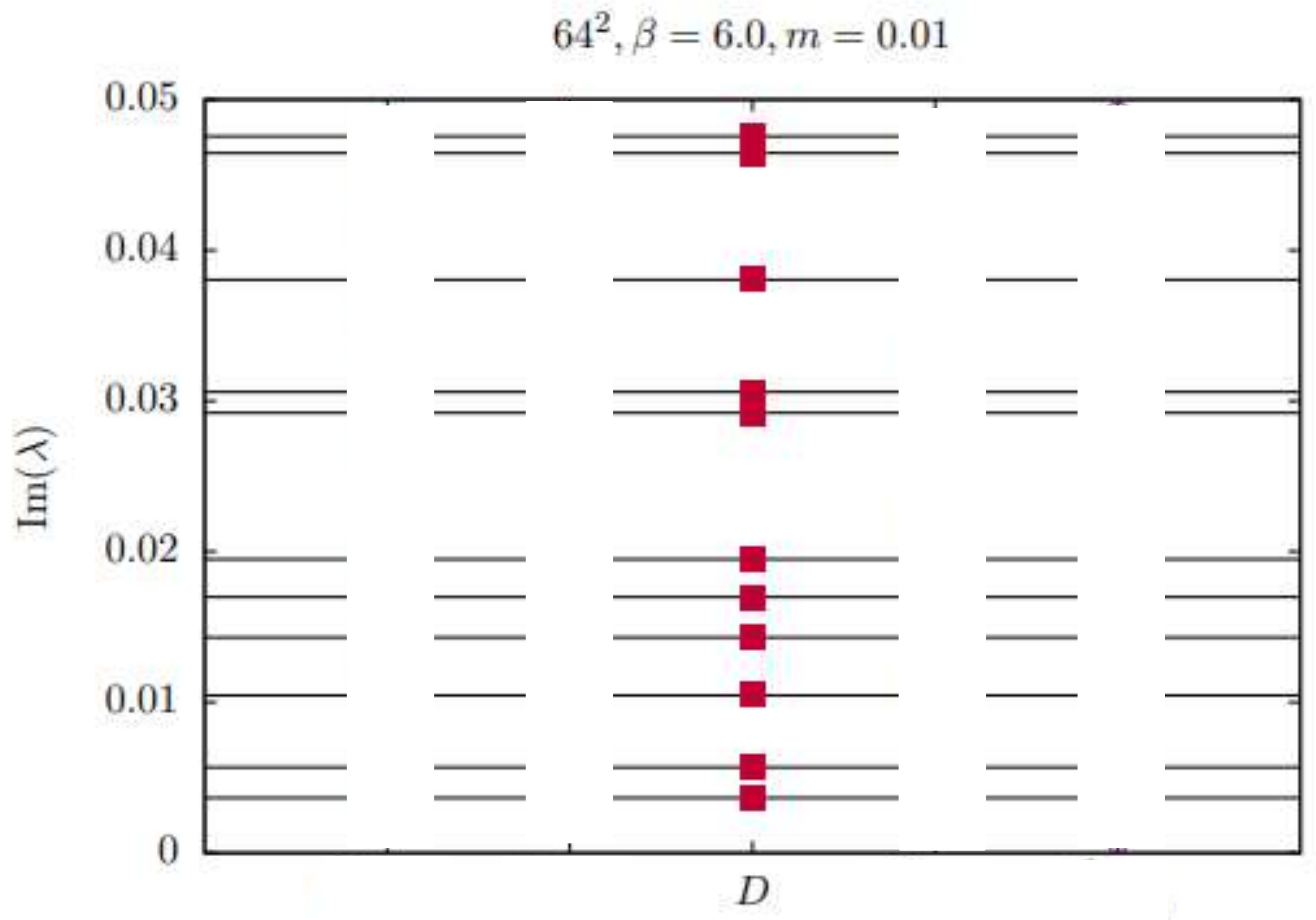


# What Operator Should We Look At?

Alternative title: we can't talk about everything

- We could take this discussion in a few directions…
- MG on the normal, Hermitian positive-definite (HPD) operator
  - Strong theoretical justifications, but has its own conditioning and engineering challenges
- Instead, we're going to discuss the "direct" operators (with all of their spectral challenges)

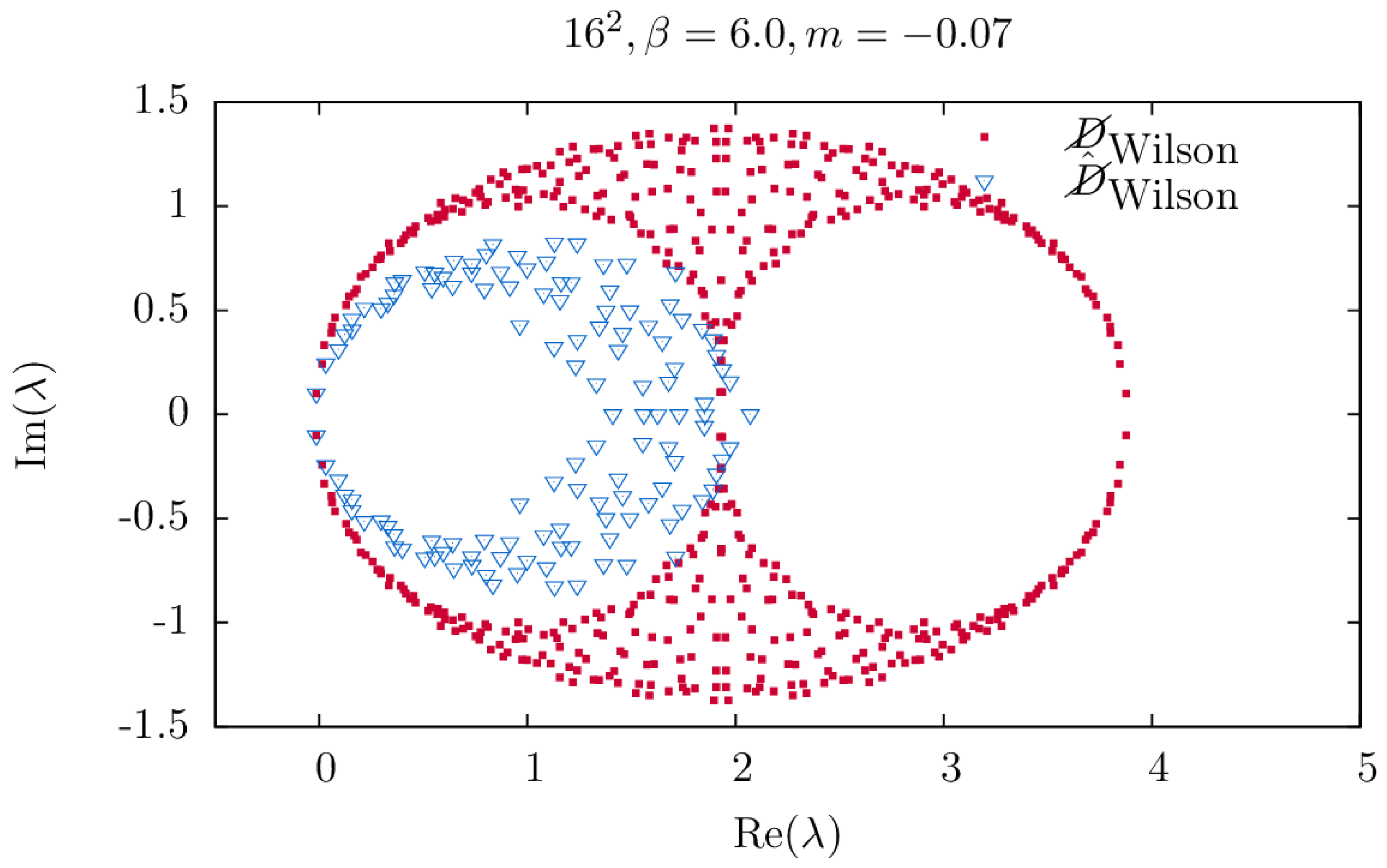


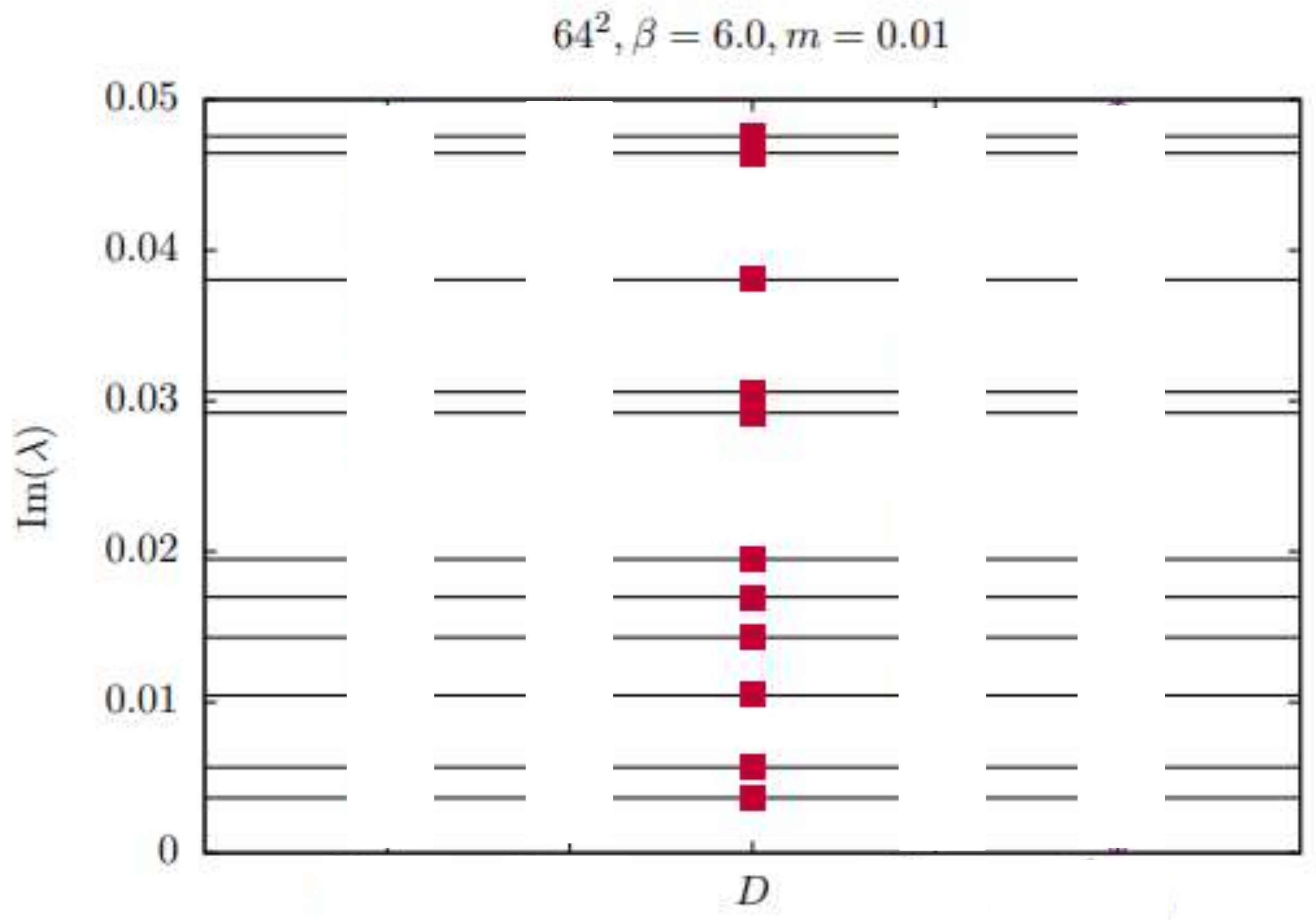



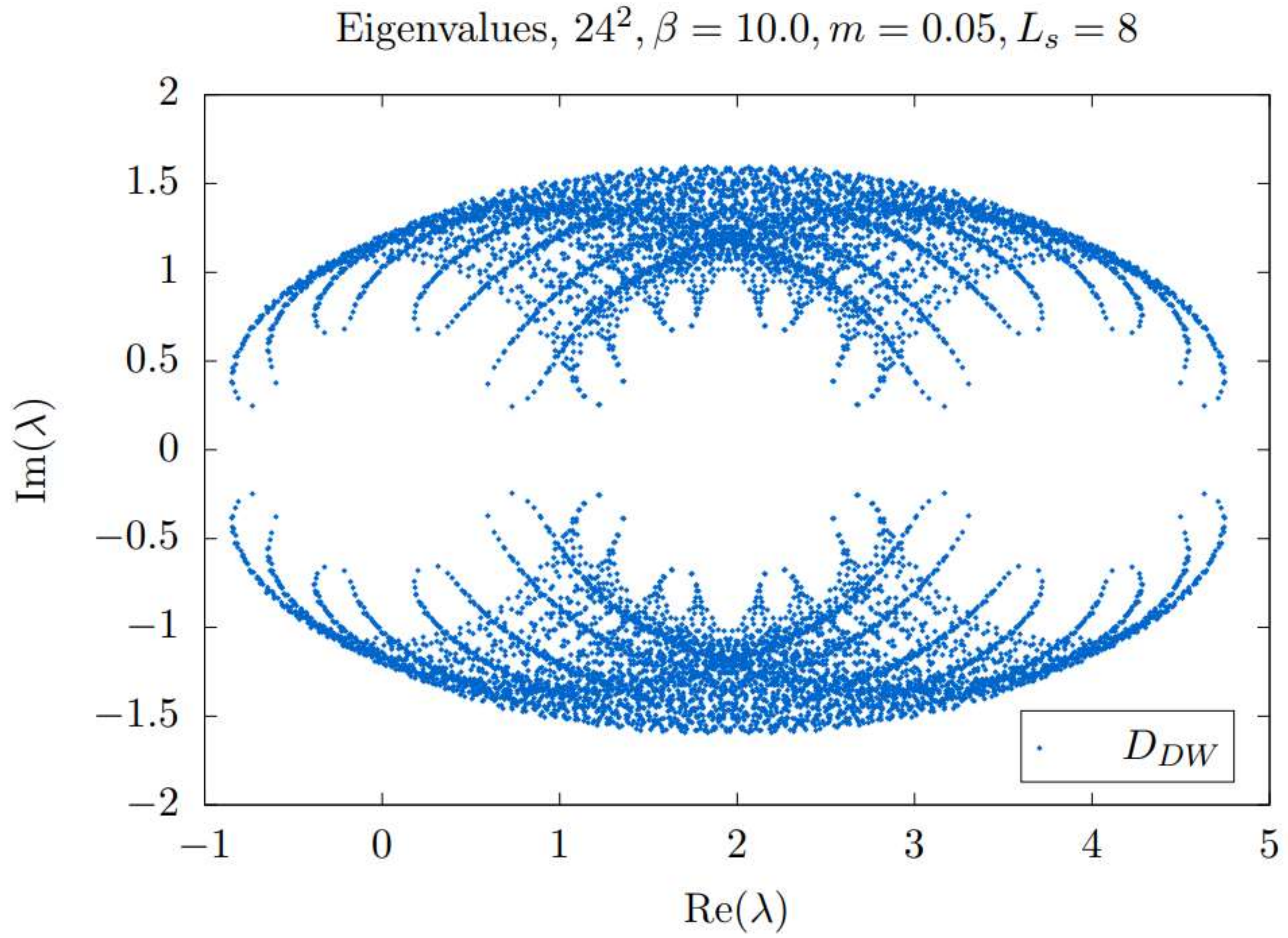
# What Operator Should We Look At?

Alternative title: we can’t talk about everything

- We could take this discussion in a few directions…
- MG on the normal, Hermitian positive-definite (HPD) operator
  - Strong theoretical justifications, but has its own conditioning and engineering challenges
- Instead, we’re going to discuss the “direct” operators (with all of their spectral challenges)



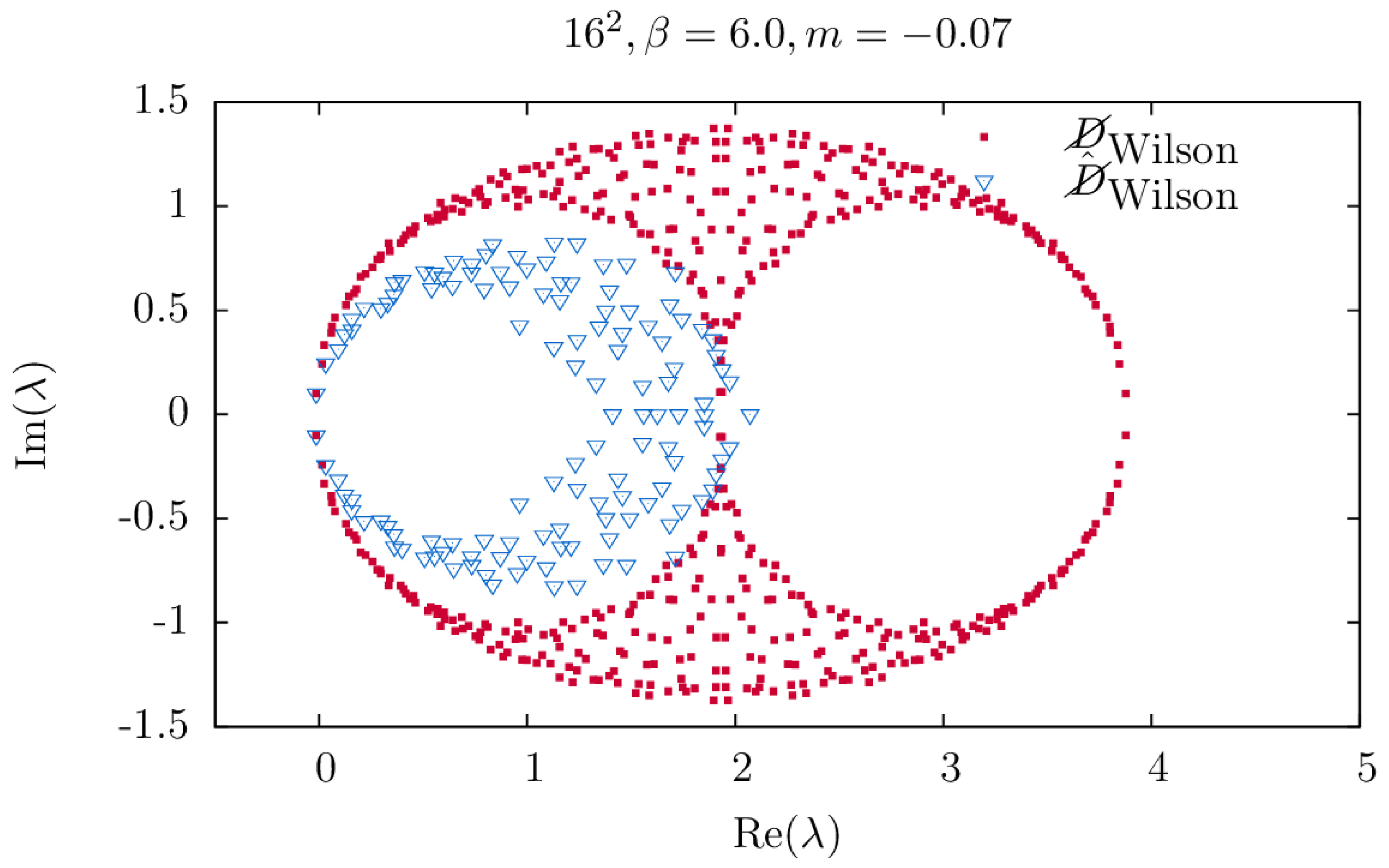



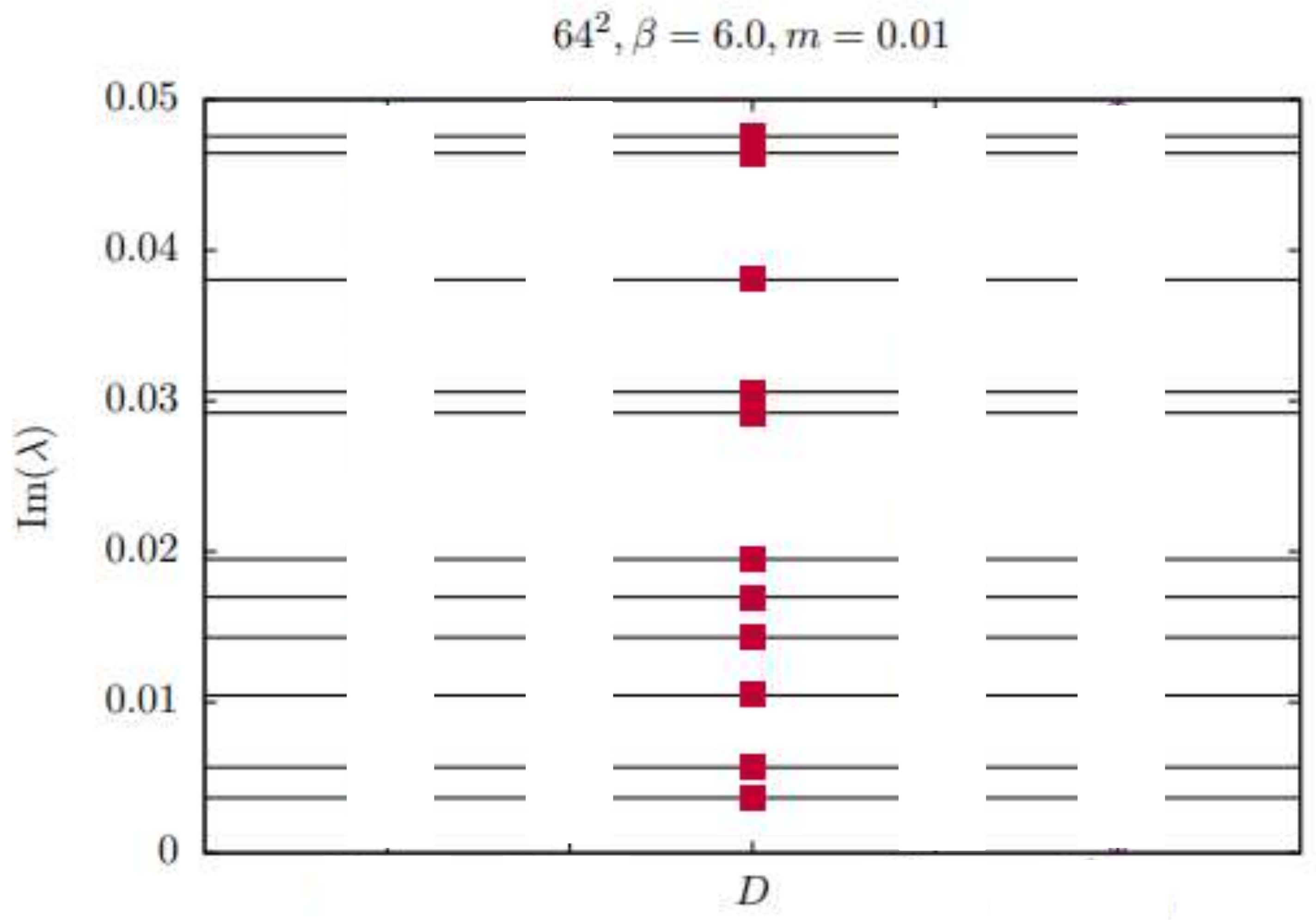



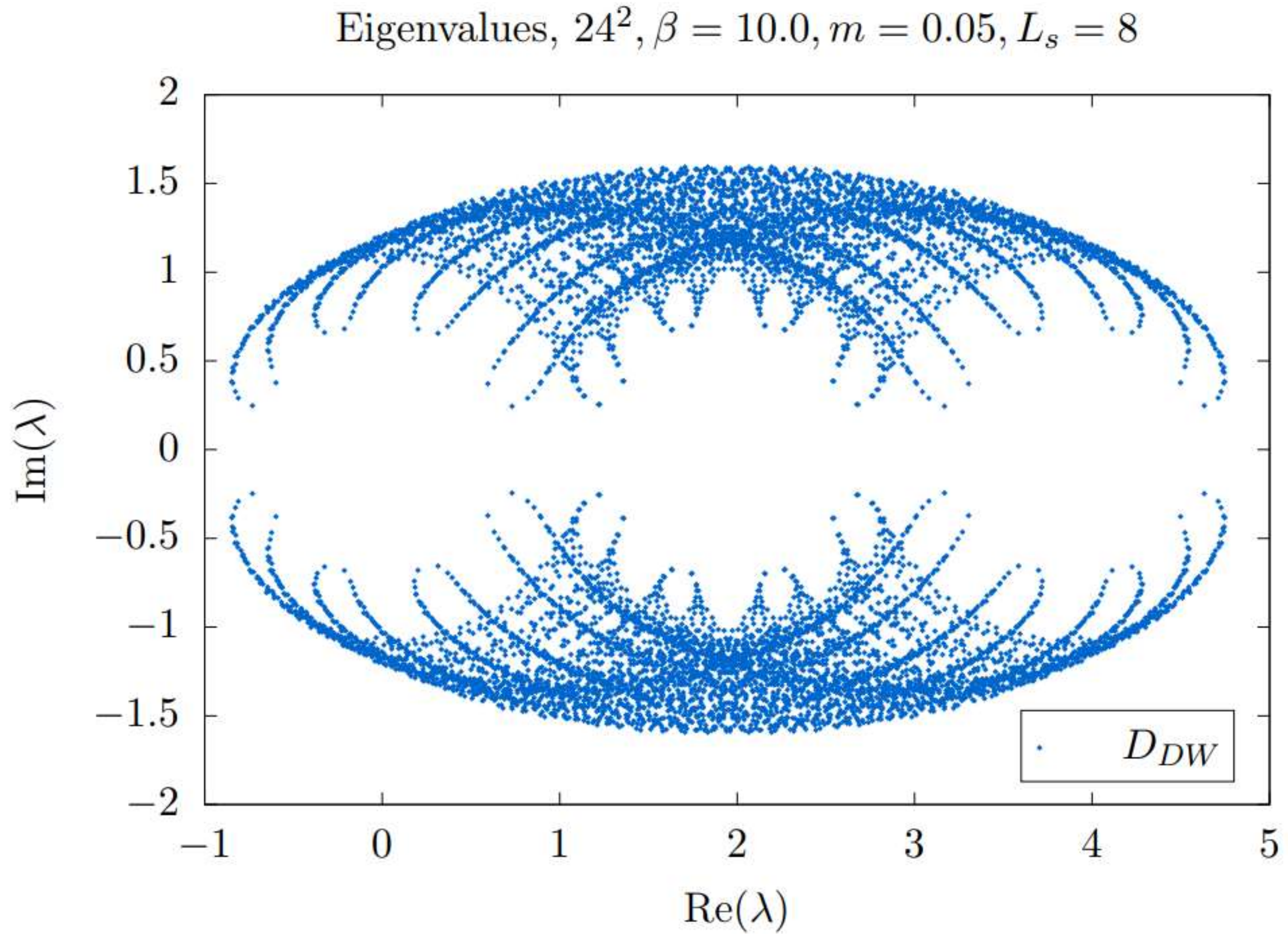
# What Operator Should We Look At?

Alternative title: we can’t talk about everything

- We could take this discussion in a few directions…
- MG on the normal, Hermitian positive-definite (HPD) operator
  - Strong theoretical justifications, but has its own conditioning and engineering challenges
- Instead, we’re going to discuss the “direct” operators (with all of their spectral challenges)




- Much of what I’m going to discuss is still agnostic of this choice

# Multiple Grids

## A Discretization-Agnostic Discussion

- Goal: successfully *and efficiently* capture the challenging (“low”, “near-null”) modes of our linear operator $D$ and deal with them in a reduced subspace

# Multiple Grids

## A Discretization-Agnostic Discussion

- Goal: successfully *and efficiently* capture the challenging (“low”, “near-null”) modes of our linear operator $D$ and deal with them in a reduced subspace
- We need three things, and we need to do it well
  - A map from the fine space to the coarser space: $R$ for restrictor
  - A map from the coarse space to the finer space: $P$ for prolongator
  - An operator that acts on the coarsened space: $\widehat{D}$, where the “hat” corresponds to the “coarsened” operator

# Multiple Grids

## A Discretization-Agnostic Discussion

- Goal: successfully *and efficiently* capture the challenging ("low", "near-null") modes of our linear operator $D$ and deal with them in a reduced subspace
- We need three things, and we need to do it well
  - A map from the fine space to the coarser space: $R$ for restrictor
  - A map from the coarse space to the finer space: $P$ for prolongator
  - An operator that acts on the coarsened space: $\widehat{D}$, where the "hat" corresponds to the "coarsened" operator
- Focus: adaptive geometric multi-grid methods

# Adaptive Geometric Multigrid

## The Near-Null Space

- “Let the operator speak”

# Adaptive Geometric Multigrid

## The Near-Null Space

- “Let the operator speak”
- Adaptively find candidate null-space vectors
  - Dynamically learn the null space and use this to define the prolongator
  - Algorithm is self learning

# Adaptive Geometric Multigrid

## The Near-Null Space

- "Let the operator speak"
- Adaptively find candidate null-space vectors
  - Dynamically learn the null space and use this to define the prolongator
  - Algorithm is self learning
- There are many approaches
  - Inverse iterations
    - "Solve" $D\overrightarrow{v_k} = \vec{0}$ with random initial guess $\overrightarrow{v_{k,0}}$
    - The exact answer is zero but Krylov solvers don't know that---after ?? iterations $\overrightarrow{v_k}$ should be rich in low modes
  - Chebyshev Filters (P. Boyle)
  - Low Eigenvectors

# Adaptive Geometric Multigrid

## The Setup

- Block-orthonormalize the near-null vectors to form the prolongator
  - $(1 - P\ R)\overrightarrow{v_k} = \vec{0}$
  - Typically use $O(4^4)$ geometric blocks

# Adaptive Geometric Multigrid

## The Setup

- Block-orthonormalize the near-null vectors to form the prolongator
  - $(1 - P\ R)\overrightarrow{v_k} = \vec{0}$
  - Typically use $O(4^4)$ geometric blocks
- LQCD-specific: Preserve "chirality" when coarsening
  - Wilson-type: $R = \gamma_5 P^\dagger \gamma_5 = P^\dagger$ -- preserve instanton modes
  - Staggered-type: $R = \varepsilon(x) P^\dagger \varepsilon(x) = P^\dagger$
    - Note: this is $\gamma_5 \otimes \tau_5$, not $\gamma_5 \otimes 1$...
  - Domain-wall-type: more complicated; the *general* Mobius $\Gamma_5$ is non-local

# Adaptive Geometric Multigrid

## The Setup

- Block-orthonormalize the near-null vectors to form the prolongator
  - $(1 - P\ R)\overrightarrow{v_k} = \vec{0}$
  - Typically use O($4^4$) geometric blocks
- LQCD-specific: Preserve “chirality” when coarsening
  - Wilson-type: $R = \gamma_5 P^\dagger \gamma_5 = P^\dagger$ -- preserve instanton modes
  - Staggered-type: $R = \varepsilon(x) P^\dagger \varepsilon(x) = P^\dagger$
    - Note: this is $\gamma_5 \otimes \tau_5$, not $\gamma_5 \otimes 1$...
  - Domain-wall-type: more complicated; the *general* Mobius $\Gamma_5$ is non-local
- Form the coarse operator via a Galerkin projection
  - $\widehat{D} = P^\dagger D\ P$

# Adaptive Geometric Multigrid

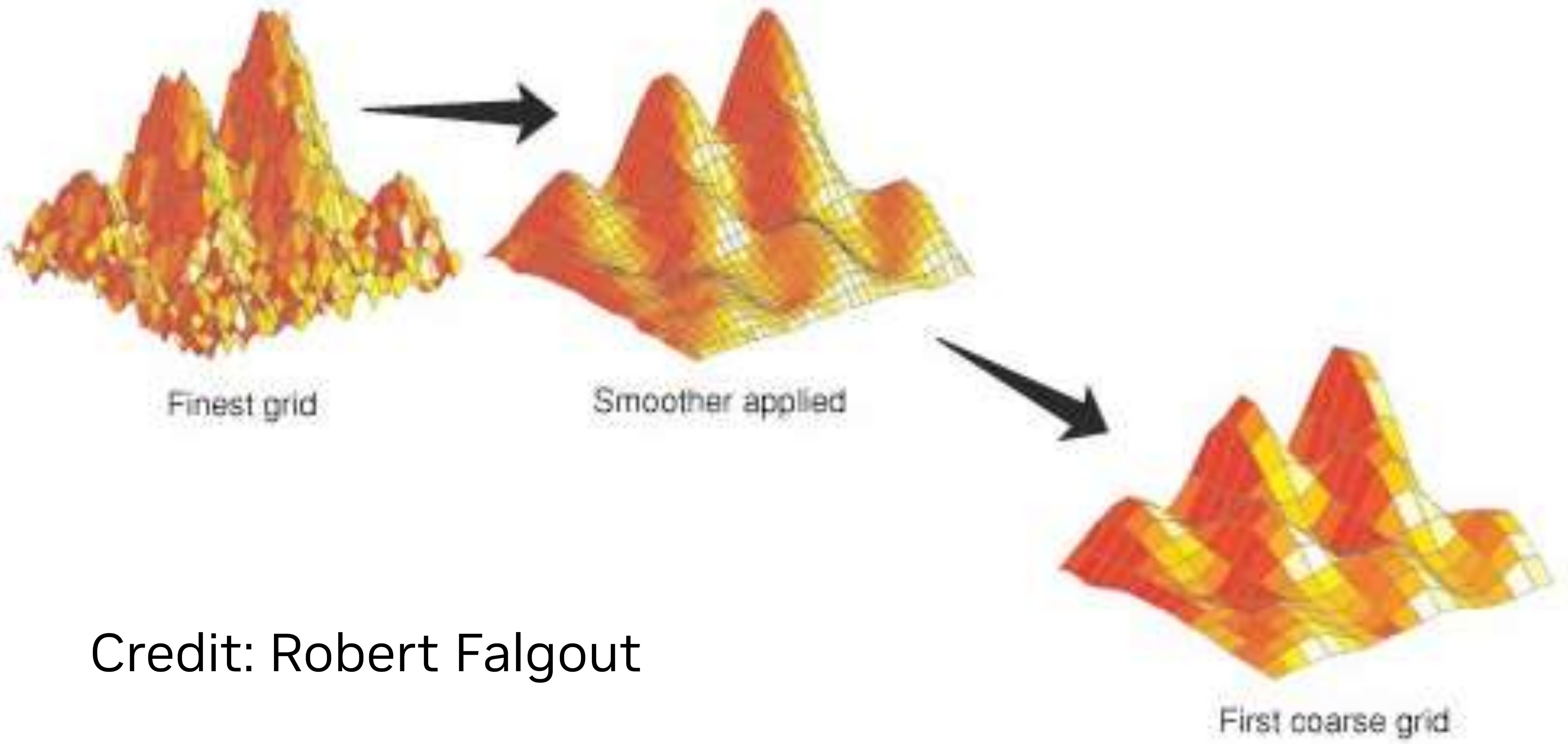
## The Setup

- Block-orthonormalize the near-null vectors to form the prolongator
  - $(1 - P\ R)\overrightarrow{v_k} = \vec{0}$
  - Typically use O($4^4$) geometric blocks
- LQCD-specific: Preserve "chirality" when coarsening
  - Wilson-type: $R = \gamma_5 P^\dagger \gamma_5 = P^\dagger$ -- preserve instanton modes
  - Staggered-type: $R = \varepsilon(x) P^\dagger \varepsilon(x) = P^\dagger$
    - Note: this is $\gamma_5 \otimes \tau_5$, not $\gamma_5 \otimes 1$...
  - Domain-wall-type: more complicated; the *general* Mobius $\Gamma_5$ is non-local
- Form the coarse operator via a Galerkin projection
  - $\hat{D} = P^\dagger D\ P$
- Recurse on coarse problem

# Adaptive Geometric Multigrid

## The Solver

- Perform an MG-preconditioned iterative Krylov solve (via GCR, FGMRES…); on a given iteration:
  - $r$ is the current iterated residual; $x$ is the current iterated solution
  - (Optional) pre-smoother: relax on the current residual with $D$
  - Restrict the smoothed residual: $\hat{r} = P^\dagger r$
  - Approximately solve the coarse system to get a coarse error correction: $\widehat{D}\hat{e} = \hat{r}$
  - Prolong the error: $e = P\,\hat{e}$
  - Correct the solution: $x \Leftarrow x + e$
  - (Optional) post-smooth on the accumulated solution with $D$



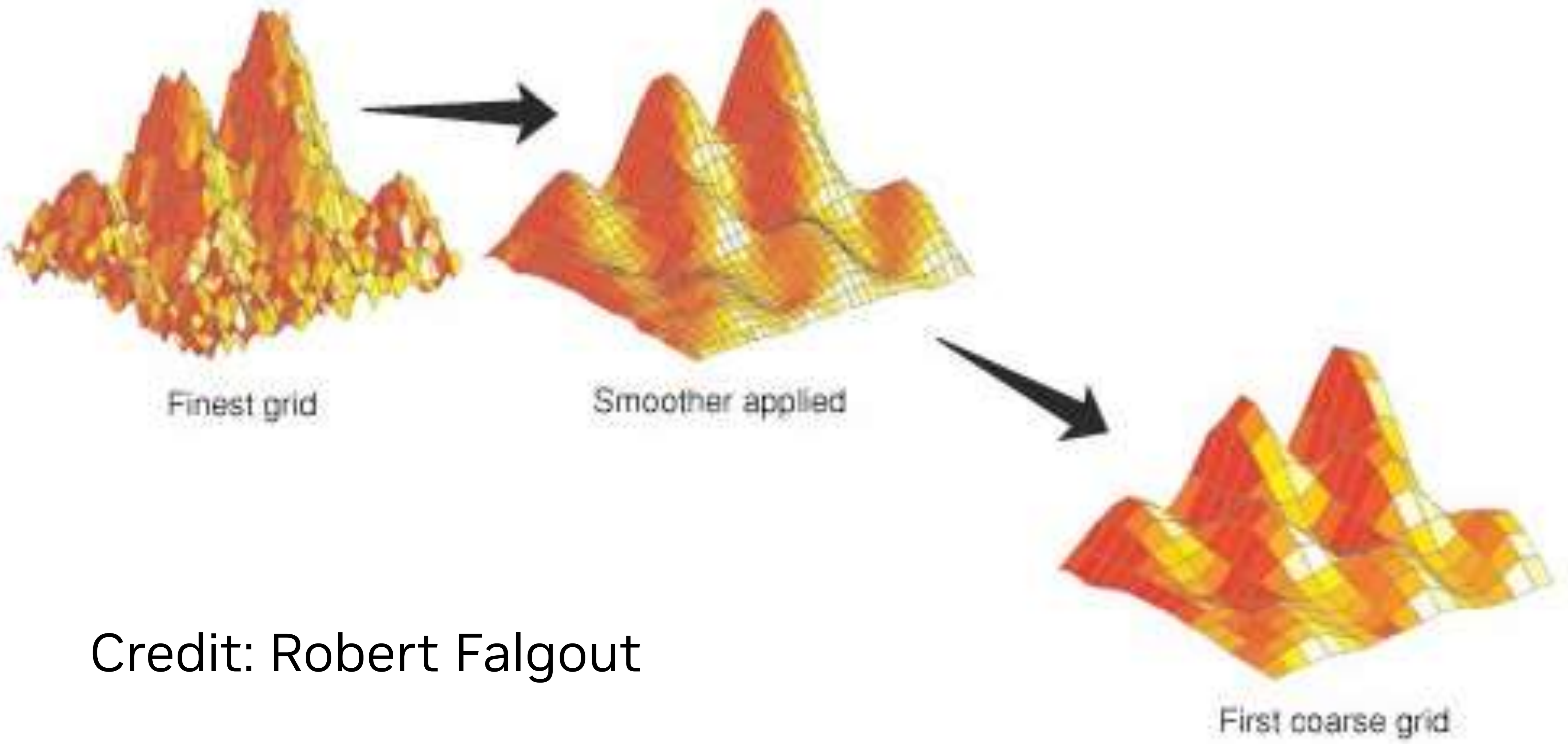

Credit: Robert Falgout

# Adaptive Geometric Multigrid

## The Solver

- Perform an MG-preconditioned iterative Krylov solve (via GCR, FGMRES…); on a given iteration:
  - $r$ is the current iterated residual; $x$ is the current iterated solution
  - (Optional) pre-smoother: relax on the current residual with $D$
  - Restrict the smoothed residual: $\hat{r} = P^{\dagger}\, r$
  - Approximately solve the coarse system to get a coarse error correction: $\widehat{D}\hat{e} = \hat{r}$
  - Prolong the error: $e = P\,\hat{e}$
  - Correct the solution: $x \Leftarrow x + e$
  - (Optional) post-smooth on the accumulated solution with $D$

**This can be done recursively**



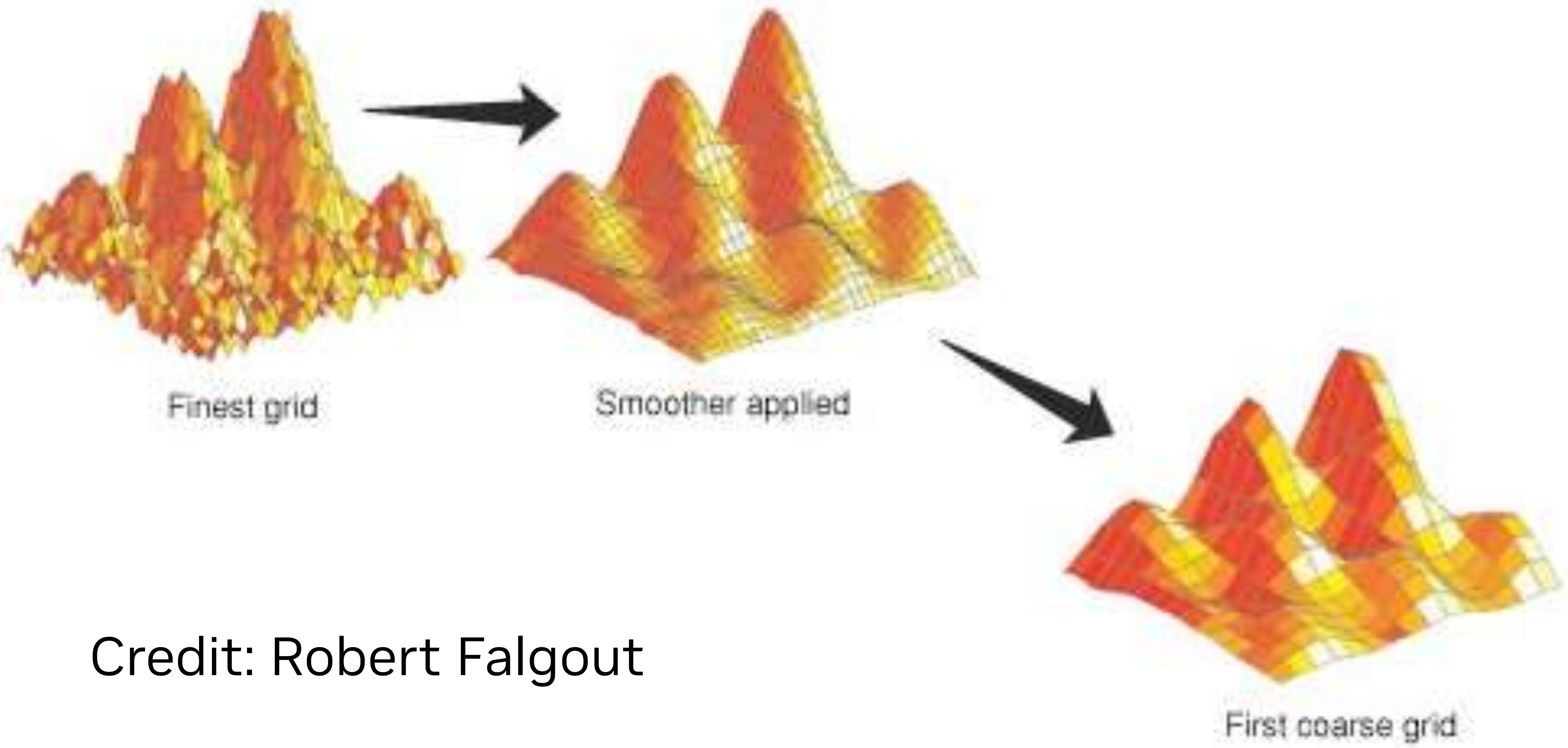

Credit: Robert Falgout

# Adaptive Geometric Multigrid

## The Solver

- Perform an MG-preconditioned iterative Krylov solve (via GCR, FGMRES…); on a given iteration:
  - $r$ is the current iterated residual; $x$ is the current iterated solution
  - (Optional) pre-smoother: relax on the current residual with $D$
  - Restrict the smoothed residual: $\hat{r} = P^{\dagger} r$
  - Approximately solve the coarse system to get a coarse error correction: $\widehat{D}\hat{e} = \hat{r}$
  - Prolong the error: $e = P\,\hat{e}$
  - Correct the solution: $x \Leftarrow x + e$
  - (Optional) post-smooth on the accumulated solution with $D$
- Notably, for the coarsest level:
  - Solve $\widehat{\widehat{D}}\hat{\hat{e}} = \hat{\hat{r}}$ (or more hats) to a fixed tolerance, or
  - Perform an SVD deflation (Howarth) plus a fixed-iteration solve

**This can be done recursively**



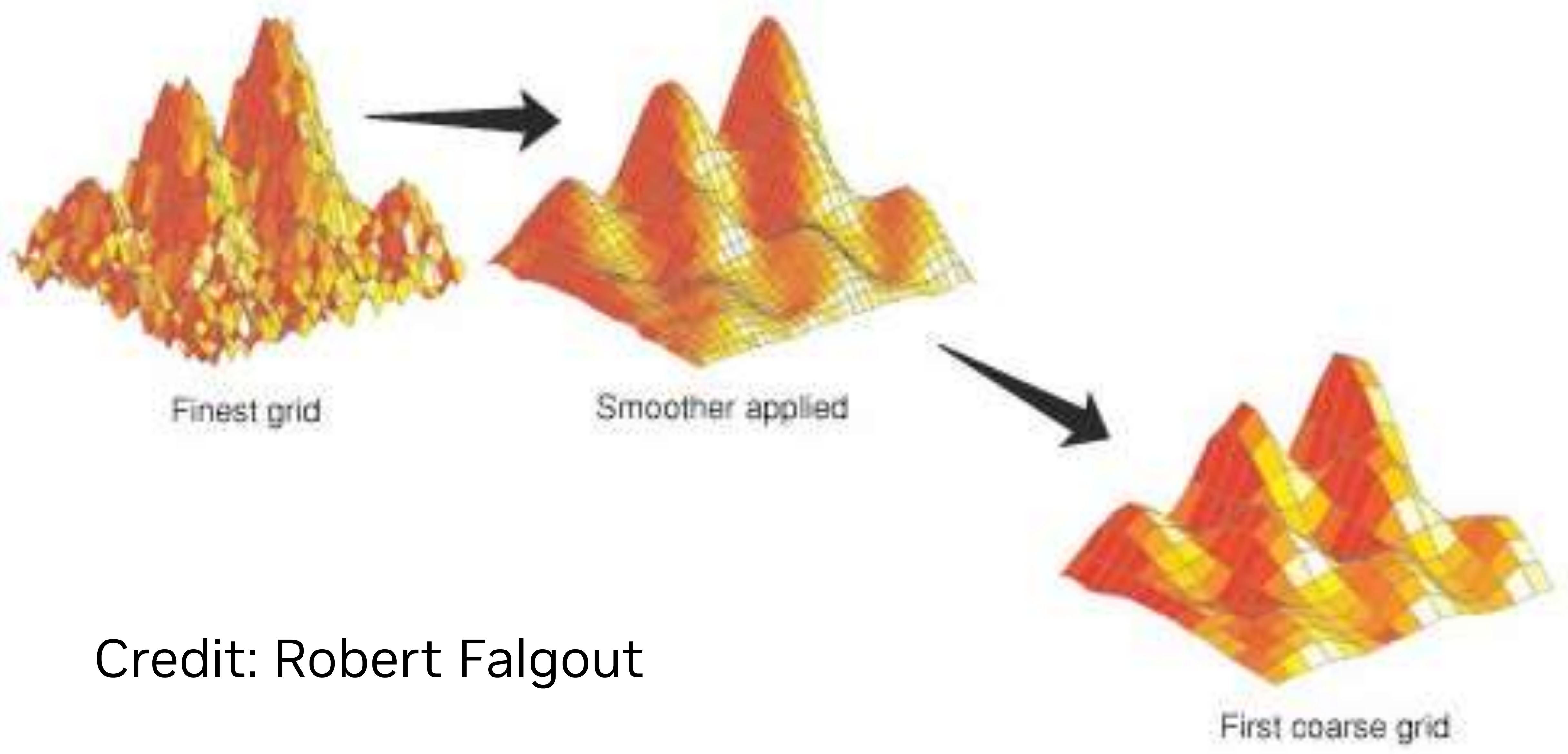

Credit: Robert Falgout

# Adaptive Geometric Multigrid

## The Solver

- Perform an MG-preconditioned iterative Krylov solve (via GCR, FGMRES…); on a given iteration:
  - $r$ is the current iterated residual; $x$ is the current iterated solution
  - (Optional) pre-smoother: relax on the current residual with $D$
  - Restrict the smoothed residual: $\hat{r} = P^{\dagger} r$
  - Approximately solve the coarse system to get a coarse error correction: $\hat{D}\hat{e} = \hat{r}$
  - Prolong the error: $e = P\,\hat{e}$
  - Correct the solution: $x \Leftarrow x + e$
  - (Optional) post-smooth on the accumulated solution with $D$
- Notably, for the coarsest level:
  - Solve $\hat{\hat{D}}\hat{\hat{e}} = \hat{\hat{r}}$ (or more hats) to a fixed tolerance, or
  - Perform an SVD deflation (Howarth) plus a fixed-iteration solve
- If you did everything right, it'll efficiently converge

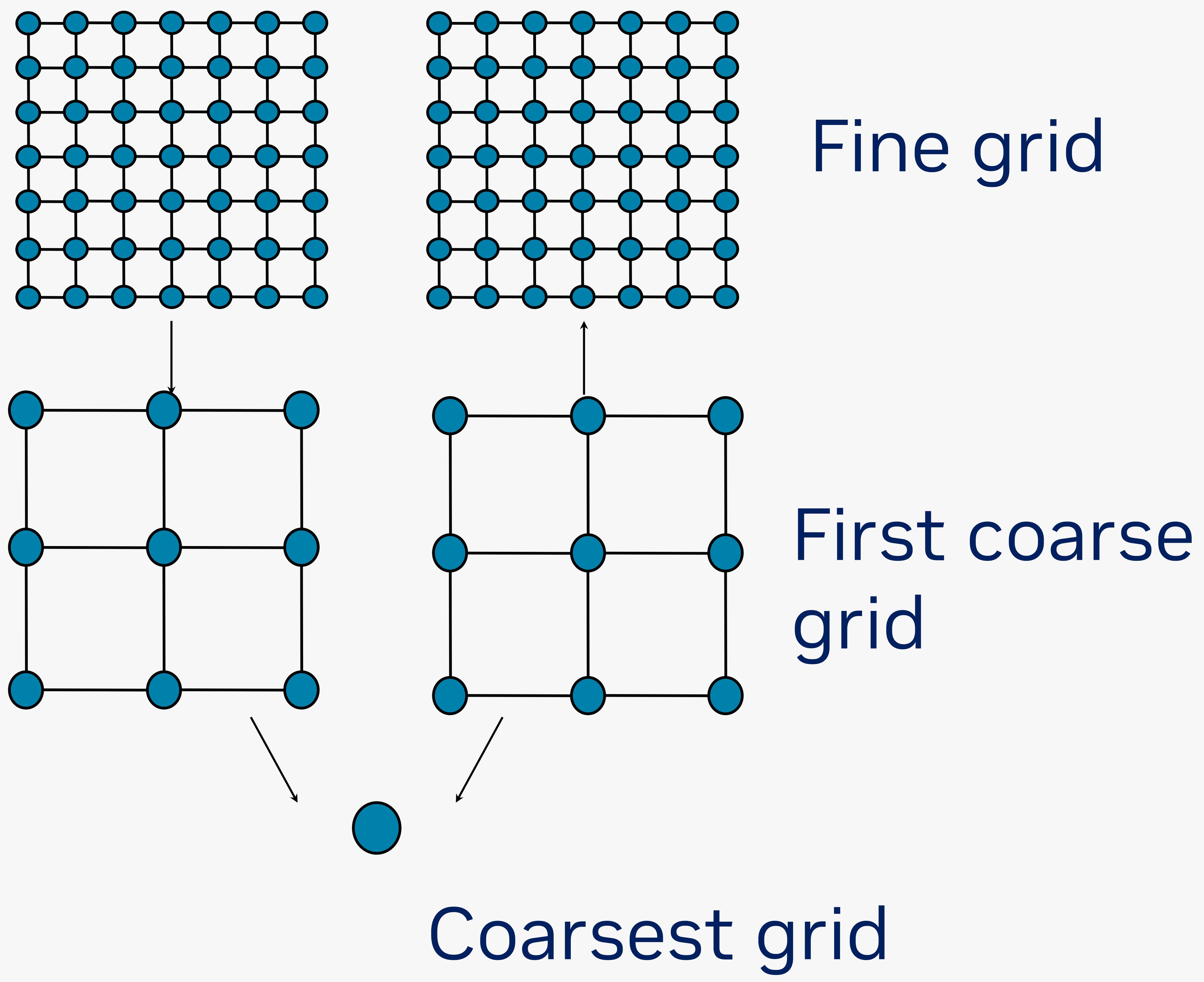
**This can be done recursively**

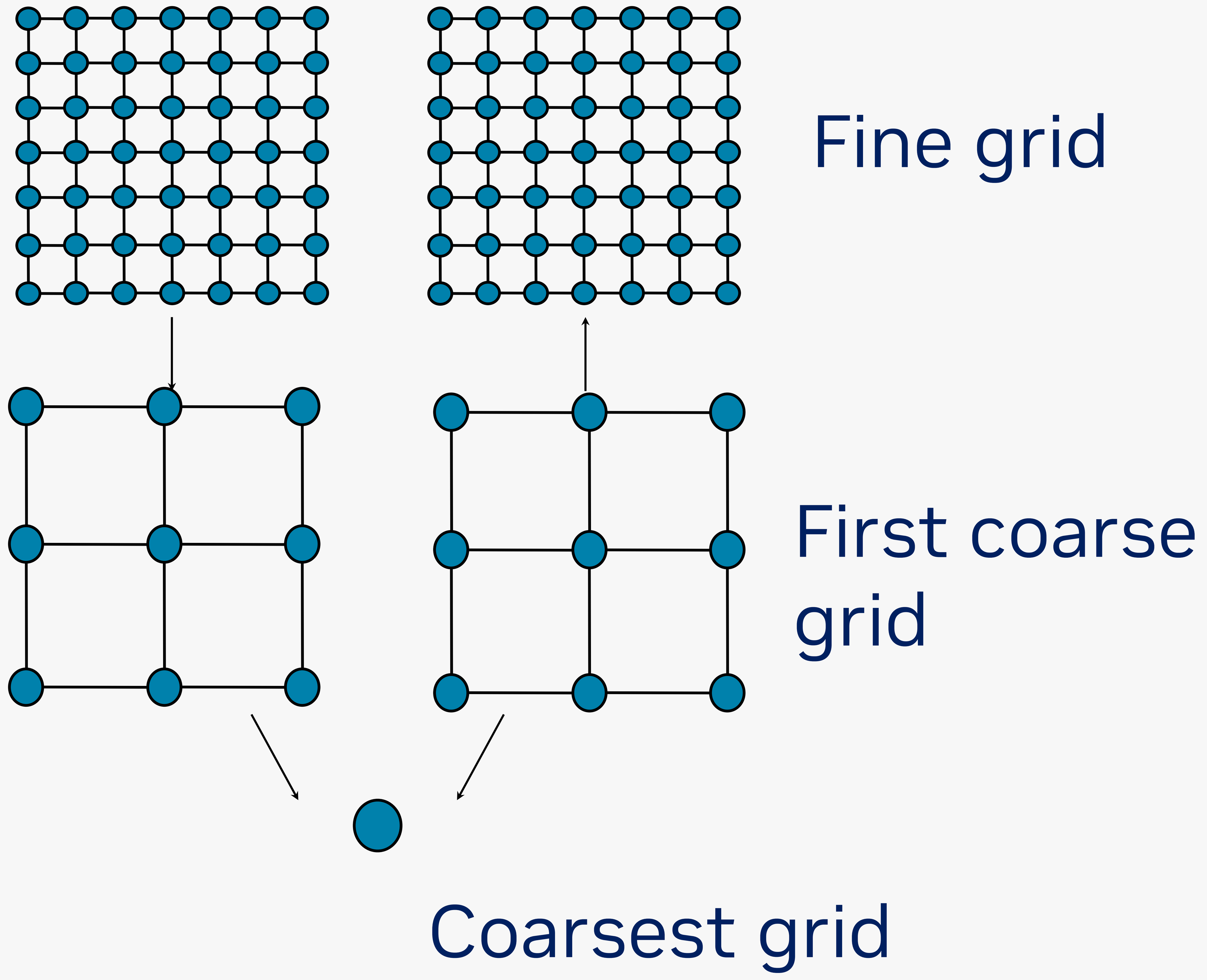



Credit: Robert Falgout

# A Zoo of Questions

Math, Physics, Software, Hardware, Algorithms, Power Bills…

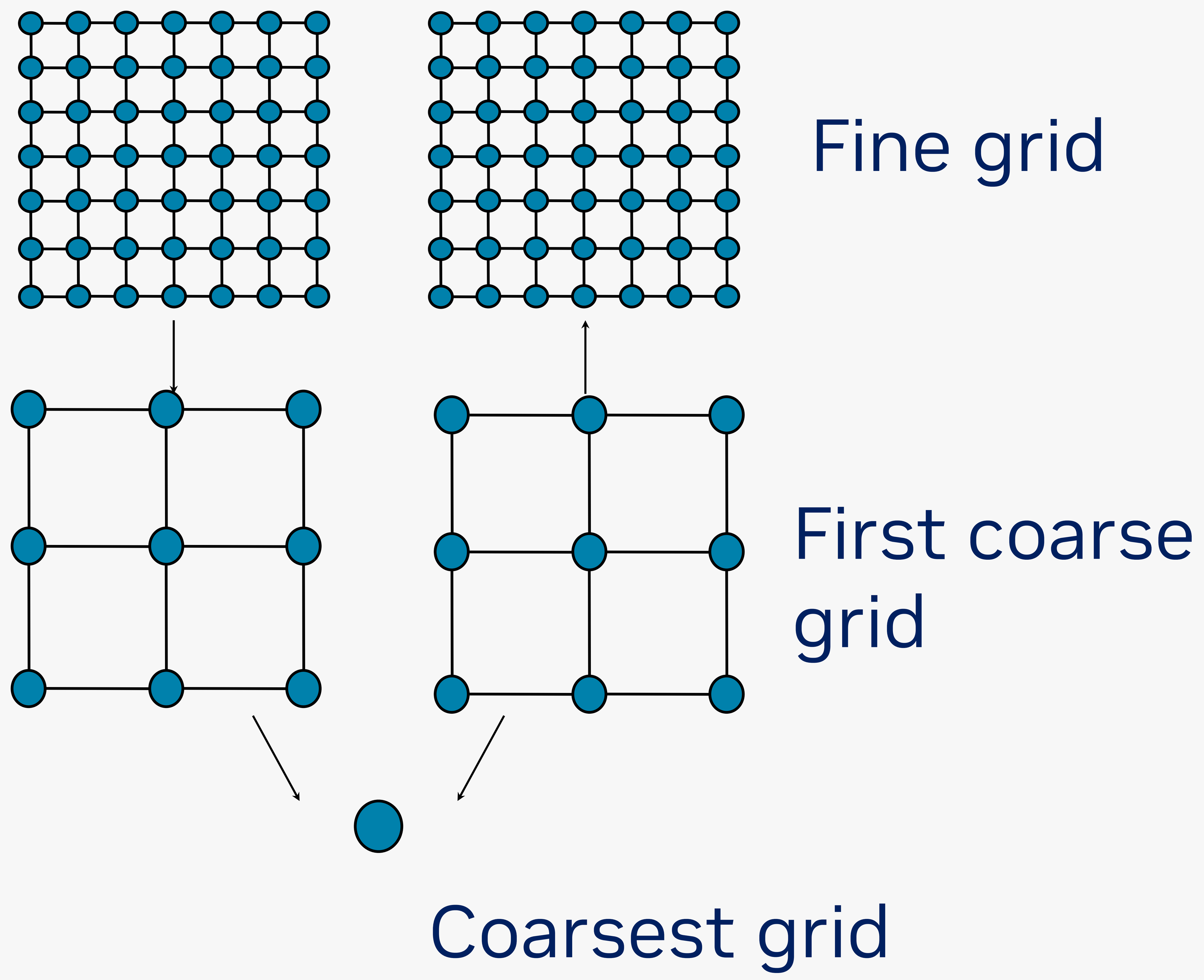





# A Zoo of Questions

Math, Physics, Software, Hardware, Algorithms, Power Bills…

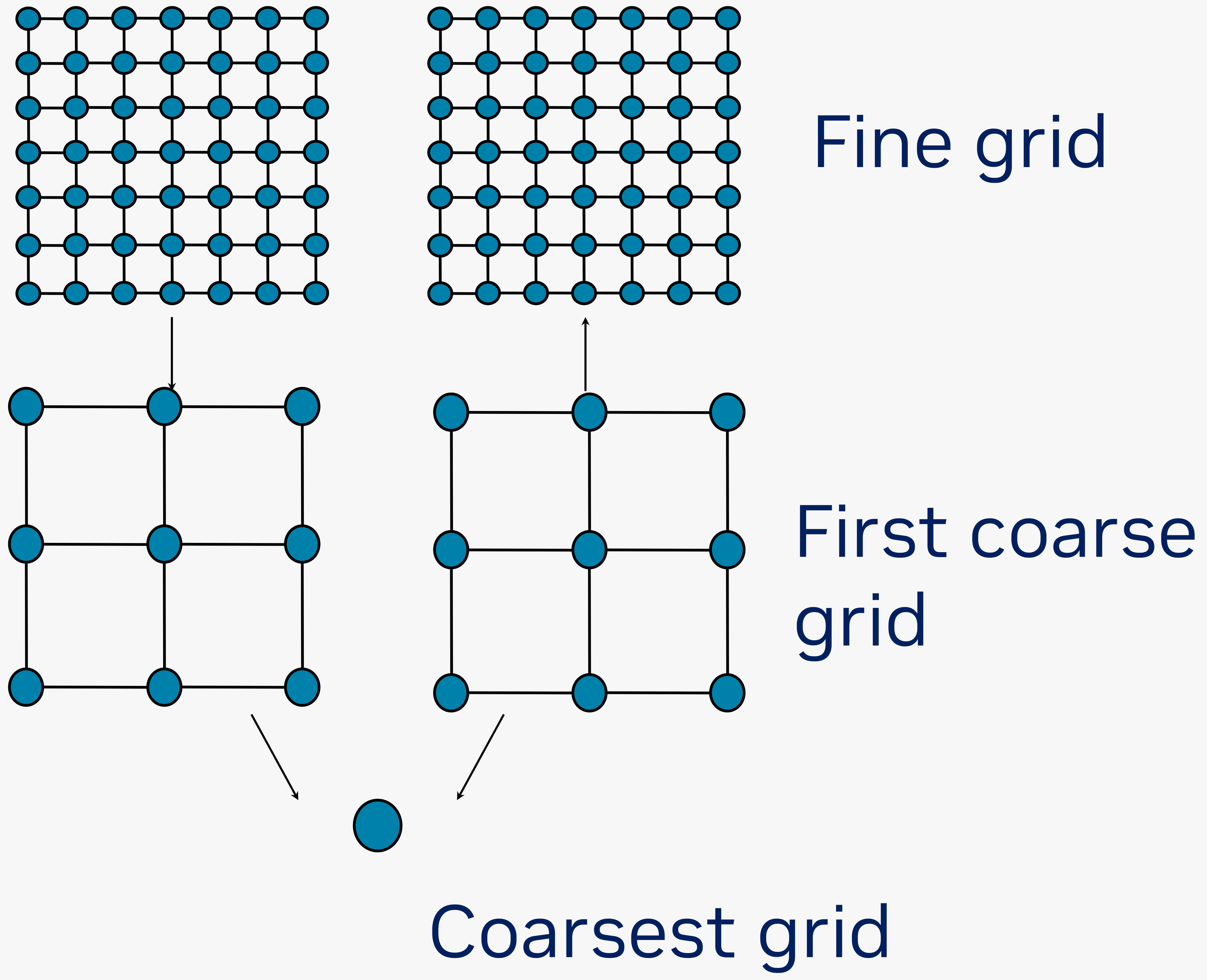
- Is the operator amenable to a Galerkin ($\widehat{D} = P^{\dagger}D\ P$) projection?




# A Zoo of Questions

## Math, Physics, Software, Hardware, Algorithms, Power Bills…

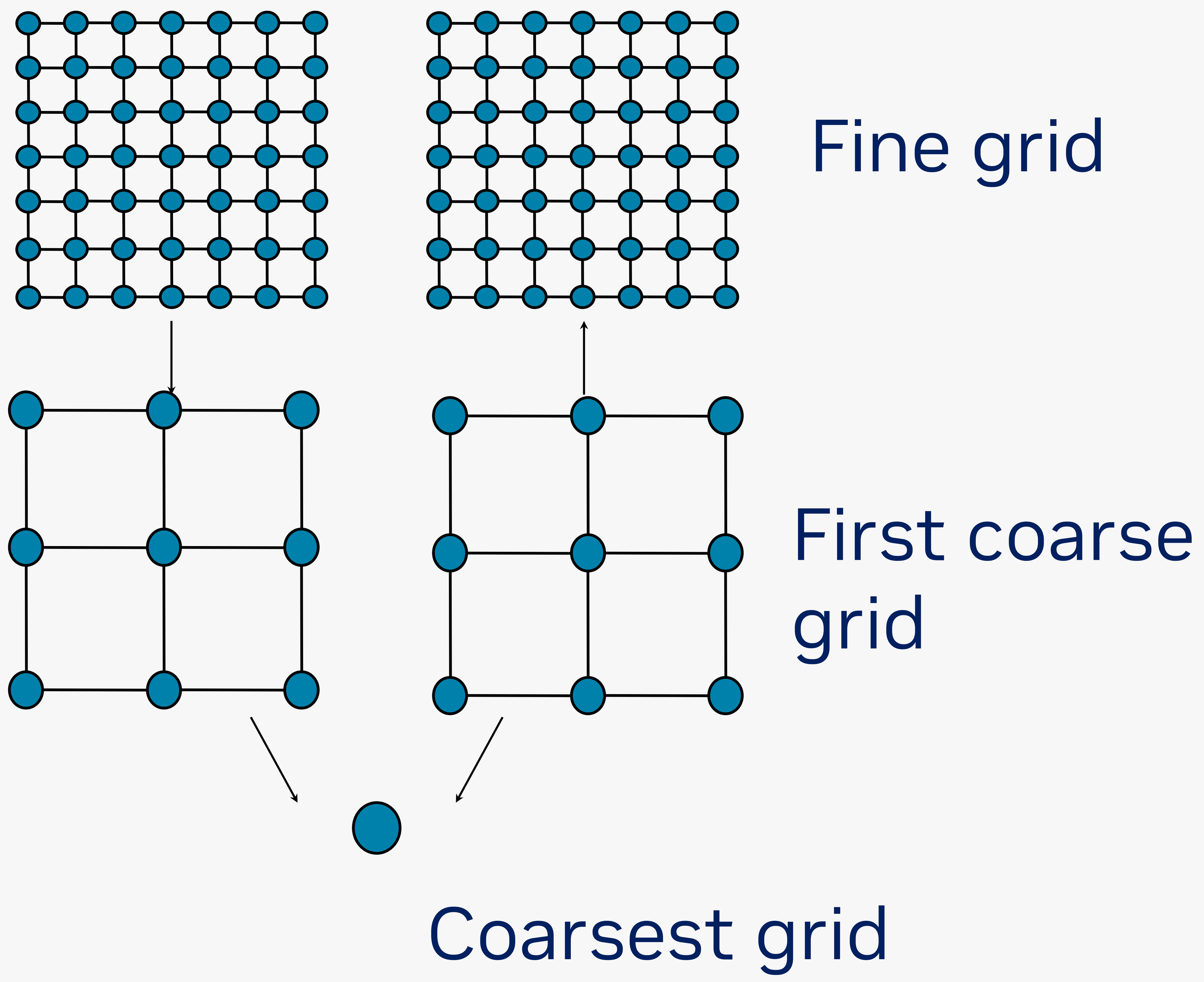
- Is the operator amenable to a Galerkin ($\widehat{D} = P^{\dagger}D\,P$) projection?
- What's the ideal way to generate near-null vectors?




# A Zoo of Questions

Math, Physics, Software, Hardware, Algorithms, Power Bills…

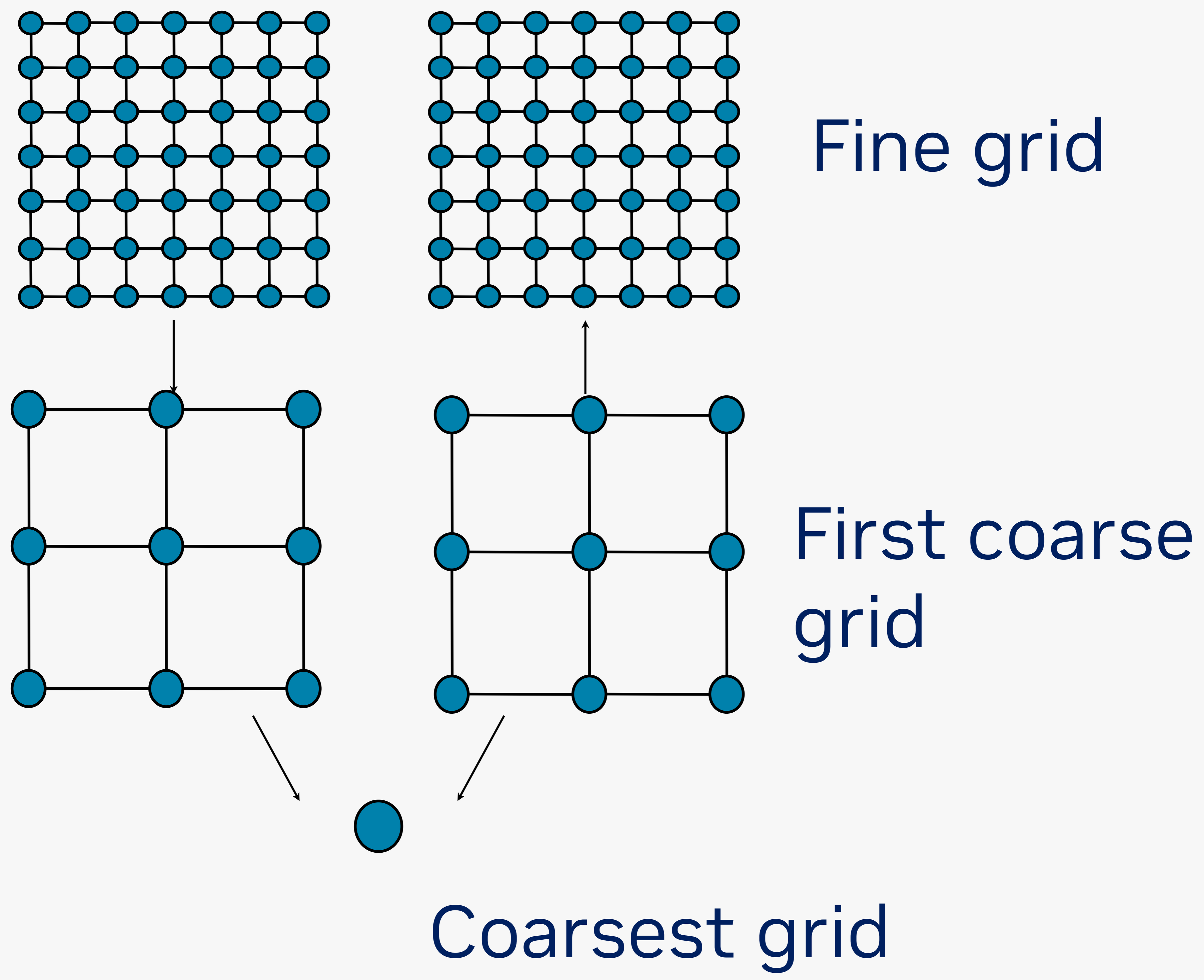
- Is the operator amenable to a Galerkin ($\widehat{D} = P^{\dagger}D\ P$) projection?
- What’s the ideal way to generate near-null vectors?
- What’s the ideal smoother?




# A Zoo of Questions

Math, Physics, Software, Hardware, Algorithms, Power Bills…

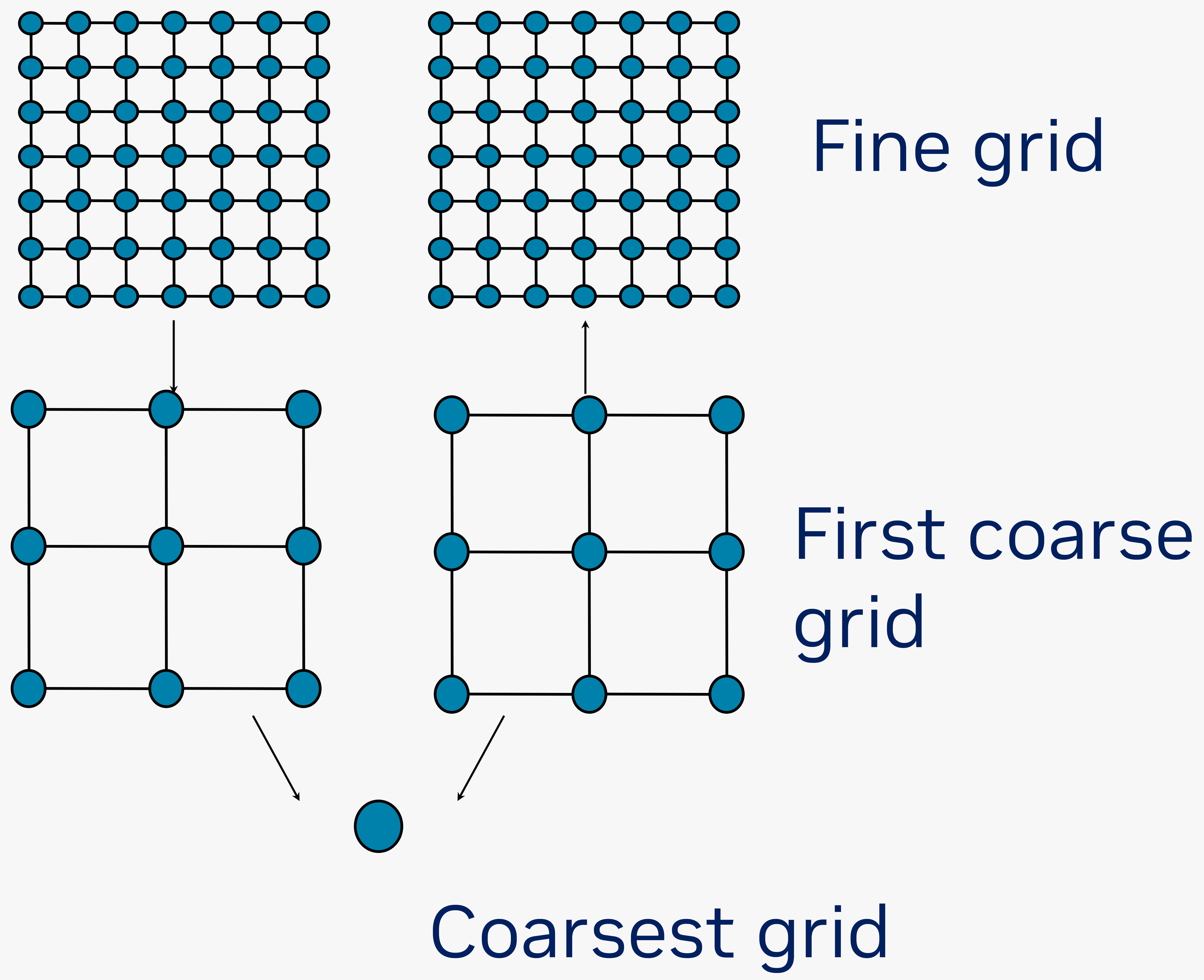
- Is the operator amenable to a Galerkin ($\widehat{D} = P^{\dagger}D\ P$) projection?
- What's the ideal way to generate near-null vectors?
- What's the ideal smoother?
- What's an optimal number of levels?




# A Zoo of Questions

Math, Physics, Software, Hardware, Algorithms, Power Bills…

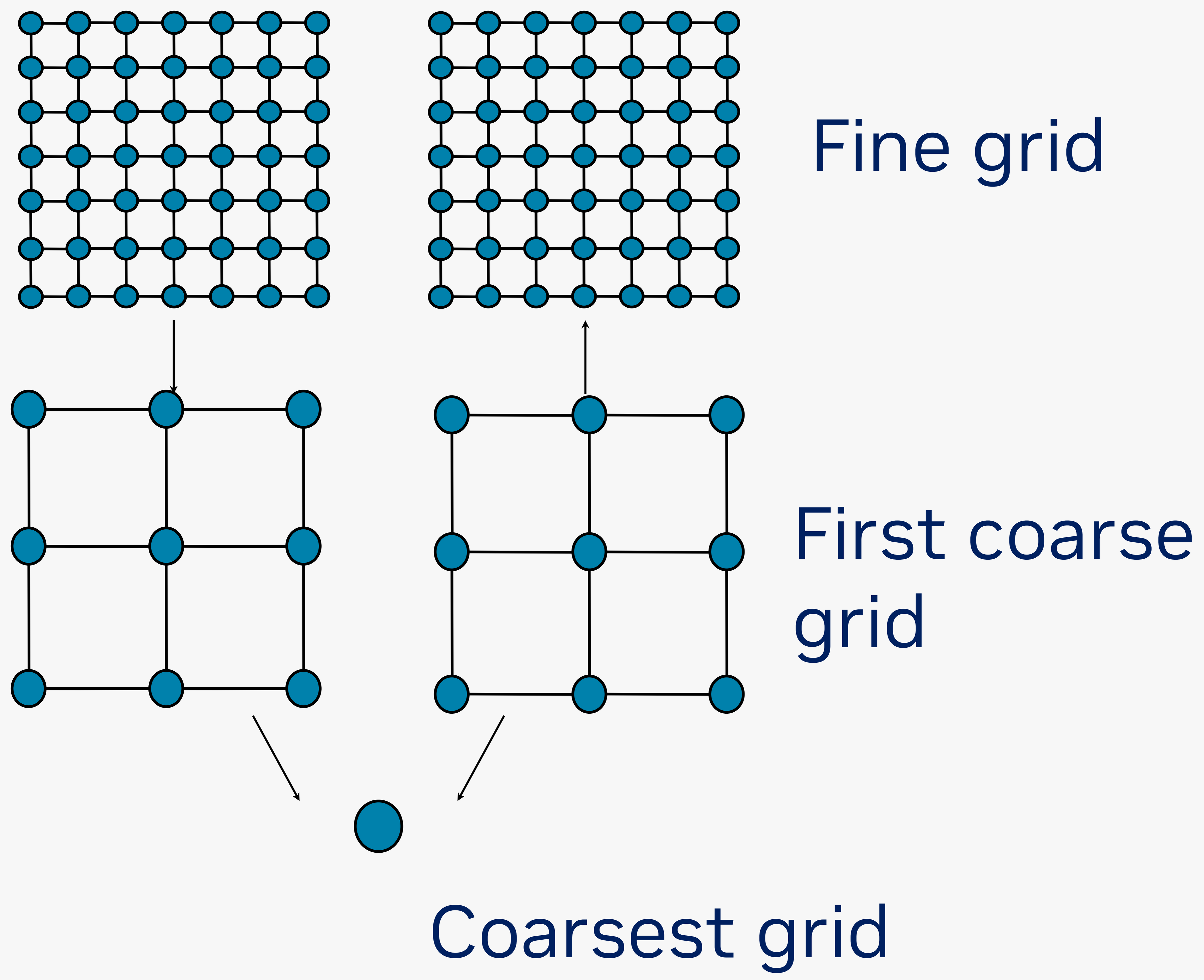
- Is the operator amenable to a Galerkin ($\widehat{D} = P^{\dagger} D\, P$) projection?
- What's the ideal way to generate near-null vectors?
- What's the ideal smoother?
- What's an optimal number of levels?
- How "accurately" do we solve on each level?




# A Zoo of Questions

Math, Physics, Software, Hardware, Algorithms, Power Bills…

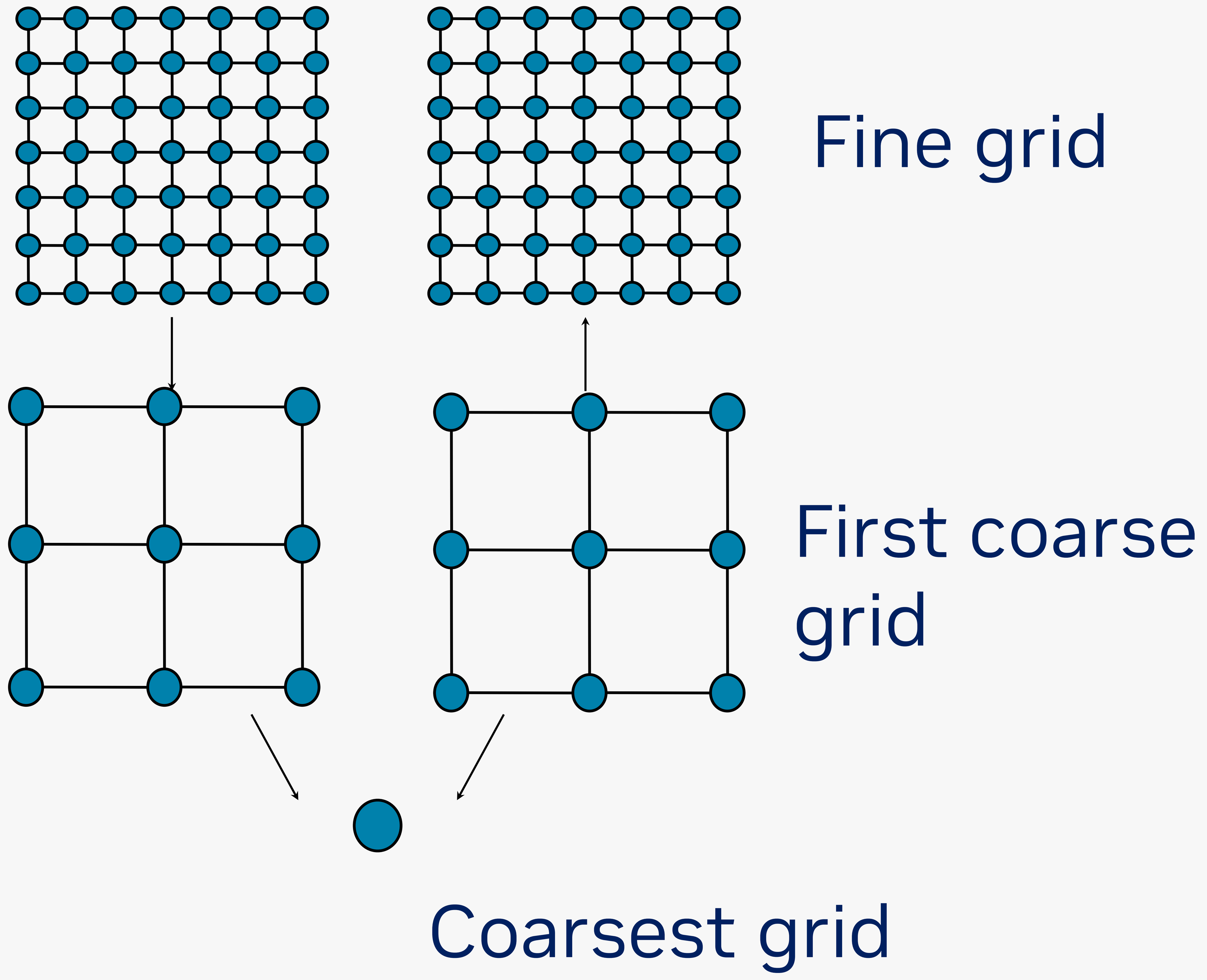
- Is the operator amenable to a Galerkin ($\widehat{D} = P^{\dagger}D\ P$) projection?
- What's the ideal way to generate near-null vectors?
- What's the ideal smoother?
- What's an optimal number of levels?
- How "accurately" do we solve on each level?
- How do I deal with remnant ill-conditioning on the coarsest level?




# A Zoo of Questions

Math, Physics, Software, Hardware, Algorithms, Power Bills…

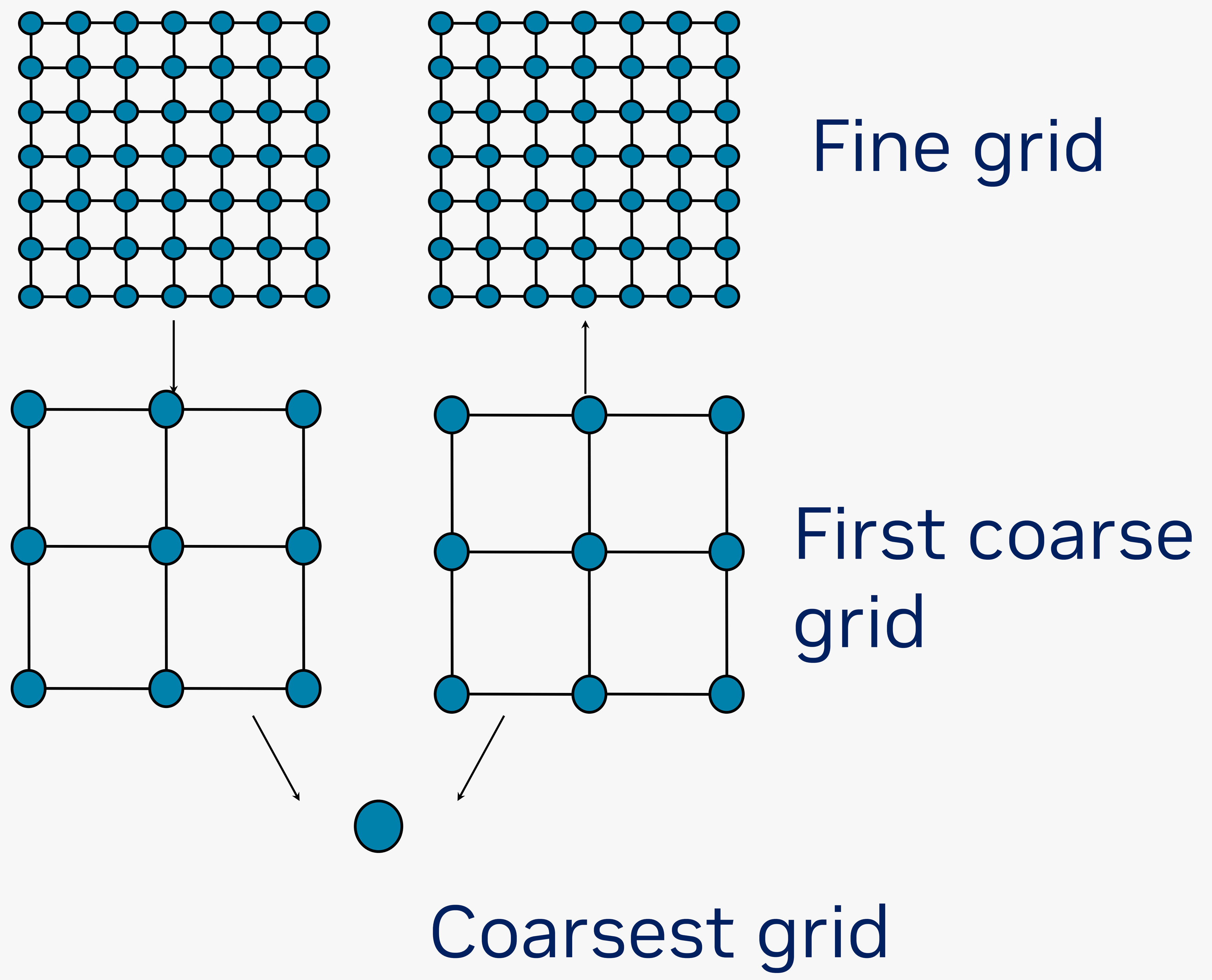
- Is the operator amenable to a Galerkin ($\widehat{D} = P^{\dagger}D\ P$) projection?
- What's the ideal way to generate near-null vectors?
- What's the ideal smoother?
- What's an optimal number of levels?
- How "accurately" do we solve on each level?
- How do I deal with remnant ill-conditioning on the coarsest level?
- How do we make it fast enough to include in HMC?




# A Zoo of Questions

## Math, Physics, Software, Hardware, Algorithms, Power Bills…

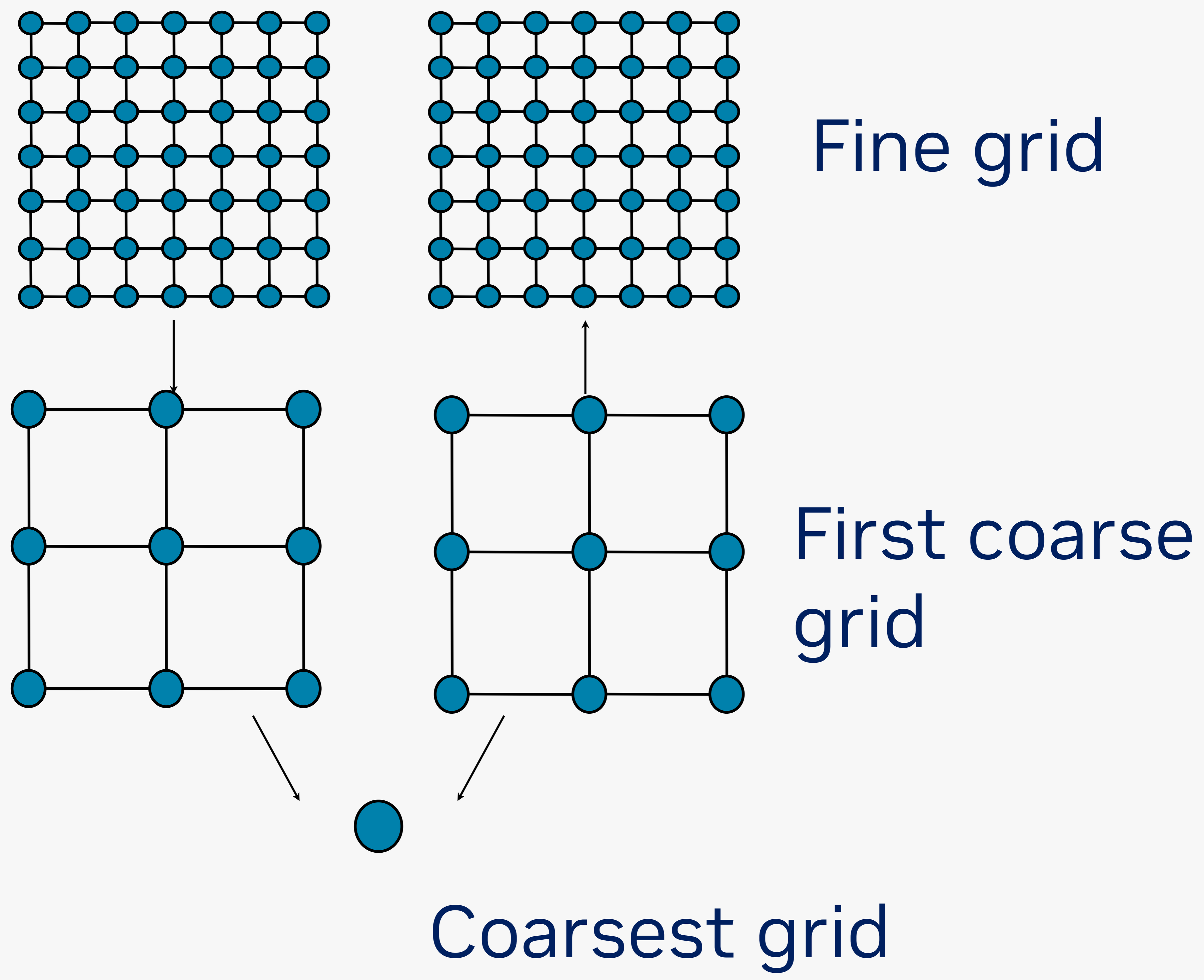
- Is the operator amenable to a Galerkin ($\widehat{D} = P^{\dagger} D\ P$) projection?
- What's the ideal way to generate near-null vectors?
- What's the ideal smoother?
- What's an optimal number of levels?
- How "accurately" do we solve on each level?
- How do I deal with remnant ill-conditioning on the coarsest level?
- How do we make it fast enough to include in HMC?
- **How do these answers change depending on the implementation and the hardware?**




# A Zoo of Questions

Math, Physics, Software, Hardware, Algorithms, Power Bills…

- Is the operator amenable to a Galerkin ($\widehat{D} = P^{\dagger} D\, P$) projection?
- What's the ideal way to generate near-null vectors?
- What's the ideal smoother?
- What's an optimal number of levels?
- How "accurately" do we solve on each level?
- How do I deal with remnant ill-conditioning on the coarsest level?
- How do we make it fast enough to include in HMC?
- **How do these answers change depending on the implementation and the hardware?**
- **How do we take advantage of AI-driven hardware features?**




# A Zoo of Questions

Math, Physics, Software, Hardware, Algorithms, Power Bills…

- Is the operator amenable to a Galerkin ($\widehat{D} = P^{\dagger}D\,P$) projection?
- What's the ideal way to generate near-null vectors?
- What's the ideal smoother?
- What's an optimal number of levels?
- How "accurately" do we solve on each level?
- How do I deal with remnant ill-conditioning on the coarsest level?
- How do we make it fast enough to include in HMC?
- **How do these answers change depending on the implementation and the hardware?**
- **How do we take advantage of AI-driven hardware features?**
- **Where does this live in an energy-constrained world?**

# Multigrid in Practice

# Vehicle for discussion: QUDA

“QCD on CUDA”… and many more these days

- “QCD on CUDA” – http://lattice.github.com/quda (open source, BSD license)

EXASCALE COMPUTING PROJECT

SciDAC
Scientific Discovery through Advanced Computing

NSF

PRACE

# Vehicle for discussion: QUDA

“QCD on CUDA”… and many more these days

- “QCD on CUDA” – http://lattice.github.com/quda (open source, BSD license)
- Effort started at Boston University in 2008, now in wide use as the GPU backend for BQCD, Chroma, CPS, MILC, openQ*D**, TIFR, etc. Provides solvers for all major fermionic discretizations, with multi-GPU support

**** See next talk by Roman Gruber, Tim Harris**

# Vehicle for discussion: QUDA

“QCD on CUDA”… and many more these days

- “QCD on CUDA” – http://lattice.github.com/quda (open source, BSD license)
- Effort started at Boston University in 2008, now in wide use as the GPU backend for BQCD, Chroma, CPS, MILC, openQ*D**, TIFR, etc. Provides solvers for all major fermionic discretizations, with multi-GPU support
- Maximize performance
  - Optimized implementations of major fermionic discretizations
  - Mixed-precision methods before they were cool
  - Eigensolvers, pure gauge algorithms, and more
  - Autotune and maximize performance
  - Batched solvers, deflation, and multi-grid acceleration
  - Tensor core acceleration
  - NVSHMEM for improving strong scaling
  - A performant algorithmic playground for exascale++

**** See next talk by Roman Gruber, Tim Harris**

ECP EXASCALE COMPUTING PROJECT

SciDAC Scientific Discovery through Advanced Computing

NSF

PRACE

# Vehicle for discussion: QUDA

“QCD on CUDA”… and many more these days

- “QCD on CUDA” – http://lattice.github.com/quda (open source, BSD license)
- Effort started at Boston University in 2008, now in wide use as the GPU backend for BQCD, Chroma, CPS, MILC, openQ*D**, TIFR, etc. Provides solvers for all major fermionic discretizations, with multi-GPU support
- Maximize performance
  - Optimized implementations of major fermionic discretizations
  - Mixed-precision methods before they were cool
  - Eigensolvers, pure gauge algorithms, and more
  - Autotune and maximize performance
  - Batched solvers, deflation, and multi-grid acceleration
  - Tensor core acceleration
  - NVSHMEM for improving strong scaling
  - A performant algorithmic playground for exascale++
- Portable: HIP (merged), SYCL (in review) and OpenMP (in development)

**** See next talk by Roman Gruber, Tim Harris**

EXASCALE COMPUTING PROJECT SciDAC Scientific Discovery through Advanced Computing NSF PRACE

# Vehicle for discussion: QUDA

"QCD on CUDA"… and many more these days

- "QCD on CUDA" – http://lattice.github.com/quda (open source, BSD license)
- Effort started at Boston University in 2008, now in wide use as the GPU backend for BQCD, Chroma, CPS, MILC, openQ*D**, TIFR, etc. Provides solvers for all major fermionic discretizations, with multi-GPU support
- Maximize performance
  - Optimized implementations of major fermionic discretizations
  - Mixed-precision methods before they were cool
  - Eigensolvers, pure gauge algorithms, and more
  - Autotune and maximize performance
  - Batched solvers, deflation, and multi-grid acceleration
  - Tensor core acceleration
  - NVSHMEM for improving strong scaling
  - A performant algorithmic playground for exascale++
- Portable: HIP (merged), SYCL (in review) and OpenMP (in development)
- A research tool for the exascale (and beyond)
  - Optimally mapping the problem to hierarchical processors and node topologies

**** See next talk by Roman Gruber, Tim Harris**

EXASCALE COMPUTING PROJECT

SciDAC
Scientific Discovery through Advanced Computing

NSF

PRACE

# QUDA Contributors

10+ Years, Lots of Contributors

- Buck Babich (NVIDIA)
- Simone Bacchio (Cyprus)
- Michael Baldfauf (Regensburg)
- Kip Barros (LANL)
- Rich Brower (Boston University)
- Nuno Cardoso (NCSA)
- Kate Clark (NVIDIA)
- Michael Cheng (Boston University)
- Carleton DeTar (Utah University)
- Justin Foley (NIH)
- Arjun Gambhir (William and Mary)
- Marco Garofalo (Bonn)
- Joel Giedt (Rensselaer Polytechnic Institute)
- Steve Gottlieb (Indiana University)
- Anthony Grebe (Fermilab)
- Kyriakos Hadjiyiannakou (Cyprus)
- Ben Hoerz (Intel)
- Dean Howarth (LBL)
- Hwancheol Jeong (Indiana University)
- Xiangyu Jiang (ITP, Chinese Academy of Sciences)
- Xiao-Yong Jin (ANL)
- Bálint Joó (NVIDIA)
- Hyung-Jin Kim (BNL -> Samsung)
- Bartek Kostrzewa (Bonn)
- Damon McDougall (AMD)
- Colin Morningstar (CMU)
- James Osborn (ANL)
- Ferenc Pittler (Cyprus)
- Claudio Rebbi (Boston University)
- Eloy Romero (William and Mary)
- Hauke Sandmeyer (Bielefeld)
- Mario Schröck (INFN)
- Aniket Sen (Bonn)
- Guochun Shi (NCSA -> Google)
- James Simone (FNAL)
- Alexei Strelchenko (FNAL)
- Jiqun Tu (NVIDIA)
- Carsten Urbach (HISKP, University of Bonn)
- Alejandro Vaquero (Utah University)
- Michael Wagman (FNAL)
- Mathias Wagner (NVIDIA)
- André Walker-Loud (LBL)
- Evan Weinberg (NVIDIA)
- Frank Winter (Jlab)
- Yi-bo Yang (CAS)

# New(s to me): QUDA bindings for Python

So new I haven't even tried them

- https://arxiv.org/abs/2411.08461
- I'm not kidding, I haven't tried them yet
- If you have---I'd love to hear your experience with them

## Use QUDA for lattice QCD calculation with Python

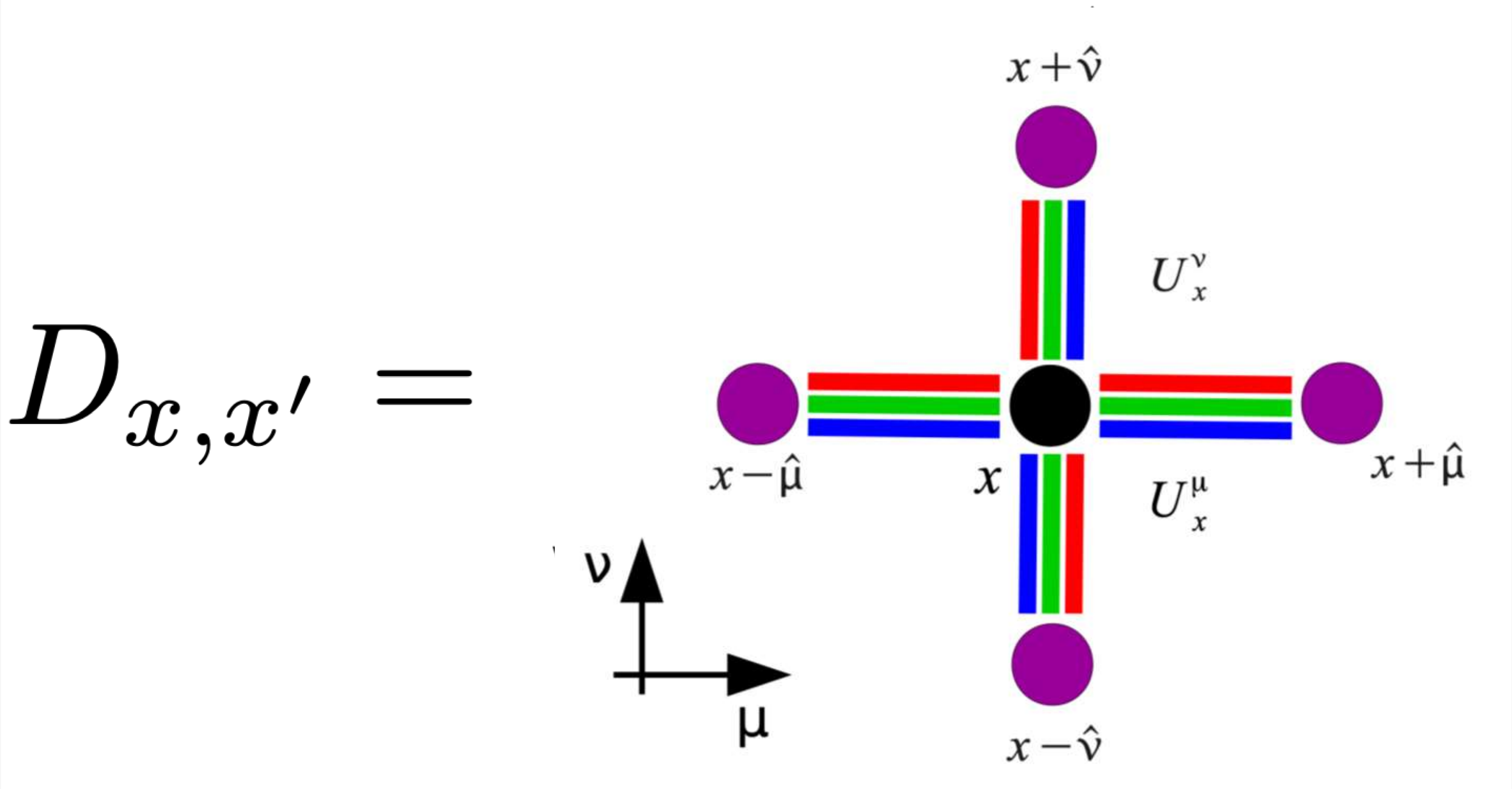
Xiangyu Jiang, Chunjiang Shi, Ying Chen, Ming Gong, Yi-Bo Yang

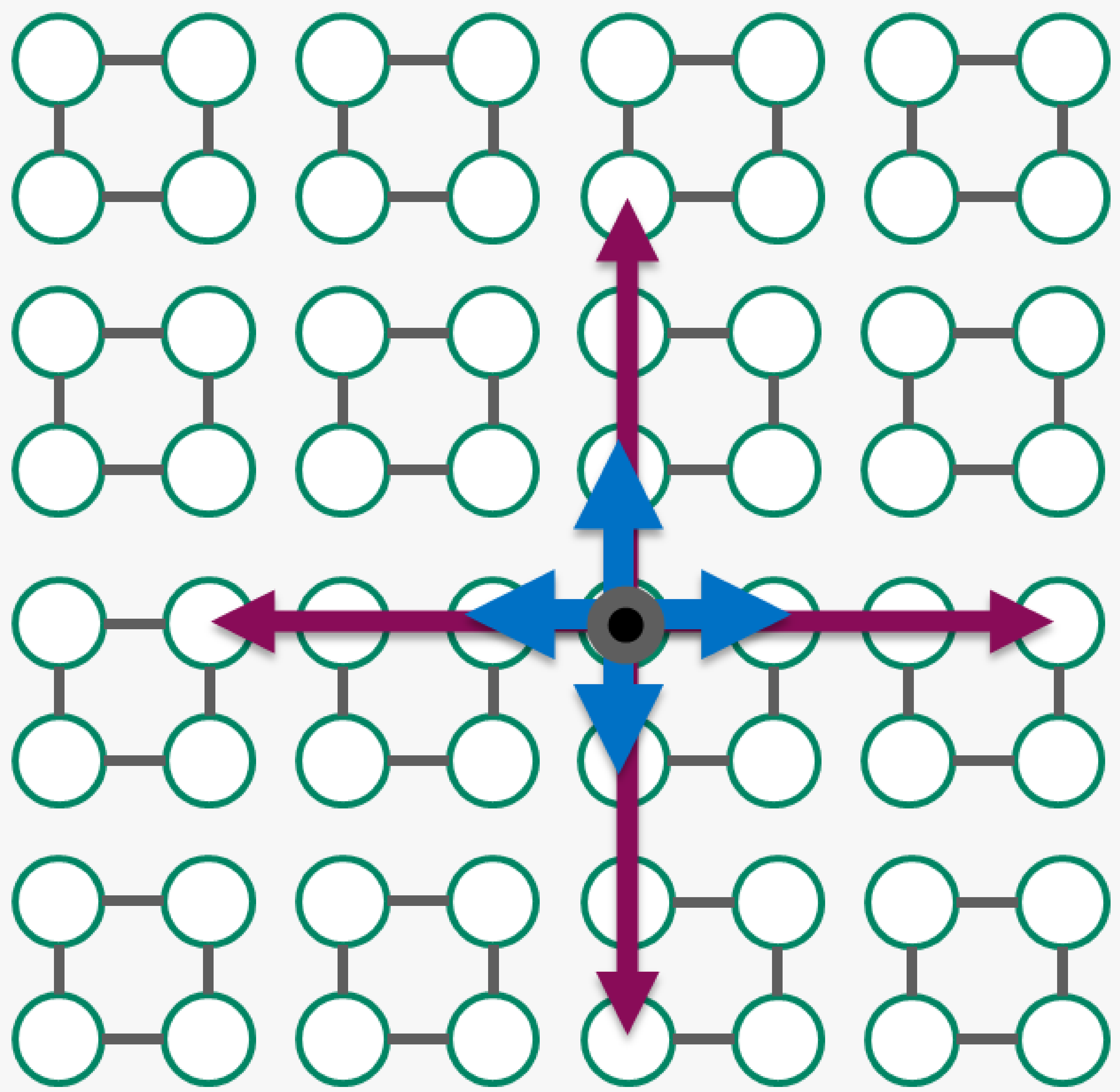
We developed PyQUDA, a Python wrapper for QUDA written in Cython, designed to facilitate lattice QCD calculations using the Python programming language. PyQUDA leverages the optimized linear algebra capabilities of NumPy/CuPy/PyTorch, along with the highly optimized lattice QCD operations provided by QUDA to accelerate research. This integration simplifies the process of writing calculation codes, enabling researchers to build more complex Python packages like EasyDistillation for specific physics objectives. PyQUDA supports a range of lattice QCD operations, including hybrid Monte Carlo (HMC) with N-flavor clover/HISQ fermions and inversion for the Wilson/clover/HISQ fermion action with the multigrid solver. It also includes utility functions for reading lattice QCD data stored in Chroma, MILC, and *[Math Processing Error]*QCD formats. Type hints are supported by stub files and multi-GPU support is provided through mpi4py.

# (Fine) Discretizations on GPUs

## Parallelism, parallelism, parallelism...

- Assign a single space-time point to each thread
  - V = XYZT threads, e.g., V = $24^4$ => $3.3\times10^6$ threads

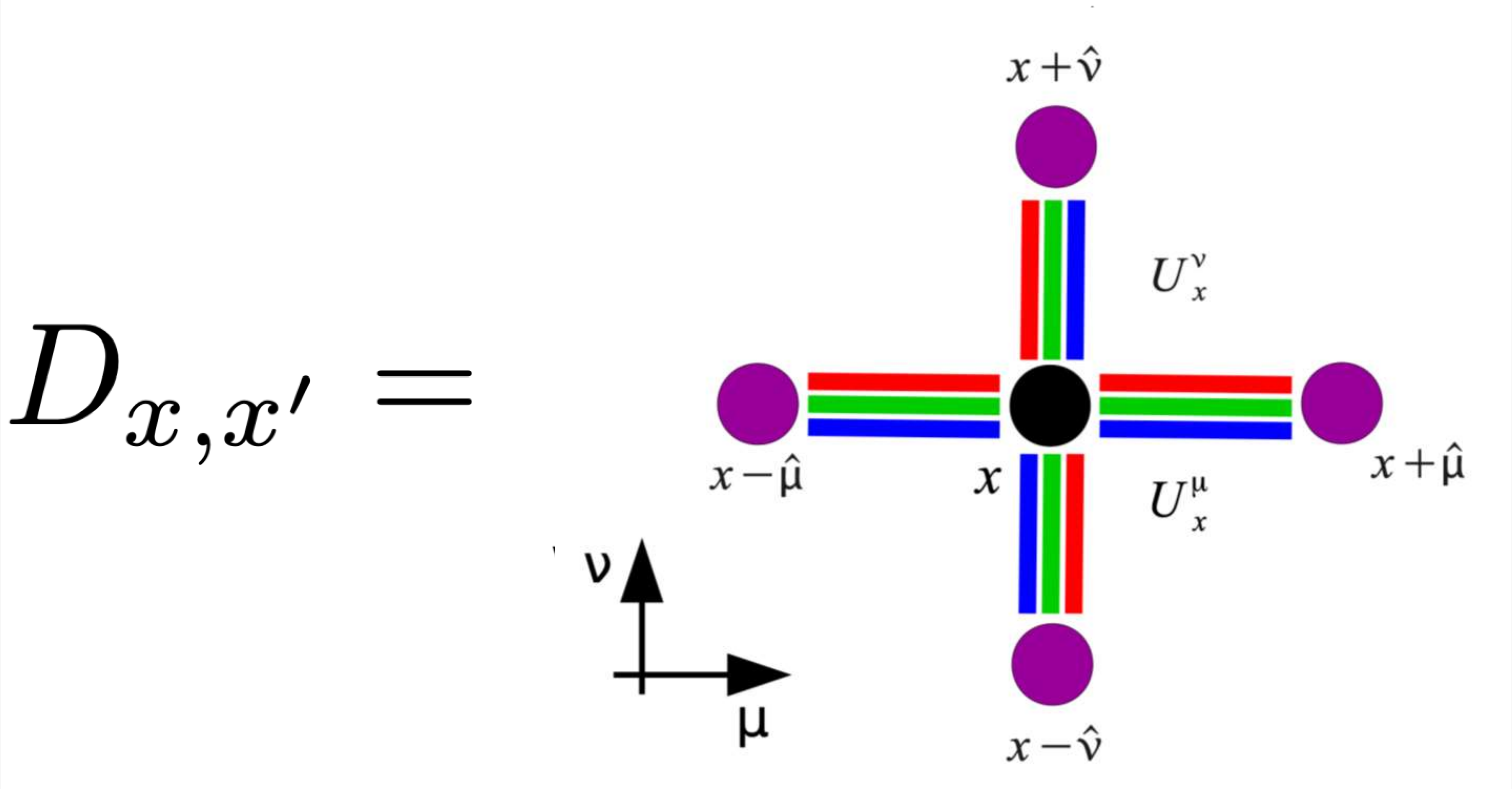


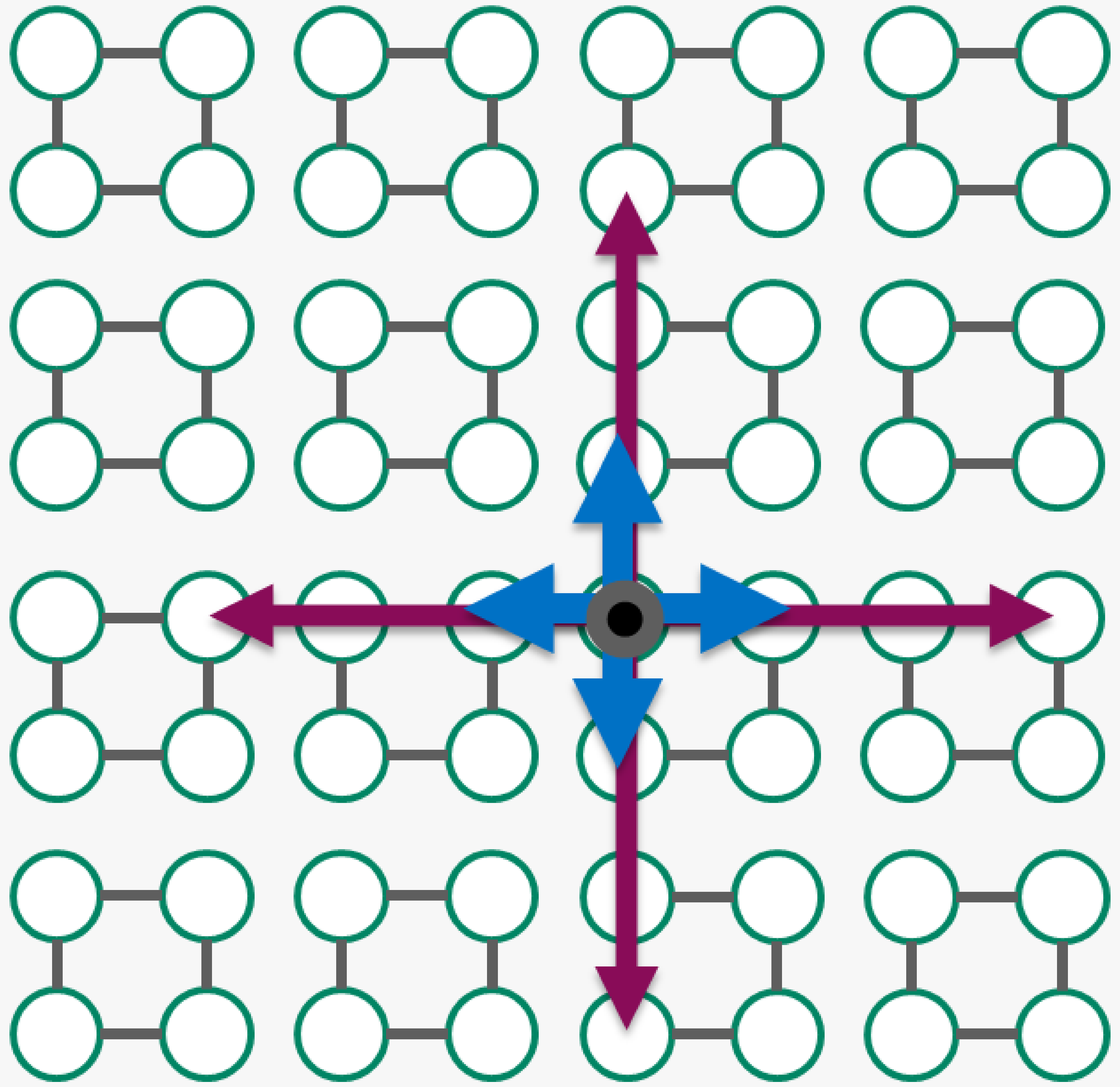



# (Fine) Discretizations on GPUs

Parallelism, parallelism, parallelism...

- Assign a single space-time point to each thread
  - V = XYZT threads, e.g., $V = 24^4 \Rightarrow 3.3\times10^6$ threads
- Each thread must:
  - Load neighboring spinors
    - Opportunity for cache re-use
  - Load gauge/fat/long links (no reuse*)
    - *We'll get to multi-rhs later

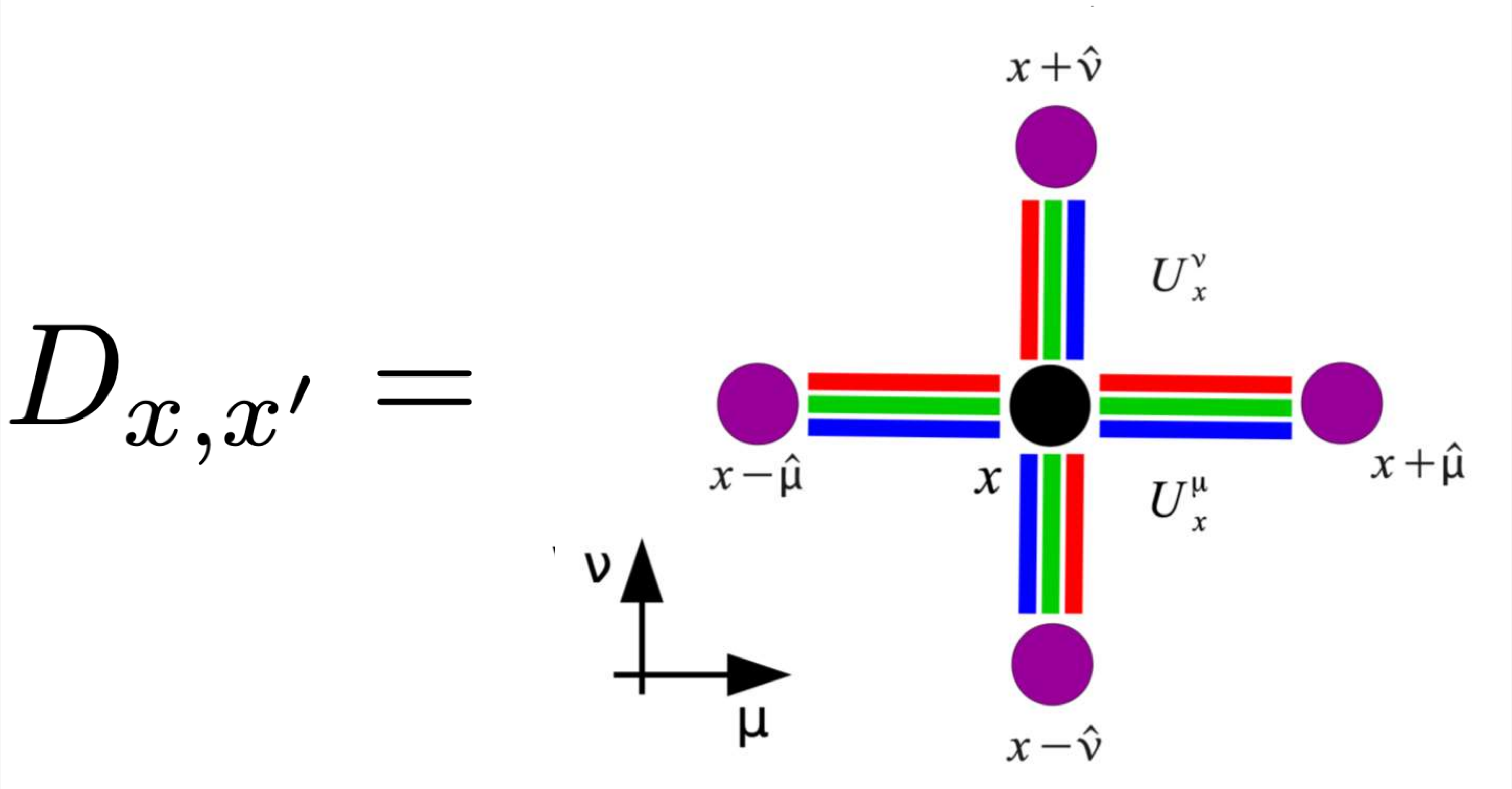


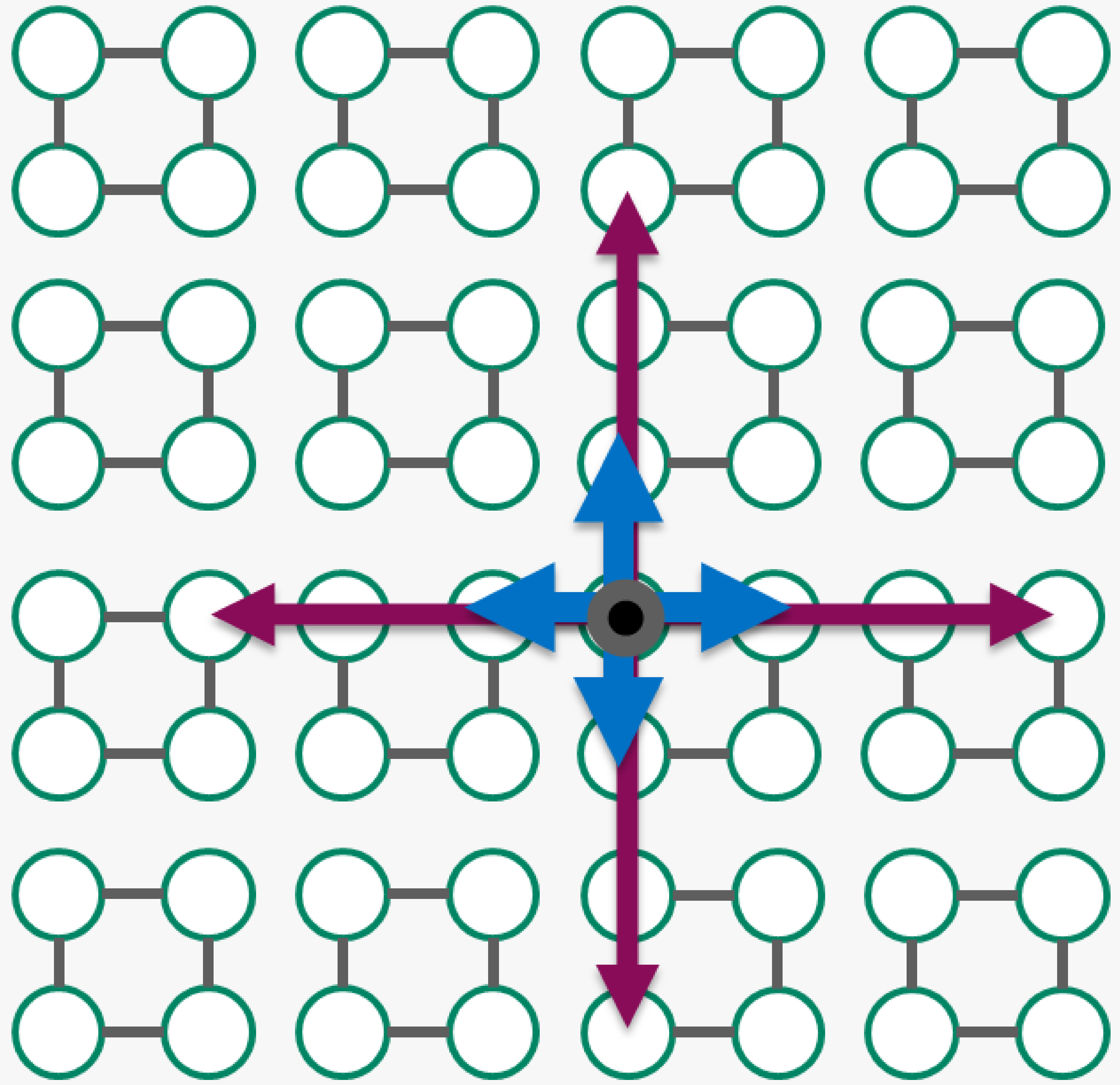



# (Fine) Discretizations on GPUs

Parallelism, parallelism, parallelism…

- Assign a single space-time point to each thread
  - V = XYZT threads, e.g., $V = 24^4$ => $3.3 \times 10^6$ threads
- Each thread must:
  - Load neighboring spinors
    - Opportunity for cache re-use
  - Load gauge/fat/long links (no reuse*)
    - *We'll get to multi-rhs later
- FP32 arithmetic intensities:
  - Wilson operator: ~0.92 (naïve)
  - HISQ operator: ~0.73 (naïve)

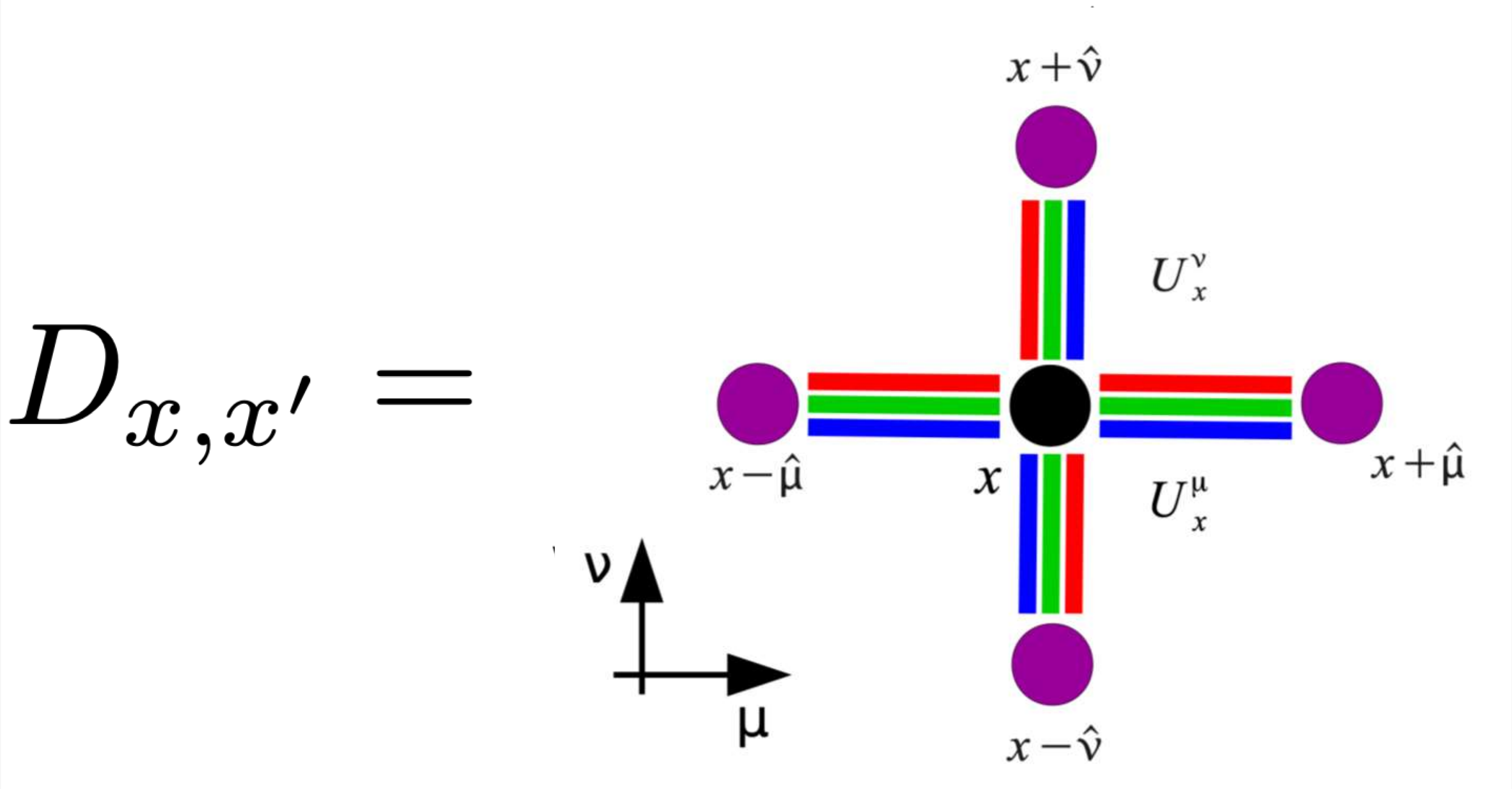


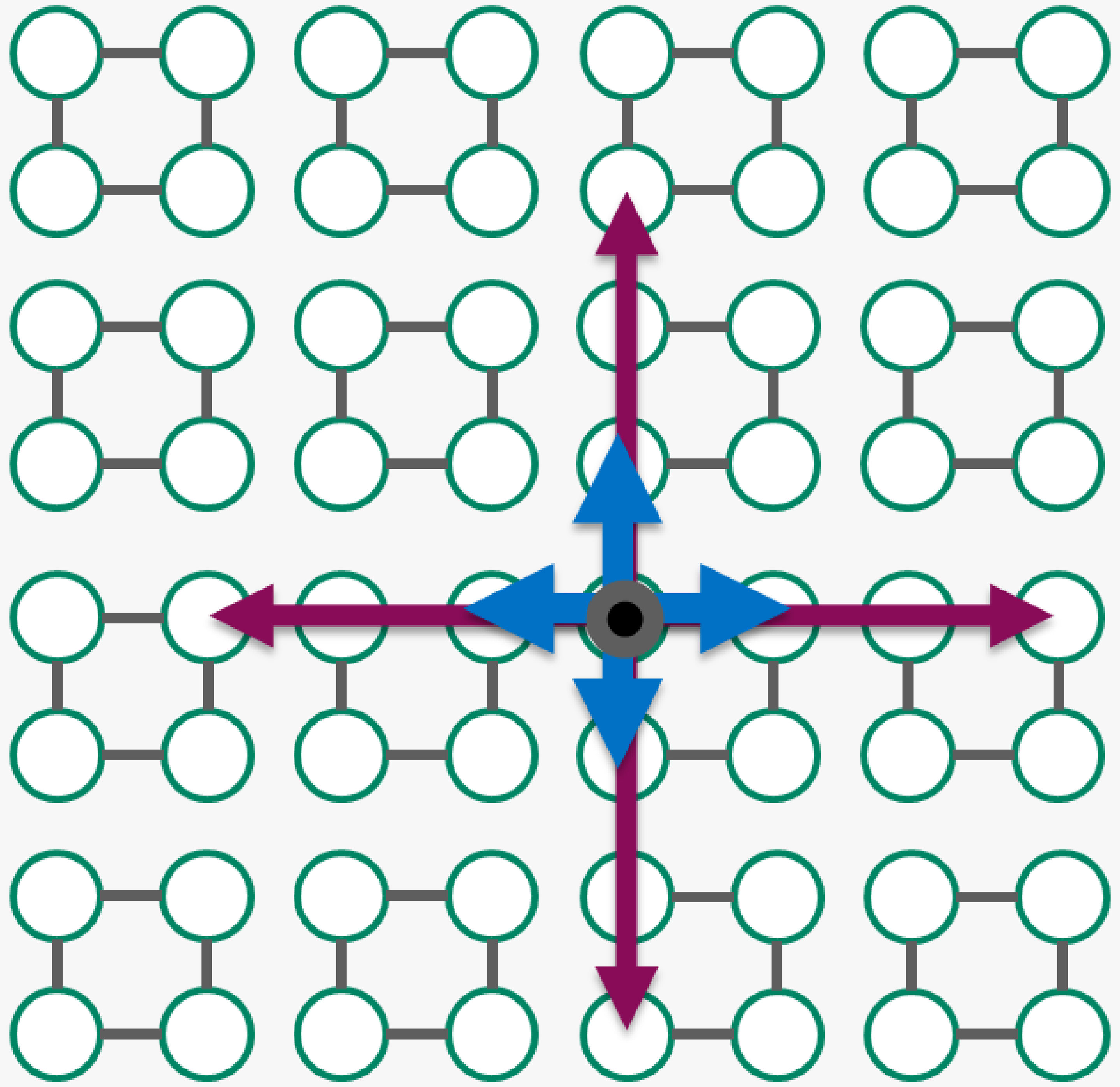



# (Fine) Discretizations on GPUs

Parallelism, parallelism, parallelism…

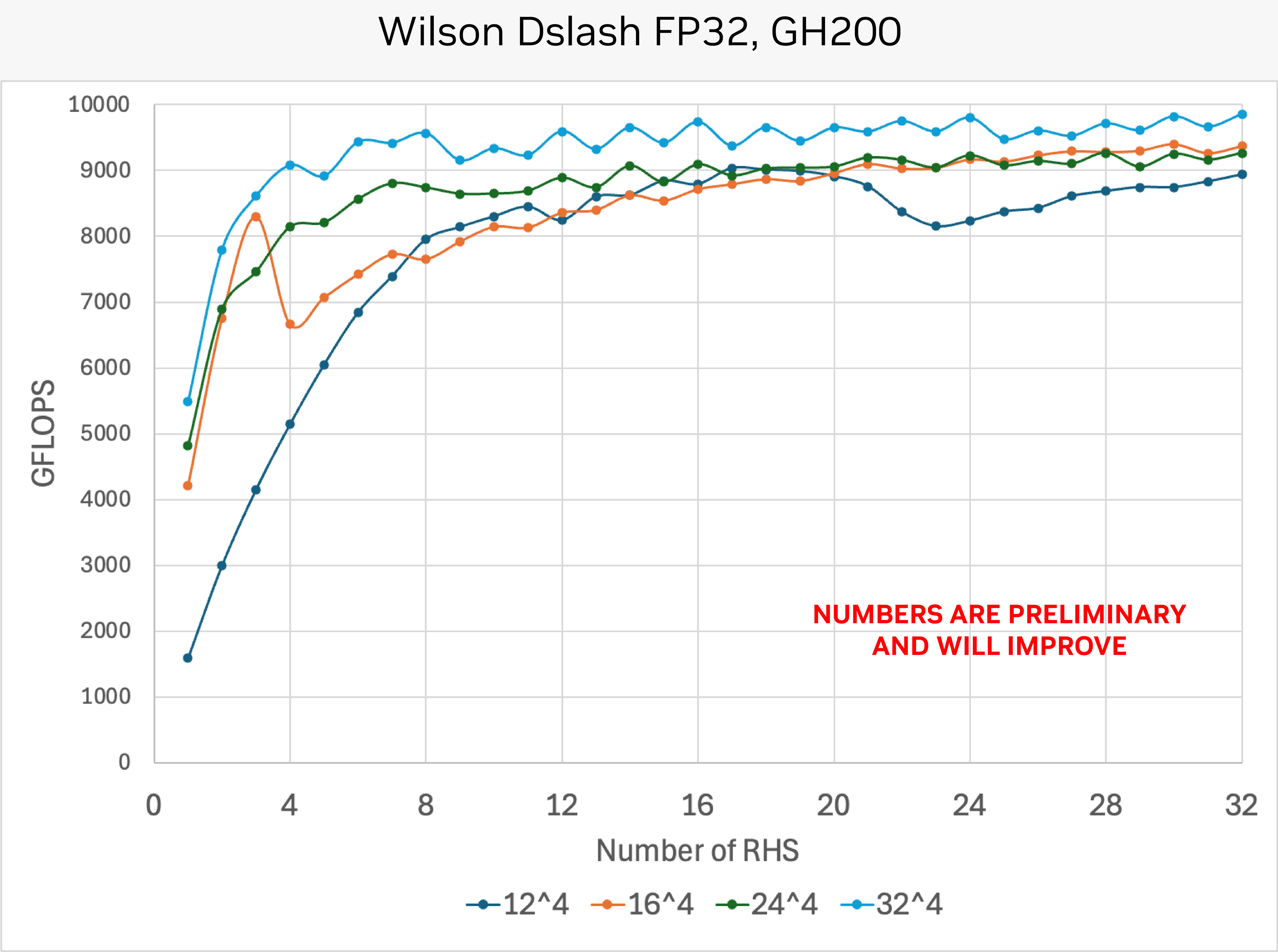
- Assign a single space-time point to each thread
  - V = XYZT threads, e.g., V = $24^4$ => $3.3 \times 10^6$ threads
- Each thread must:
  - Load neighboring spinors
    - Opportunity for cache re-use
  - Load gauge/fat/long links (no reuse*)
    - *We'll get to multi-rhs later
- FP32 arithmetic intensities:
  - Wilson operator: ~0.92 (naïve)
  - HISQ operator: ~0.73 (naïve)
- QUDA reduces memory traffic
  - SU(3) matrices: 18 -> 12 or 8 reals
  - HISQ U(3) long links: 18 -> 13 or 9 reals
  - Mixed-precision solvers: custom 16-bit fixed point representation



# Batched Wilson Dslash

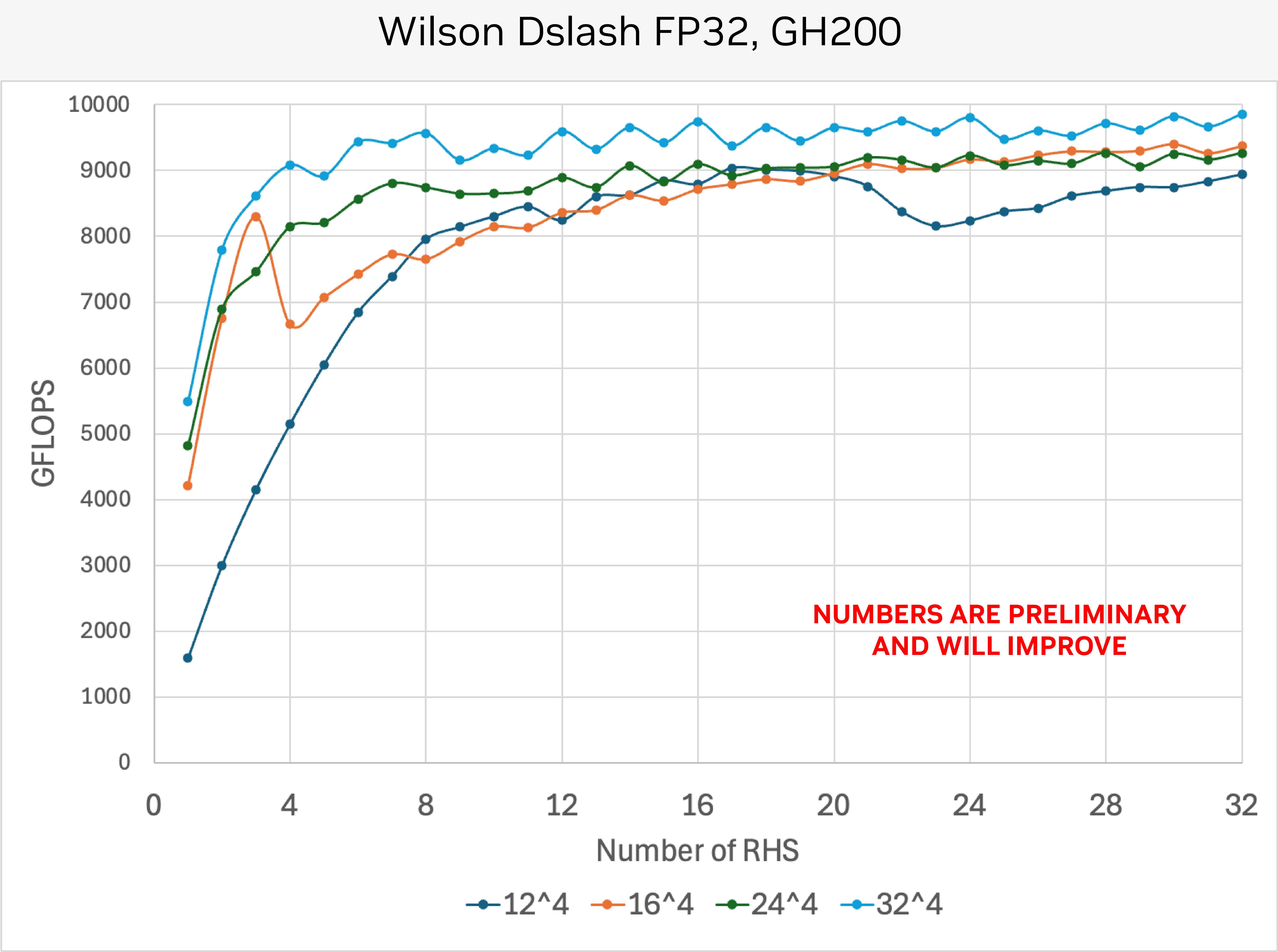
Parallelism, parallelism, parallelism, parallelism



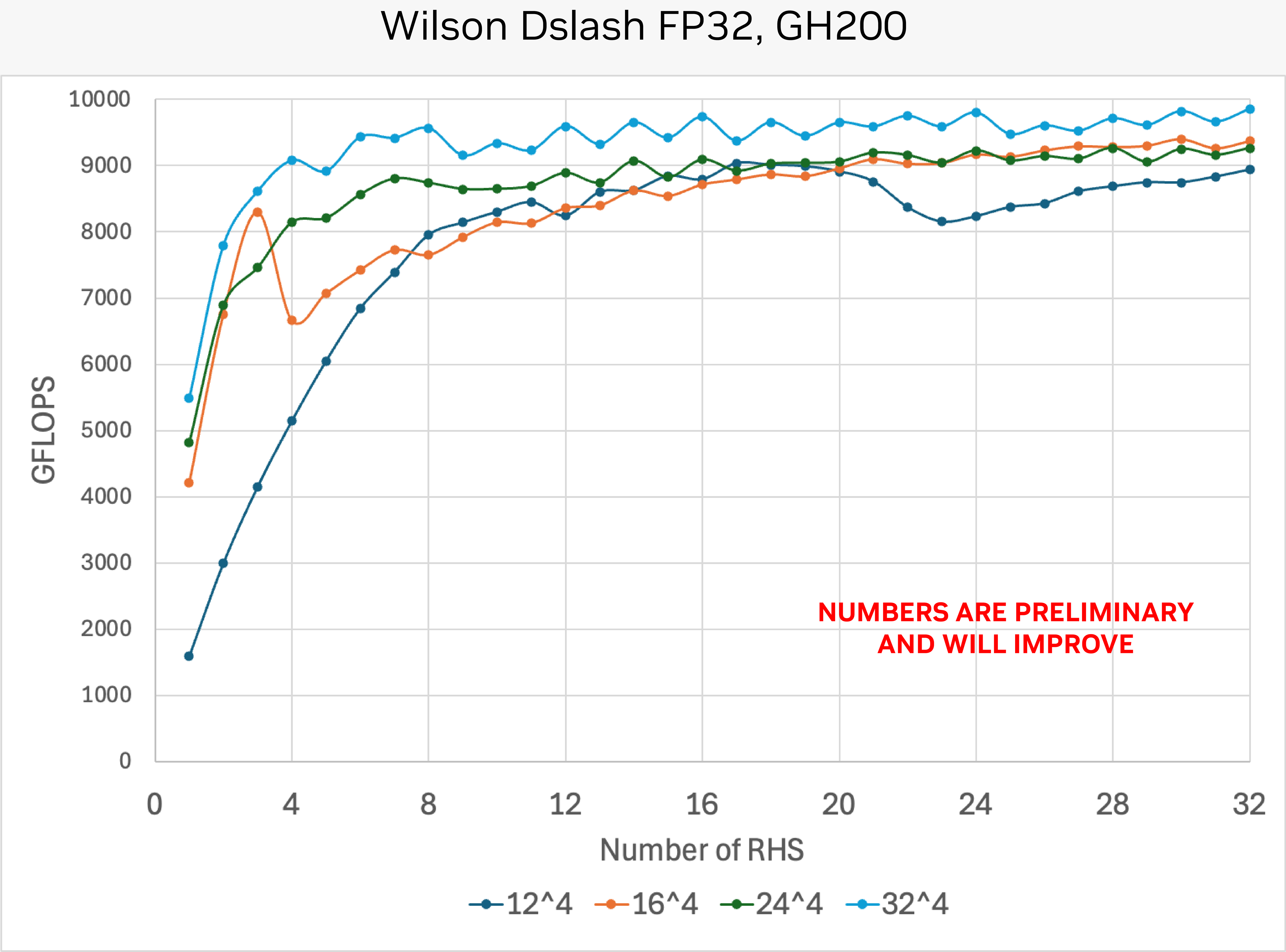

- Smaller volumes see the biggest boost in performance
  - Parallelism + Locality

# Batched Wilson Dslash

Parallelism, parallelism, parallelism, parallelism



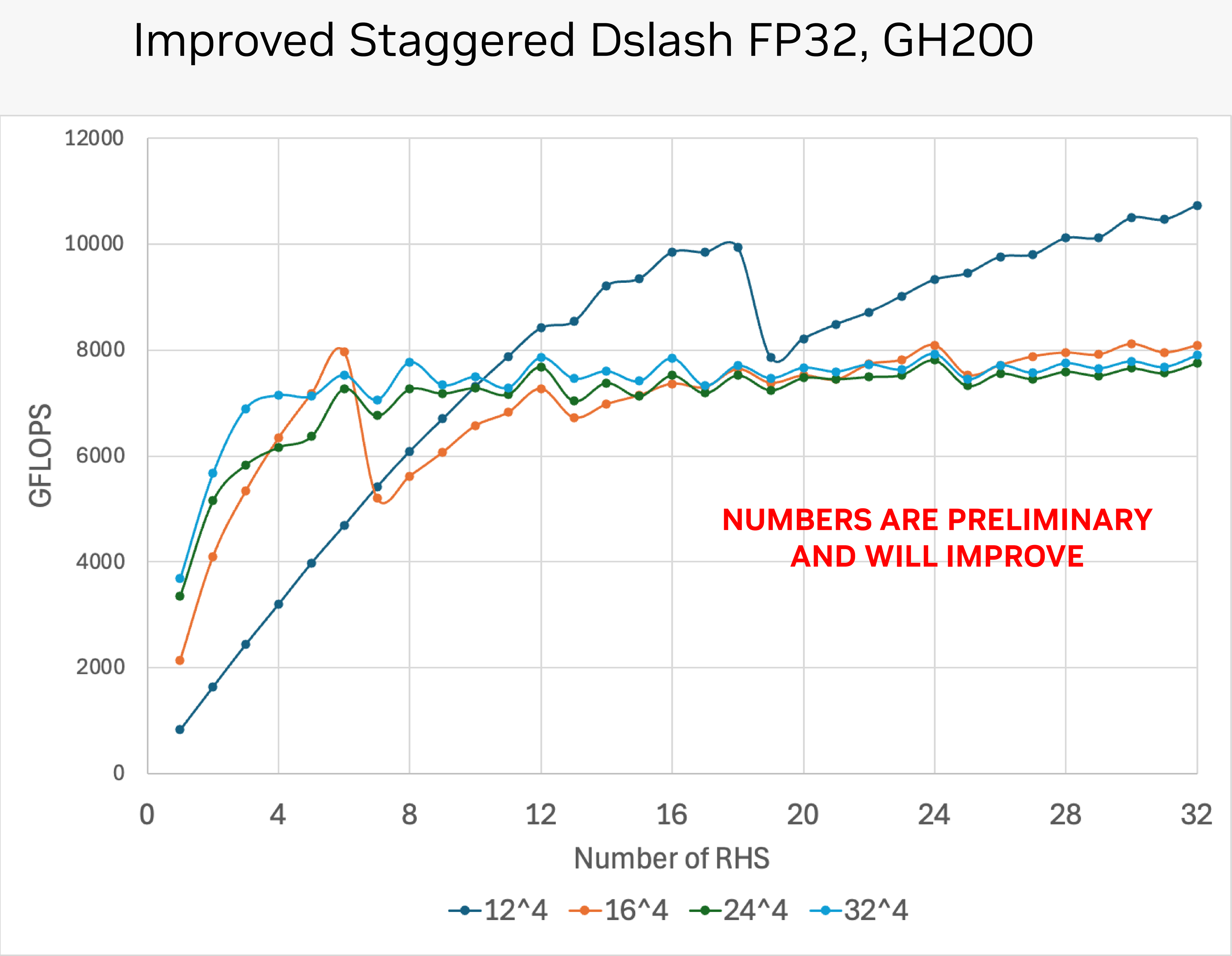

- Smaller volumes see the biggest boost in performance
  - Parallelism + Locality
- Larger volumes on see boost due to locality




# Batched Wilson Dslash

## Parallelism, parallelism, parallelism, parallelism

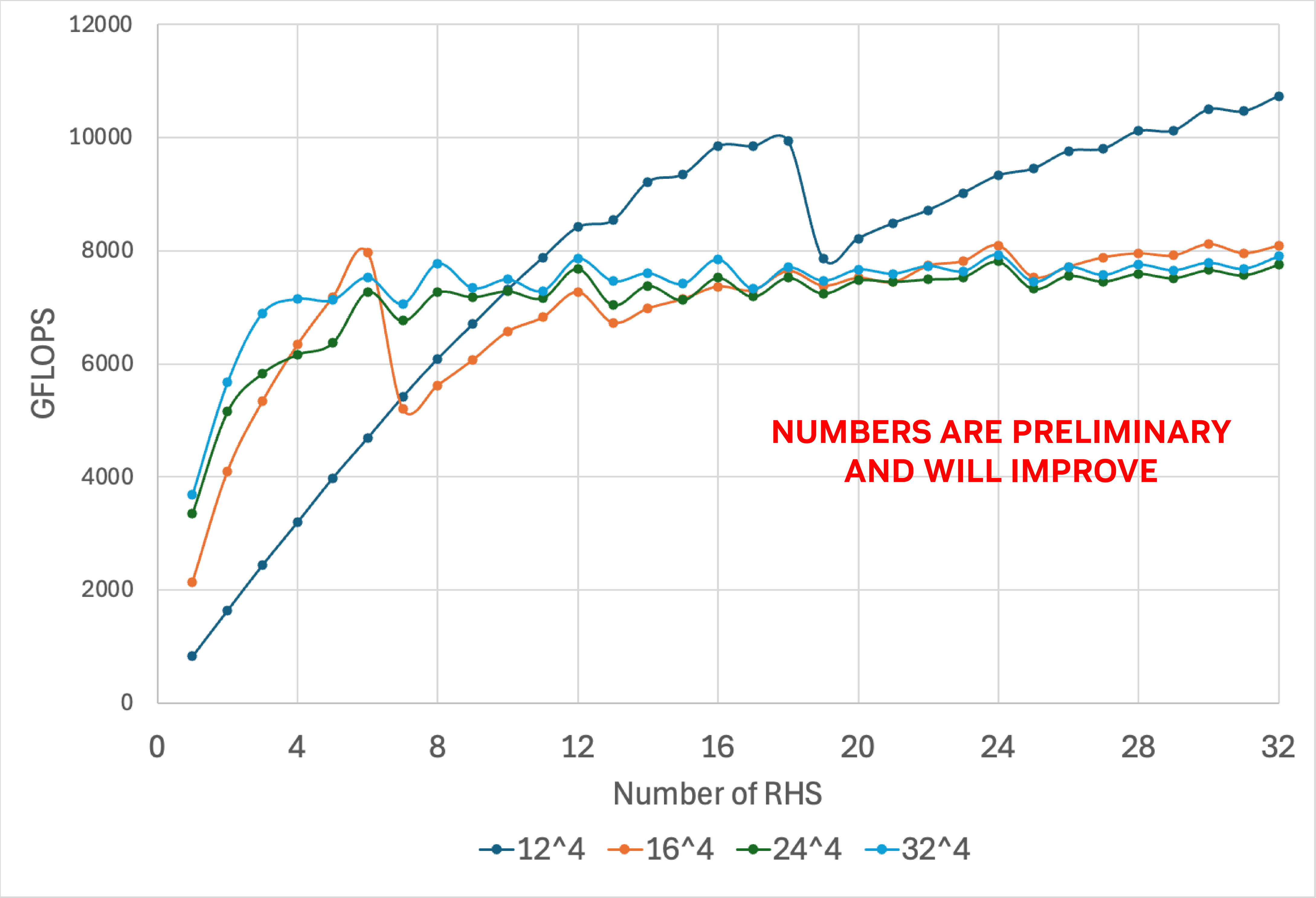
- Smaller volumes see the biggest boost in performance
  - Parallelism + Locality
- Larger volumes on see boost due to locality
- QUDA lets the *autotuner* decide how many sources to include in each block
  - More sources per block? Reuse of gauge fields
  - Fewer sources per block? Spatial/temporal reuse of spinors

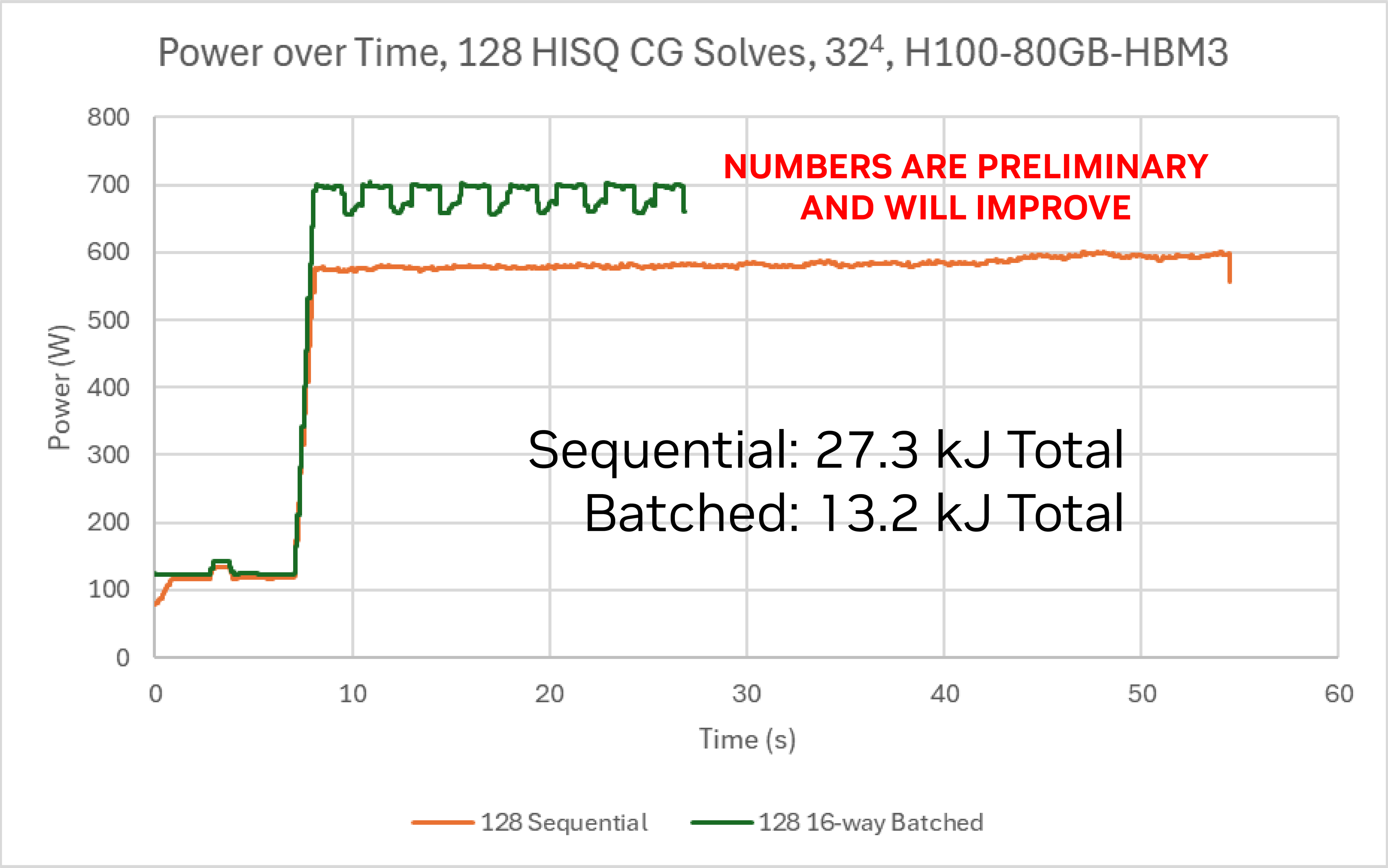



# Batched Improved Staggered

- Similar story for staggered
  - Larger speedups due to increased locality of staggered operator
  - $12^4$ has L1 cache quantization effects

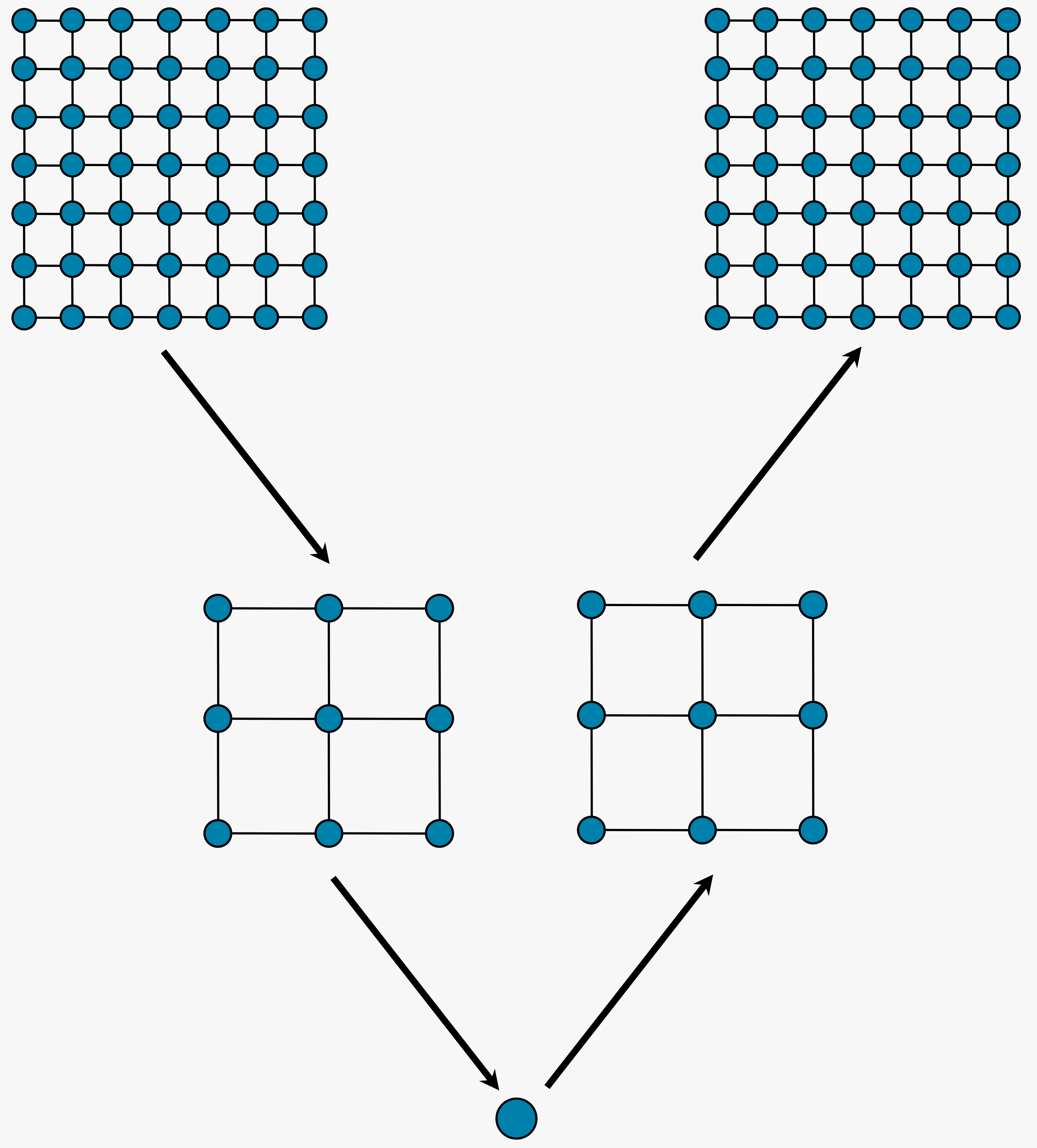
Improved Staggered Dslash FP32, GH200



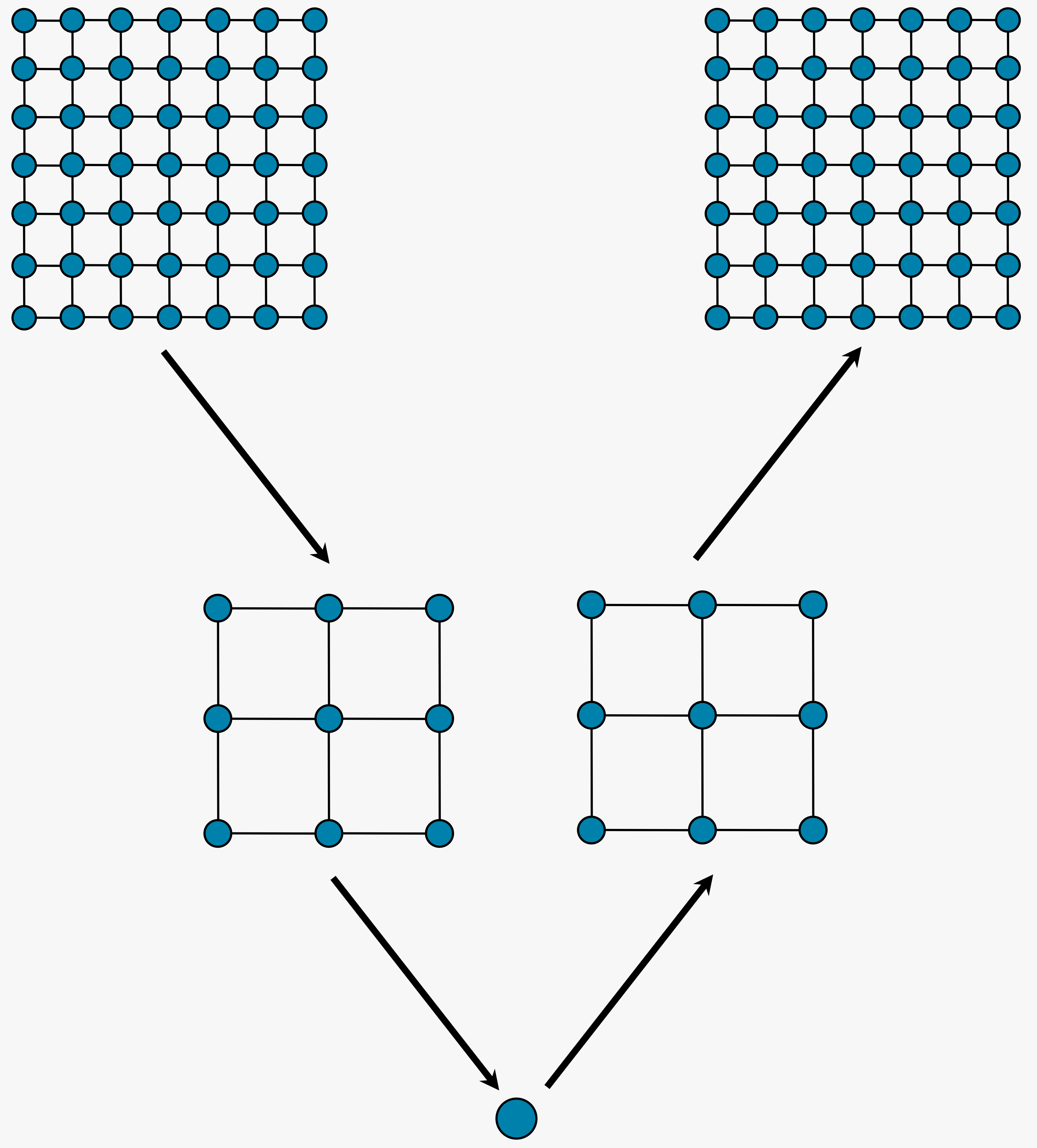

# Batched Improved Staggered

- Similar story for staggered
  - Larger speedups due to increased locality of staggered operator
  - $12^4$ has L1 cache quantization effects
- Preview: batching not only saves time, but energy
  - Moving electrons takes energy (intro physics)
  - Batching increases cache locality
  - Electrons don't need to move as far
  - Energy requirements go down



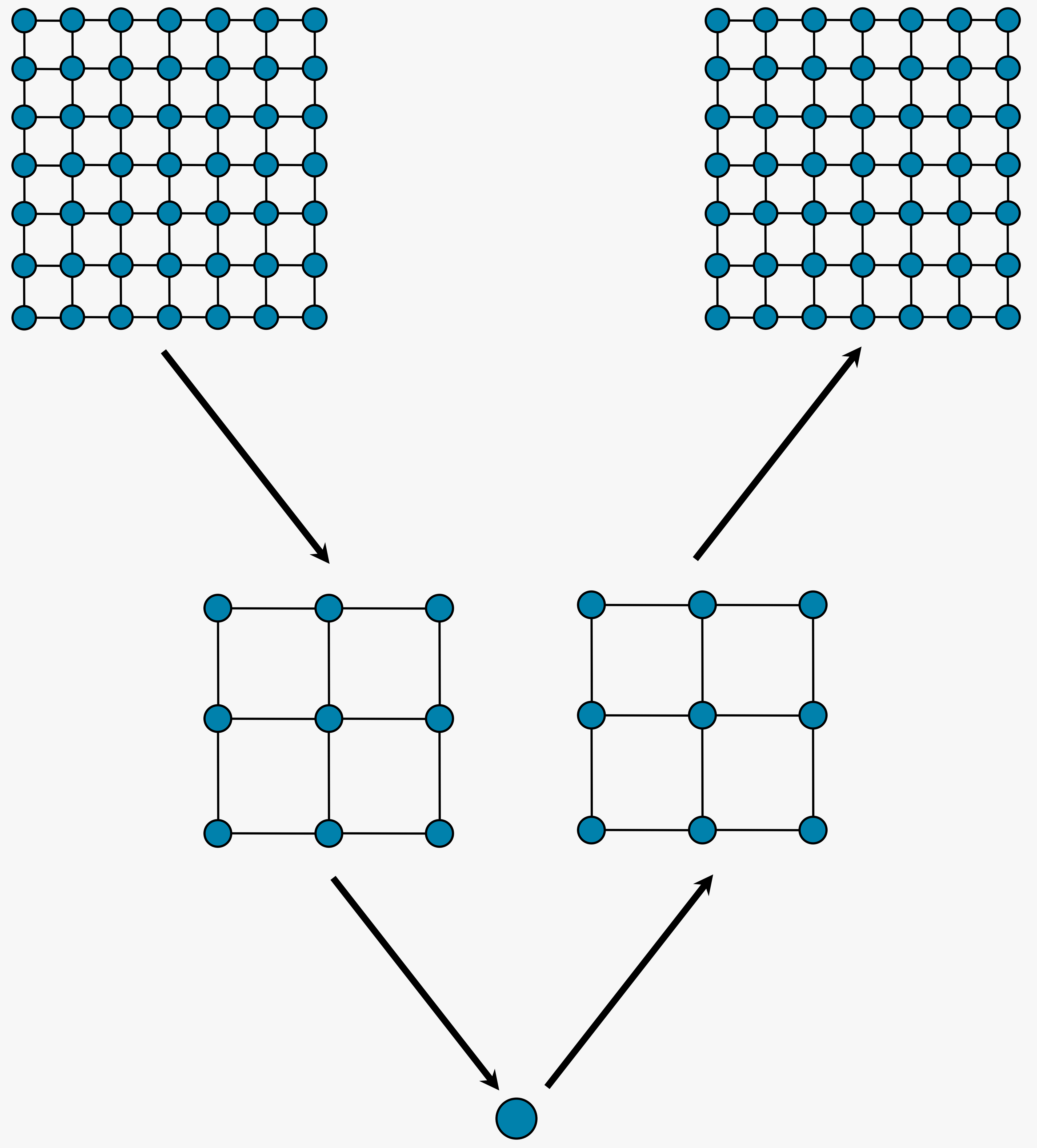
# The Challenge of Multigrid on the GPU

- Fine grids run very efficiently
  - High parallel throughput problem



# The Challenge of Multigrid on the GPU

- Fine grids run very efficiently
  - High parallel throughput problem
- Coarse grids are worst possible scenario
  - More cores than degrees of freedom
  - Increasingly serial and latency bound
  - Little's law (bytes = bandwidth * latency)
  - Amdahl's law limiter



# The Challenge of Multigrid on the GPU

- Fine grids run very efficiently
  - High parallel throughput problem
- Coarse grids are worst possible scenario
  - More cores than degrees of freedom
  - Increasingly serial and latency bound
  - Little’s law (bytes = bandwidth * latency)
  - Amdahl’s law limiter
- Multigrid exposes many of the problems we see at the exascale

# Ingredients for Parallel Adaptive Multigrid

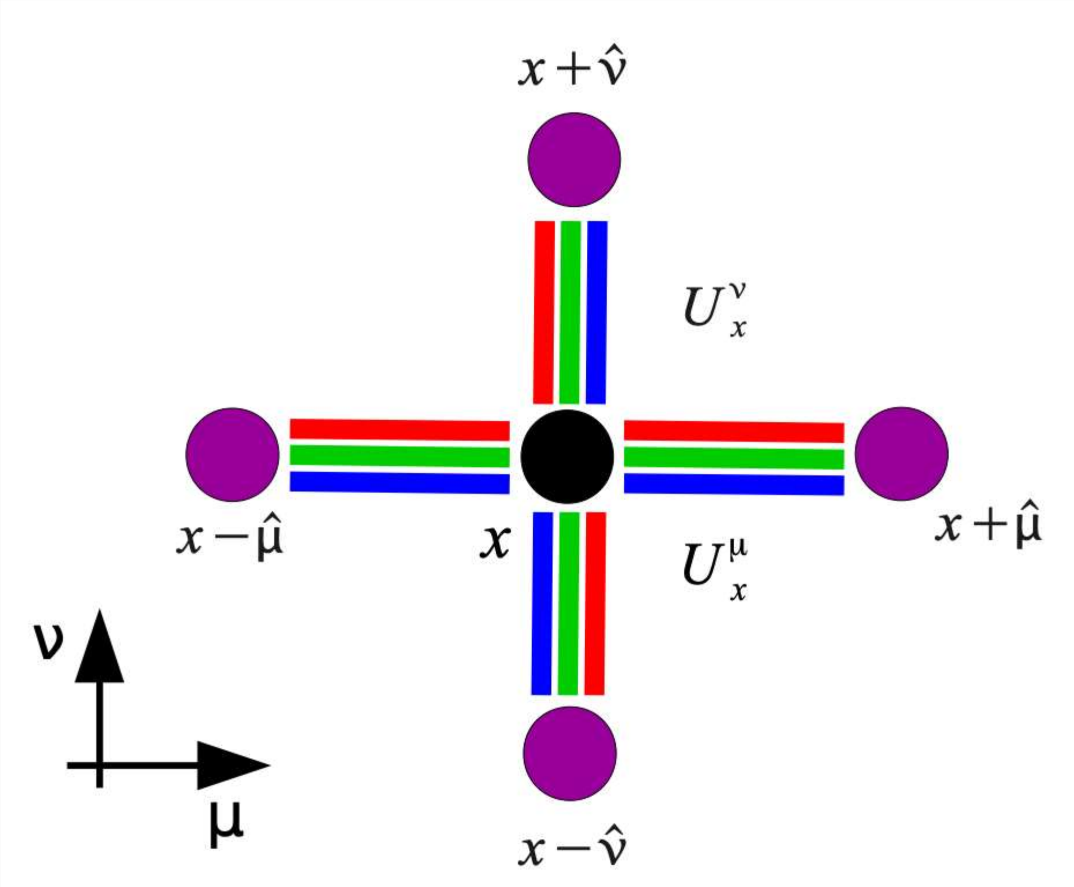
Parallelism, parallelism, parallelism...

- Multigrid setup
  - Block orthogonalization of null space vectors
  - Batched QR decomposition

# Ingredients for Parallel Adaptive Multigrid

Parallelism, parallelism, parallelism…

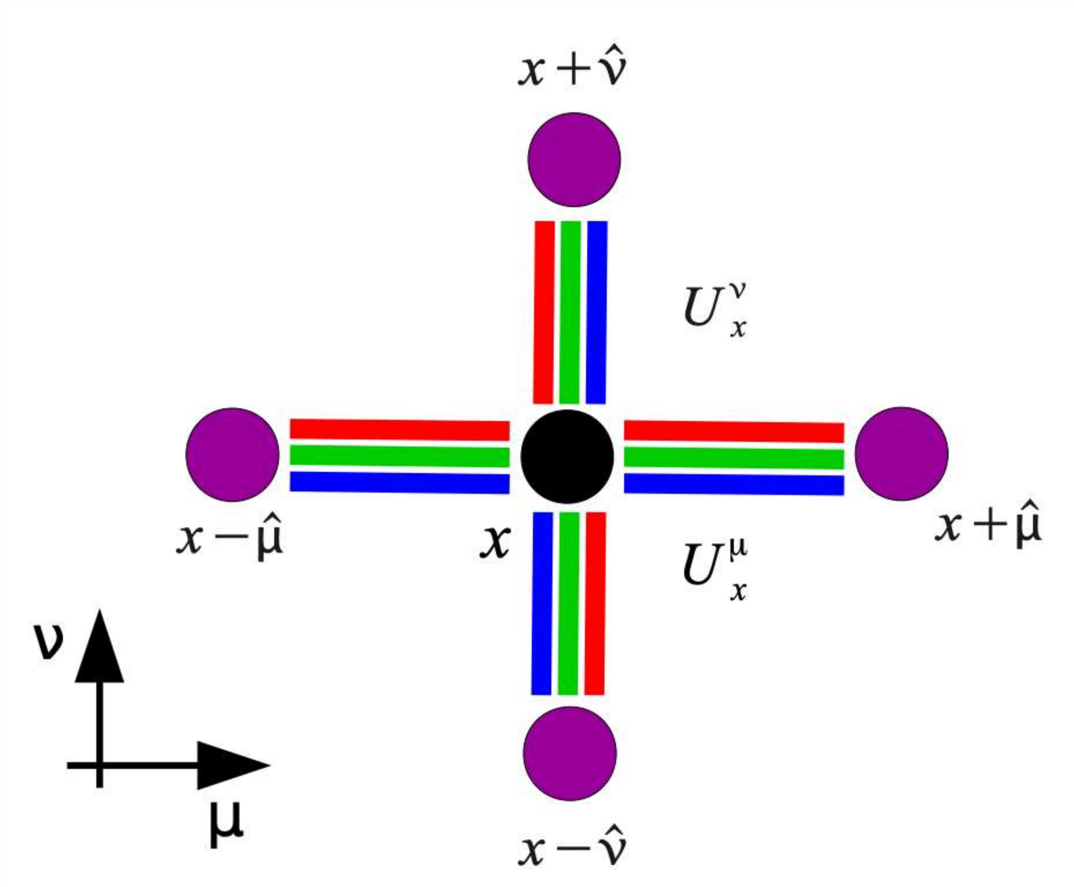
- Multigrid setup
  - Block orthogonalization of null space vectors
  - Batched QR decomposition
- Smoothing (relaxation on a given grid)
  - Repurpose existing solvers

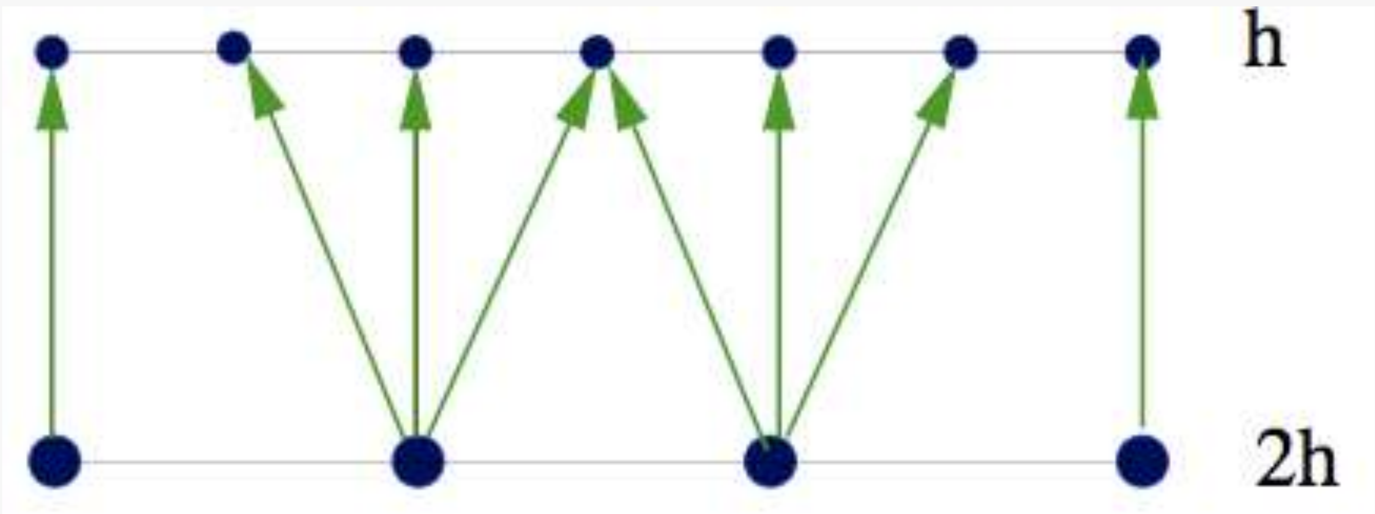


# Ingredients for Parallel Adaptive Multigrid

Parallelism, parallelism, parallelism…

- Multigrid setup
  - Block orthogonalization of null space vectors
  - Batched QR decomposition
- Smoothing (relaxation on a given grid)
  - Repurpose existing solvers
- Prolongation
  - interpolation from coarse grid to fine grid
  - one-to-many mapping

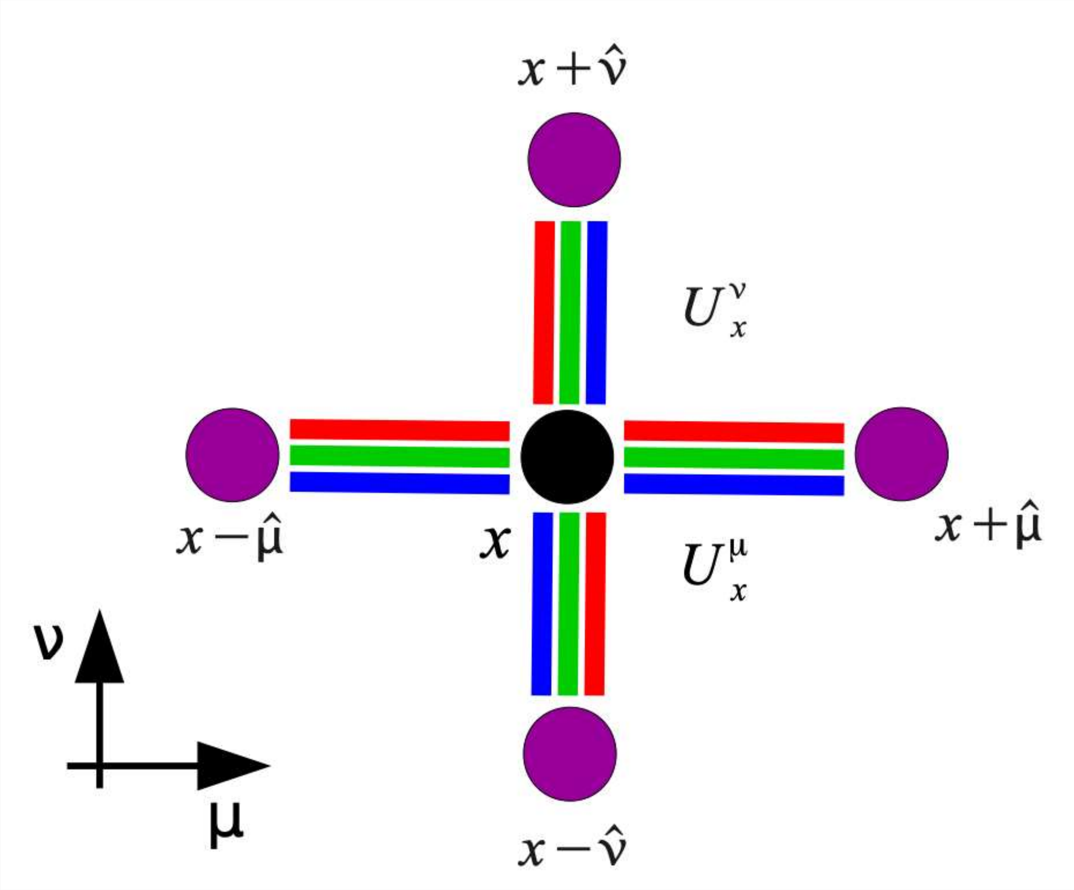


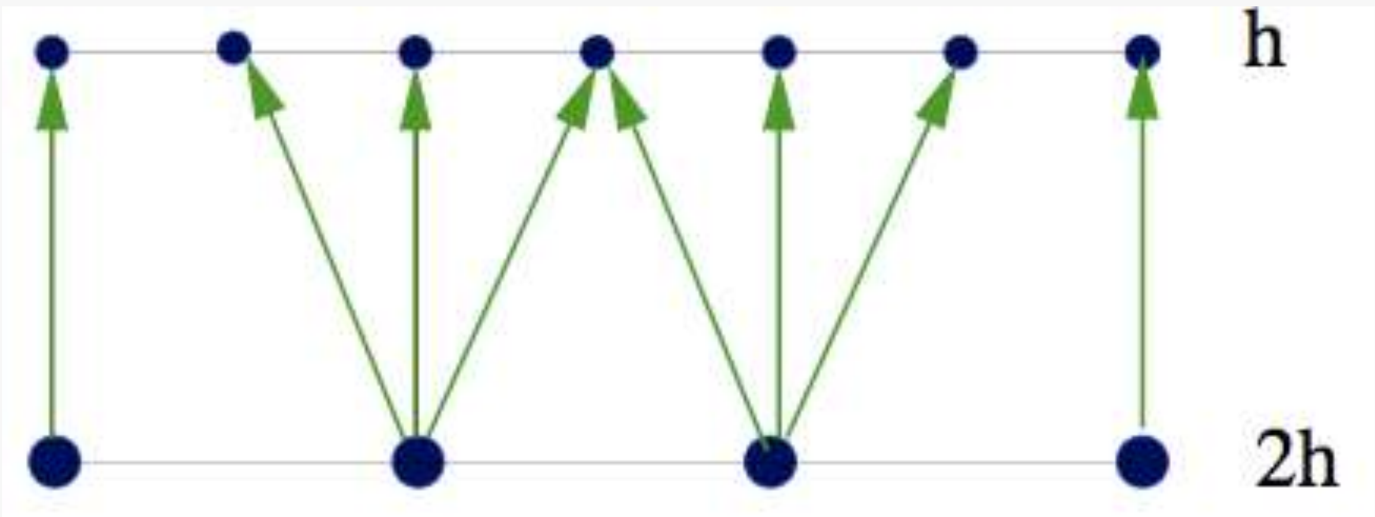



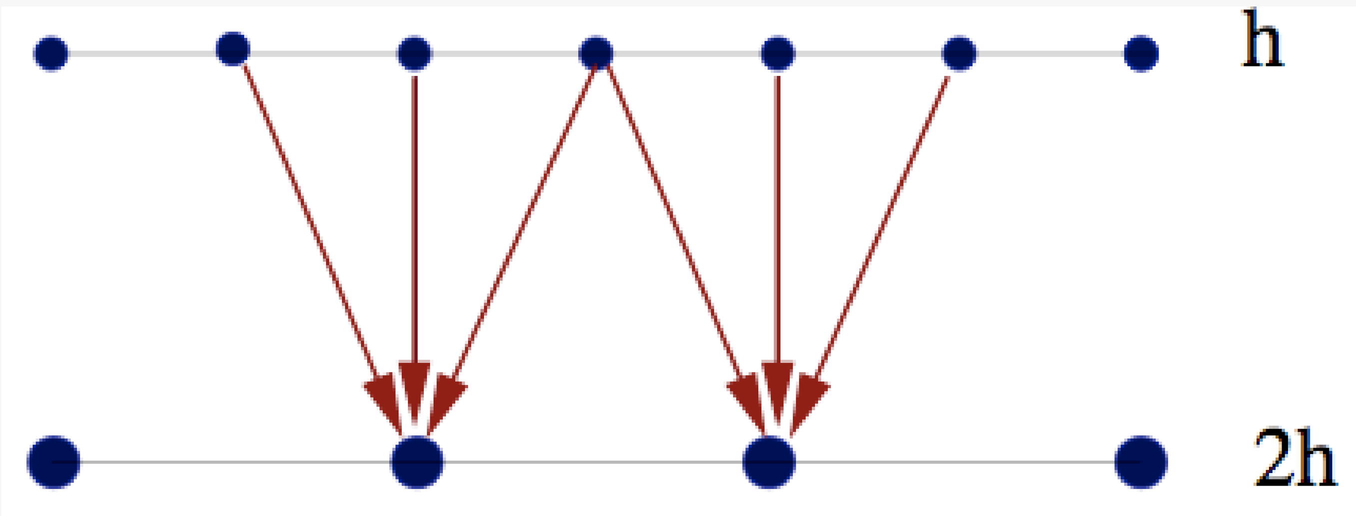
# Ingredients for Parallel Adaptive Multigrid

## Parallelism, parallelism, parallelism…

- Multigrid setup
  - Block orthogonalization of null space vectors
  - Batched QR decomposition
- Smoothing (relaxation on a given grid)
  - Repurpose existing solvers
- Prolongation
  - interpolation from coarse grid to fine grid
  - one-to-many mapping
- Restriction
  - restriction from fine grid to coarse grid
  - many-to-one mapping



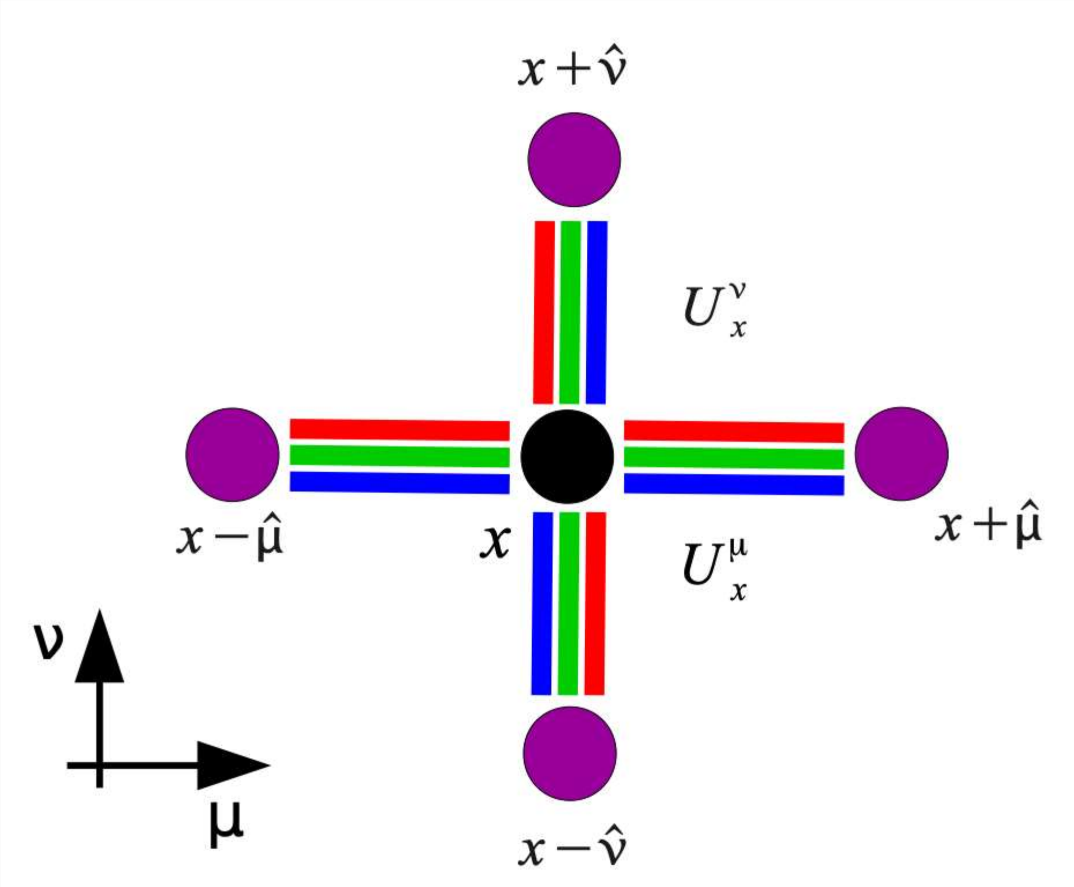



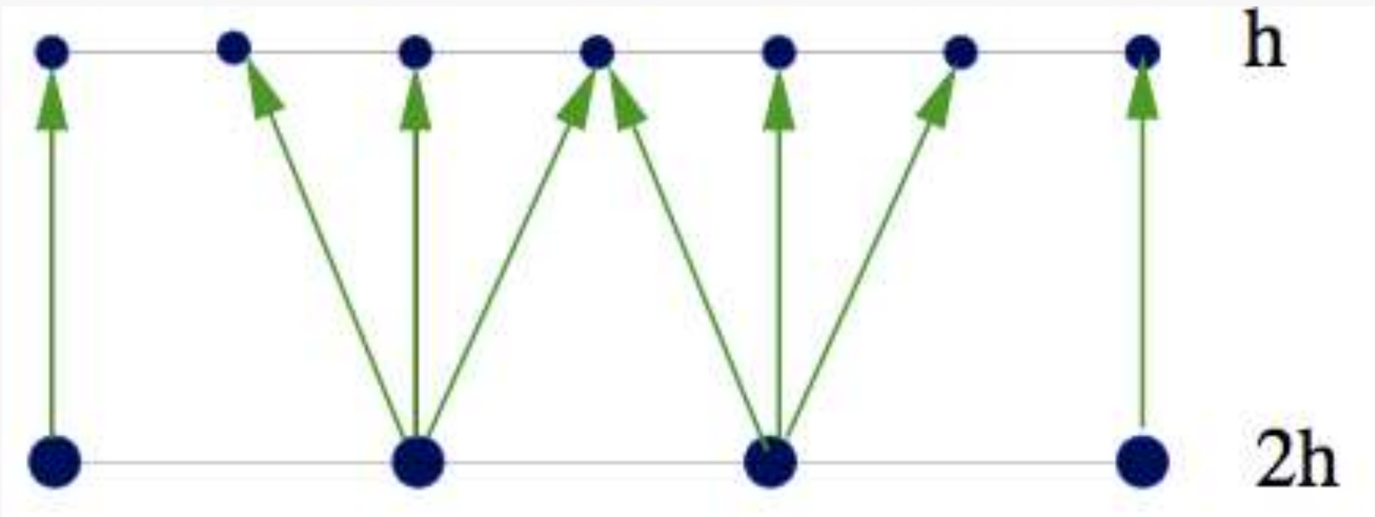



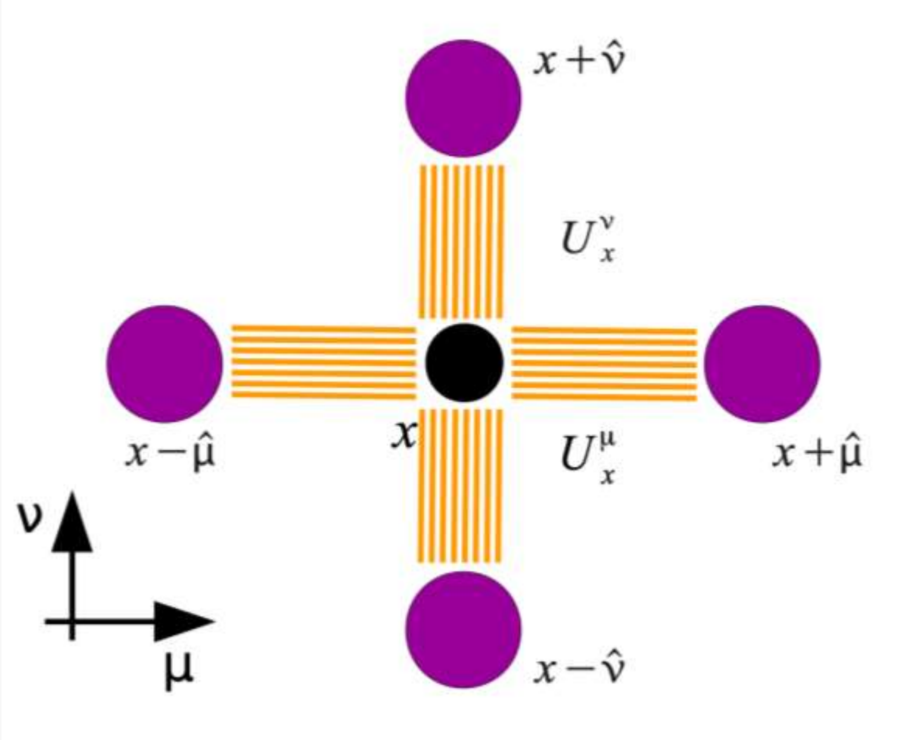
# Ingredients for Parallel Adaptive Multigrid

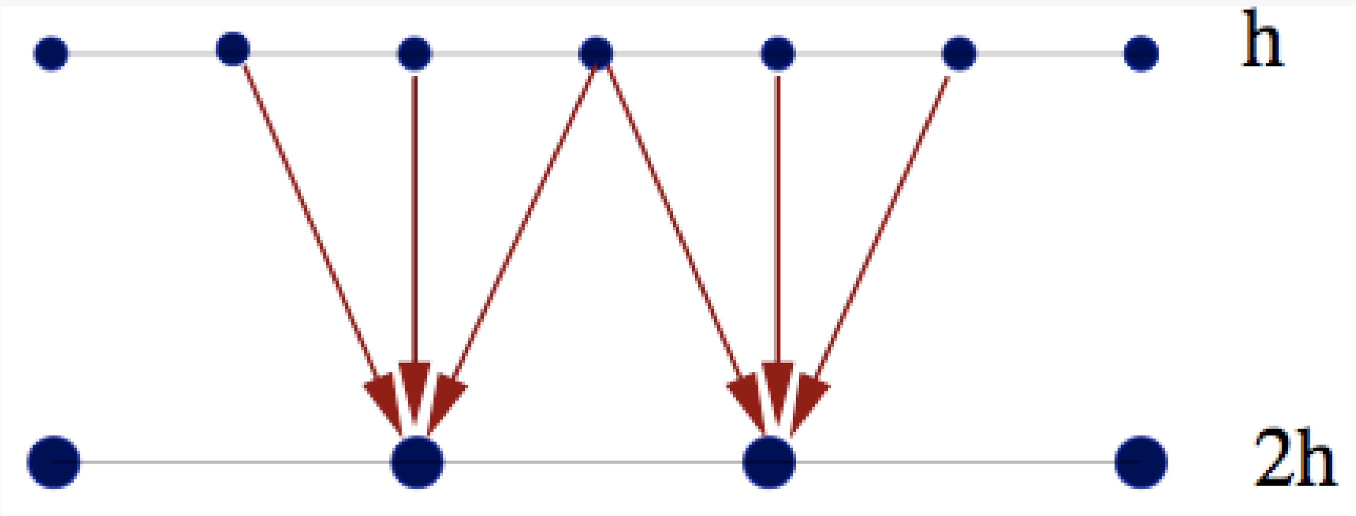
Parallelism, parallelism, parallelism…

- Multigrid setup
  - Block orthogonalization of null space vectors
  - Batched QR decomposition
- Smoothing (relaxation on a given grid)
  - Repurpose existing solvers
- Prolongation
  - interpolation from coarse grid to fine grid
  - one-to-many mapping
- Restriction
  - restriction from fine grid to coarse grid
  - many-to-one mapping
- Coarse Operator construction (setup)
  - Evaluate $P^{\dagger}D\ P$ locally
  - Batched (small) dense matrix multiplication



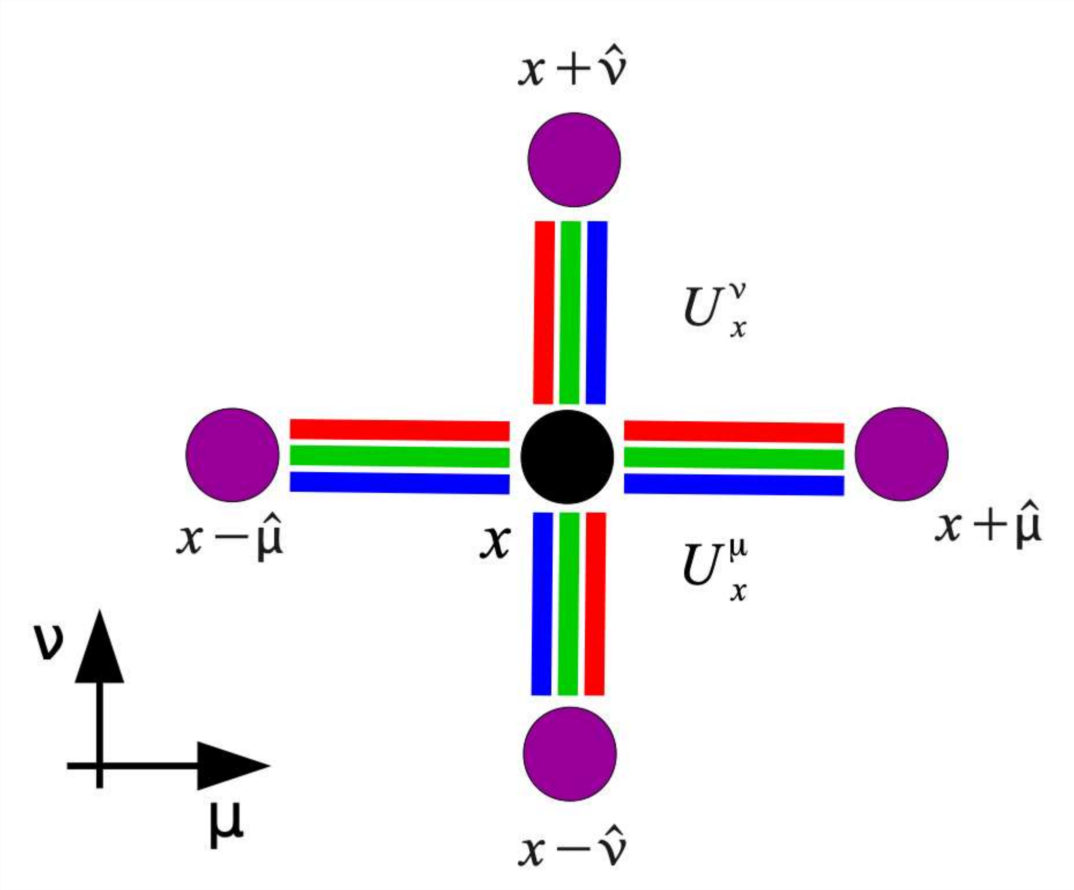



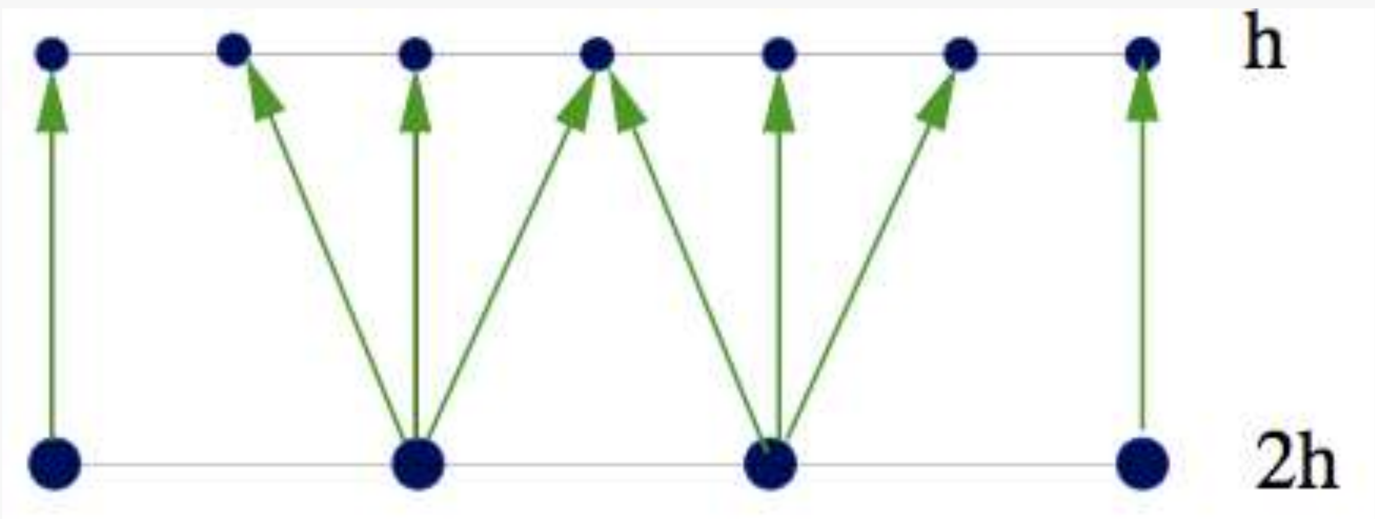



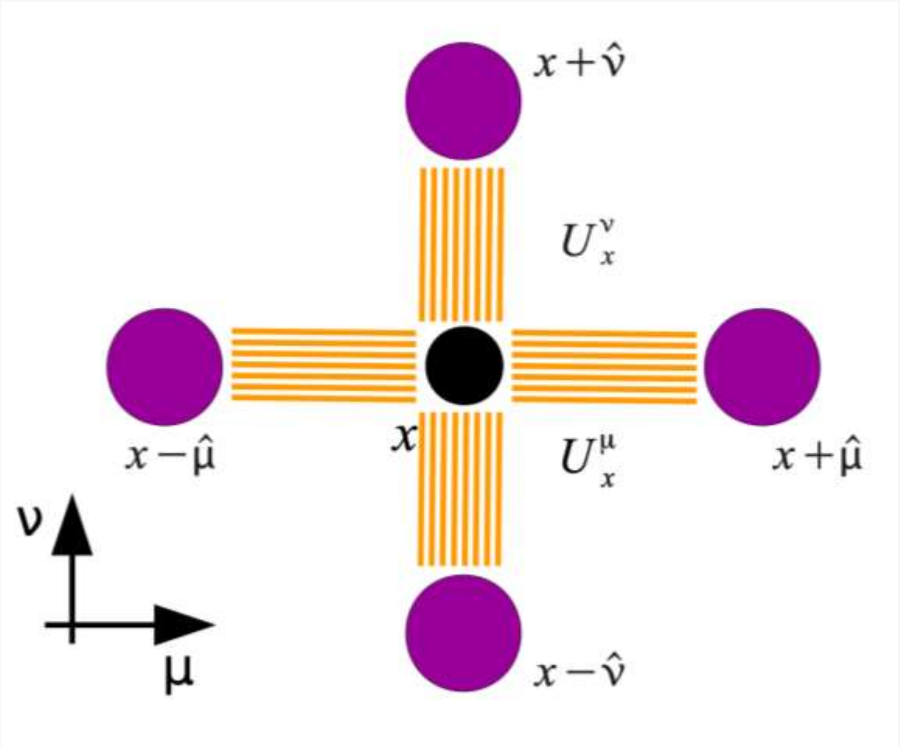
# Ingredients for Parallel Adaptive Multigrid

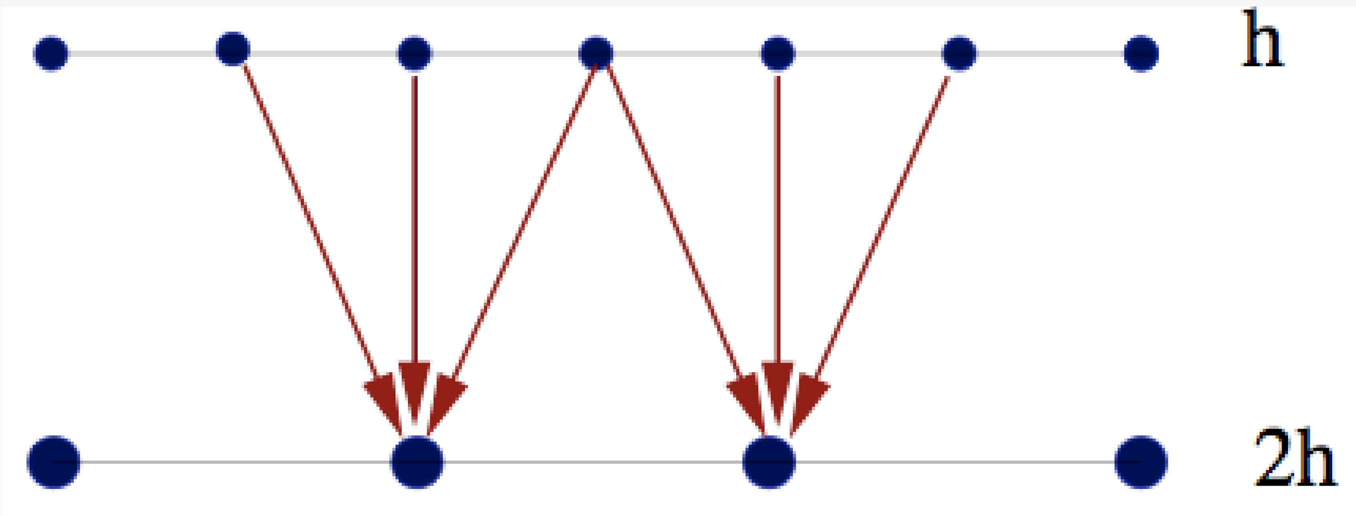
Parallelism, parallelism, parallelism...

- Multigrid setup
  - Block orthogonalization of null space vectors
  - Batched QR decomposition
- Smoothing (relaxation on a given grid)
  - Repurpose existing solvers
- Prolongation
  - interpolation from coarse grid to fine grid
  - one-to-many mapping
- Restriction
  - restriction from fine grid to coarse grid
  - many-to-one mapping
- Coarse Operator construction (setup)
  - Evaluate $P^{\dagger}D\ P$ locally
  - Batched (small) dense matrix multiplication
- Coarse grid solver
  - Need optimal coarse-grid operator



# Coarse Grid Operator

- Coarse operator looks like a Dirac operator (many more colors)
  - Link matrices have dimension $2N_v$ x $2N_v$ (e.g., 48 x 48)

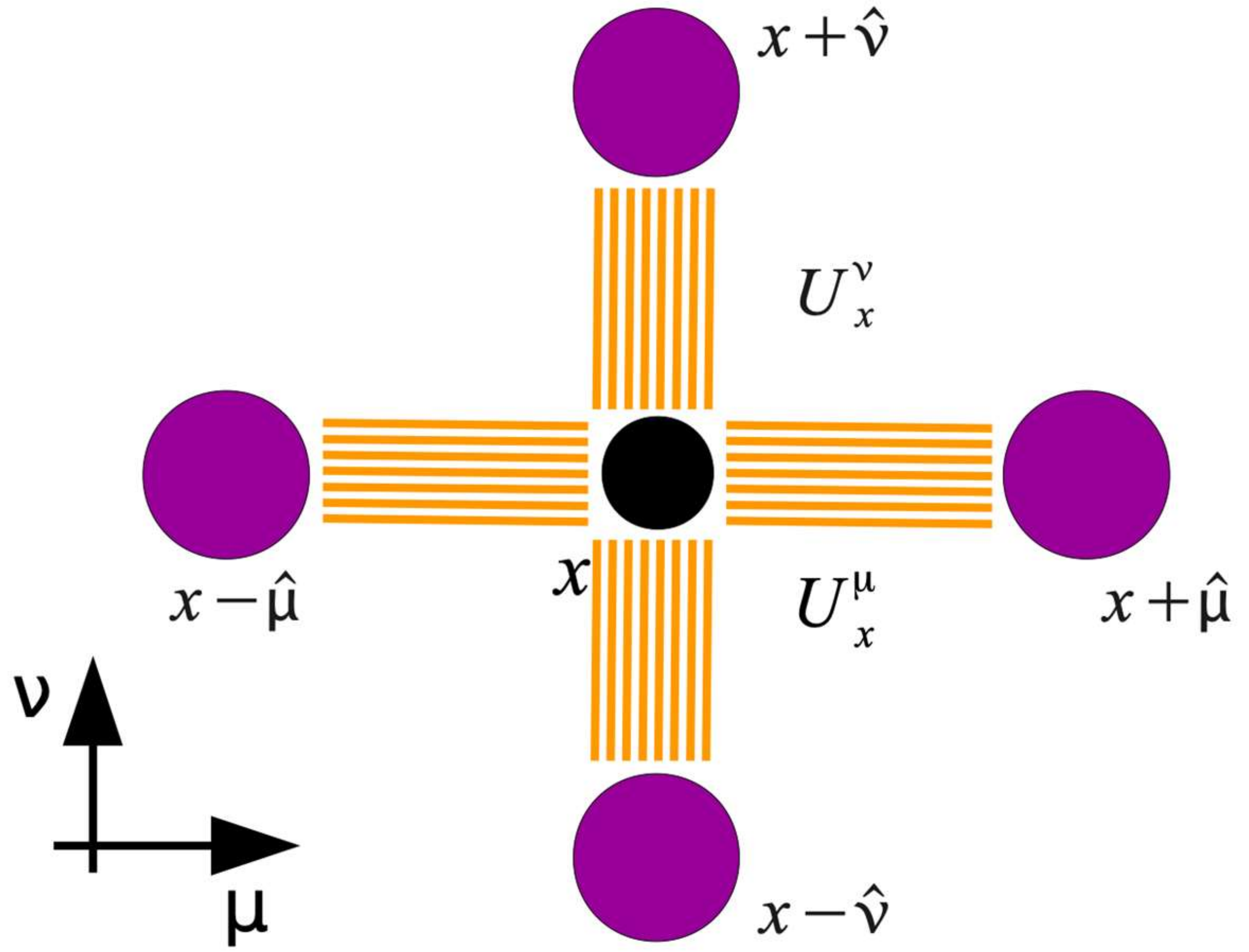
$$\hat{D}_{\mathbf{i}\hat{s}\hat{c},\mathbf{j}\hat{s}'\hat{c}'} = -\sum_{\mu} \left[ Y^{-\mu}_{\mathbf{i}\hat{s}\hat{c},\mathbf{j}\hat{s}'\hat{c}'} \delta_{\mathbf{i}+\mu,\mathbf{j}} + Y^{+\mu\dagger}_{\mathbf{i}s\hat{c},\mathbf{j}s'\hat{c}'} \delta_{\mathbf{i}-\mu,\mathbf{j}} \right] + \left( M - X_{\mathbf{i}\hat{s}\hat{c},\mathbf{j}\hat{s}'\hat{c}'} \right) \delta_{\mathbf{i}\hat{s}\hat{c},\mathbf{j}\hat{s}'\hat{c}'} .$$

# Coarse Grid Operator

- Coarse operator looks like a Dirac operator (many more colors)
  - Link matrices have dimension $2N_v \times 2N_v$ (e.g., 48 x 48)

$$\hat{D}_{\mathbf{i}\hat{s}\hat{c},\mathbf{j}\hat{s}'\hat{c}'} = -\sum_{\mu}\left[Y^{-\mu}_{\mathbf{i}\hat{s}\hat{c},\mathbf{j}\hat{s}'\hat{c}'}\delta_{\mathbf{i}+\mu,\mathbf{j}} + Y^{+\mu\dagger}_{\mathbf{i}\hat{s}\hat{c},\mathbf{j}\hat{s}'\hat{c}'}\delta_{\mathbf{i}-\mu,\mathbf{j}}\right] + \left(M - X_{\mathbf{i}\hat{s}\hat{c},\mathbf{j}\hat{s}'\hat{c}'}\right)\delta_{\mathbf{i}\hat{s}\hat{c},\mathbf{j}\hat{s}'\hat{c}'}\,.$$

- Fine vs. Coarse grid parallelization
  - Fine grid operator has plenty of grid-level parallelism
    - E.g., 16x16x16x16 = 65536 lattice sites
  - Coarse grid operator has diminishing grid-level parallelism
    - first coarse grid 4x4x4x4 = 256 lattice sites
    - second coarse grid 2x2x2x2 = 16 lattice sites

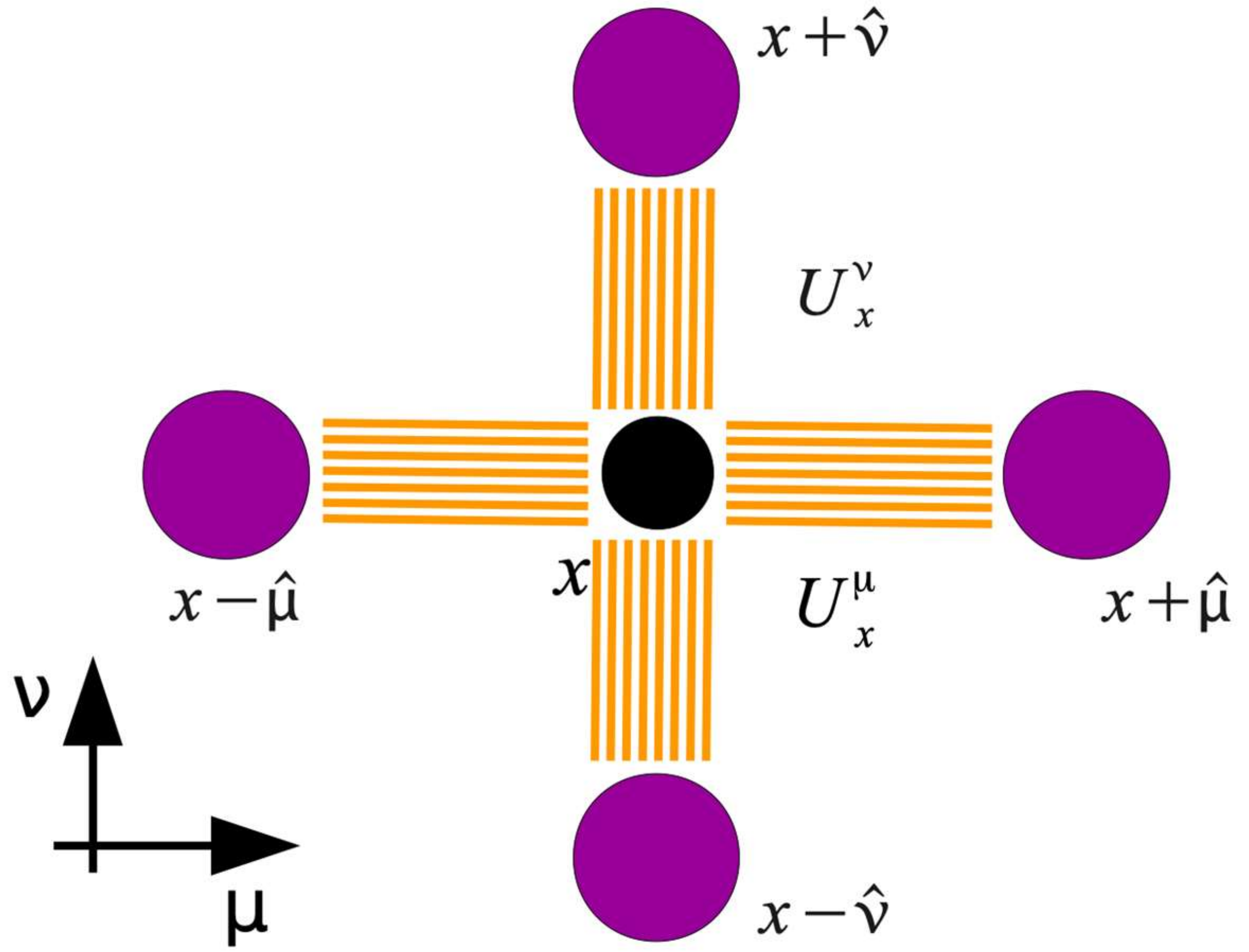


# Coarse Grid Operator

- Coarse operator looks like a Dirac operator (many more colors)
  - Link matrices have dimension $2N_v$ x $2N_v$ (e.g., 48 x 48)

$$\hat{D}_{\mathbf{i}\hat{s}\hat{c},\mathbf{j}\hat{s}'\hat{c}'} = -\sum_{\mu}\left[Y^{-\mu}_{\mathbf{i}\hat{s}\hat{c},\mathbf{j}\hat{s}'\hat{c}'}\delta_{\mathbf{i}+\mu,\mathbf{j}} + Y^{+\mu\dagger}_{\mathbf{i}\hat{s}\hat{c},\mathbf{j}\hat{s}'\hat{c}'}\delta_{\mathbf{i}-\mu,\mathbf{j}}\right] + \left(M - X_{\mathbf{i}\hat{s}\hat{c},\mathbf{j}\hat{s}'\hat{c}'}\right)\delta_{\mathbf{i}\hat{s}\hat{c},\mathbf{j}\hat{s}'\hat{c}'}.$$

- Fine vs. Coarse grid parallelization
  - Fine grid operator has plenty of grid-level parallelism
    - E.g., 16x16x16x16 = 65536 lattice sites
  - Coarse grid operator has diminishing grid-level parallelism
    - first coarse grid 4x4x4x4 = 256 lattice sites
    - second coarse grid 2x2x2x2 = 16 lattice sites
- Need to consider finer-grained parallelization
  - Increase parallelism to use all GPU resources
  - Load balancing



# Sources of Parallelism

- Matrix-*Vector* parallelism
  - Splitting up the constituent dot products is a source of reuse

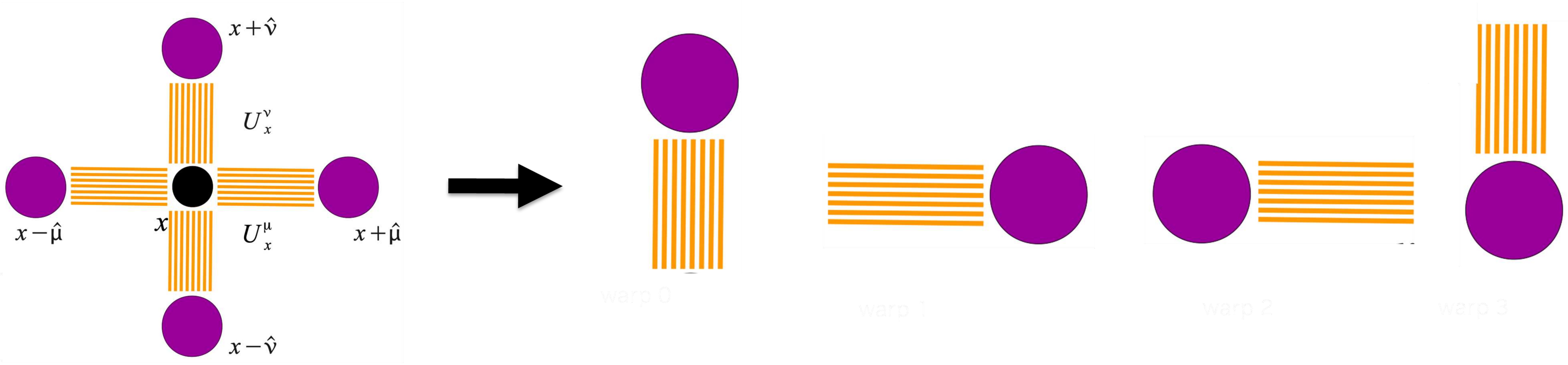
$$\downarrow \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} += \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

# Sources of Parallelism

- Matrix-*Vector* parallelism
  - Splitting up the constituent dot products is a source of reuse

$$\downarrow \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} += \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

- Direction parallelism
  - Note: the input coarse spinor is a source of directional cache reuse

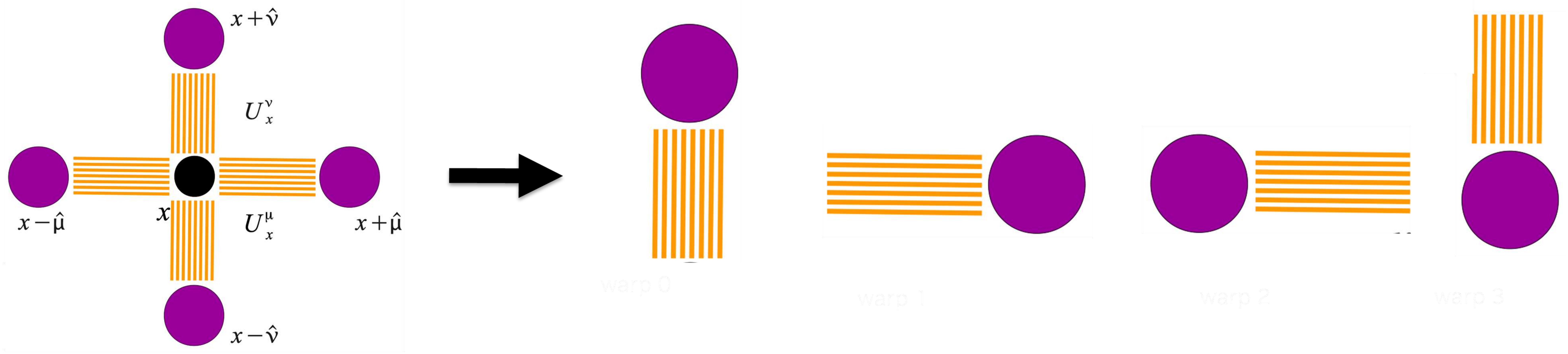


# Sources of Parallelism

- Matrix-*Vector* parallelism
  - Splitting up the constituent dot products is a source of reuse

$$\begin{pmatrix}c_0\\c_1\\c_2\\c_3\end{pmatrix} \mathrel{+}= \begin{pmatrix}a_{00} & a_{01} & a_{02} & a_{03}\\ a_{10} & a_{11} & a_{12} & a_{13}\\ a_{20} & a_{21} & a_{22} & a_{23}\\ a_{30} & a_{31} & a_{32} & a_{33}\end{pmatrix}\begin{pmatrix}b_0\\b_1\\b_2\\b_3\end{pmatrix}$$

- Direction parallelism
  - Note: the input coarse spinor is a source of directional cache reuse




- Dot-product parallelism:

$$\begin{pmatrix}a_{00} & a_{01} & a_{02} & a_{03}\end{pmatrix}\begin{pmatrix}b_0\\b_1\\b_2\\b_3\end{pmatrix} \Rightarrow \begin{pmatrix}a_{00} & a_{01}\end{pmatrix}\begin{pmatrix}b_0\\b_1\end{pmatrix} + \begin{pmatrix}a_{02} & a_{03}\end{pmatrix}\begin{pmatrix}b_2\\b_3\end{pmatrix}$$

# Twisted Clover Example

## The March of Optimization

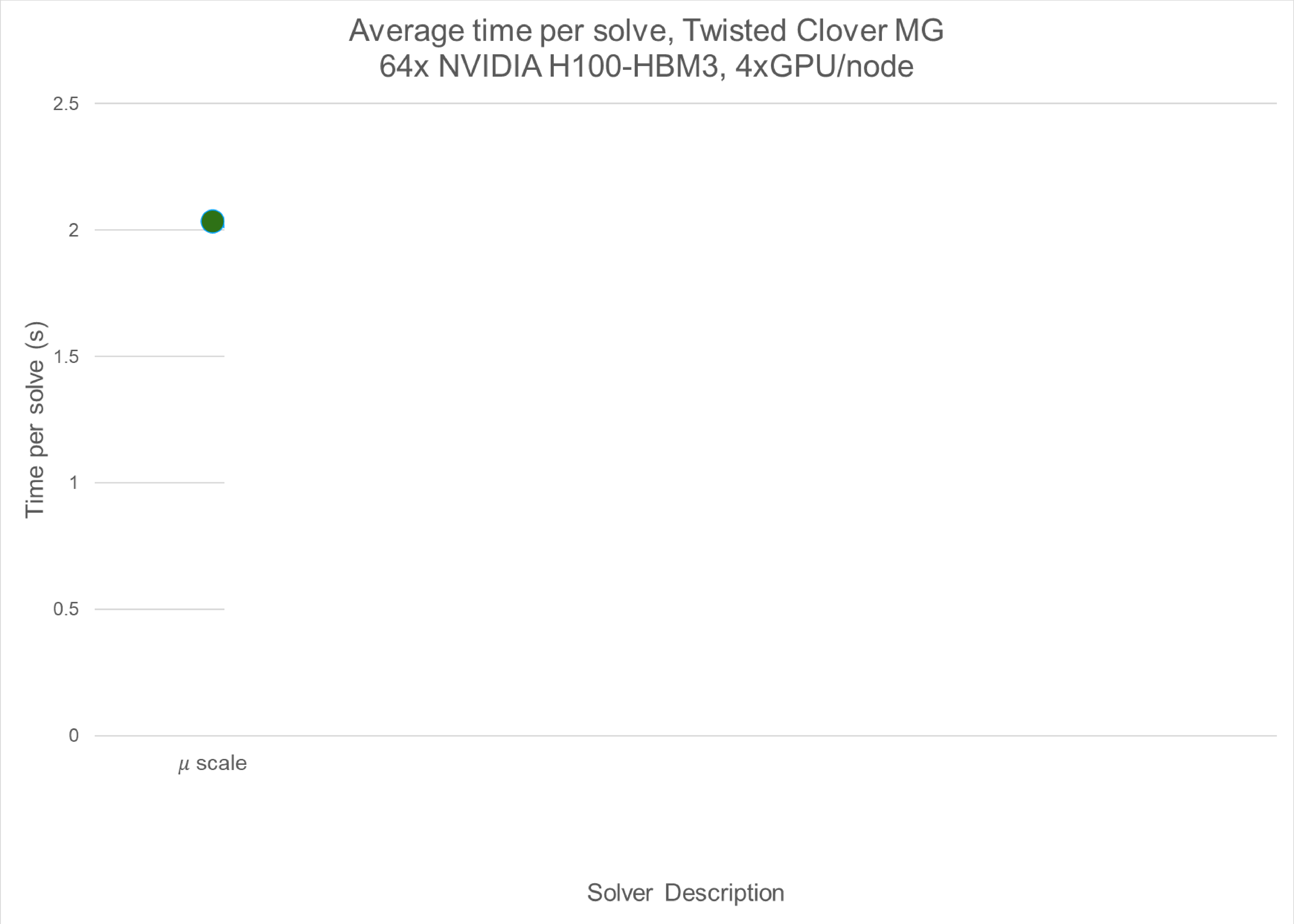
- Thank you to the ETMC collaboration for this configuration:
  - $64^3$x128 physical-point pion
  - Iwasaki gauge action, $\beta = 1.778$,
  - Physical pion twisted clover fermion action, $\kappa = 0.13947, \mu = 0.000720, c_{sw} = 1.69$

A modernization of https://github.com/lattice/quda/wiki/Twisted-clover-deflated-multigrid from Dean Howarth

# Twisted Clover Example

## The March of Optimization

- Thank you to the ETMC collaboration for this configuration:
  - $64^3$x128 physical-point pion
  - Iwasaki gauge action, $\beta = 1.778$,
  - Physical pion twisted clover fermion action, $\kappa = 0.13947, \mu = 0.000720, c_{sw} = 1.69$
- The starting point: 3-level multigrid
  - Aggregate 1: $4^4$ to $16^3$x32 volume, $N_c = 24, N_s = 2$
  - Aggregate 2: $2^4$ to $8^3$x16 volume, $N_c = 24, N_s = 2$
  - Coarsest level "$\mu$" enhancement: 70
  - Preconditioned solver: GCR
  - Smoother: GCR(0,4)
  - Coarsest-level solver: GCR

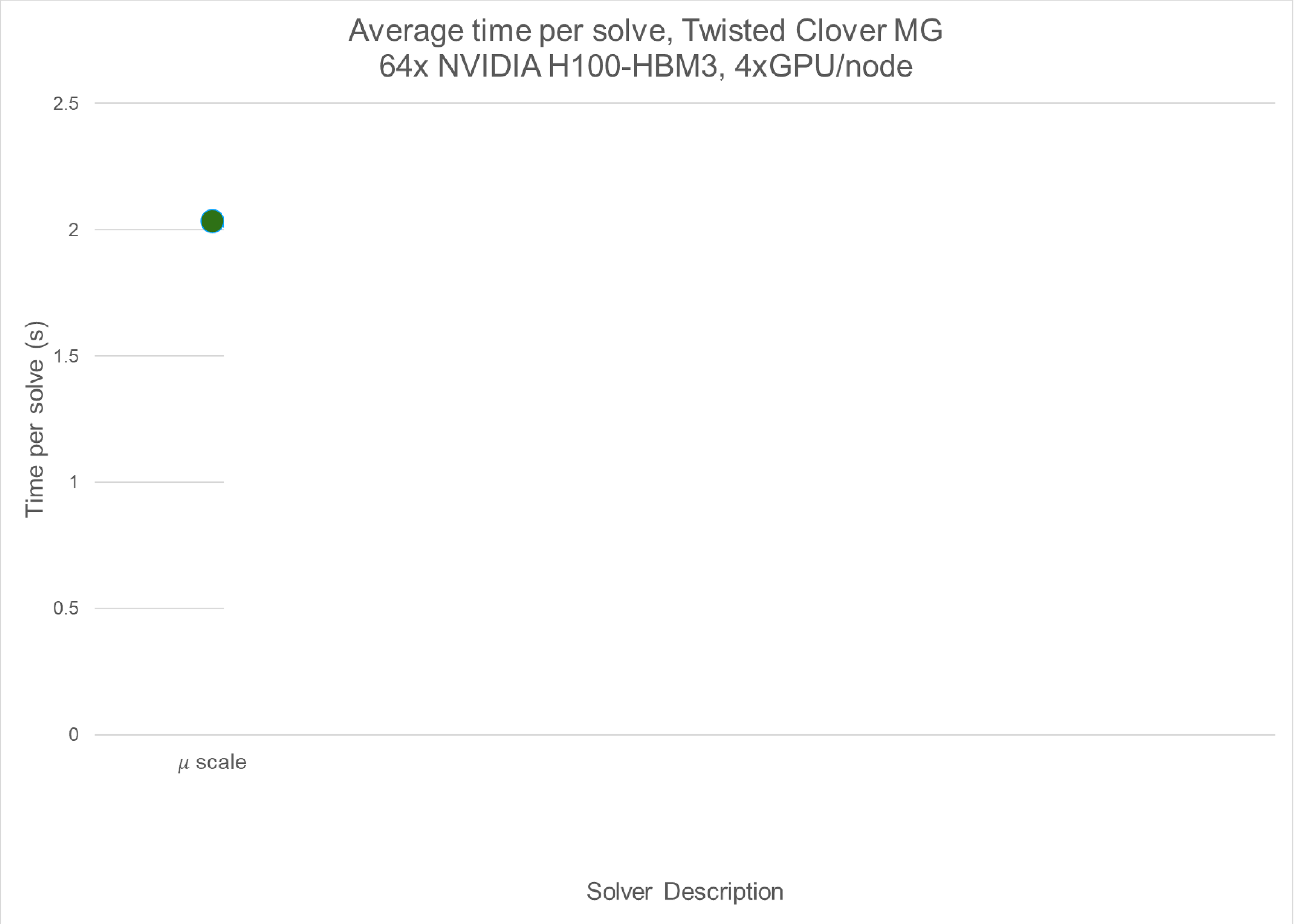



A modernization of https://github.com/lattice/quda/wiki/Twisted-clover-deflated-multigrid from Dean Howarth

# Twisted Clover Example

## Communication-avoiding solvers

- We'll switch to communication-avoiding solvers for the smoothers and coarsest-level solver
  - CA-GCR, based on CA-CG from https://research.nvidia.com/sites/default/files/pubs/2016-04_S-Step-and-Communication-Avoiding/nvr-2016-003.pdf
  - Generate $D\vec{x}, D^2\vec{x}, D^3\vec{x}, \ldots$ minimize the residual in one batched go
  - Gram-Schmidt instead of modified Gram-Schmidt

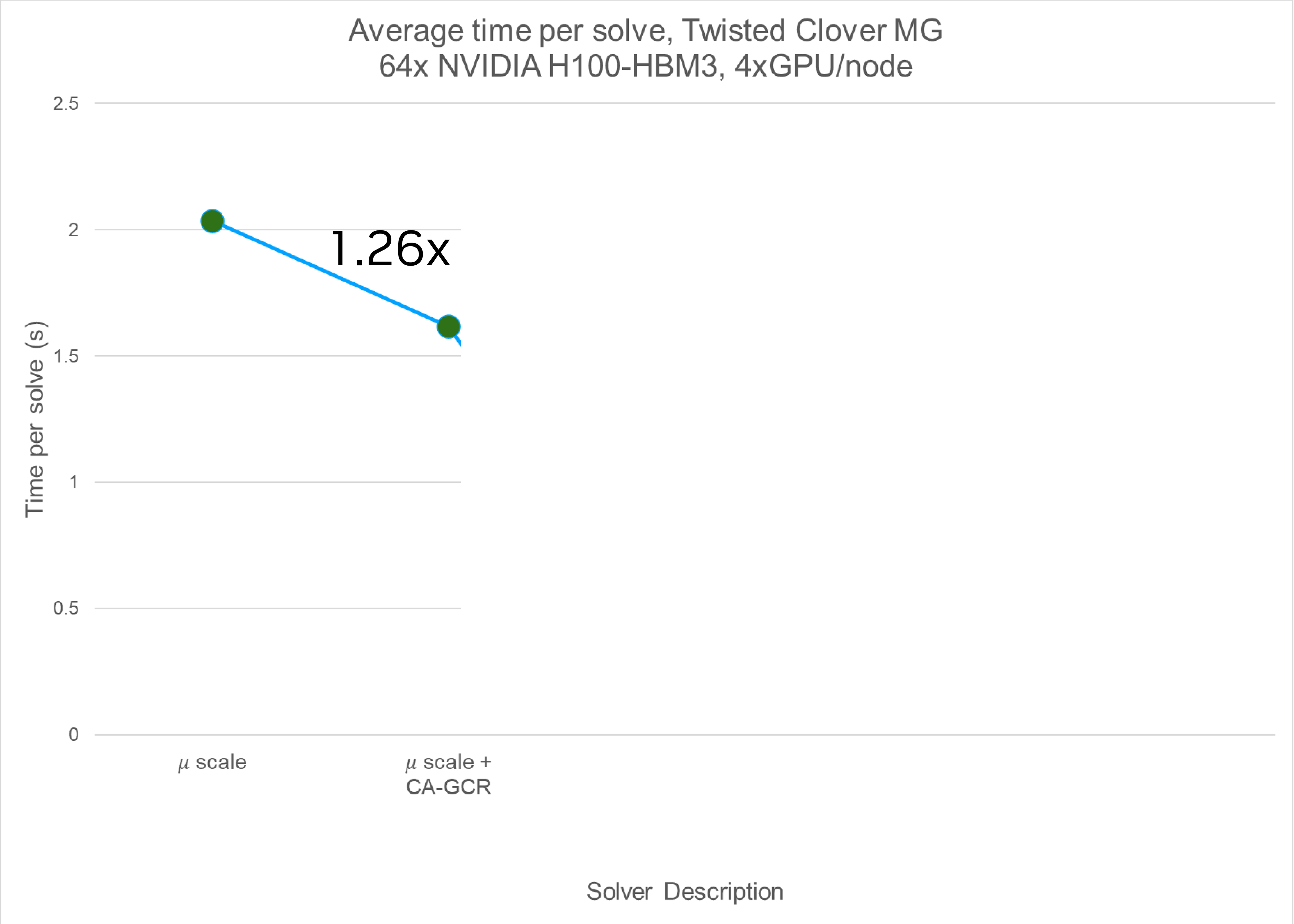



A modernization of https://github.com/lattice/quda/wiki/Twisted-clover-deflated-multigrid from Dean Howarth

# Twisted Clover Example

## Communication-avoiding solvers

- We'll switch to communication-avoiding solvers for the smoothers and coarsest-level solver
  - CA-GCR, based on CA-CG from https://research.nvidia.com/sites/default/files/pubs/2016-04_S-Step-and-Communication-Avoiding/nvr-2016-003.pdf
  - Generate $D\vec{x}, D^2\vec{x}, D^3\vec{x}, \ldots$ minimize the residual in one batched go
  - Gram-Schmidt instead of modified Gram-Schmidt
- New setup:
  - Aggregate 1: $4^4$ to $16^3$x32 volume, $N_c = 24, N_s = 2$
  - Aggregate 2: $2^4$ to $8^3$x16 volume, $N_c = 24, N_s = 2$
  - Coarsest level "$\mu$" enhancement: 70
  - Preconditioned solver: GCR
  - Smoother: CA-GCR(0,4)
  - Coarsest-level solver: CA-GCR

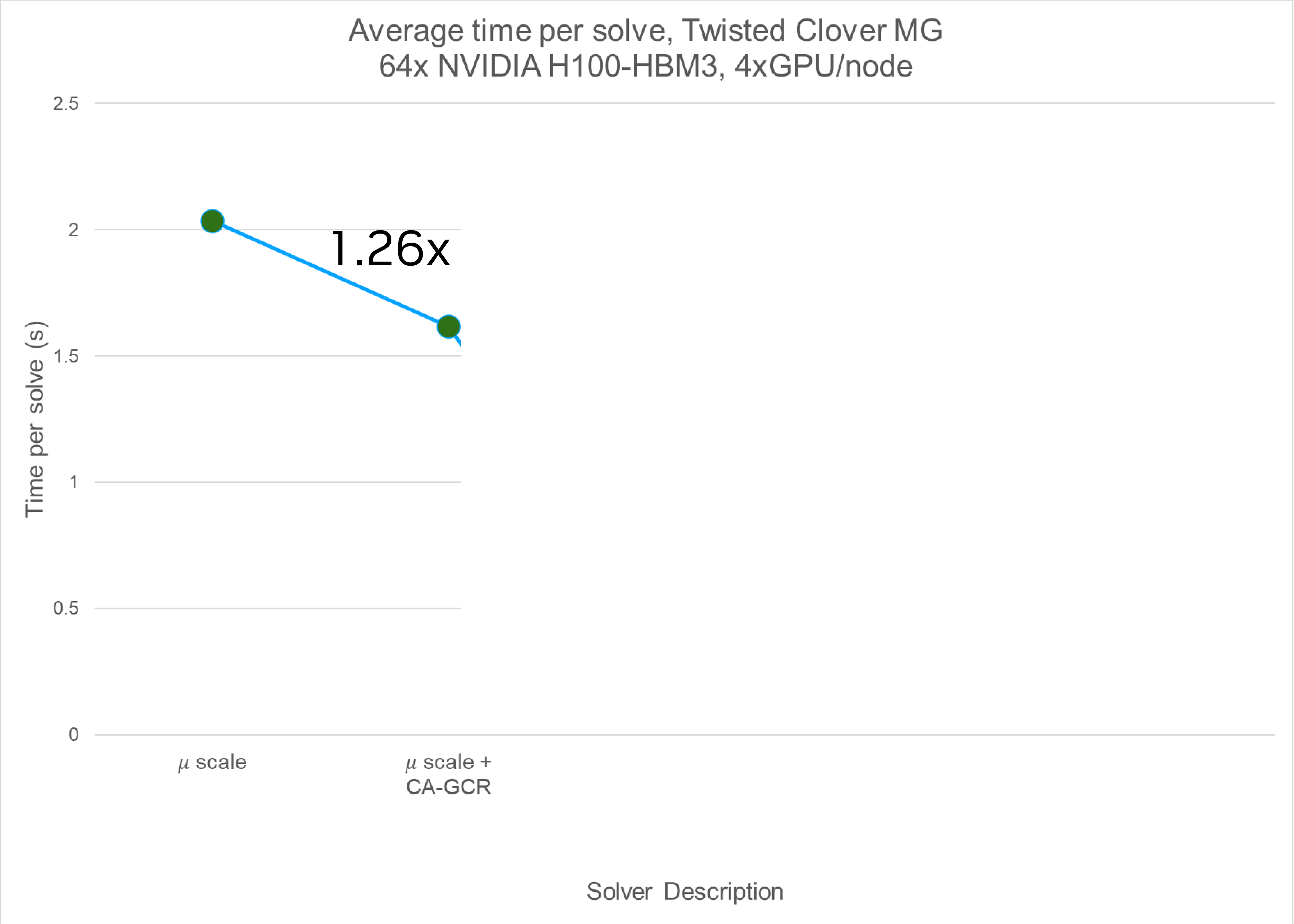



A modernization of https://github.com/lattice/quda/wiki/Twisted-clover-deflated-multigrid from Dean Howarth

# Twisted Clover Example

## Coarsest-level SVD deflation

- Last, we'll *deflate* the coarsest level instead of using a "mu" enhancement
  - Singular value deflation---a generalization of eigenvalue deflation
  - Work by Dean Howarth

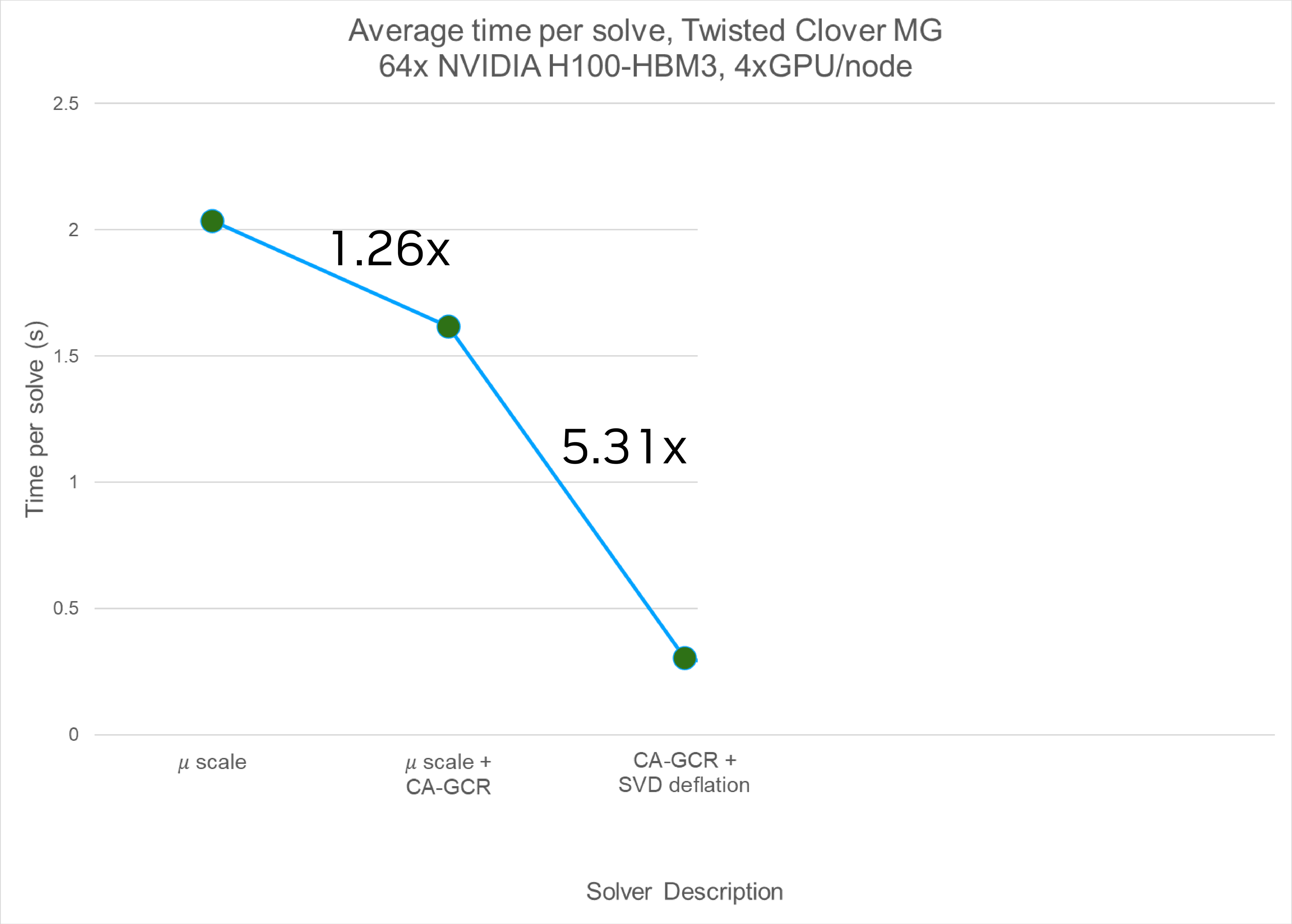



A modernization of https://github.com/lattice/quda/wiki/Twisted-clover-deflated-multigrid from Dean Howarth

# Twisted Clover Example

## Coarsest-level SVD deflation

- Last, we'll *deflate* the coarsest level instead of using a "mu" enhancement
  - Singular value deflation---a generalization of eigenvalue deflation
  - Work by Dean Howarth
- New setup:
  - Aggregate 1: $4^4$ to $16^3$x32 volume, $N_c = 24, N_s = 2$
  - Aggregate 2: $2^4$ to $8^3$x16 volume, $N_c = 24, N_s = 2$
  - No "$\mu$" enhancement
  - Preconditioned solver: GCR
  - Smoother: CA-GCR(0,4)
  - Coarsest-level solver: SVD-deflated CA-GCR
  - 1,024 deflation vectors




A modernization of https://github.com/lattice/quda/wiki/Twisted-clover-deflated-multigrid from Dean Howarth

# Multigrid on Modern Systems


# NVIDIA Grace Hopper Superchip

"super" - more than a "chip"
NVIDIA CPU + NVIDIA GPU w/o compromises

- **NVIDIA Grace CPU**
  - 72 Arm-v9 Neoverse V2 CPU cores with SVE2.
    - → Throughput: 3.6 TFLOP/s
  - Memory:
    - →High capacity: ≤ 480 GB LPDDR5X
    - →High System Memory bandwidth: ≤ 500 GB/s
- **NVIDIA Hopper GPU**
  - →High throughput: 60 TFLOP/s
  - Memory:
    - → Capacity: 96 GB HBM3 / 144 GB HBM3e
    - → Extreme bandwidth ≤ 4000 GB/s / 5000 GB/s
  - ≤ 18x NVLink 4 → 900 GB/s
  - → Threads are threads


# NVIDIA Grace Hopper Superchip

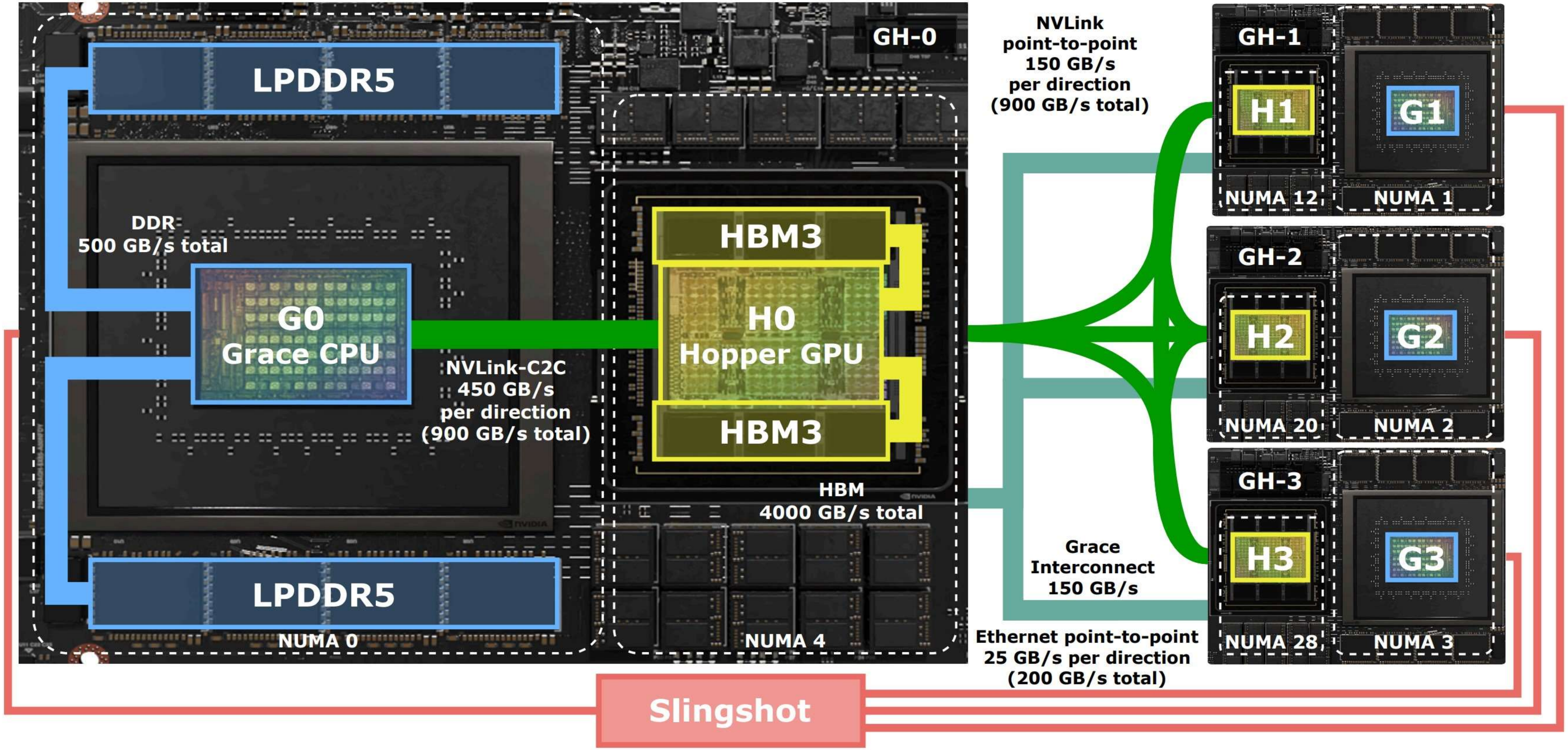
Soul is the new **NVLink-C2C** CPU ←→ GPU interconnect

- **Memory consistency**: ease of use
  - → *All* threads – GPU and CPU – access system memory: C++ `new`, `malloc`, `mmap`'ed files, atomics, …
  - → Fast automatic page migrations
  - → Threads cache peer memory → Less migrations
- **High-bandwidth**: 900 GB/s (same as peer NVLink 4)
  - → GPU reads or writes local/peer LPDDR5X at ~peak BW
- **Low-latency**: GPU→HBM latency
  - → GPU reads or writes LPDDR5X at ~HBM3 latency

For all threads in the system
**memory tastes like memory**
expected behavior + latency + bandwidth.

# Building up a Modern Node

4 x Grace-Hopper Superchips




Understanding Data Movement in Tightly Coupled Heterogeneous Systems: A Case Study with the Grace Hopper Superchip [ 2408.11556 (arxiv.org) ]

# Assemble it Into a Killer System: ALPS @ CSCS

A completely un-biased choice of a modern system totally not hand-picked for this workshop


### System Specification

#### Overview

| | |
|---|---|
| Model | HPE Cray EX |
| Interconnect | HPC Cray Slingshot-11 with 200 Gbps injection bandwidth per module / GPU |

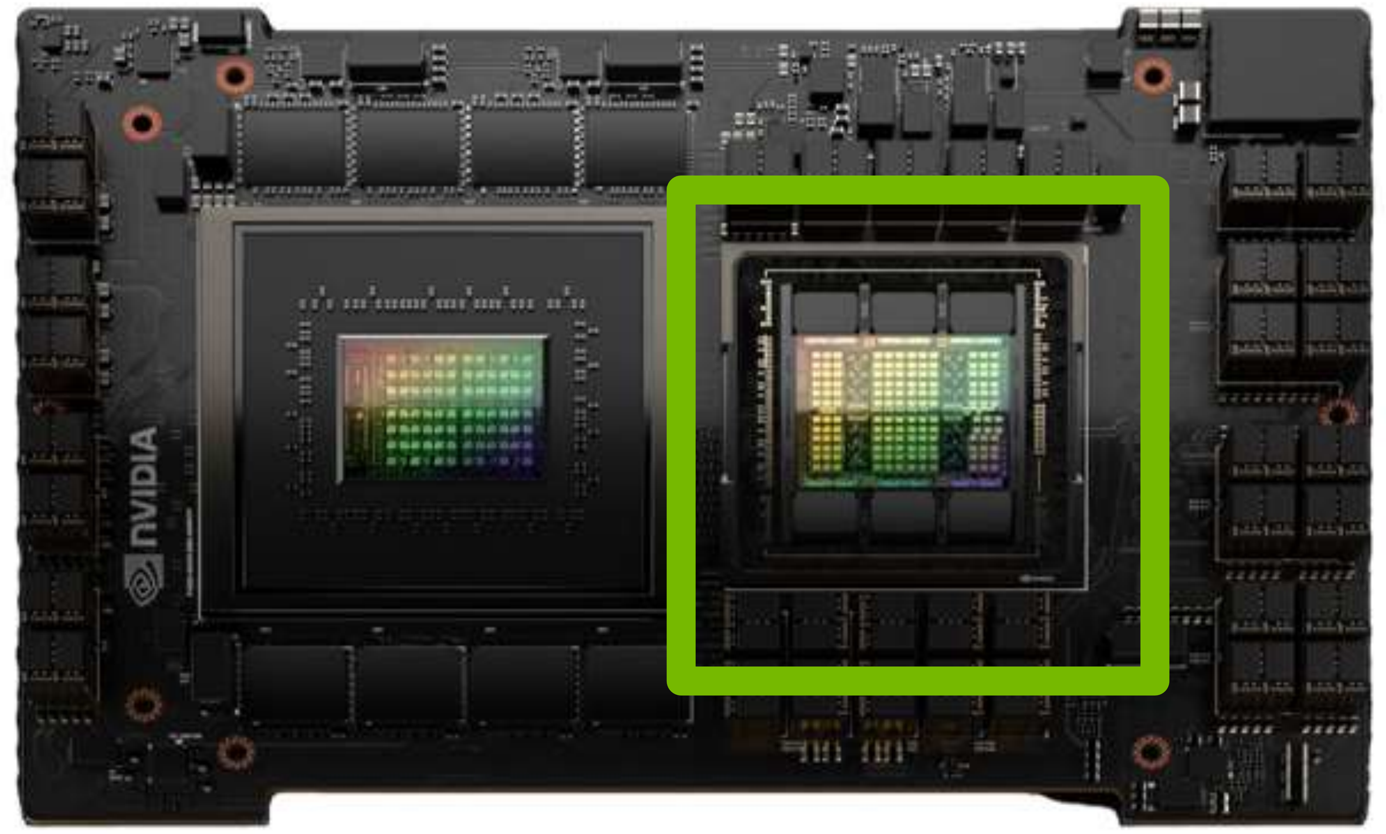
https://www.cscs.ch/computers/alps

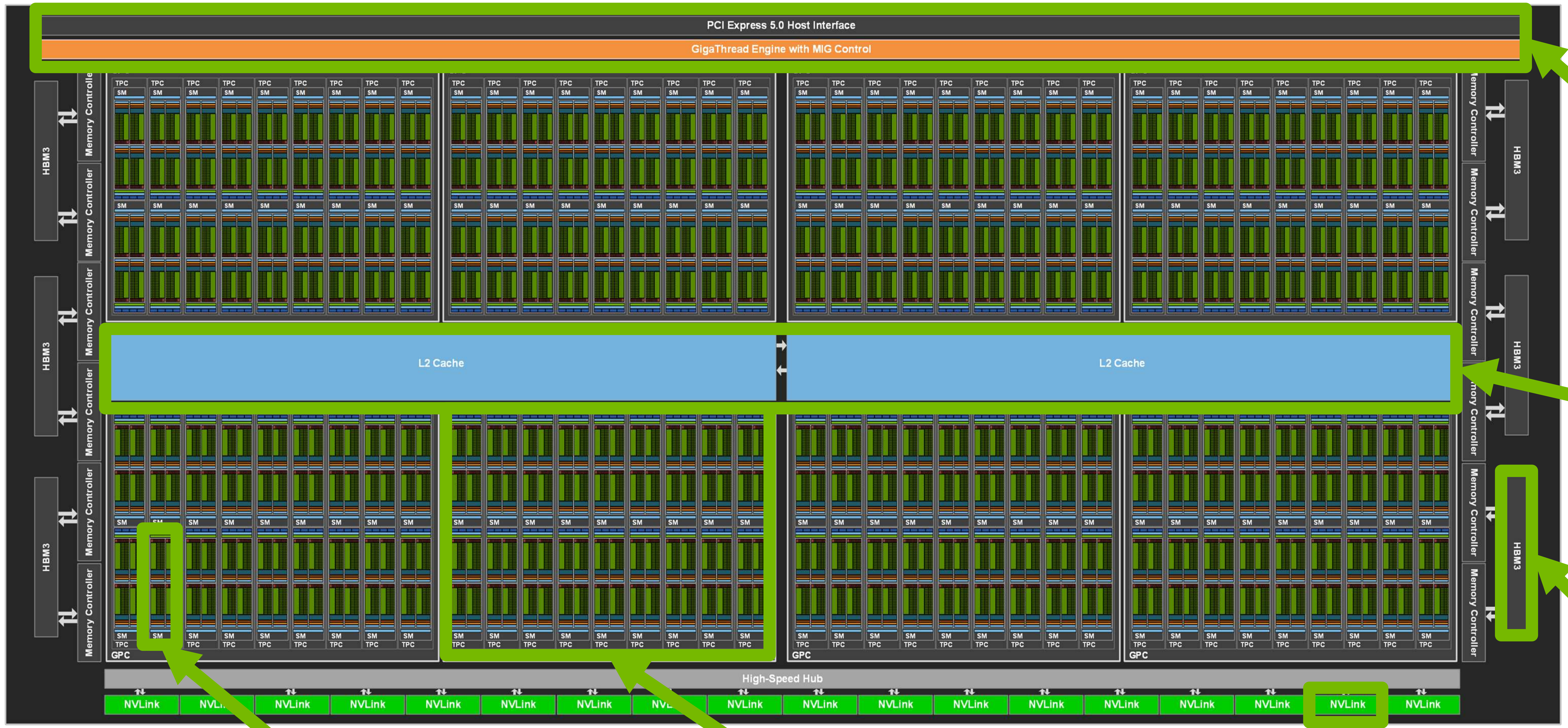
#### Nodes Overview

| # of nodes | # of sockets per node | Total # of sockets | Processor(s) | Specifications | TFlops |
|---|---|---|---|---|---|
| 2,688 | 4 | 10,752 | NVIDIA Grace-Hopper | 72 ARM cores, 128 GB LPDDR 5X RAM, H100 GPU with 96 GB HBM3 memory | n/a |

# Hopper GPU Architecture

A hierarchically-organized beast






2nd Gen Multi-Instance GPU
Confidential Computing
PCIe Gen5

Larger 60 MB L2

96GB HBM3, 4 TB/s bandwidth

132 SMs
4th Gen Tensor Core

GPU Processing Clusters (GPC)
"Thread Block Clusters"

4th Gen NVLink
900 GB/s total bandwidth

# If you can’t beat them, join them

## Tensor Cores

- An increasing proportion of GPU die area is spent on AI

# If you can't beat them, join them

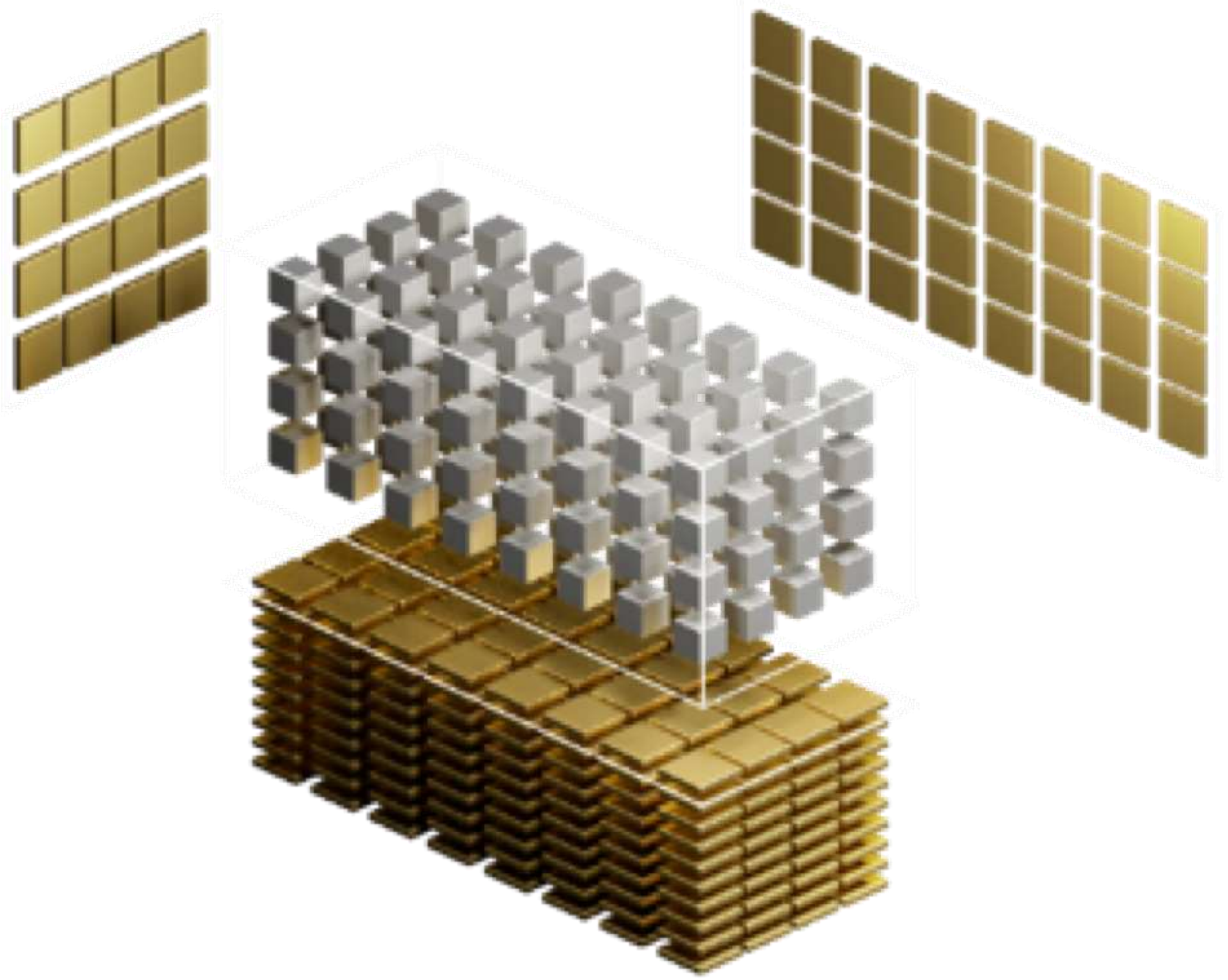
## Tensor Cores

- An increasing proportion of GPU die area is spent on AI
- There is a plethora of tensor cores of various precisions for AI

# If you can’t beat them, join them

## Tensor Cores

- An increasing proportion of GPU die area is spent on AI
- There is a plethora of tensor cores of various precisions for AI
- Tensor cores accelerate matrix-matrix multiplication (GEMMs)



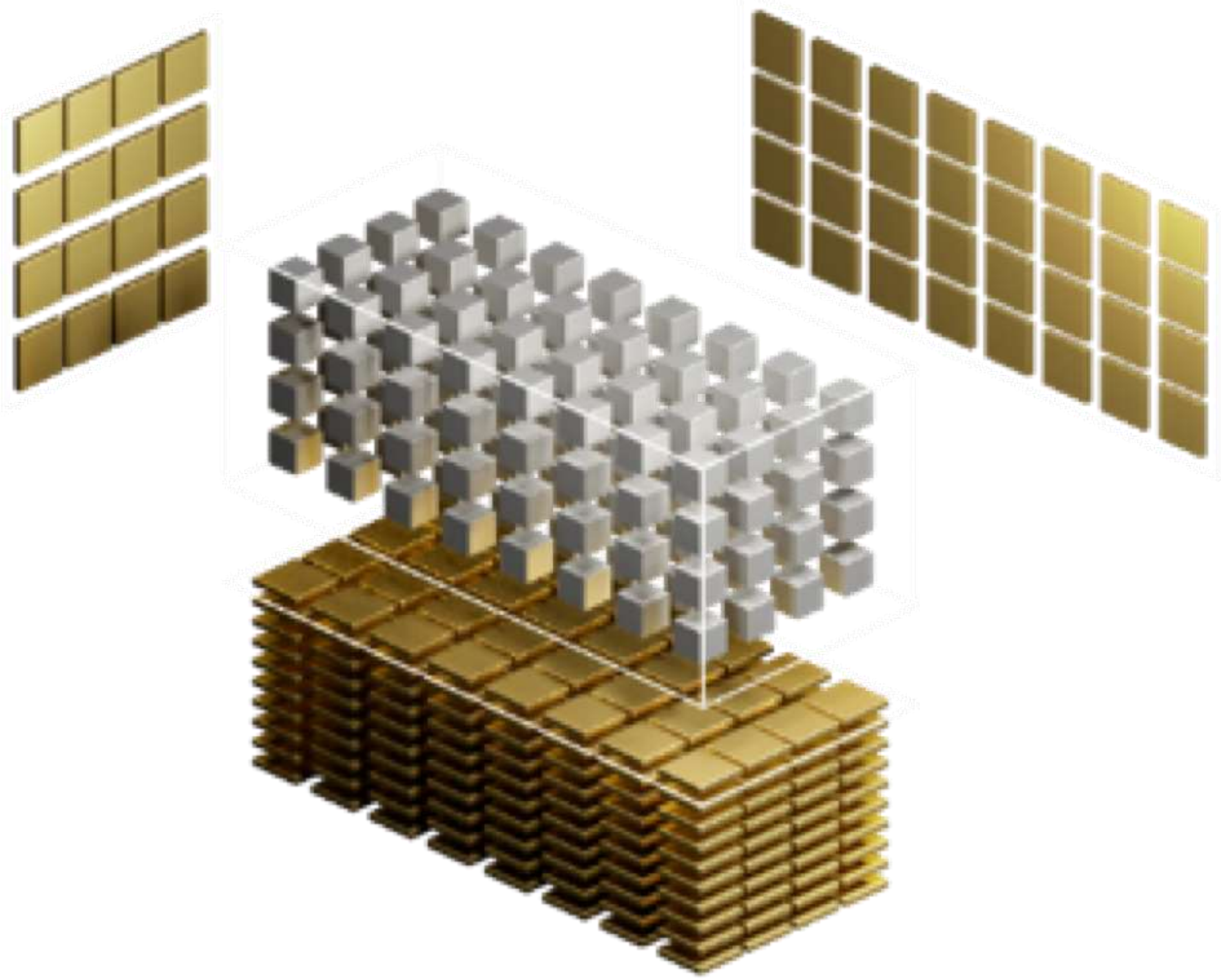
# If you can't beat them, join them

## Tensor Cores

- An increasing proportion of GPU die area is spent on AI
- There is a plethora of tensor cores of various precisions for AI
- Tensor cores accelerate matrix-matrix multiplication (GEMMs)
- Combine multiple low-precision tensor-core operations to emulate higher precision

$$C = AB = (A_{hi} + A_{lo})(B_{hi} + B_{lo})$$
$$\sim (A_{hi}B_{hi} + A_{hi}B_{lo} + A_{lo}B_{hi})$$

  - FP32 ~ 3xTF32
  - QUDA half ~ 3x BF16



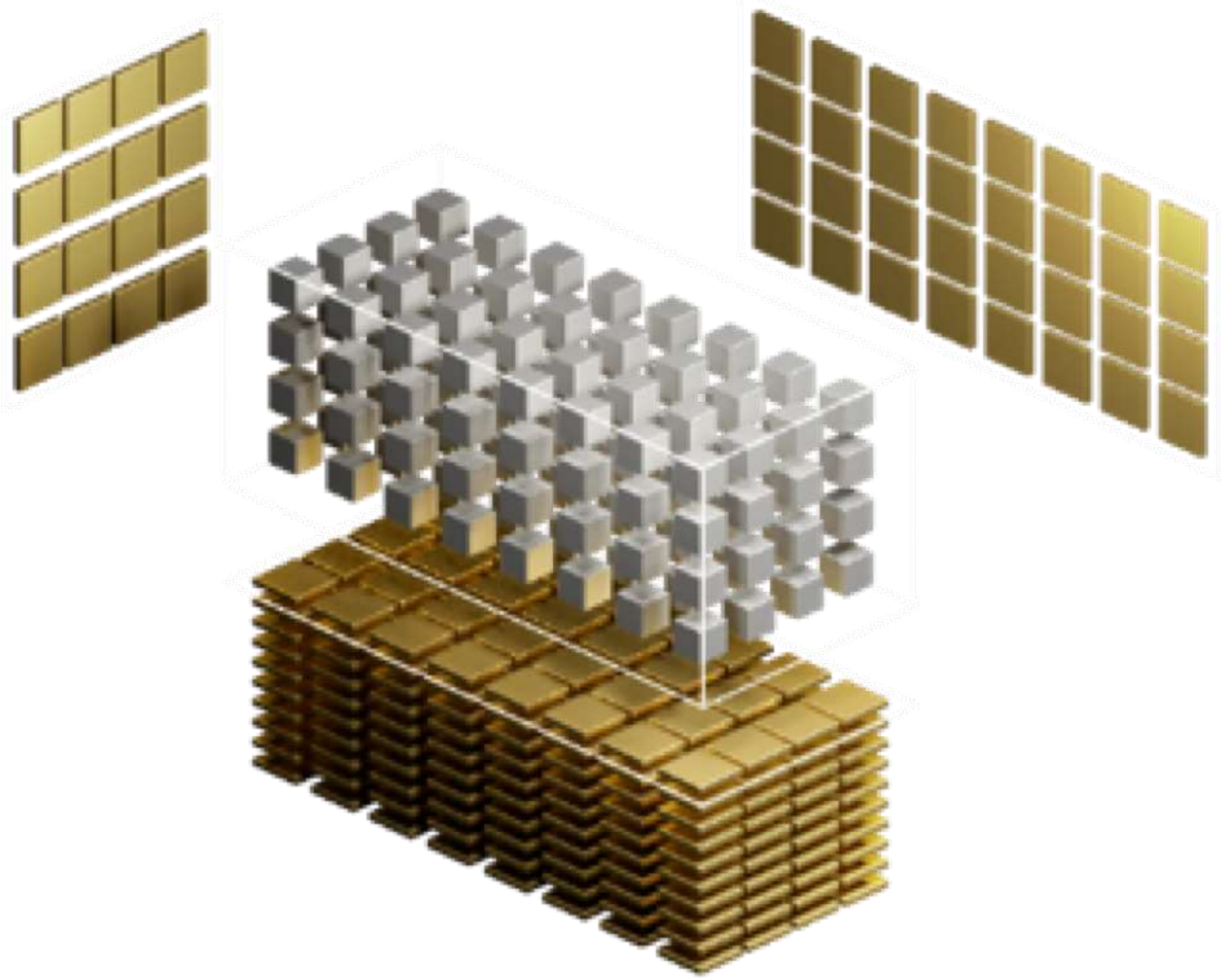
# If you can't beat them, join them

## Tensor Cores

- An increasing proportion of GPU die area is spent on AI
- There is a plethora of tensor cores of various precisions for AI
- Tensor cores accelerate matrix-matrix multiplication (GEMMs)
- Combine multiple low-precision tensor-core operations to emulate higher precision

$$C = AB = (A_{hi} + A_{lo})(B_{hi} + B_{lo})$$
$$\sim (A_{hi}B_{hi} + A_{hi}B_{lo} + A_{lo}B_{hi})$$

  - FP32 ~ 3xTF32
  - QUDA half ~ 3x BF16
- If you have a big enough GEMM, tensor cores rock



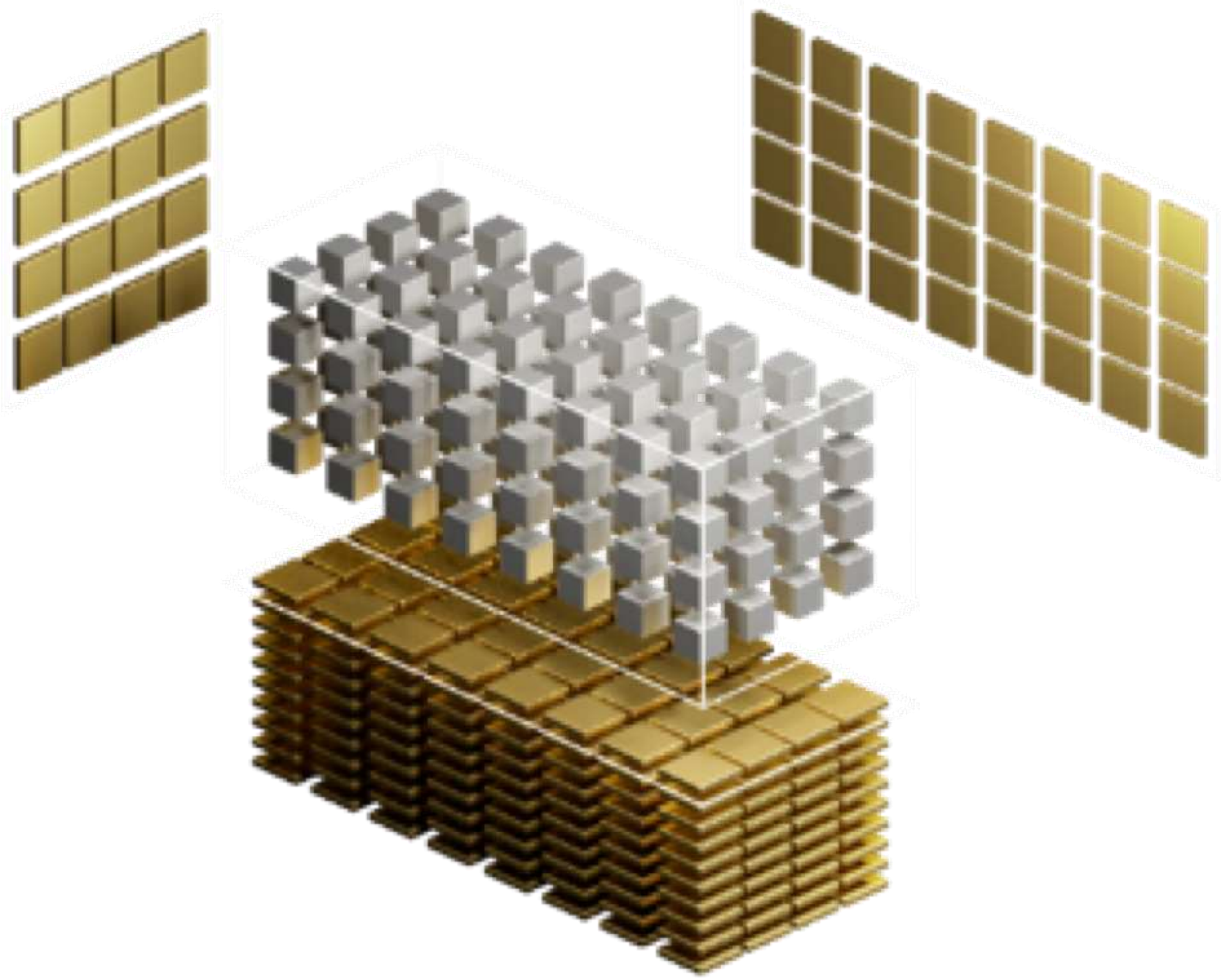
# If you can't beat them, join them

## Tensor Cores

- An increasing proportion of GPU die area is spent on AI
- There is a plethora of tensor cores of various precisions for AI
- Tensor cores accelerate matrix-matrix multiplication (GEMMs)
- Combine multiple low-precision tensor-core operations to emulate higher precision

$$C = AB = (A_{hi} + A_{lo})(B_{hi} + B_{lo})$$
$$\sim (A_{hi}B_{hi} + A_{hi}B_{lo} + A_{lo}B_{hi})$$

  - FP32 ~ 3xTF32
  - QUDA half ~ 3x BF16
- If you have a big enough GEMM, tensor cores rock
- QUDA's MG for LQCD has many tensor-core-friendly factors: 24, 32, 64…



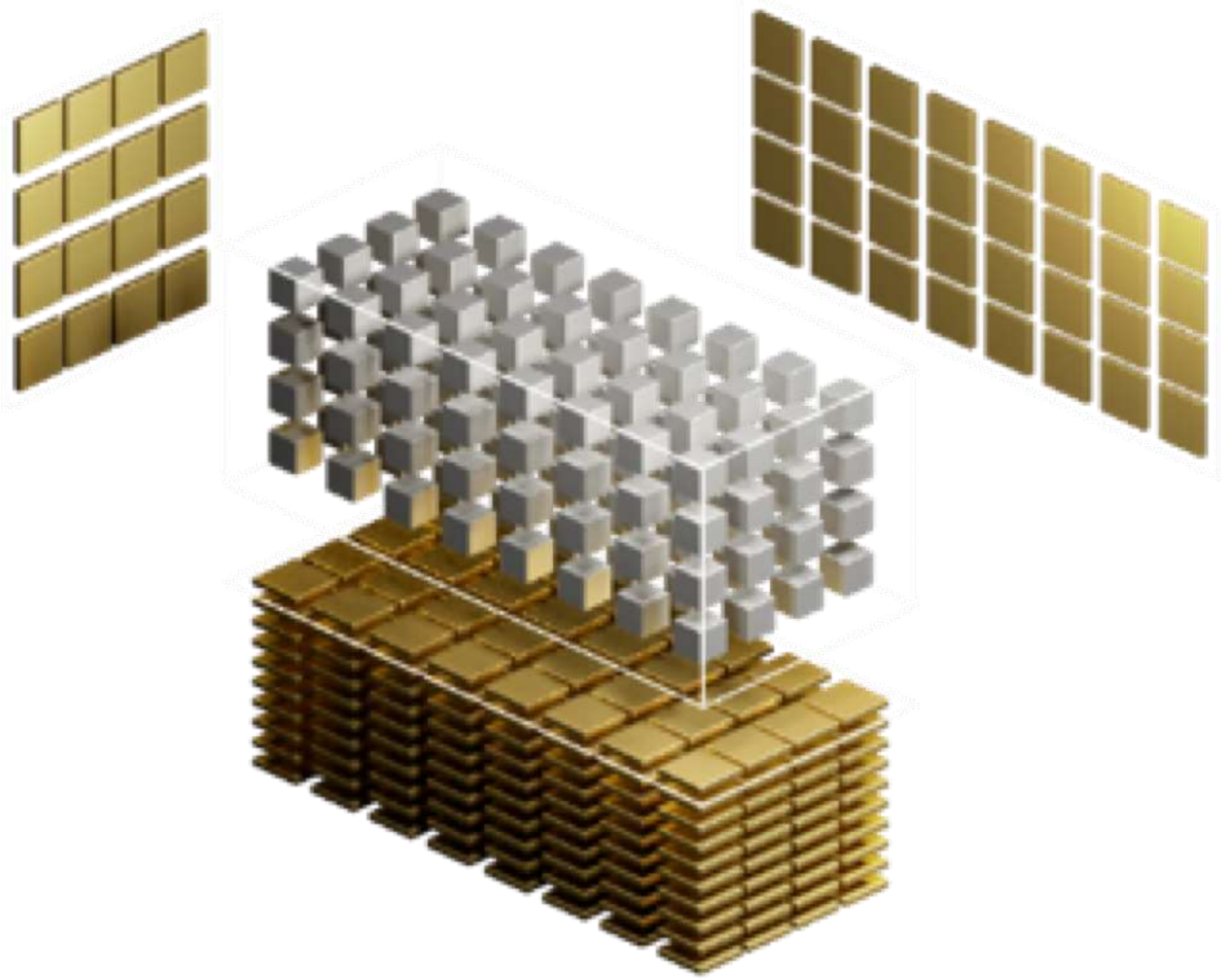
# If you can't beat them, join them

## Tensor Cores

- An increasing proportion of GPU die area is spent on AI
- There is a plethora of tensor cores of various precisions for AI
- Tensor cores accelerate matrix-matrix multiplication (GEMMs)
- Combine multiple low-precision tensor-core operations to emulate higher precision

$$C = AB = (A_{hi} + A_{lo})(B_{hi} + B_{lo})$$
$$\sim (A_{hi}B_{hi} + A_{hi}B_{lo} + A_{lo}B_{hi})$$

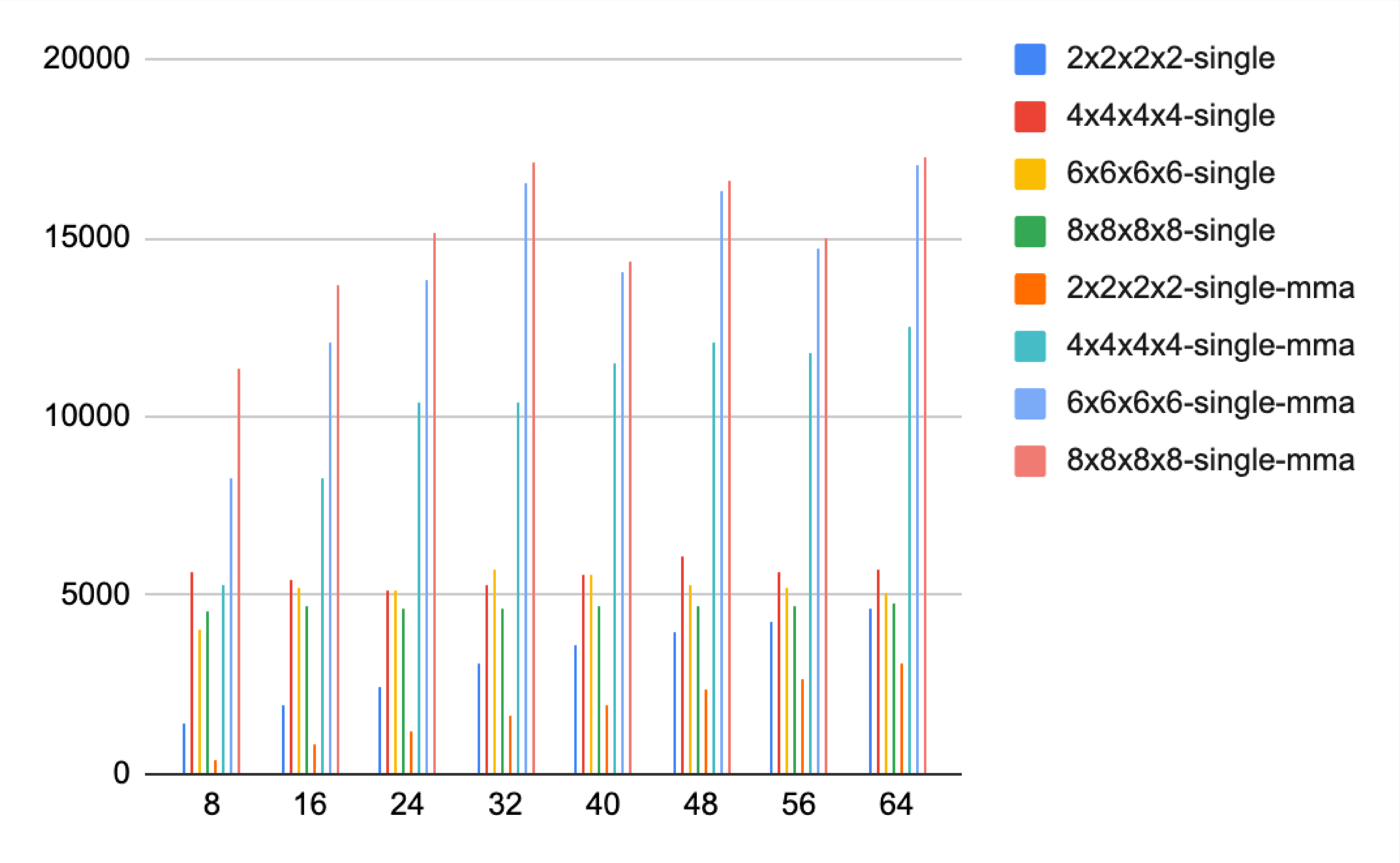
  - FP32 ~ 3xTF32
  - QUDA half ~ 3x BF16
- If you have a big enough GEMM, tensor cores rock
- QUDA's MG for LQCD has many tensor-core-friendly factors: 24, 32, 64…
- We just need to find the GEMMs!



# GEMMs in Multigrid

## Tensor Cores

- There are a lot of linear operations that act on a single vector
- These can also be batched: matrix-vector becomes matrix-matrix
- Multigrid has perhaps the greatest to benefit from MRHS
    - Coarse operator has more “colours” so more locality
    - Coarse grids are extremely parallelism challenged




Tensor-core accelerated multi-RHS coarse single-precision Dslash (A100)

**5 TFLOPS -> 15 TFLOPS**

# Multigrid + Multiple Right-Hand Sides

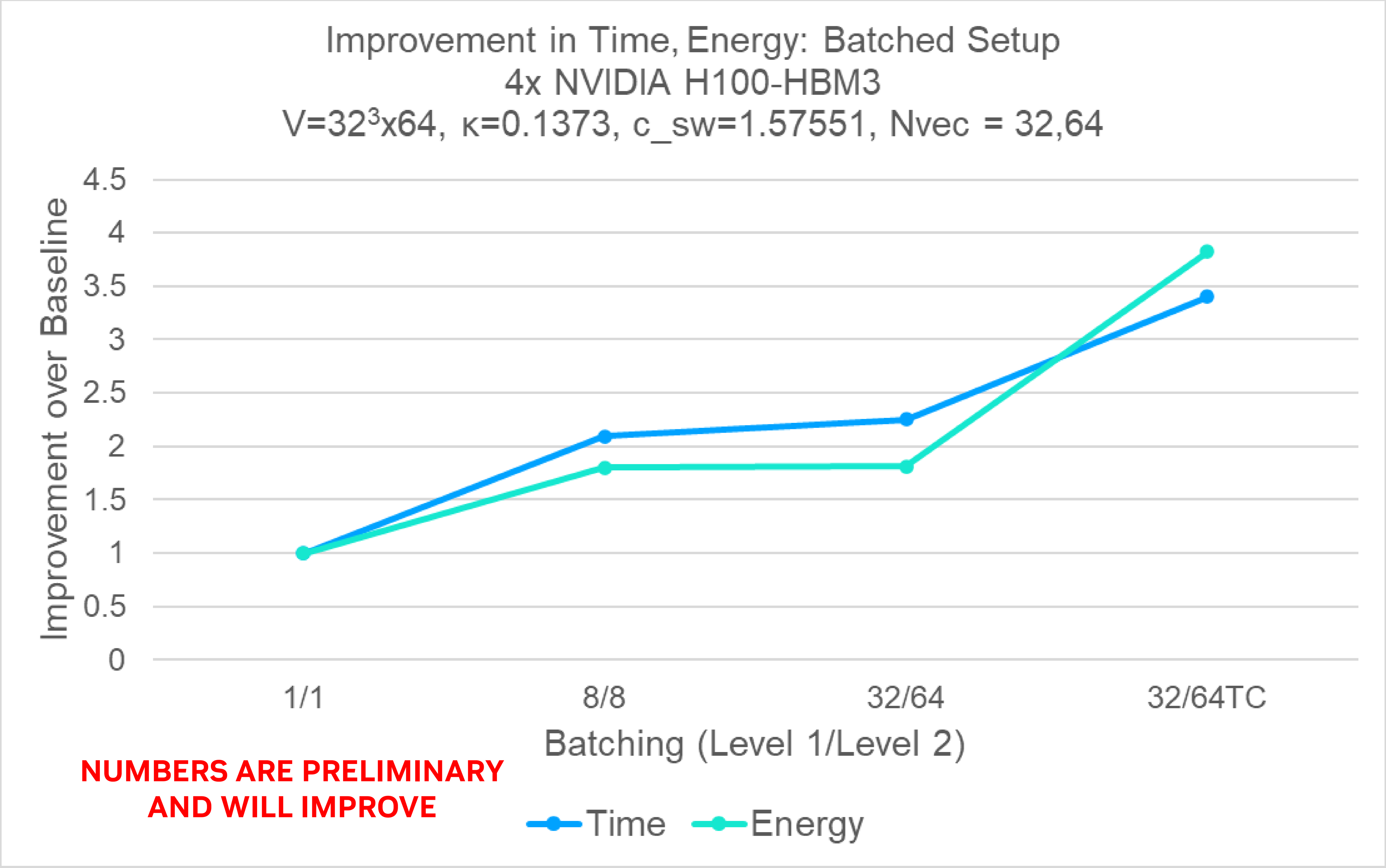
## Setup

- There is always scope for batched operations during MG setup:
  - Batched generation of near-null vectors: coarse dslash
  - Batched generation of lowest-level singular vectors
  - Batched block orthogonalization
  - Batched link coarsening

# Multigrid + Multiple Right-Hand Sides

## Setup

- There is always scope for batched operations during MG setup:
  - Batched generation of near-null vectors: coarse dslash
  - Batched generation of lowest-level singular vectors
  - Batched block orthogonalization
  - Batched link coarsening
- On the right
  - Batched and tensor-core accelerated near-null vector generation
  - Batched and, for coarse operator coarsening, tensor-core accelerated link coarsening




**3.4x faster**

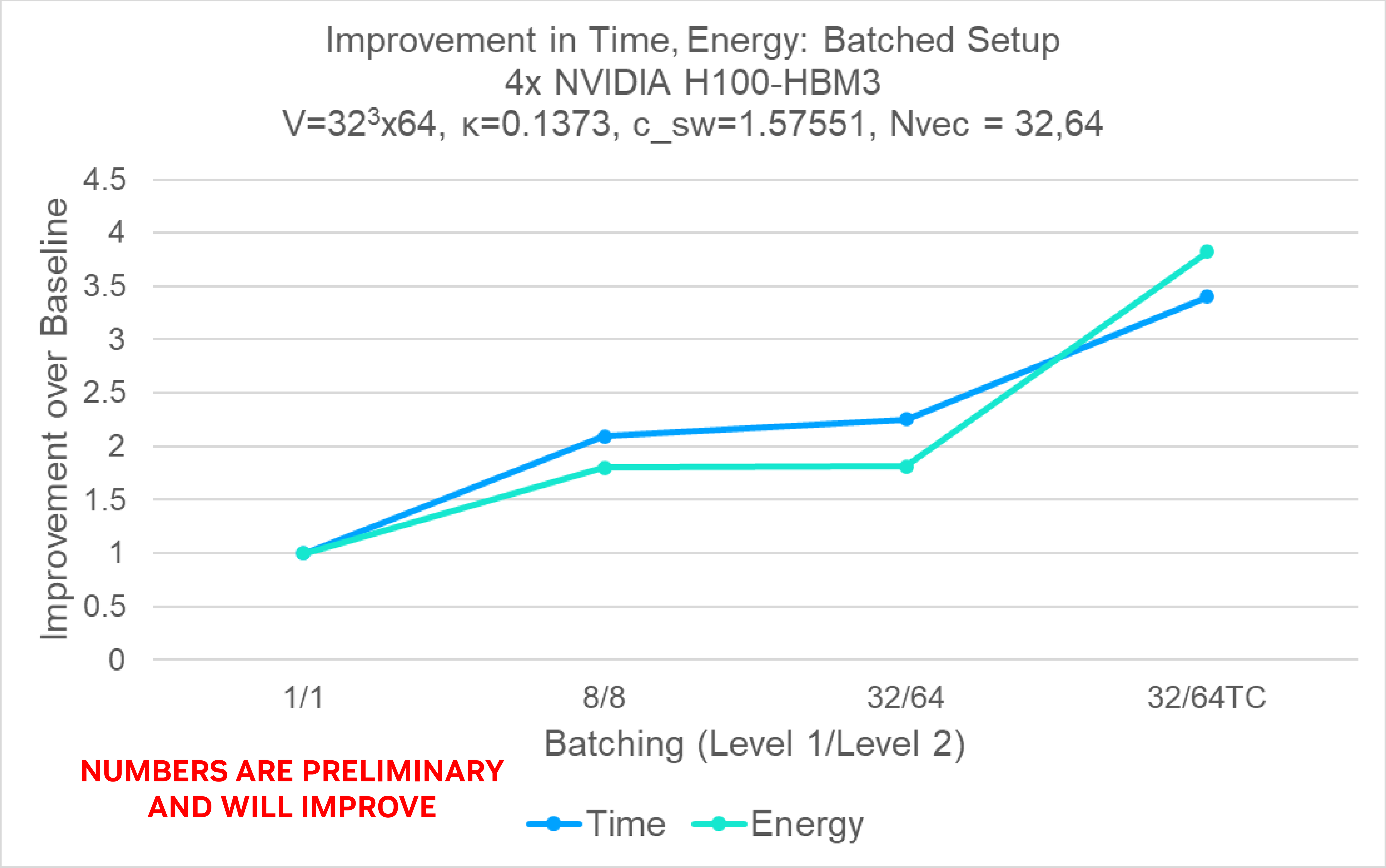
**and**

**3.8x less energy**

# Multigrid + Multiple Right-Hand Sides

## Setup

- There is always scope for batched operations during MG setup:
  - Batched generation of near-null vectors: coarse dslash
  - Batched generation of lowest-level singular vectors
  - Batched block orthogonalization
  - Batched link coarsening
- On the right
  - Batched and tensor-core accelerated near-null vector generation
  - Batched and, for coarse operator coarsening, tensor-core accelerated link coarsening
- Speedups will only increase as optimization progresses




**3.4x faster**
**and**
**3.8x less energy**

# Multigrid + Multiple Right-Hand Sides

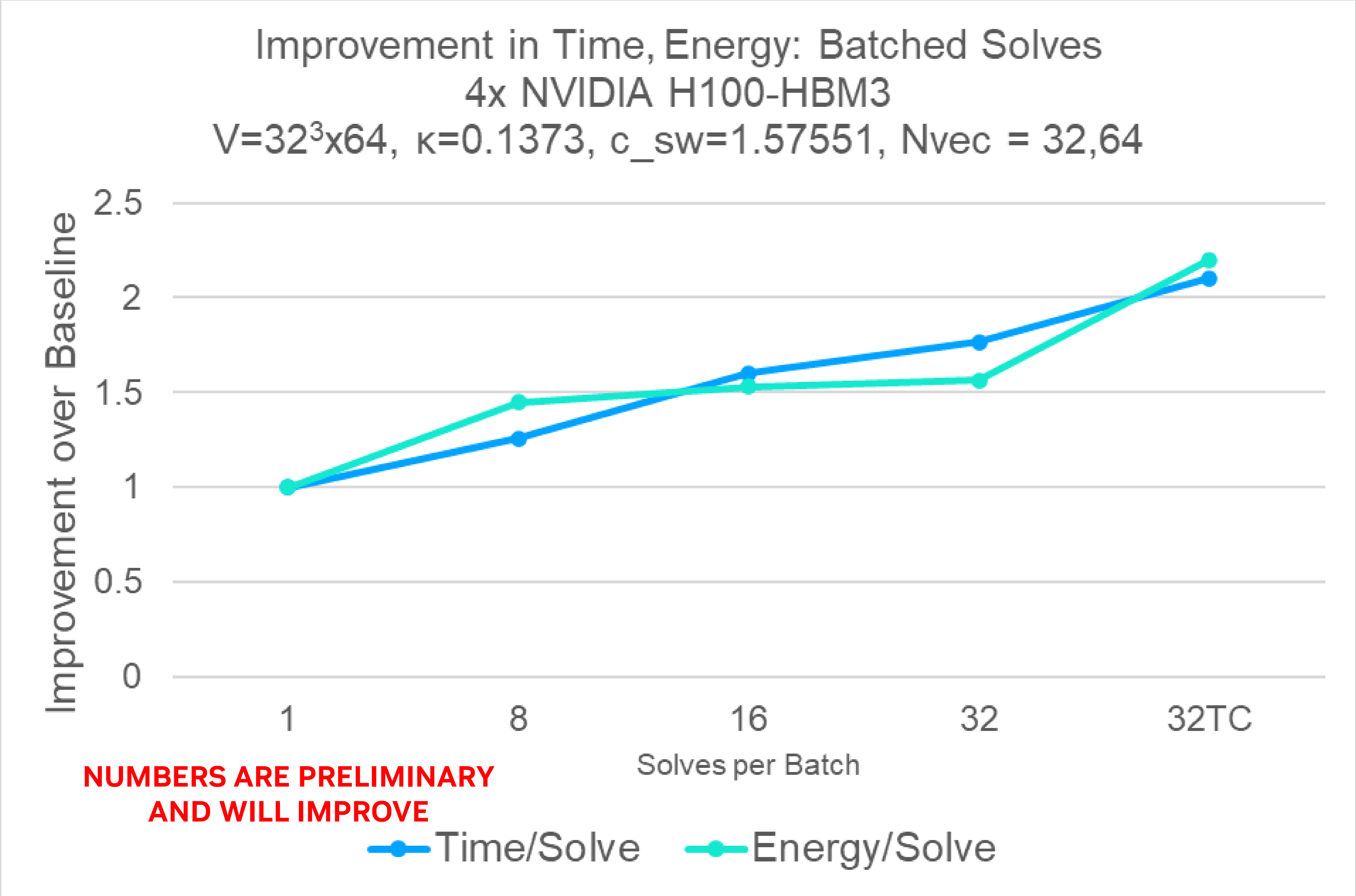
## Solver

- During MG solves… if they're batched (multiple sources)
  - Batched coarse dslash
  - Batched prolongator, restrictor
  - Batched SVD deflation

# Multigrid + Multiple Right-Hand Sides

## Solver

- During MG solves… if they're batched (multiple sources)
  - Batched coarse dslash
  - Batched prolongator, restrictor
  - Batched SVD deflation
- On the right
  - Batched and tensor-core accelerated coarse dslash
  - Batched but not (yet) tensor-core accelerated prolongator and restrictor
  - Batched SVD deflation

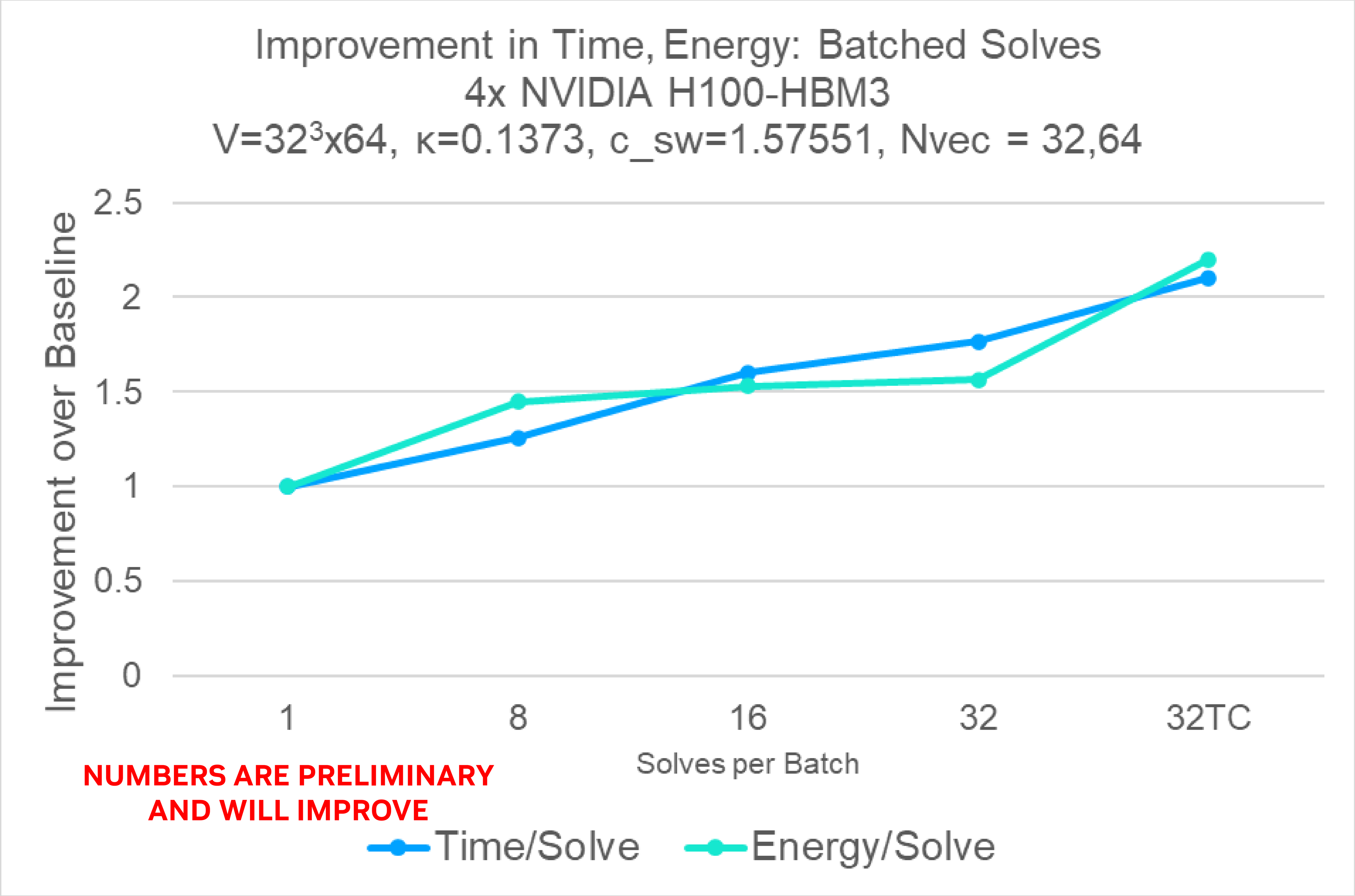



**2.1x faster**
**and**
**2.2x less energy**

# Multigrid + Multiple Right-Hand Sides

## Solver

- During MG solves… if they're batched (multiple sources)
  - Batched coarse dslash
  - Batched prolongator, restrictor
  - Batched SVD deflation
- On the right
  - Batched and tensor-core accelerated coarse dslash
  - Batched but not (yet) tensor-core accelerated prolongator and restrictor
  - Batched SVD deflation
- Again, speedups will only increase as optimization progresses




**2.1x faster**
**and**
**2.2x less energy**

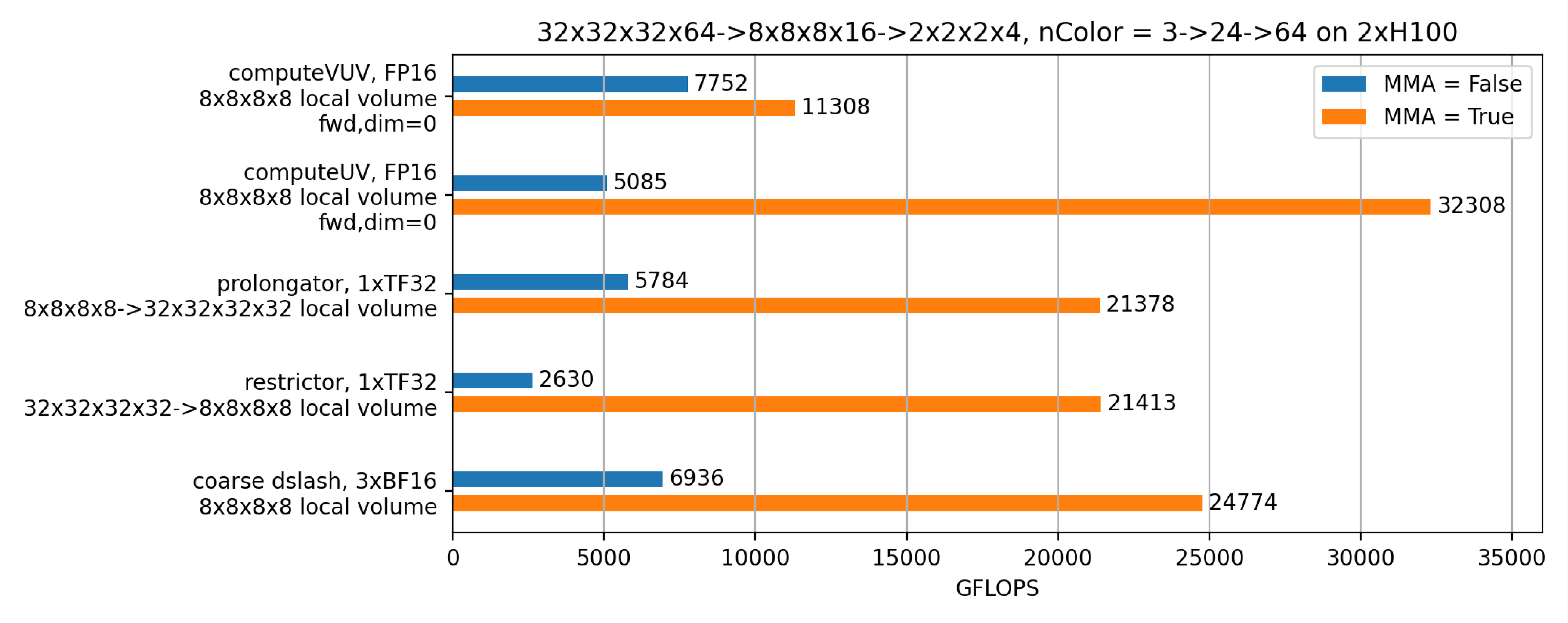
# Improvements beget Improvements

- Multi-RHS motivates a retuning of algorithmic parameters
  - Significant cost reduction for setup provides scope to improve preconditioner quality
  - As we increase RHS, we can get a better solver at constant iteration cost

# Improvements beget Improvements

- Multi-RHS motivates a retuning of algorithmic parameters
  - Significant cost reduction for setup provides scope to improve preconditioner quality
  - As we increase RHS, we can get a better solver at constant iteration cost
- This calculus can change with each improvement…

# Improvements beget Improvements

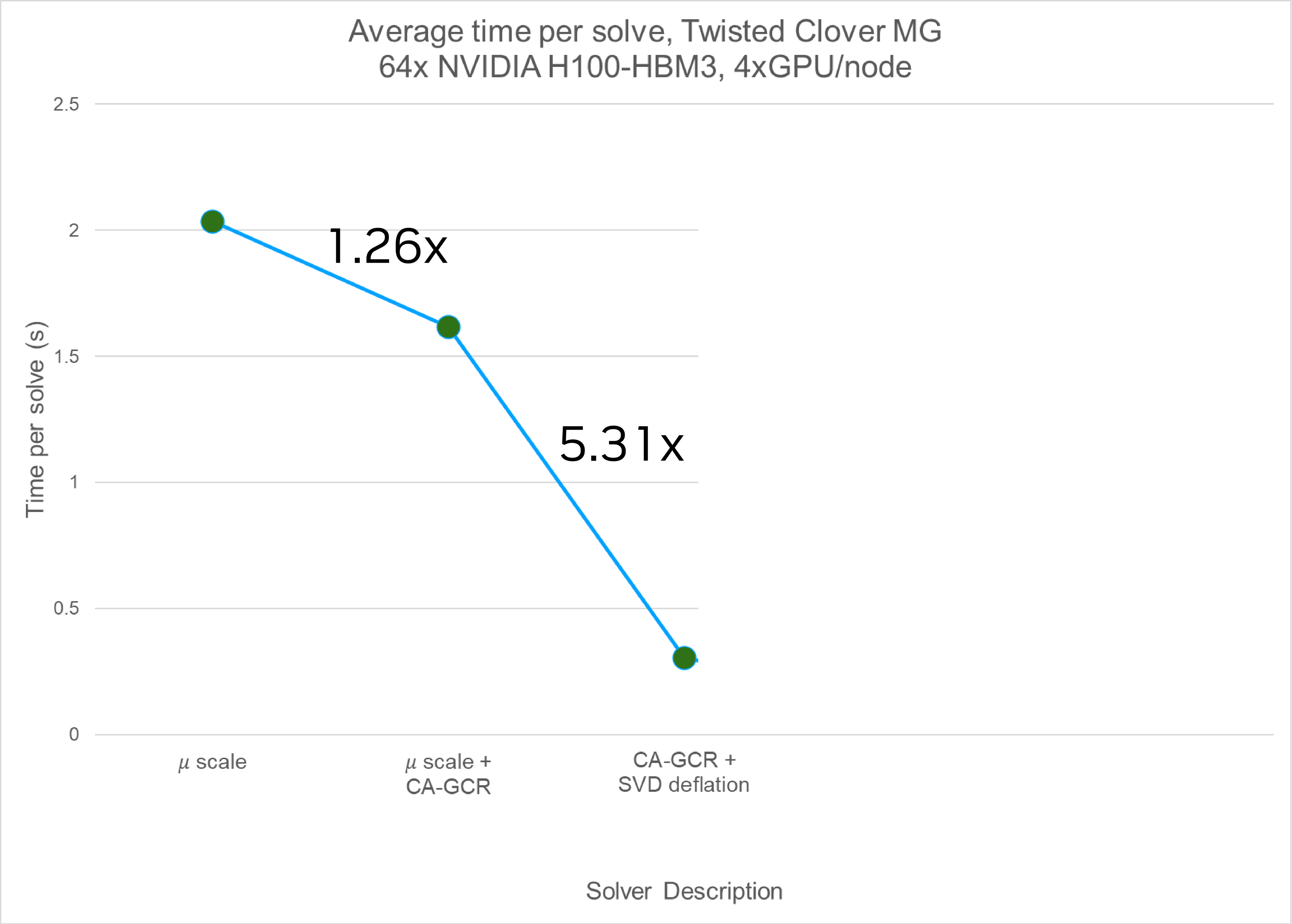


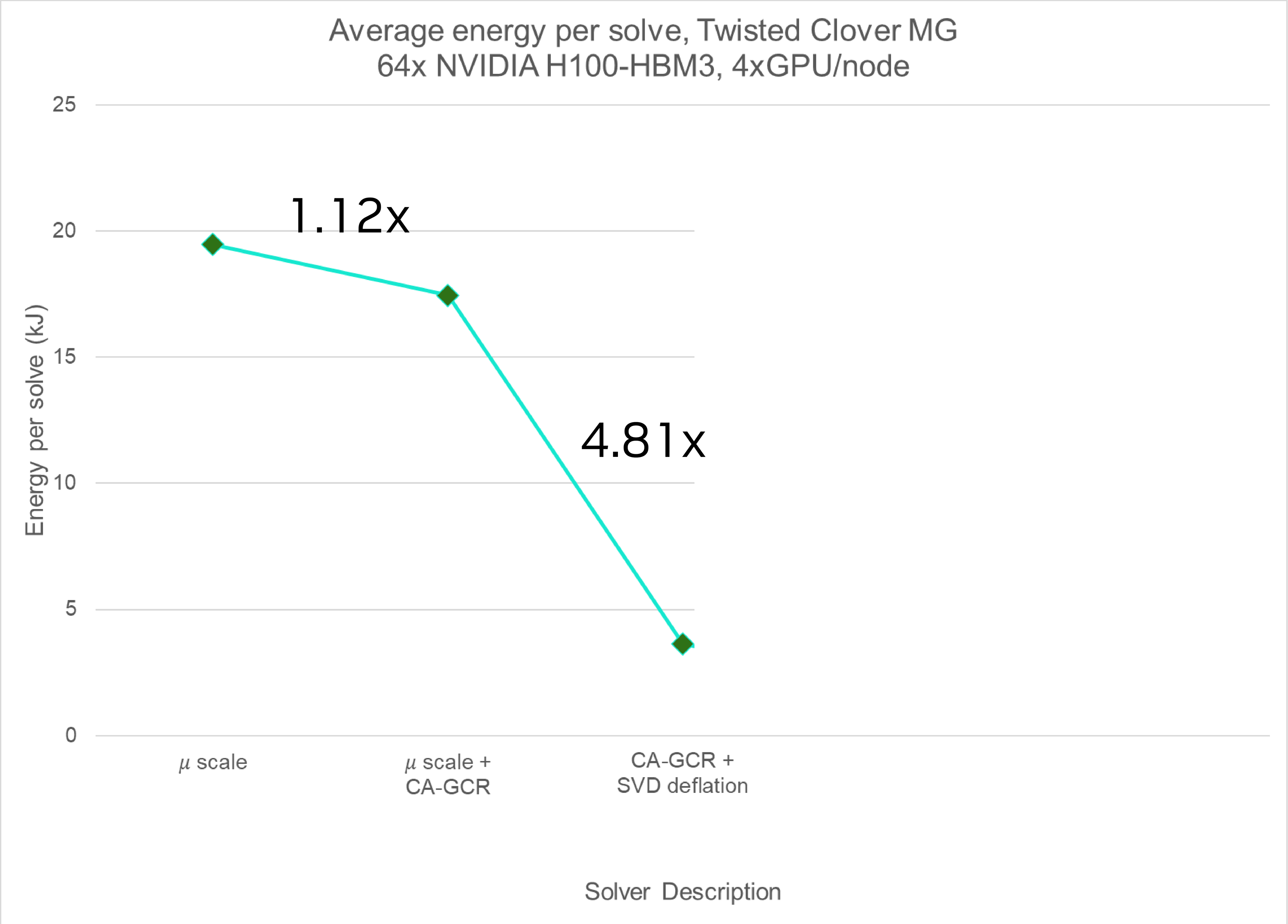

- Multi-RHS motivates a retuning of algorithmic parameters
  - Significant cost reduction for setup provides scope to improve preconditioner quality
  - As we increase RHS, we can get a better solver at constant iteration cost
- This calculus can change with each improvement…
- …and algorithmic improvements can keep coming
  - Preliminary: tensor-core-accelerated prolongator and restrictor…
  - …Among other TC-accelerated portions of MG

# Revisiting the Twisted Clover Example

## Energy Consumption

- “Optimization” doesn’t necessarily (just) refer to time to solution
- It can also refer to *energy* to solution
  - Which doesn’t *always* correlate, but often does



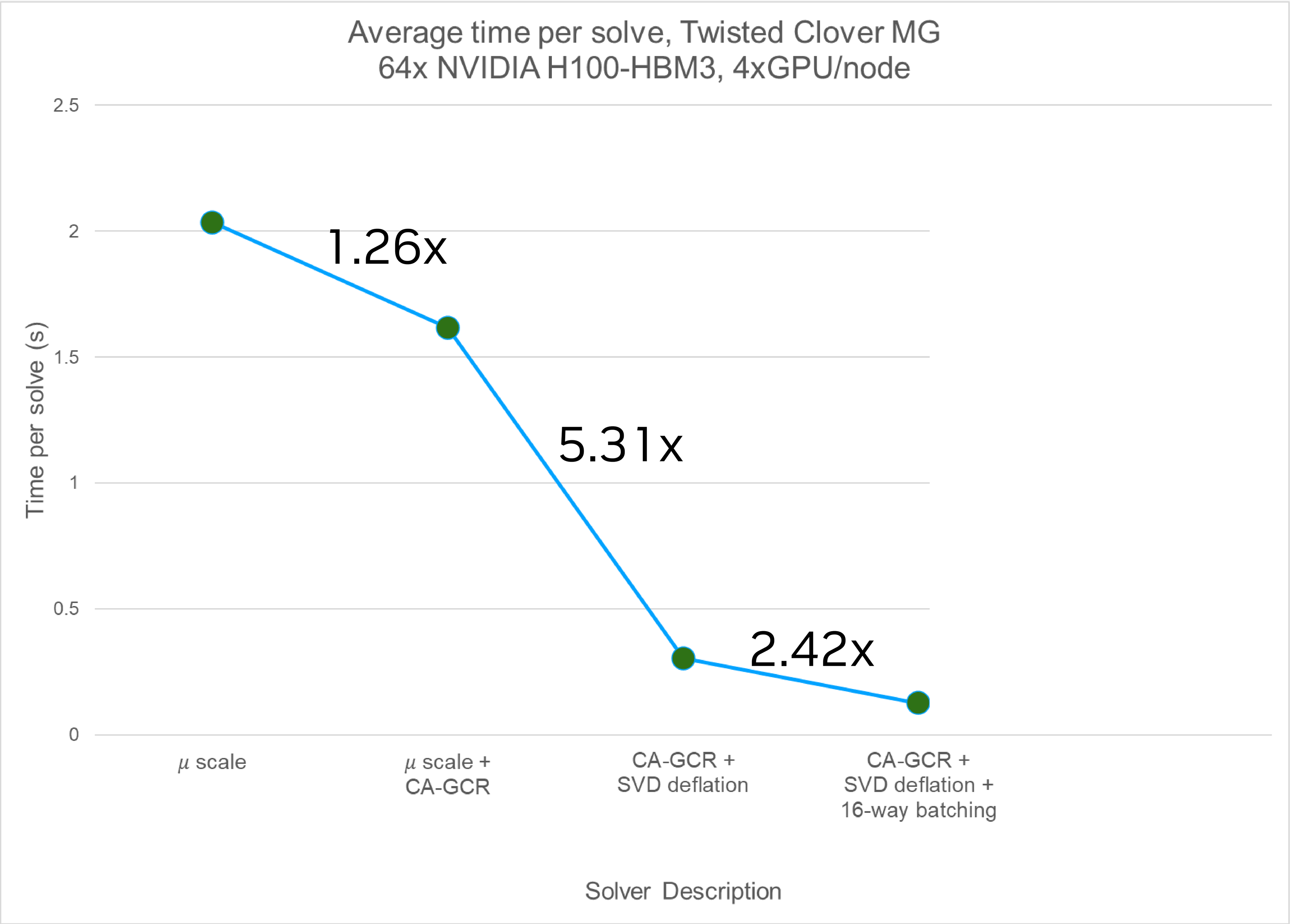



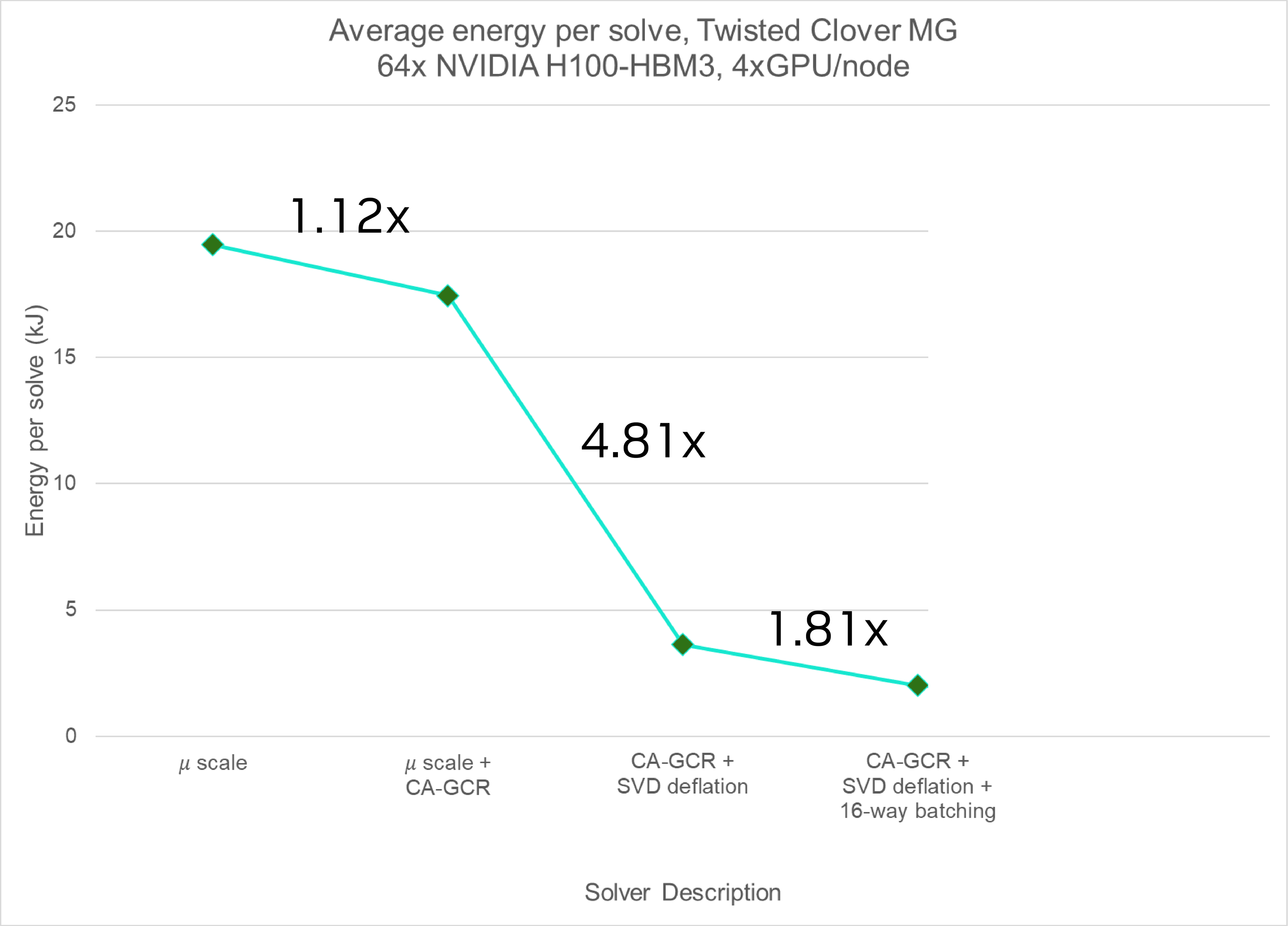

A modernization of https://github.com/lattice/quda/wiki/Twisted-clover-deflated-multigrid from Dean Howarth

# Twisted Clover Example

## Batching Solves

- Our next step isn't necessarily a novel idea, but it keys in on energy efficiency
- Here we begin *batching* operations
  - Without tensor cores… for now
- There are *always* operations to batch in setup
  - I'm not showing the setup because I'm still fighting with the block Lanczos
- There are not always operations to batch in the solver phase
  - You may only need one solve (HMC)
  - …but take the results on the right as a proxy for improvements



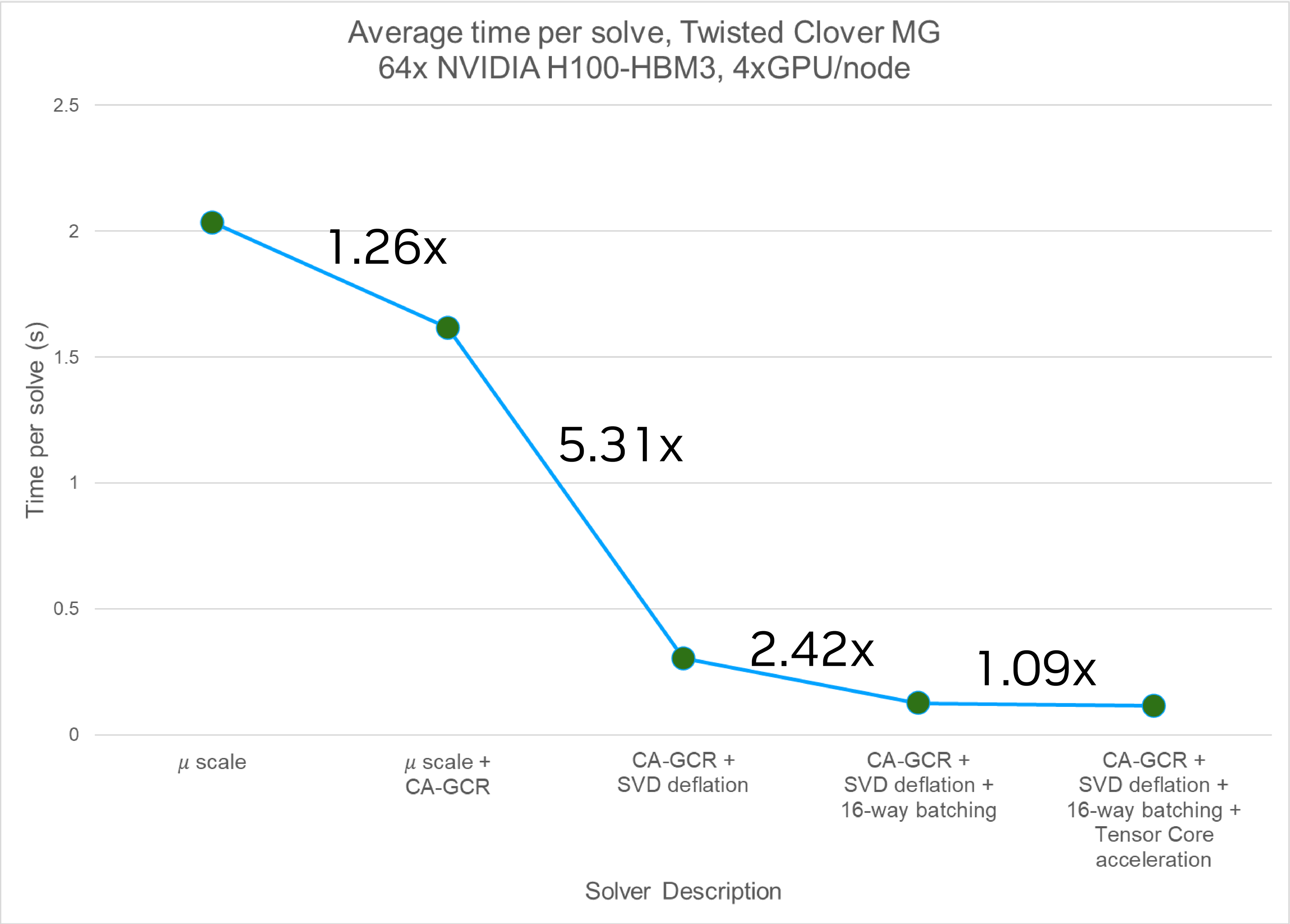



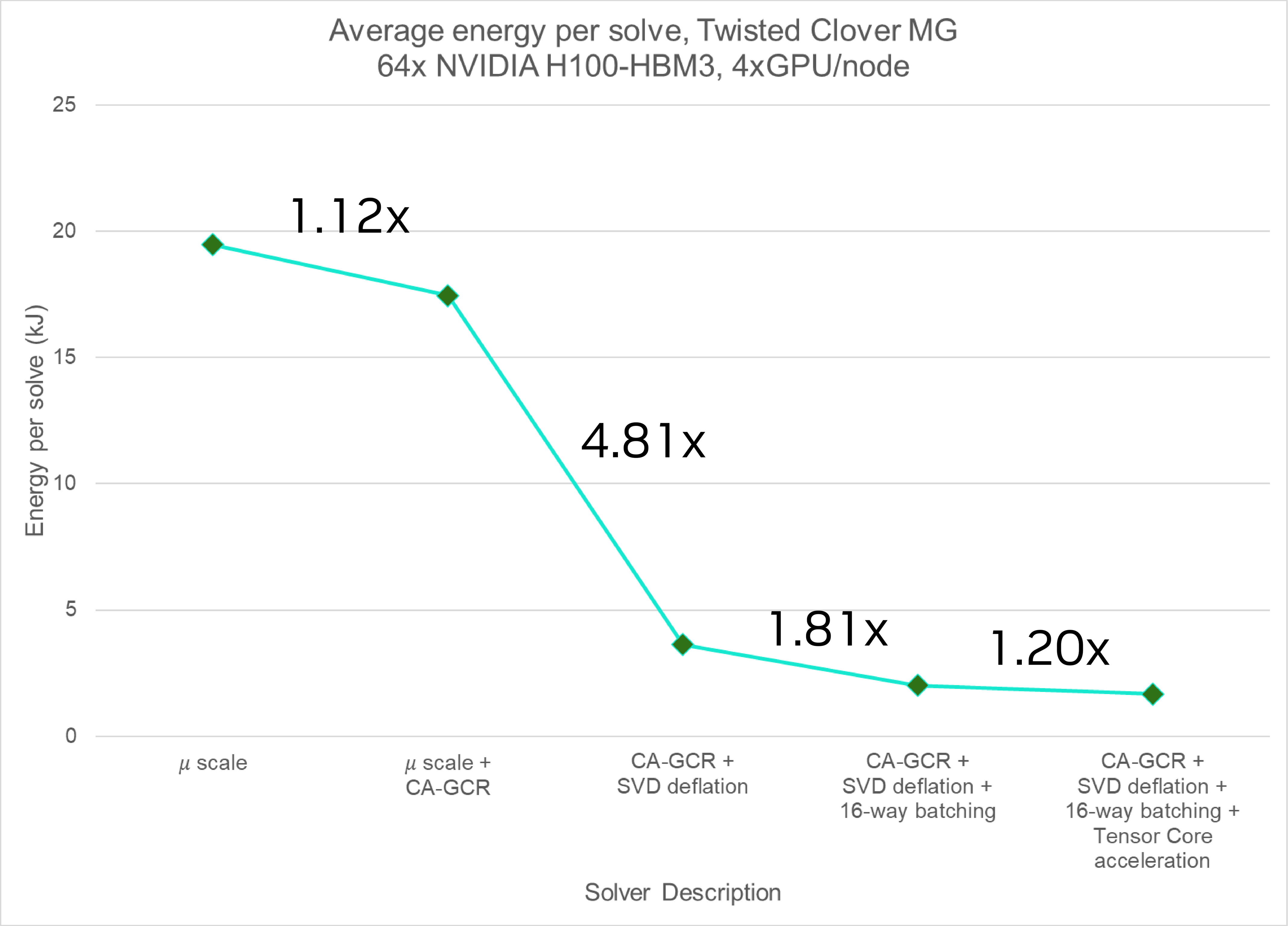

A modernization of https://github.com/lattice/quda/wiki/Twisted-clover-deflated-multigrid from Dean Howarth

# Twisted Clover Example

## Tensor Cores

- Last, we include the tensor core acceleration
  - Energy savings outpace time-to-solution improvements
  - Tensor cores by construction promote matrix-multiply ultra-locality
- Aggregate benefits:
  - Time to solution: 17.55x
  - Energy to solution: 11.62x
  - *Each step contributed*



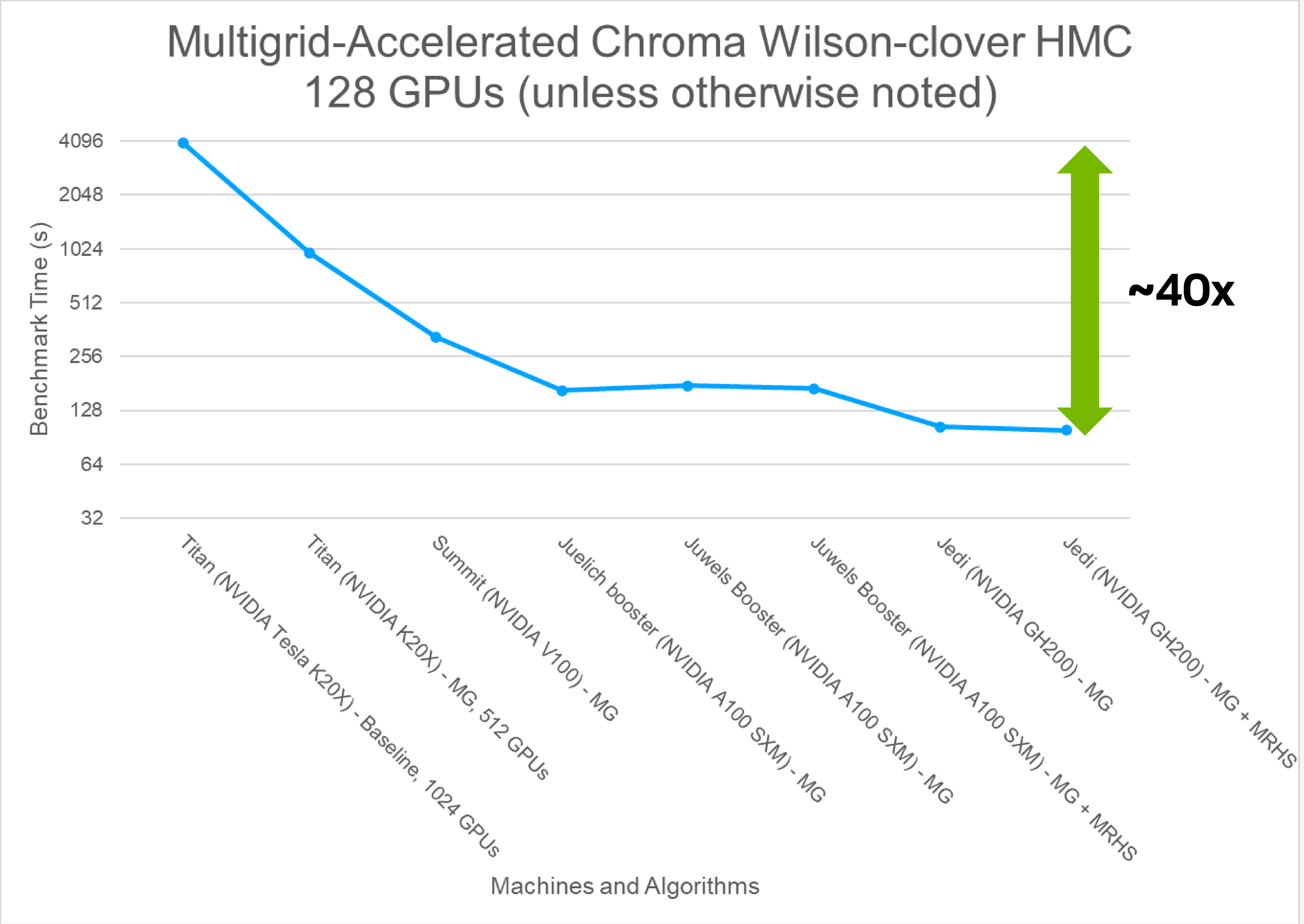

A modernization of https://github.com/lattice/quda/wiki/Twisted-clover-deflated-multigrid from Dean Howarth

# A History of Algorithms and Machines

## Chroma + QDP-JIT + QUDA

- There is a storied history of MG-accelerated Wilson-clover HMC driven by Chroma
- HMC typically dominated by solving the Dirac equation, but
  - Few solves per linear system
  - Can be bound by heavy solves (c.f. Hasenbusch mass preconditioning)
- Multigrid setup must run at speed of light
  - Reuse and evolve multigrid setup where possible
  - Use the same null space for all
  - Evolve null space as the gauge field evolves (Lüscher 2007)
  - Update null space when the preconditioner degrades too much on lightest mass
- Machines plus algorithms has made this faster

**Benchmark Time to Solution**




Chroma w/ QDP-JIT and QUDA

$V=64^3 x128$ sites, $m_\pi$ ~172 MeV

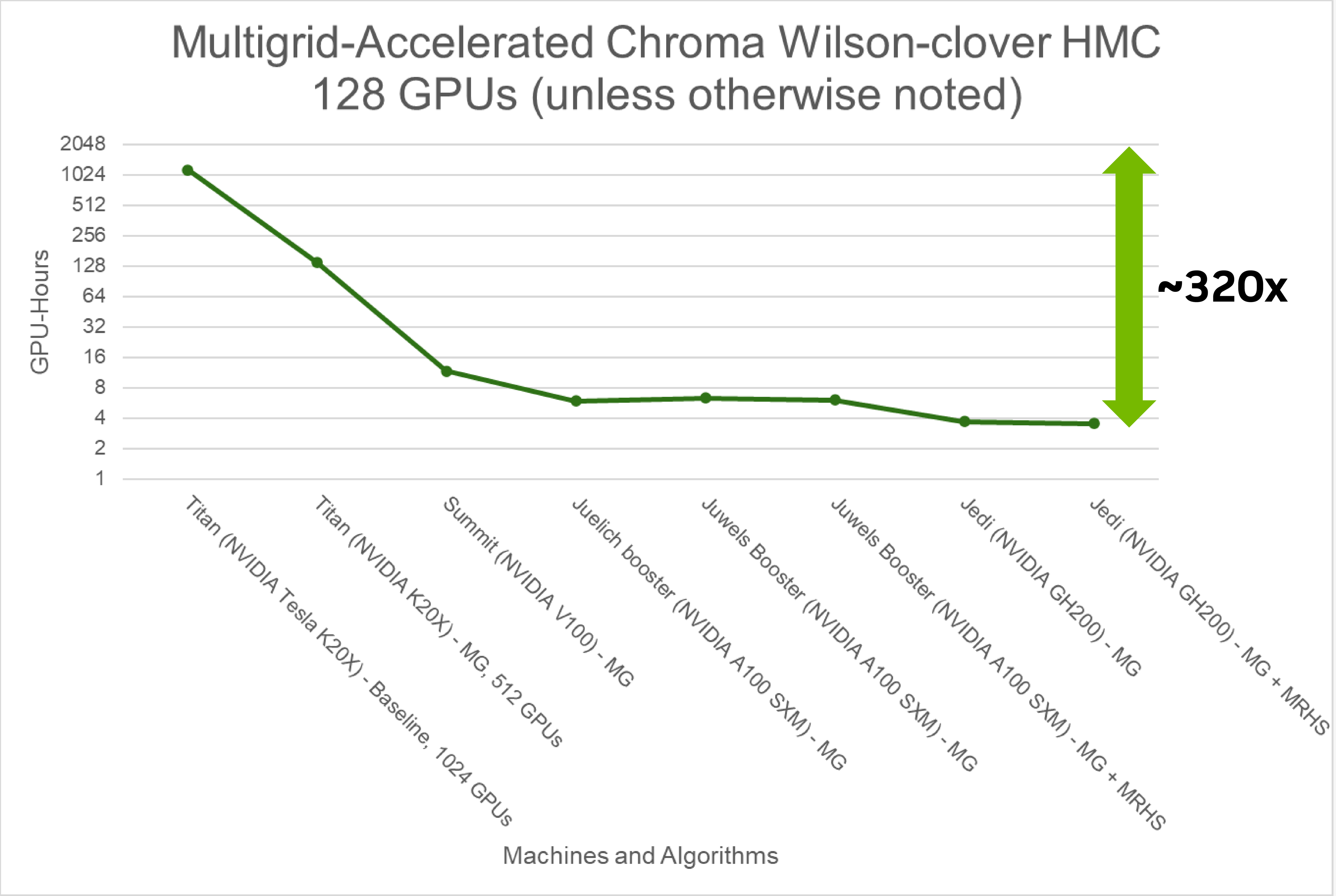
(QDP-JIT by F. Winter, Jefferson Lab)

# The Intersection of Algorithms and Machines

Chroma + QDP-JIT + QUDA

- There is a storied history of MG-accelerated Wilson-clover HMC driven by Chroma
- HMC typically dominated by solving the Dirac equation, but
  - Few solves per linear system
  - Can be bound by heavy solves (c.f. Hasenbusch mass preconditioning)
- Multigrid setup must run at speed of light
  - Reuse and evolve multigrid setup where possible
  - Use the same null space for all
  - Evolve null space as the gauge field evolves (Lüscher 2007)
  - Update null space when the preconditioner degrades too much on lightest mass
- Machines plus algorithms has made this faster
- **And makes fixed allocations go further**

**Benchmark GPU-Hours**




Chroma w/ QDP-JIT and QUDA

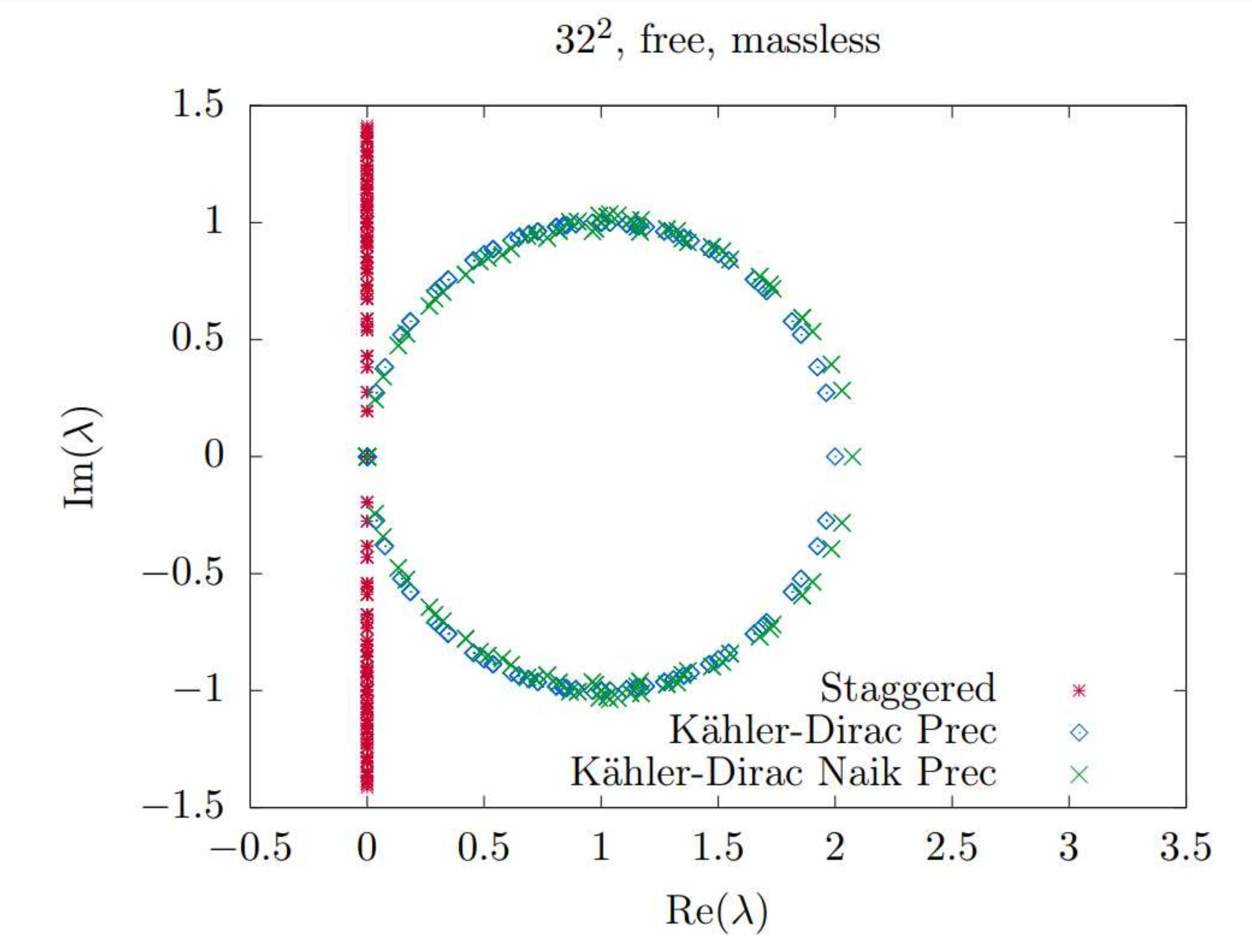
$V=64^3 \times 128$ sites, $m_\pi \sim 172$ MeV

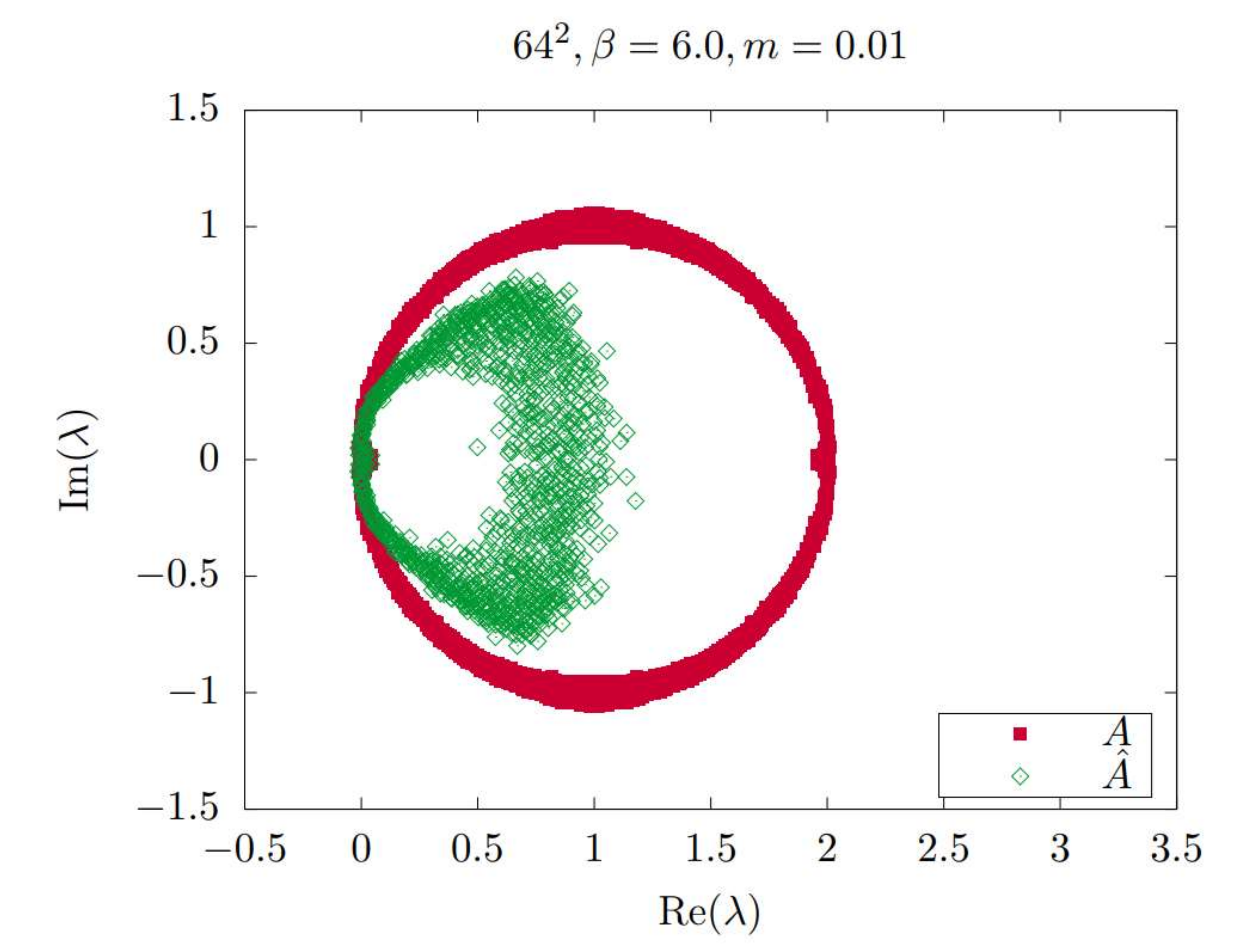
(QDP-JIT by F. Winter, Jefferson Lab)

# Staggered Fermions: Kahler-Dirac preconditioning

## Spectral deformations

- 2-d paper: arXiv:1801.07823
- Core idea: spectral deformation by Kahler-Dirac structure
  - Each $2^d$ hypercube of staggered dof = one lattice Kahler-Dirac fermion
  - Block-precondition by this $2^d$ structure
- Deforms anti-Hermitian indefinite spectrum into (roughly) circular spectrum
- Carries similar spectral properties as Wilson-clover after coarsening
- Implemented in QUDA, exposed in MILC



# Batched HISQ Multigrid

## Setup *and* Solve

- The “gotchas” of Staggered/HISQ
  - Four Dirac fermions means 4x the zero modes
  - The “fundamental” unit of degrees of freedom is the $2^4$ hypercube

# Batched HISQ Multigrid

## Setup *and* Solve

- The “gotchas” of Staggered/HISQ
  - Four Dirac fermions means 4x the zero modes
  - The “fundamental” unit of degrees of freedom is the $2^4$ hypercube
- Consequences:
  - Aggregates can be larger, ~$O(6^4)$…
  - …but there needs to be more coarse d.o.f.
  - …more benefit from multi-RHS

# Batched HISQ Multigrid

## Setup *and* Solve

- The “gotchas” of Staggered/HISQ
  - Four Dirac fermions means 4x the zero modes
  - The “fundamental” unit of degrees of freedom is the $2^4$ hypercube
- Consequences:
  - Aggregates can be larger, ~$O(6^4)$…
  - …but there needs to be more coarse d.o.f.
  - …more benefit from multi-RHS
- Example:
  - Fine level: $24^3$x48
  - Intermediate level: $6^3$x8, $N_c = 64, N_s = 2$
  - Coarsest level $2^3$x4, $N_c = 96, N_s = 2$


Setup: **12.6 sec, 4.58 Tflops** to **5.46 sec, 10.6 Tflops**

Solve throughput: 0.24 sec/solve to 0.15 sec/solve

# Batched HISQ Multigrid

## Setup *and* Solve

- The "gotchas" of Staggered/HISQ
  - Four Dirac fermions means 4x the zero modes
  - The "fundamental" unit of degrees of freedom is the $2^4$ hypercube
- Consequences:
  - Aggregates can be larger, ~$O(6^4)$…
  - …but there needs to be more coarse d.o.f.
  - …more benefit from multi-RHS
- Example:
  - Fine level: $24^3$x48
  - Intermediate level: $6^3$x8, $N_c = 64, N_s = 2$
  - Coarsest level $2^3$x4, $N_c = 96, N_s = 2$
- More degrees of freedom means more wins from multi-RHS


Setup: **12.6 sec, 4.58 Tflops** to **5.46 sec, 10.6 Tflops**
Solve throughput: 0.24 sec/solve to 0.15 sec/solve

# Batched HISQ Multigrid

## Setup *and* Solve

- The “gotchas” of Staggered/HISQ
  - Four Dirac fermions means 4x the zero modes
  - The “fundamental” unit of degrees of freedom is the $2^4$ hypercube
- Consequences:
  - Aggregates can be larger, ~$O(6^4)$…
  - …but there needs to be more coarse d.o.f.
  - …more benefit from multi-RHS
- Example:
  - Fine level: $24^3$x48
  - Intermediate level: $6^3$x8, $N_c = 64, N_s = 2$
  - Coarsest level $2^3$x4, $N_c = 96, N_s = 2$
- More degrees of freedom means more wins from multi-RHS
- Multi-RHS saves power *and* time

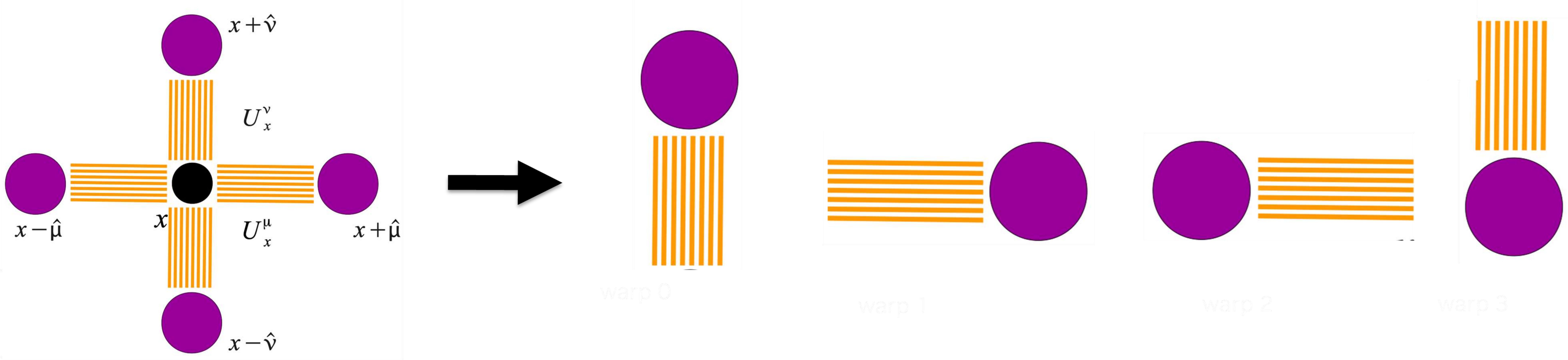

Setup: **12.6 sec, 4.58 Tflops** to **5.46 sec, 10.6 Tflops**
Solve throughput: 0.24 sec/solve to 0.15 sec/solve

# There’s More? Other Thoughts and Questions

# What We Haven't Covered

## We've come so far and have so far to go

- In **C**UDA:
  - (Even better) asynchronous SIMT: overlapping memory transactions and compute in a kernel
  - Tensor Memory Accelerator (TMA): Automatic stride & address generation up to tensors of rank 5
  - Coarse gauge links have a parity, checker-board coordinate, direction, row, column… we need all 5
  - Ex, for the coarse dslash: Overlap computing one direction with fetching the next




- In **Q**UDA:
  - The depths of HISQ multigrid
  - Future work on domain-wall/Mobius multigrid
- …and countless more

# And Don’t Forget…

## Where we started

- Multigrid: A class of algorithms that mitigate critical slowing down
  - And that’s nice, but the devil’s really in the details

# And Don’t Forget…

## Where we started

- Multigrid: A class of algorithms that mitigate critical slowing down
  - And that’s nice, but the devil’s really in the details
- Reality: Time (and energy) to solution is the only thing that matters
  - And that’s what must inform the algorithmic and implementation decisions that get made

# And Don’t Forget…

## Where we started

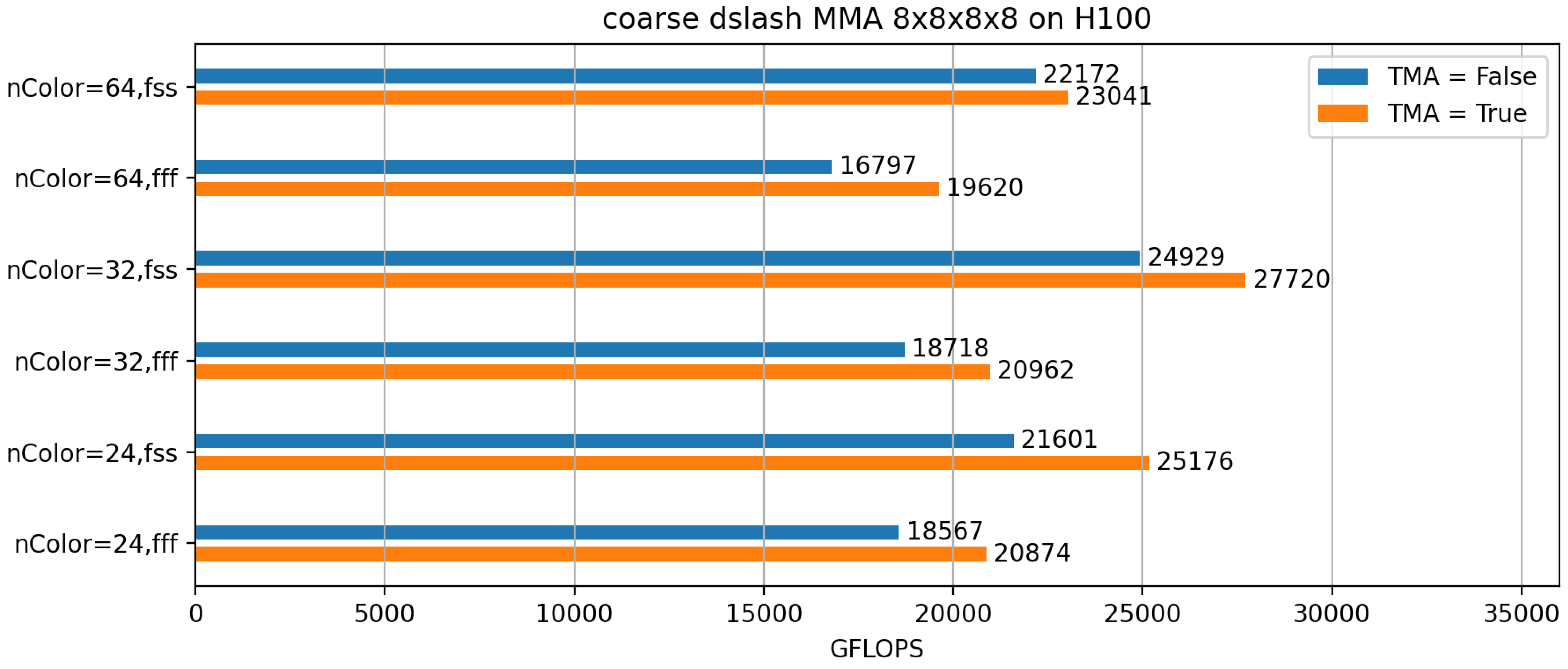
- Multigrid: A class of algorithms that mitigate critical slowing down
  - And that’s nice, but the devil’s really in the details
- Reality: Time (and energy) to solution is the only thing that matters
  - And that’s what must inform the algorithmic and implementation decisions that get made
- Energy Efficiency: Move fewer electrons a shorter distance (and accomplish the same goal)
  - And it’s not just feel-good, time-to-solution often comes along for the ride

NVIDIA

# Backup

# Preliminary TMA Coarse Dslash Numbers

Work by Jiqun Tu

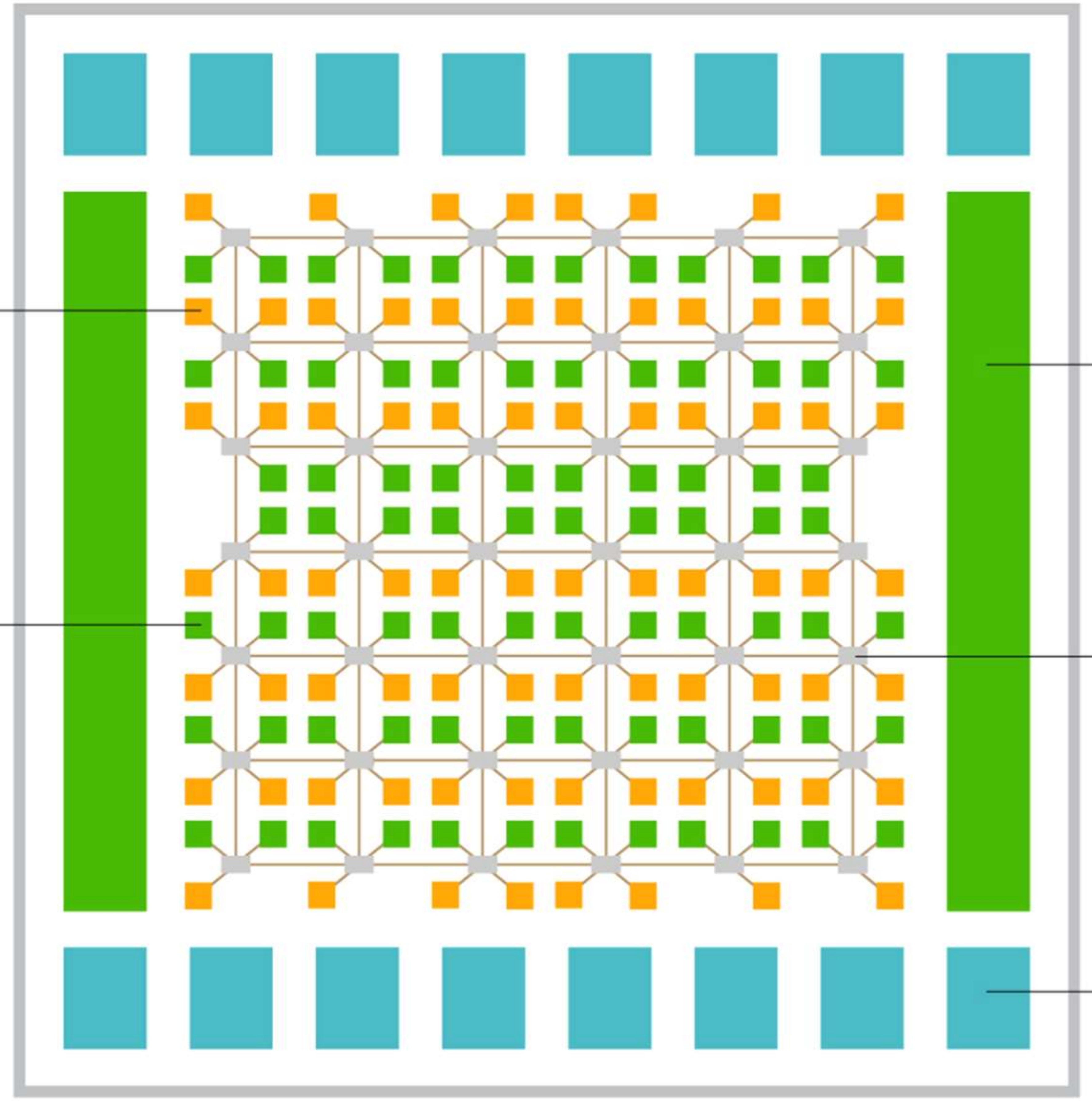



- Current state (December 11, 2024) is available at https://github.com/lattice/quda/pull/1497

# Grace Architecture

The CPU building block of the Grace-Hopper superchip

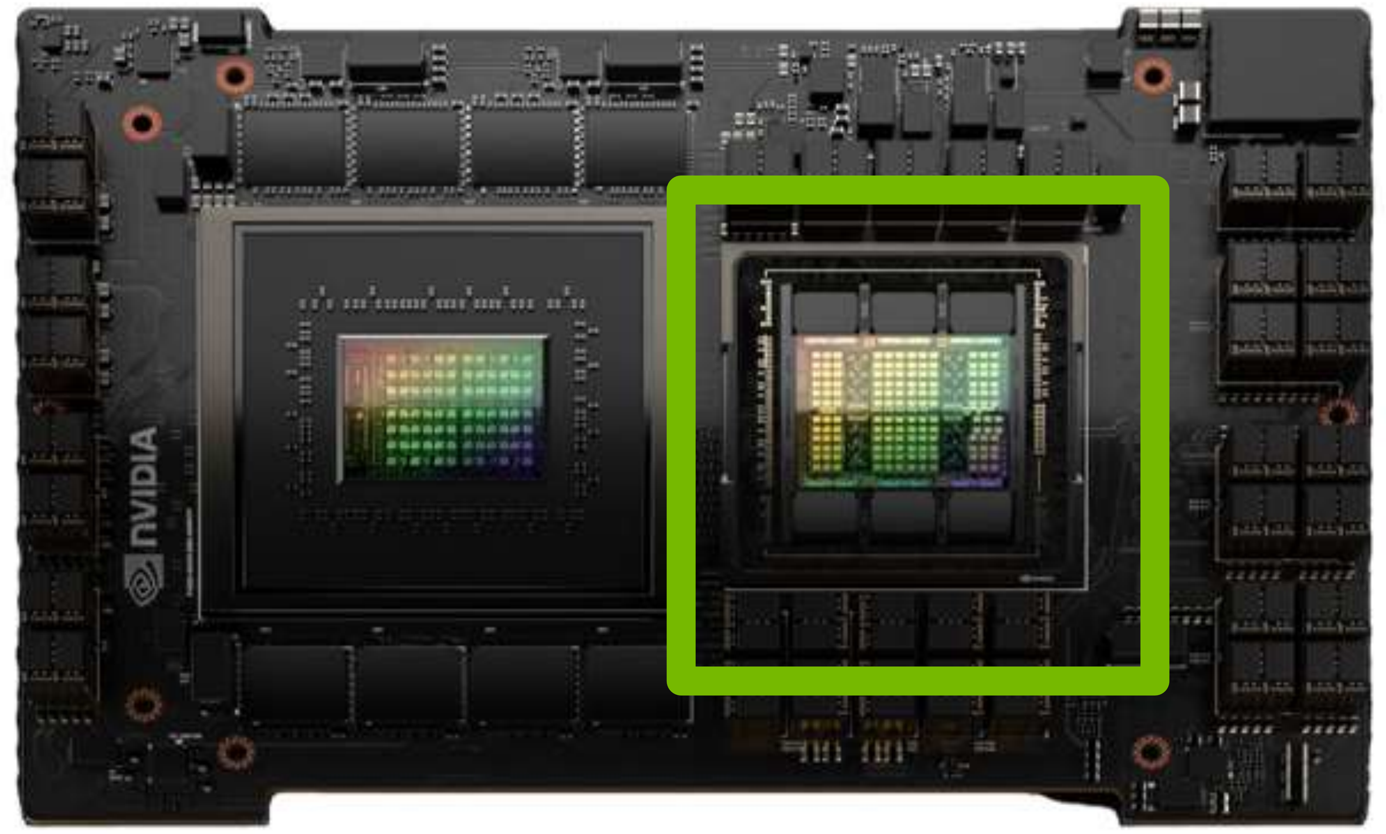
- **High Performance Power Efficient Cores**
  - 72 flagship Arm Neoverse V2 Cores ( Armv9-A )
  - 4x128b SVE2 SIMD units per core ( SVE2 / NEON )
  - 3.16 GHz Base Clock / 2.7 GHz Vector Clock
  - **3.6 FP64 TFLOP/s**
- **Scalable Coherency Fabric**
  - **3.2 TB/s of bisection bandwidth** connects CPU cores, NVLink-C2C, memory, and system IO
- **High-Bandwidth Low-Power Memory**
  - Up to 480 GB of LPDDR5X memory that delivers **up to 500 GB/s of memory bandwidth**
- **Coherent Chip-to-Chip Connections**
  - NVLink-C2C with **900 GB/s raw bidirectional bandwidth** for coherent connection to CPU or GPU
  - ~7x BW that can be delivered by PCIe Gen 5 link
  - Supports up to 4 chip coherency over coherent NVLink

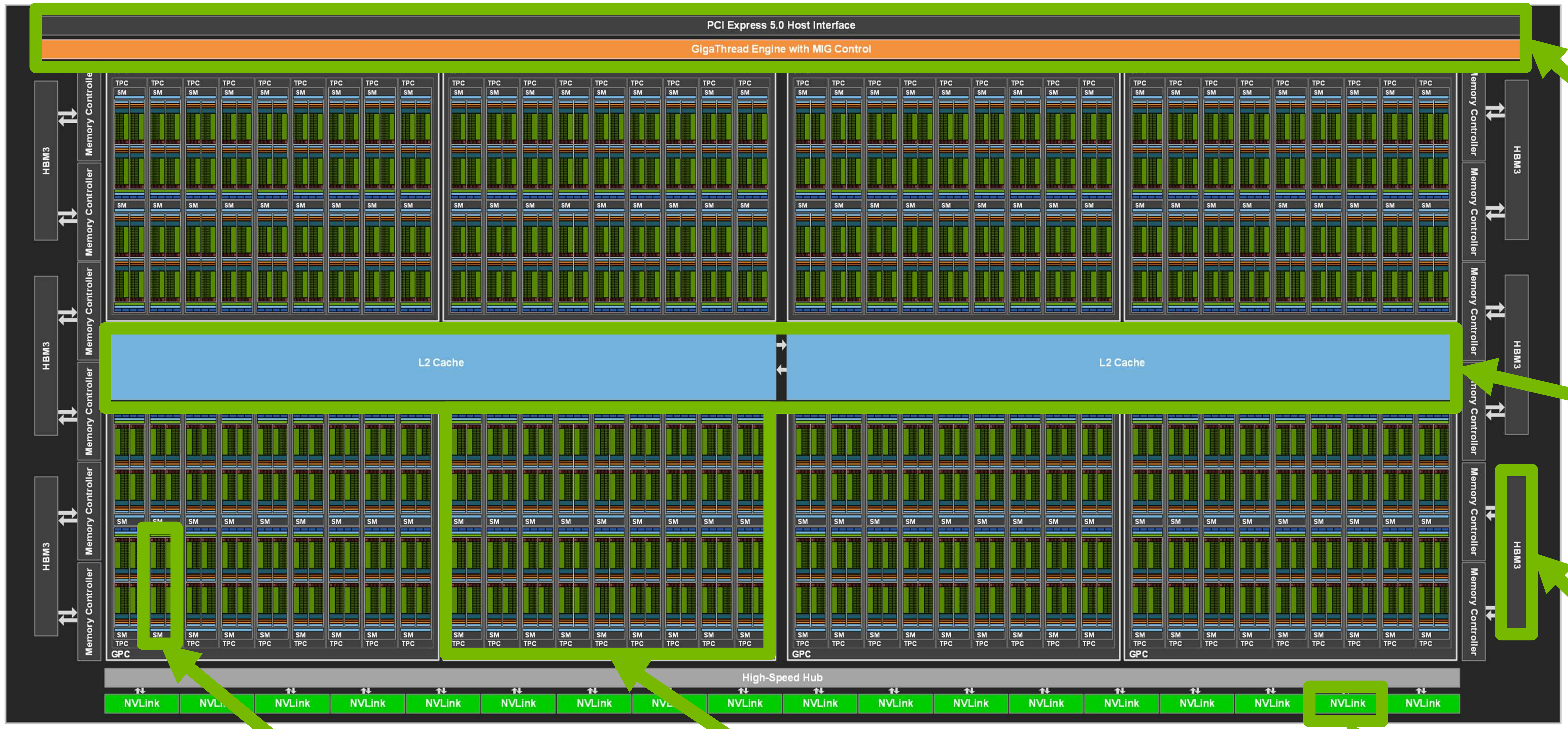



*Example possible fabric topology for illustrative purposes*

# Hopper GPU Architecture

The GPU building block of the Grace-Hopper superchip






2nd Gen Multi-Instance GPU
Confidential Computing
PCIe Gen5

Larger 60 MB L2

96GB HBM3, 4 TB/s bandwidth

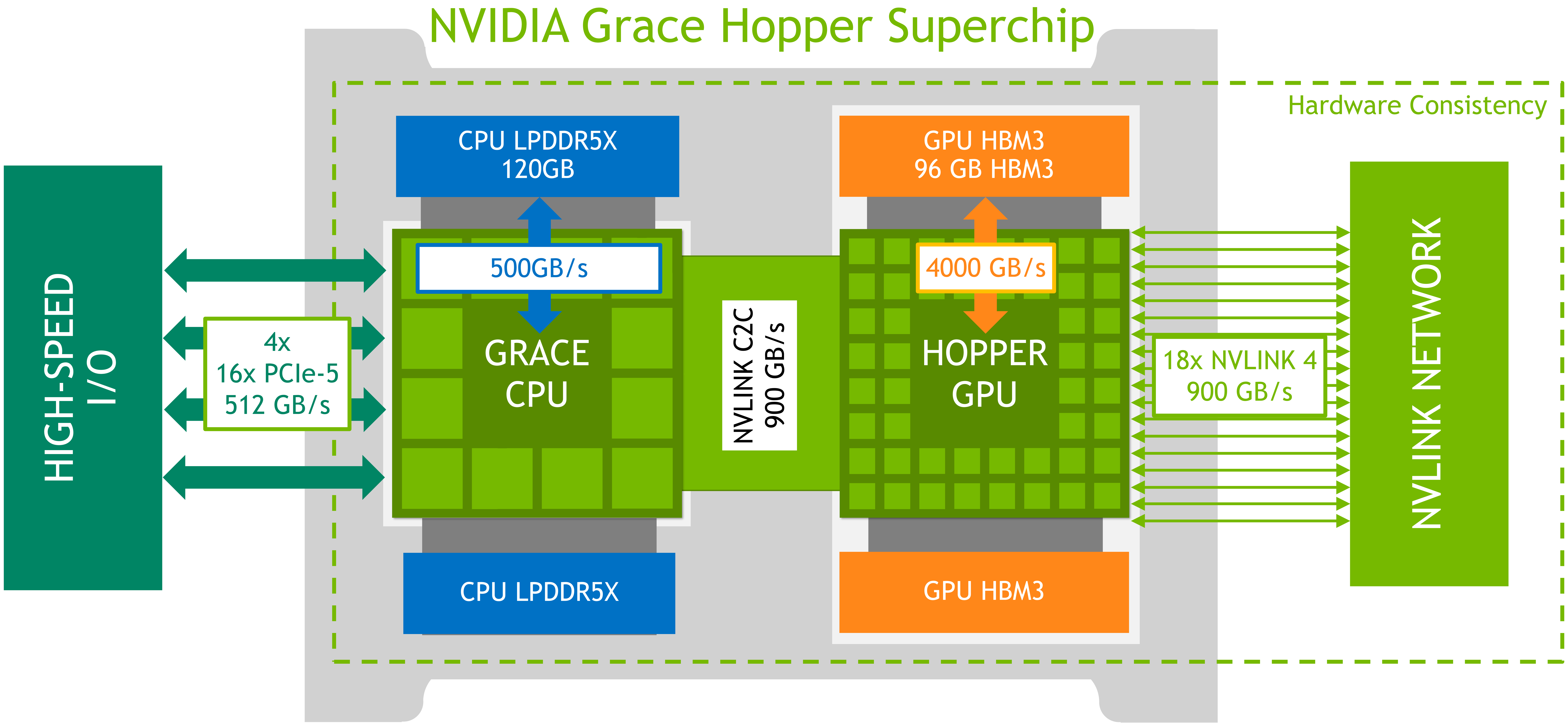
132 SMs
4th Gen Tensor Core

GPU Processing Clusters (GPC) "Thread Block Clusters"

4th Gen NVLink
900 GB/s total bandwidth

# Grace Hopper Superchip

GPU can access CPU memory at CPU memory speeds




https://resources.nvidia.com/en-us-grace-cpu/nvidia-grace-hopper

# Node Architecture of Jupiter (Jedi) Supercomputer

## 4 x Grace-Hopper Superchips

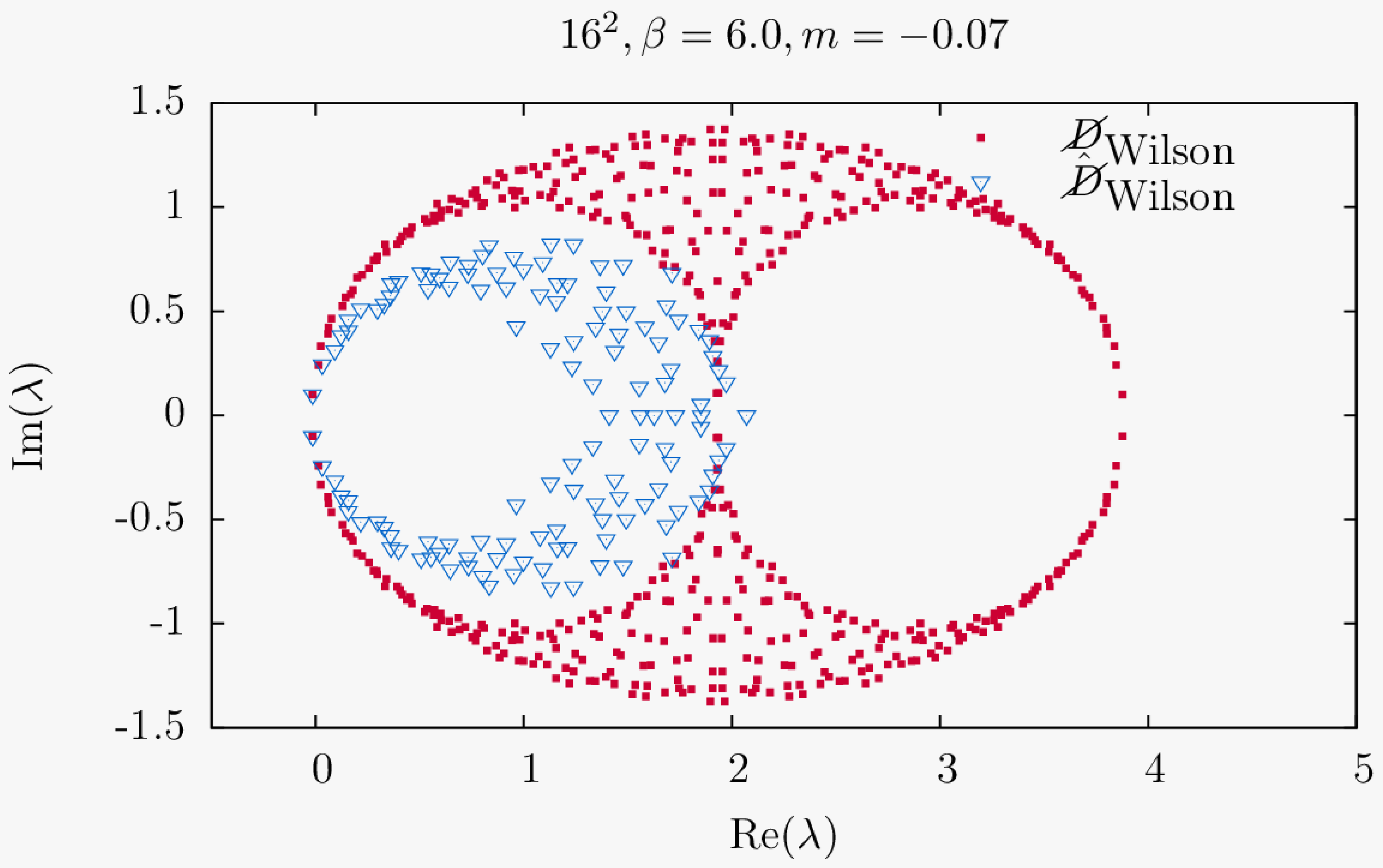


NVIDIA

# Wilson-Clover: the Standard Bearer

Optional subtitle

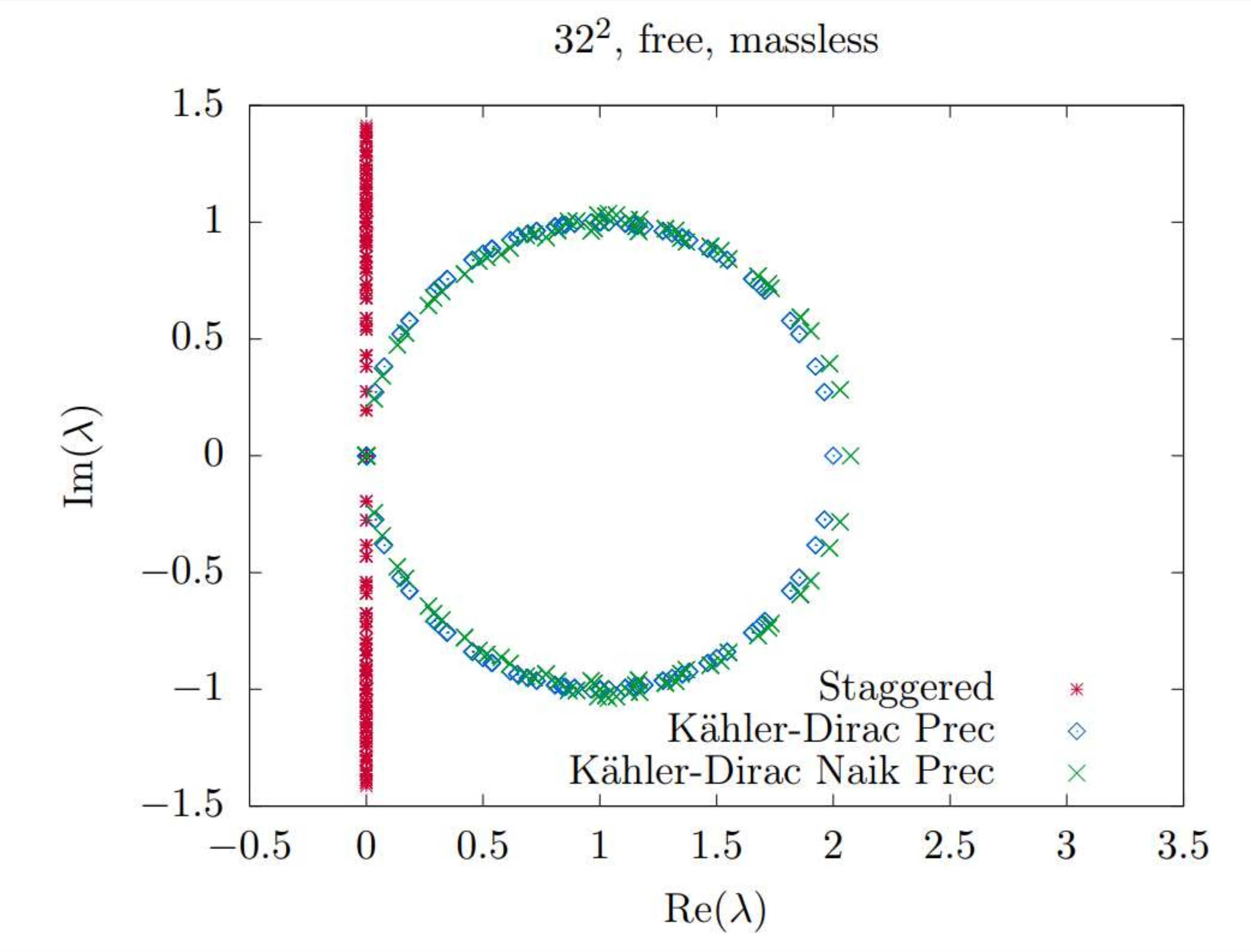
- Brannick et al 2008, Babich et al 2010
- Multiple implementations (QUDA, Grid, DD-$\alpha$AMG, apologies for others I've missed)
- The Wilson operator is a "model" operator
  - Low modes near complex origin
  - High modes gapped from origin in the real direction
- Has been successfully extended to twisted mass, twisted clover
  - Well-documented issue of severely ill-conditioned eigenvalues in coarse operator
  - State-of-the-art Solution: SVD deflation of coarsest level

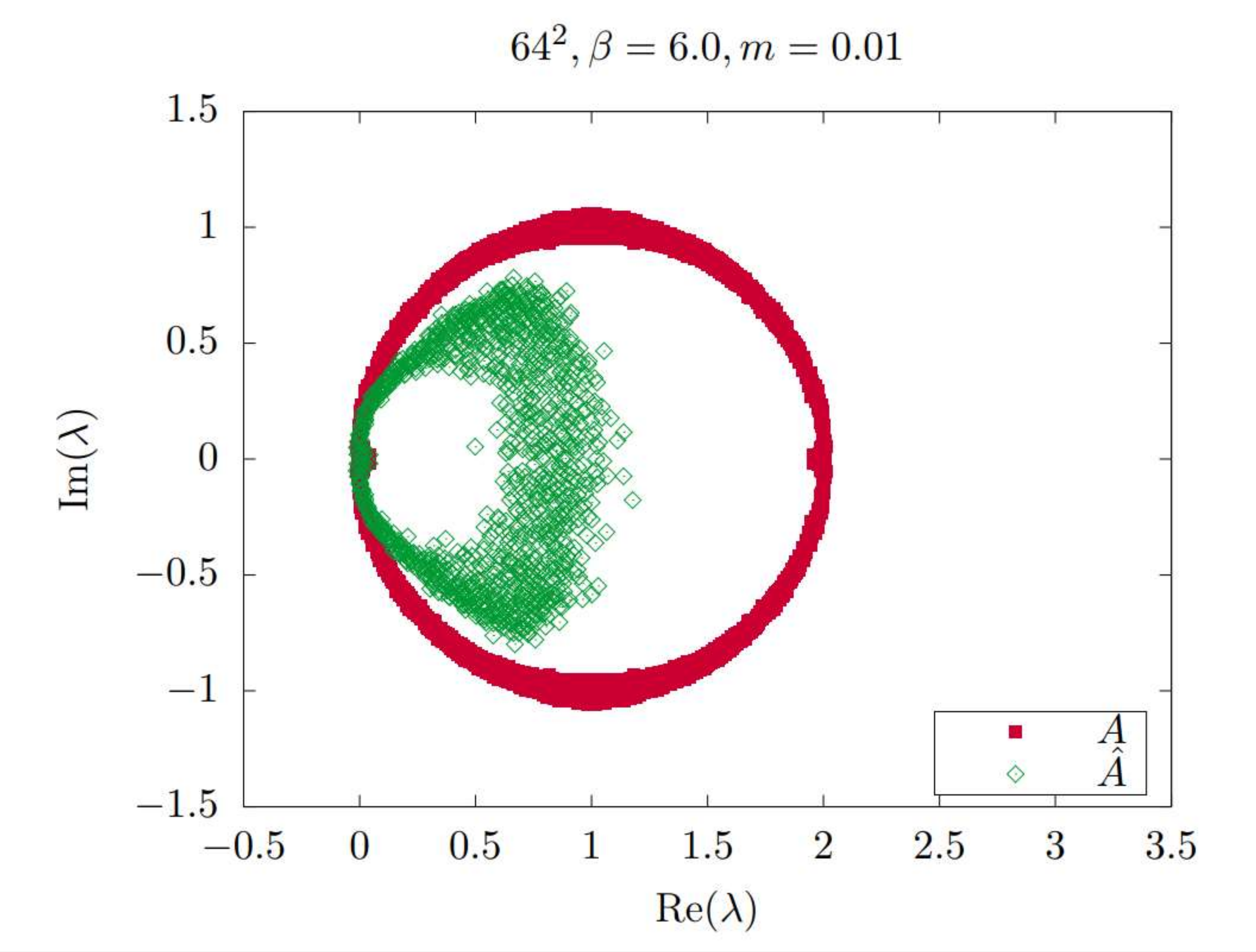


# Staggered Fermions: Kahler-Dirac preconditioning

## Spectral deformations

- 2-d paper: arXiv:1801.07823
- Core idea: spectral deformation by Kahler-Dirac structure
  - Each $2^d$ hypercube of staggered dof = one lattice Kahler-Dirac fermion
  - Block-precondition by this $2^d$ structure
- Deforms anti-Hermitian indefinite spectrum into (roughly) circular spectrum
- Carries similar spectral properties as Wilson-clover after coarsening
- Implemented in QUDA, exposed in MILC

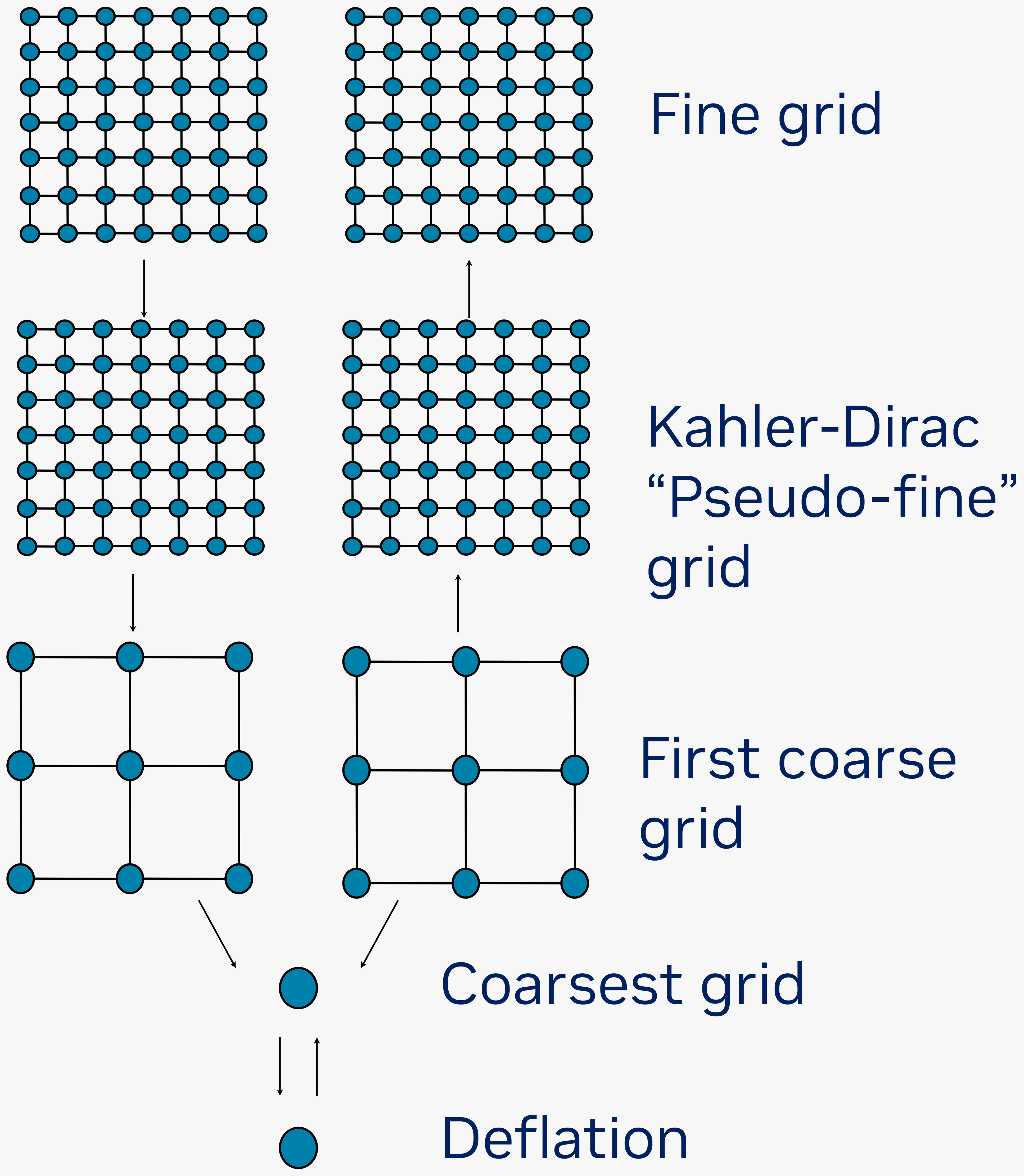


# Five-level Algorithm

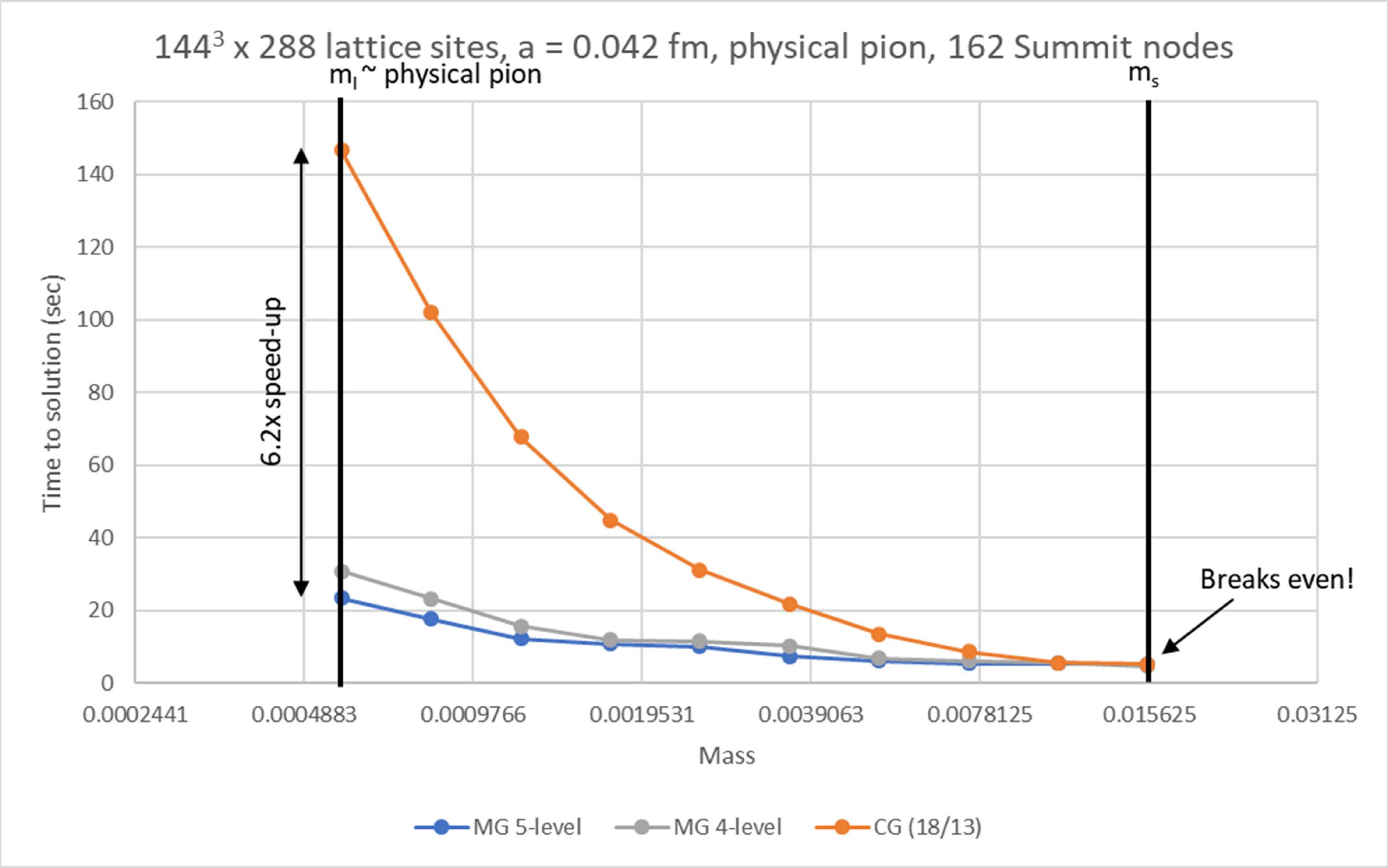
## Spectral deformations

- Fine level: outer staggered/HISQ solver
- Second level: “pseudo-fine” block preconditioned level
  - Unitary transformation for staggered operator (before block preconditioning)
  - HISQ operator: drop Naik term, corrected on fine level by smoother
- Traditional MG aggregation from there:
  - Third level: Nc = 64 x Nspin = 2
  - Fourth level: Nc = 96 x Nspin = 2
  - Fifth level: Deflation



# HISQ MG Algorithm on Summit

FIXME

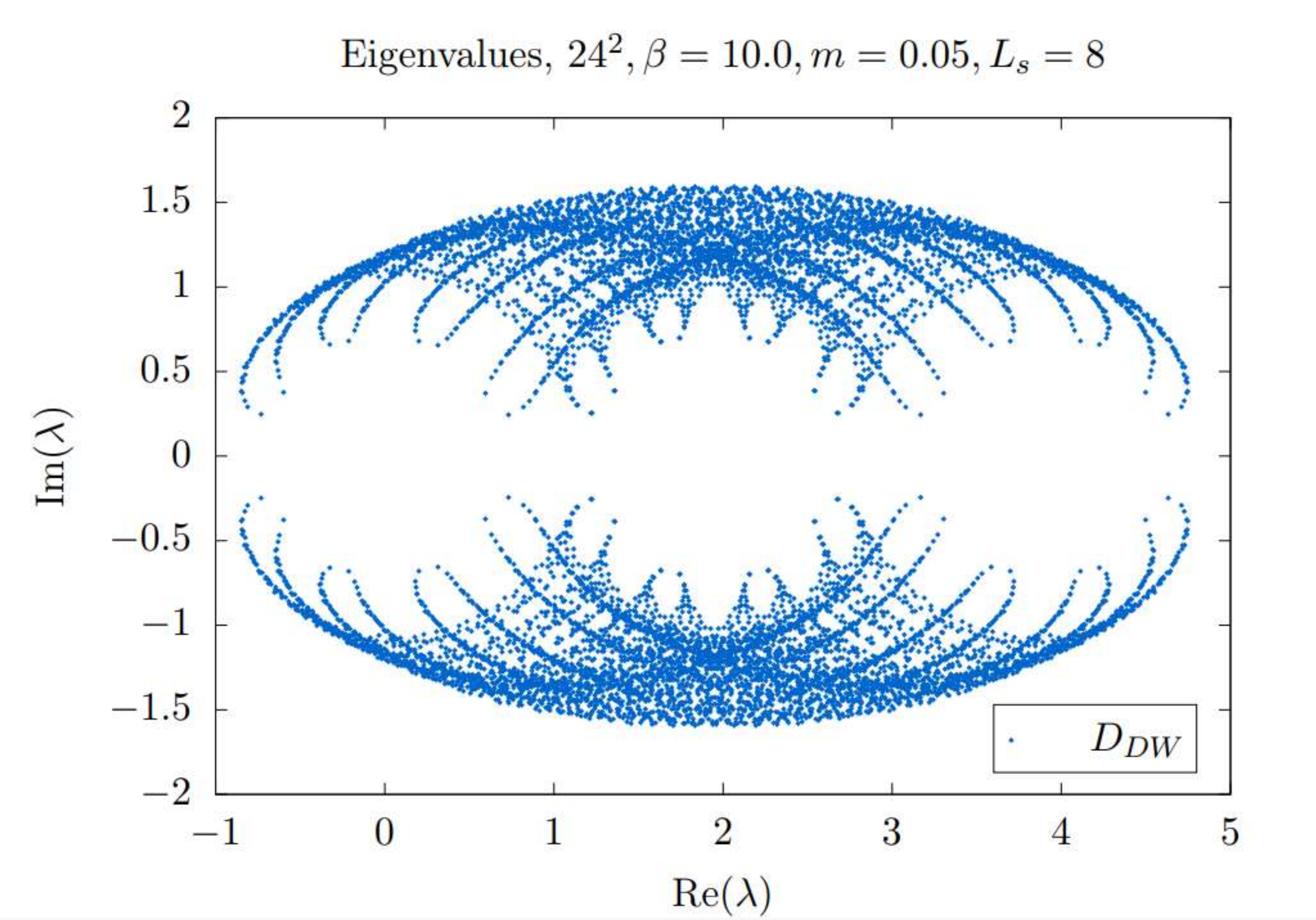



Physical pion mass configuration courtesy of Carleton DeTar (MILC collaboration)

# Chiral Fermions: Domain Wall

## Spectral deformations

- The challenge: maximally indefinite spectrum
  - (Heavily) violates half-plane condition, subverts rates of convergence proofs
- Methods on the normal operator:
  - Cohen et al 2011, Boyle 2014
  - **Latest & Greatest from Peter: MG-Preconditioned Block CG, arXiv:2409.03904 + previous talk!**
- Other recent work:
  - Comparison of Domain Wall Fermion Multigrid Methods (Boyle & Yamaguchi, 2021, arXiv: 2103.05034)
  - Approximate Pauli-Villars preconditioned operator in 2-d in Brower et al, arXiv:2004.07732 (demonstrated in 4-d by Boyle)
  - Four-level hierarchically deflated conjugate residual (HDCR) on Hermitian indefinite operator in Grid, (Boyle, arXiv:1611.06944)

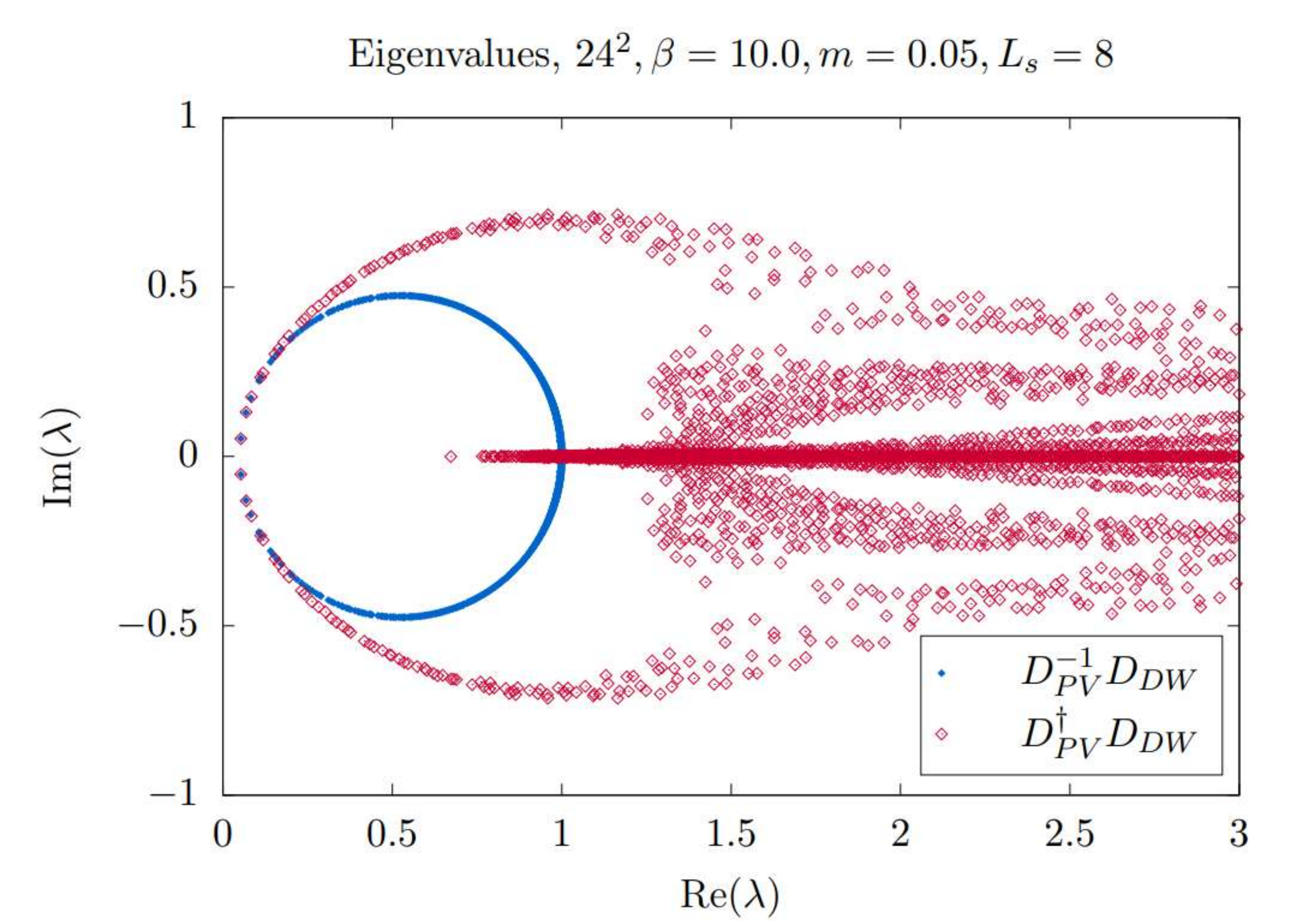



Note: Spectrum for 2-d Schwinger model;
4-d QCD has 5 “eyes” in the burger plot

# Approximate Pauli-Villars Preconditioning

## Spectral deformations

- Demonstrated in 2-d: arXiv:2004.07732, 4-d by Boyle in 29 hours
- Motivator is multigrid on DPV-1 Ddwf (effective overlap)
- Three steps:
  - Replace DPV-1 with DPV † --- still obeys half-plane condition
  - Perform a Galerkin coarsening of the 4-d operator on each slice, separately preconditioning Ddwf and DPV
  - Only prolong/restrict on chiral boundaries
- Idea of coarsening Wilson kernel (equiv. Hermitian kernel) applies to all formulations, overlap
- Implemented in Grid in 29 hours, Implementation in QUDA a WIP




$$D_{DW}(m)_{s's} = \begin{bmatrix} D_W(M_5)+1 & P_- & 0 & \cdots & -mP_+ \\ P_+ & D_W(M_5)+1 & P_- & \cdots & 0 \\ 0 & P_+ & D_W(M_5)+1 & \cdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & P_- \\ -mP_- & 0 & \cdots & P_+ & D_W(M_5)+1 \end{bmatrix}$$

$$\widehat{D}_{DW}(m) = \begin{bmatrix} \widehat{D}_W(M_5)+1 & \widehat{P}_- & 0 & \cdots & -m\widehat{P}_+ \\ \widehat{P}_+ & \widehat{D}_W(M_5)+1 & \widehat{P}_- & \cdots & 0 \\ 0 & \widehat{P}_+ & \widehat{D}_W(M_5)+1 & \cdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \widehat{P}_- \\ -m\widehat{P}_- & 0 & \cdots & \widehat{P}_+ & \widehat{D}_W(M_5)+1 \end{bmatrix}$$

$$\widehat{r}_s = \begin{cases} \mathbb{P}^\dagger r_1 & \text{for} \quad s = 1 \\ 0 & \text{for} \quad s > 1 \end{cases}$$