System architectures with adaptive accelerators for genomics

Sang-Woo Jun

Assistant Professor, Department of Computer Science

University of California, Irvine

UCI

System architectures with adaptive accelerators for genomics

Sang-Woo Jun

Assistant Professor, Department of Computer Science

University of California, Irvine

ARDA

UCI

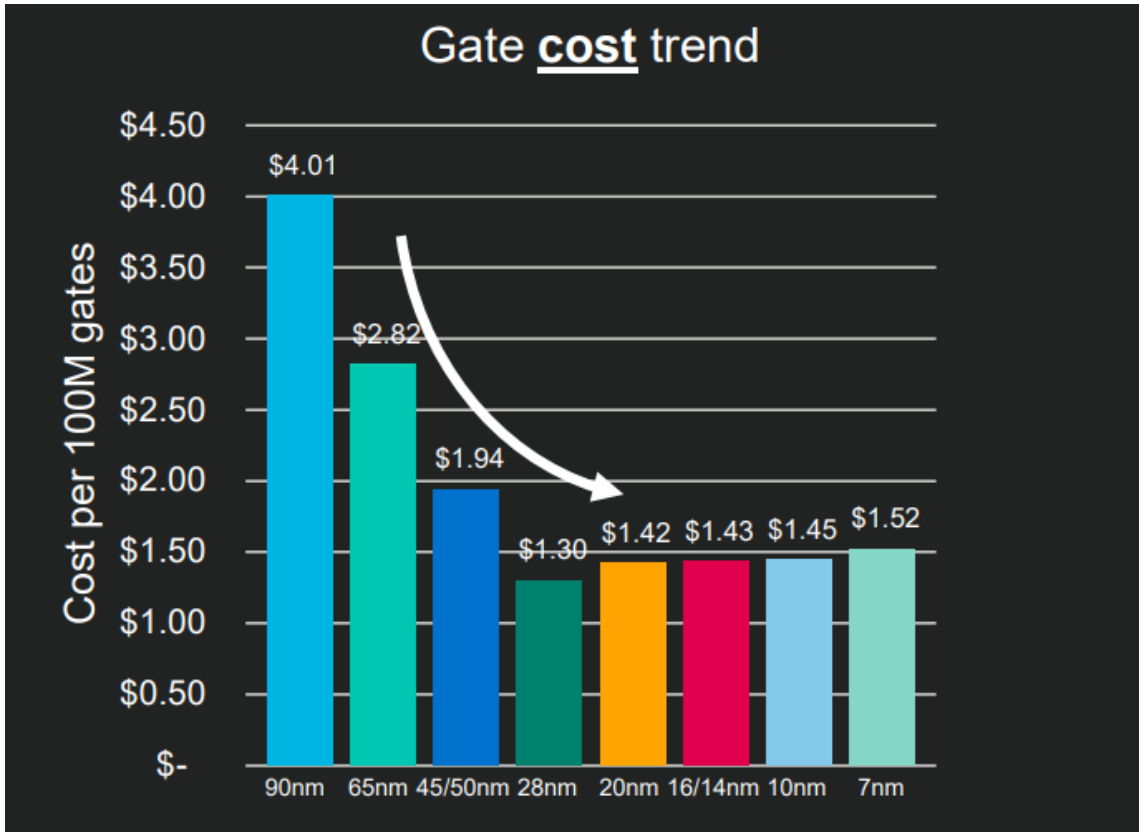
Working to Bridge the Silos…

Photo credit: https://unsplash.com/@waldemarbrandt67w

The Usual Doom and Gloom

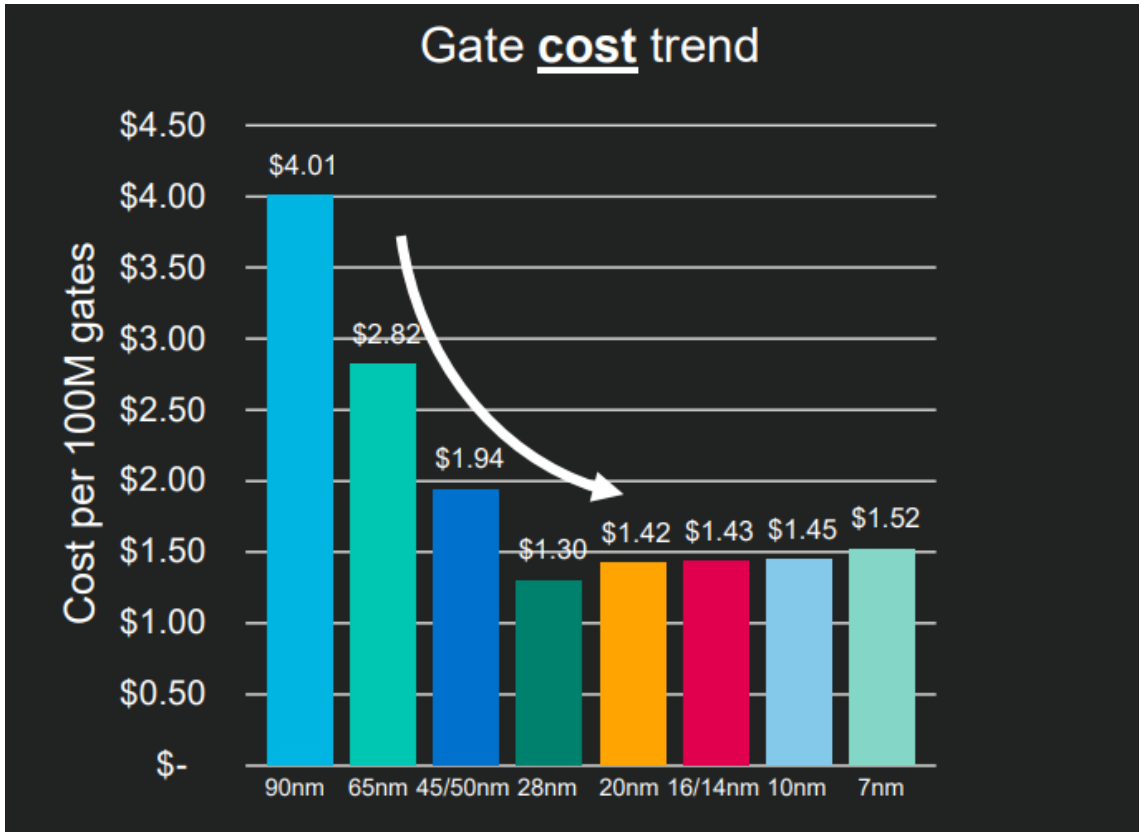
[3] Database Architects, “The Great CPU Stagnation” 2023
[4] Marvell 2020 Investor day – Slide 43

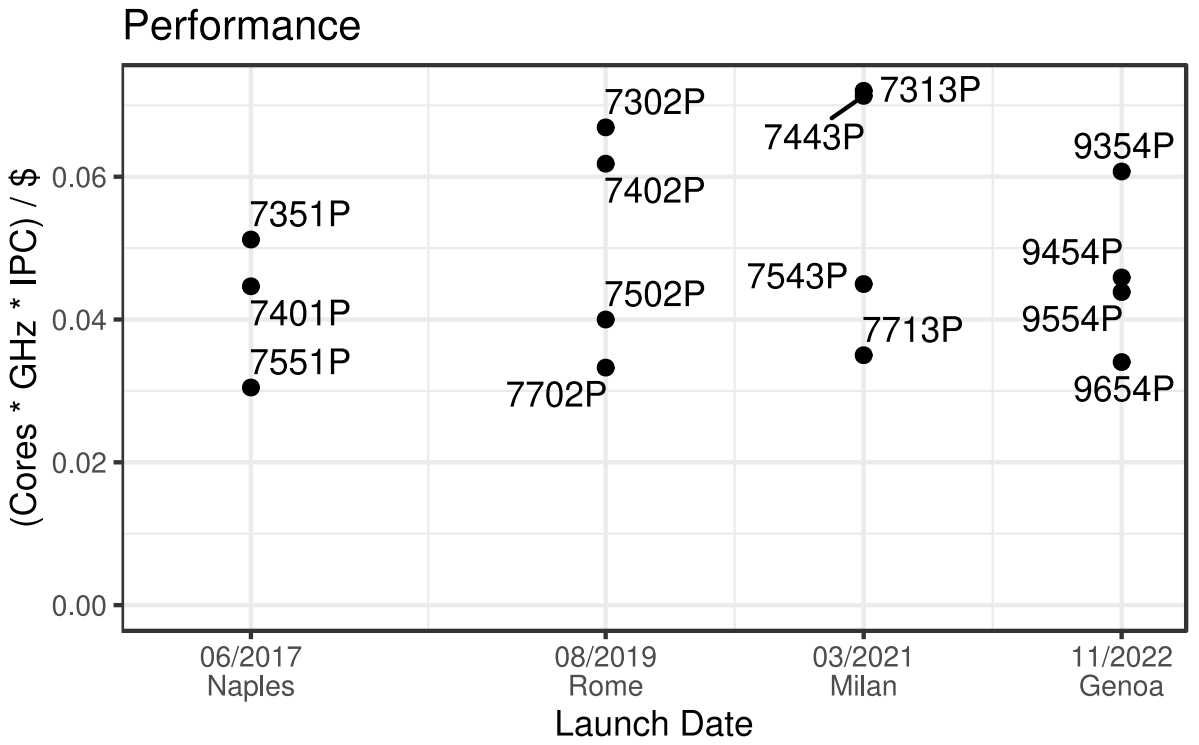
The Usual Doom and Gloom



[3] Database Architects, "The Great CPU Stagnation" 2023
[4] Marvell 2020 Investor day – Slide 43

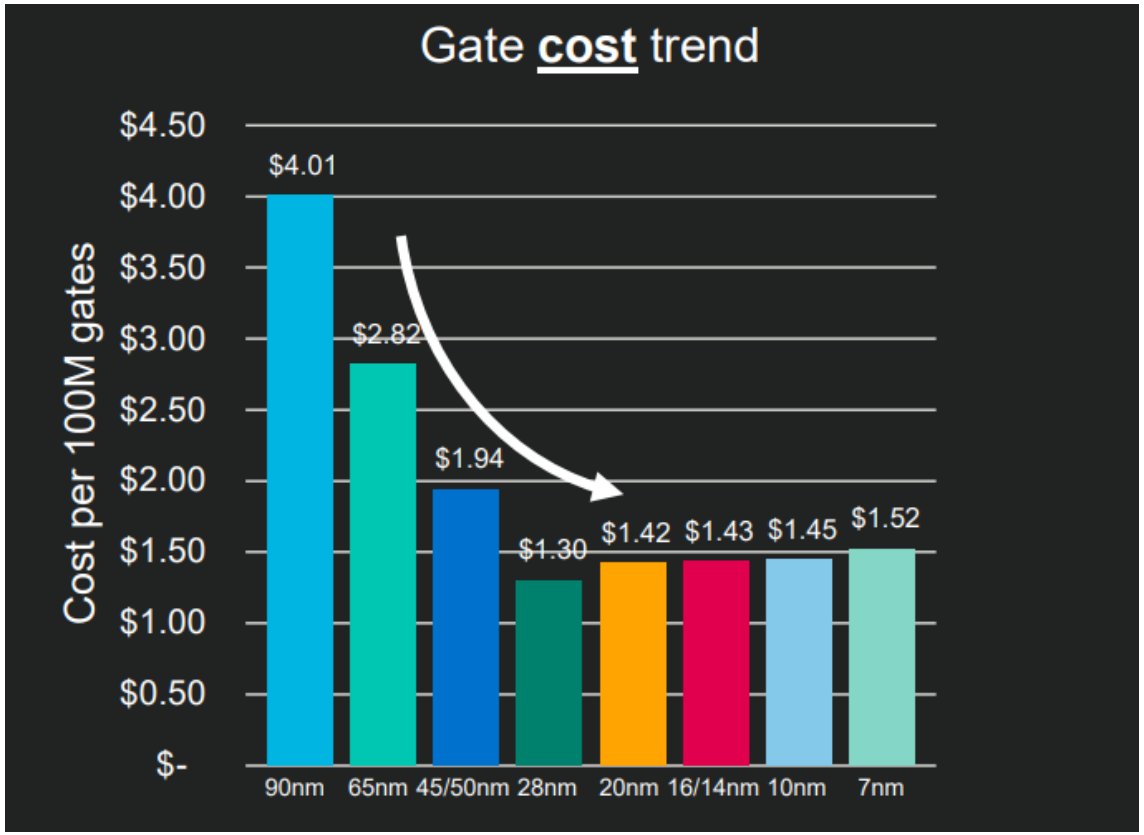
The Usual Doom and Gloom

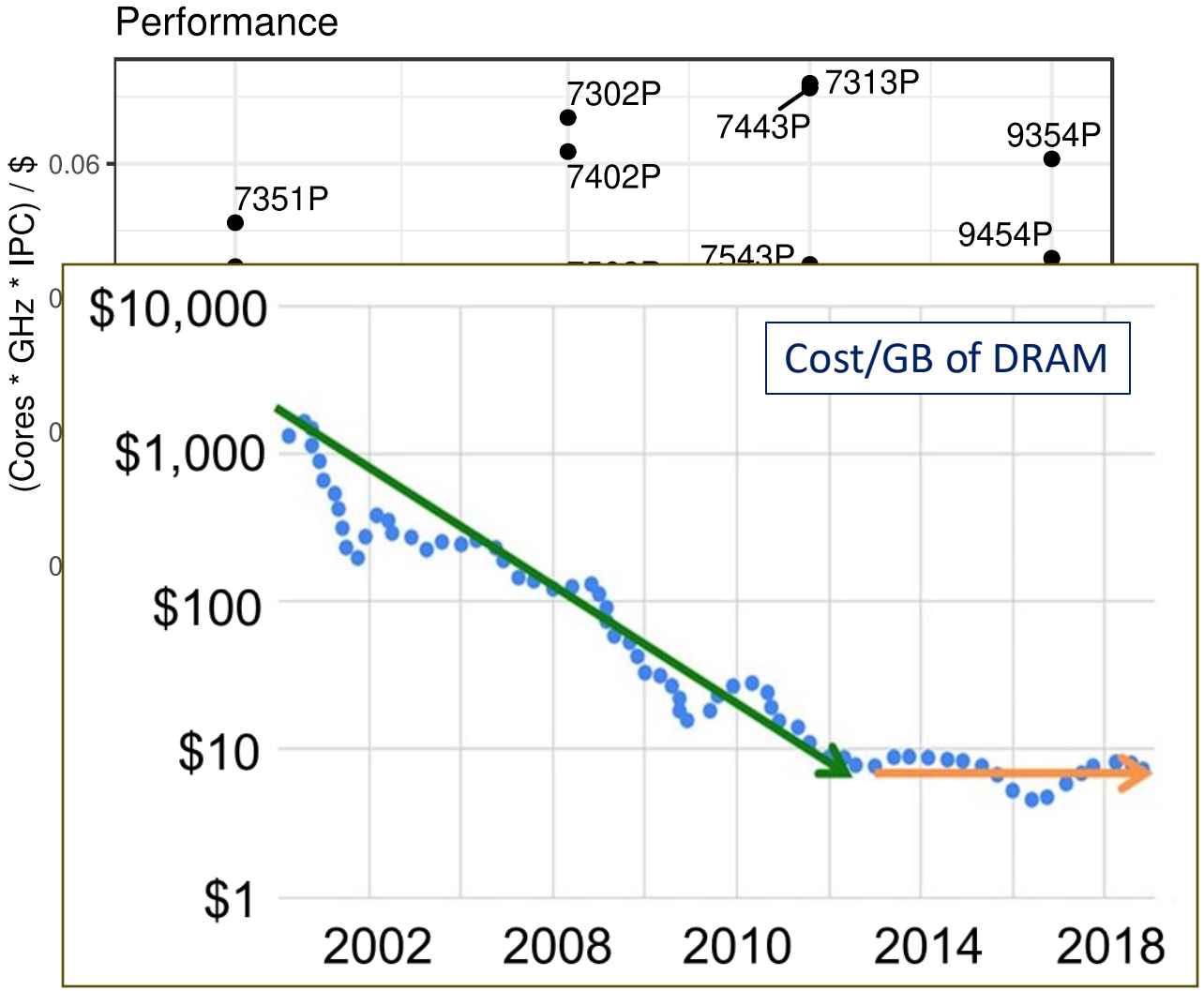




[3] Database Architects, "The Great CPU Stagnation" 2023
[4] Marvell 2020 Investor day – Slide 43

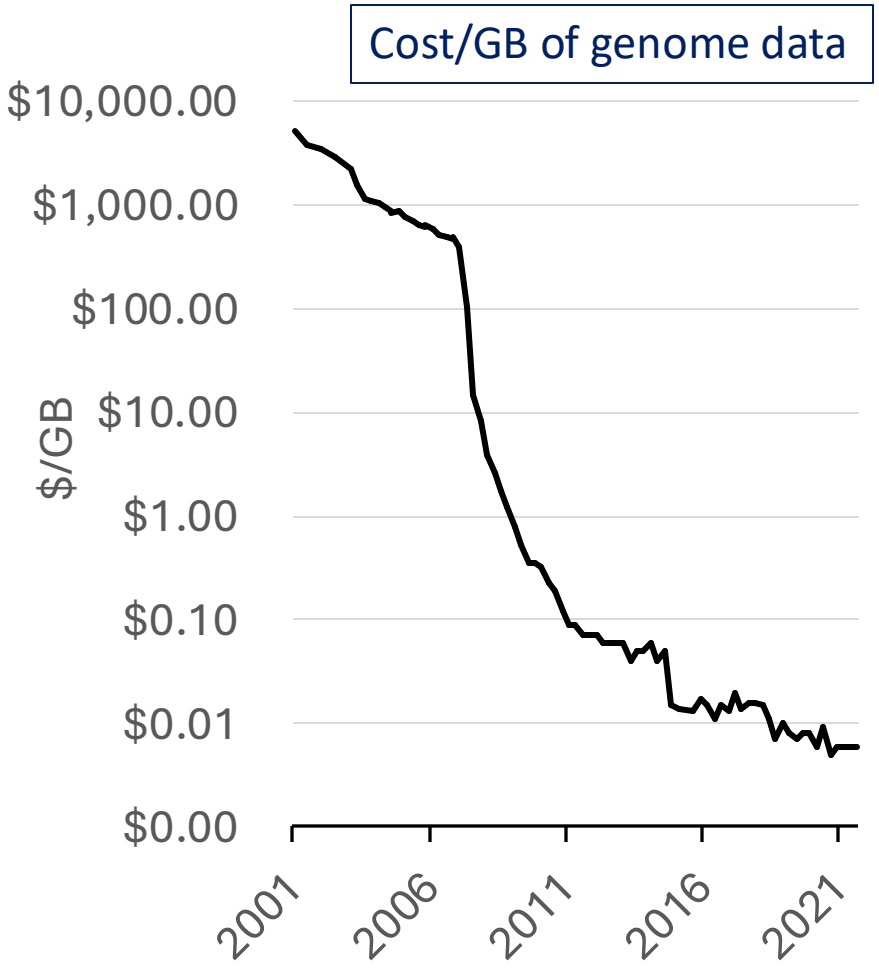
The Usual Doom and Gloom

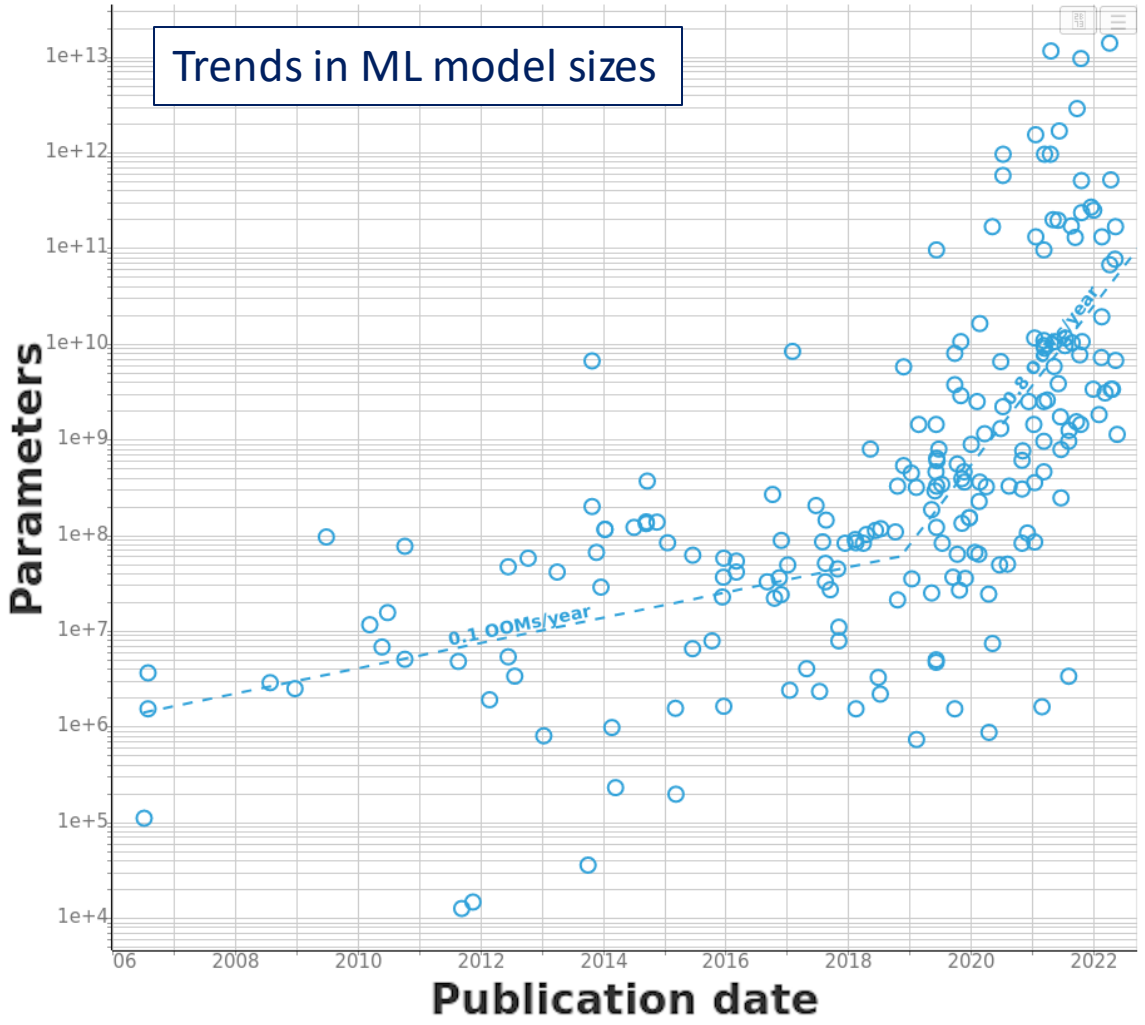




[3] Database Architects, "The Great CPU Stagnation" 2023
[4] Marvell 2020 Investor day – Slide 43

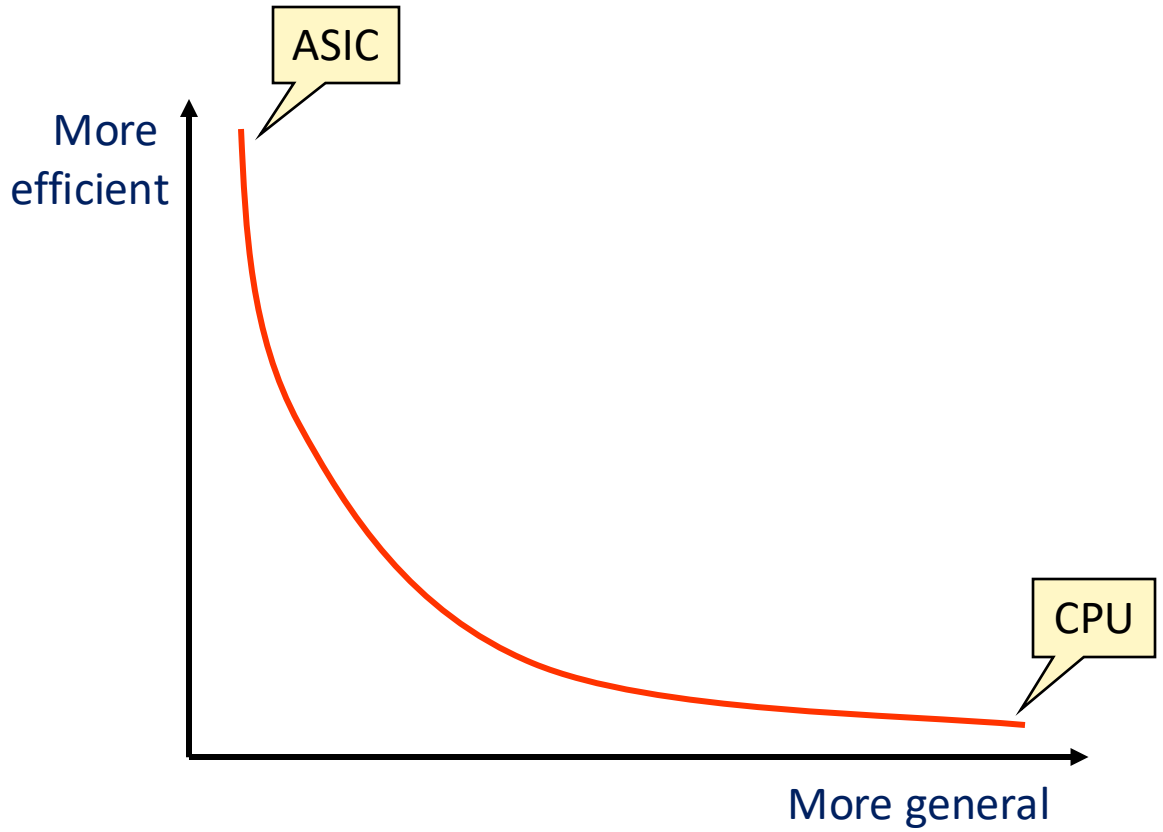
But, We Need More Performance!



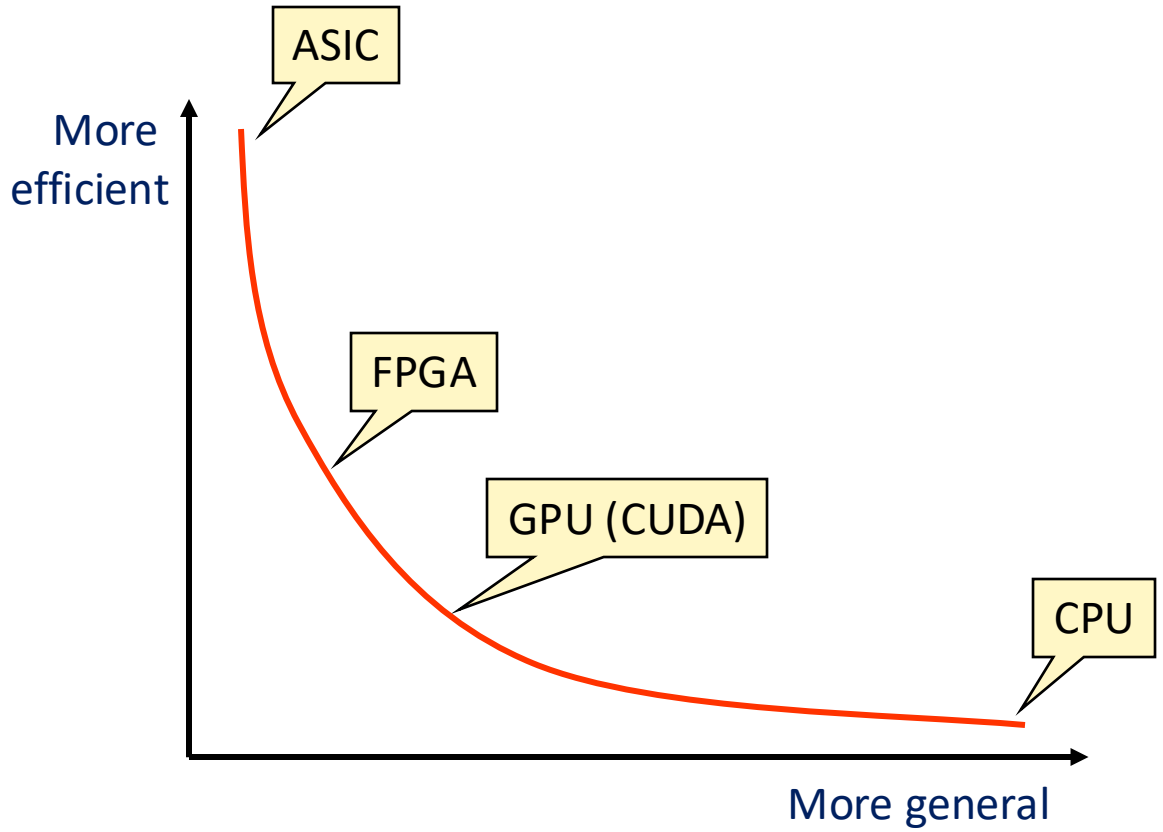


https://epoch.ai/blog/machine-learning-model-sizes-and-the-parameter-gap

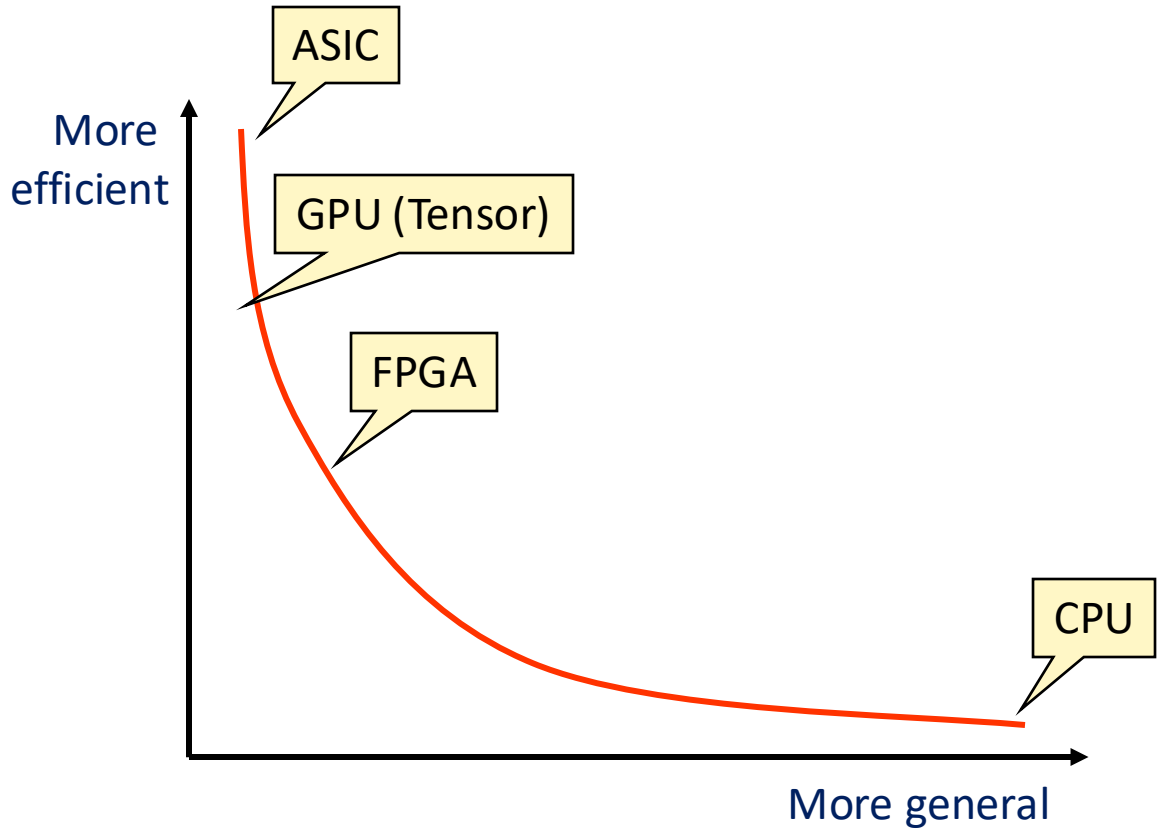
Specialization for Performance & Efficiency



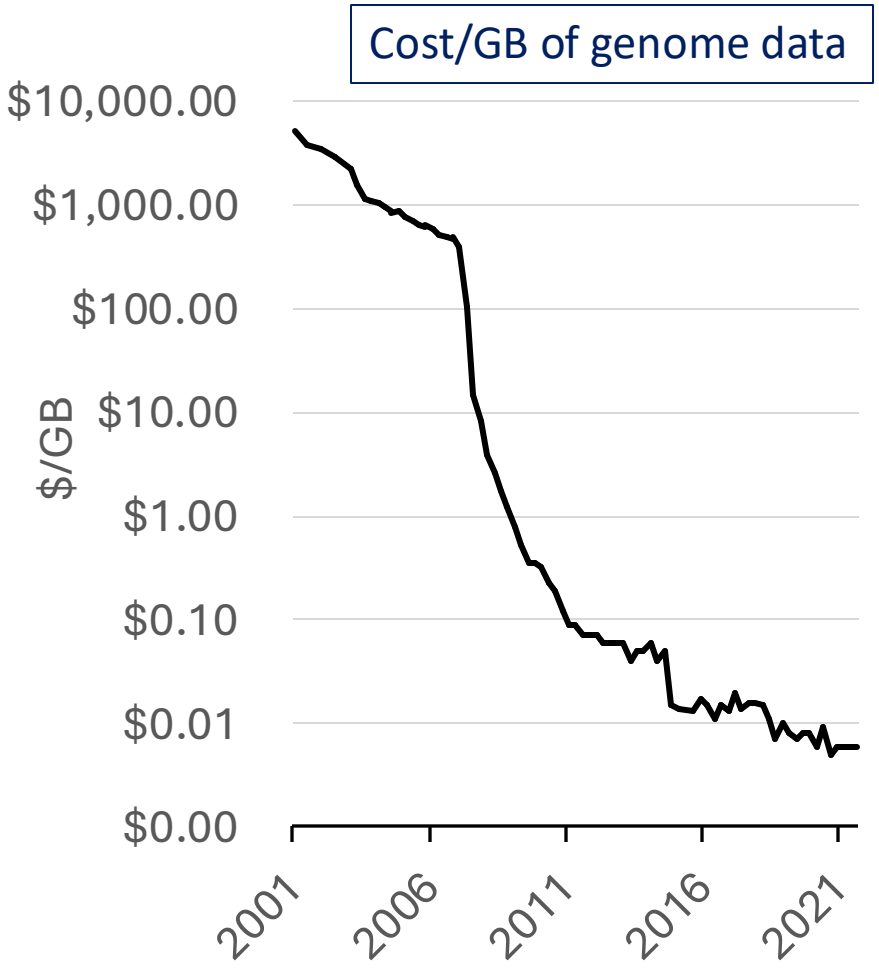
Specialization for Performance & Efficiency

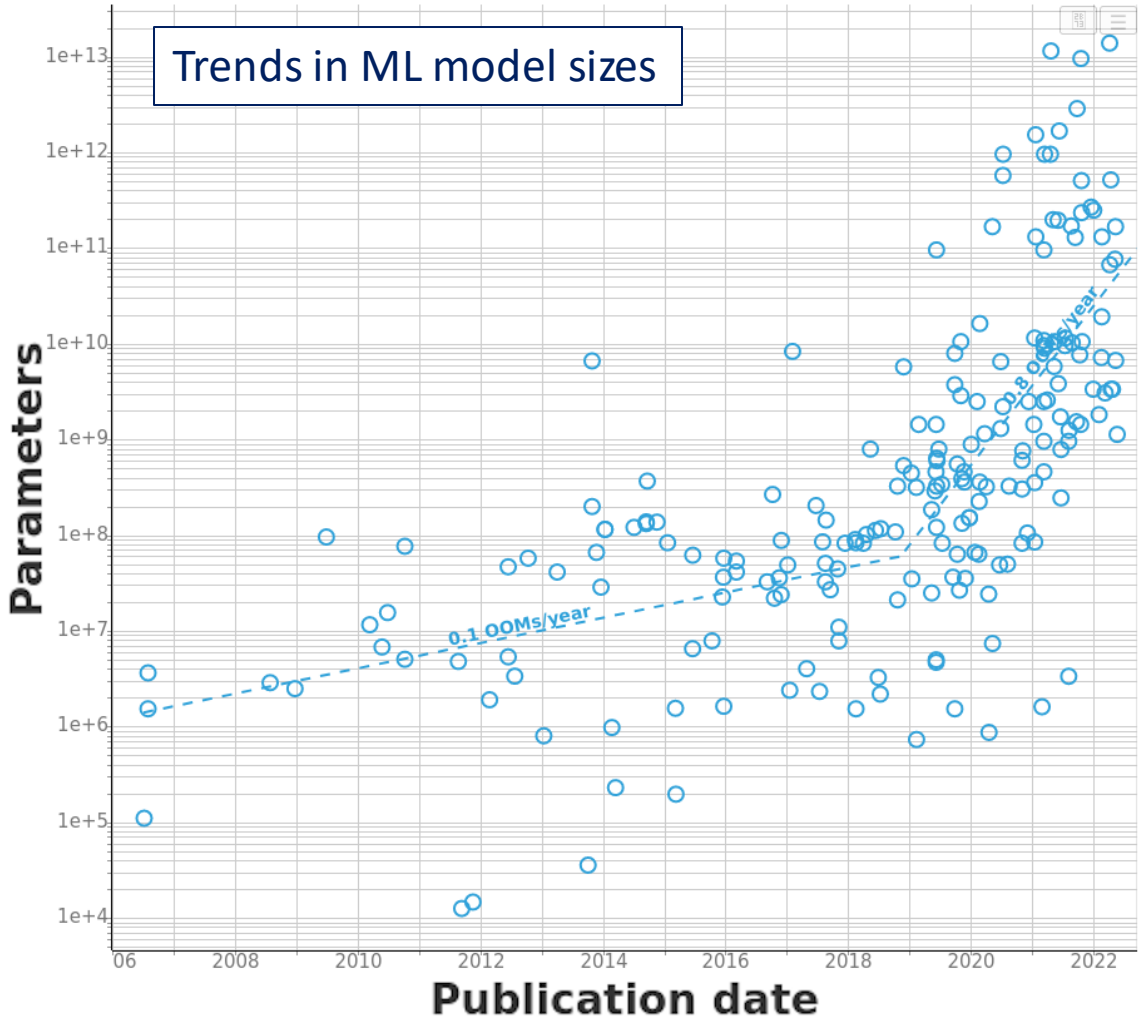


Specialization for Performance & Efficiency



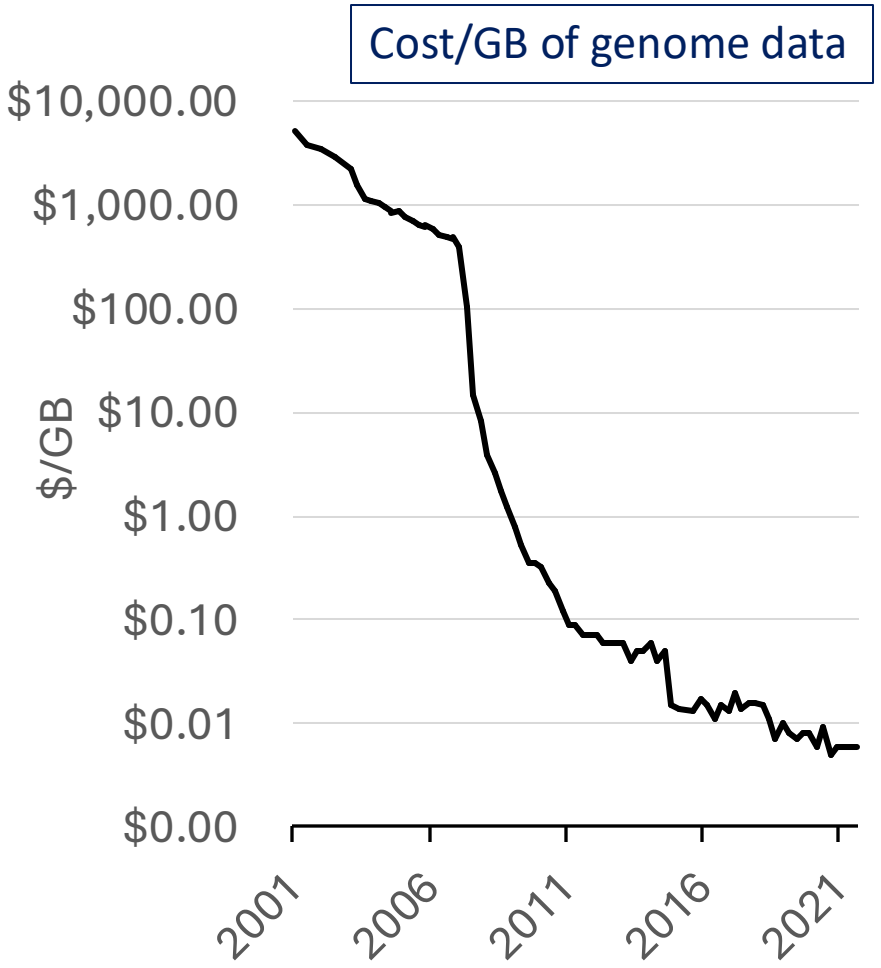
But, We Need More Performance!

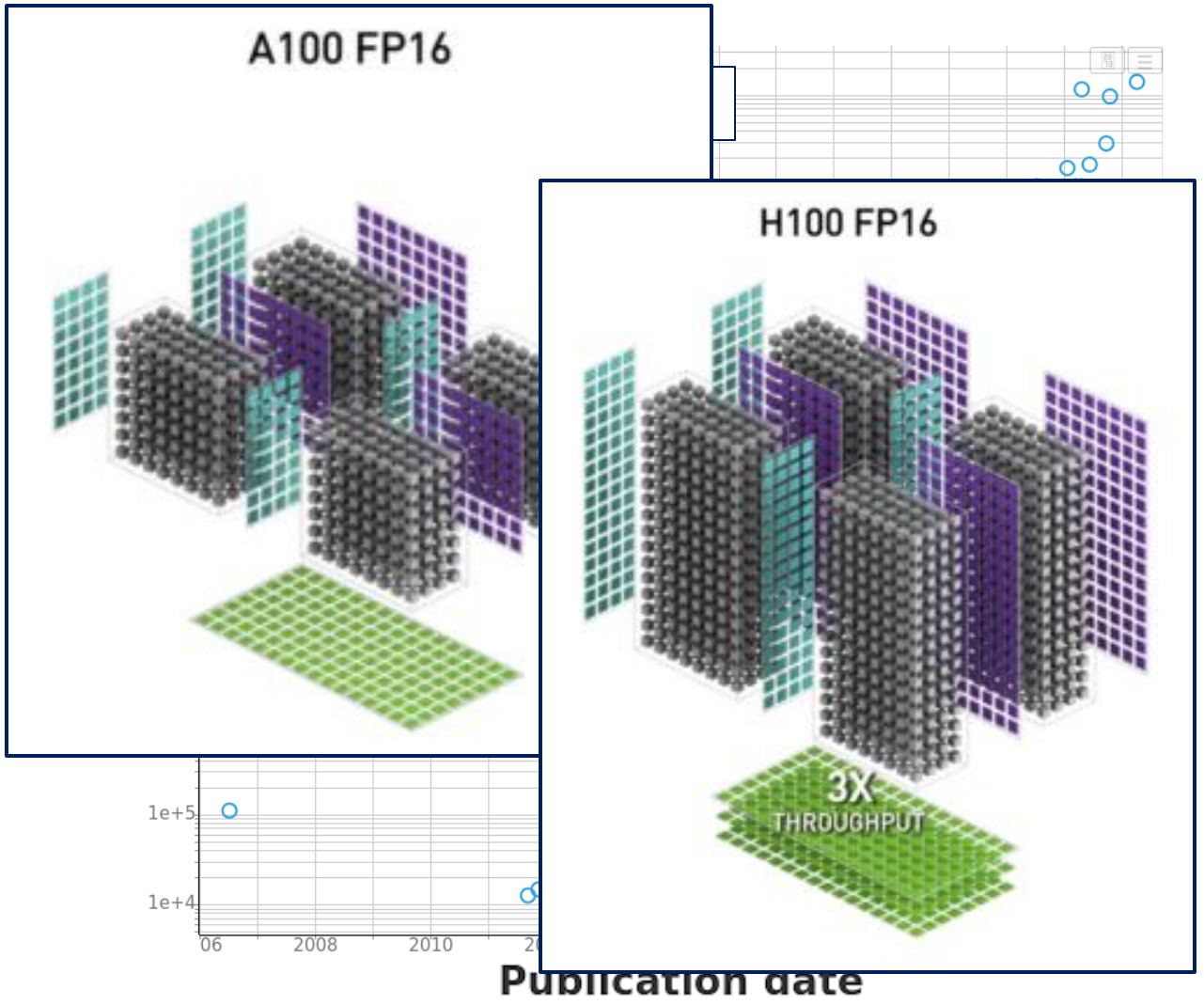




https://epoch.ai/blog/machine-learning-model-sizes-and-the-parameter-gap

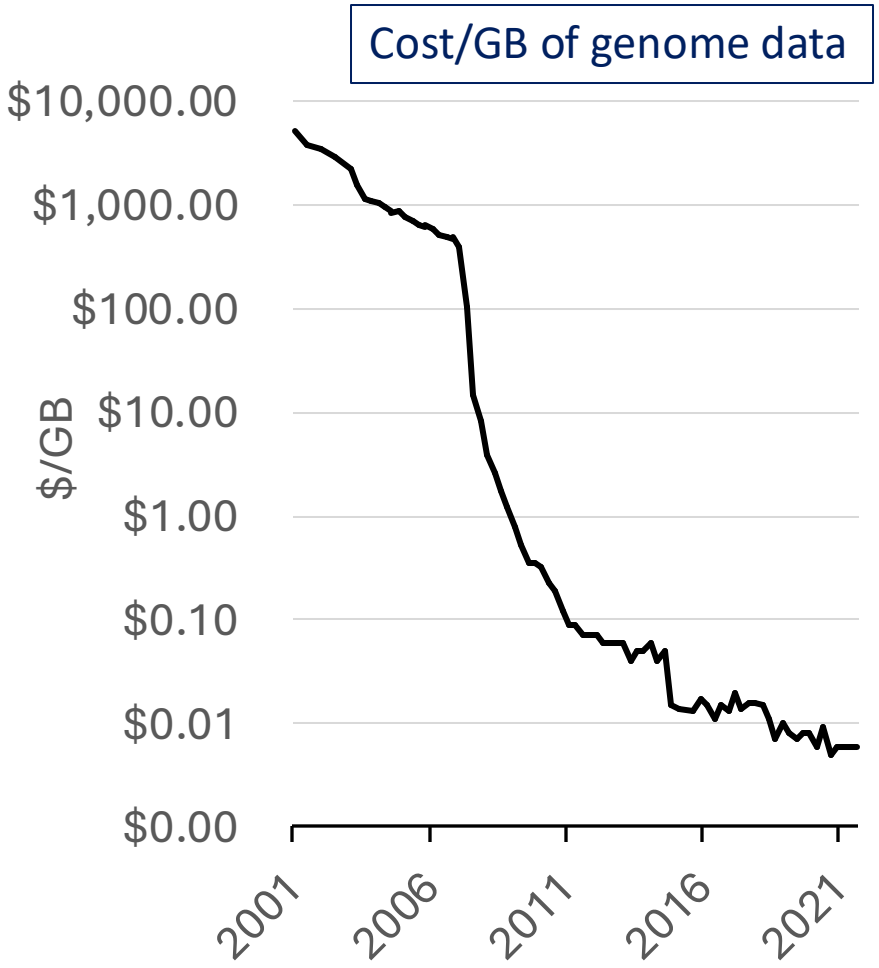
But, We Need More Performance!

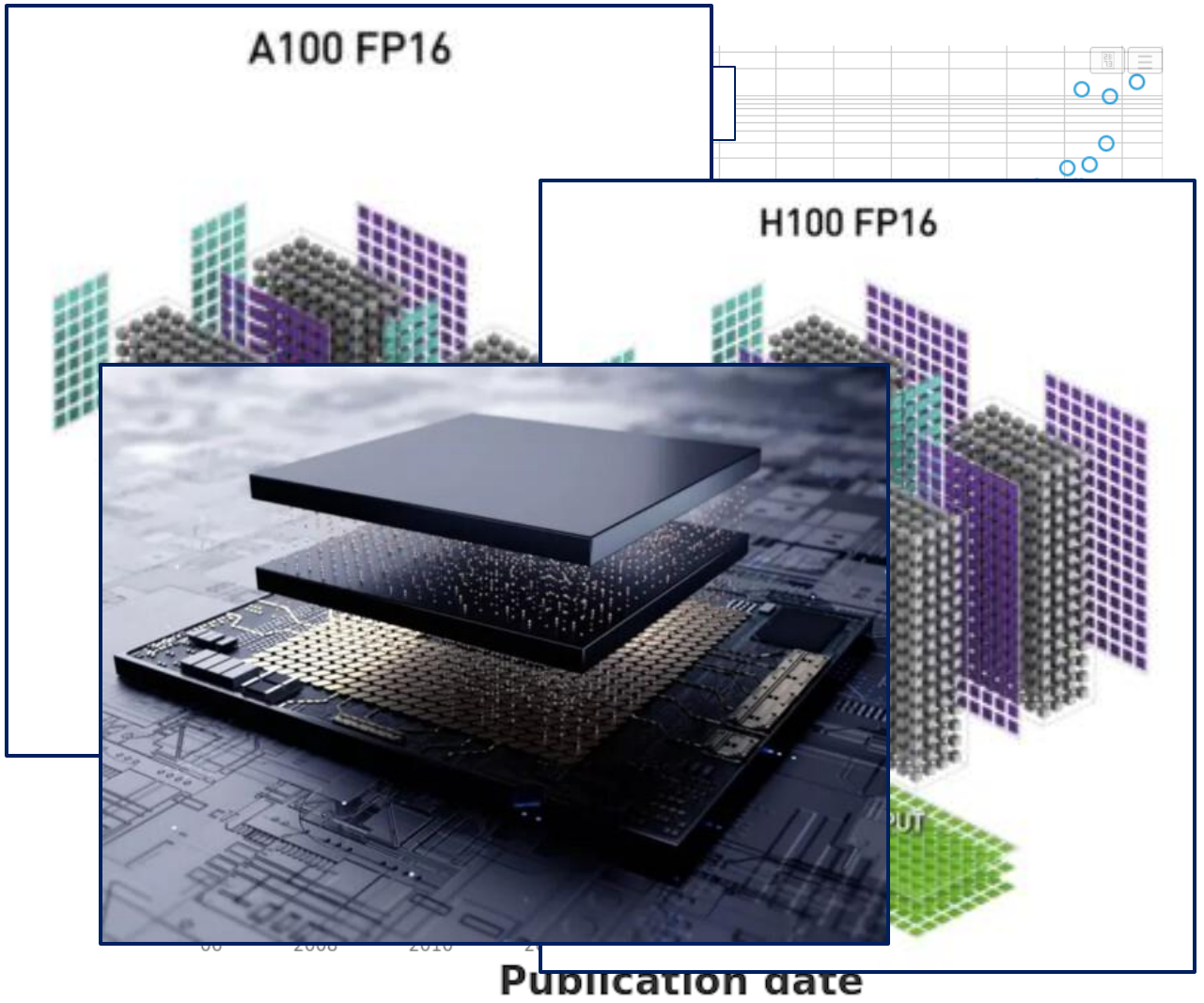




https://epoch.ai/blog/machine-learning-model-sizes-and-the-parameter-gap

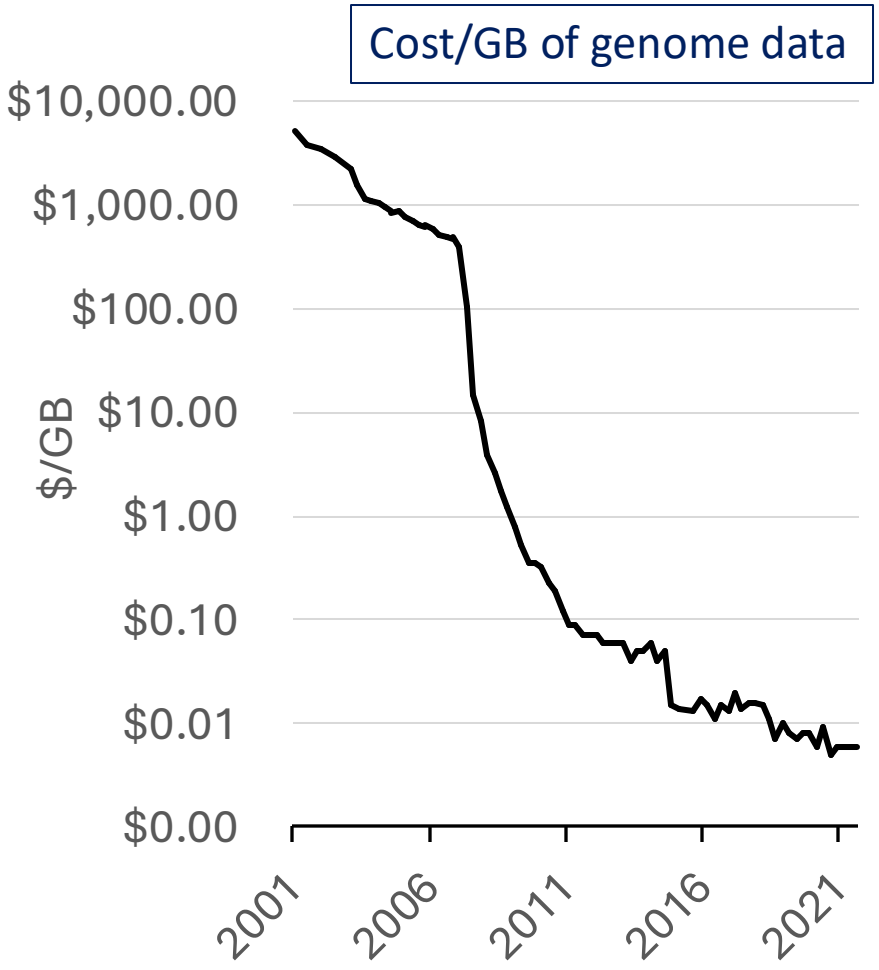
But, We Need More Performance!

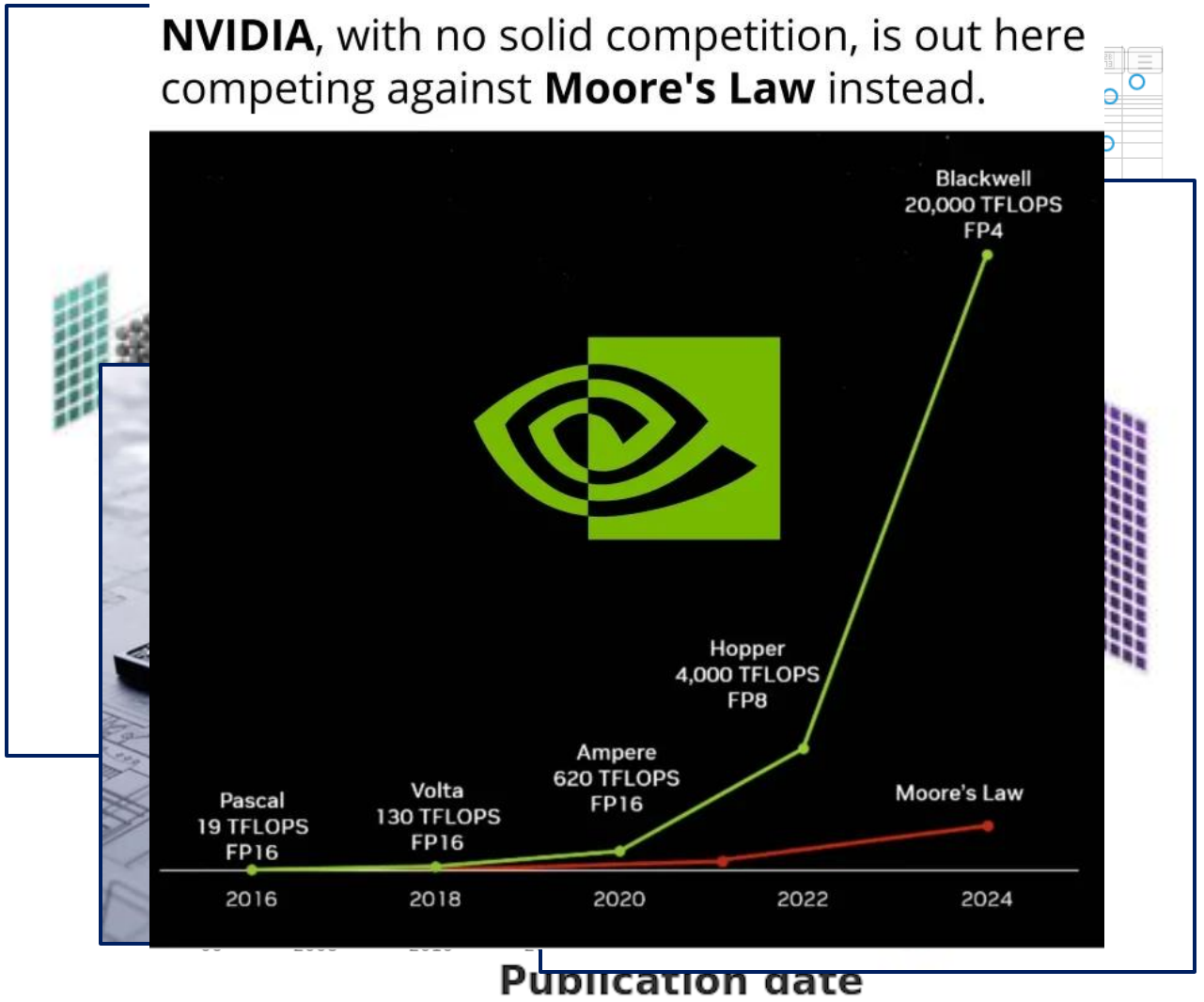




https://epoch.ai/blog/machine-learning-model-sizes-and-the-parameter-gap

But, We Need More Performance!





https://epoch.ai/blog/machine-learning-model-sizes-and-the-parameter-gap

GPU Scalability Trend

- Volta – V100 (2017) – 12nm ~$10,000 at release
  - 32 bit CUDA: ~14 TFLOPS
  - 32 bit tensor: ~112 TFLOPS
  - 21B transistors – ~300 W - 815 mm^2
- Ampere – A100 (2020) – 7nm ~$10,000 at release
  - 32 bit CUDA: ~19.5 TFLOPS
  - 32 bit tensor: ~156 TFLOPS *TF32 != FP32!
  - 52B transistors – ~300 W - 826 mm^2
- Hopper – H100 (2022) – 4nm ~$25,000 at release
  - 32 bit CUDA: ~67 TFLOPS
  - 32 bit tensor: ~400 TFLOPS (higher with sparsity support) *TF32 != FP32!
  - 80B transistors – ~300 W - 814 mm^2
- Blackwell – B100 (2024) – 4nm ~$35,000 at release
  - 32 bit CUDA: ~60 TFLOPS
  - 32 bit tensor: ~900 TFLOPS (higher with sparsity support) *TF32 != FP32!
  - 208B transistors - ?? mm^2

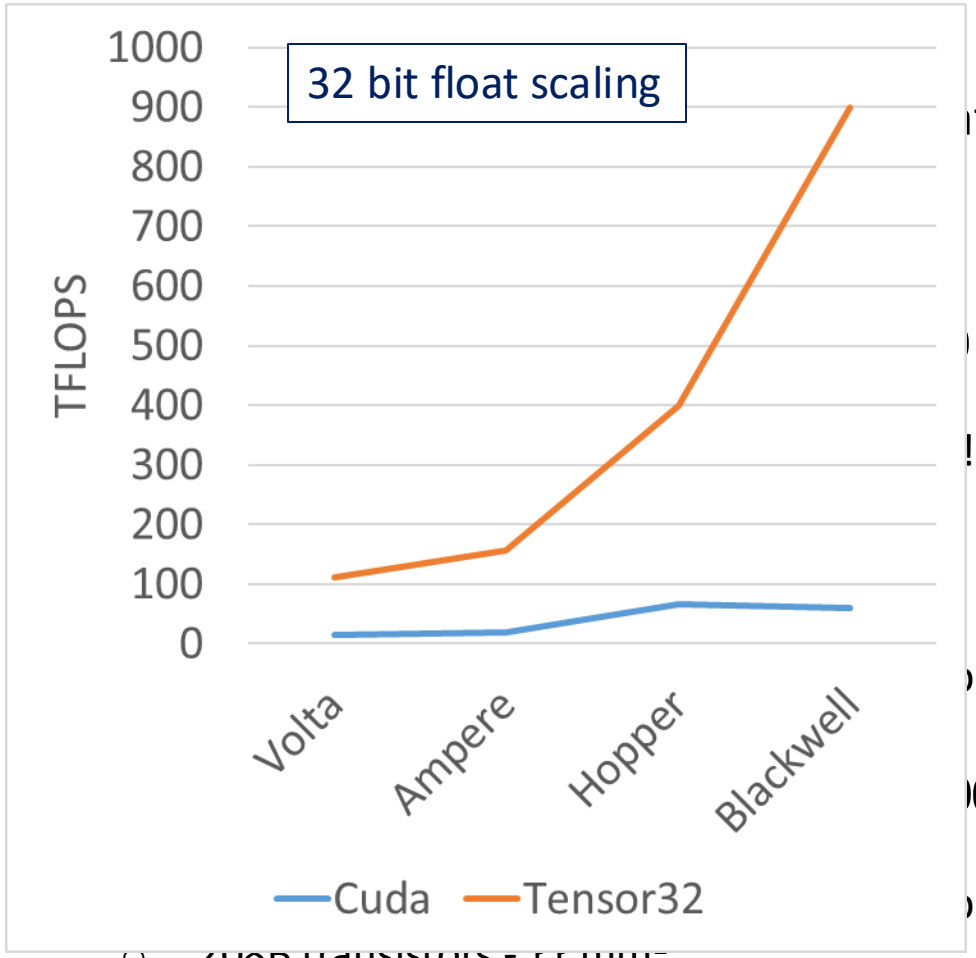
GPU Scalability Trend

- Volta – V100 (2017) – 12nm ~$10,000 at release
  - 32 bit CUDA: ~14 TFLOPS
  - 32 bit tensor: ~112 TFLOPS
  - 21B transistors – ~300 W - 815 mm^2
- Ampere – A100 (2020) – 7nm ~$10,000 at release
  - 32 bit CUDA: ~19.5 TFLOPS
  - 32 bit tensor: ~156 TFLOPS *TF32 != FP32!
  - 52B transistors – ~300 W - 826 mm^2
- Hopper – H100 (2022) – 4nm ~$25,000 at release
  - 32 bit CUDA: ~67 TFLOPS
  - 32 bit tensor: ~400 TFLOPS (higher with sparsity support) *TF32 != FP32!
  - 80B transistors – ~300 W - 814 mm^2
- Blackwell – B100 (2024) – 4nm ~$35,000 at release
  - 32 bit CUDA: ~60 TFLOPS
  - 32 bit tensor: ~900 TFLOPS (higher with sparsity support) *TF32 != FP32!
  - 208B transistors - ?? mm^2

GPU Scalability Trend

- Volta – V100 (2017) – 12nm ~$10,000 at release
  - 32 bit CUDA: ~14 TFLOPS
  - 32 bit tensor: ~112 TFLOPS
  - 21B transistors – ~300 W - 815 mm^2
- Ampere – A100 (2020) – 7nm ~$10,000 at release
  - 32 bit CUDA: ~19.5 TFLOPS
  - 32 bit tensor: ~156 TFLOPS *TF32 != FP32!
  - 52B transistors – ~300 W - 826 mm^2
- Hopper – H100 (2022) – 4nm ~$25,000 at release
  - 32 bit CUDA: ~67 TFLOPS
  - 32 bit tensor: ~400 TFLOPS (higher with sparsity support) *TF32 != FP32!
  - 80B transistors – ~300 W - 814 mm^2
- Blackwell – B100 (2024) – 4nm ~$35,000 at release
  - 32 bit CUDA: ~60 TFLOPS
  - 32 bit tensor: ~900 TFLOPS (higher with sparsity support) *TF32 != FP32!
  - 208B transistors - ?? mm^2

GPU Scalability Trend



t release

at release

at release

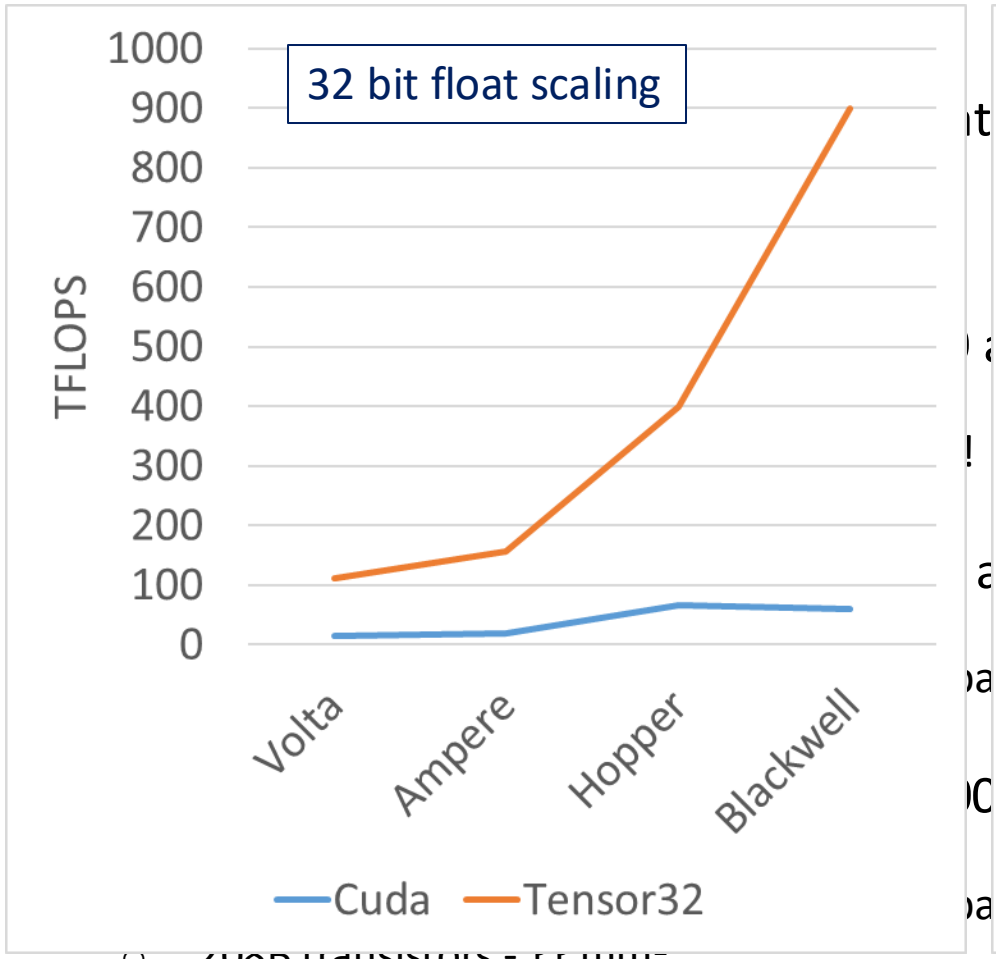
arsity support) *TF32 != FP32!

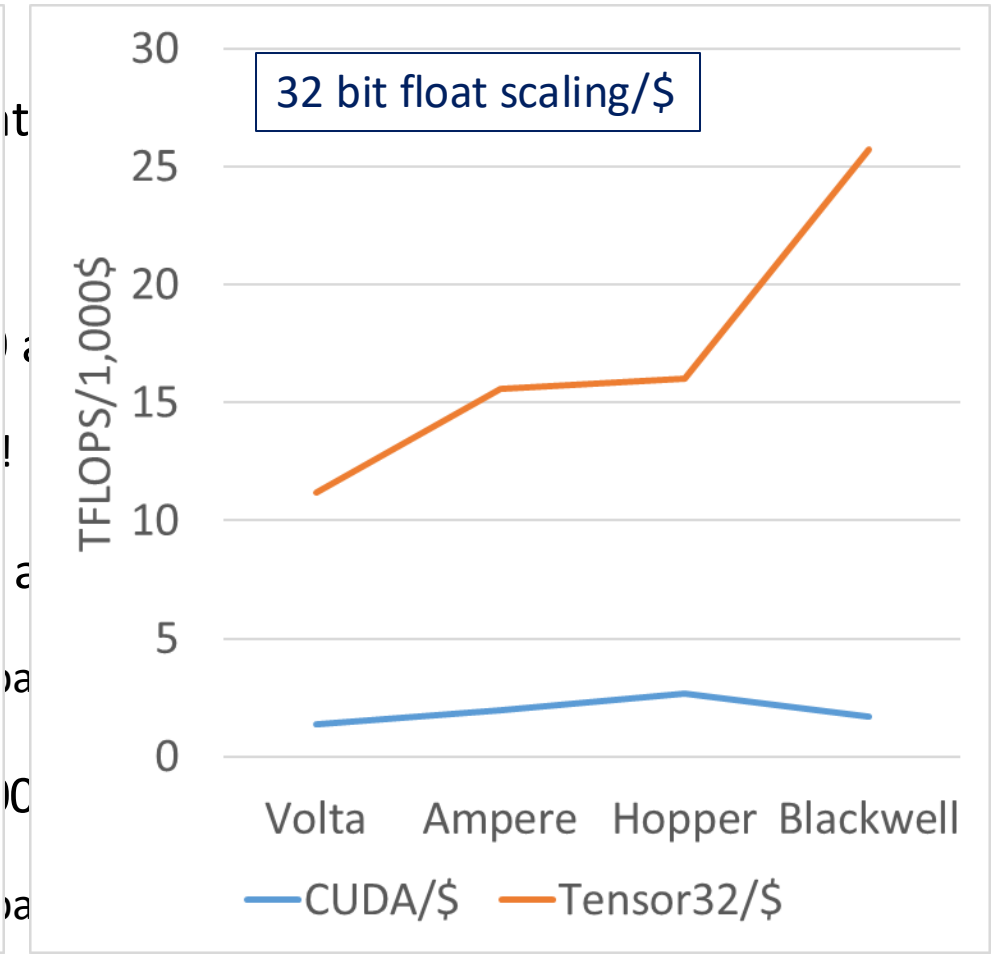
0 at release

arsity support) *TF32 != FP32!

- 208B transistors - ?? mm²

GPU Scalability Trend

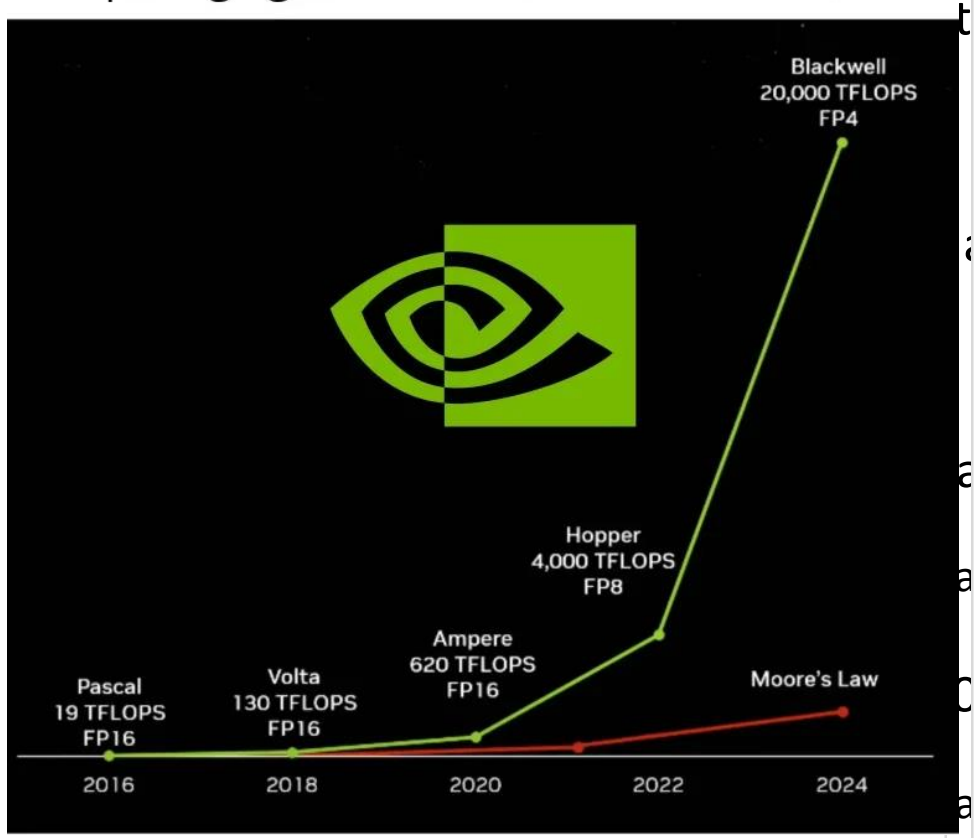


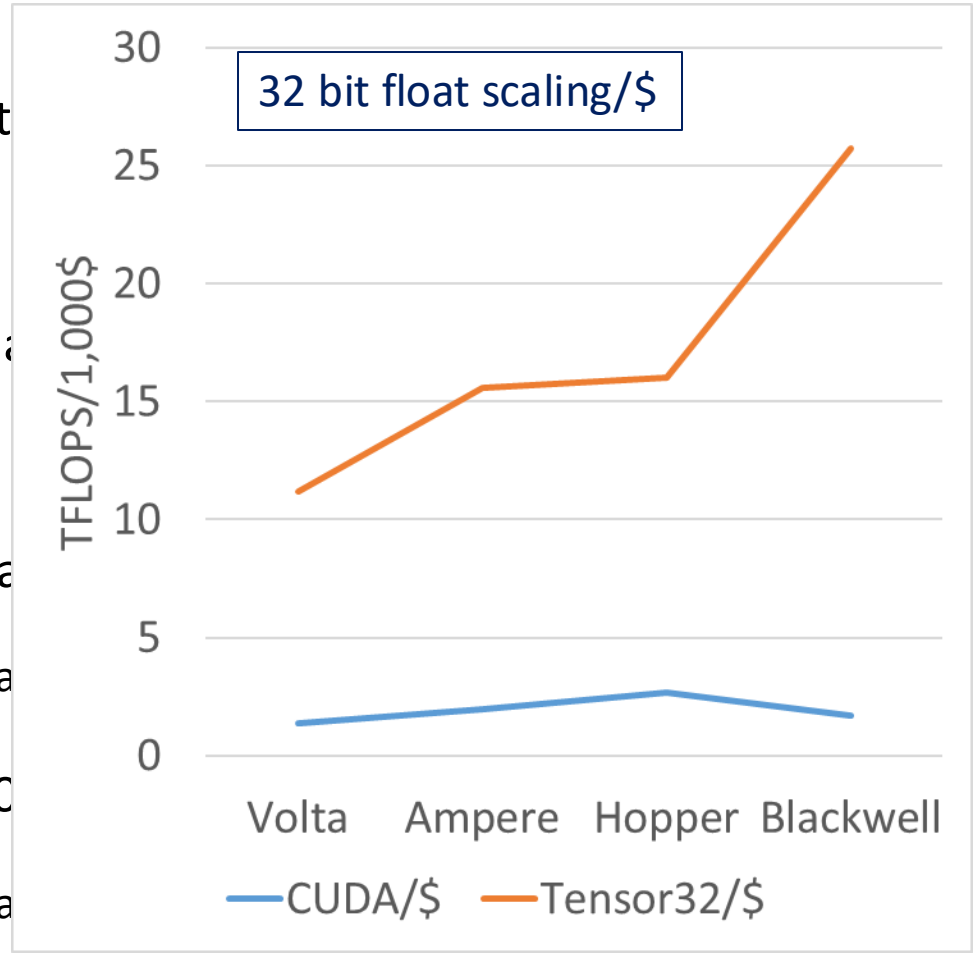


- 208B transistors - ?? mm²

GPU Scalability Trend

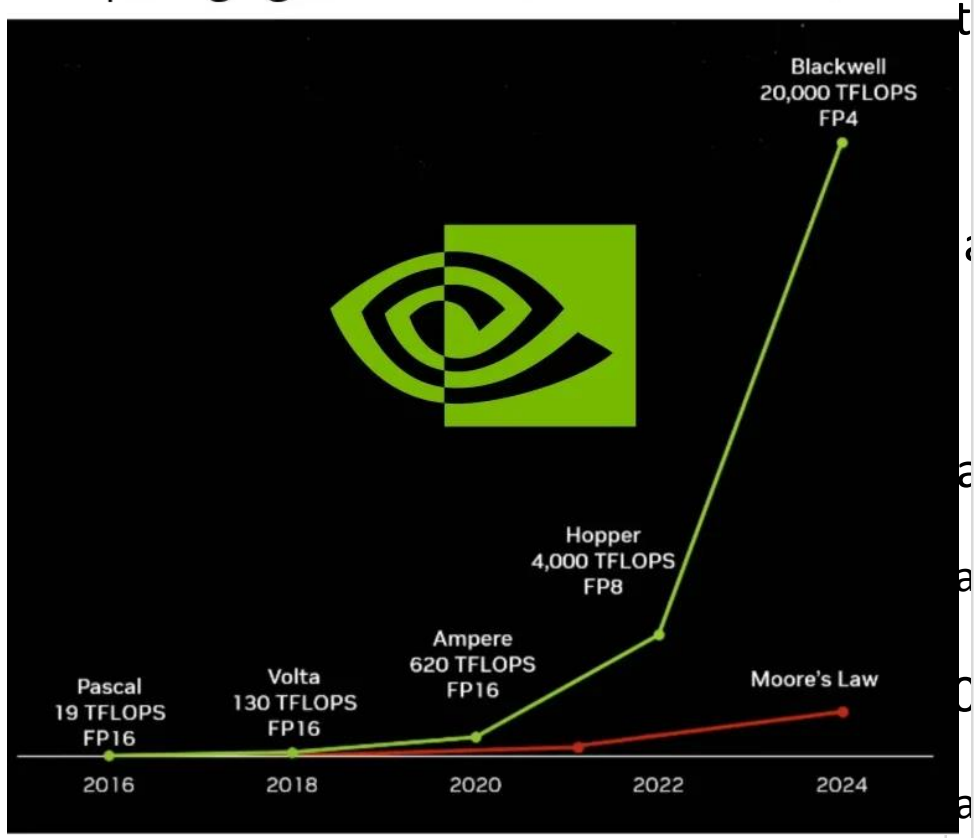
NVIDIA, with no solid competition, is out here competing against **Moore's Law** instead.

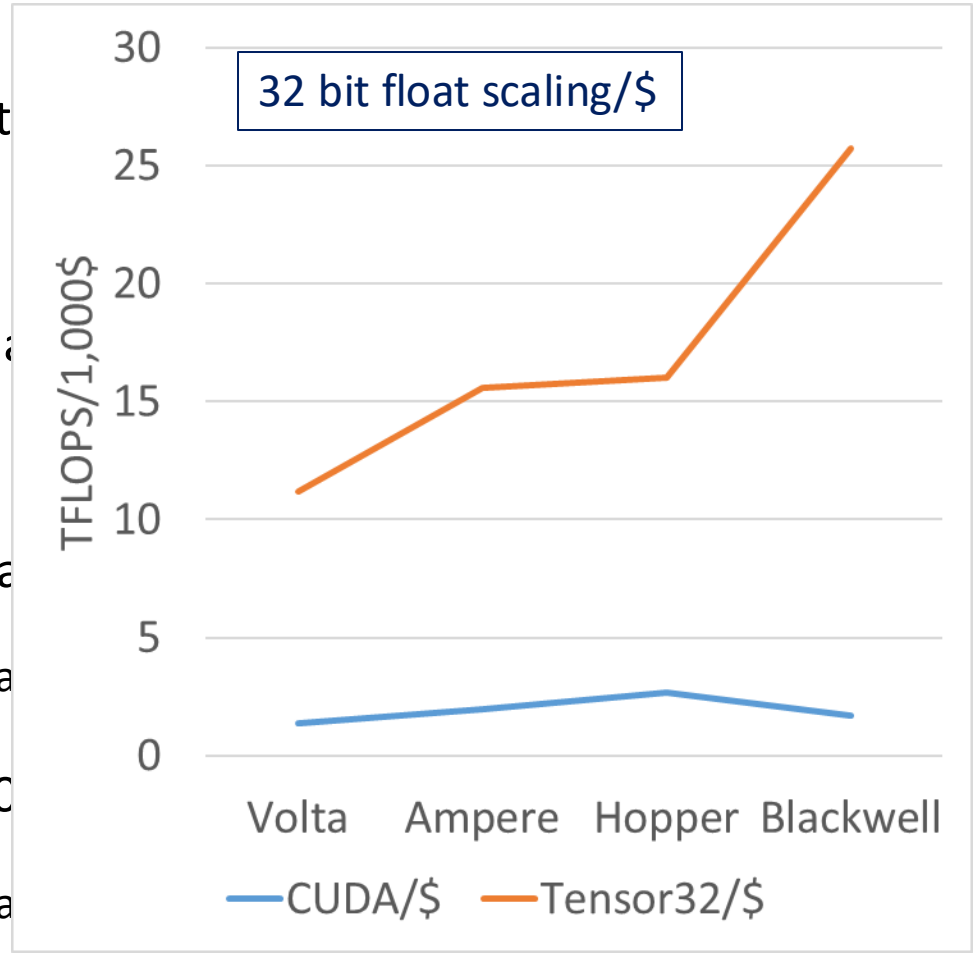




GPU Scalability Trend

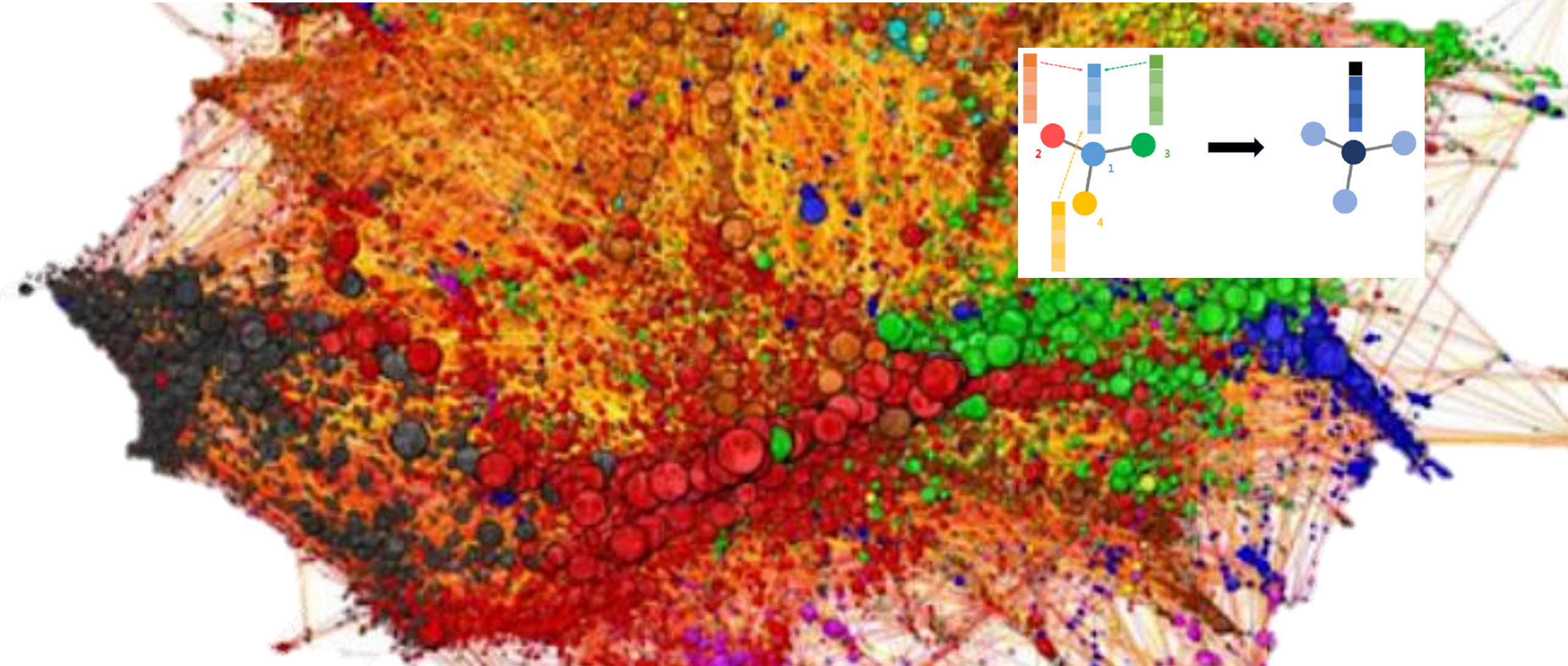
NVIDIA, with no solid competition, is out here competing against **Moore's Law** instead.





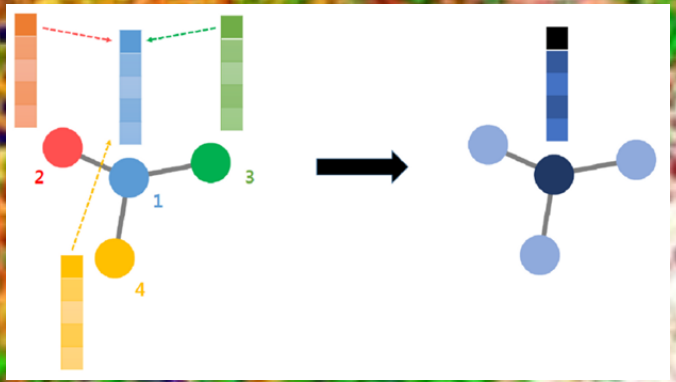
What about the rest of us?

An Example: Graph Neural Networks!



An Example: Graph Neural Networks!

Irregular computation patterns

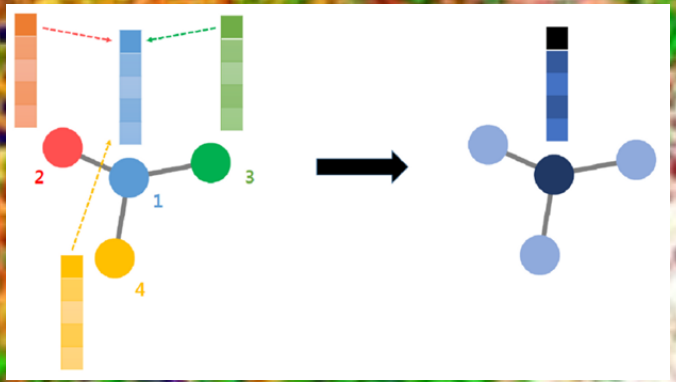


Low warp utilization

An Example: Graph Neural Networks!

Irregular computation patterns

Irregular memory accesses



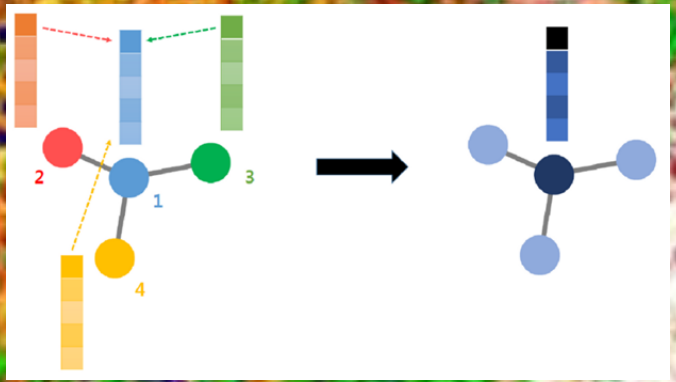
Low warp utilization

An Example: Graph Neural Networks!

Irregular computation patterns

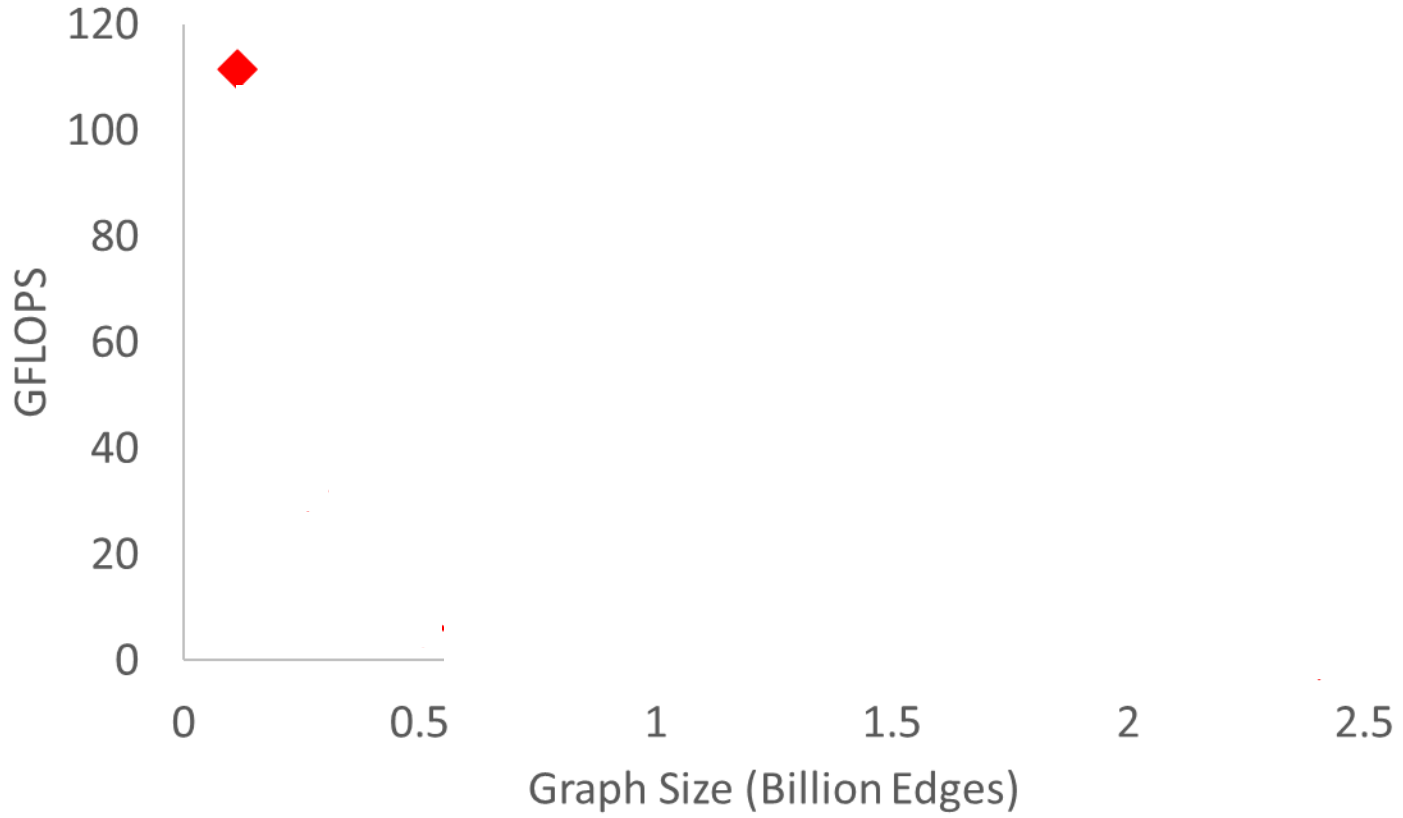
Irregular memory accesses

Graphs larger than GPU memory

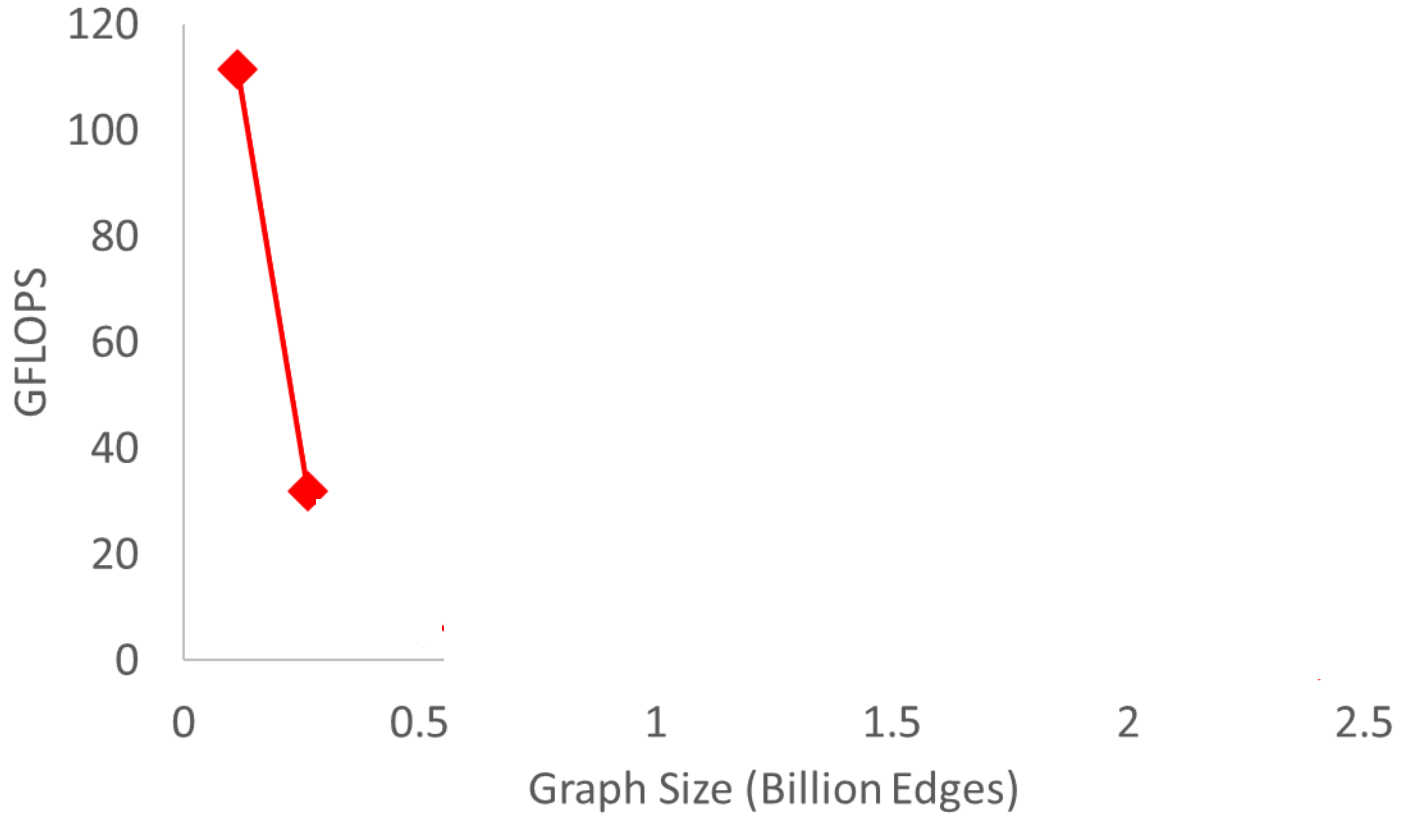


Low warp utilization

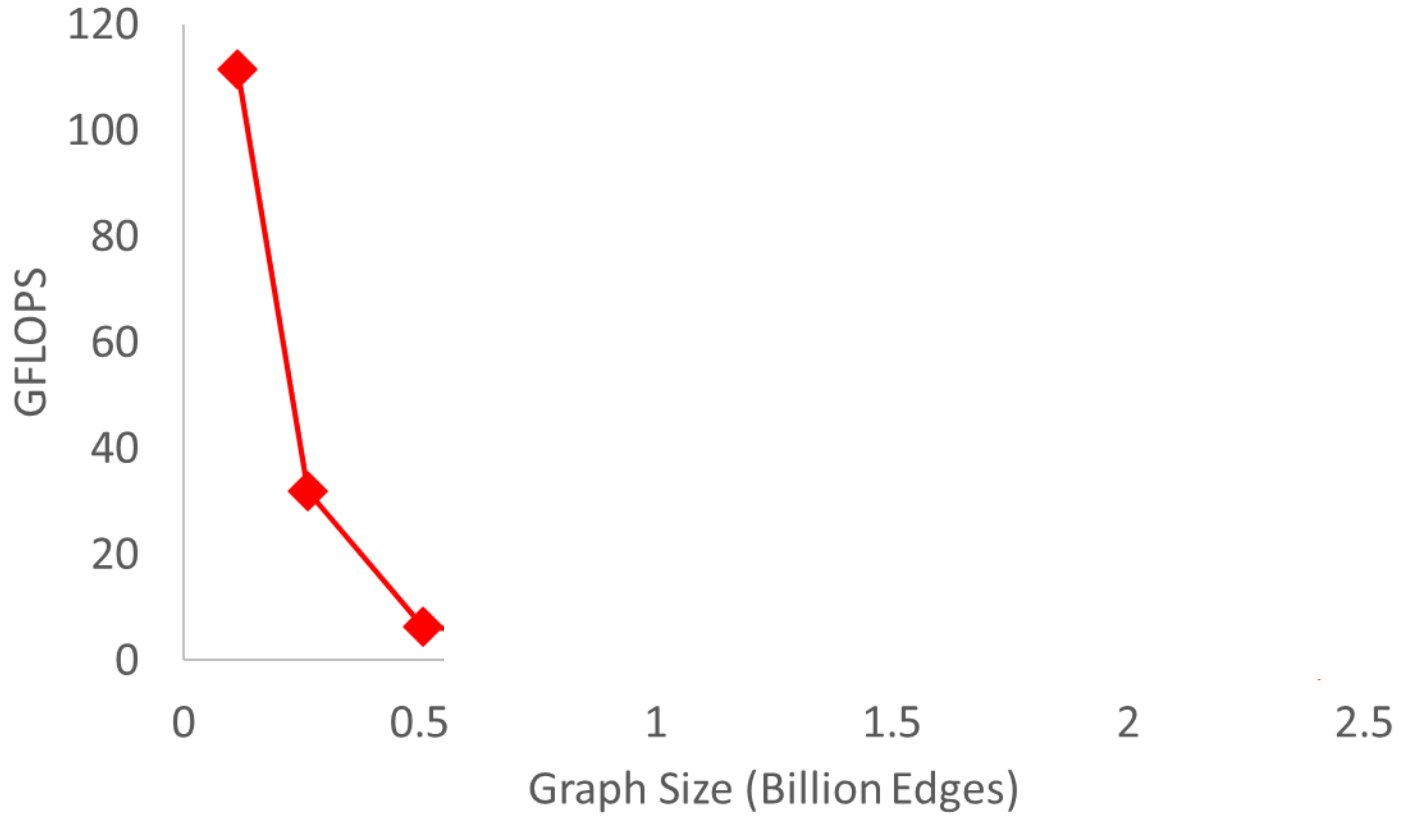
GPU for GCN in practice



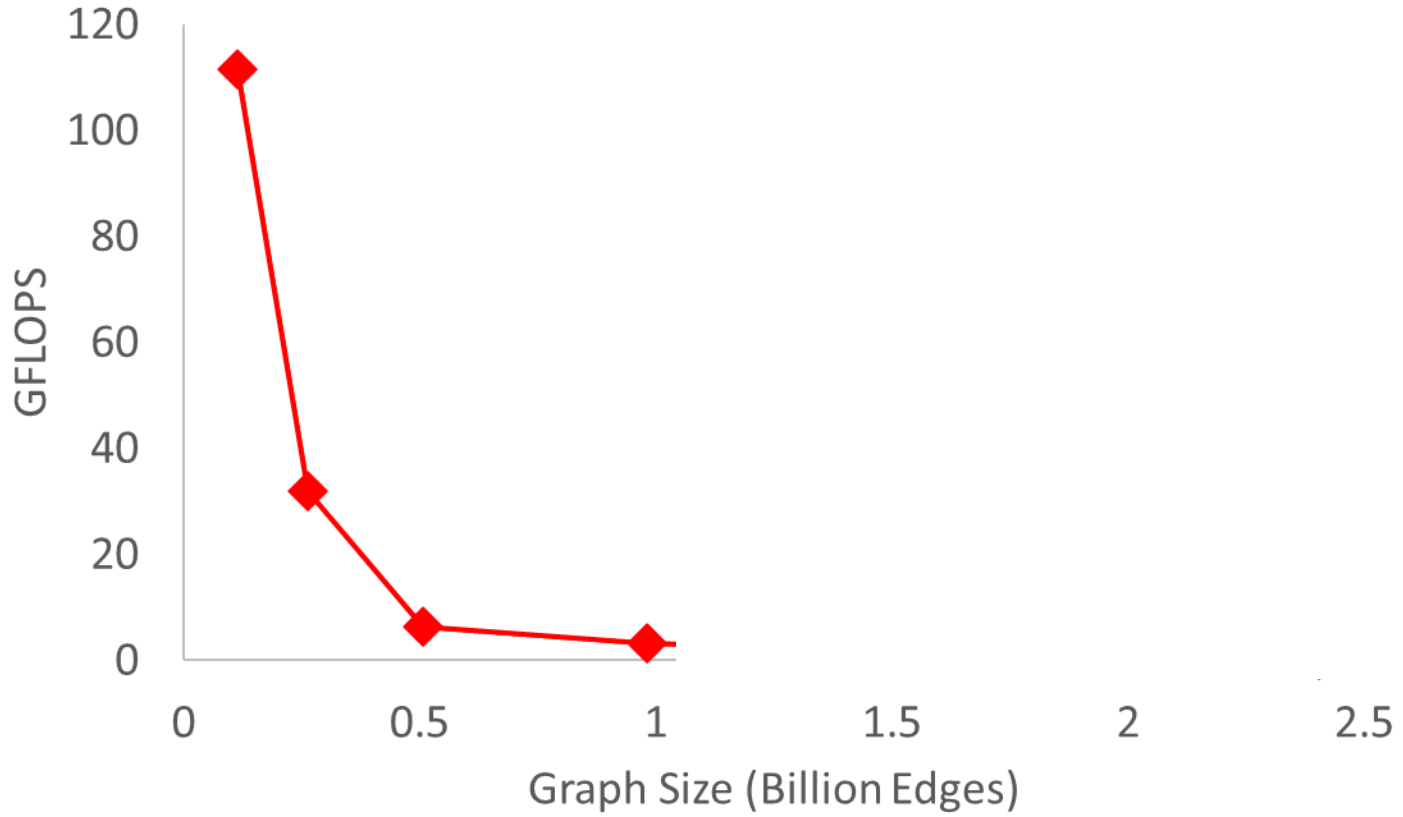
GPU for GCN in practice



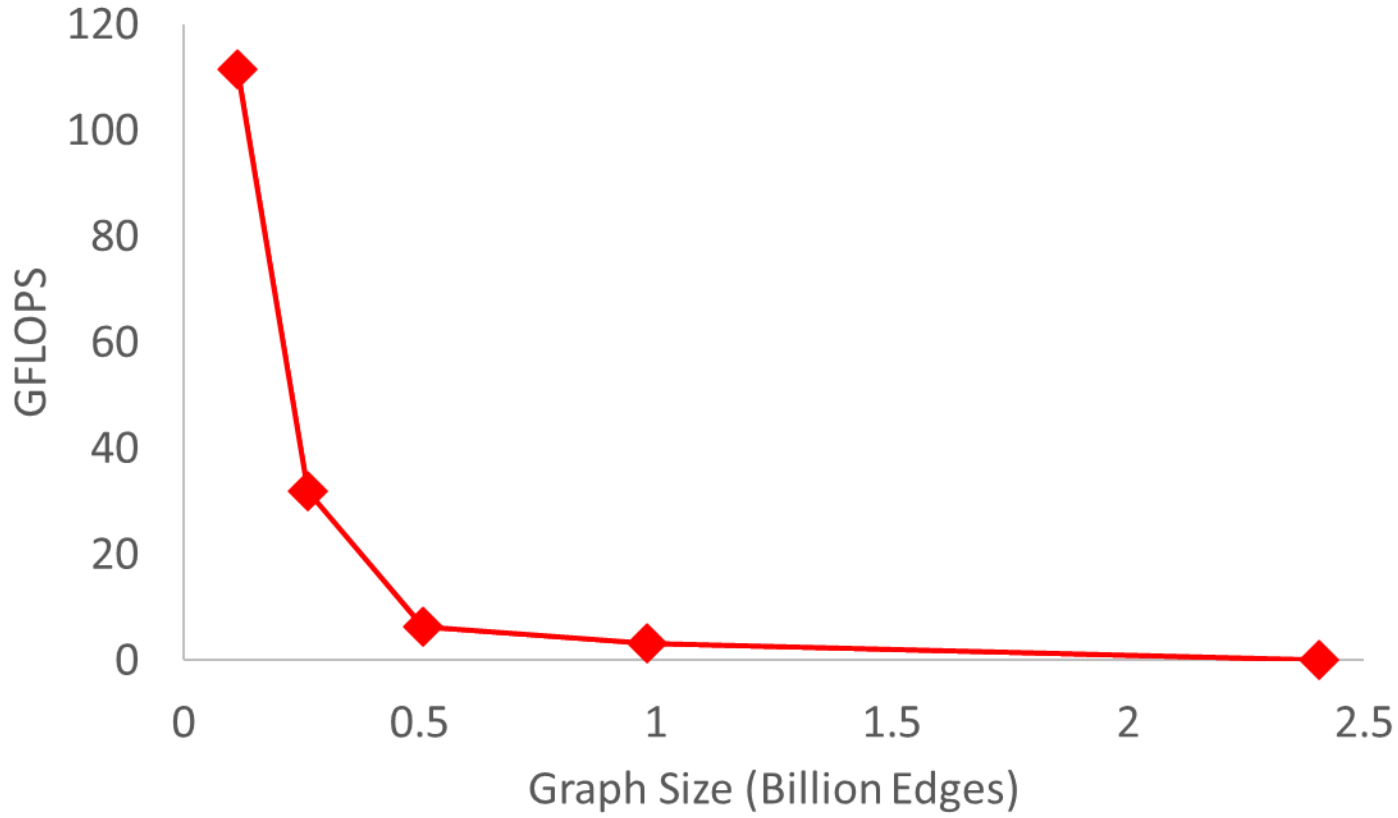
GPU for GCN in practice



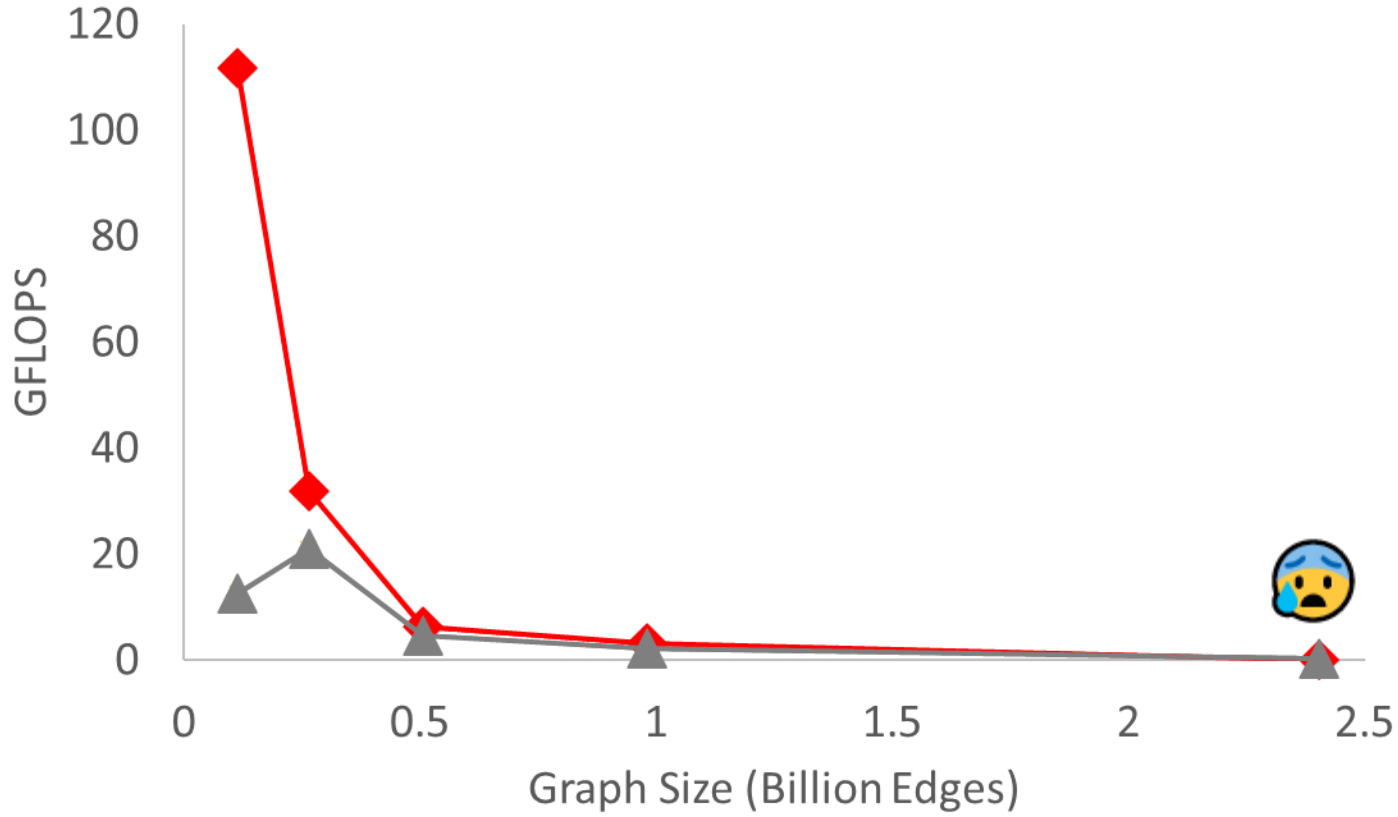
GPU for GCN in practice



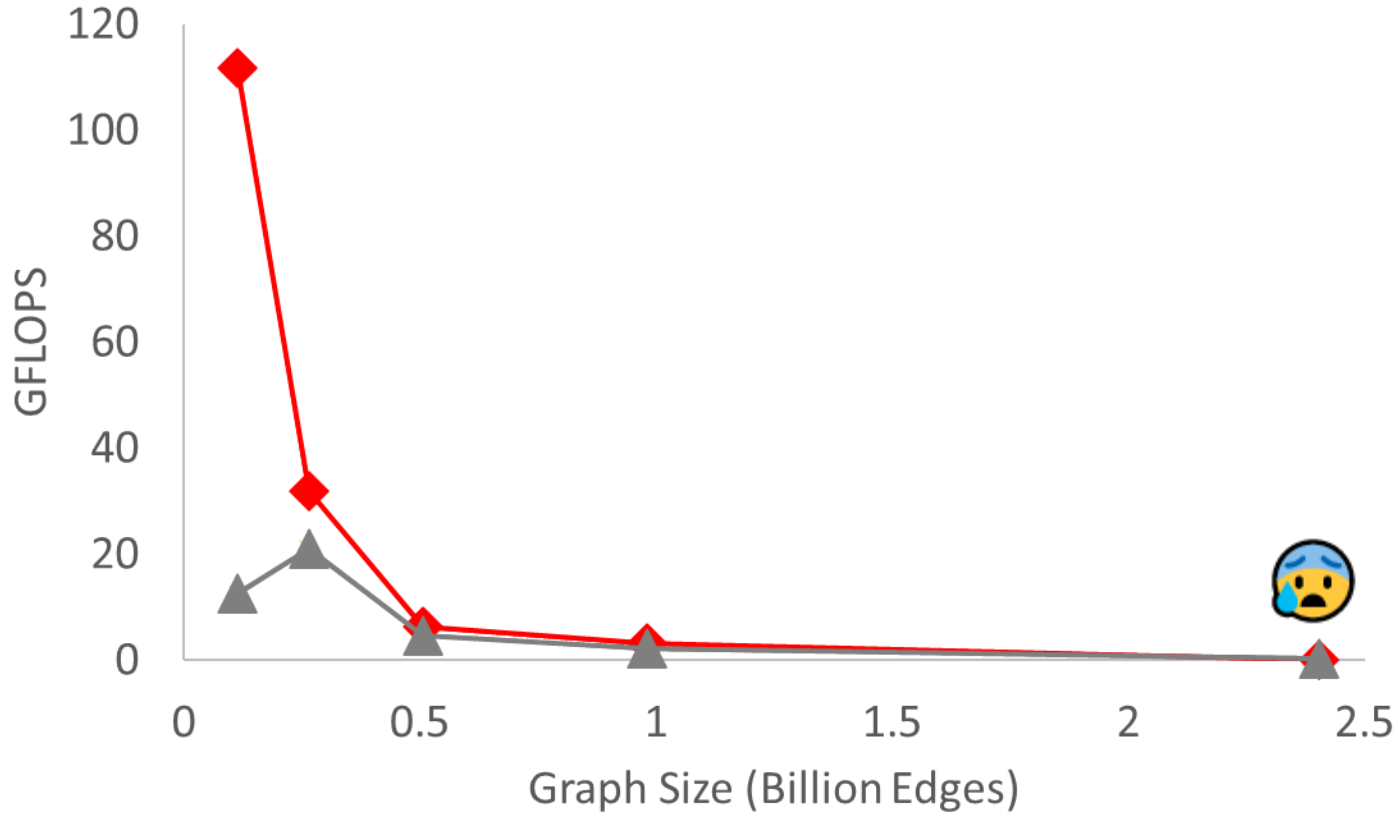
GPU for GCN in practice



GPU for GCN in practice

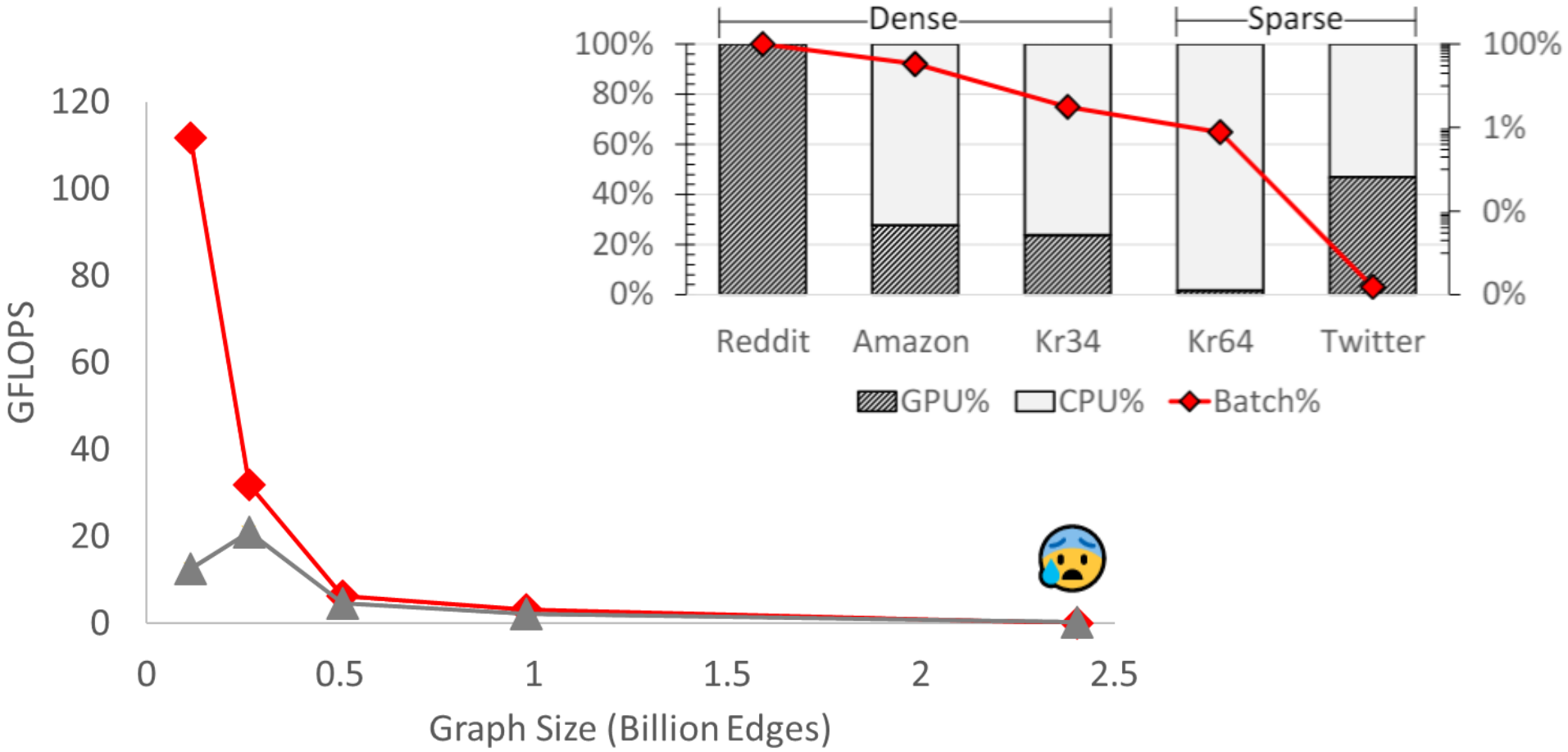


GPU for GCN in practice



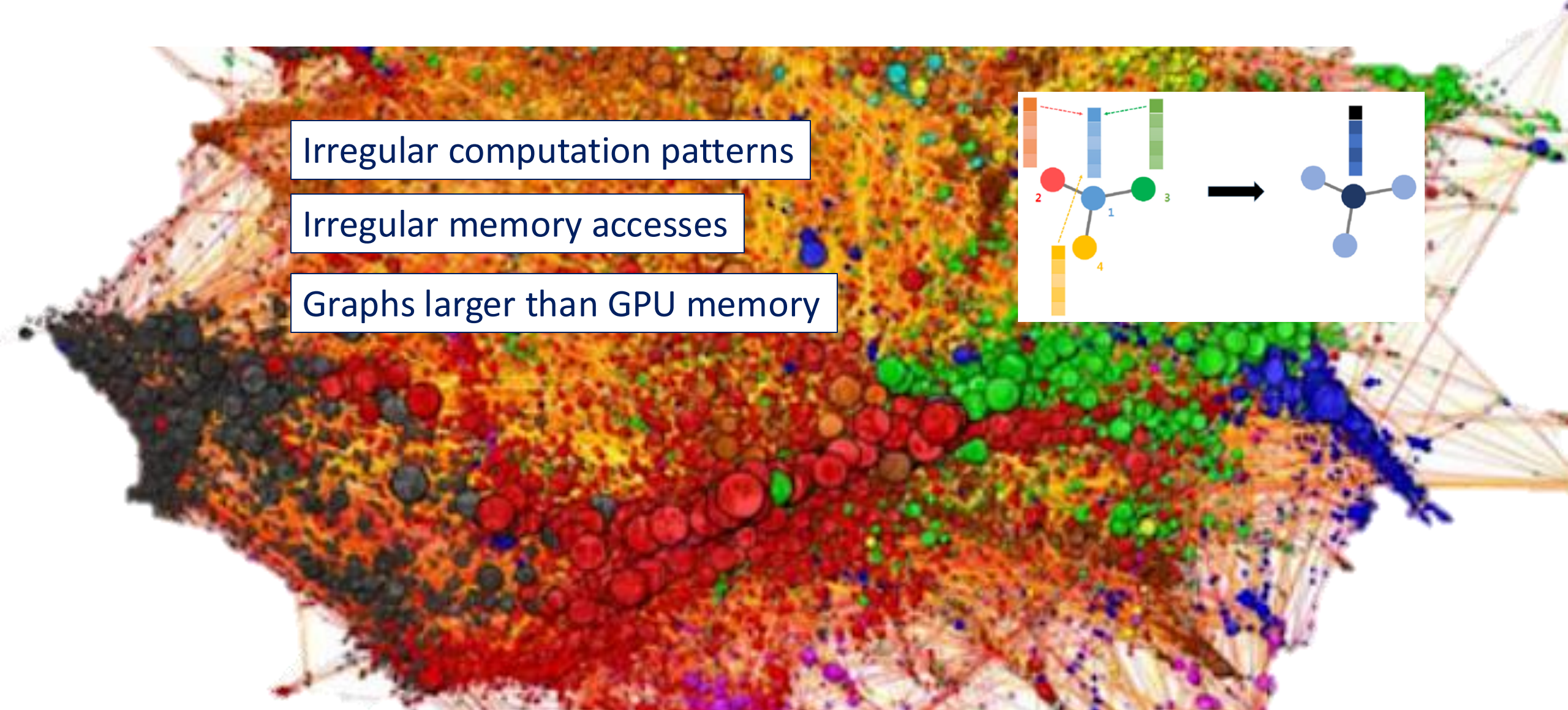
Isn’t GPU throughput supposed to be a multi-TFLOP?

GPU for GCN in practice



Isn’t GPU throughput supposed to be a multi-TFLOP?

An Example: Graph Neural Networks!



Irregular computation patterns

Irregular memory accesses

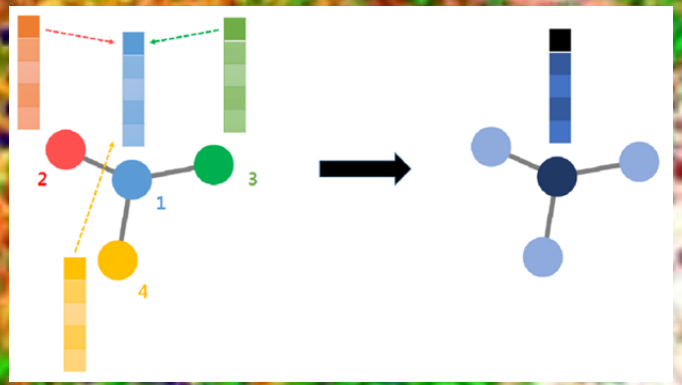
Graphs larger than GPU memory

An Example: Graph Neural Networks!

Irregular computation patterns

Irregular memory accesses

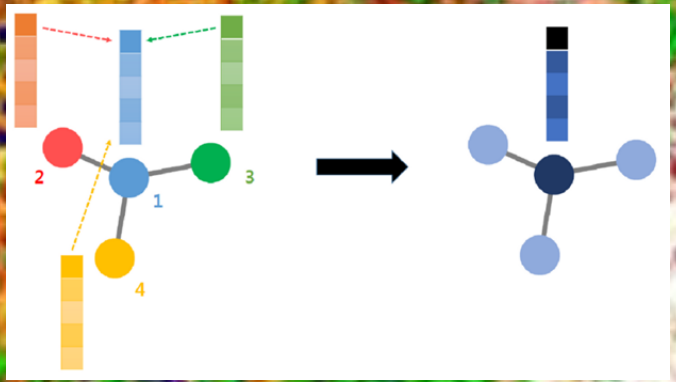
Graphs larger than GPU memory



Can FPGAs save us?

An Example: Graph Neural Networks!

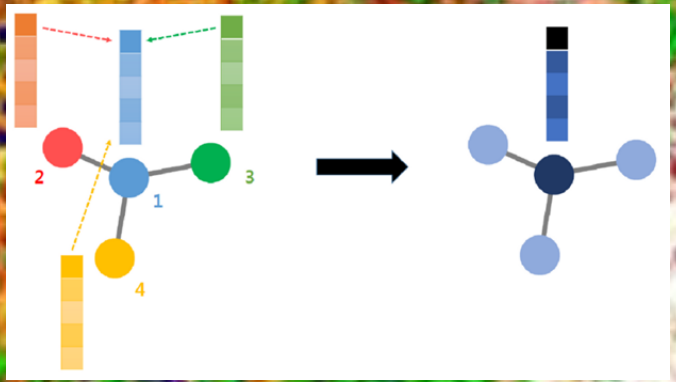
- Irregular computation patterns
- Irregular memory accesses
- Graphs larger than GPU memory



Can FPGAs save us?

An Example: Graph Neural Networks!

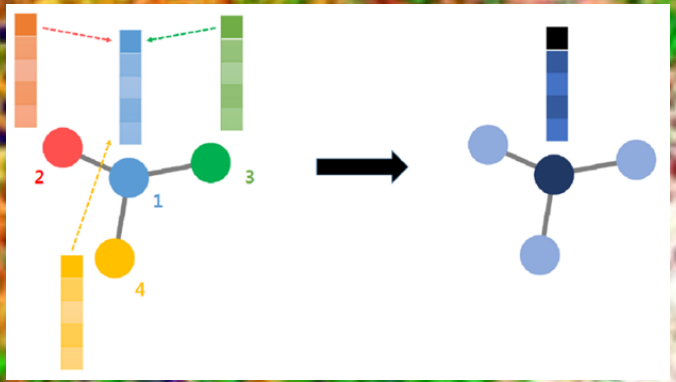
- Irregular computation patterns
- Irregular memory accesses
- Graphs larger than GPU memory



Can FPGAs save us?

An Example: Graph Neural Networks!

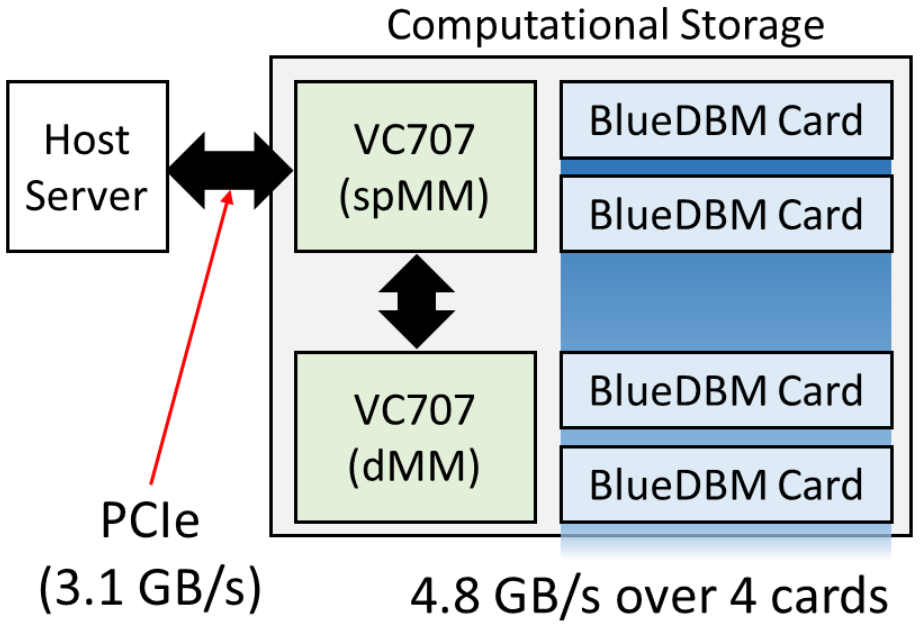
- Irregular computation patterns
- Irregular memory accesses
- Graphs larger than GPU memory

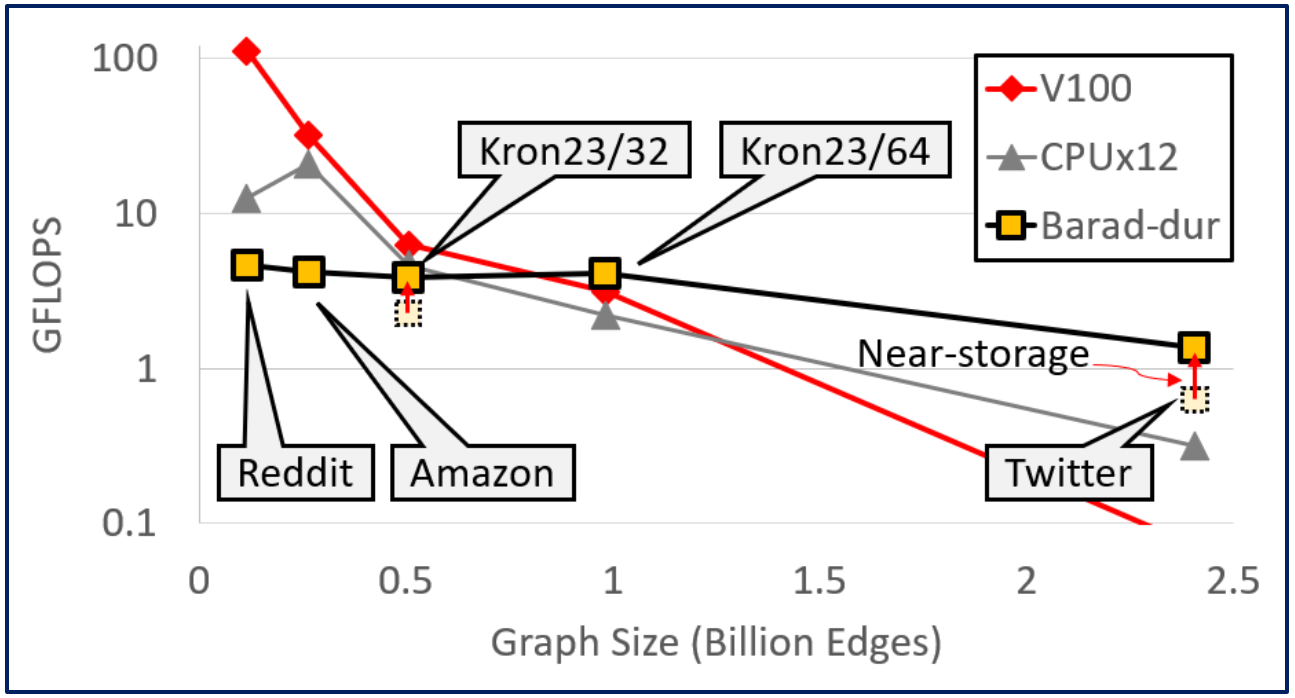


Can FPGAs save us?

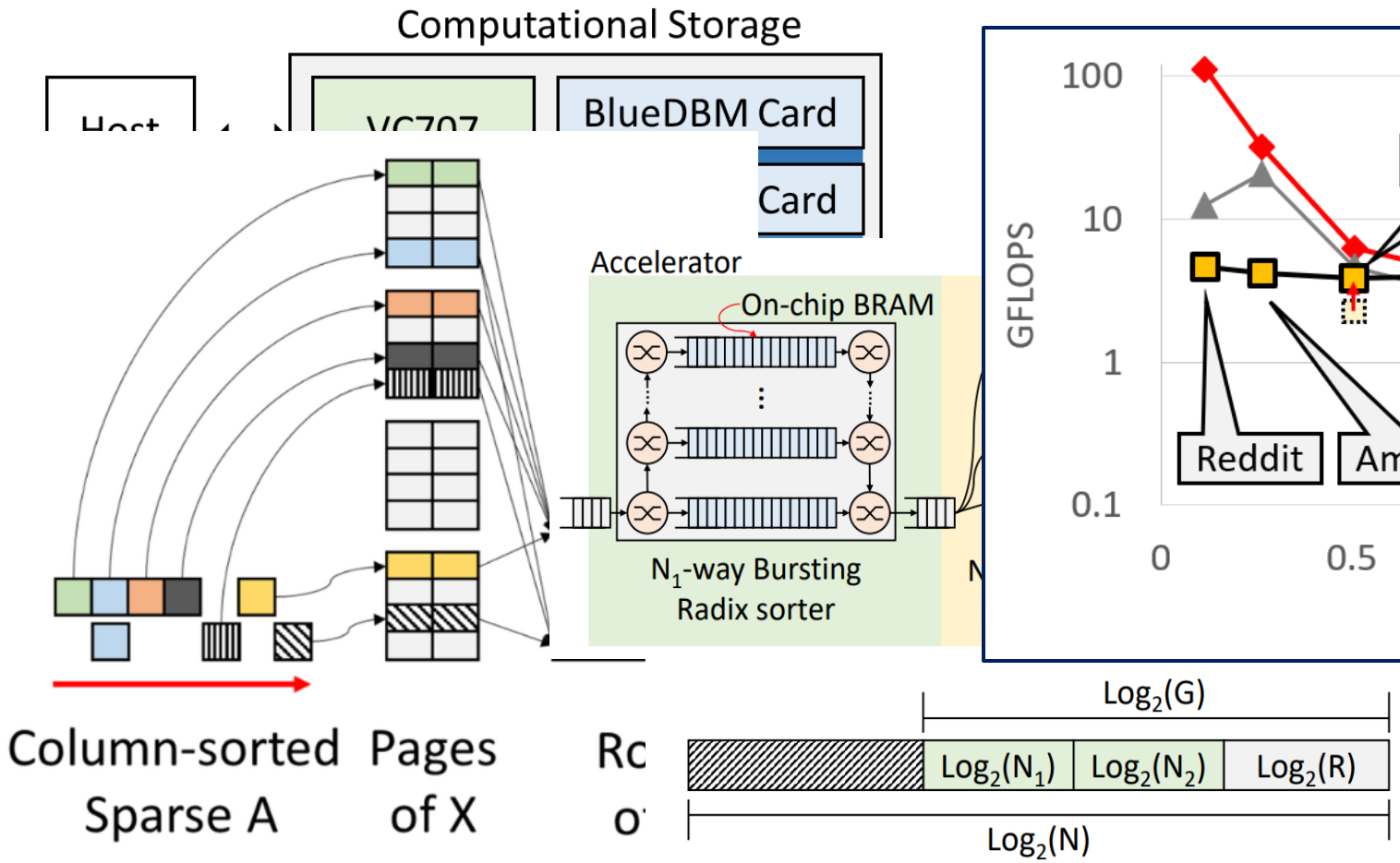
Not by itself!

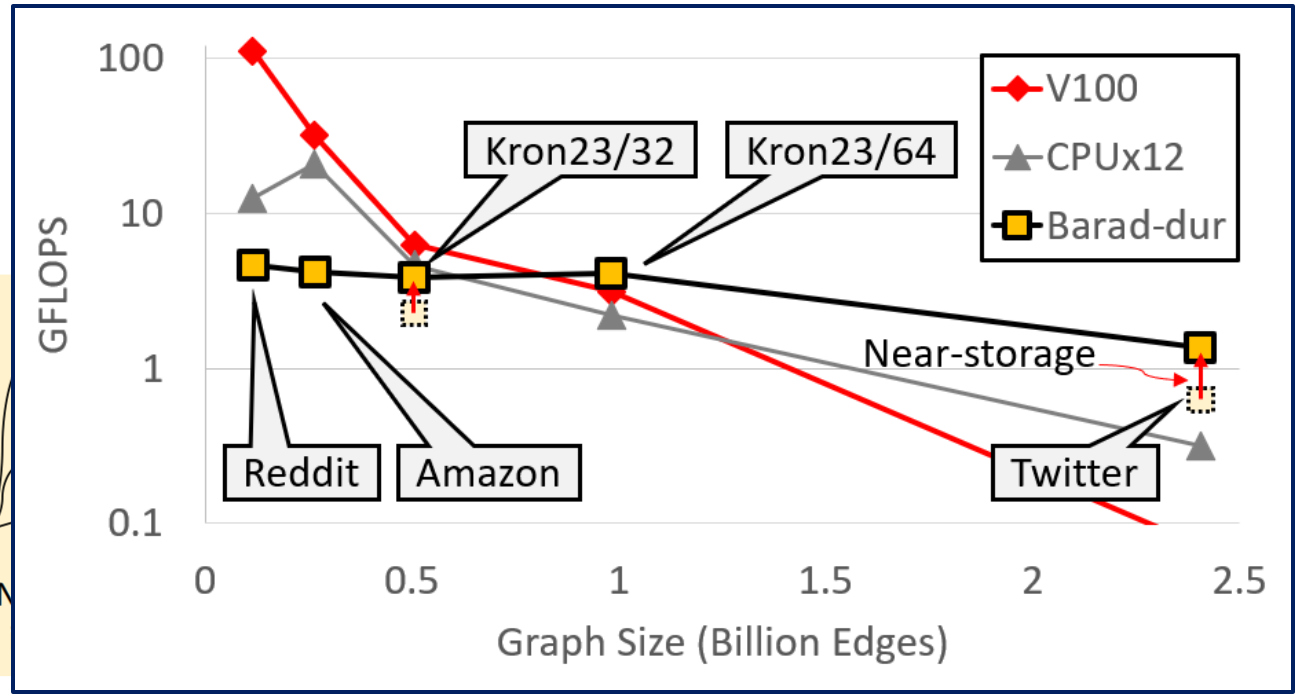
Quick Plug -- A Whole-System Solution: Barad-dur [An et. al., PACT 2023]



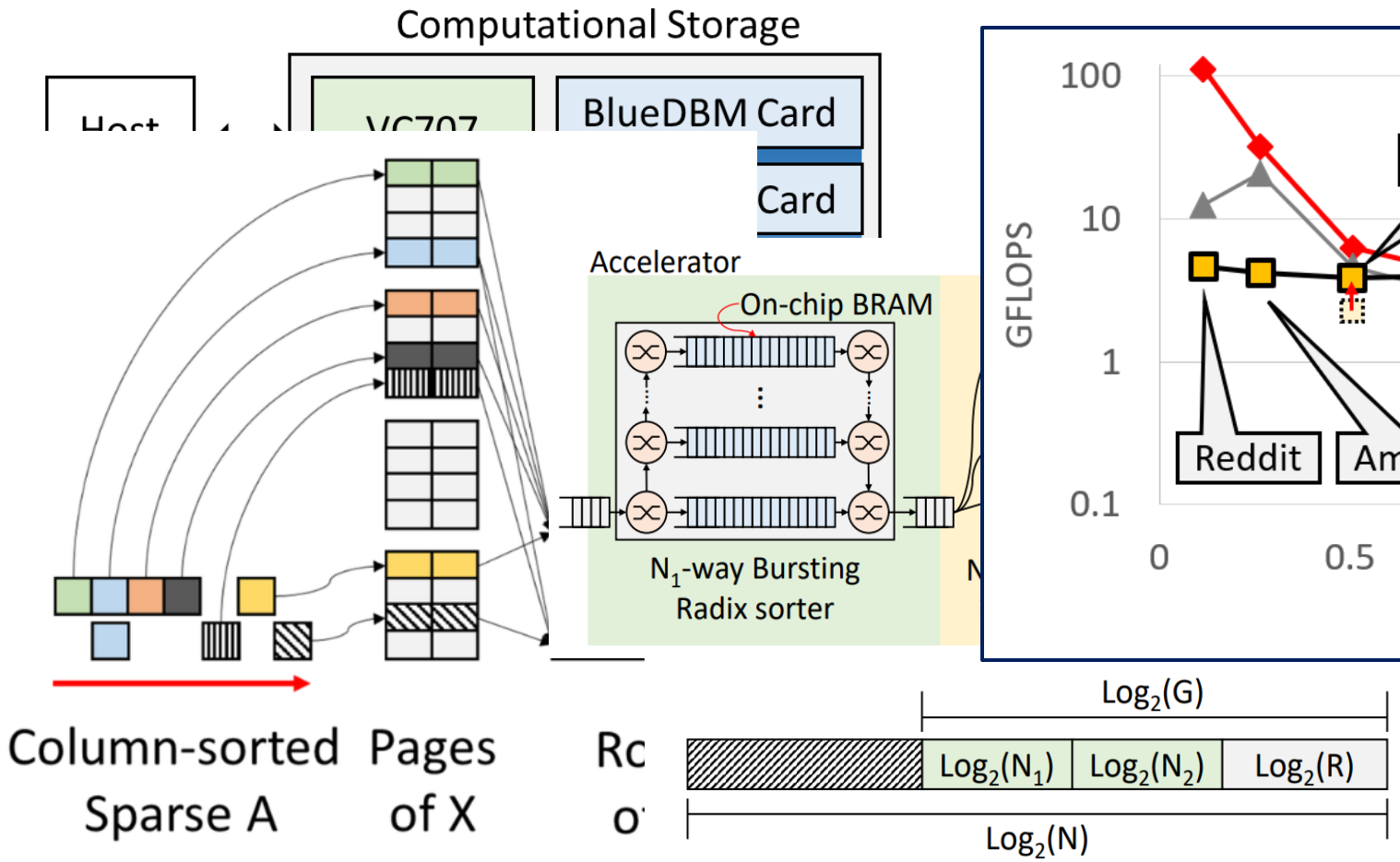


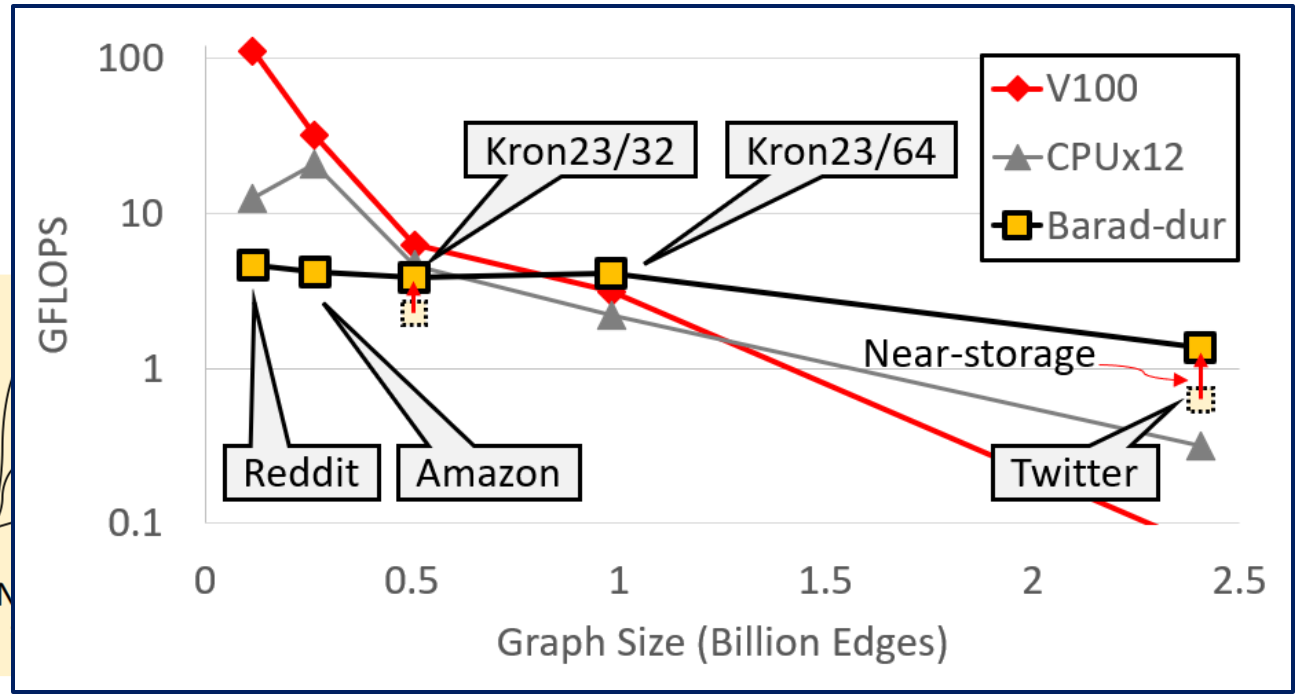
Quick Plug -- A Whole-System Solution: Barad-dur [An et. al., PACT 2023]





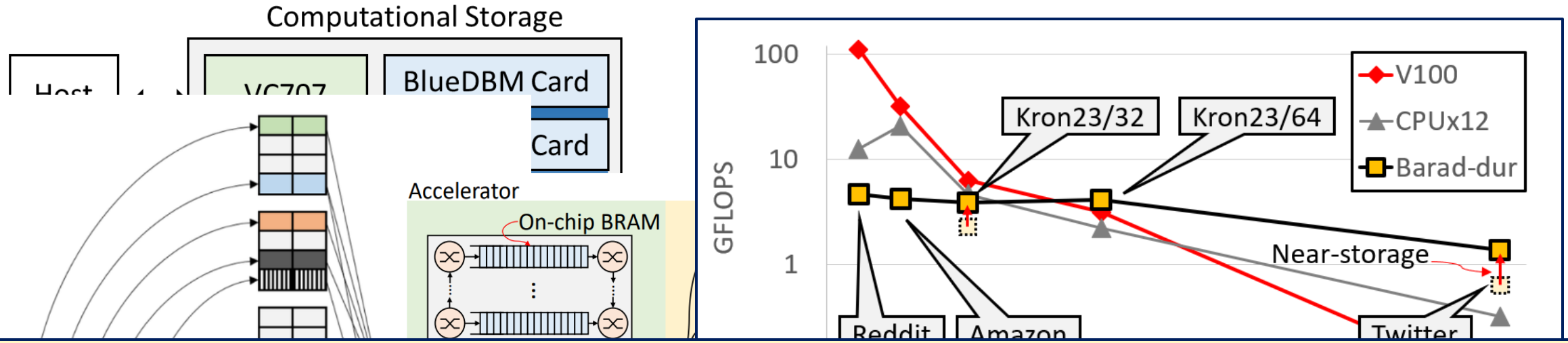
Quick Plug -- A Whole-System Solution: Barad-dur [An et. al., PACT 2023]



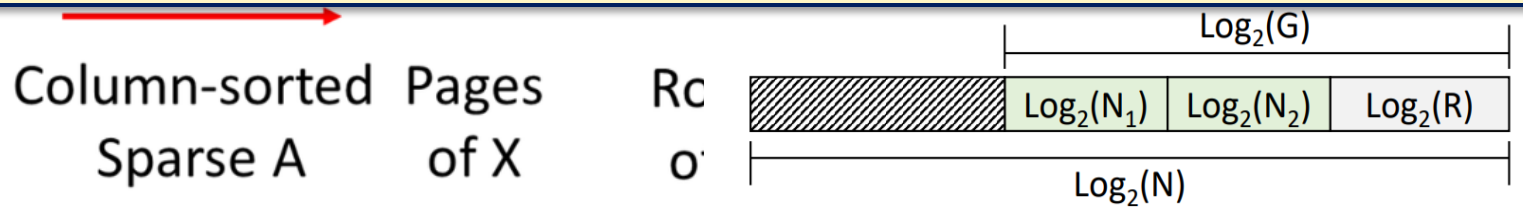


Not today's topic…

Quick Plug -- A Whole-System Solution: Barad-dur [An et. al., PACT 2023]

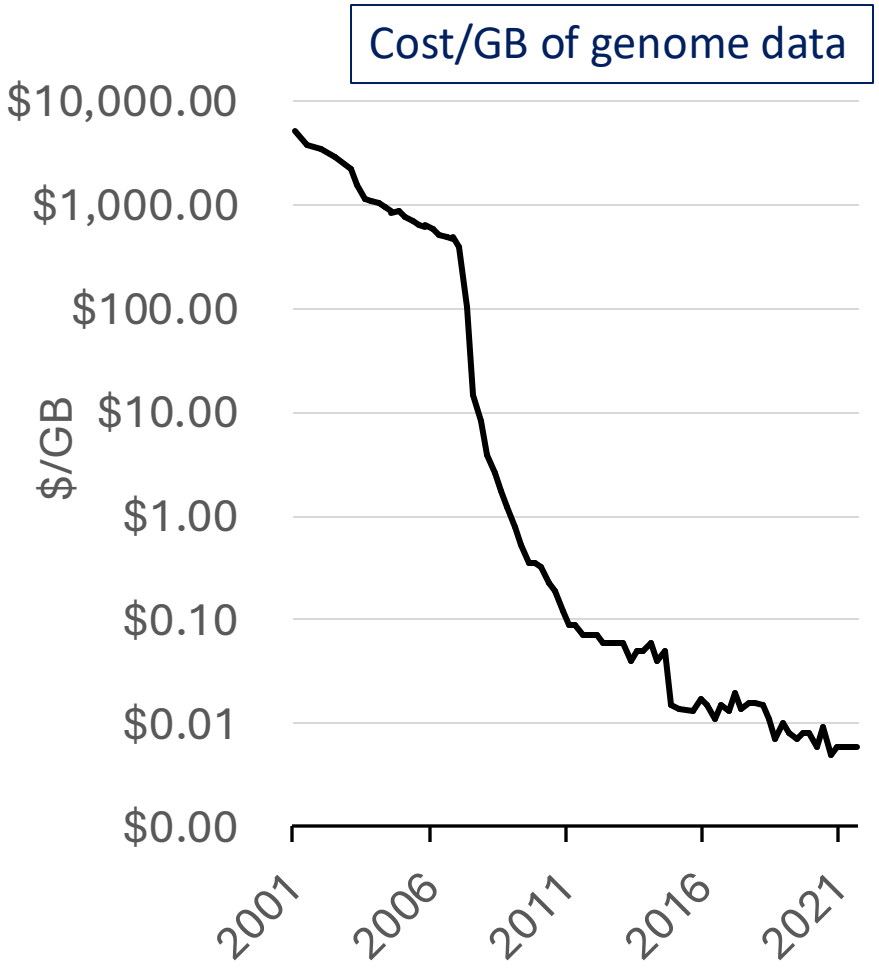


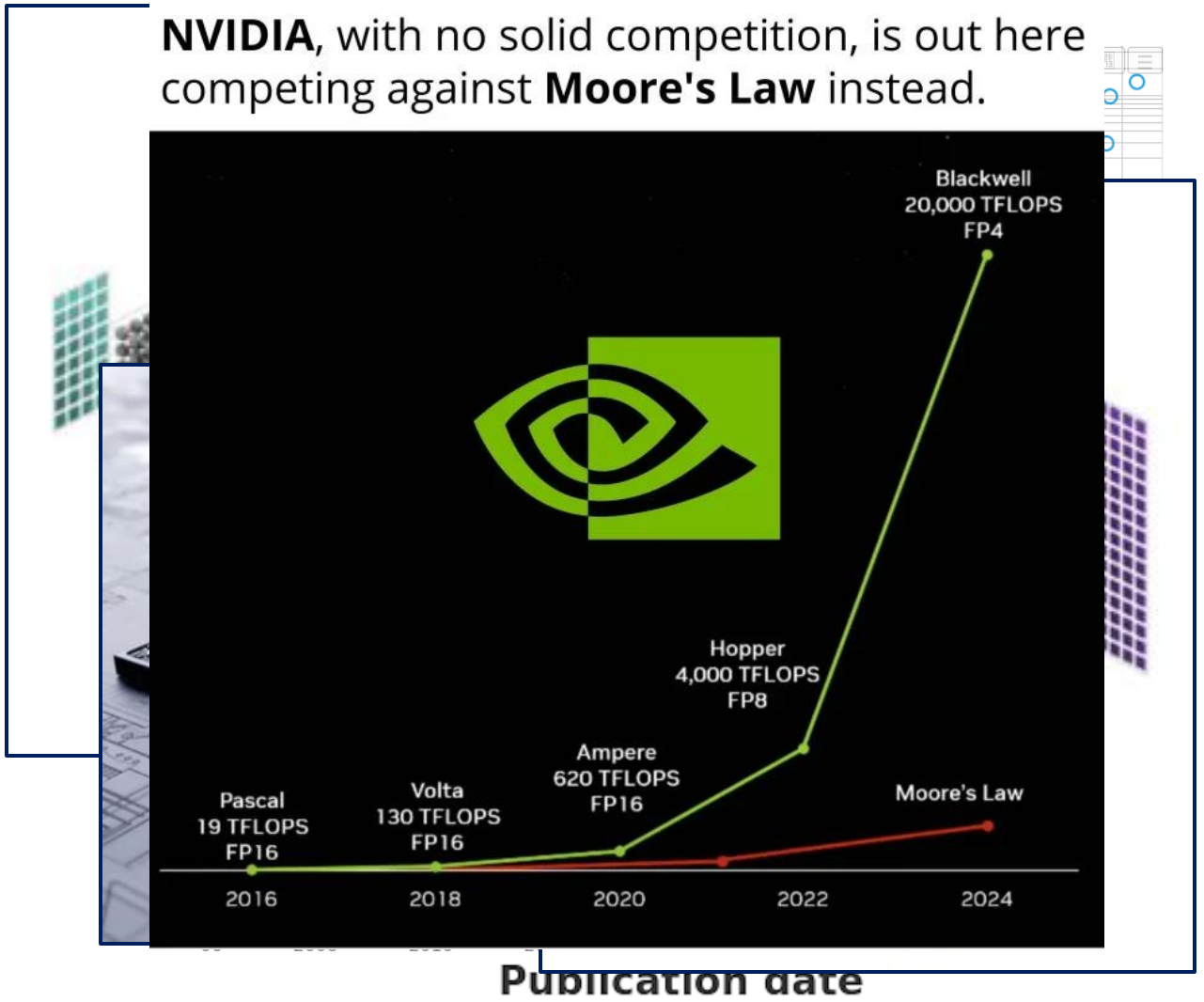
Repeated discovery:
Algorithm and system architecture must co-optimize with hardware acceleration!



Not today's topic…

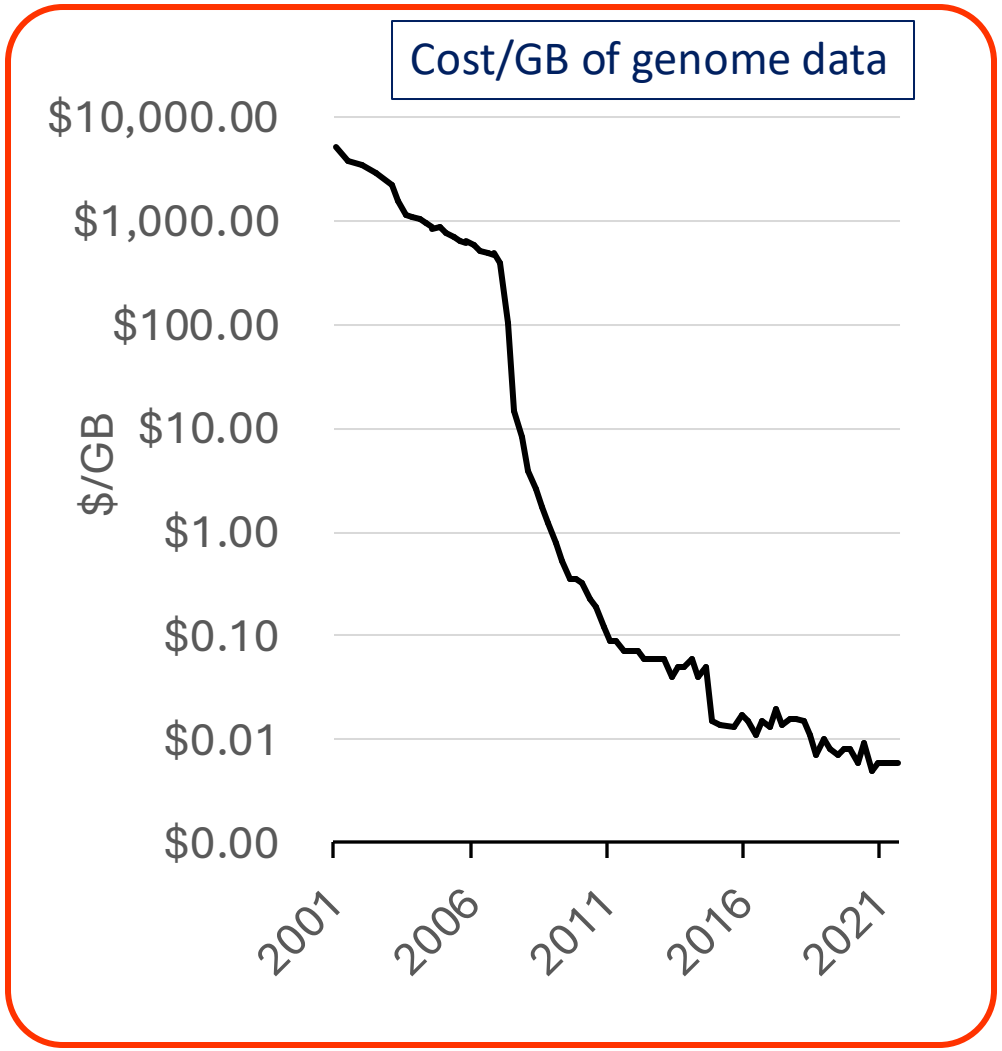
But, We Need More Performance!

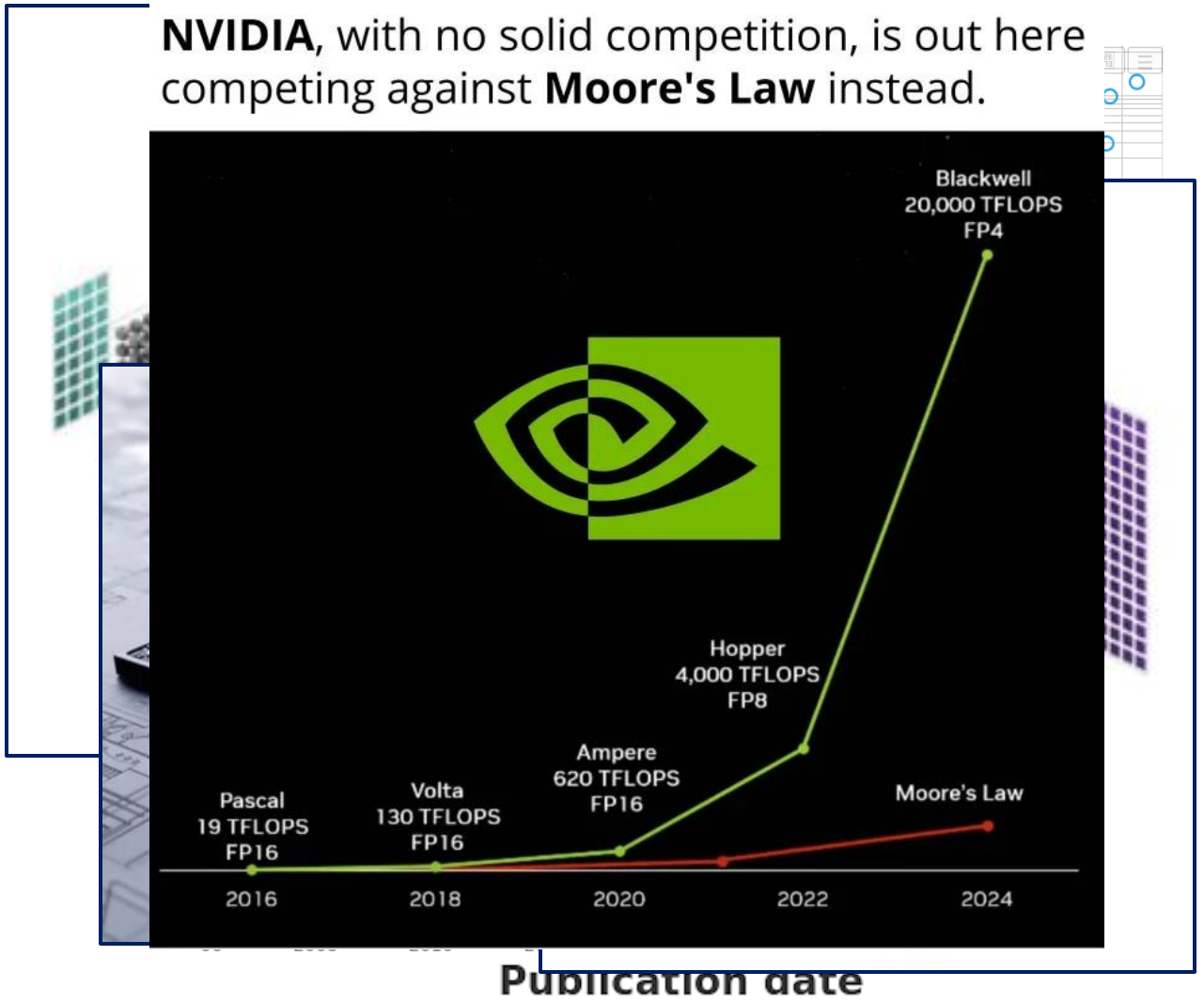




https://epoch.ai/blog/machine-learning-model-sizes-and-the-parameter-gap

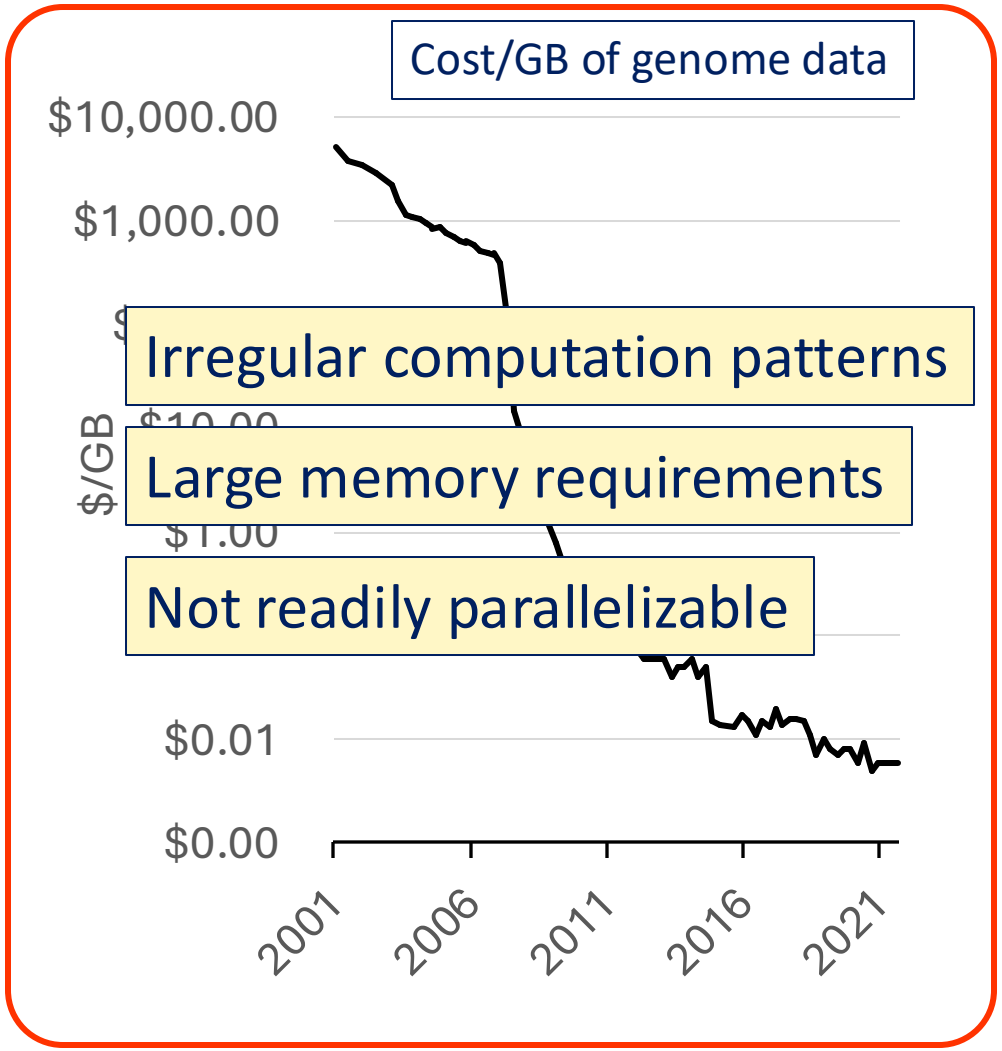
But, We Need More Performance!

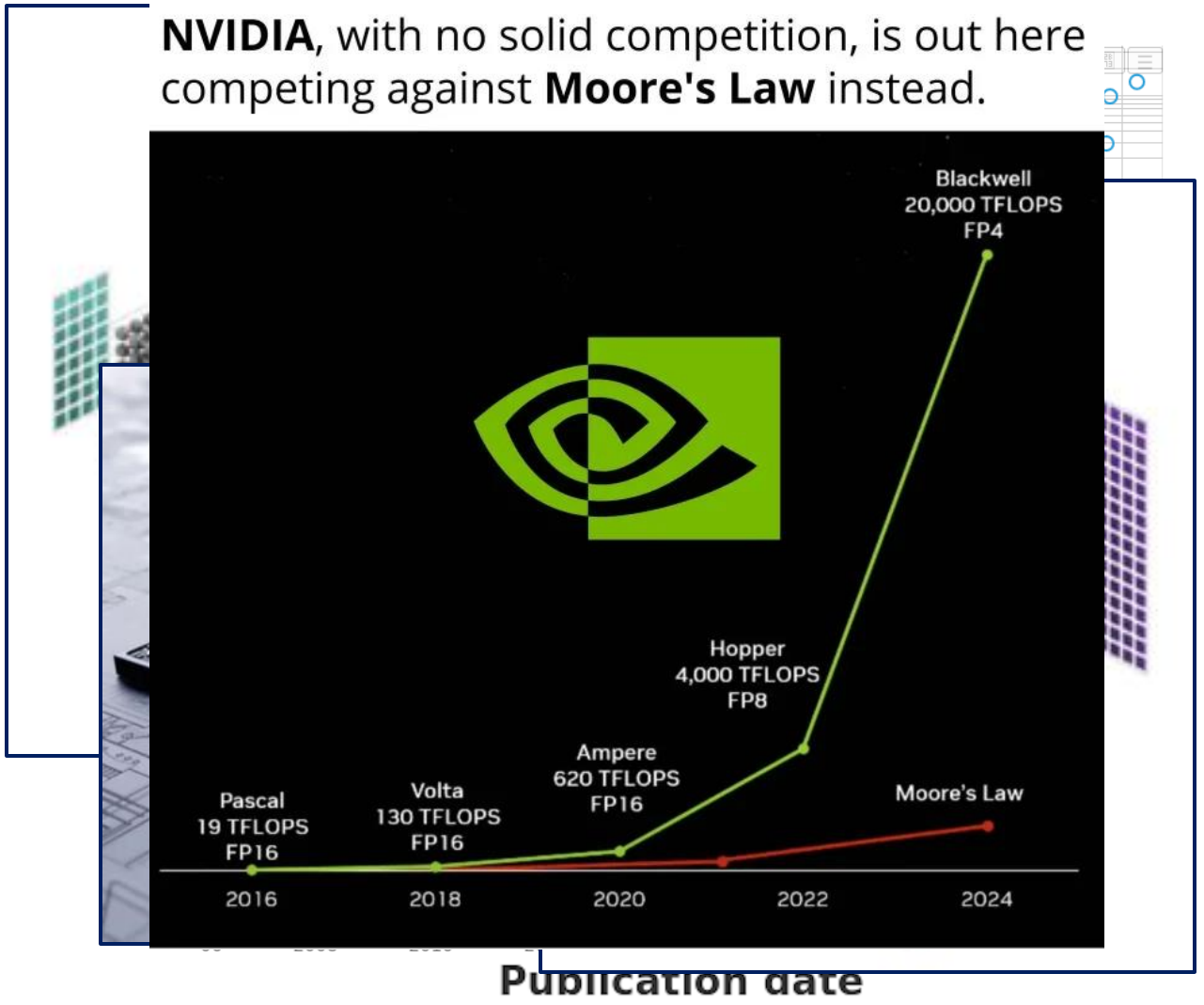




https://epoch.ai/blog/machine-learning-model-sizes-and-the-parameter-gap

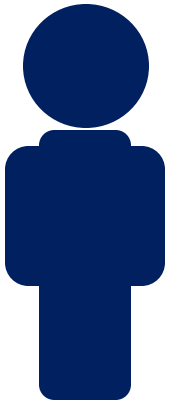
But, We Need More Performance!





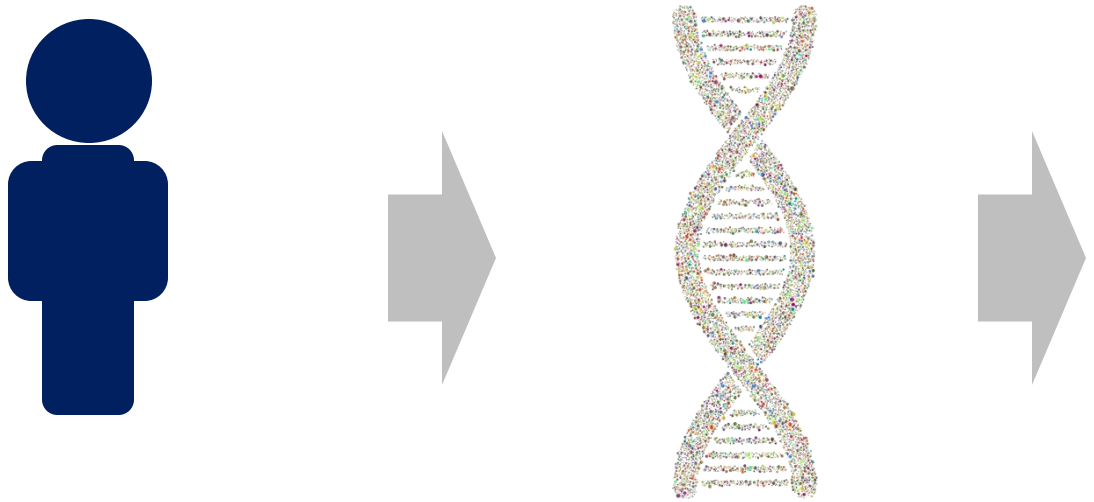
https://epoch.ai/blog/machine-learning-model-sizes-and-the-parameter-gap

Another Application Target: Precision (“Personalized”) Medicine



Cancer Patient

Another Application Target: Precision (“Personalized”) Medicine



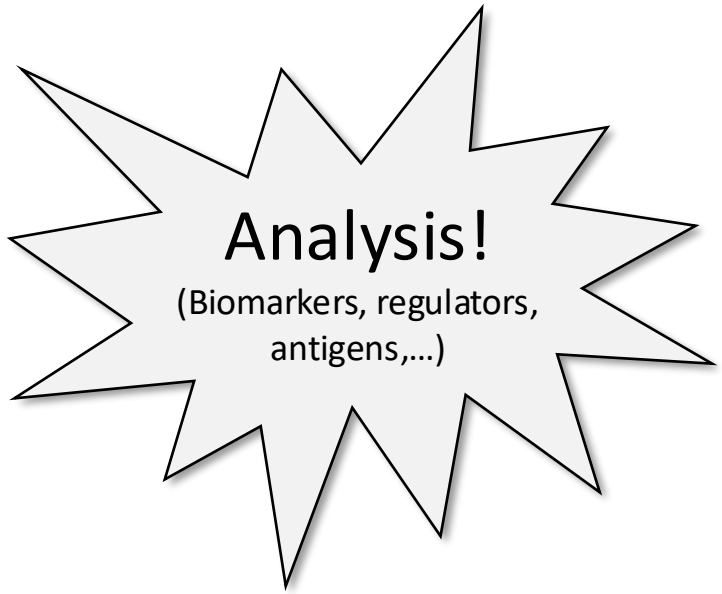
Cancer Patient

Sequenced Genome

Another Application Target: Precision ("Personalized") Medicine

Cancer Patient

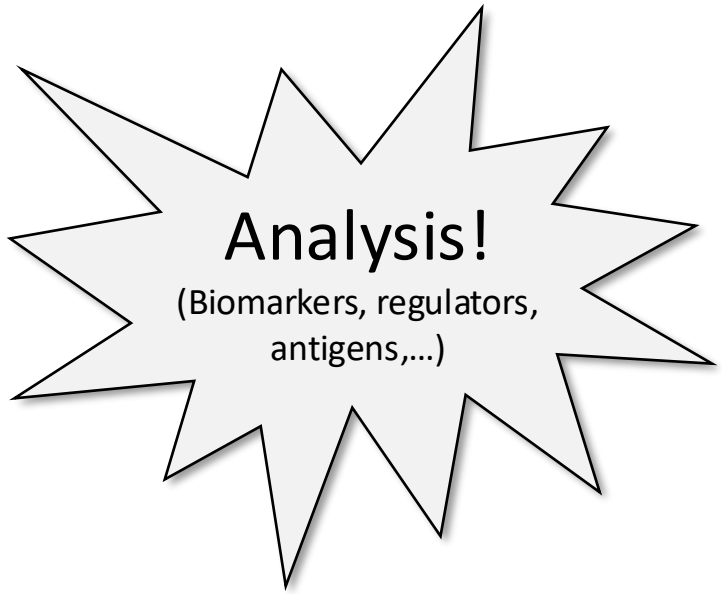
Sequenced Genome



Another Application Target: Precision ("Personalized") Medicine

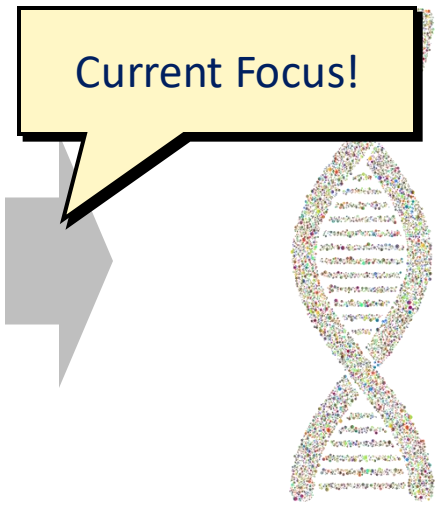
Cancer Patient

Sequenced Genome

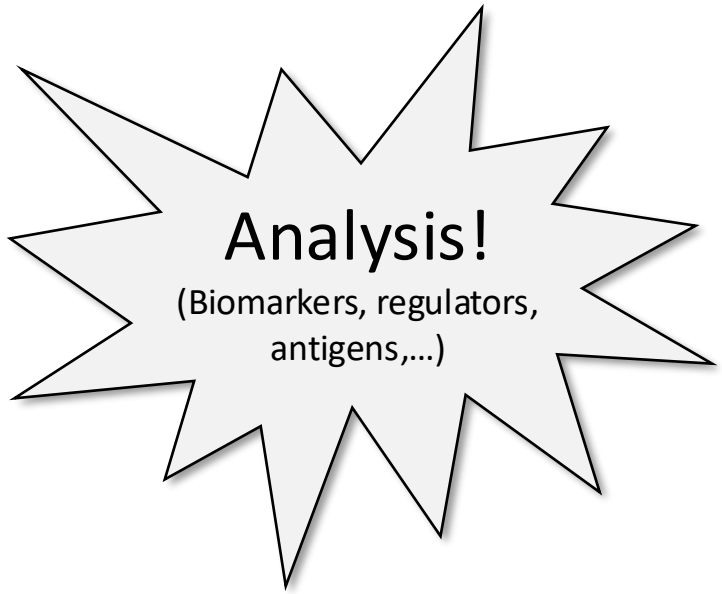


Another Application Target: Precision ("Personalized") Medicine

Cancer Patient

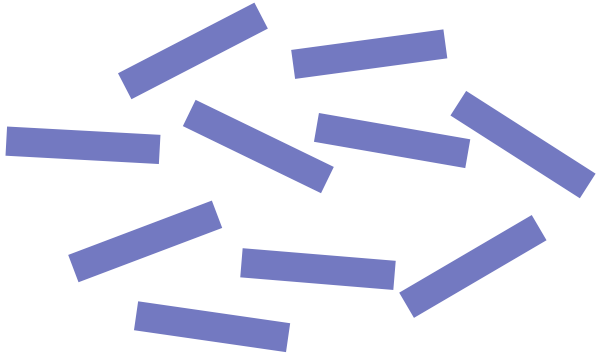


Sequenced Genome



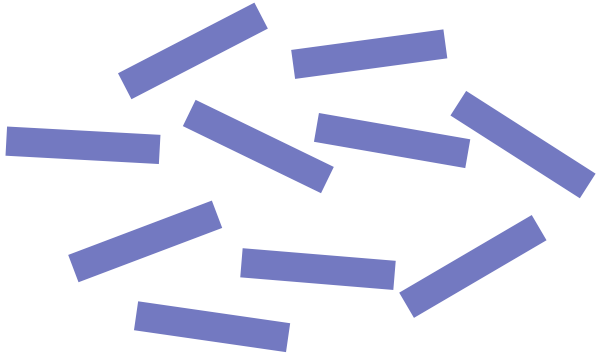
Genome Assembly Methods

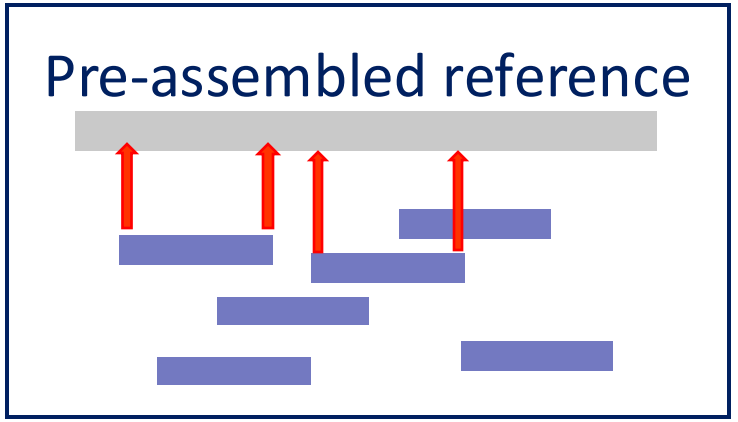
Long read samples



Genome Assembly Methods

Long read samples

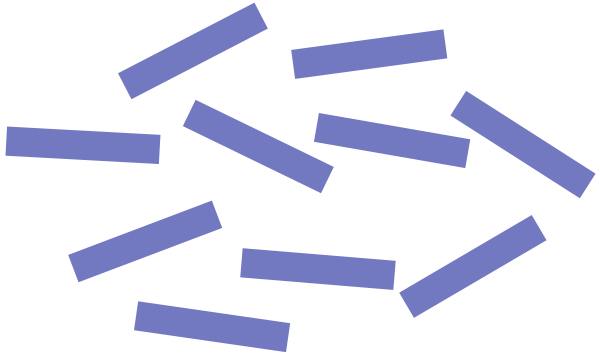




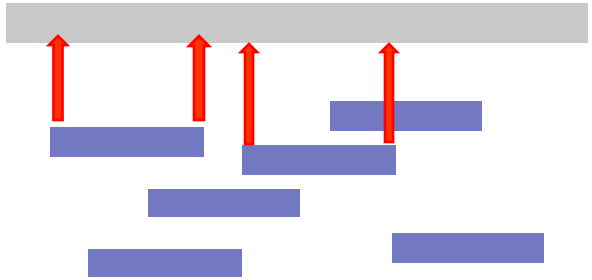
Reference-Based Assembly

Genome Assembly Methods

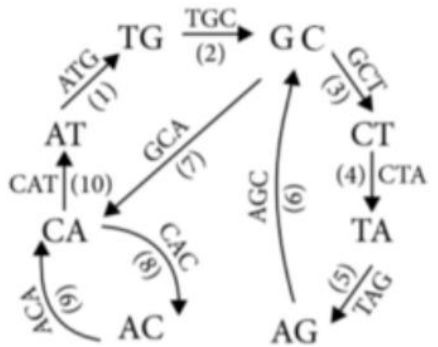
Long read samples



Pre-assembled reference



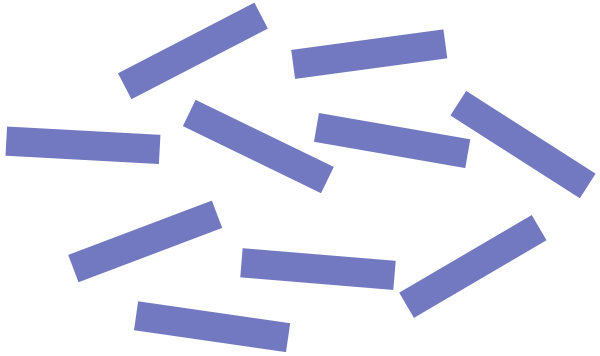
Reference-Based Assembly



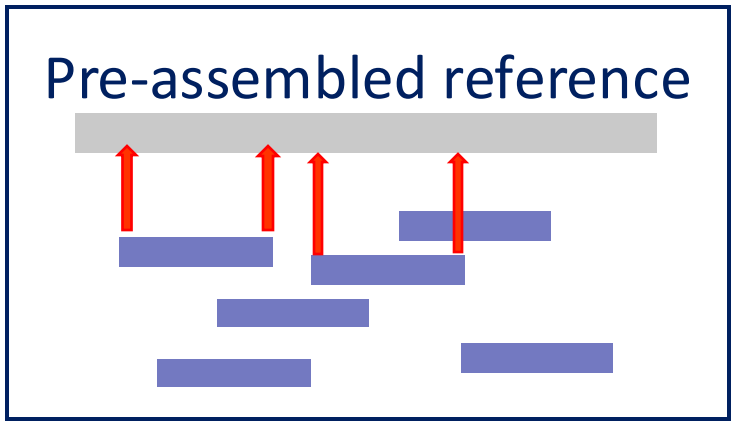
De-Novo Assembly

Genome Assembly Methods

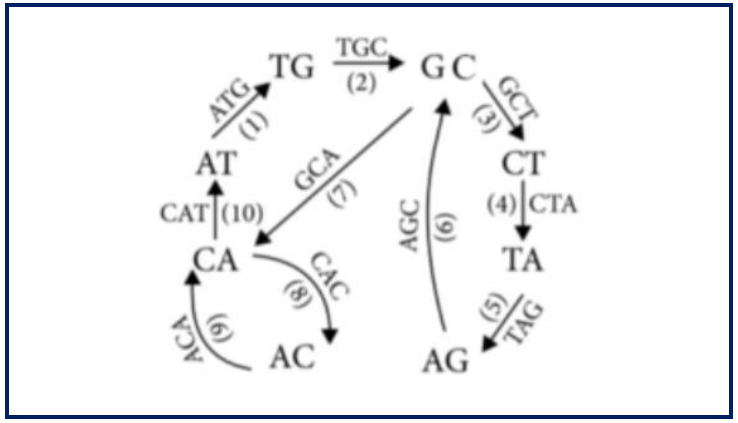
Long read samples



[1][2][3]



Reference-Based Assembly



De-Novo Assembly

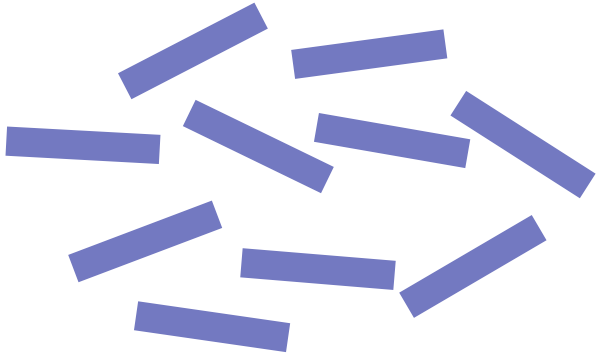
[1] Chaisson, Mark JP, Richard K. Wilson, and Evan E. Eichler. "Genetic variation and the de novo assembly of human genomes." Nature Reviews Genetics 16.11 (2015): 627-640.

[2] Ashley, Euan A. "Towards precision medicine." Nature Reviews Genetics 17.9 (2016): 507-522.

[3] Meyn, Stephen. “A critical tool for human genomics and precision medicine: De novo human genome assembly.” University of Wisconsin–Madison Research Blog

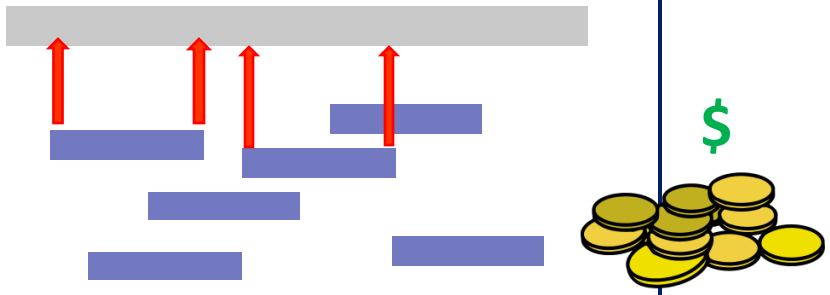
Genome Assembly Methods

Long read samples

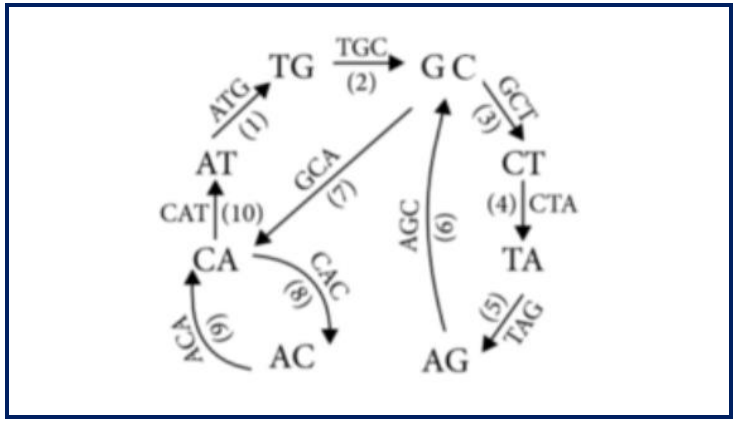


[1][2][3]

Pre-assembled reference



Reference-Based Assembly



De-Novo Assembly

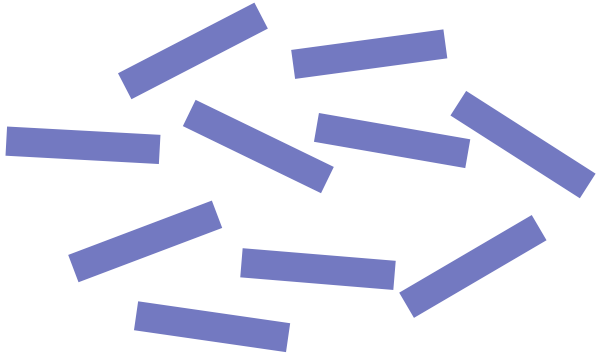
[1] Chaisson, Mark JP, Richard K. Wilson, and Evan E. Eichler. "Genetic variation and the de novo assembly of human genomes." Nature Reviews Genetics 16.11 (2015): 627-640.

[2] Ashley, Euan A. "Towards precision medicine." Nature Reviews Genetics 17.9 (2016): 507-522.

[3] Meyn, Stephen. “A critical tool for human genomics and precision medicine: De novo human genome assembly.” University of Wisconsin–Madison Research Blog

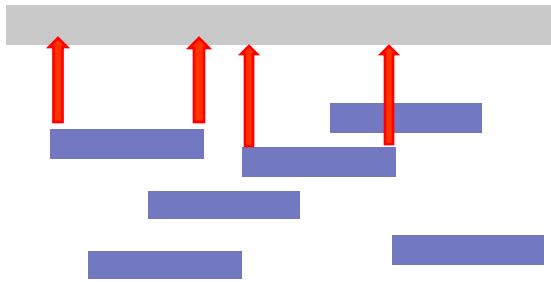
Genome Assembly Methods

Long read samples



[1][2][3]

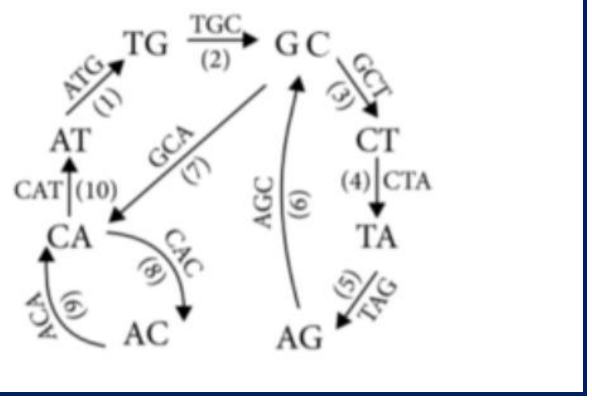
Pre-assembled reference



$

Reference-Based Assembly

$$$$



De-Novo Assembly

[1] Chaisson, Mark JP, Richard K. Wilson, and Evan E. Eichler. "Genetic variation and the de novo assembly of human genomes." Nature Reviews Genetics 16.11 (2015): 627-640.

[2] Ashley, Euan A. "Towards precision medicine." Nature Reviews Genetics 17.9 (2016): 507-522.

[3] Meyn, Stephen. “A critical tool for human genomics and precision medicine: De novo human genome assembly.” University of Wisconsin–Madison Research Blog

De Novo Assembly for Personalized Medicine

> "However, *de novo* assembly, particularly of short reads, is computationally intense and impractical for clinical genome sequencing" [2]

[2] Ashley, Euan A. "Towards precision medicine." Nature Reviews Genetics 17.9 (2016): 507-522.

De Novo Assembly for Personalized Medicine

"However, *de novo* assembly, particularly of short reads,
is computationally intense
and impractical for clinical genome sequencing" [2]

"We have been running a single NextDenovo instance
for 1 year on a 1 TB AWS instance.
We hope it will finish soon"
-- One of our research collaborators

[2] Ashley, Euan A. "Towards precision medicine." Nature Reviews Genetics 17.9 (2016): 507-522.

De Novo Assembly for Personalized Medicine

"However, *de novo* assembly, particularly of short reads,
is computationally intense
and impractical for clinical genome sequencing" [2]

"We have been running a single NextDenovo instance
for 1 year on a 1 TB AWS instance.
We hope it will finish soon"
-- One of our research collaborators

Hurrah! A systems research problem!

[2] Ashley, Euan A. "Towards precision medicine." Nature Reviews Genetics 17.9 (2016): 507-522.

What to accelerate, for De Novo Assembly?

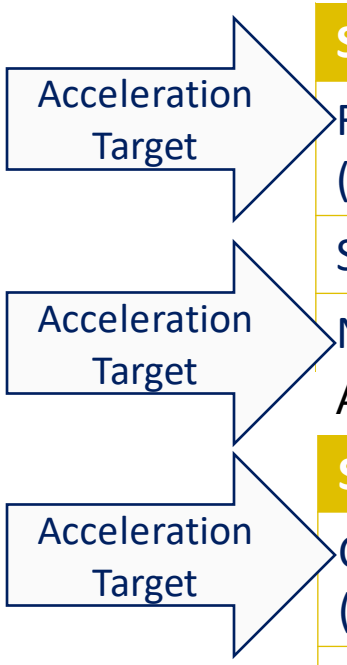
Correction

Step	Mem (GB)	Time (s)
Raw_align (minimap2)	9	2,203
Sort	< 9	176
Next_correct	< 9	1,851

Alignment

Step	Mem (GB)	Time (s)
Cns_align (minimap2)	8	3,907
Ctg_graph	< 8	7.8
Ctg_align (minimap2)	12	390
Ctg_cns	40	58

What to accelerate, for De Novo Assembly?



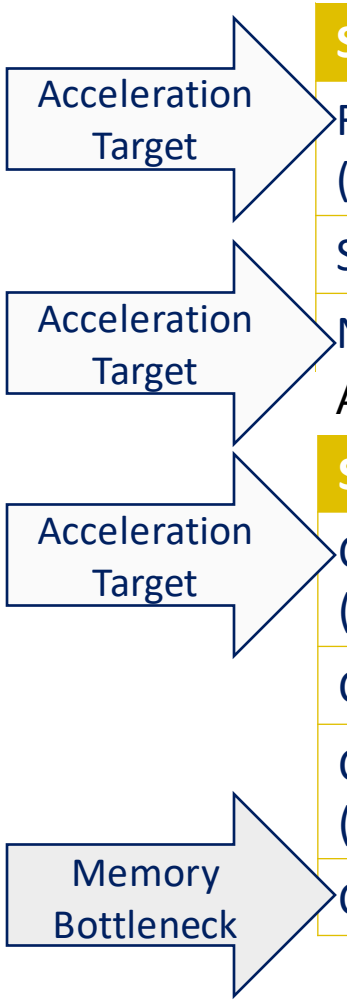
Correction

Step	Mem (GB)	Time (s)
Raw_align (minimap2)	9	2,203
Sort	< 9	176
Next_correct	< 9	1,851

Alignment

Step	Mem (GB)	Time (s)
Cns_align (minimap2)	8	3,907
Ctg_graph	< 8	7.8
Ctg_align (minimap2)	12	390
Ctg_cns	40	58

What to accelerate, for De Novo Assembly?



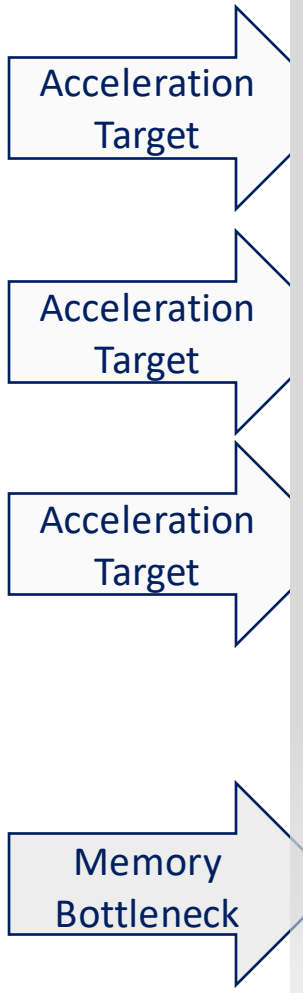
Correction

Step	Mem (GB)	Time (s)
Raw_align (minimap2)	9	2,203
Sort	< 9	176
Next_correct	< 9	1,851

Alignment

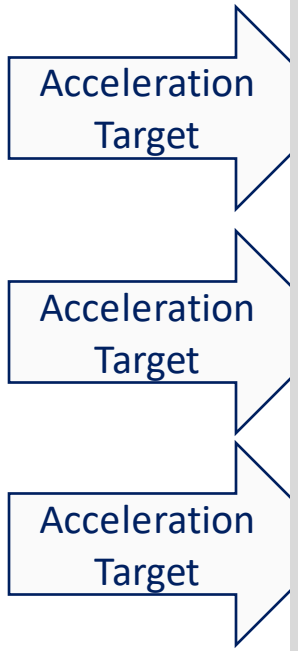
Step	Mem (GB)	Time (s)
Cns_align (minimap2)	8	3,907
Ctg_graph	< 8	7.8
Ctg_align (minimap2)	12	390
Ctg_cns	40	58

What to accelerate, for De Novo Assembly?



Ctg_align (minimap2)	12	390
Ctg_cns	40	58

What to accelerate, for De Novo Assembly?



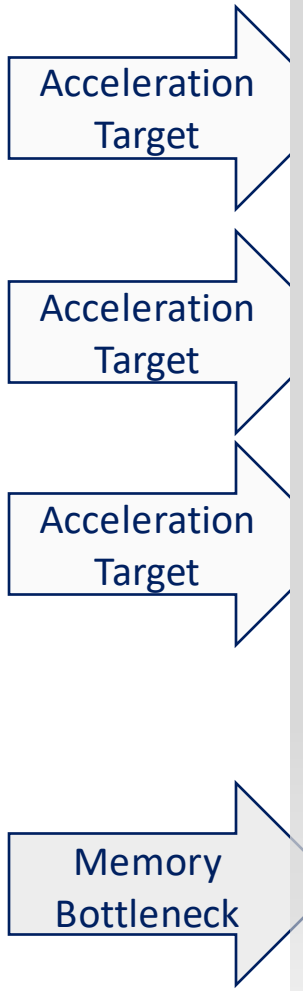
Ctg_align (minimap2)	12	390
Ctg_cns	40	58

Goal 1: Graph Construction and Traversal

What to accelerate, for De Novo Assembly?

Goal 2: “Minimap 2”
For N-to-N genome alignment

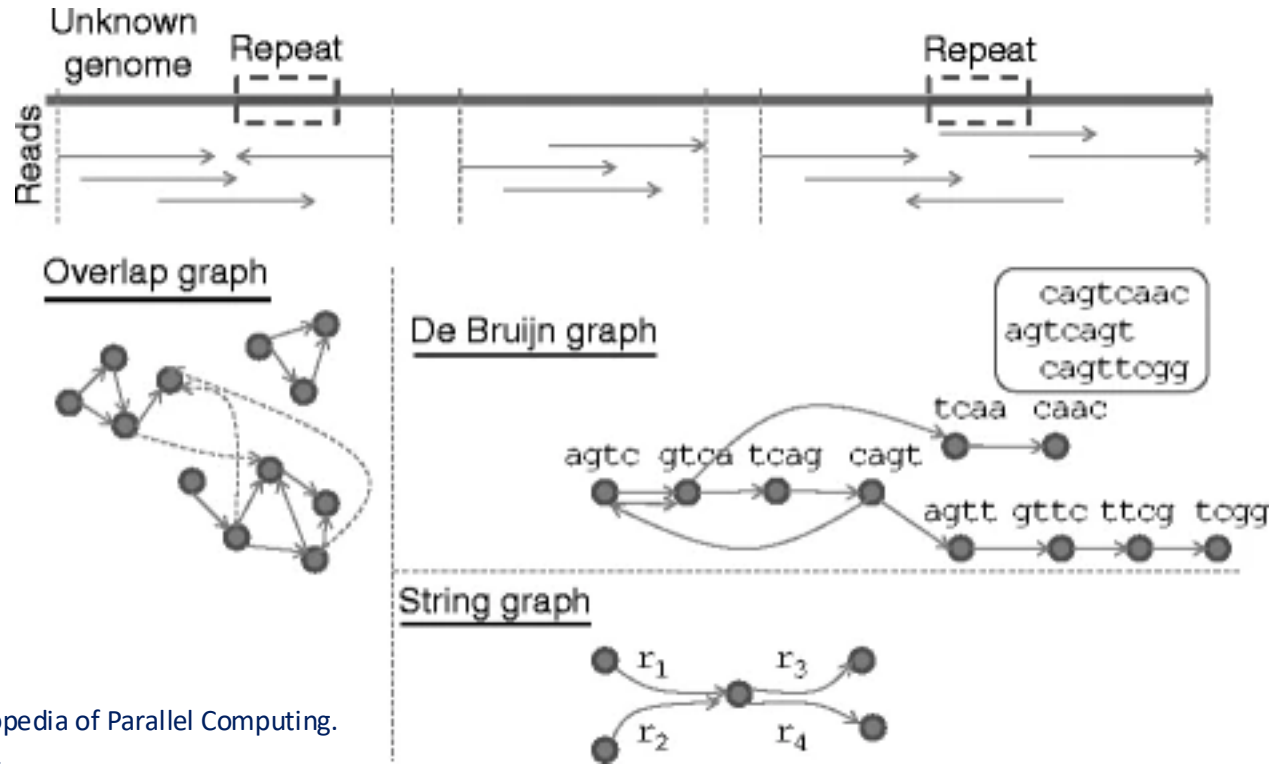
Goal 1: Graph Construction and Traversal



Ctg_align (minimap2)	12	390
Ctg_cns	40	58

Graph Analytics in De Novo Assembly

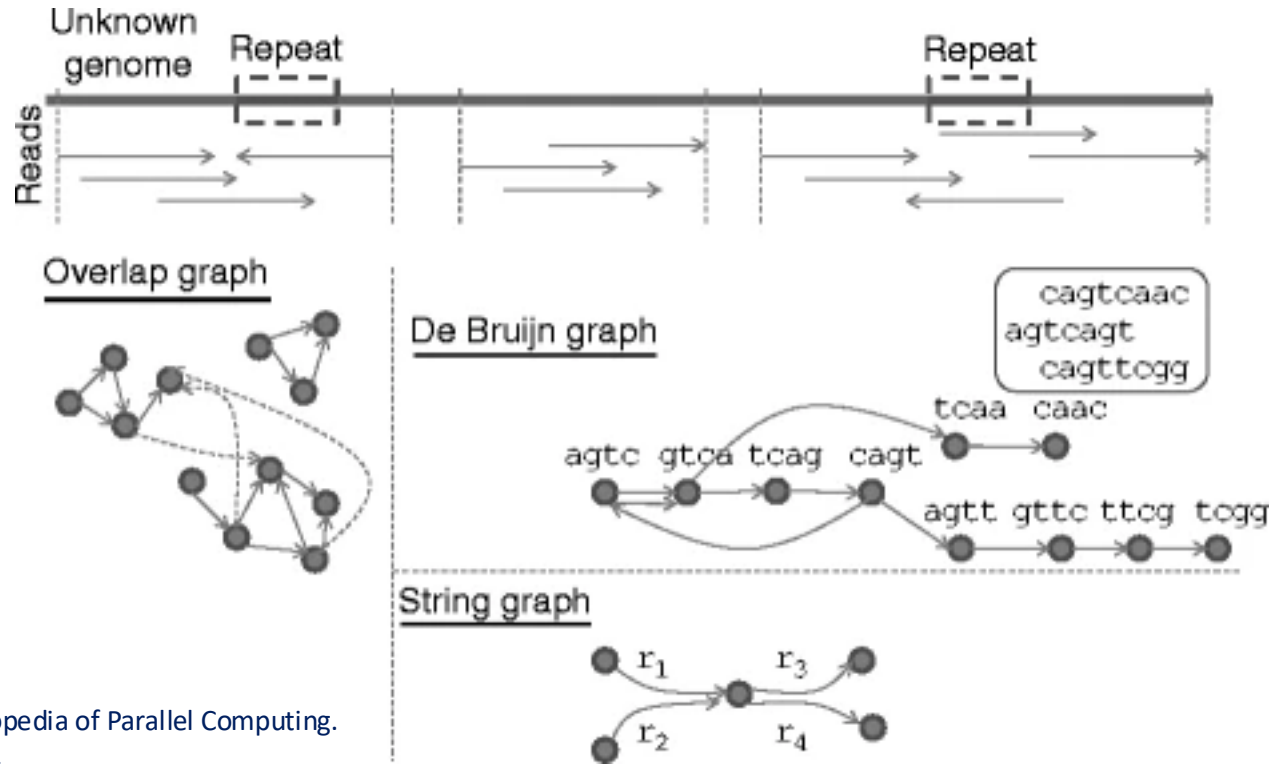
- Big source of scalability concerns: Handling graphs
  - Overlap graphs, De Bruijn Graphs, String Graphs, …



Kalyanaraman, A. (2011). Genome Assembly. In: Padua, D. (eds) Encyclopedia of Parallel Computing. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-09766-4_402

Graph Analytics in De Novo Assembly

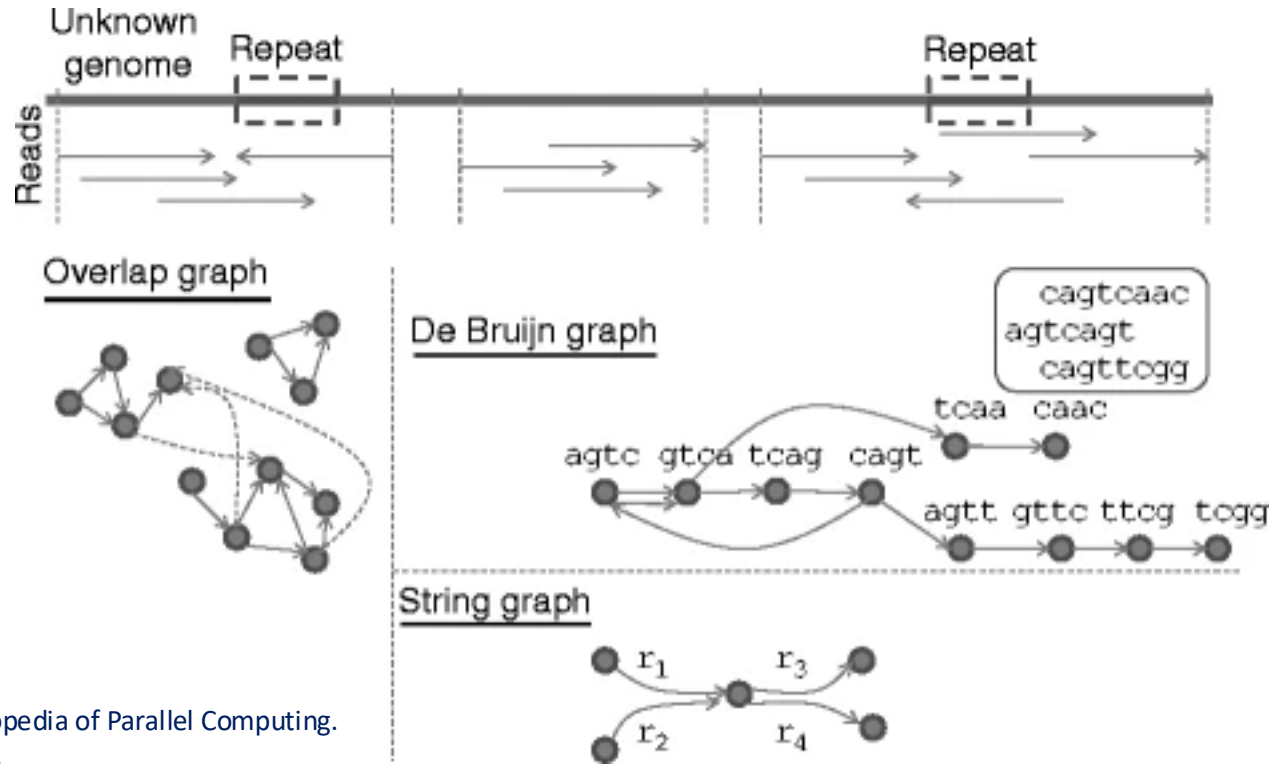
- Big source of scalability concerns: Handling graphs
  - Overlap graphs, De Bruijn Graphs, String Graphs, …
  - Quite large!
    - +500 GB for Human
    - TBs for some plants (Pine, Onion, …)



Kalyanaraman, A. (2011). Genome Assembly. In: Padua, D. (eds) Encyclopedia of Parallel Computing. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-09766-4_402

Graph Analytics in De Novo Assembly

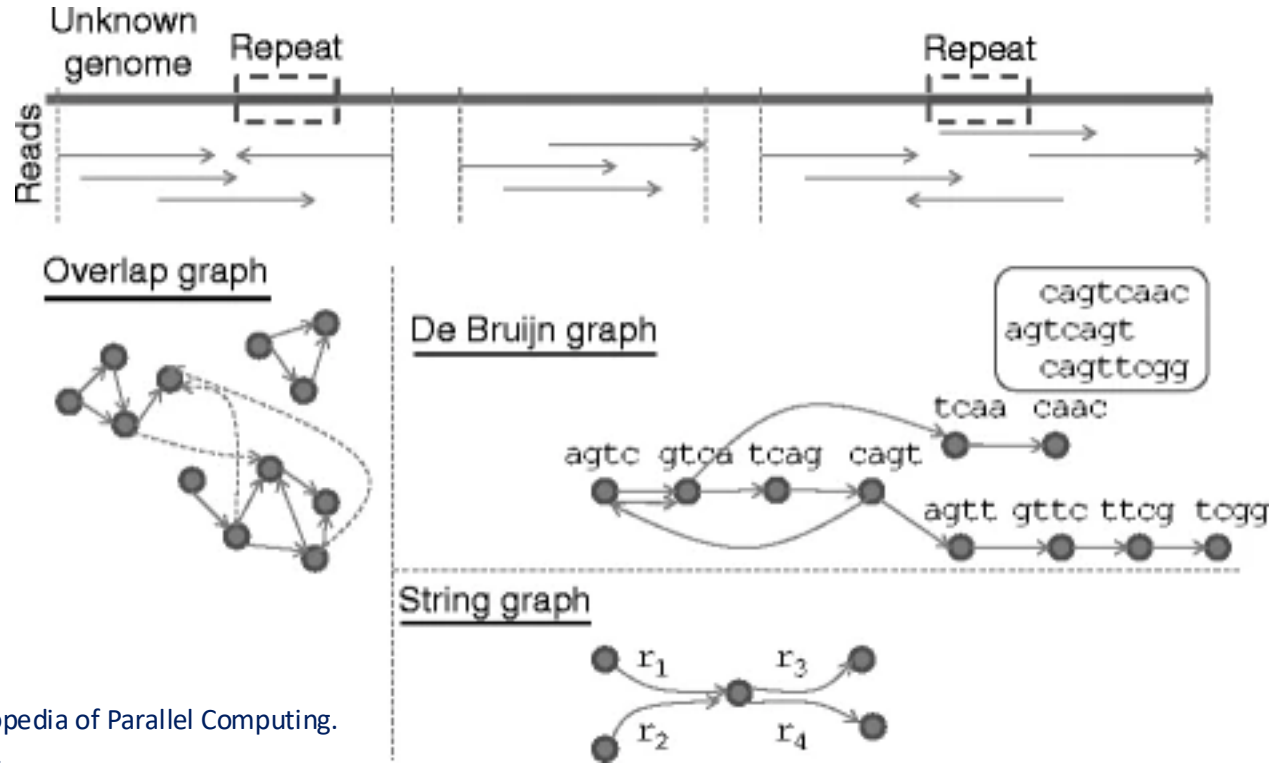
- Big source of scalability concerns: Handling graphs
  - Overlap graphs, De Bruijn Graphs, String Graphs, …
  - Quite large!
    - +500 GB for Human
    - TBs for some plants (Pine, Onion, …)
  - Vertices are small (few bytes)



Kalyanaraman, A. (2011). Genome Assembly. In: Padua, D. (eds) Encyclopedia of Parallel Computing. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-09766-4_402

Graph Analytics in De Novo Assembly

- Big source of scalability concerns: Handling graphs
  - Overlap graphs, De Bruijn Graphs, String Graphs, …
  - Quite large!
    - +500 GB for Human
    - TBs for some plants (Pine, Onion, …)
  - Vertices are small (few bytes)
  - Construct, then traverse



Kalyanaraman, A. (2011). Genome Assembly. In: Padua, D. (eds) Encyclopedia of Parallel Computing. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-09766-4_402

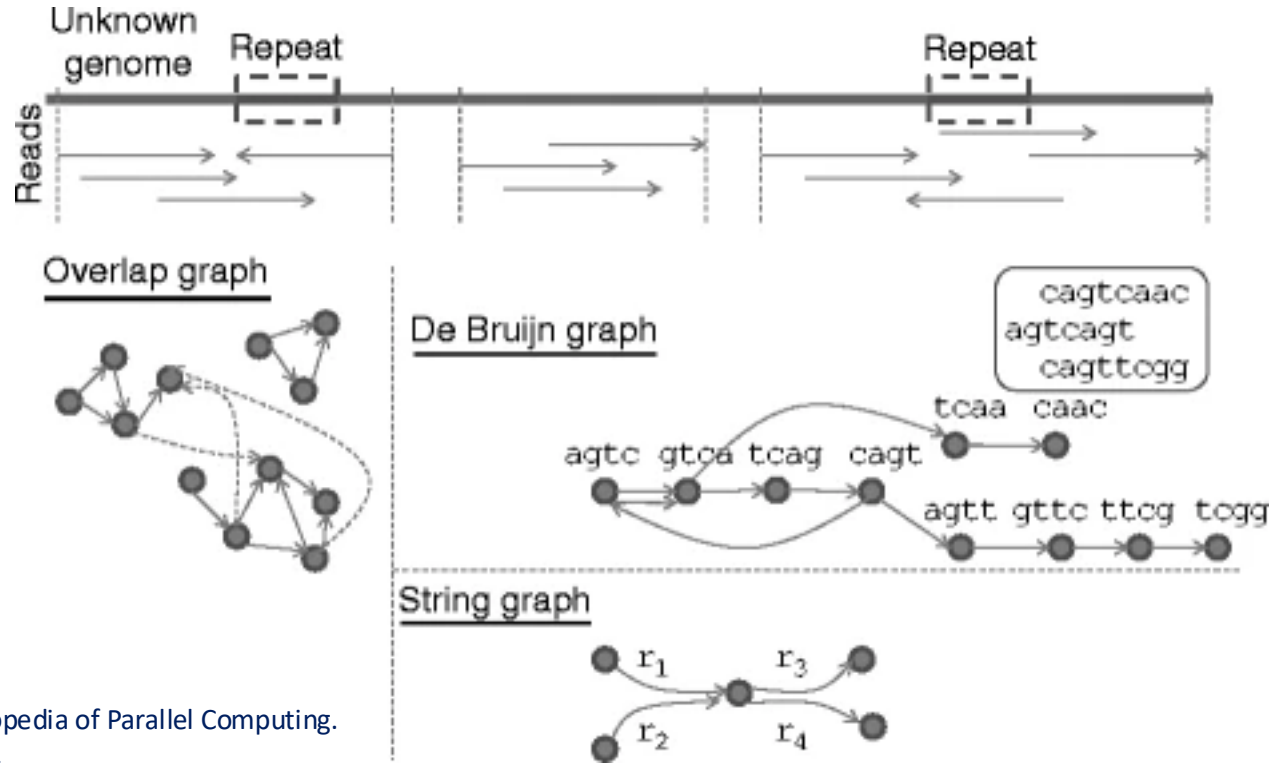
Graph Analytics in De Novo Assembly

- Big source of scalability concerns: Handling graphs
  - Overlap graphs, De Bruijn Graphs, String Graphs, …
  - Quite large!
    - +500 GB for Human
    - TBs for some plants (Pine, Onion, …)
  - Vertices are small (few bytes)
  - Construct, then traverse

Irregular computation patterns

Large memory requirements

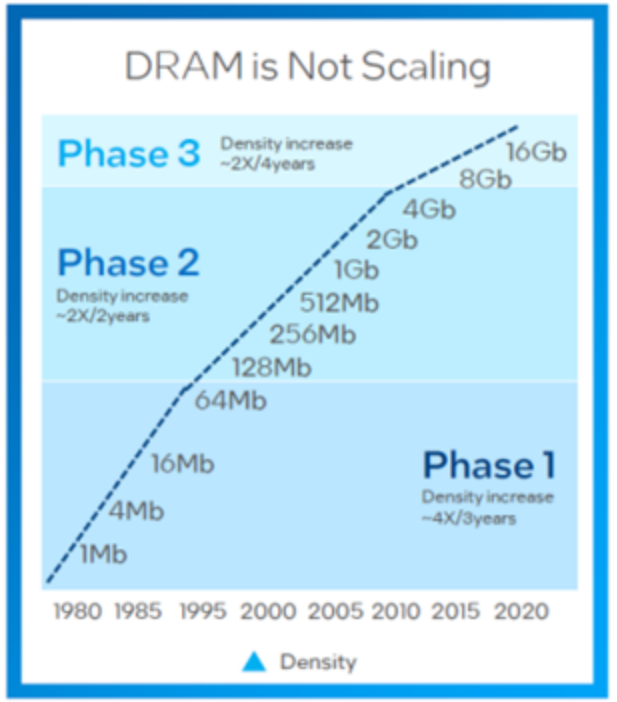
Not readily parallelizable

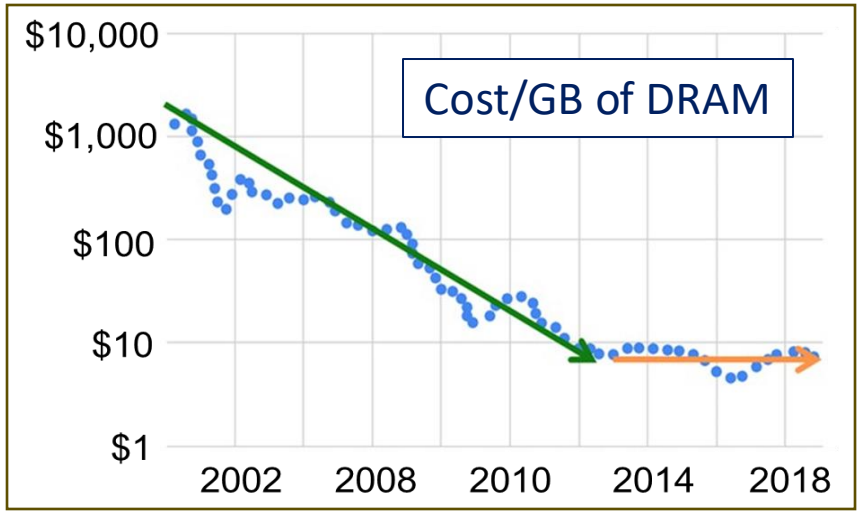


Kalyanaraman, A. (2011). Genome Assembly. In: Padua, D. (eds) Encyclopedia of Parallel Computing. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-09766-4_402

High-Performance Graph Analytics in SSDs

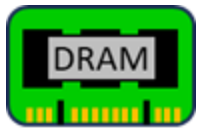
- Slowing DRAM density scaling
  - Graphs scaling faster than memory can!
- SSDs are cheaper… Can we use those instead?





High-Performance Graph Analytics in SSDs

- Slowing DRAM density scaling
  - Graphs scaling faster than memory can!
- SSDs are cheaper… Can we use those instead?

DDR5 

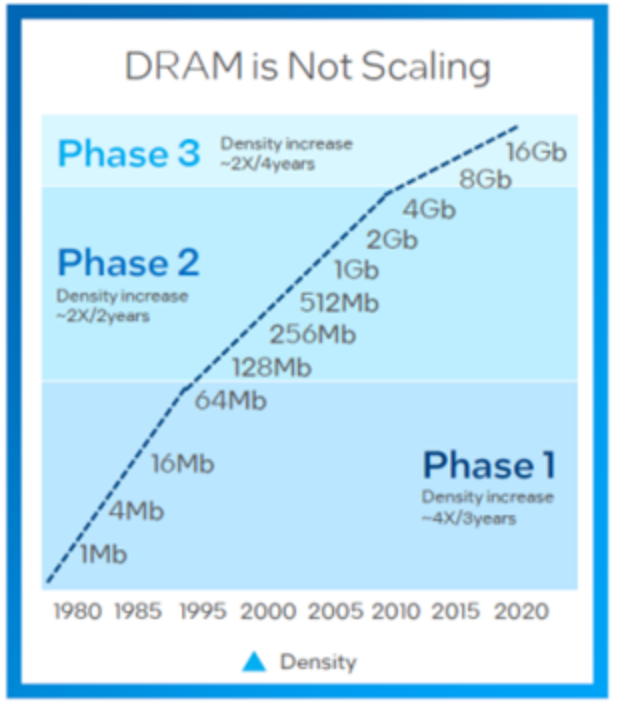
$3,000/TB

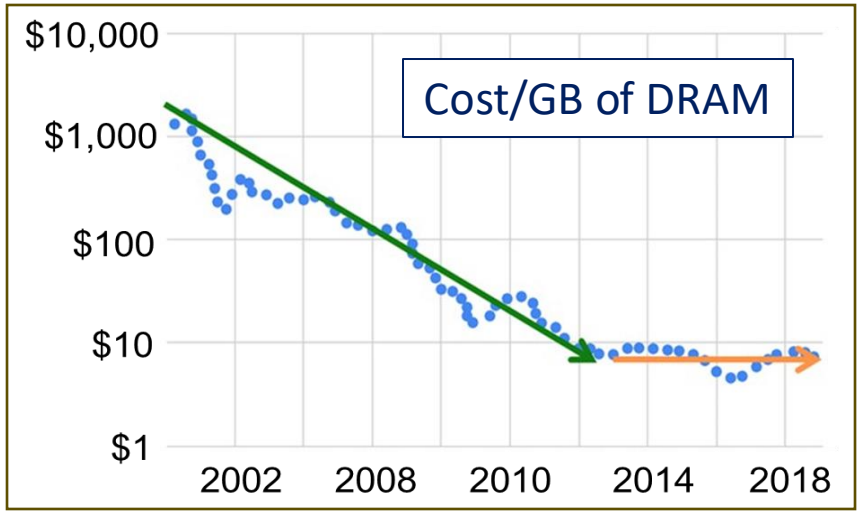
200 W/TB

NVMe 

$50/TB

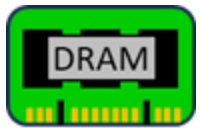
<10 W/TB





High-Performance Graph Analytics in SSDs

- Slowing DRAM density scaling
  - Graphs scaling faster than memory can!
- SSDs are cheaper… Can we use those instead?

DDR5 

$3,000/TB

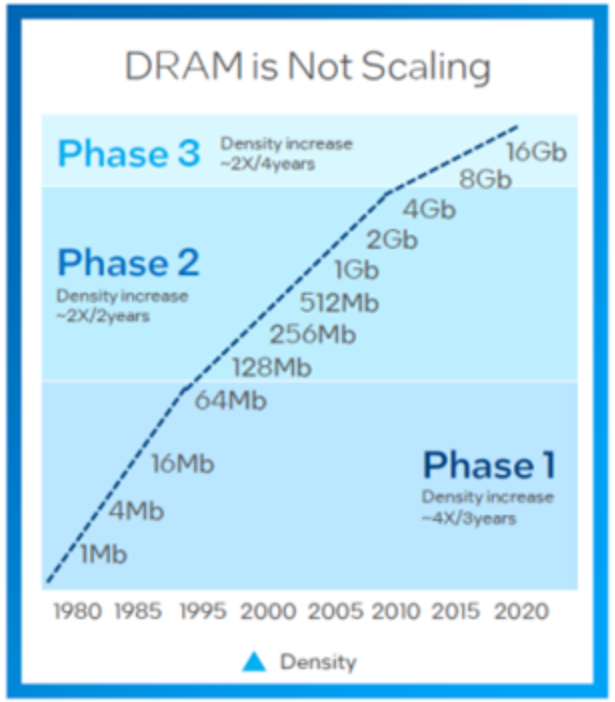
200 W/TB

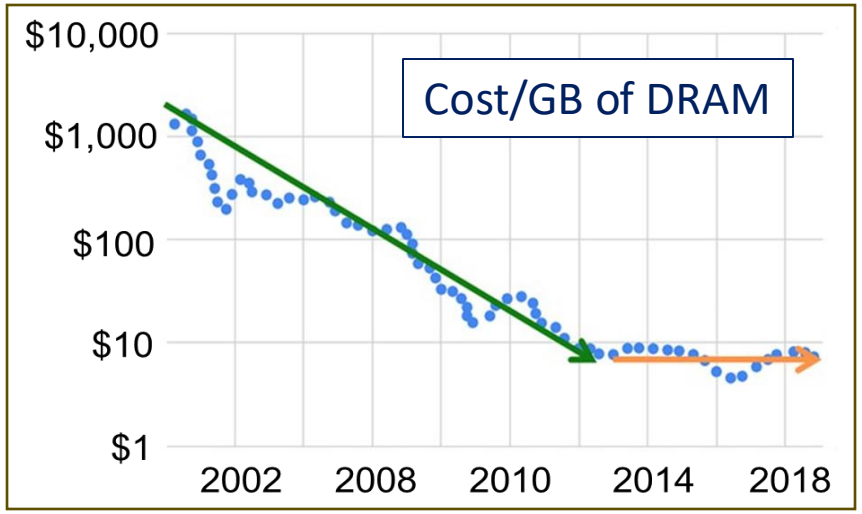
NVMe 

$50/TB

<10 W/TB

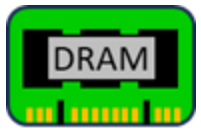
Unfortunately, they are also slow…





High-Performance Graph Analytics in SSDs

- Slowing DRAM density scaling
  - Graphs scaling faster than memory can!
- SSDs are cheaper… Can we use those instead?

DDR5 

$3,000/TB

200 W/TB

NVMe 

$50/TB

<10 W/TB

Unfortunately, they are also slow…

400+ GB/s

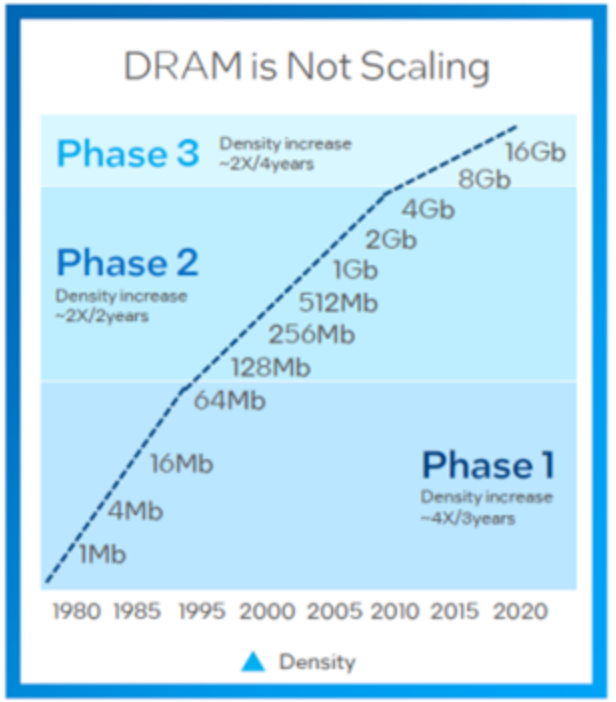
~10 **ns**

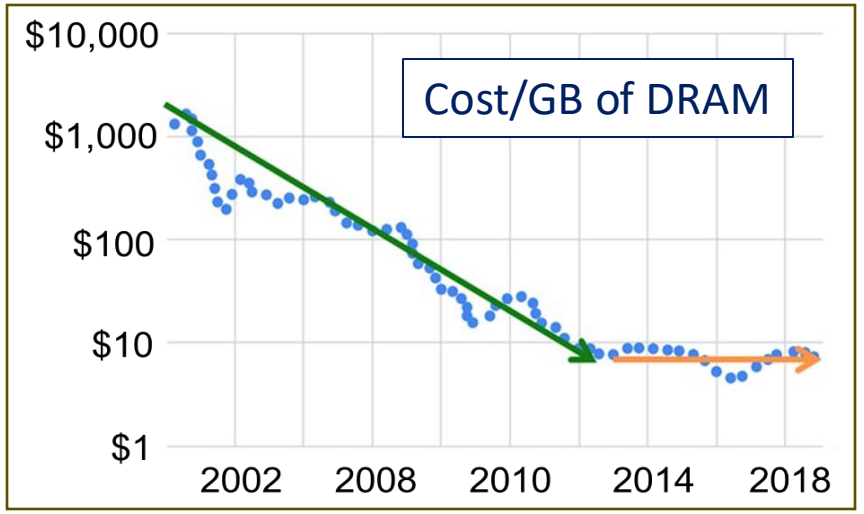
64 Byte cache lines

10+ GB/s

~10 **µs**

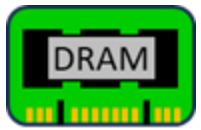
8 – 64 KB pages





High-Performance Graph Analytics in SSDs

- Slowing DRAM density scaling
  - Graphs scaling faster than memory can!
- SSDs are cheaper… Can we use those instead?

DDR5 

$3,000/TB

200 W/TB

NVMe 

$50/TB

<10 W/TB

Unfortunately, they are also slow…

400+ GB/s

~10 **ns**

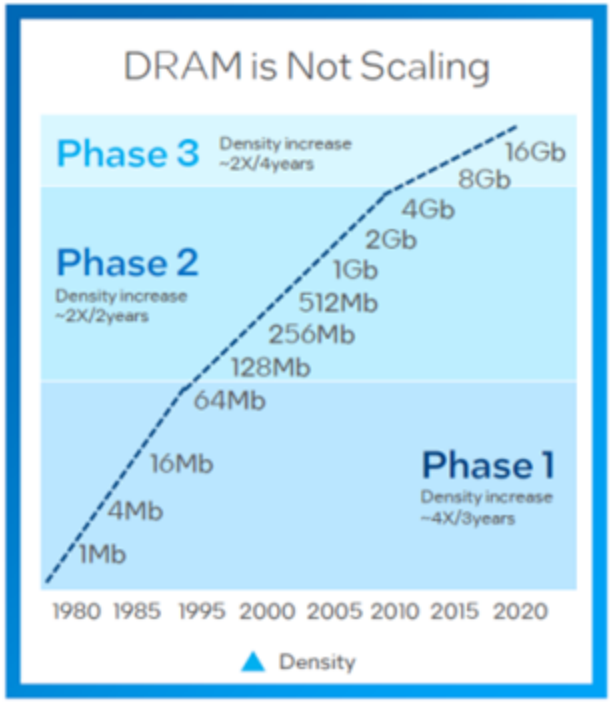
64 Byte cache lines

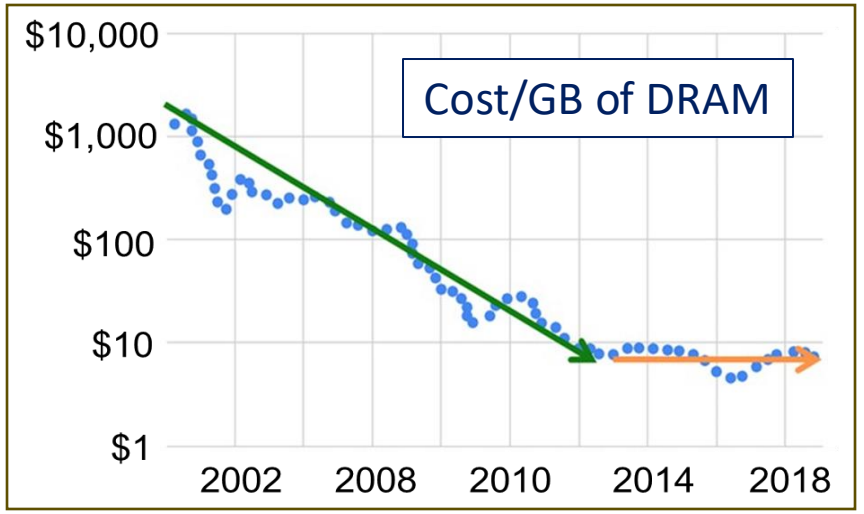
10+ GB/s

~10 **µs**

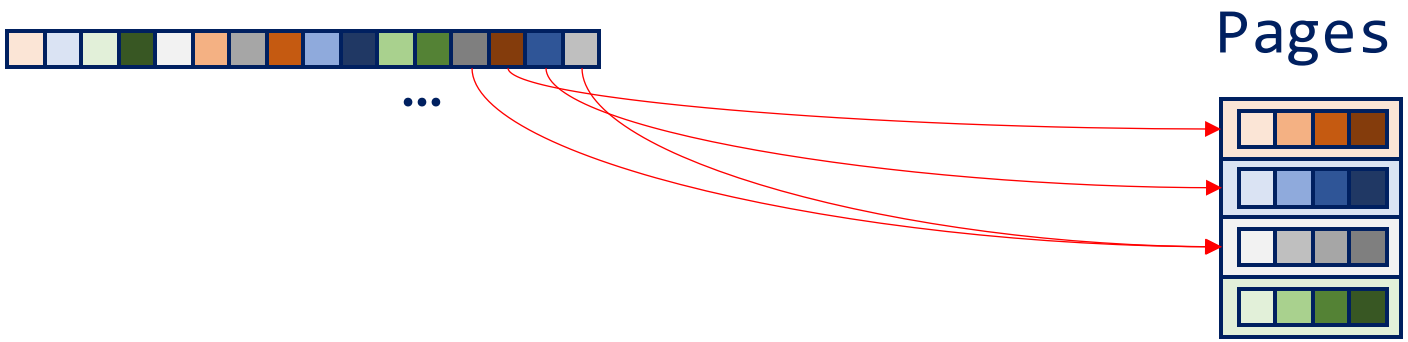
8 – 64 KB pages

I/O amplification for random accesses ☹

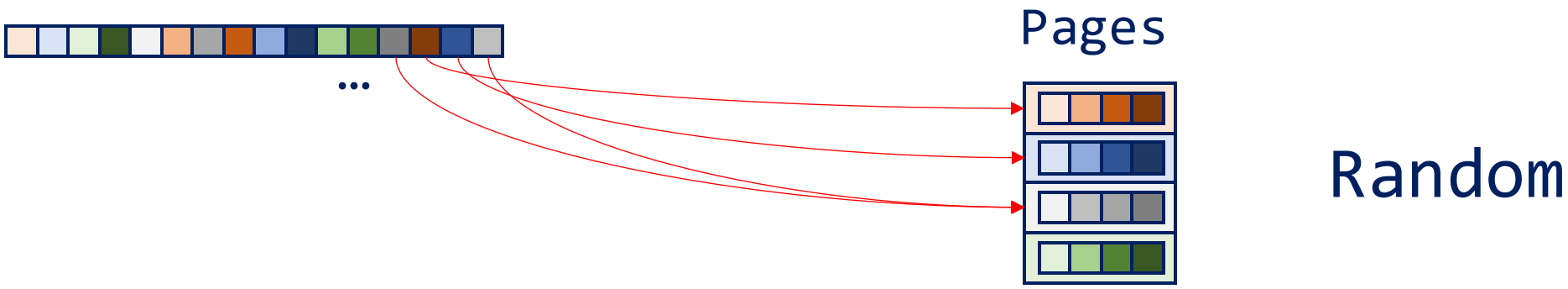




Our Solution: Sorting Accesses [Kang et. al., DAC 2024]

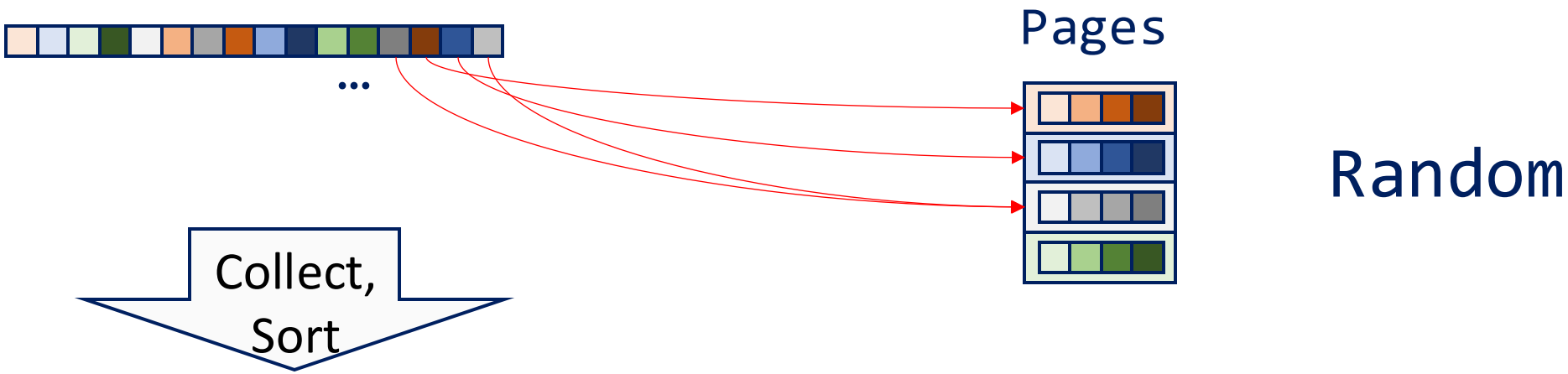


Our Solution: Sorting Accesses [Kang et. al., DAC 2024]

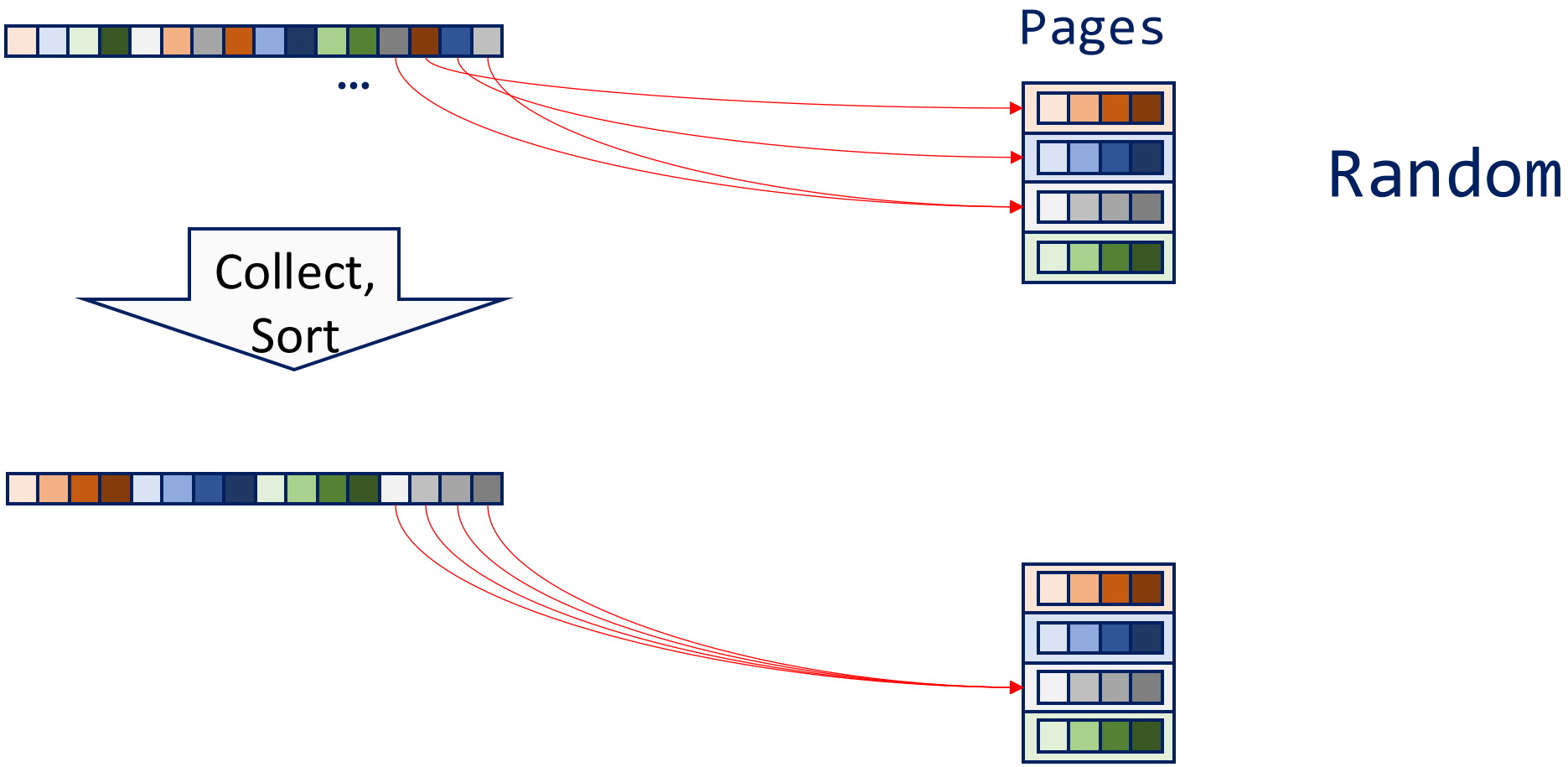


Random

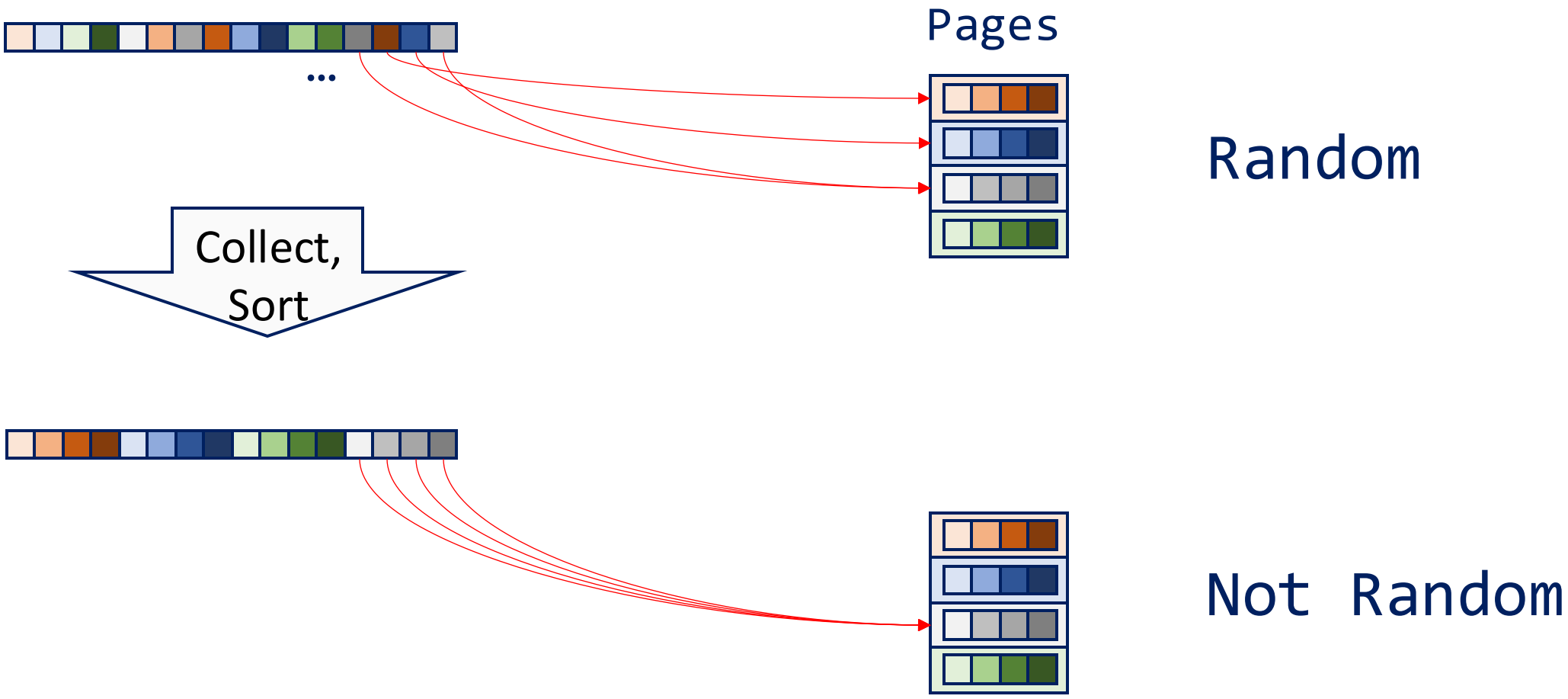
Our Solution: Sorting Accesses [Kang et. al., DAC 2024]



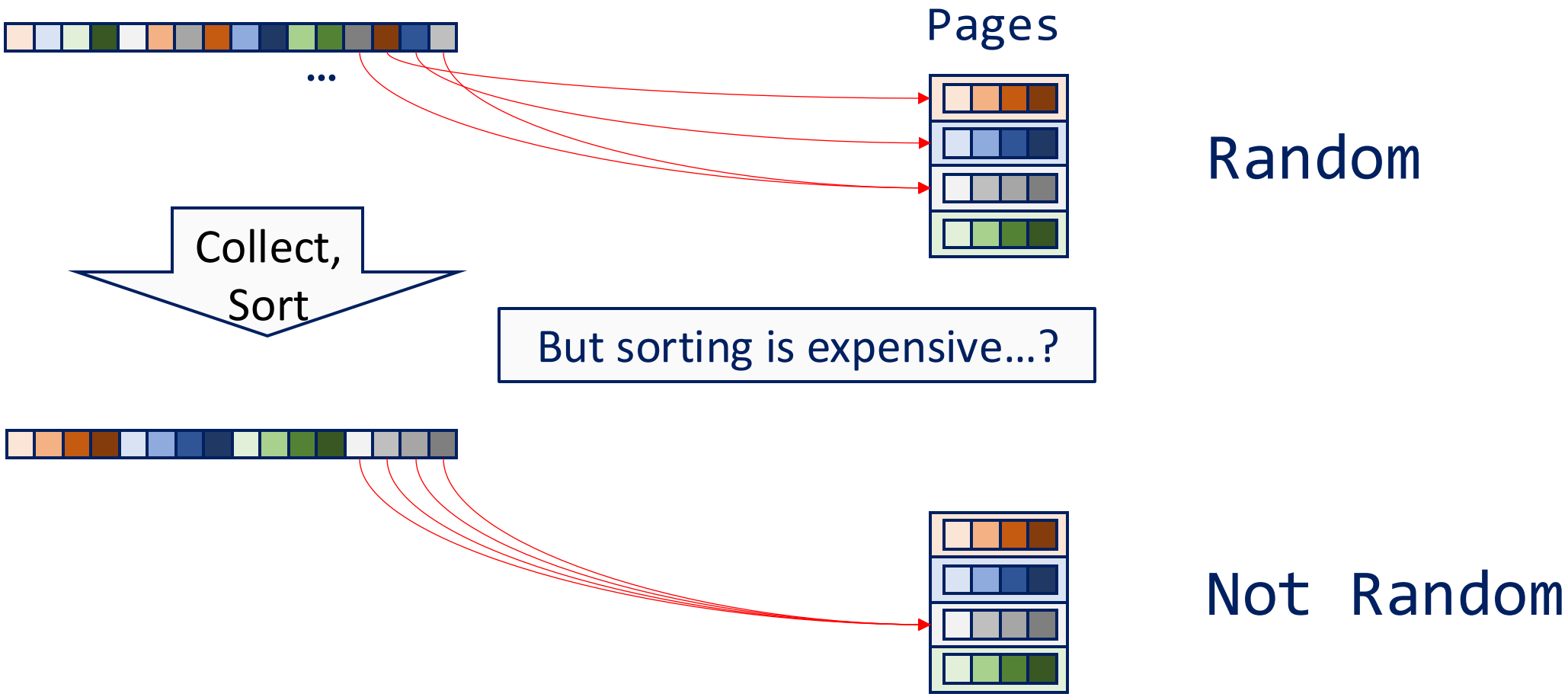
Our Solution: Sorting Accesses [Kang et. al., DAC 2024]



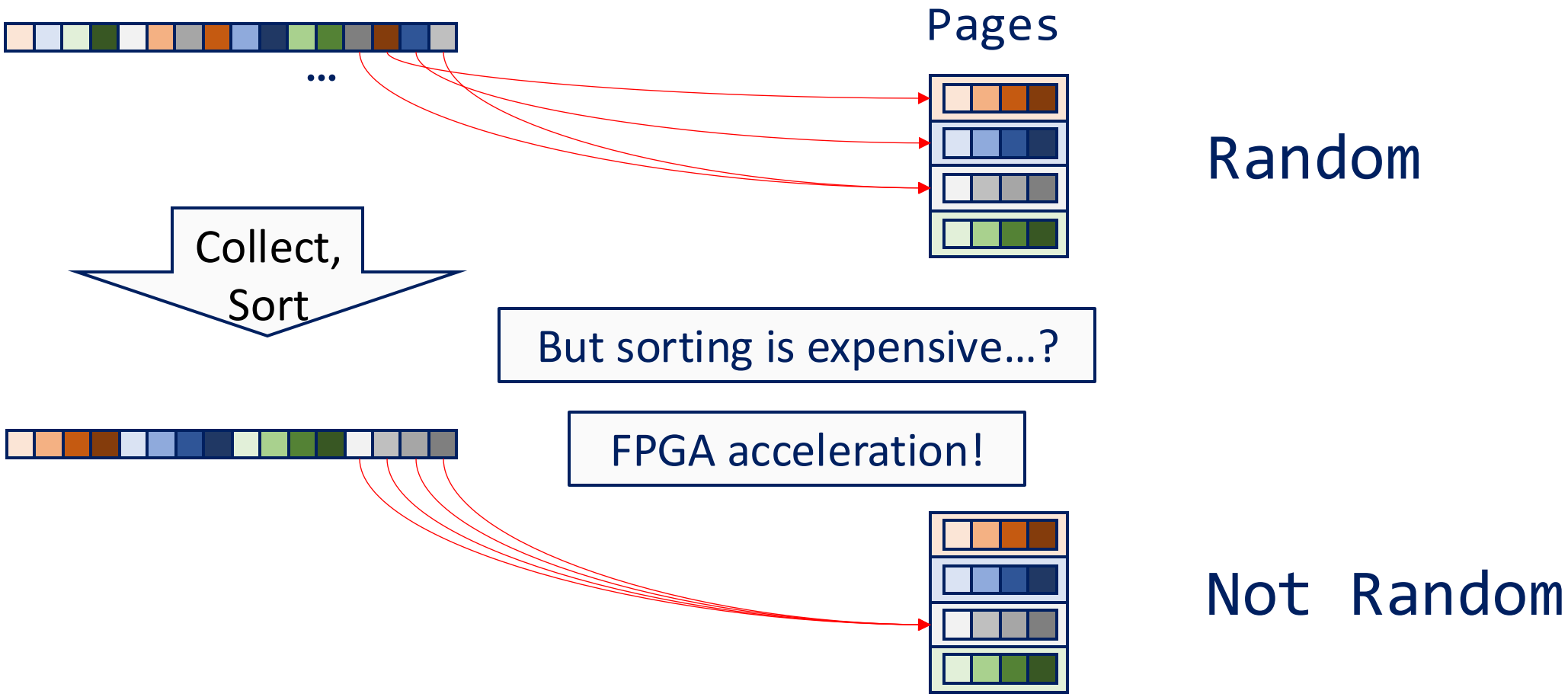
Our Solution: Sorting Accesses [Kang et. al., DAC 2024]



Our Solution: Sorting Accesses [Kang et. al., DAC 2024]

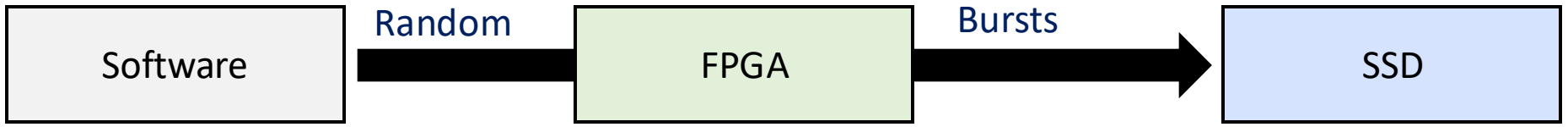


Our Solution: Sorting Accesses [Kang et. al., DAC 2024]

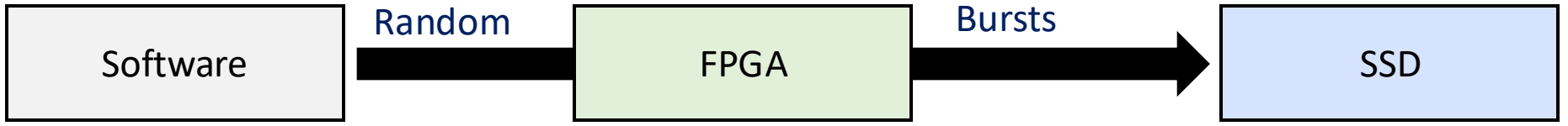


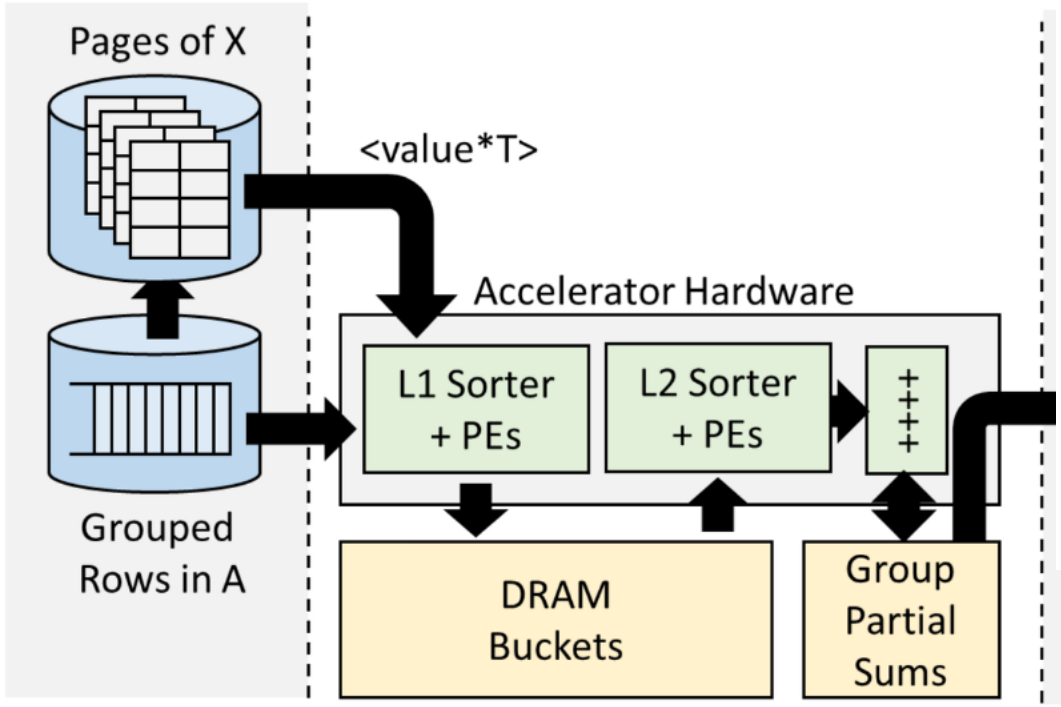
FPGA Radix Sorter

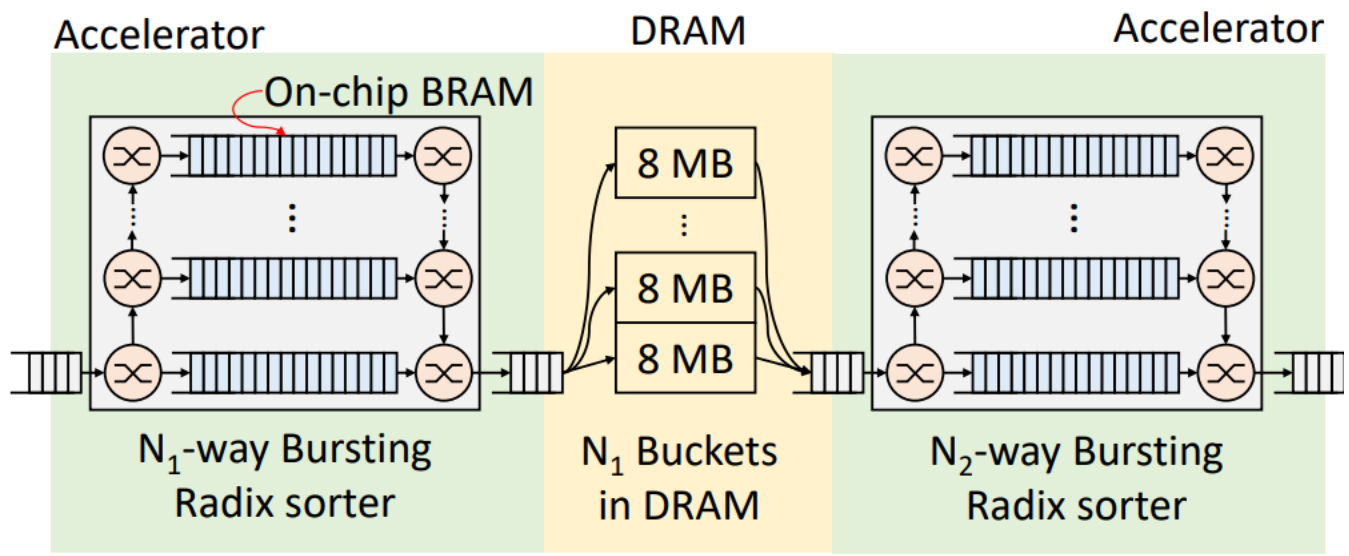
FPGA Radix Sorter

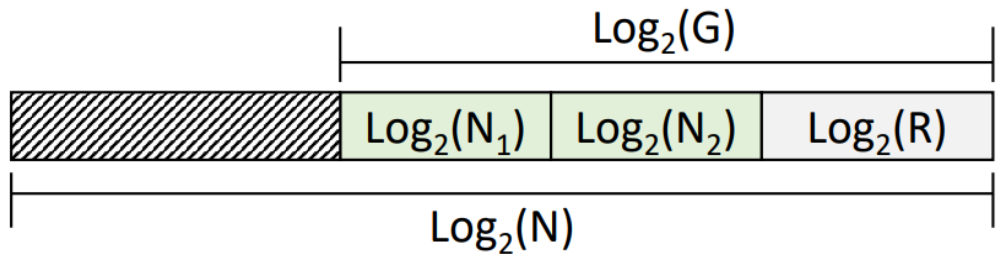


FPGA Radix Sorter

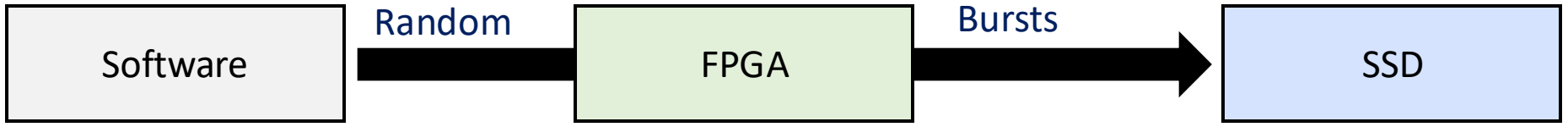


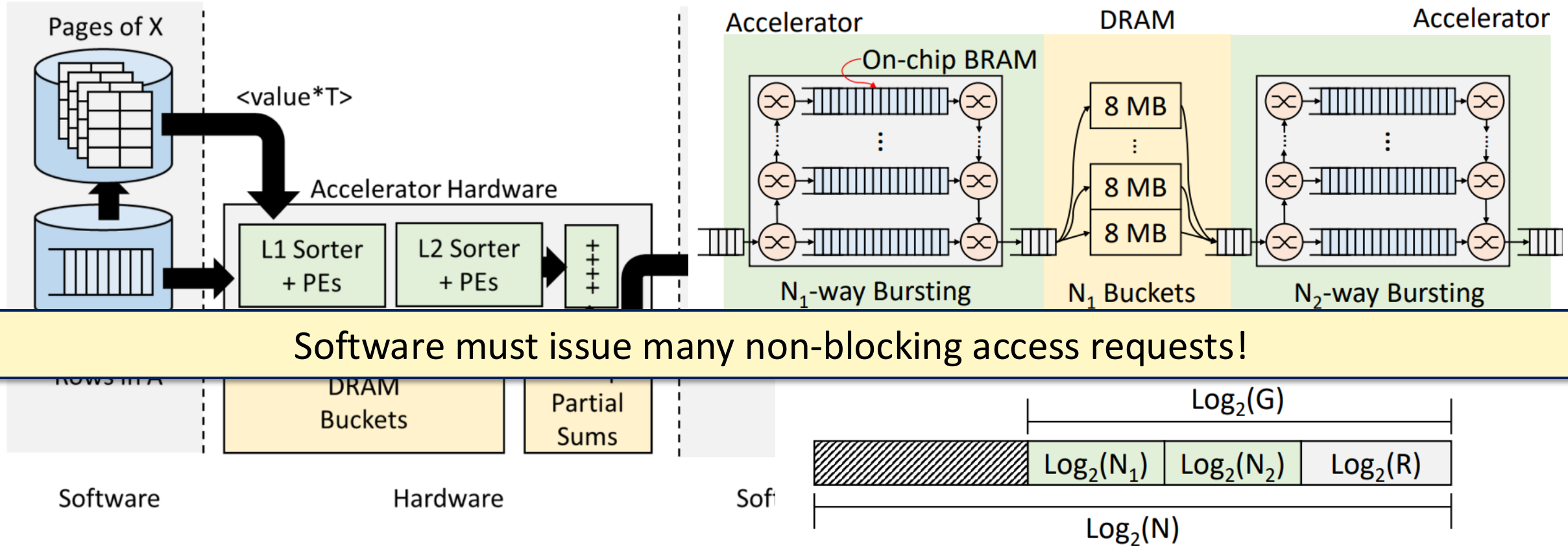






FPGA Radix Sorter





Software must issue many non-blocking access requests!

Common Abstraction for Graph Access

- Targeting near-storage acceleration (e.g., SmartSSD)
  - Key idea: Asynchronous query with callback
    - Programmer-specified callback function called when data is ready

Common Abstraction for Graph Access

- Targeting near-storage acceleration (e.g., SmartSSD)
  - Key idea: Asynchronous query with callback
    - Programmer-specified callback function called when data is ready

```
foreach vertex.getNeighbors(callback=myCallback)

function myCallback(src, dst[]) begin
        … application-specific logic …
end
```

Common Abstraction for Graph Access

- Targeting near-storage acceleration (e.g., SmartSSD)
  - Key idea: Asynchronous query with callback
    - Programmer-specified callback function called when data is ready

Out-of-order, Latency-Insensitive

```
foreach vertex.getNeighbors(callback=myCallback)

function myCallback(src, dst[]) begin
        … application-specific logic …
end
```

Common Abstraction for Graph Access

- Targeting near-storage acceleration (e.g., SmartSSD)
  - Key idea: Asynchronous query with callback
    - Programmer-specified callback function called when data is ready

Out-of-order, Latency-Insensitive

```
foreach vertex.getNeighbors(callback=myCallback)

function myCallback(src, dst[]) begin
        … application-specific logic …
end
```

- Many queries can be in flight at once (>millions)

Common Abstraction for Graph Access

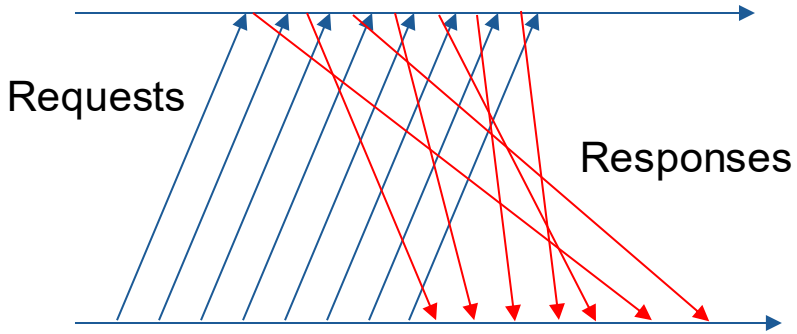
- Targeting near-storage acceleration (e.g., SmartSSD)
  - Key idea: Asynchronous query with callback
    - Programmer-specified callback function called when data is ready

Out-of-order, Latency-Insensitive

```
foreach vertex.getNeighbors(callback=myCallback)

function myCallback(src, dst[]) begin
        … application-specific logic …
end
```

- Many queries can be in flight at once (>millions)



Common Abstraction for Graph Access

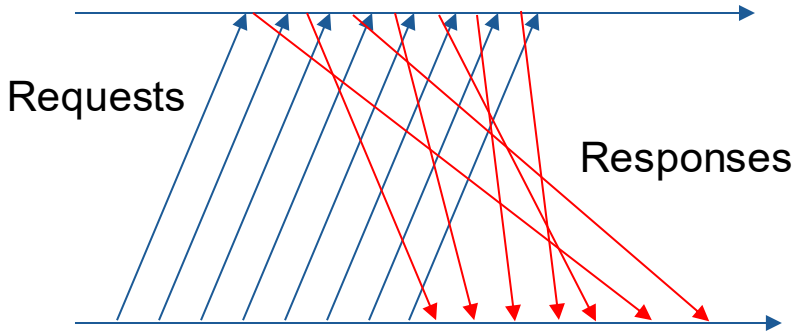
- Targeting near-storage acceleration (e.g., SmartSSD)
  - Key idea: Asynchronous query with callback
    - Programmer-specified callback function called when data is ready

Out-of-order, Latency-Insensitive

```
foreach vertex.getNeighbors(callback=myCallback)

function myCallback(src, dst[]) begin
        … application-specific logic …
end
```

- Many queries can be in flight at once (>millions)
- Storage access latency can be hidden



Common Abstraction for Graph Access

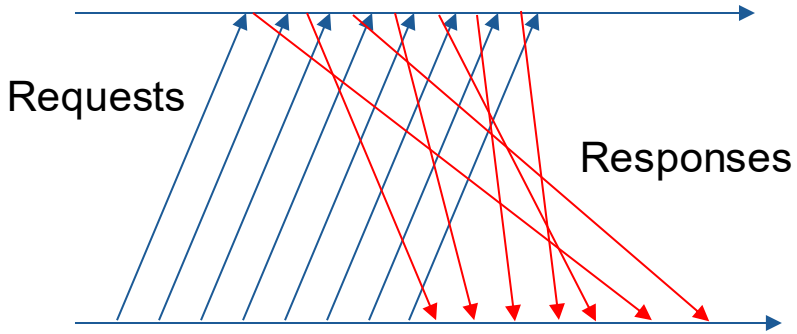
- Targeting near-storage acceleration (e.g., SmartSSD)
  - Key idea: Asynchronous query with callback
    - Programmer-specified callback function called when data is ready

Out-of-order, Latency-Insensitive

```
foreach vertex.getNeighbors(callback=myCallback)

function myCallback(src, dst[]) begin
        … application-specific logic …
end
```

- Many queries can be in flight at once (>millions)
- Storage access latency can be hidden
- Transparently group accesses to the same page



Common Abstraction for Graph Access

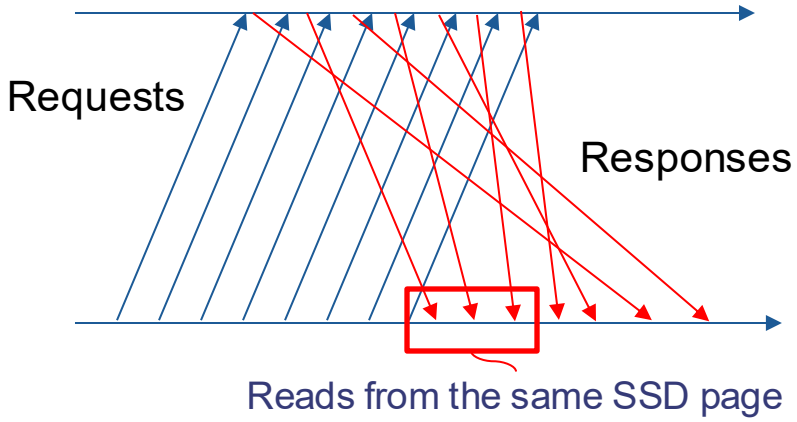
- Targeting near-storage acceleration (e.g., SmartSSD)
  - Key idea: Asynchronous query with callback
    - Programmer-specified callback function called when data is ready

Out-of-order, Latency-Insensitive

```
foreach vertex.getNeighbors(callback=myCallback)

function myCallback(src, dst[]) begin
            … application-specific logic …
end
```

- Many queries can be in flight at once (>millions)
- Storage access latency can be hidden
- Transparently group accesses to the same page



Common Abstraction for Graph Access

- Targeting near-storage acceleration (e.g., SmartSSD)
  - Key idea: Asynchronous query with callback
    - Programmer-specified callback function called when data is ready

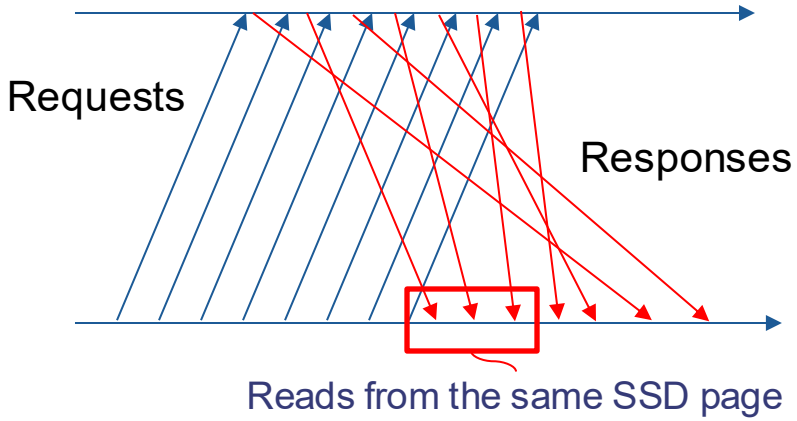
Out-of-order, Latency-Insensitive

```
foreach vertex.getNeighbors(callback=myCallback)

function myCallback(src, dst[]) begin
        … application-specific logic …
end
```

- Many queries can be in flight at once (>millions)
- Storage access latency can be hidden
- Transparently group accesses to the same page

Minimize I/O amplification!



Common Abstraction for Graph Access

- Targeting near-storage acceleration (e.g., SmartSSD)
  - Key idea: Asynchronous query with callback
    - Programmer-specified callback function called when data is ready

Out-of-order, Latency-Insensitive

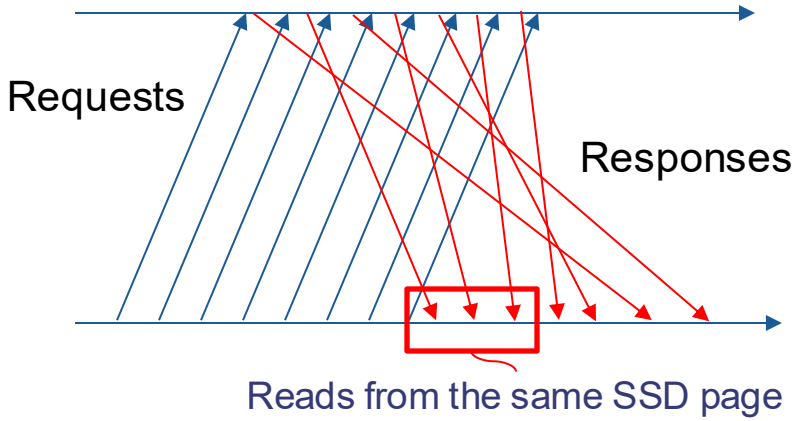
```
foreach vertex.getNeighbors(callback=myCallback)

function myCallback(src, dst[]) begin
        … application-specific logic …
end
```

- Many queries can be in flight at once (>millions)
- Storage access latency can be hidden
- Transparently group accesses to the same page

Minimize I/O amplification!

- Other transparent optimizations can be hidden



A Library of Optimizations to Hide

- Access re-organization (Done)
  - Burst-sorting accelerator to group accesses to the same page

A Library of Optimizations to Hide

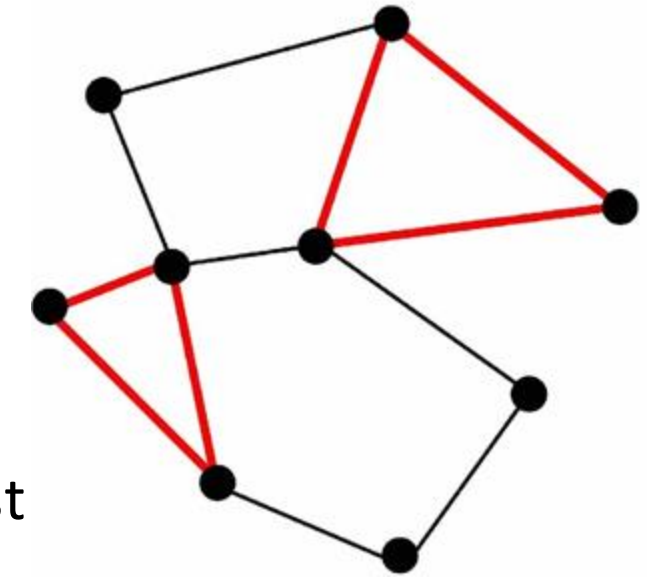
- ❑ Access re-organization (Done)
  - Burst-sorting accelerator to group accesses to the same page
- ❑ Probabilistic filtering (Done)
  - Use bloom filter to avoid storage reads which will return negative results
  - e.g., Nonexistent graph edges, Nodes with no outgoing edge

A Library of Optimizations to Hide

- ❑ Access re-organization (Done)
  - Burst-sorting accelerator to group accesses to the same page
- ❑ Probabilistic filtering (Done)
  - Use bloom filter to avoid storage reads which will return negative results
  - e.g., Nonexistent graph edges, Nodes with no outgoing edge
- ❑ Compression (In Progress)
  - Application-specific compression, e.g., LZ4, ZFP, XOR, VarInt
  - Reference-based compression

Preliminary Evaluation: Triangle Counting

- Counts the number of triangles in a graph
- Important application
  - One of four benchmarks in MIT/Lincoln Labs GraphChallenge[9]
- Involves two neighborhood queries
  - For each V,
    enumerate permutations of neighbor(V) → (A,B)
    check whether B ∈ neighbor(A)
  - Bloom filter trained on graph edges
    Avoid neighborhood queries for A if edge(A,B) doesn't exist



[9] https://graphchallenge.mit.edu/challenges

Experimental Setup

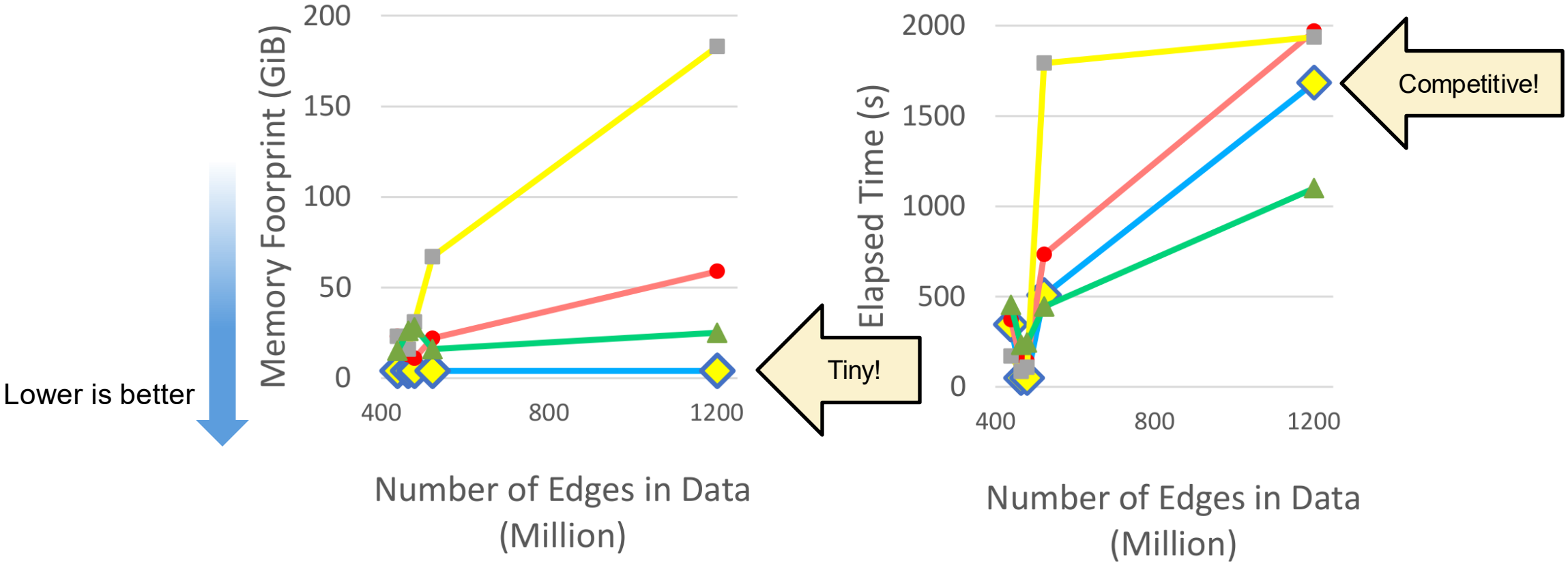
- State-of-the-art baselines:
  - GraphBLAS
  - HPEC graph challenge champions: Karypis (CPU), TRUST (GPU)
  - A lot more which failed from memory limitations (e.g., Neo4J)
- Dell T640 server w/ 24-Core Xeon Gold and 200 GB DRAM, V100 GPU
  - + **One** Samsung SmartSSD for SSD+FPGA
  - Our approach only used 4 threads + 4 GB memory

Experimental Setup

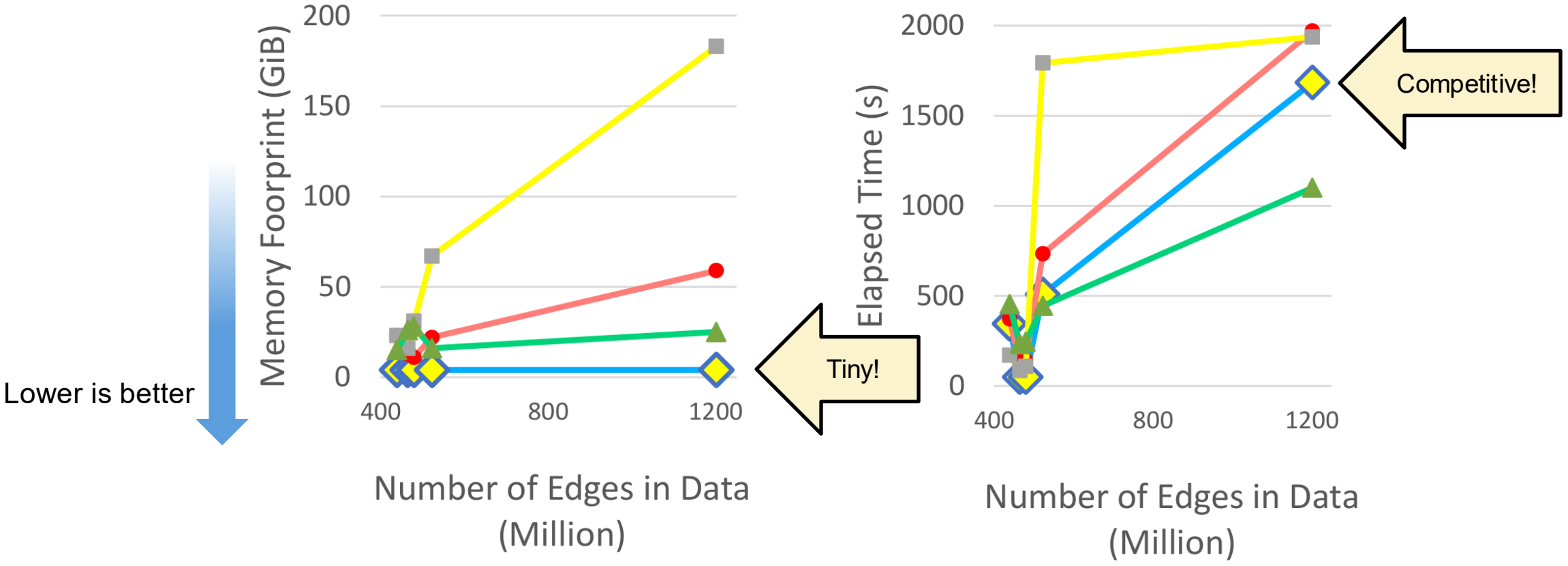
- State-of-the-art baselines:
  - GraphBLAS
  - HPEC graph challenge champions: Karypis (CPU), TRUST (GPU)
  - A lot more which failed from memory limitations (e.g., Neo4J)
- Dell T640 server w/ 24-Core Xeon Gold and 200 GB DRAM, V100 GPU
  - + **One** Samsung SmartSSD for SSD+FPGA
  - Our approach only used 4 threads + 4 GB memory

Graph	Edge # (Billion)
DARPA	0.44
V1r	0.46
MAWI	0.48
Graph500	1.05
Twitter	1.46

Results: Performance Improvements

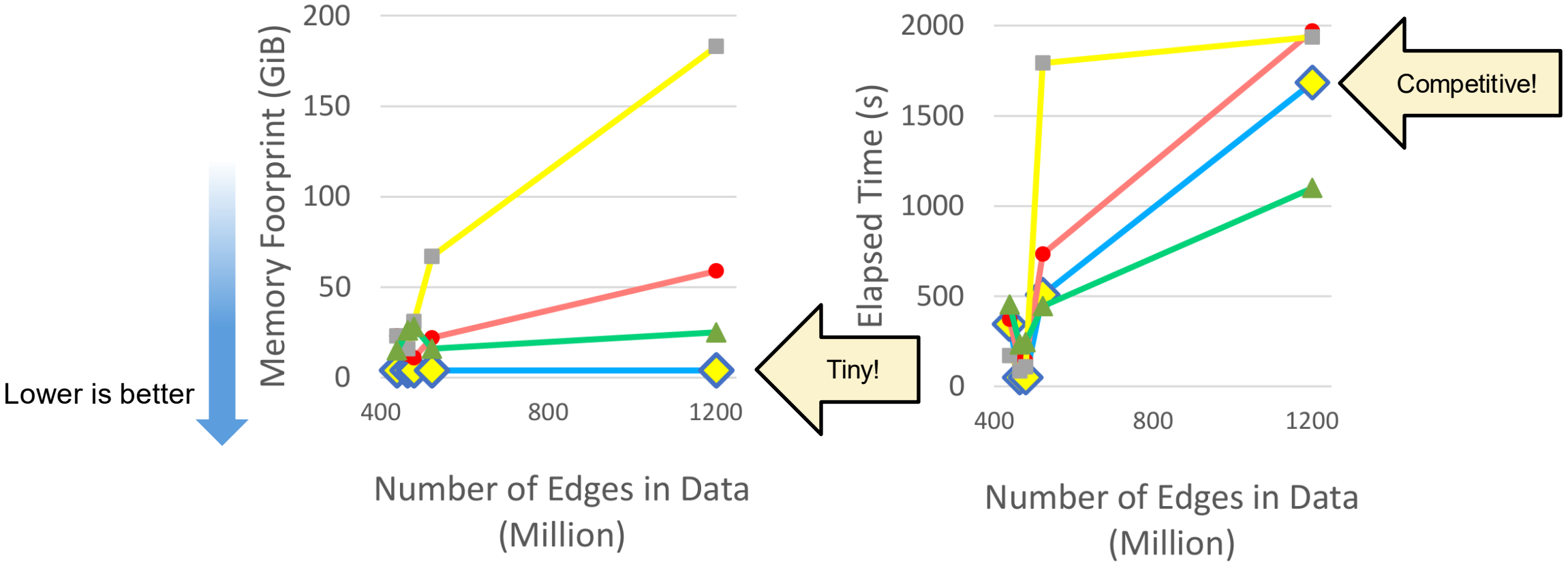


Results: Performance Improvements



¼ Cost, Comparable performance

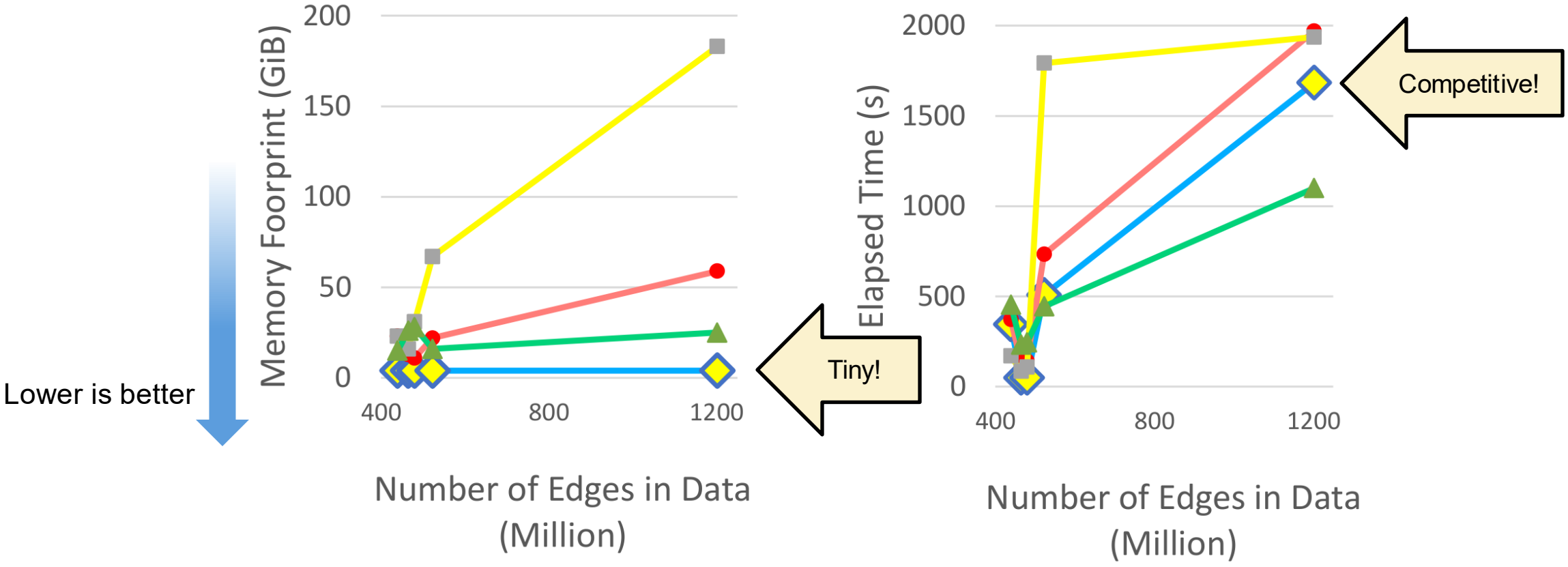
Results: Performance Improvements



¼ Cost, Comparable performance

60% - 95% Bloom filtering rate with 0.5 GB Bloom filter

Results: Performance Improvements



Emphasis: Considers whole-system affects (e.g., storage latency, I/O amplification)

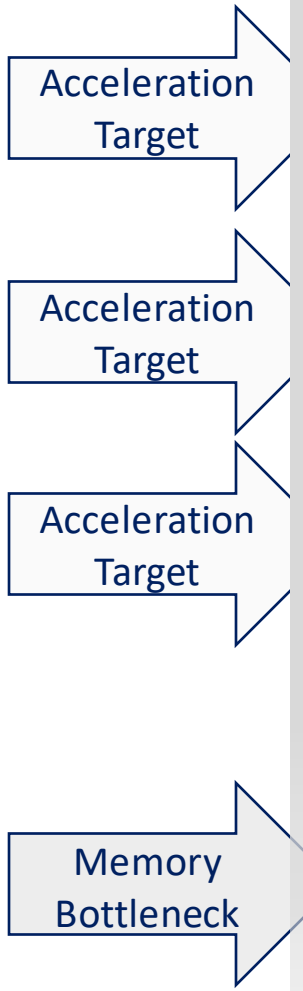
¼ Cost, Comparable performance

60% - 95% Bloom filtering rate with 0.5 GB Bloom filter

What to accelerate, for De Novo Assembly?

Goal 2: “Minimap 2”
For N-to-N genome alignment

Goal 1: Graph Construction and Traversal



Ctg_align (minimap2)	12	390
Ctg_cns	40	58

What to accelerate, for De Novo Assembly?

Goal 2: “Minimap 2”
For N-to-N genome alignment

Goal 1: Graph Construction and Traversal

Acceleration Target

Acceleration Target

Acceleration Target

Memory Bottleneck

Ctg_align (minimap2)	12	390
Ctg_cns	40	58

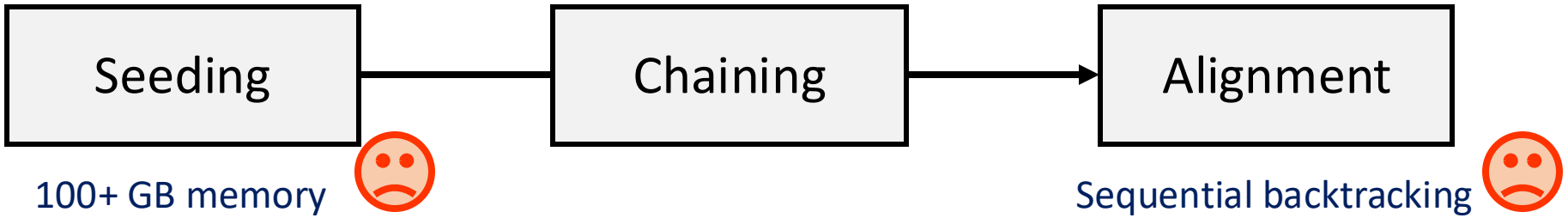
Ongoing Work 2: Alignment Accelerator

- Many De Novo tools internally use “Minimap2”
  - Input: Reference, reads
  - Output: mapping between them
- De Novo does not use a reference, reads act also as reference
  - Massively increased work: 10x or more!

Ongoing Work: Alignment Accelerator

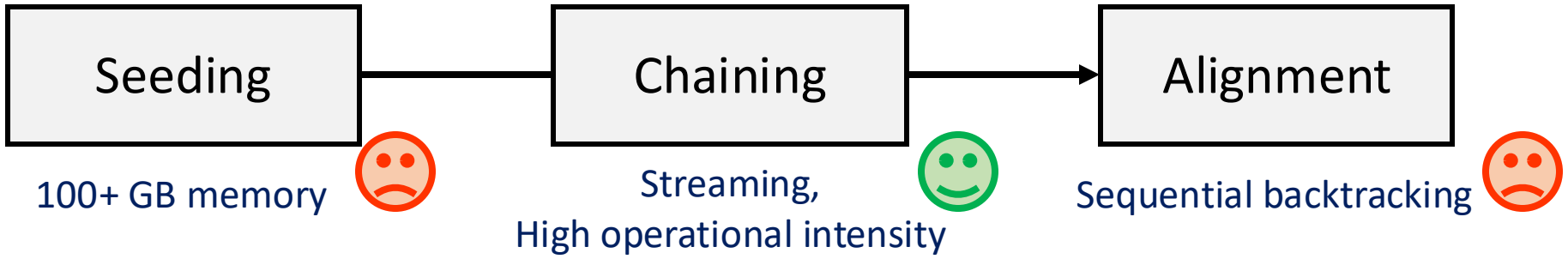
Ongoing Work: Alignment Accelerator

Ongoing Work: Alignment Accelerator



Ongoing Work: Alignment Accelerator

Ongoing Work: Alignment Accelerator



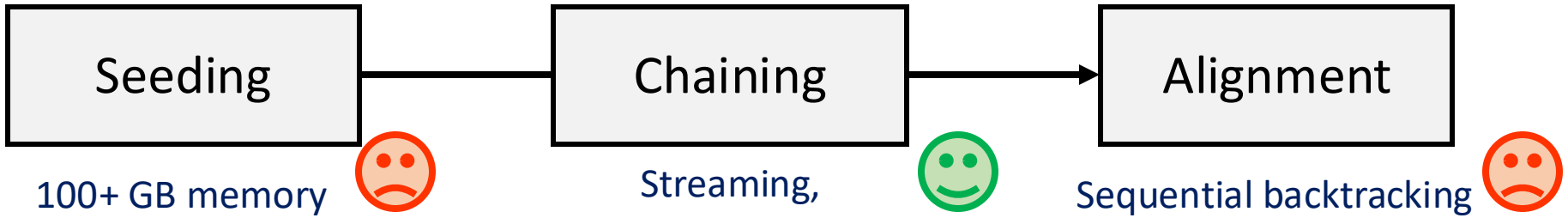
Irregular computation patterns

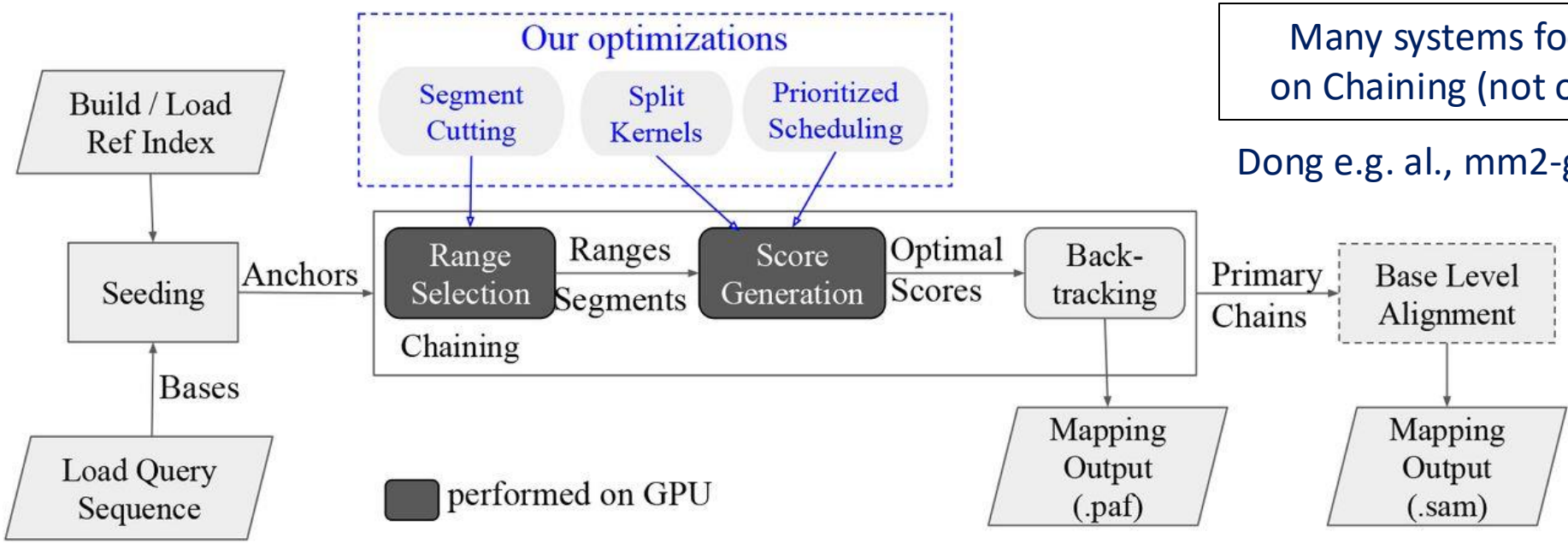
Large memory requirements

Not readily parallelizable

Ongoing Work: Alignment Accelerator

Ongoing Work: Alignment Accelerator

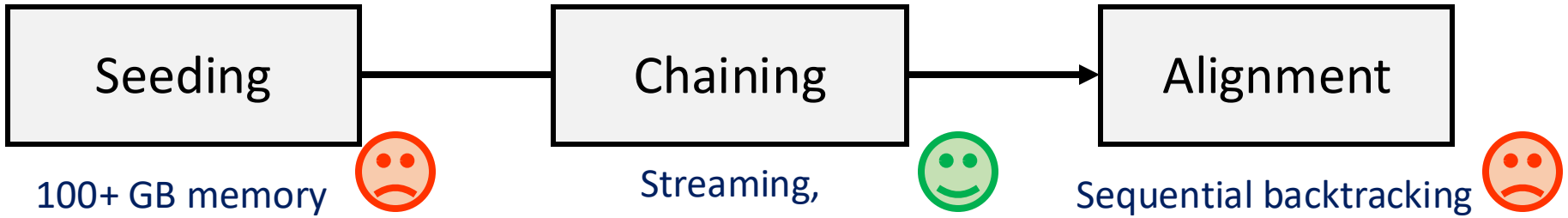


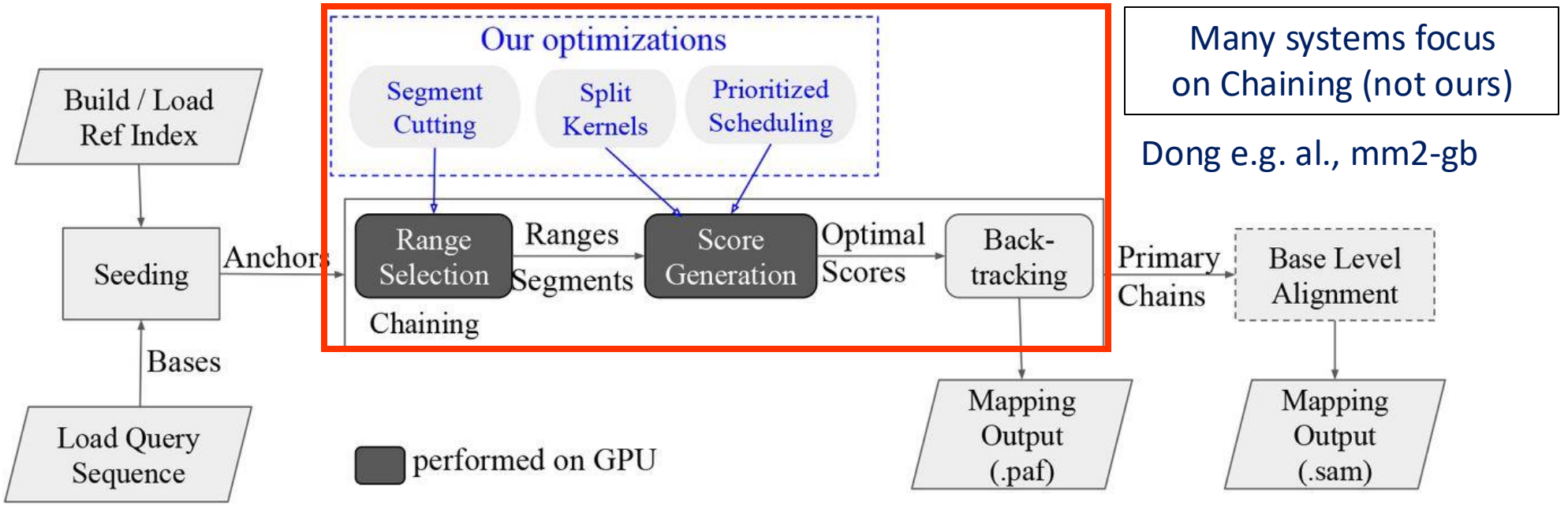


Many systems focus on Chaining (not ours)

Dong e.g. al., mm2-gb

Ongoing Work: Alignment Accelerator

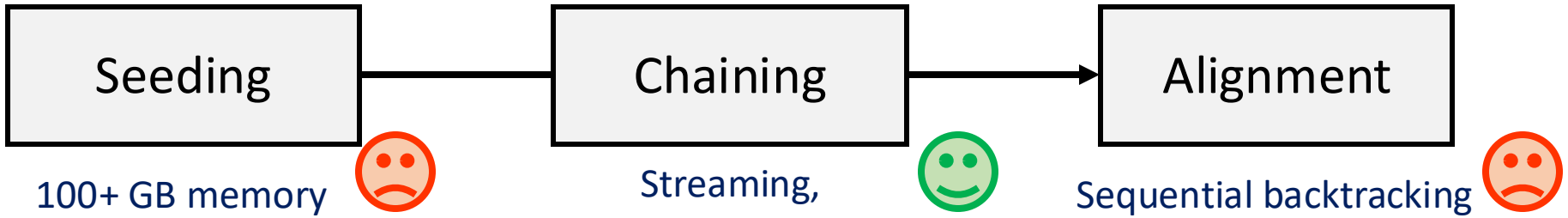


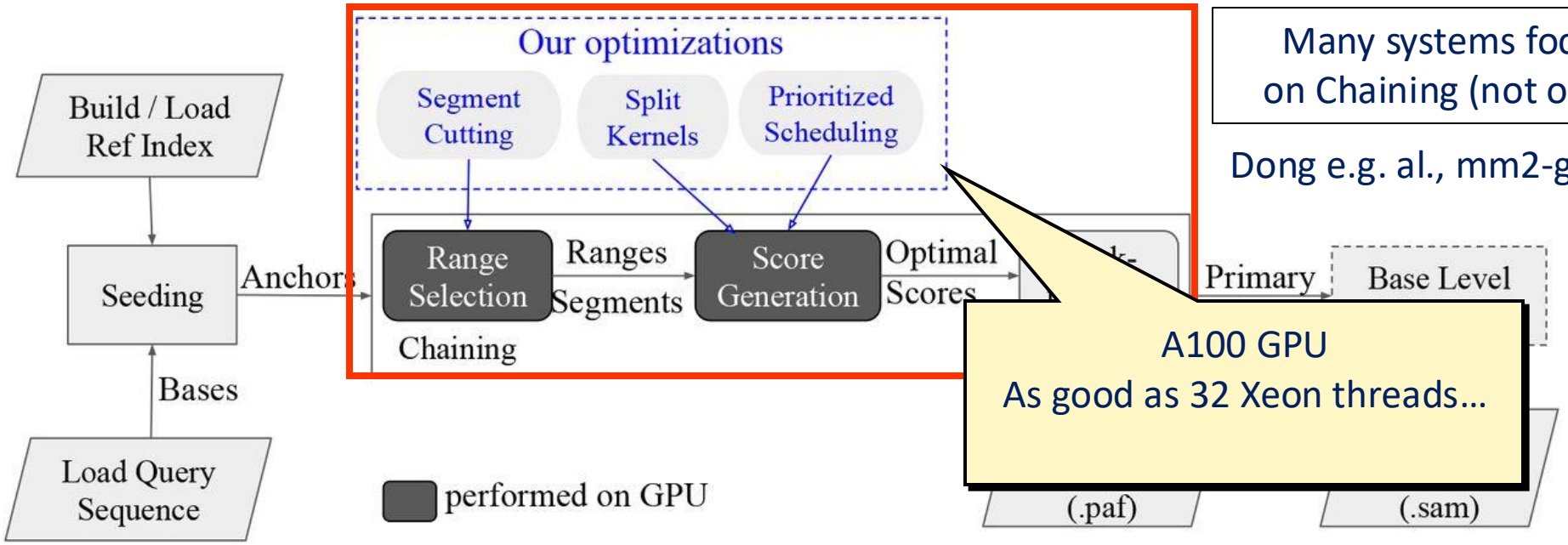


Many systems focus on Chaining (not ours)

Dong e.g. al., mm2-gb

Ongoing Work: Alignment Accelerator



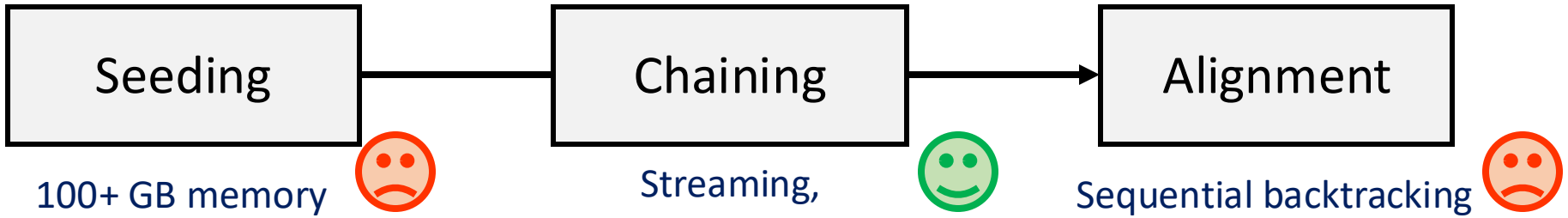


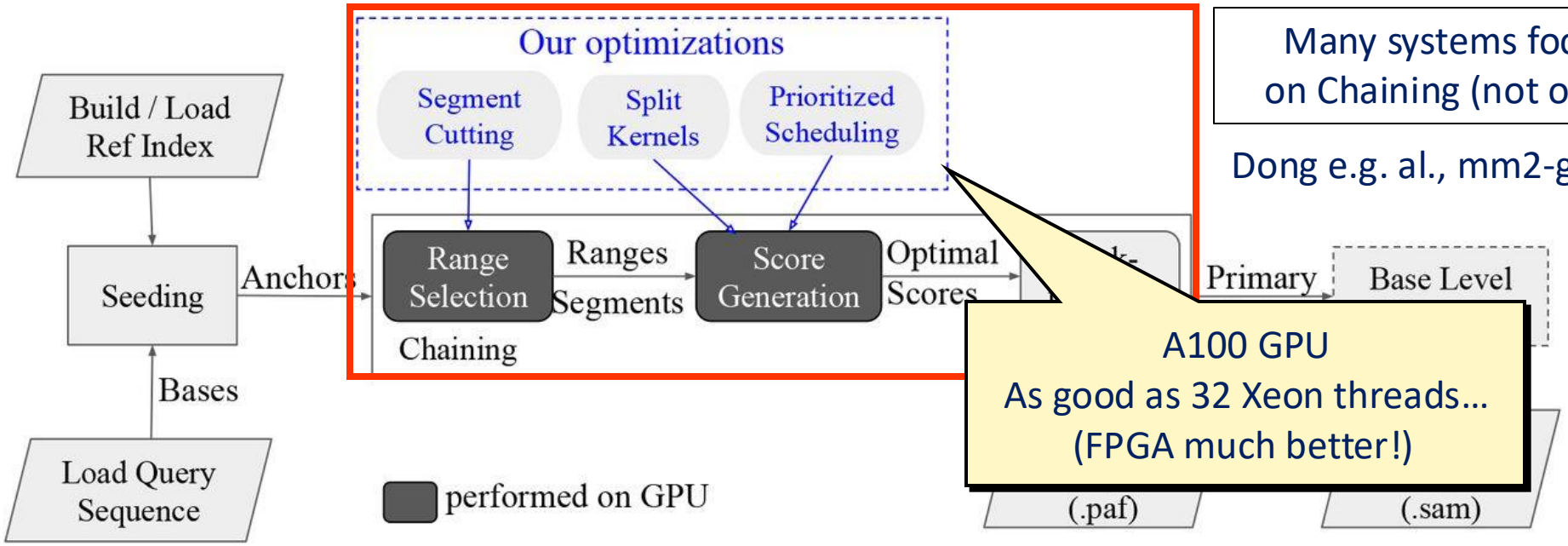
Many systems focus on Chaining (not ours)

Dong e.g. al., mm2-gb

A100 GPU
As good as 32 Xeon threads…

Ongoing Work: Alignment Accelerator



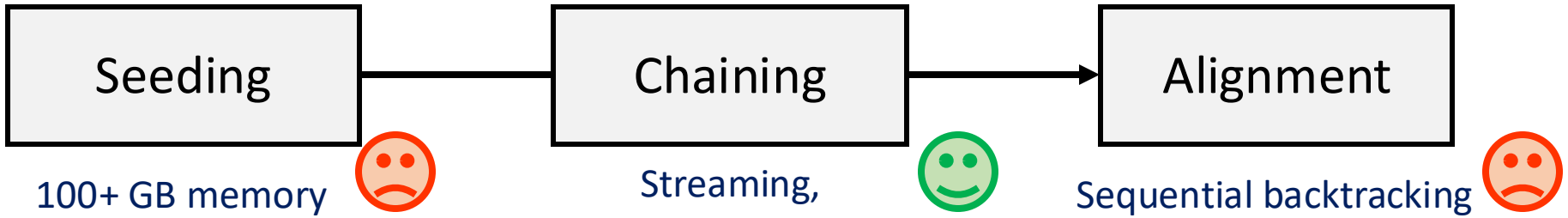


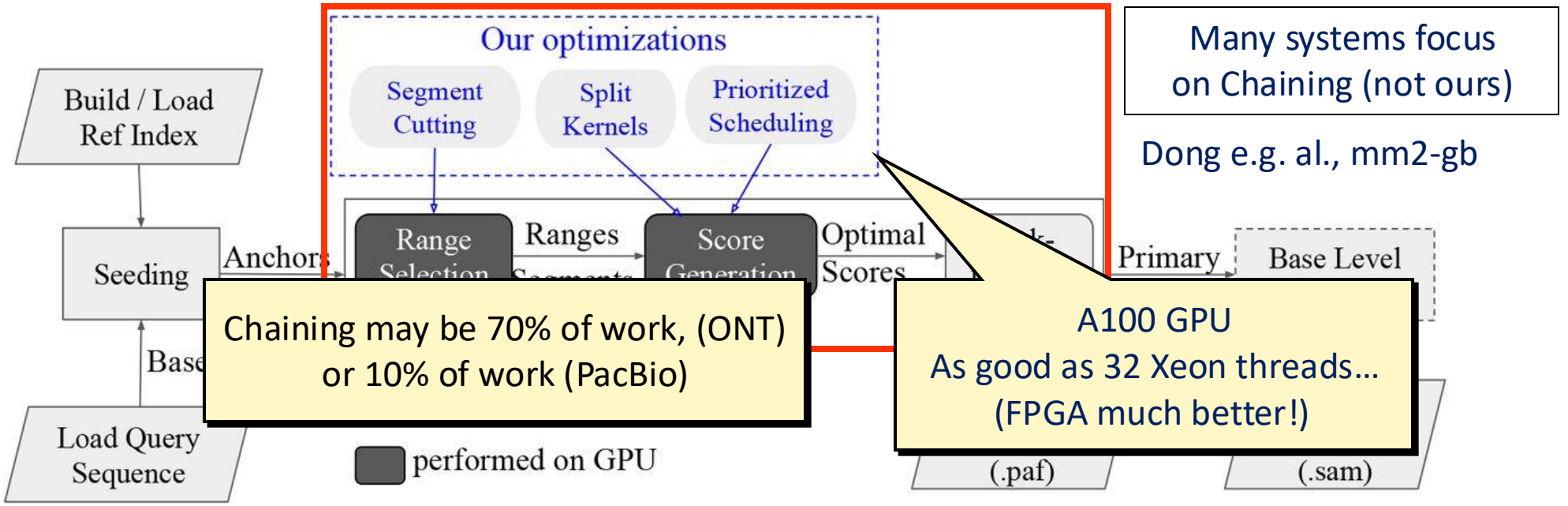
Many systems focus on Chaining (not ours)

Dong e.g. al., mm2-gb

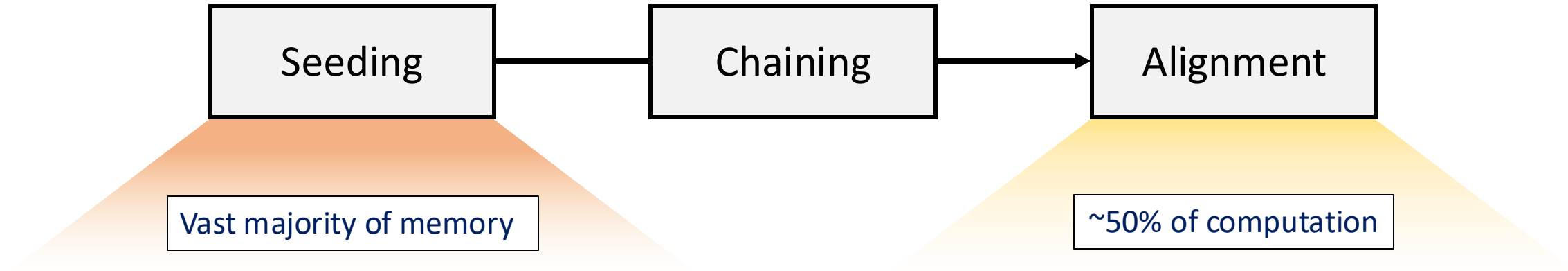
A100 GPU
As good as 32 Xeon threads…
(FPGA much better!)

Ongoing Work: Alignment Accelerator

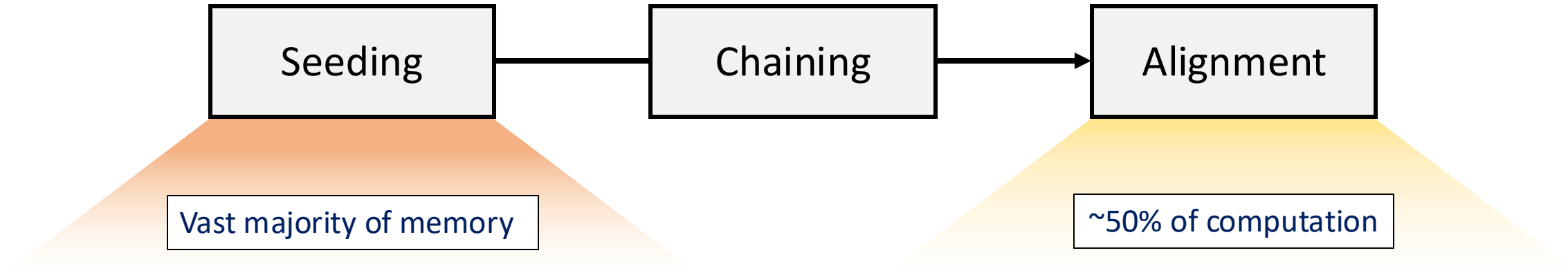




Ongoing Work: Alignment Accelerator

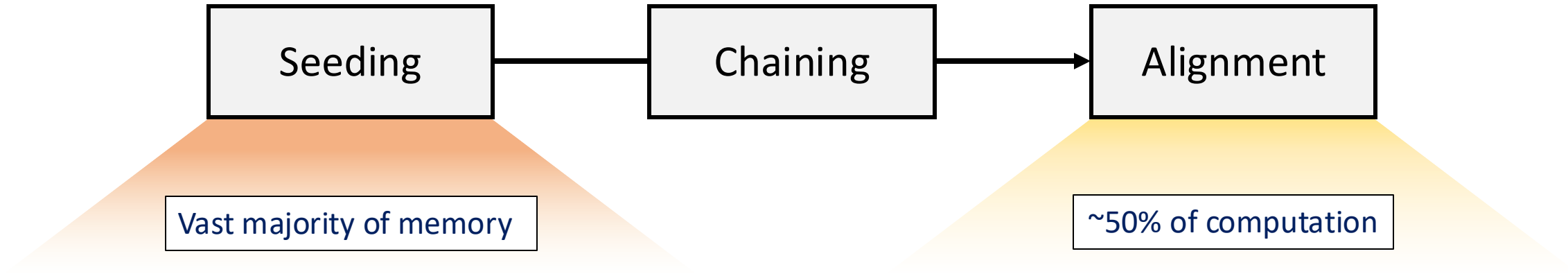


Ongoing Work: Alignment Accelerator

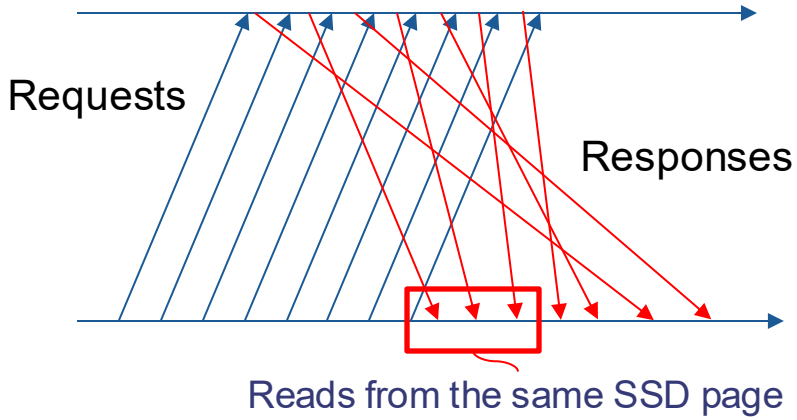


- Random-access during hash construction
- Random-access during hash lookup

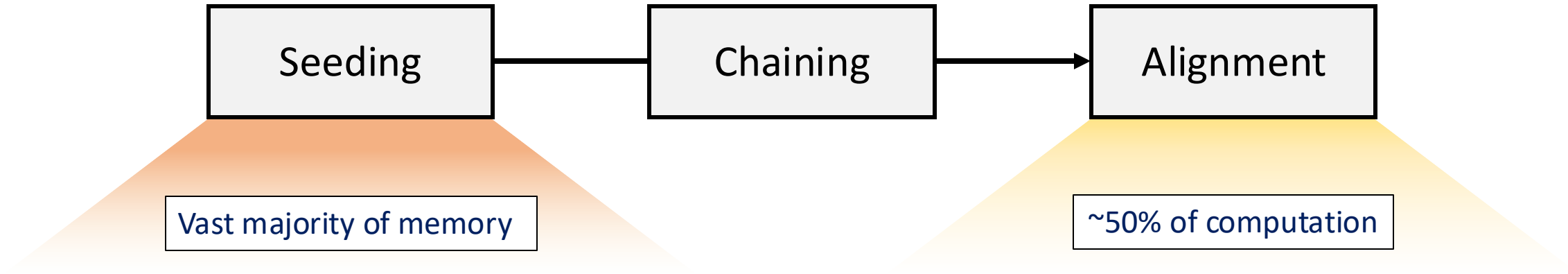
Ongoing Work: Alignment Accelerator



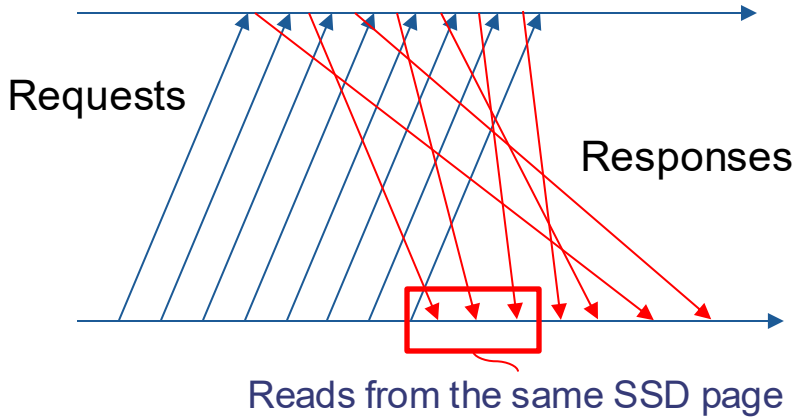
- Random-access during hash construction
- Random-access during hash lookup



Ongoing Work: Alignment Accelerator

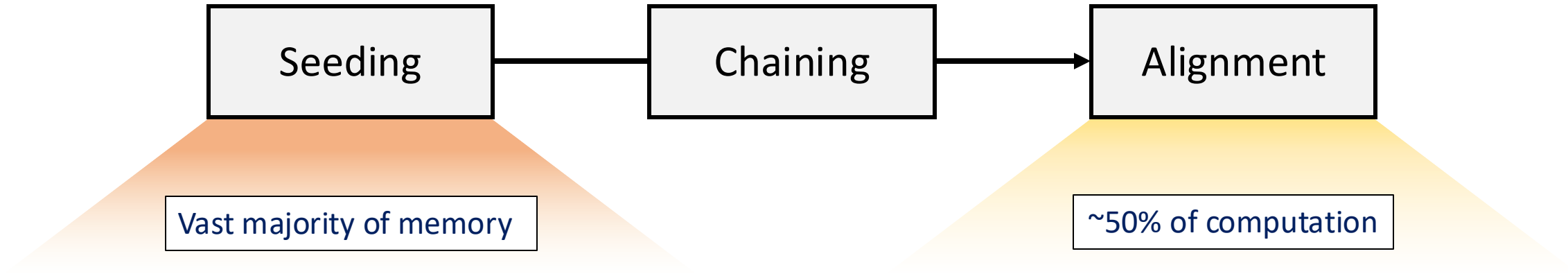


- Random-access during hash construction
- Random-access during hash lookup

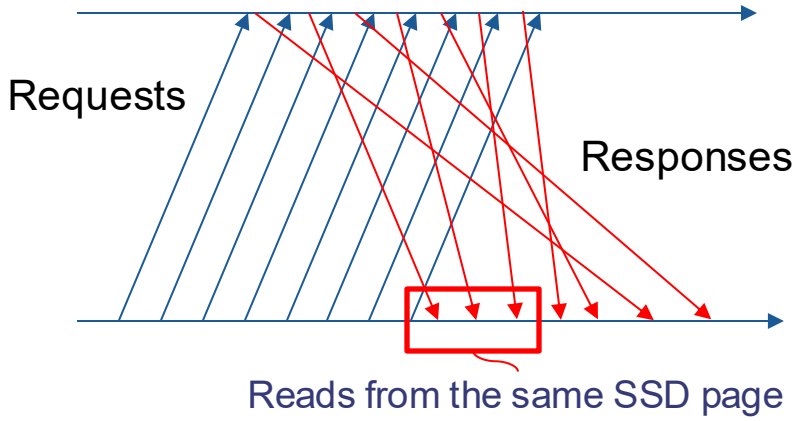


e.g., Kang et. al., “BunchBloomer”, FPL 2022

Ongoing Work: Alignment Accelerator



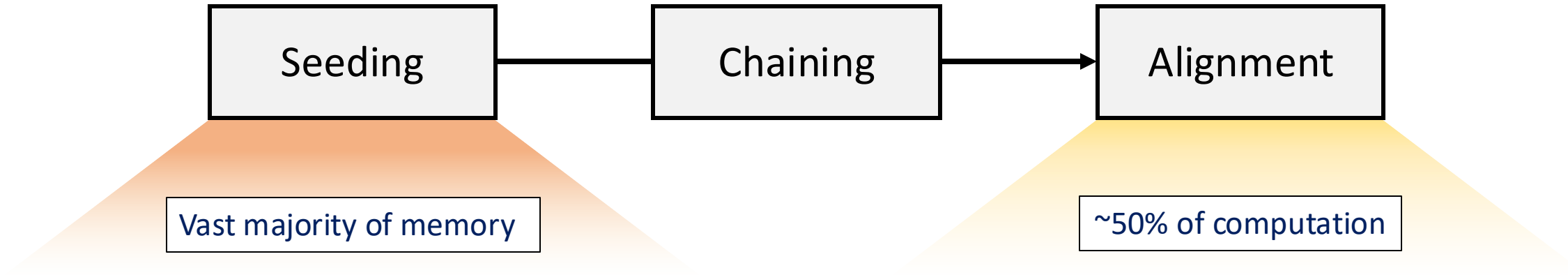
- Random-access during hash construction
- Random-access during hash lookup



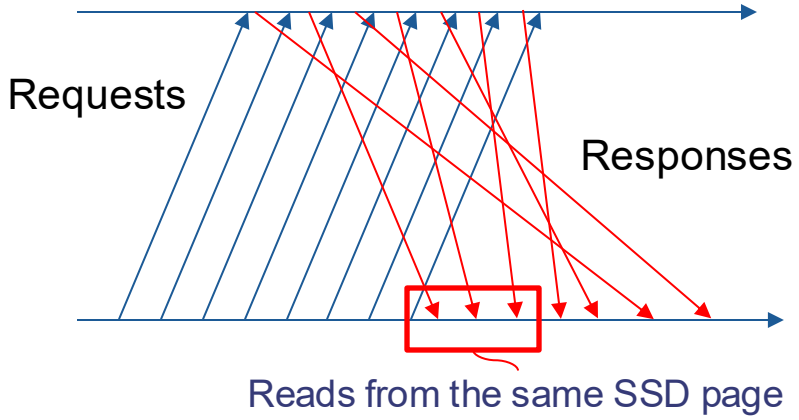
- Backtracking needs fast clock (CPU?)
- Score matrix is too large… (PCIe bottleneck!)

e.g., Kang et. al., “BunchBloomer”, FPL 2022

Ongoing Work: Alignment Accelerator

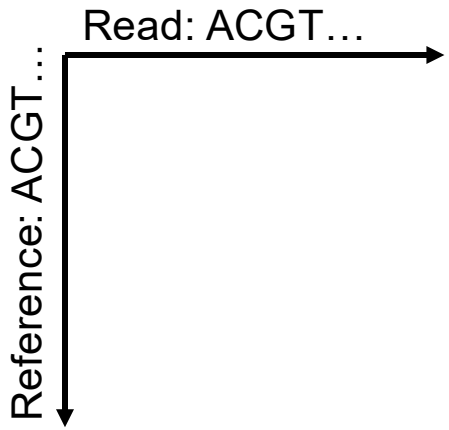


- Random-access during hash construction
- Random-access during hash lookup



e.g., Kang et. al., “BunchBloomer”, FPL 2022

- Backtracking needs fast clock (CPU?)
- Score matrix is too large… (PCIe bottleneck!)

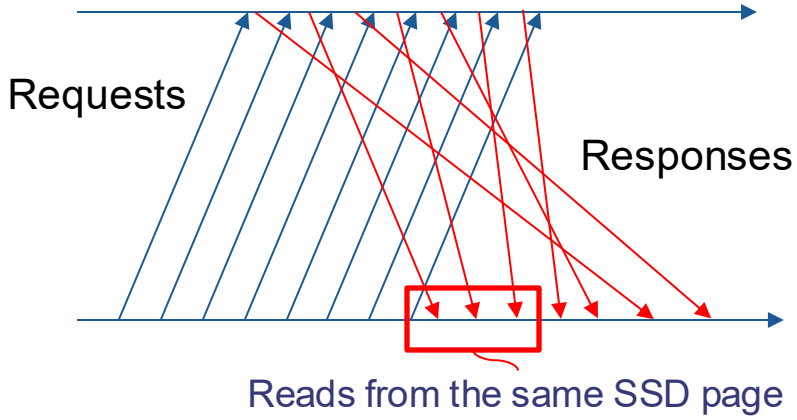


Ongoing Work: Alignment Accelerator

Vast majority of memory

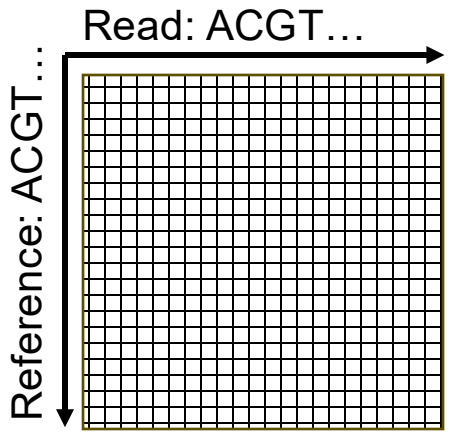
~50% of computation

- Random-access during hash construction
- Random-access during hash lookup



e.g., Kang et. al., “BunchBloomer”, FPL 2022

- Backtracking needs fast clock (CPU?)
- Score matrix is too large… (PCIe bottleneck!)



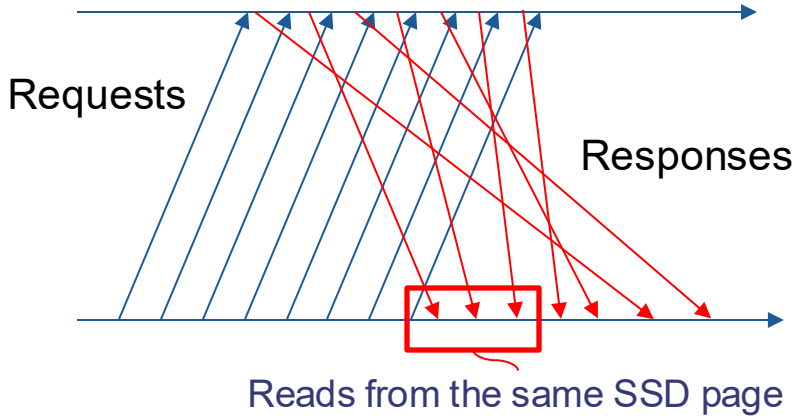
- Score matrix computation: Parallel

Ongoing Work: Alignment Accelerator

Vast majority of memory

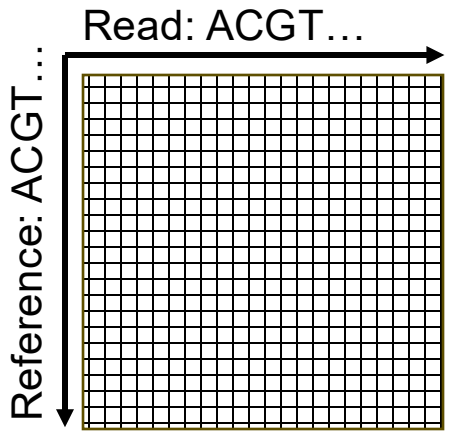
~50% of computation

- Random-access during hash construction
- Random-access during hash lookup



e.g., Kang et. al., “BunchBloomer”, FPL 2022

- Backtracking needs fast clock (CPU?)
- Score matrix is too large… (PCIe bottleneck!)



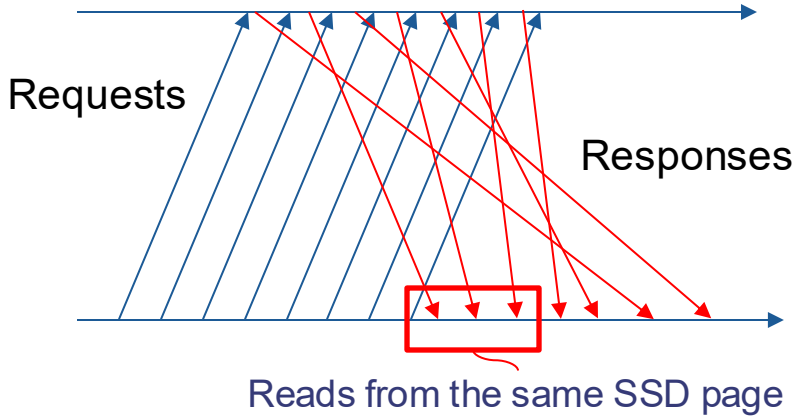
- Score matrix computation: Parallel
- Score matrix: Large

Ongoing Work: Alignment Accelerator

Vast majority of memory

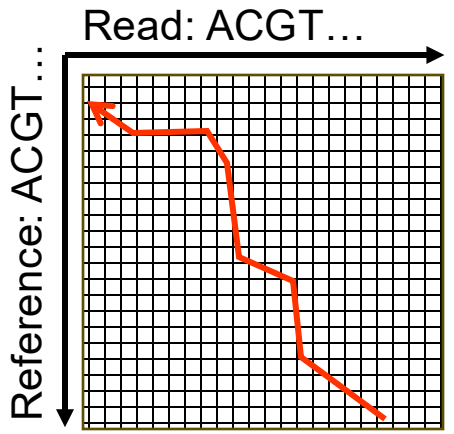
~50% of computation

- Random-access during hash construction
- Random-access during hash lookup



e.g., Kang et. al., “BunchBloomer”, FPL 2022

- Backtracking needs fast clock (CPU?)
- Score matrix is too large… (PCIe bottleneck!)



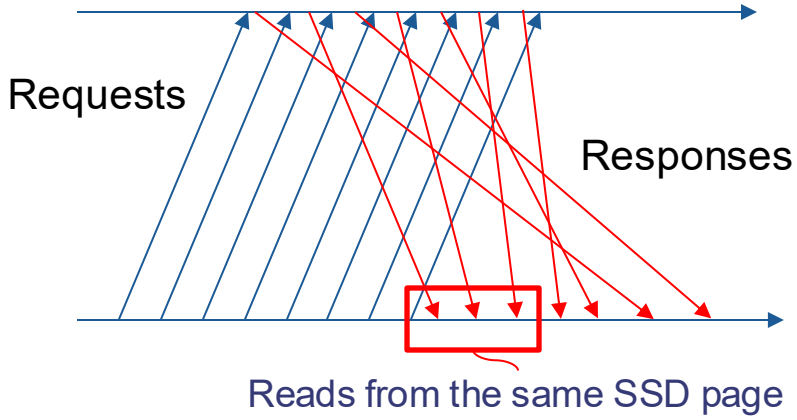
- Score matrix computation: Parallel
- Score matrix: Large
- Backtracking: Sequential

Ongoing Work: Alignment Accelerator

Vast majority of memory

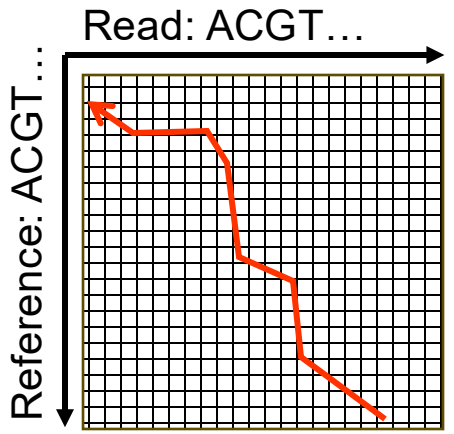
~50% of computation

- Random-access during hash construction
- Random-access during hash lookup



e.g., Kang et. al., “BunchBloomer”, FPL 2022

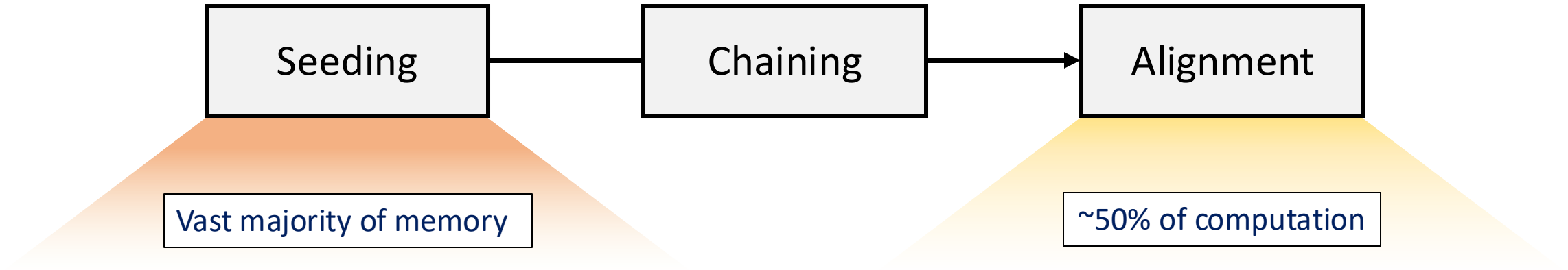
- Backtracking needs fast clock (CPU?)
- Score matrix is too large… (PCIe bottleneck!)



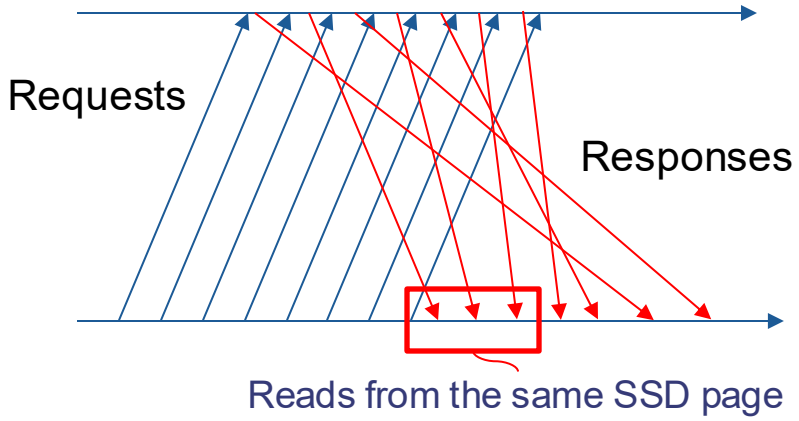
- Score matrix computation: Parallel
- Score matrix: Large
- Backtracking: Sequential

- Solution 1: Compress the matrix

Ongoing Work: Alignment Accelerator

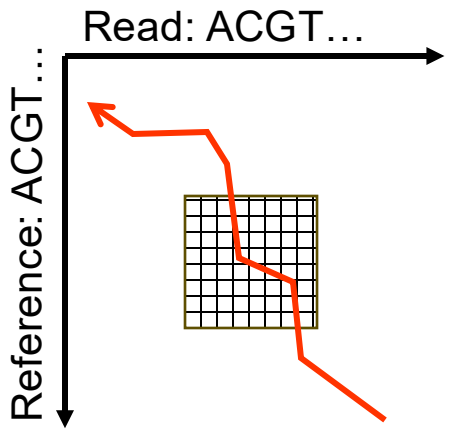


- Random-access during hash construction
- Random-access during hash lookup



e.g., Kang et. al., “BunchBloomer”, FPL 2022

- Backtracking needs fast clock (CPU?)
- Score matrix is too large… (PCIe bottleneck!)



- Score matrix computation: Parallel
- Score matrix: Large
- Backtracking: Sequential

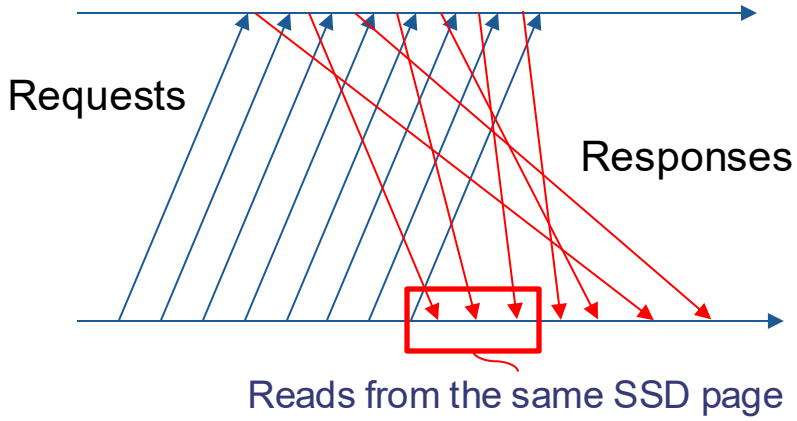
- Solution 1: Compress the matrix

Ongoing Work: Alignment Accelerator

Vast majority of memory

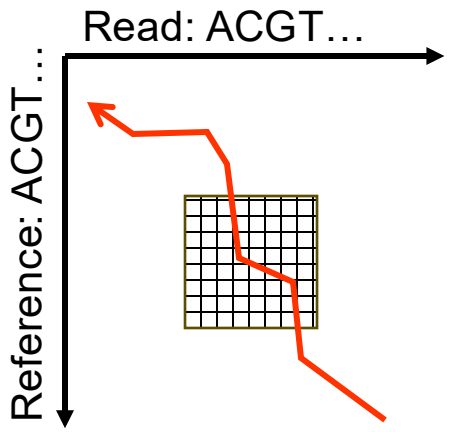
~50% of computation

- Random-access during hash construction
- Random-access during hash lookup



e.g., Kang et. al., “BunchBloomer”, FPL 2022

- Backtracking needs fast clock (CPU?)
- Score matrix is too large… (PCIe bottleneck!)



- Score matrix computation: Parallel
- Score matrix: Large
- Backtracking: Sequential

- Solution 1: Compress the matrix
- Solution 2: Parallel backtracking

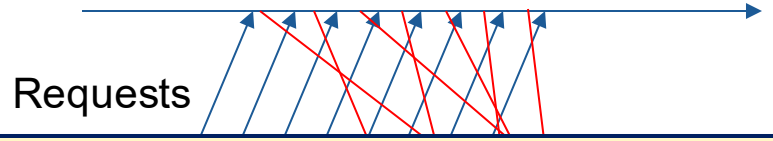
Ongoing Work: Alignment Accelerator

Vast majority of memory

~50% of computation

- Random-access during hash construction
- Random-access during hash lookup

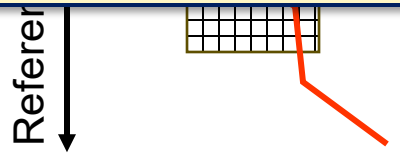
- Backtracking needs fast clock (CPU?)
- Score matrix is too large… (PCIe bottleneck!)



Goal: Drop-in replacement of Minimap2
1/10 memory, 10x performance

Reads from the same SSD page

e.g., Kang et. al., “BunchBloomer”, FPL 2022



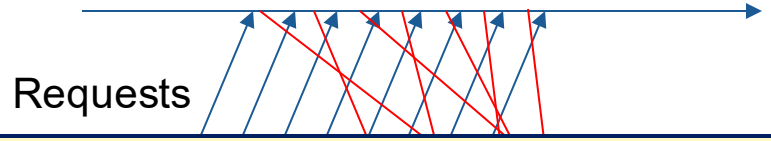
- Solution 1: Compress the matrix
- Solution 2: Parallel backtracking

Ongoing Work: Alignment Accelerator

Vast majority of memory

~50% of computation

- Random-access during hash construction
- Random-access during hash lookup

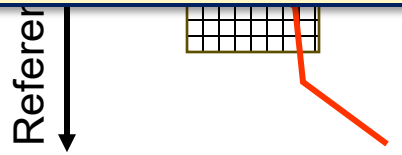


- Backtracking needs fast clock (CPU?)
- Score matrix is too large… (PCIe bottleneck!)

Goal: Drop-in replacement of Minimap2
1/10 memory, 10x performance … Stay tuned!

Reads from the same SSD page

e.g., Kang et. al., “BunchBloomer”, FPL 2022

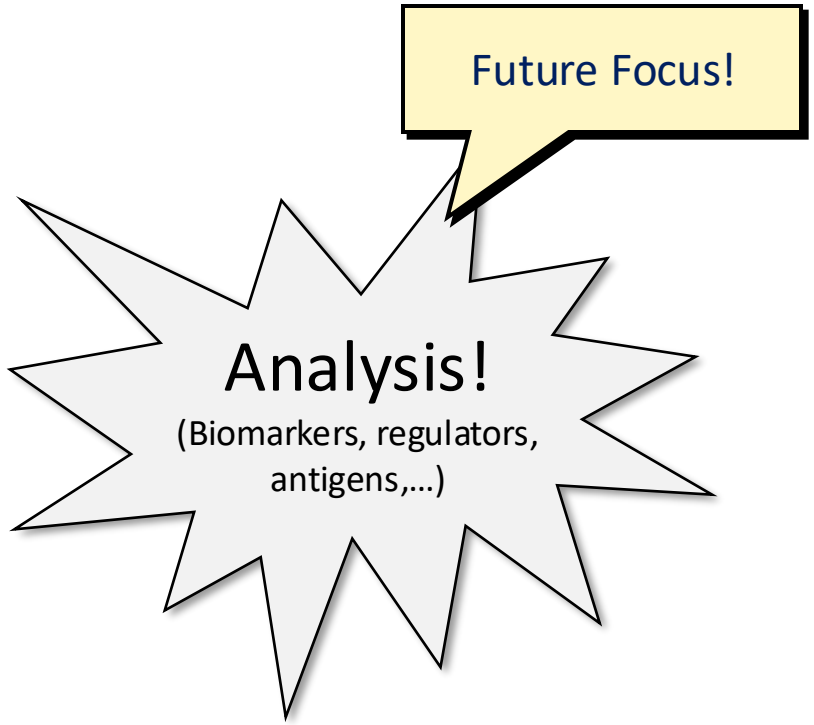


- Solution 1: Compress the matrix
- Solution 2: Parallel backtracking

Long-Term Goal:
Precision ("Personalized") Medicine

Cancer Patient

Sequenced Genome



Our Efforts So Far…

ISCA 2018	NVM + FPGA	vertex-centric graph analytics
Frontiers 2021	NVM	Genomic graphs (SMuFin)
FPL 2022	DRAM + FPGA	Genomic graphs (De Bruijn)
PACT 2023	NVM + FPGA	Graph Neural Networks
DAC 2024	NVM + FPGA	Software-Driven graph analytics

…

Our Efforts So Far…

ISCA 2018	NVM + FPGA	vertex-centric graph analytics
Frontiers 2021	NVM	Genomic graphs (SMuFin)
FPL 2022	DRAM + FPGA	Genomic graphs (De Bruijn)
PACT 2023	NVM + FPGA	Graph Neural Networks
DAC 2024	NVM + FPGA	Software-Driven graph analytics

Genome Compression Graph Compression Parallel Backtracking

…

ARDA is Interested in a LOT of things!

ARDA is Interested in a LOT of things!

- ❑ Graph Neural Networks
- ❑ Edge processing – Earthquakes and Wildfires
- ❑ Edge processing – Smart Agriculture
- ❑ Processing-In-Memory
- ❑ Accelerating Program Analysis
- ❑ Scientific Computing – Symbolic Regression

- ❑ Oh my!

Students Involved

- PhD Se-Min Lim @ UCI
  - Scalable Graph Neural Networks with near-storage acceleration

- PhD Seongyoung Kang @ UCI
  - Scalable Subgraph Isomorphism with near-storage acceleration
  - Triangle counting demo being developed
    - Plan to present to Samsung collaborators (Xuebin Yao, Reza Soltaniyeh)

- PhD Esmerald Aliaj @ UCI
  - Compiler support for hardware kernel generation