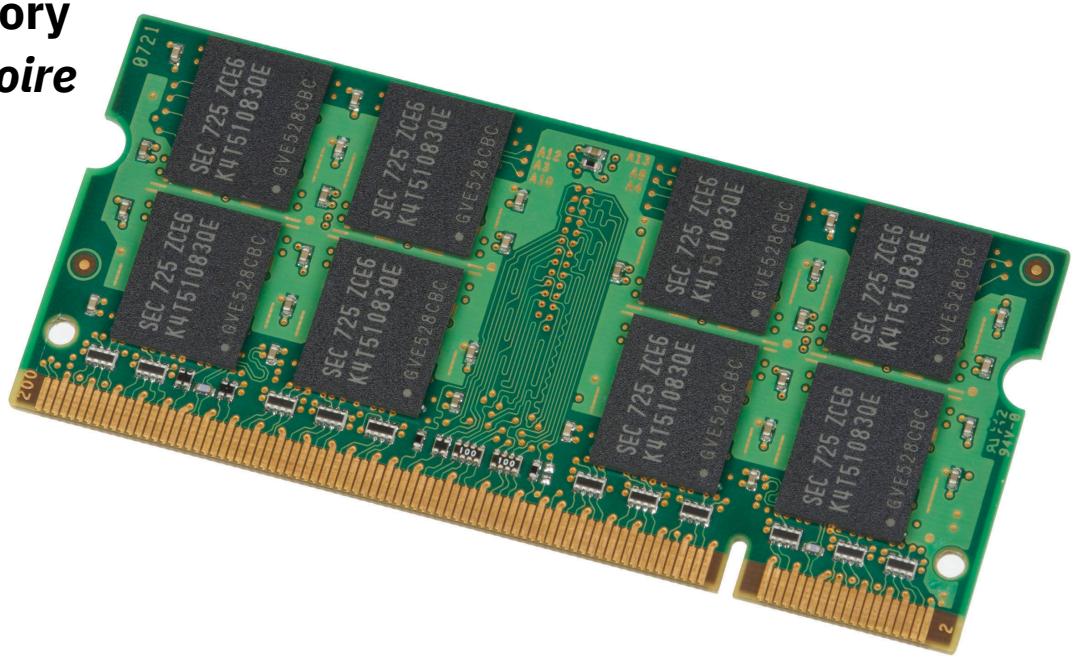
Automatic Derivation of Memory Allocations for Polyhedral Programs

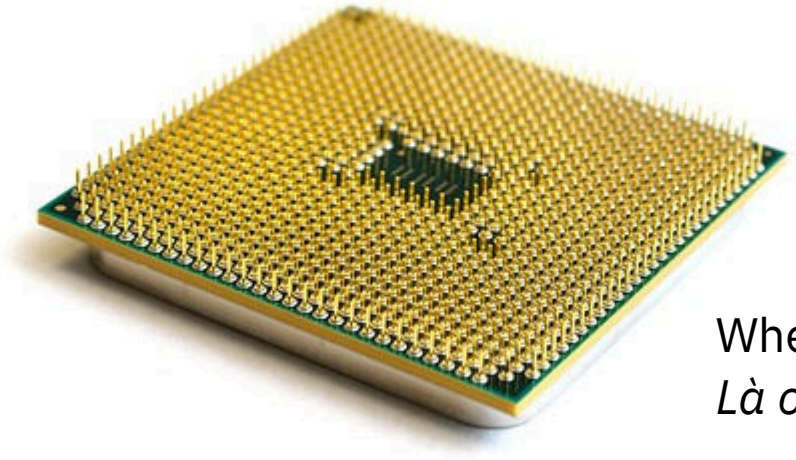
Corentin Ferry – February 19, 2024

Ph.D. Defense

Memory
Mémoire

Where the data is
Là où sont les données



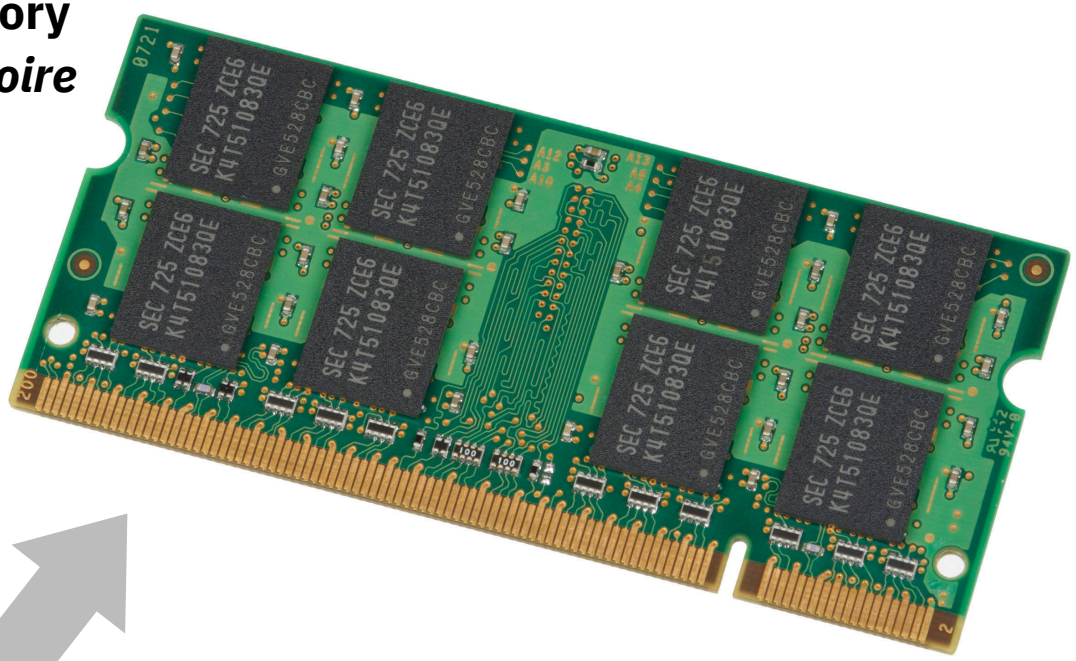


Processor
Processeur

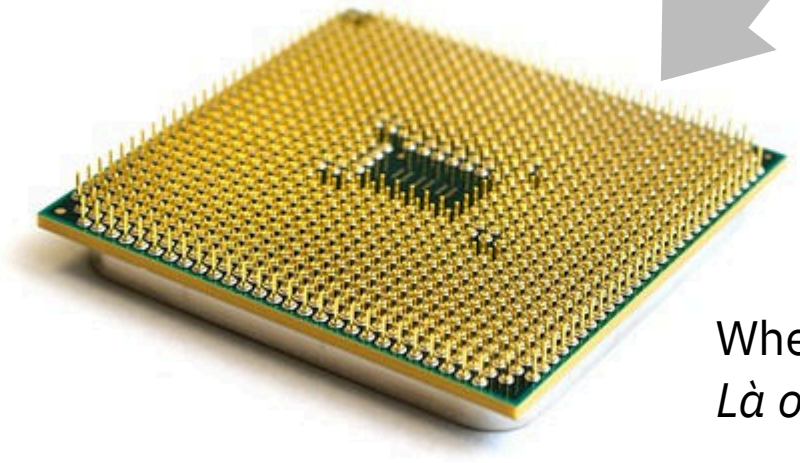
Where the programs run
Là où s'exécutent les programmes

Memory
Mémoire

Where the data is
Là où sont les données



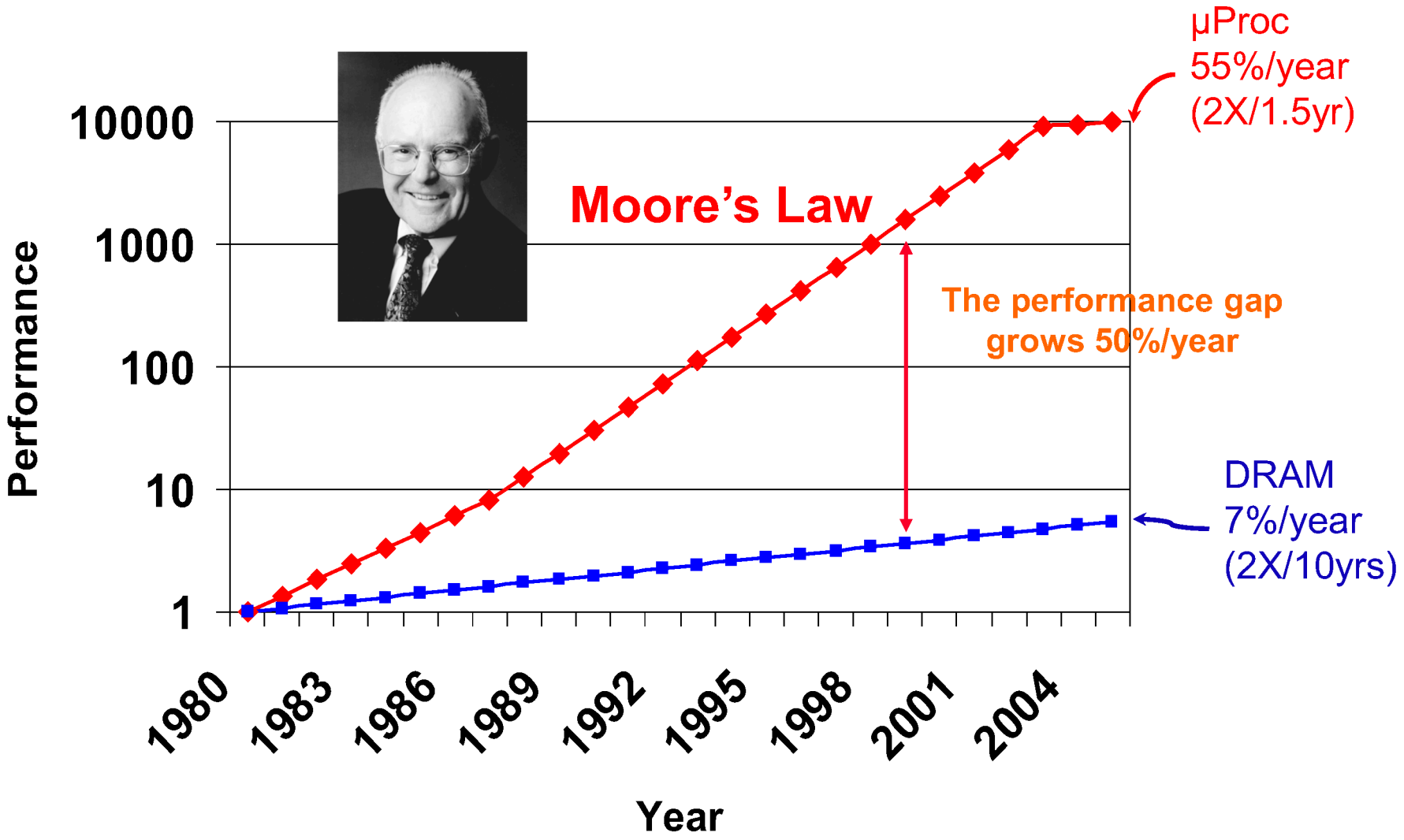
Data exchange is… way too slow!
Les données circulent trop lentement !



Processor
Processeur

Where the programs run
Là où s'exécutent les programmes

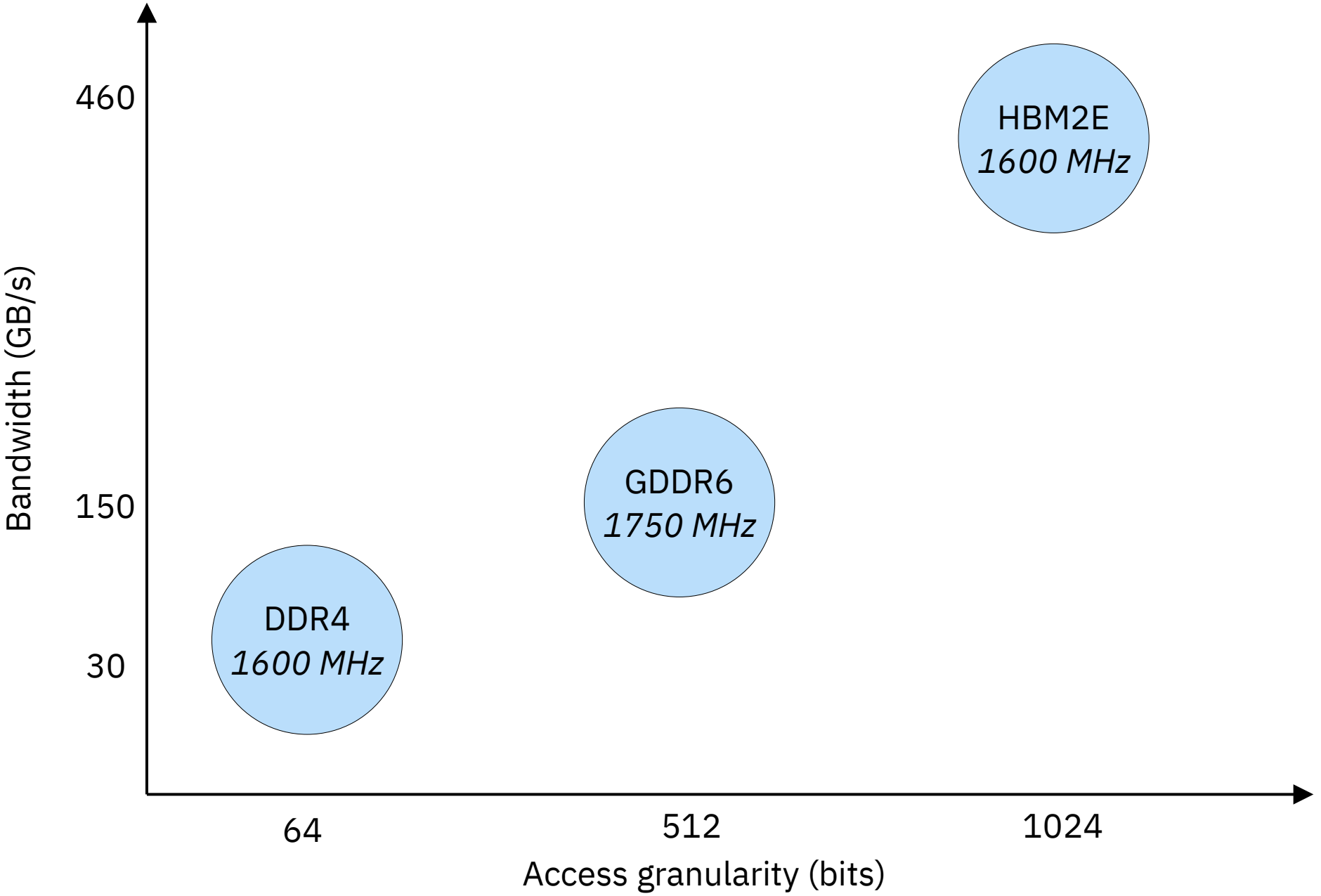
CPU vs Memory Performance



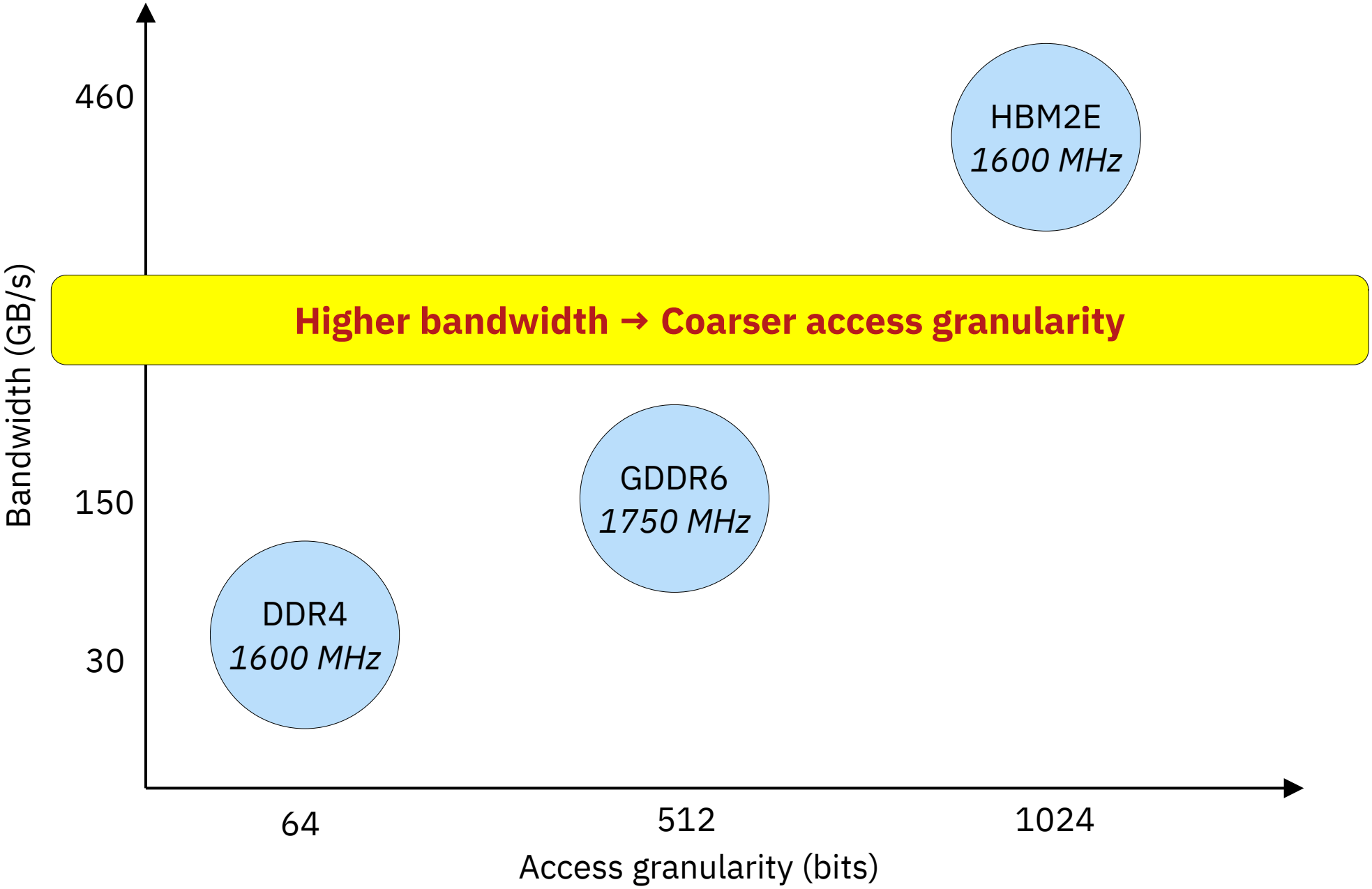
Prof. Sean Lee's Slide

https://www.sci.utah.edu/~mb/Teaching/Week3/mem-hierarchy.pdf

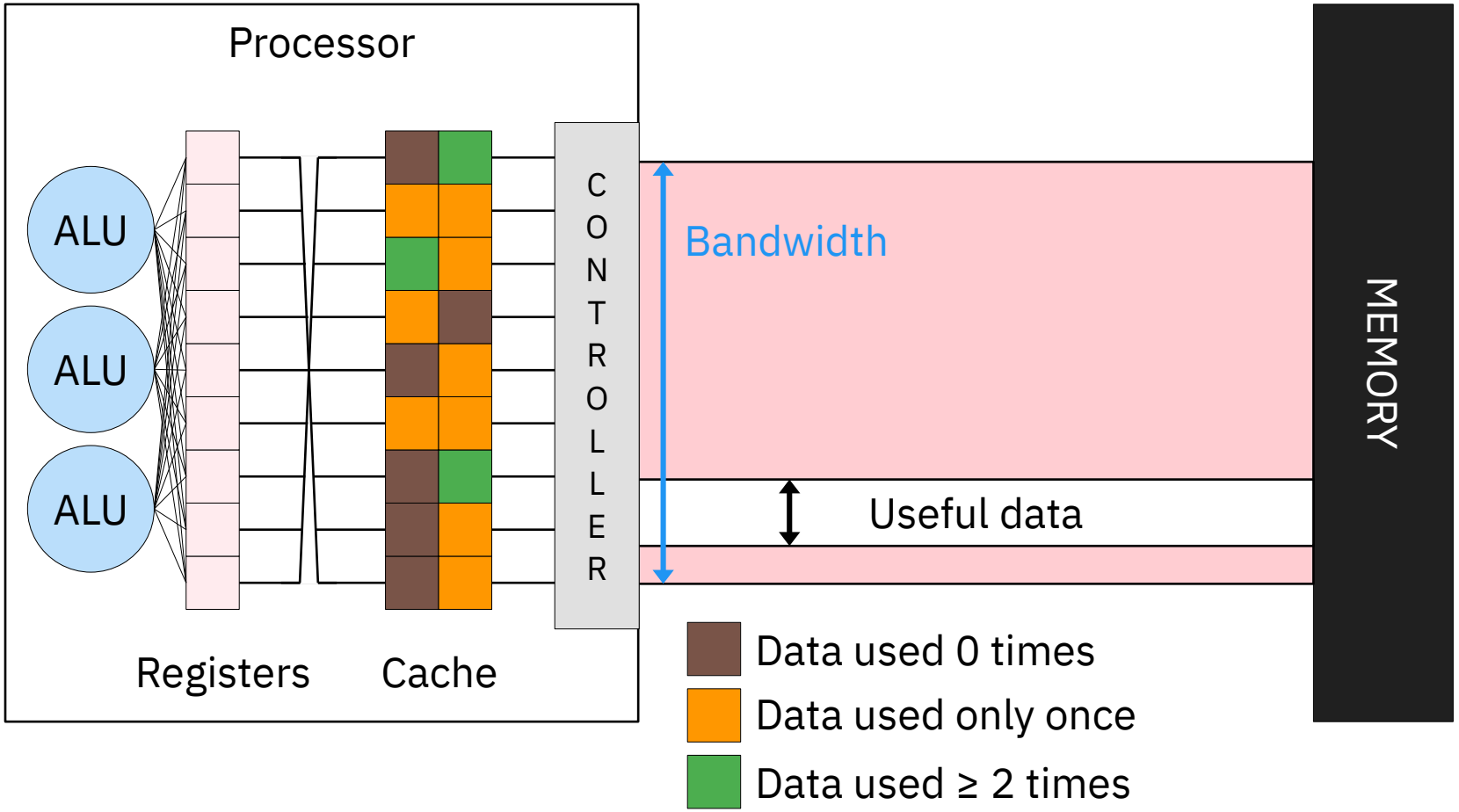
Multiple Memory Technologies



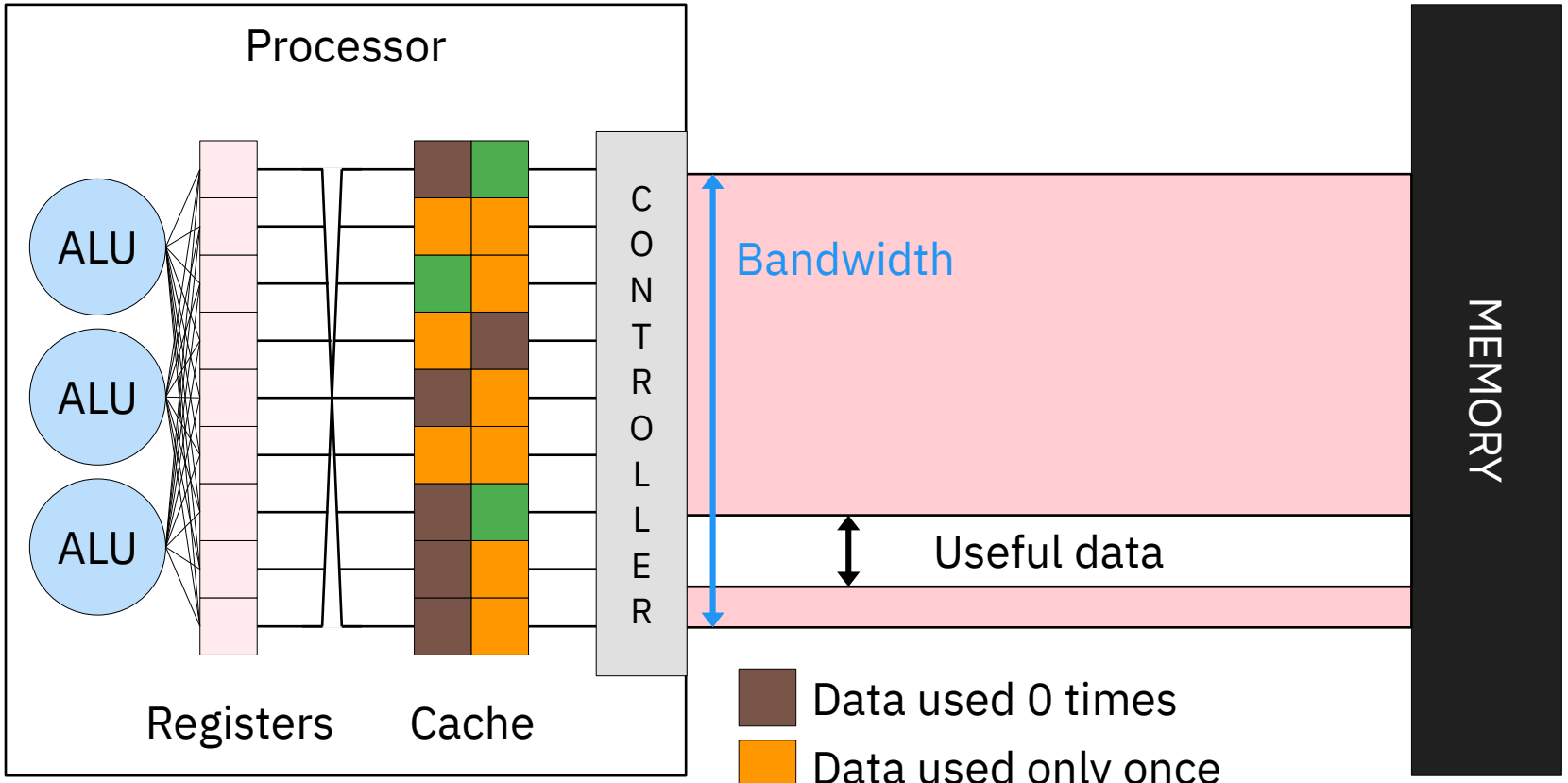
Multiple Memory Technologies



Programs are Still Bandwidth-Bound!

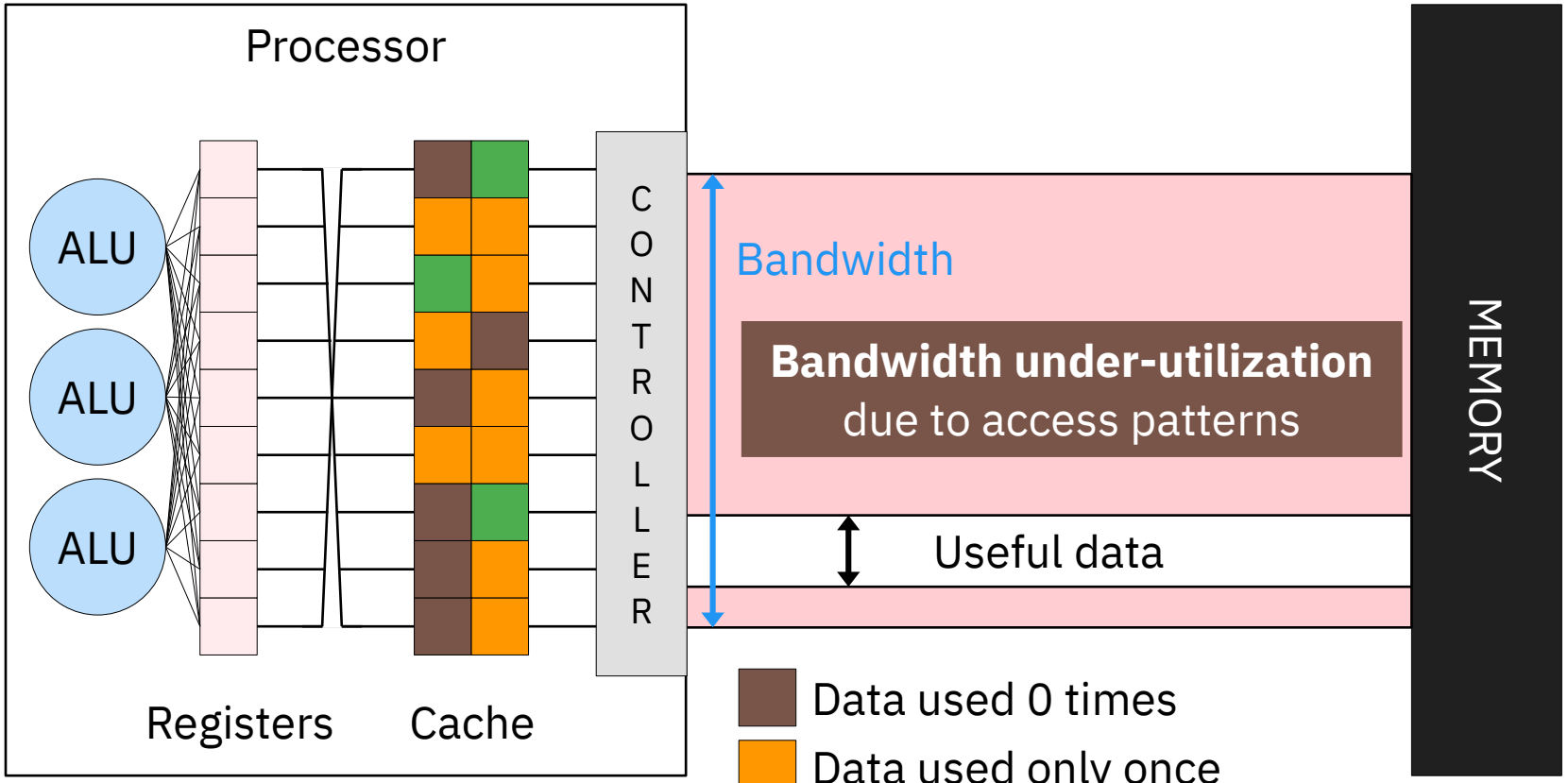


Programs are Still Bandwidth-Bound!



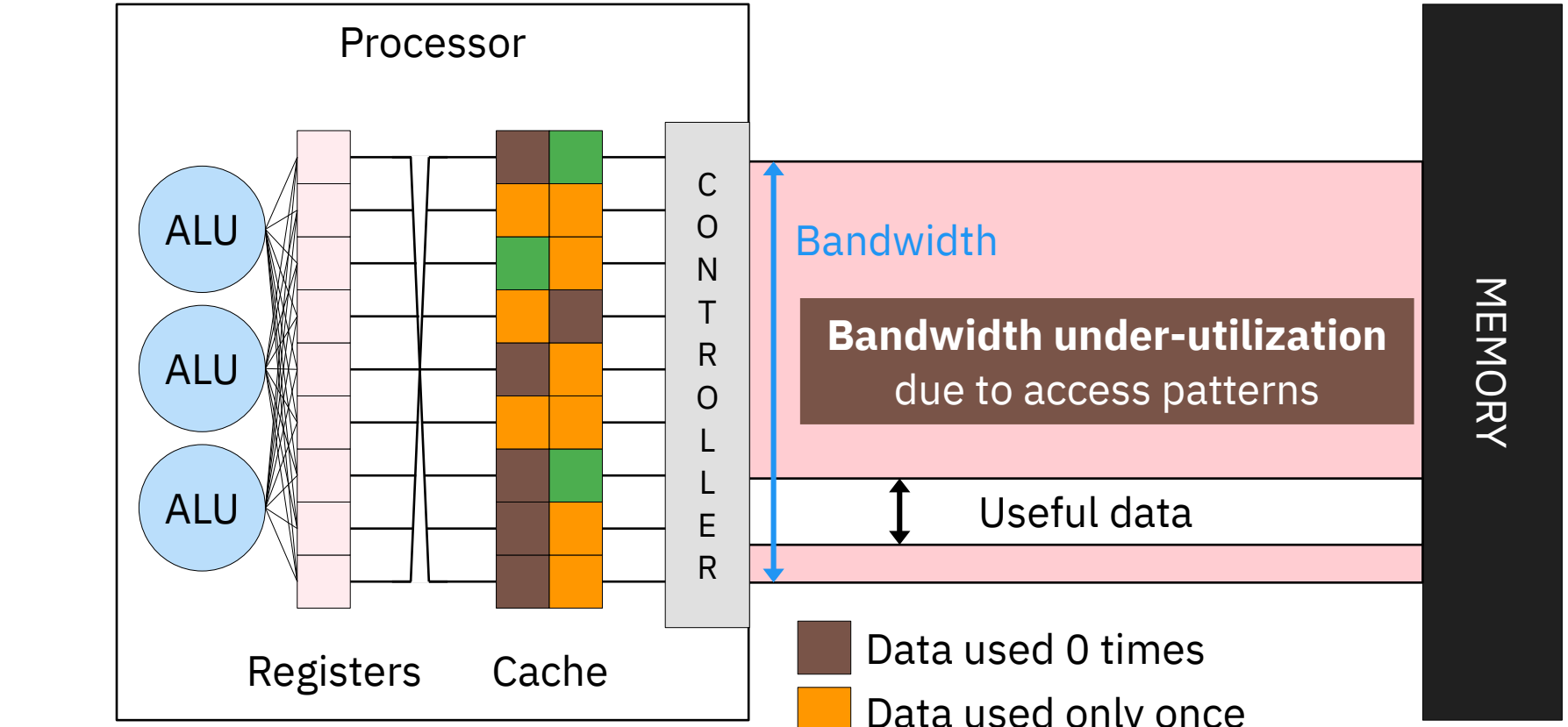
Excessive usage of memory, not enough re-use of on-chip memories

Programs are Still Bandwidth-Bound!



Excessive usage of memory, not enough re-use of on-chip memories

Programs are Still Bandwidth-Bound!



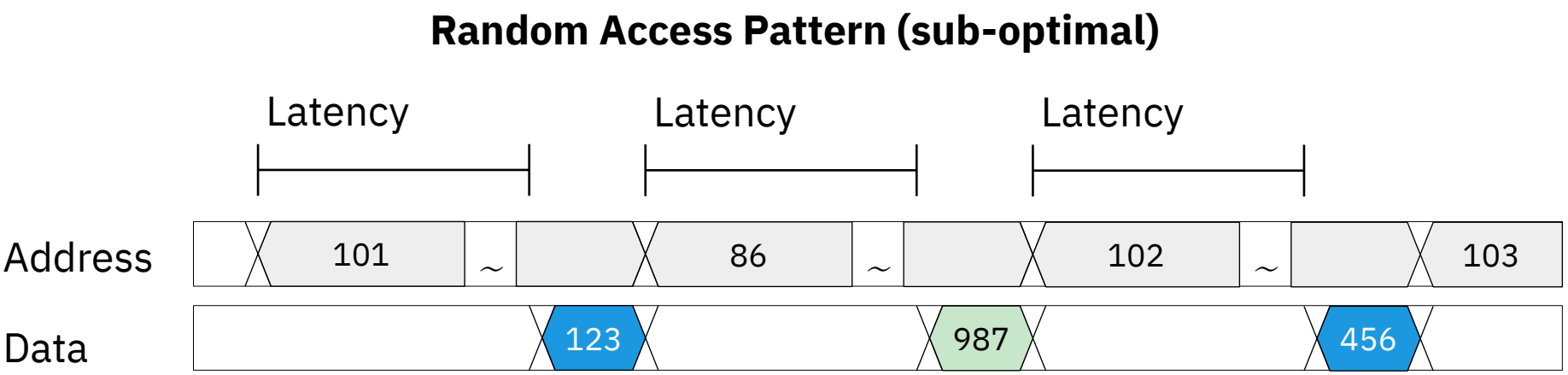
Excessive usage of memory, not enough re-use of on-chip memories

High bandwidth available, but programs still bandwidth-bound!

Outline

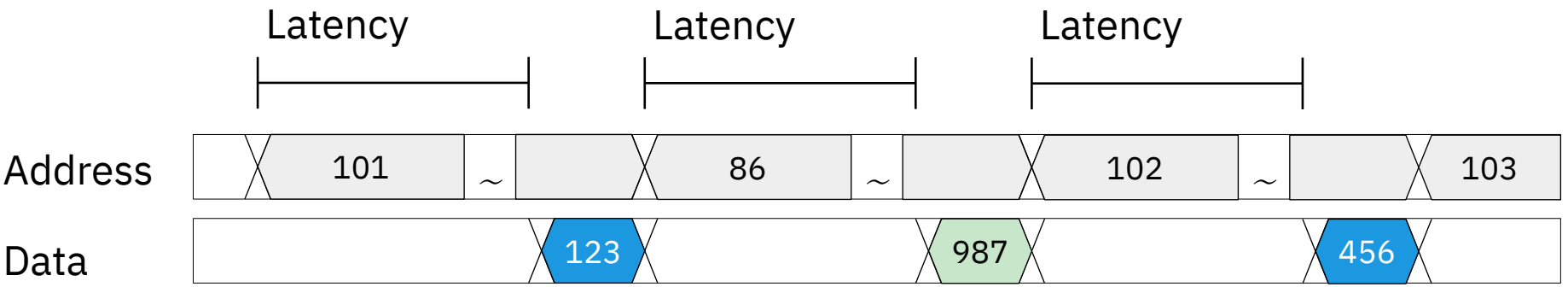
- Introduction and Motivation
- **Background**
  - **Memory controllers**
  - **FPGAs and High-Level Synthesis**
  - **Polyhedral model and high-level transformations**
- Proposed Solutions
- Conclusions

On the good usage of memory controllers

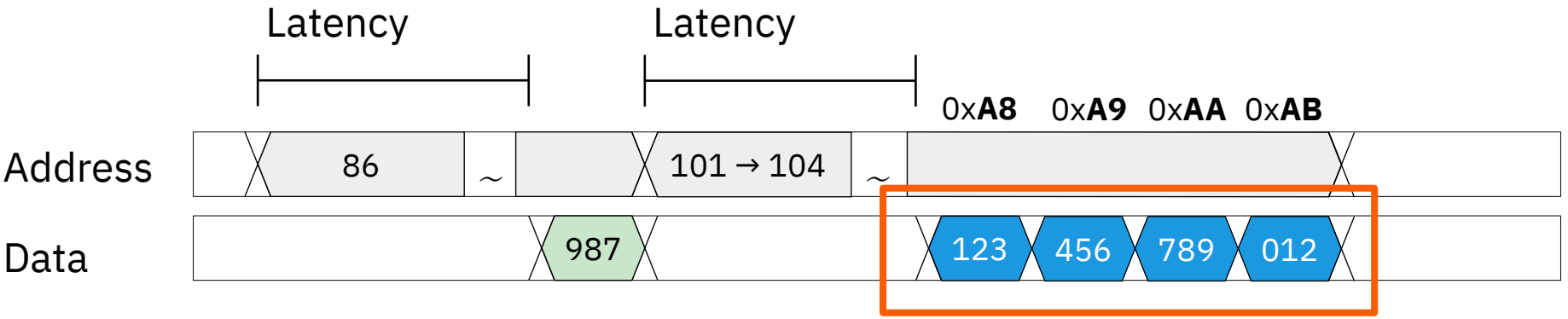


On the good usage of memory controllers

Random Access Pattern (sub-optimal)

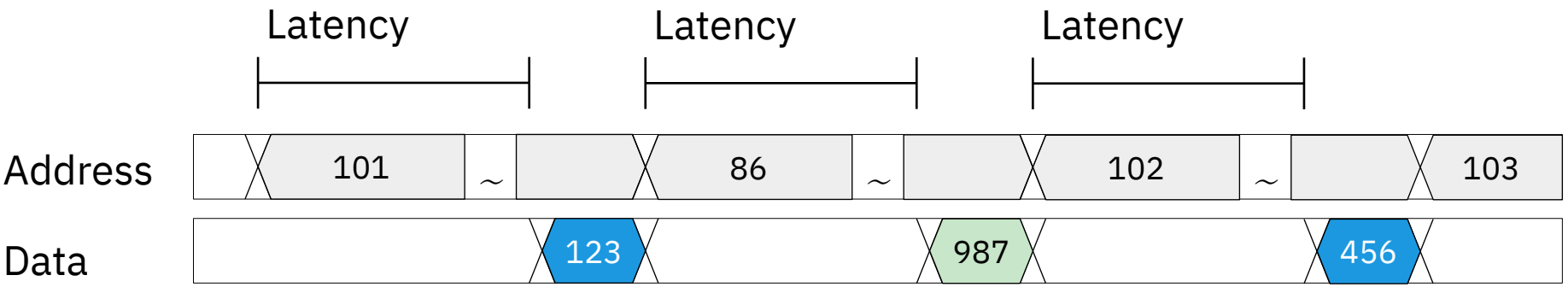


Burst Access Pattern (optimal)

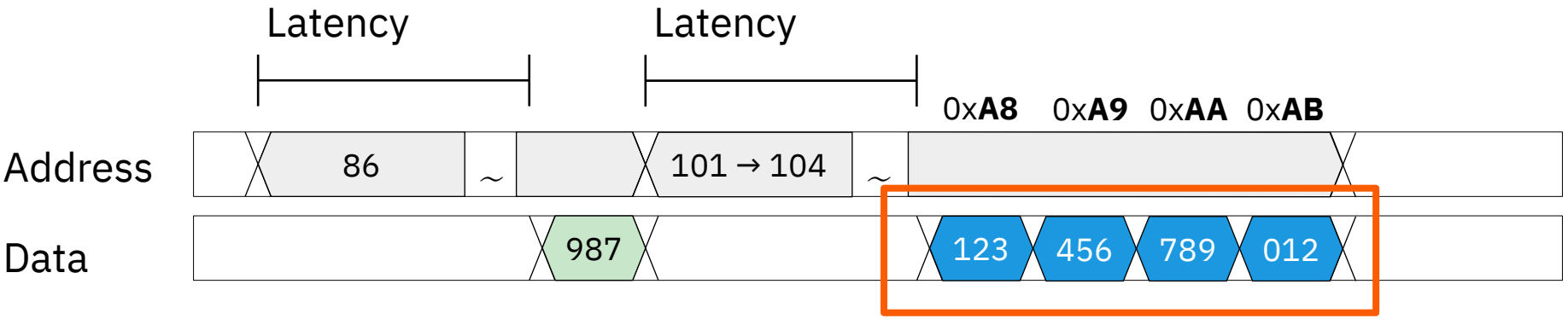


On the good usage of memory controllers

Random Access Pattern (sub-optimal)



Burst Access Pattern (optimal)

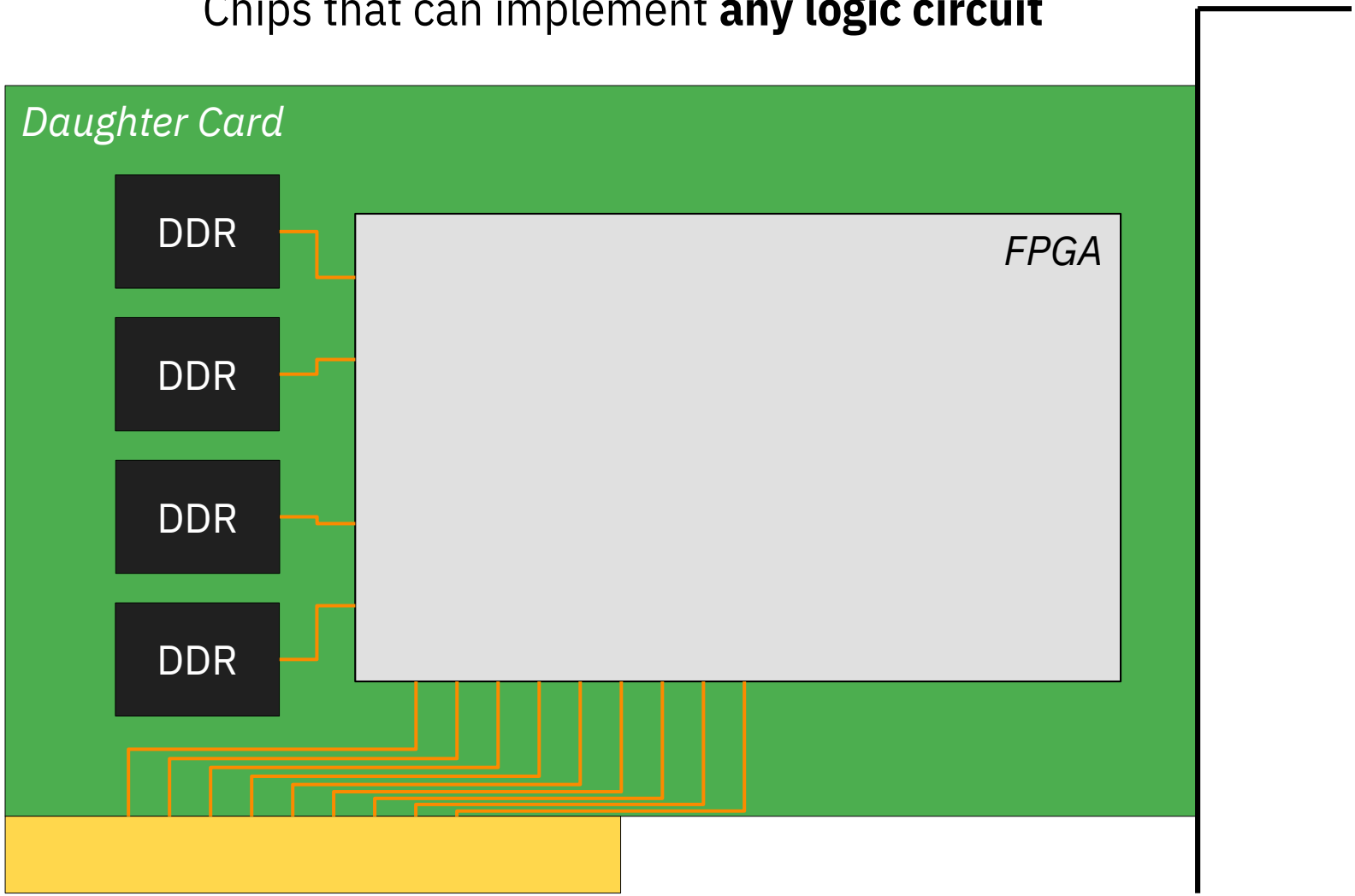


Burst = Access consecutive addresses in a row → requires **contiguity**

What is an FPGA ?

Field-Programmable Gate Array

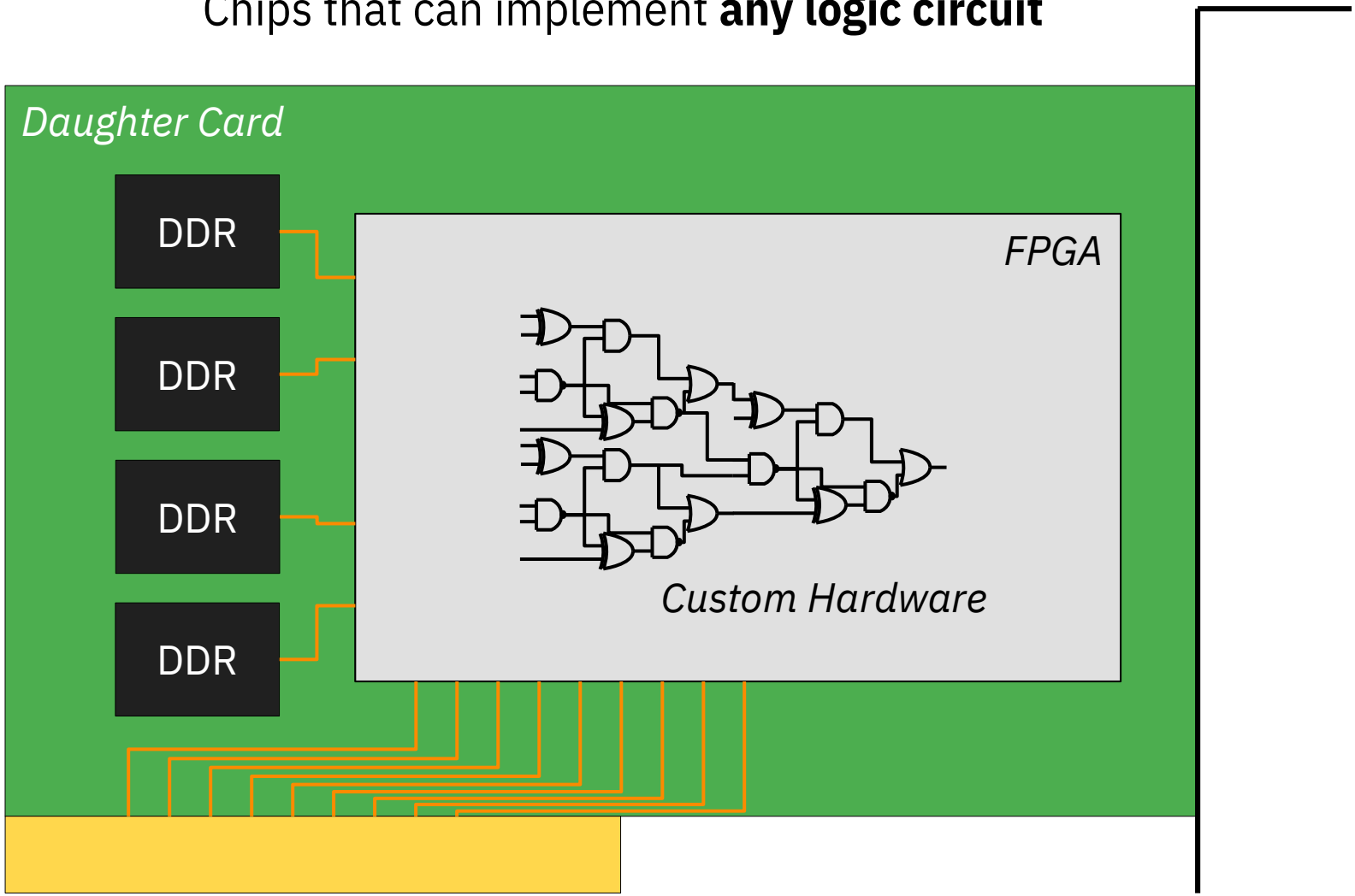
Chips that can implement **any logic circuit**



What is an FPGA ?

Field-Programmable Gate Array

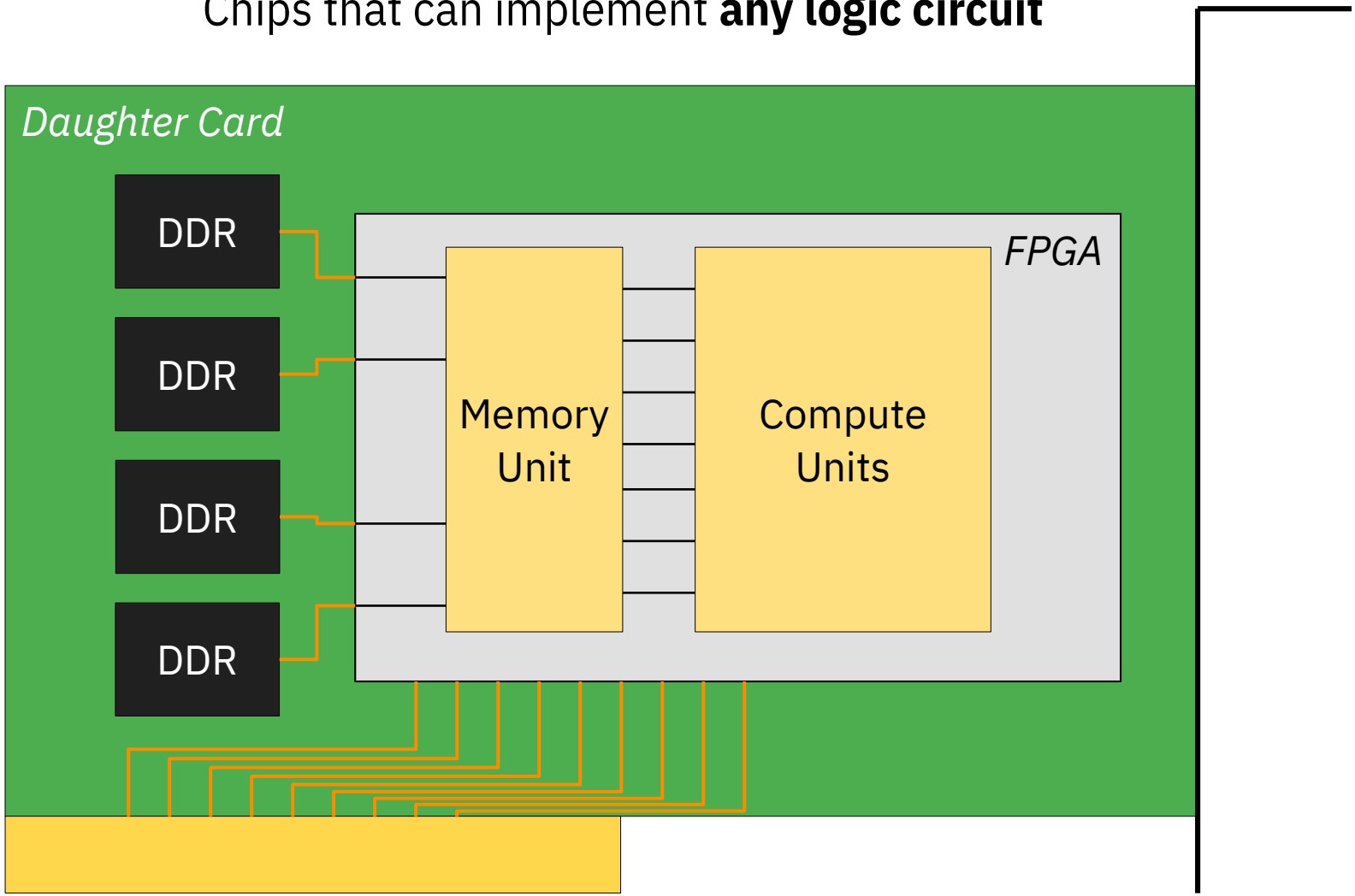
Chips that can implement **any logic circuit**



What is an FPGA ?

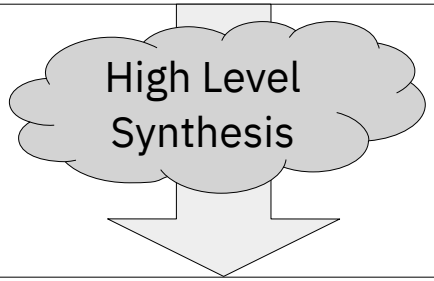
Field-Programmable Gate Array

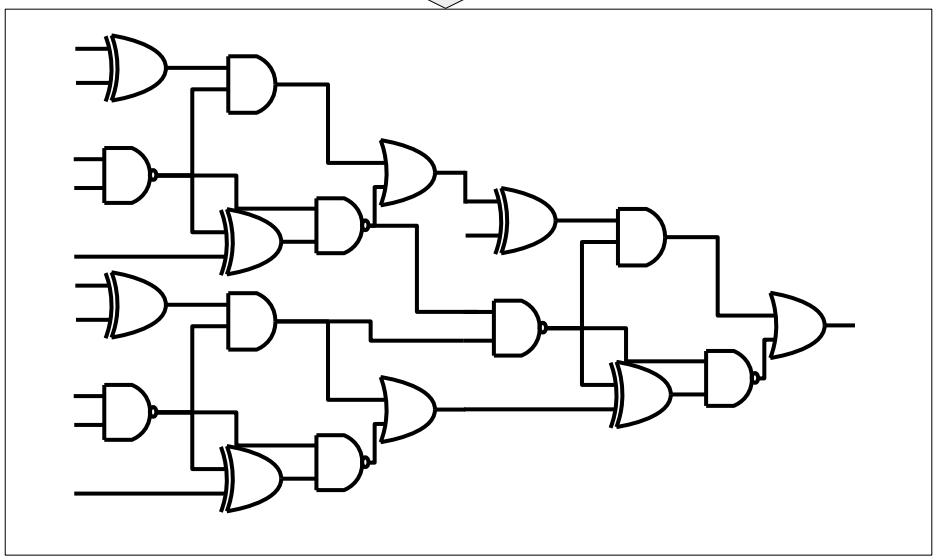
Chips that can implement **any logic circuit**



High-Level Synthesis of FPGA Archs

```
for(int i = 1; i <= N-1; i++) {
  A[i] = (B[i-1] + B[i] + B[i+1]) / 3;
}
```

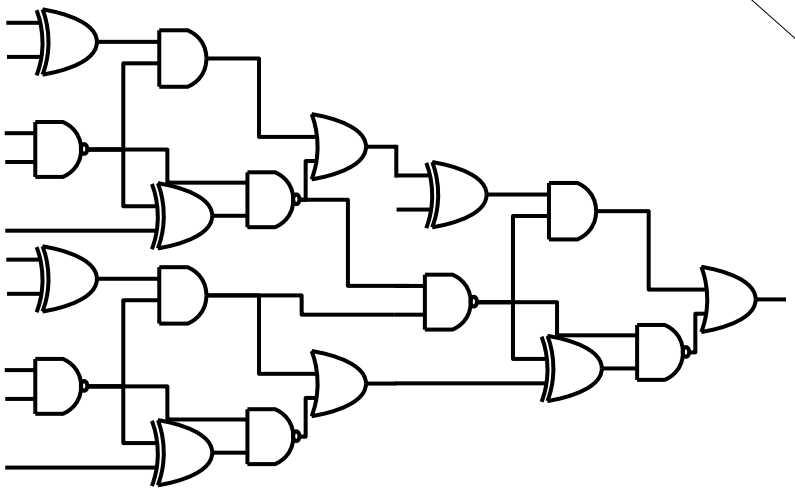


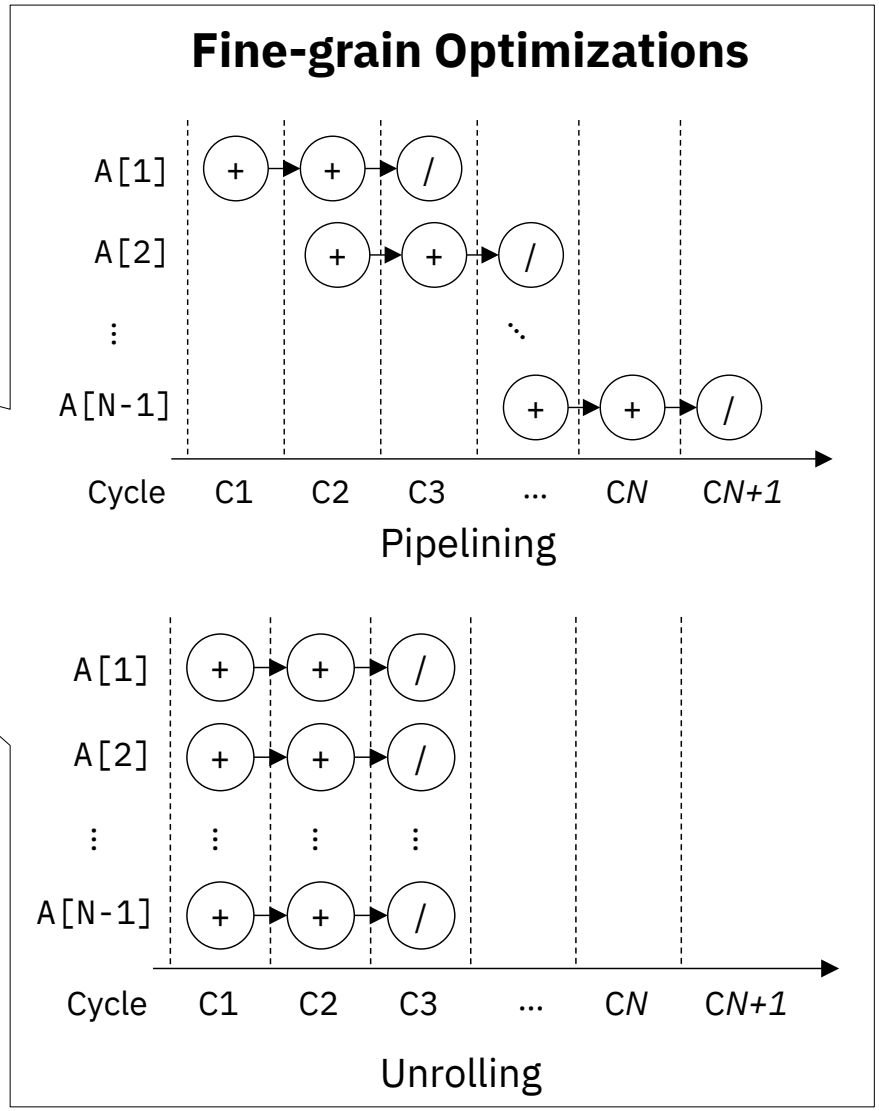


High-Level Synthesis of FPGA Archs

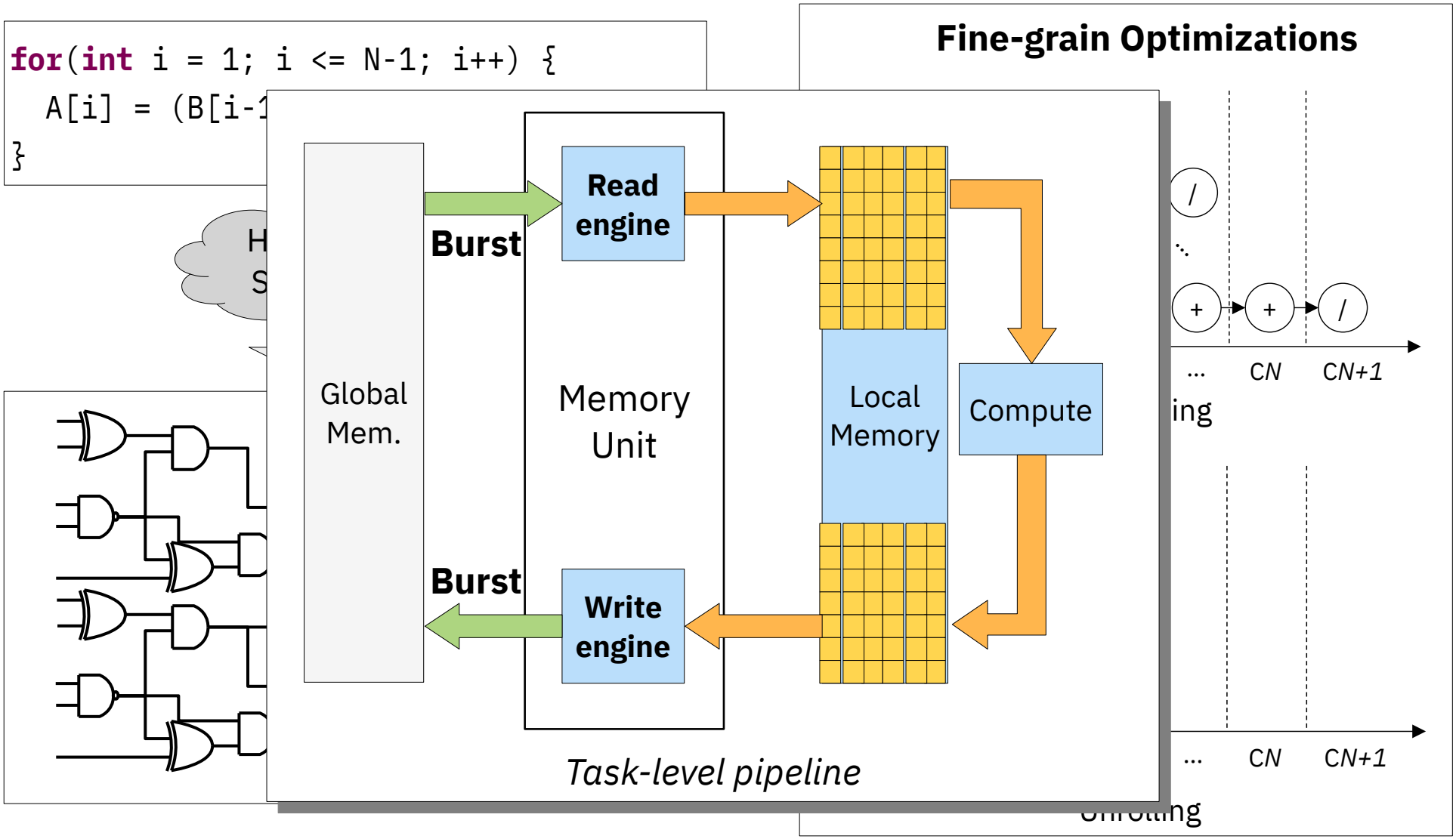
```
for(int i = 1; i <= N-1; i++) {
  A[i] = (B[i-1] + B[i] + B[i+1]) / 3;
}
```

High Level Synthesis

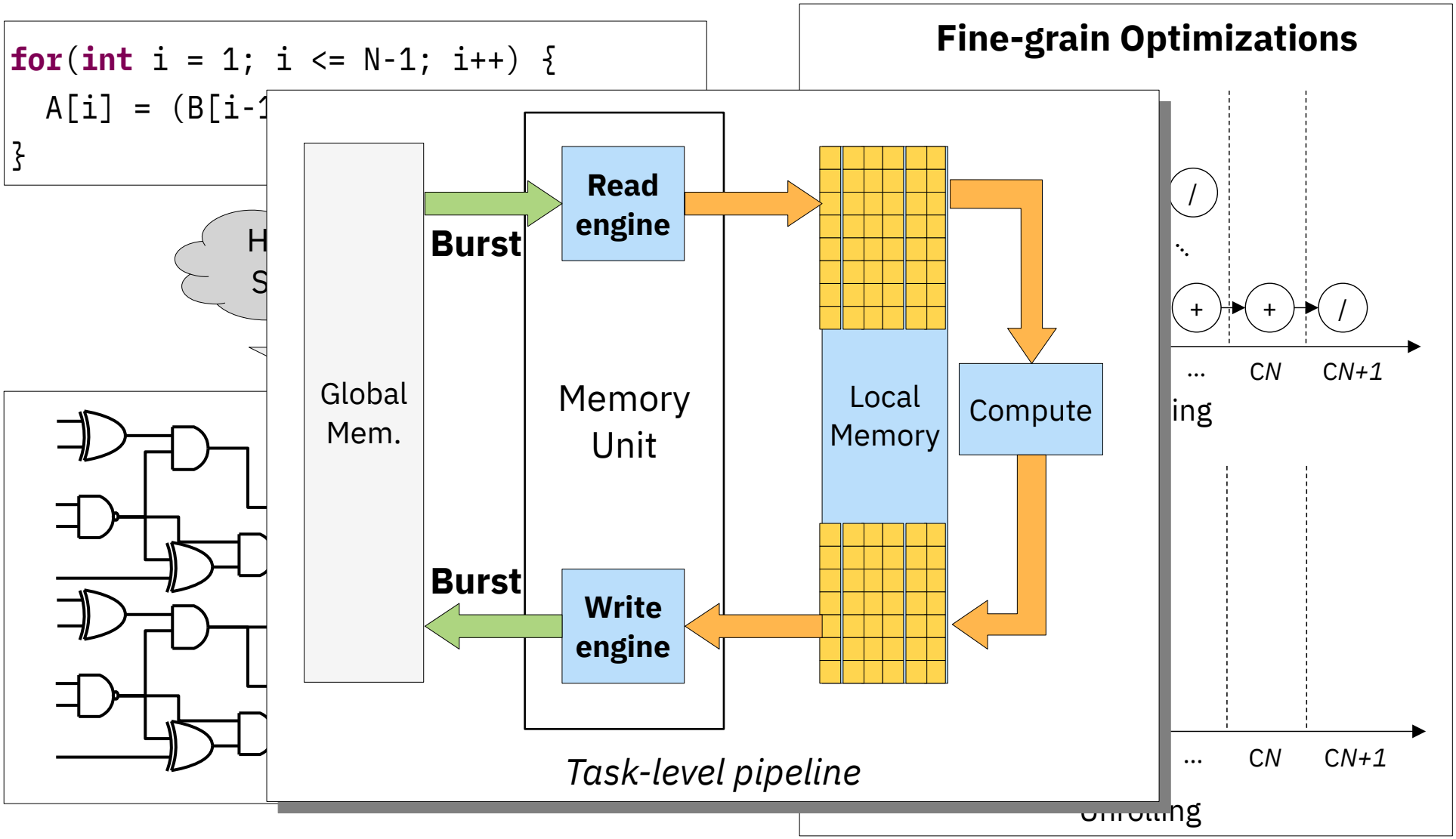




High-Level Synthesis of FPGA Archs



High-Level Synthesis of FPGA Archs

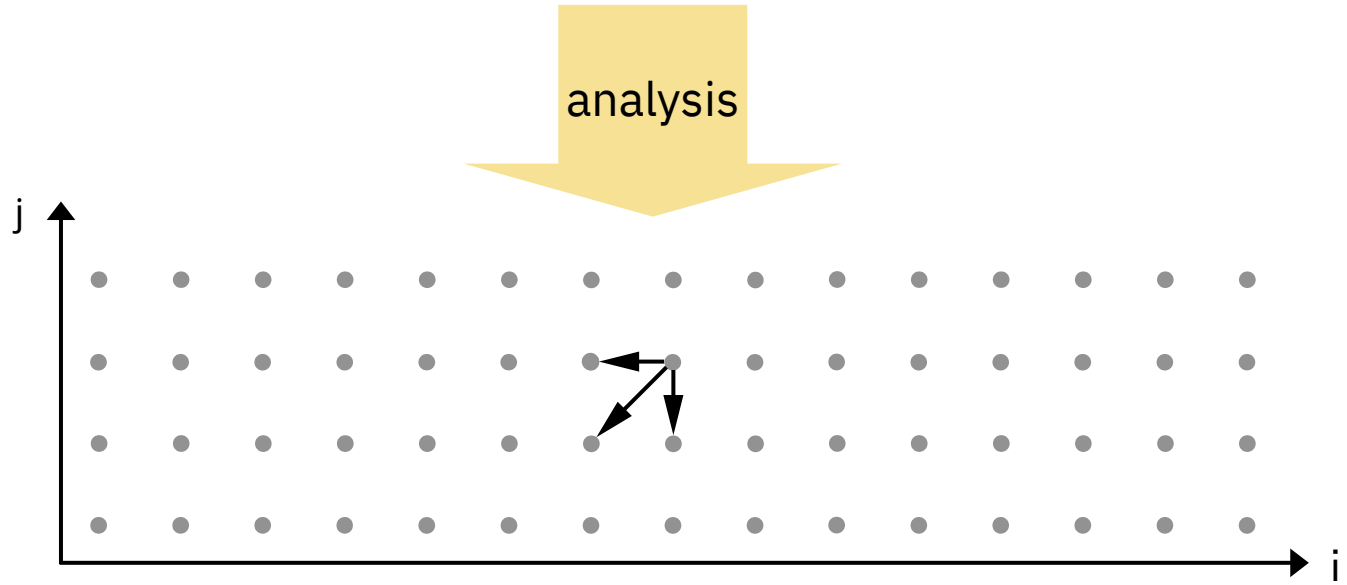


HLS → parallel architecture + burst memory interface

The polyhedral model

- Target: Computational kernels that admit a *polyhedral representation*
  - Iteration space + dependence function (e.g. from Array Dataflow Analysis [1])

```
for(int i = 0; i < N; ++i) {
  for(int j = 0; j < M; ++j) {
    C[i][j] = max(C[i][j-1], C[i-1][j], C[i-1][j-1] + W);
  }
}
```



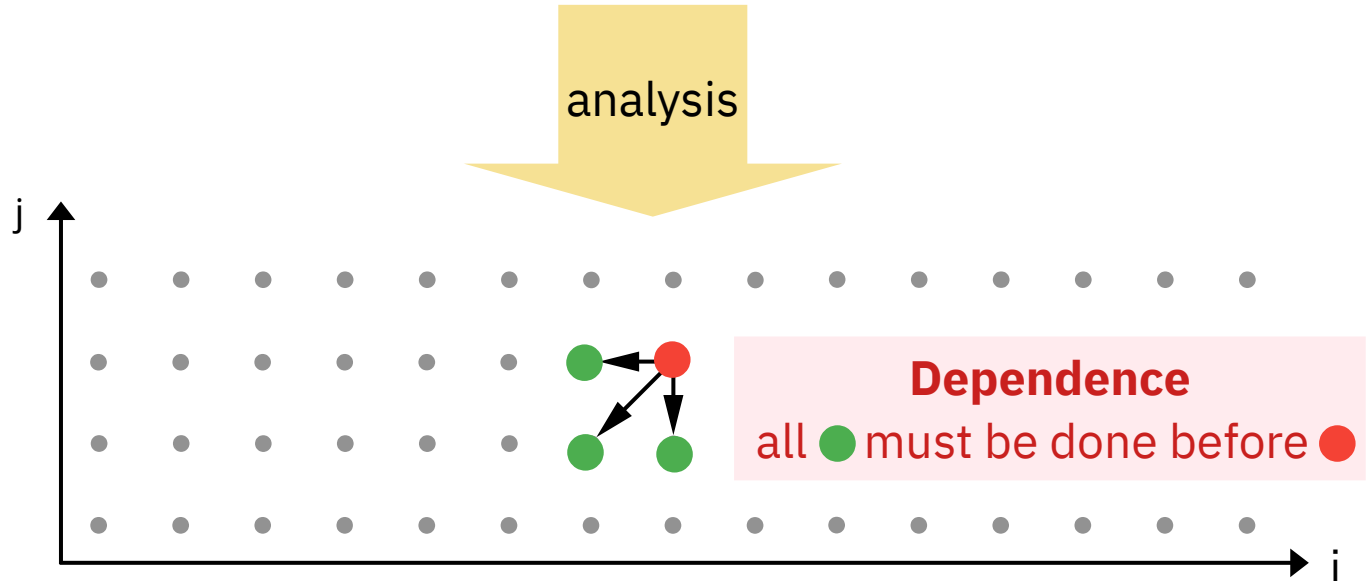
Smith-Waterman kernel iteration space and dependences

[1] Feautrier, P. *Dataflow analysis of array and scalar references.* International Journal of Parallel Programming, Springer Science and Business Media LLC, 1991, 20, 23-53

The polyhedral model

- Target: Computational kernels that admit a *polyhedral representation*
  - Iteration space + dependence function (e.g. from Array Dataflow Analysis [1])

```
for(int i = 0; i < N; ++i) {
  for(int j = 0; j < M; ++j) {
    C[i][j] = max(C[i][j-1], C[i-1][j], C[i-1][j-1] + W);
  }
}
```



Smith-Waterman kernel iteration space and dependences

[1] Feautrier, P. *Dataflow analysis of array and scalar references.* International Journal of Parallel Programming, Springer Science and Business Media LLC, 1991, 20, 23-53

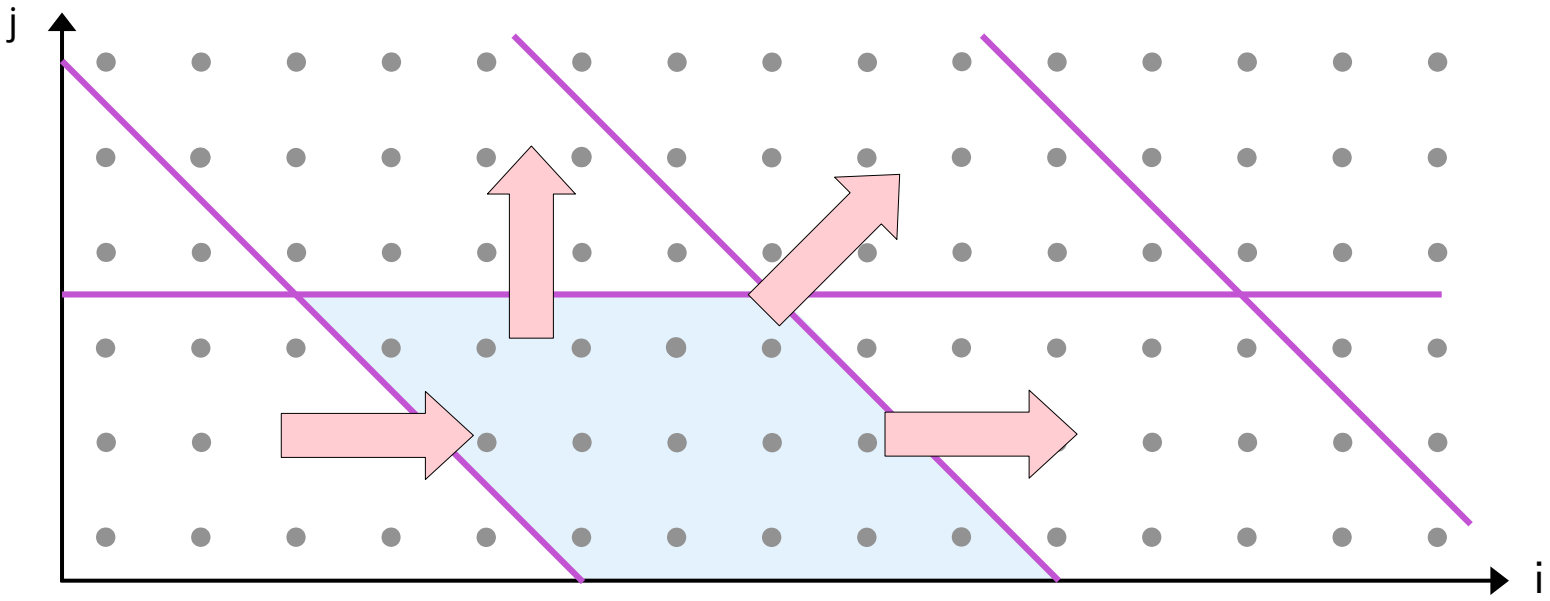
A Locality Optimization: Loop Tiling

- Break up the iteration space ➜ **Improve Locality**

  Each tile's footprint fits in local memory

- Cut must be **legal with respect to dependences**

  *No back-and-forth dependences between tiles **(atomicity)***



Tiling of Smith-Waterman kernel iteration space

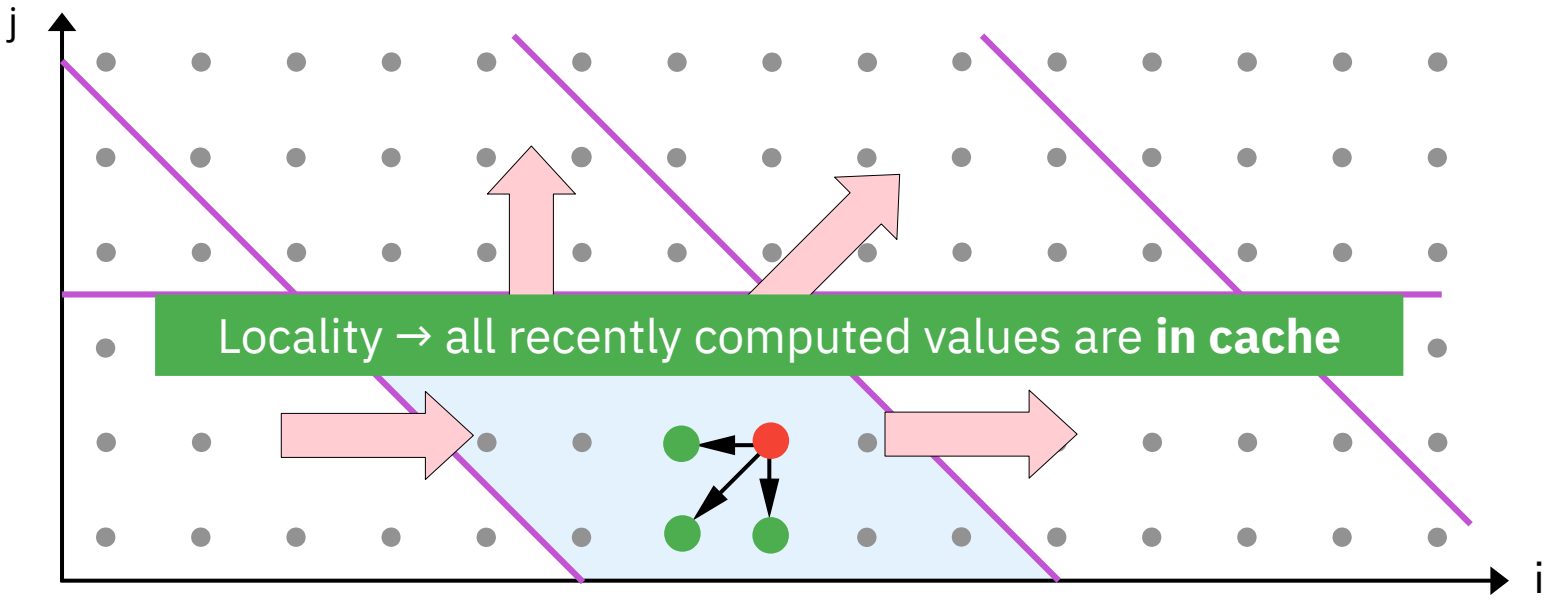
A Locality Optimization: Loop Tiling

- Break up the iteration space ➜ **Improve Locality**

  Each tile's footprint fits in local memory

- Cut must be **legal with respect to dependences**

  *No back-and-forth dependences between tiles **(atomicity)***



Tiling of Smith-Waterman kernel iteration space

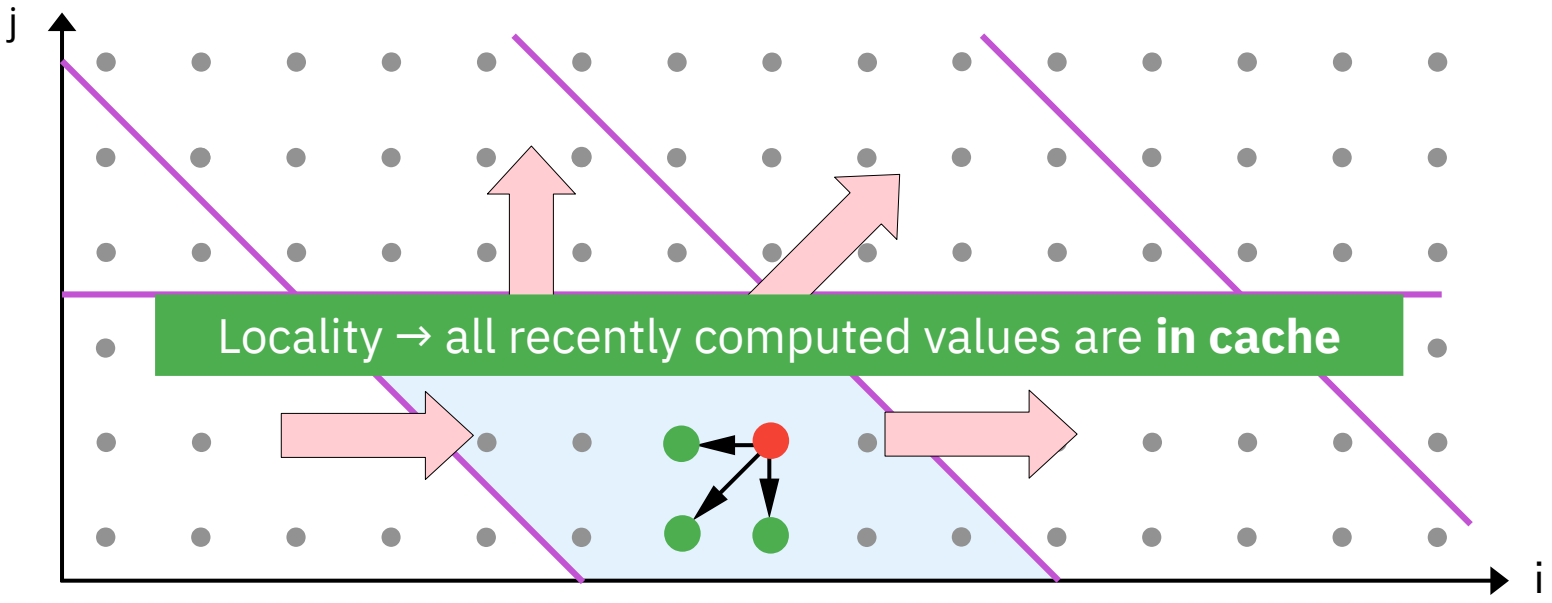
A Locality Optimization: Loop Tiling

- Break up the iteration space ➜ **Improve Locality**

  Each tile's footprint fits in local memory

- Cut must be **legal with respect to dependences**

  *No back-and-forth dependences between tiles **(atomicity)***



Tiling of Smith-Waterman kernel iteration space

Apply Loop Tiling for locality *then* look for spatial locality optimizations

Many Polyhedral Locality Optimizations

- **Loop tiling**

  Tile shapes : overlapped [1], diamond [2], algebraic [3], etc.

  Tile size : automatic selection [4, 5], optimization [6], etc.

Many Polyhedral Locality Optimizations

- **Loop tiling**

  Tile shapes : overlapped [1], diamond [2], algebraic [3], etc.

  Tile size : automatic selection [4, 5], optimization [6], etc.

- **Cache-specific** optimizations

  Cache miss equations [7], conflict avoidance by padding [8], etc.

Many Polyhedral Locality Optimizations

- **Loop tiling**

  Tile shapes : overlapped [1], diamond [2], algebraic [3], etc.

  Tile size : automatic selection [4, 5], optimization [6], etc.

- **Cache-specific** optimizations

  Cache miss equations [7], conflict avoidance by padding [8], etc.

- **Automated** in optimizing compilers (e.g. Polly [9])

[1] Zhou, X.; Giacalone, J.-P.; Garzarán, M. J.; Kuhn, R. H.; Ni, Y. & Padua, D. *Hierarchical overlapped tiling.* Proceedings of the Tenth International Symposium on Code Generation and Optimization, ACM, 2012.

[2] Bondhugula, U.; Bandishti, V. & Pananilath, I. *Diamond Tiling: Tiling Techniques to Maximize Parallelism for Stencil Computations.*IEEE Transactions on Parallel and Distributed Systems, Institute of Electrical and Electronics Engineers (IEEE), 2017, 28, 1285-1298ceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, ACM, 2021

[3] Rossetti, C. & Clauss, P. *Algebraic Tiling.* IMPACT 2023, 13th International Workshop onPolyhedral Compilation Techniques, Jan 2023, Toulouse, France, 2023

[4] Coleman, S. & McKinley, K. S. *Tile Size Selection Using Cache Organization and Data Layout.* SIGPLAN Not., Association for Computing Machinery, 1995, 30, 279–290

[5] Mehta, S.; Beeraka, G. & Yew, P.-C. *Tile Size Selection Revisited.* ACM Trans. Archit. Code Optim., Association for Computing Machinery, 2013, 10

[6] Bondhugula, U.; Hartono, A.; Ramanujam, J. & Sadayappan, P. *A Practical Automatic Polyhedral Program Optimization System.* ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), 2008

[7] Ghosh, S.; Martonosi, M. & Malik, S. *Cache miss equations.* ACM Transactions on Programming Languages and Systems, Association for Computing Machinery (ACM), 1999, 21, 703-746

[8] Hong, C.; Bao, W.; Cohen, A.; Krishnamoorthy, S.; Pouchet, L.-N.; Rastello, F.; Ramanujam, J. & Sadayappan, P. *Effective Padding of Multidimensional Arrays to Avoid Cache Conflict Misses.* SIGPLAN Not., Association for Computing Machinery, 2016, 51, 129–144

[9] Grosser, T.; Groesslinger, A. & Lengauer, C. *Polly — Performing Polyhedral Optimizations On A Low-Level Intermediate Representation.* Parallel Processing Letters, World Scientific Pub Co Pte Lt, 2012, 22, 1250010

Many Polyhedral Locality Optimizations

- **Loop tiling**

  Tile shapes : overlapped [1], diamond [2], algebraic [3], etc.

  Tile size : automatic selection [4, 5], optimization [6], etc.

- **Cache-specific** optimizations

  Cache miss equations [7], conflict avoidance by padding [8], etc.

- **Automated** in optimizing compilers (e.g. Polly [9])

Locality → well-studied, addressed issue

This work = "Spatial" locality (contiguity)

[1] Zho
Genera

[2] Bor
Distributed Systems, Institute of Electrical and Electronics Engineers (IEEE), 2017, 28, 1285-1298ceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, ACM, 2021

[3] Rossetti, C. & Clauss, P. *Algebraic Tiling.* IMPACT 2023, 13th International Workshop onPolyhedral Compilation Techniques, Jan 2023, Toulouse, France, 2023

[4] Coleman, S. & McKinley, K. S. *Tile Size Selection Using Cache Organization and Data Layout.* SIGPLAN Not., Association for Computing Machinery, 1995, 30, 279–290

[5] Mehta, S.; Beeraka, G. & Yew, P.-C. *Tile Size Selection Revisited.* ACM Trans. Archit. Code Optim., Association for Computing Machinery, 2013, 10

[6] Bondhugula, U.; Hartono, A.; Ramanujam, J. & Sadayappan, P. *A Practical Automatic Polyhedral Program Optimization System.* ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), 2008

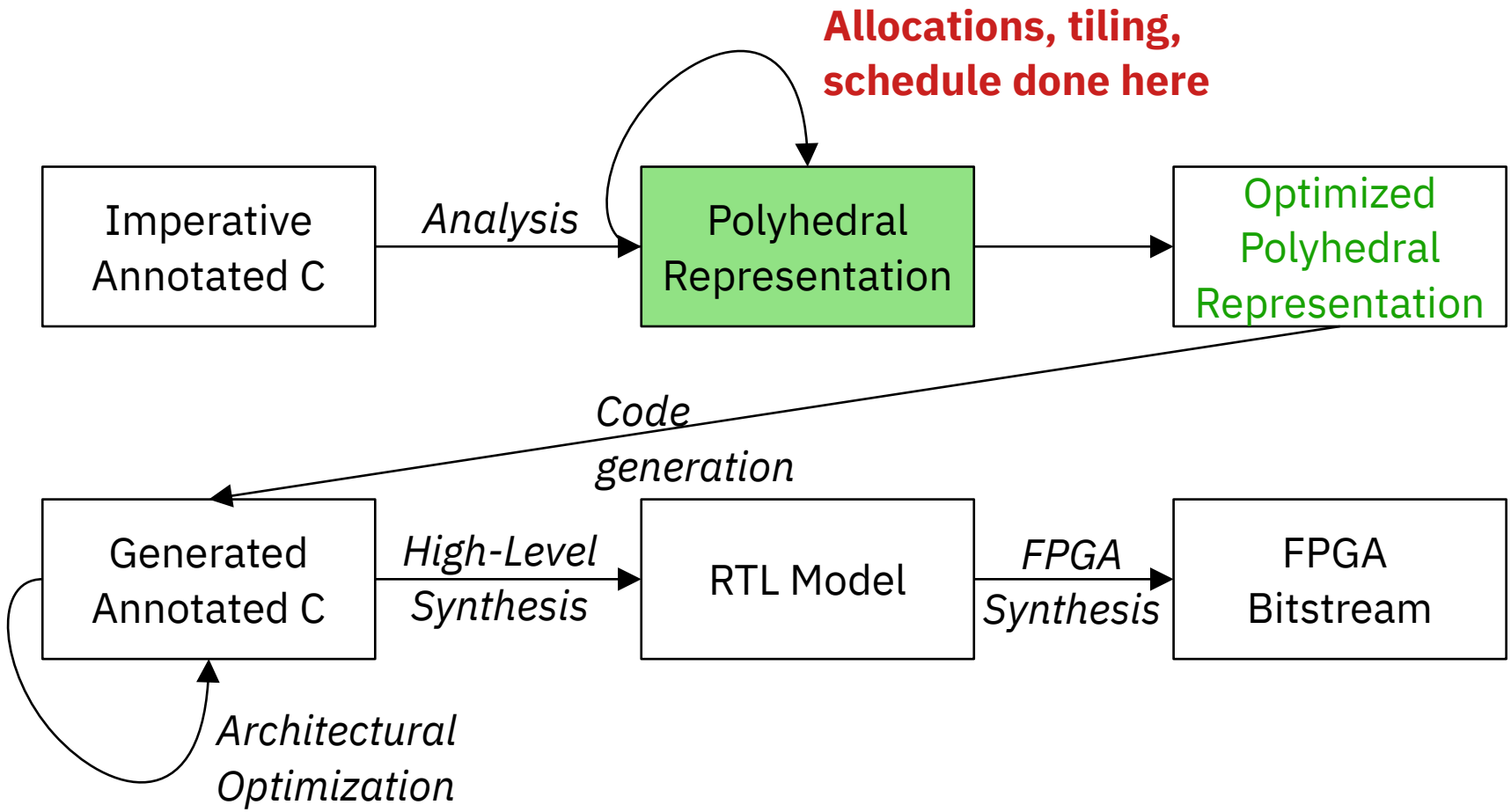
[7] Ghosh, S.; Martonosi, M. & Malik, S. *Cache miss equations.* ACM Transactions on Programming Languages and Systems, Association for Computing Machinery (ACM), 1999, 21, 703-746

[8] Hong, C.; Bao, W.; Cohen, A.; Krishnamoorthy, S.; Pouchet, L.-N.; Rastello, F.; Ramanujam, J. & Sadayappan, P. *Effective Padding of Multidimensional Arrays to Avoid Cache Conflict Misses.* SIGPLAN Not., Association for Computing Machinery, 2016, 51, 129–144

[9] Grosser, T.; Groesslinger, A. & Lengauer, C. *Polly — Performing Polyhedral Optimizations On A Low-Level Intermediate Representation.* Parallel Processing Letters, World Scientific Pub Co Pte Lt, 2012, 22, 1250010

Polyhedral HLS Compiler Flow

- Optimizations on **Polyhedral Representation** (very high-level) and imperative code (high-level C/C++)
- Output : FPGA bitstream



Memory access pattern optimizations

- Decomposition of **Inter-Tile Communications**:

  Datharthri et al., 2013 [3]: Flow-In/Flow-Out partitioning

  → Our work = one specific case, **static determination at compile time**

  Zhao et al, 2021 [4]: partitioning + layout

  → Our work = **« generalization » to uniform dependences**

Memory access pattern optimizations

- Decomposition of **Inter-Tile Communications**:

  Datharthri et al., 2013 [3]: Flow-In/Flow-Out partitioning

  → Our work = one specific case, **static determination at compile time**

  Zhao et al, 2021 [4]: partitioning + layout

  → Our work = **« generalization » to uniform dependences**

- Memory Layout for **Host-Accelerator Communications**:

  Ozturk et al., 2009 [5]: data tiling + compression

  → Our work = **finer-grain data breakdown** amenable to compression

Memory access pattern optimizations

- Decomposition of **Inter-Tile Communications**:

  Datharthri et al., 2013 [3]: Flow-In/Flow-Out partitioning

  → Our work = one specific case, **static determination at compile time**

  Zhao et al, 2021 [4]: partitioning + layout

  → Our work = **« generalization » to uniform dependences**

- Memory Layout for **Host-Accelerator Communications**:

  Ozturk et al., 2009 [5]: data tiling + compression

  → Our work = **finer-grain data breakdown** amenable to compression

- **Allocation** from Polyhedral Representation:

  Yuki and Rajopadhye, 2013 [6]: reduce memory footprint with uniform dependences

  → Our work = **objective is bandwidth utilization**

[3] Dathathri, R.; Reddy, C.; Ramashekar, T. & Bondhugula, U. *Generating Efficient Data Movement Code for Heterogeneous Architectures with Distributed-Memory.* Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques, IEEE, 2013

[4] Zhao, T.; Hall, M.; Johansen, H. & Williams, S. *Improving communication by optimizing on-node data movement with data layout* Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, ACM, 2021

[5] Ozturk, O.; Kandemir, M. & Irwin, M. *Using Data Compression for Increasing Memory System Utilization*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Institute of Electrical and Electronics Engineers (IEEE), 2009, 28, 901-914

[6] Yuki, T. & Rajopadhye, S. *Memory allocations for tiled uniform dependence programs* IMPACT 2013, 2013, 13

Memory access pattern optimizations

- Decomposition of **Inter-Tile Communications**:

  Datharthri et al., 2013 [3]: Flow-In/Flow-Out partitioning

  → Our work = one specific case, **static determination at compile time**

  Zhao et al, 2021 [4]: partitioning + layout

  → Our work = **« generalization » to uniform dependences**

- Memory Layout for **Host-Accelerator Communications**:

  Ozturk et al., 2009 [5]: data tiling + compression

  → Our work = **finer-grain data breakdown** amenable to compression

- **Allocation** from Polyhedral Representation:

  Yuki and Rajopadhye, 2013 [6]: reduce memory footprint with uniform dependences

  → Our work = **objective is bandwidth utilization**

No existing compile-time, systematic data allocation from polyhedral representation

[3] Dathathri, R.; Reddy, C.; Ramashekar, T. & Bondhugula, U. *Generating Efficient Data Movement Code for Heterogeneous Architectures with Distributed-Memory.* Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques, IEEE, 2013

[4] Zha LAN Sympo

[5] Ozt Circuits and Systems, Institute of Electrical and Electronics Engineers (IEEE), 2009, 28, 901-914

[6] Yuki, T. & Rajopadhye, S. *Memory allocations for tiled uniform dependence programs* IMPACT 2013, 2013, 13

Approaches in this dissertation

Systematically derive memory allocations from the program itself

- **Goal: Lower transfer time, higher bandwidth utilization**

Approaches in this dissertation

Systematically derive memory allocations from the program itself

- **Goal: Lower transfer time, higher bandwidth utilization**

- **Methods:**
  - Use loop tiling for **locality**
  - Allocate only the necessary data into global memory = **low redundancy**
  - Access the data **contiguously**

Approaches in this dissertation

Systematically derive memory allocations from the program itself

- **Goal: Lower transfer time, higher bandwidth utilization**

- **Methods:**
  - Use loop tiling for **locality**
  - Allocate only the necessary data into global memory = **low redundancy**
  - Access the data **contiguously**

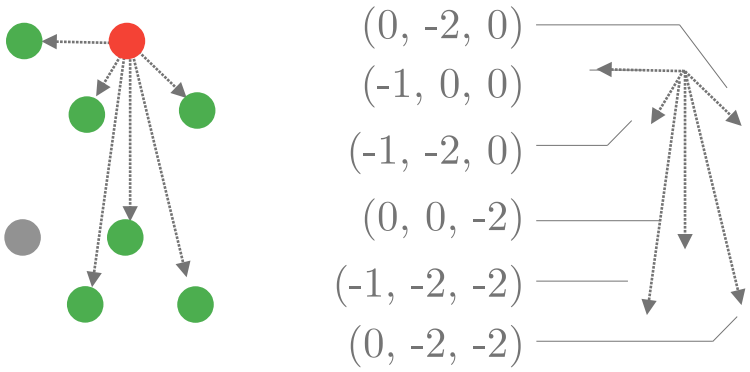
- **Solutions:**
  - A contiguous allocation for **rectangular tiles**, uniform dependences
  - A contiguous, irredundant allocation for **any tiling**, uniform dependences
  - A study on the case of **broadcast dependences**

Outline

- Introduction and Motivation
- Background
- Proposed Solutions
  - **Contiguous Allocation for Rectangular Tiles**
  - Partitioning the Data Flow of Programs with Uniform Dependences
  - Contiguous, Compressed Memory Allocation for Uniform Dependence Programs
  - Partitioning the Data Flow of Programs with Affine Dependences
- Conclusions

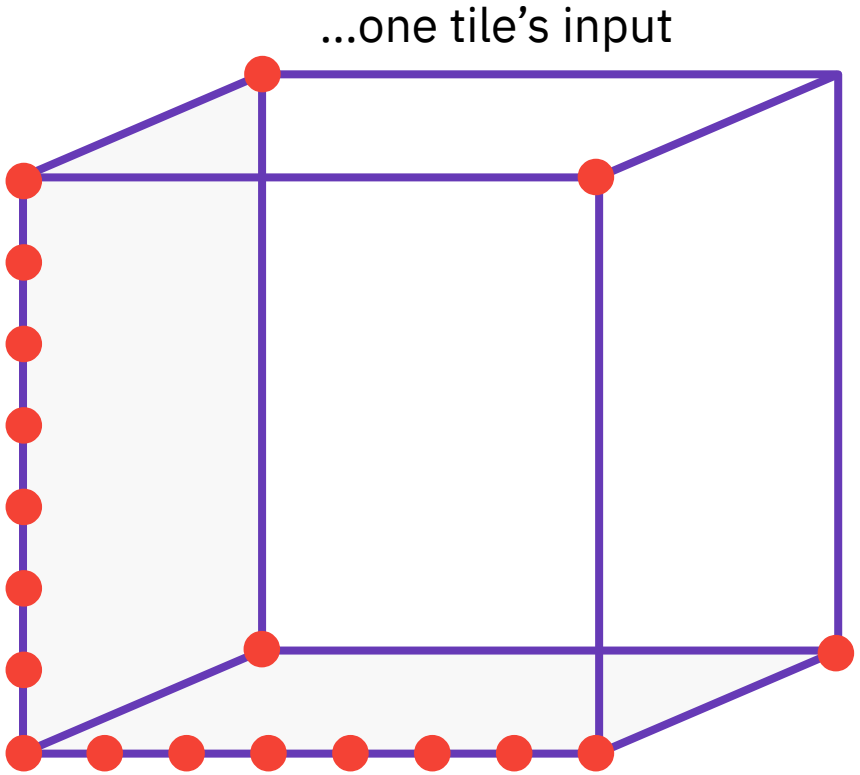
Tales of Tiles: the Flow-In Iterations...

One point's input...



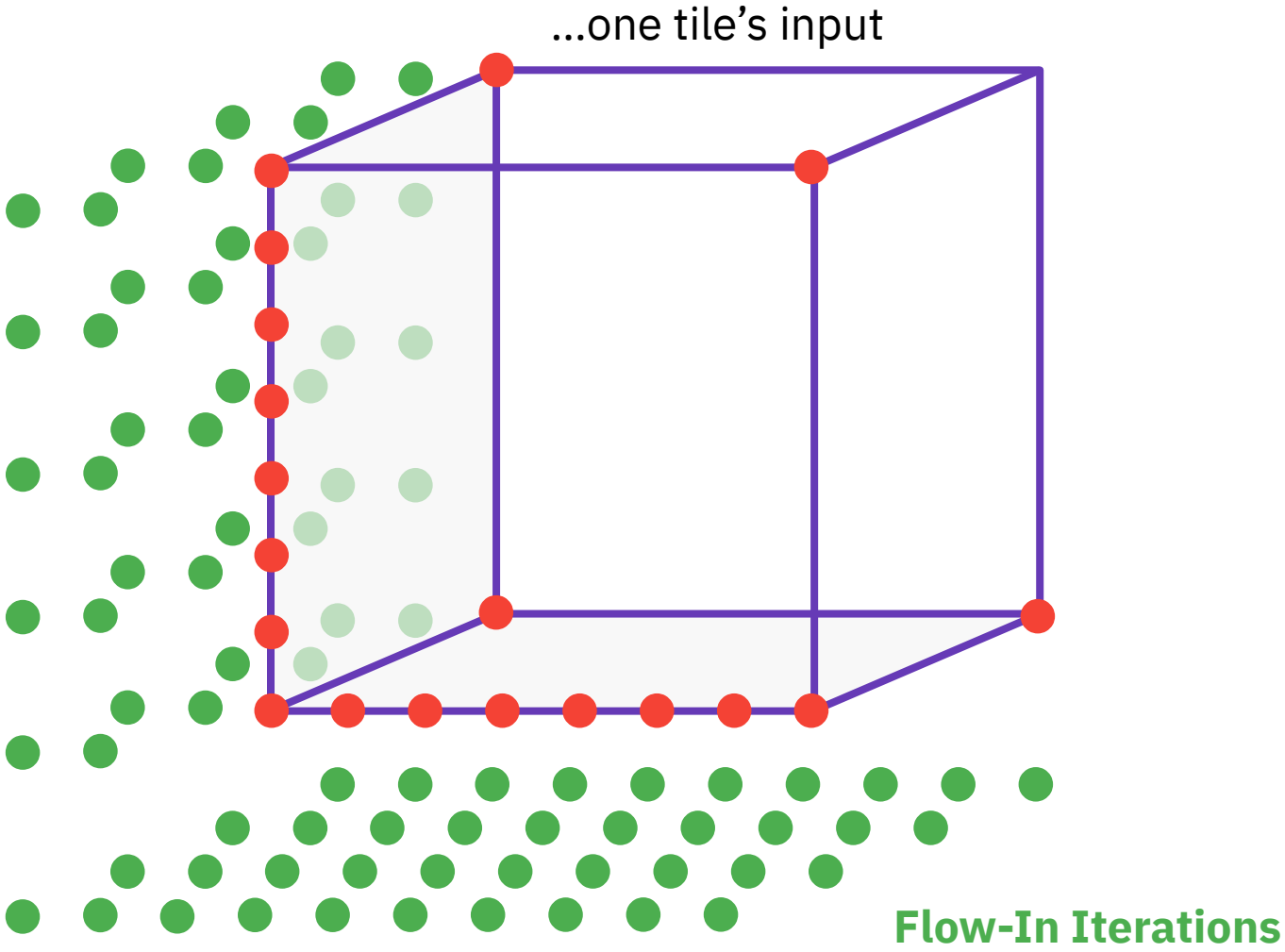
Bondhugula, U. *Compiling Affine Loop Nests for Distributed-Memory Parallel Architectures.* Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, ACM, 2013

Tales of Tiles: the Flow-In Iterations...



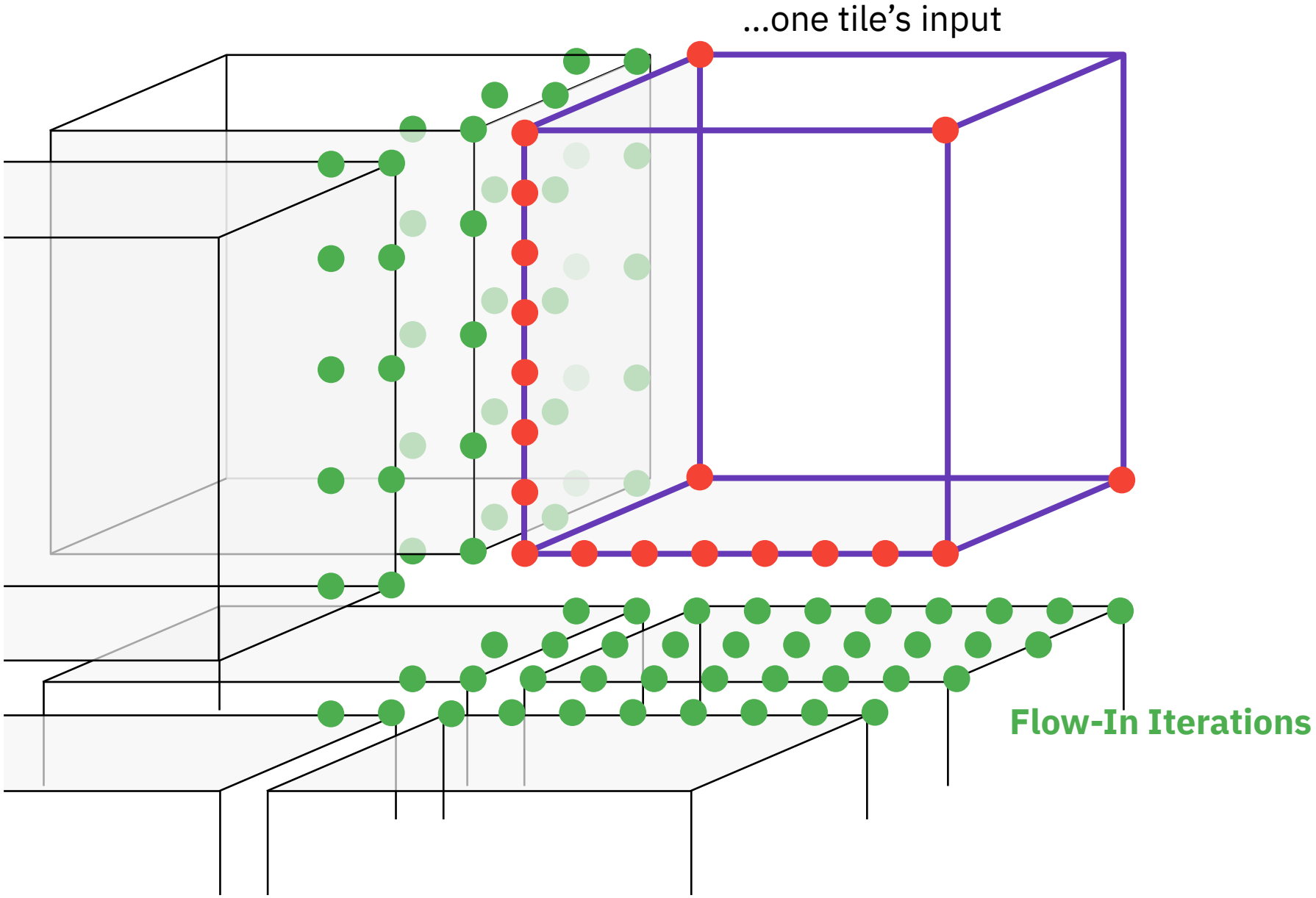
Bondhugula, U. *Compiling Affine Loop Nests for Distributed-Memory Parallel Architectures.* Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, ACM, 2013

Tales of Tiles: the Flow-In Iterations...



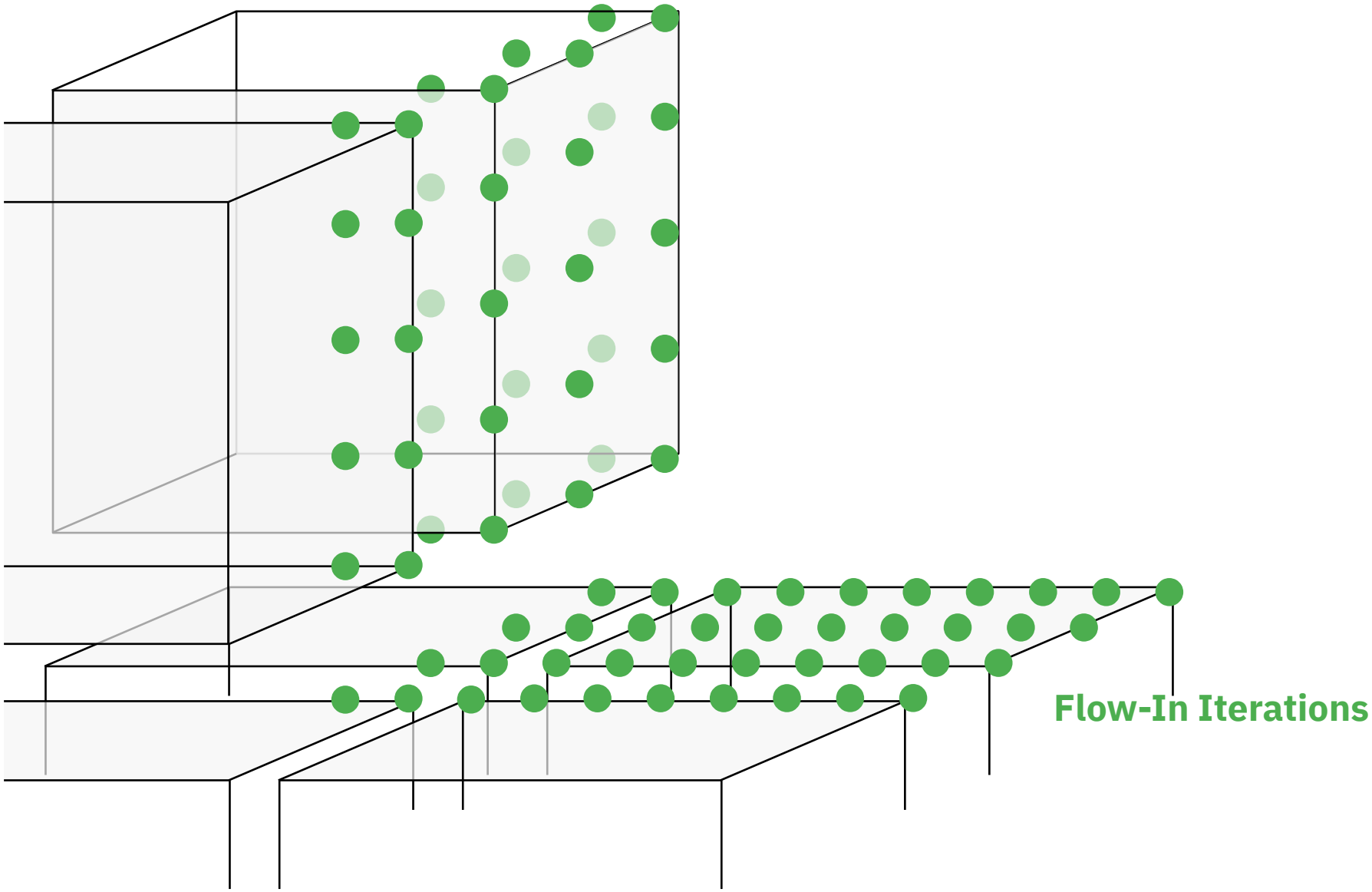
Bondhugula, U. *Compiling Affine Loop Nests for Distributed-Memory Parallel Architectures.* Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, ACM, 2013

Tales of Tiles: the Flow-In Iterations...



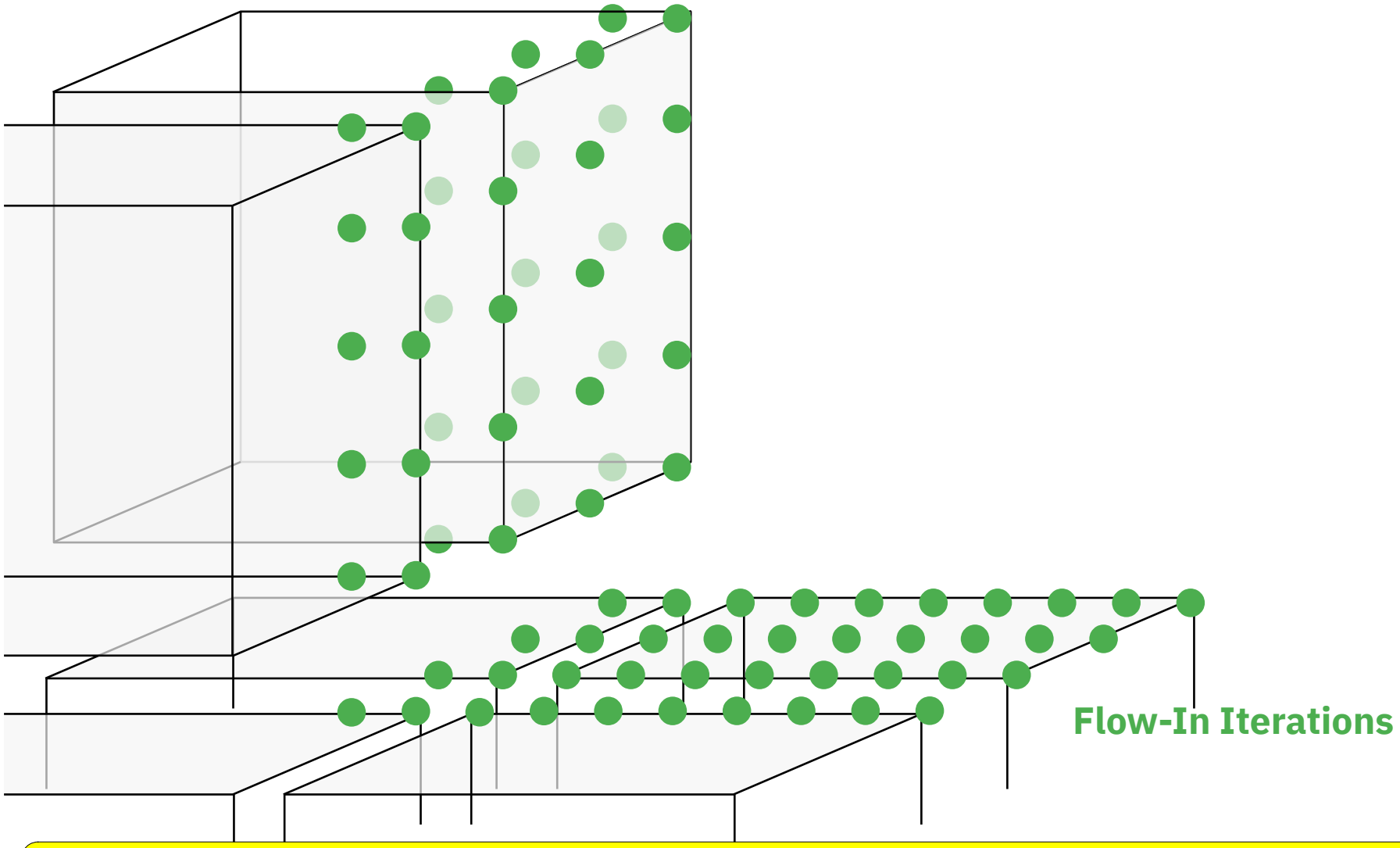
Bondhugula, U. *Compiling Affine Loop Nests for Distributed-Memory Parallel Architectures.* Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, ACM, 2013

Tales of Tiles: the Flow-In Iterations...



Bondhugula, U. *Compiling Affine Loop Nests for Distributed-Memory Parallel Architectures.* Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, ACM, 2013

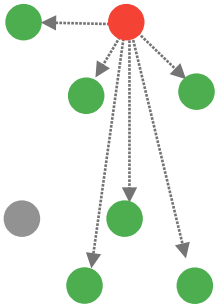
Tales of Tiles: the Flow-In Iterations...

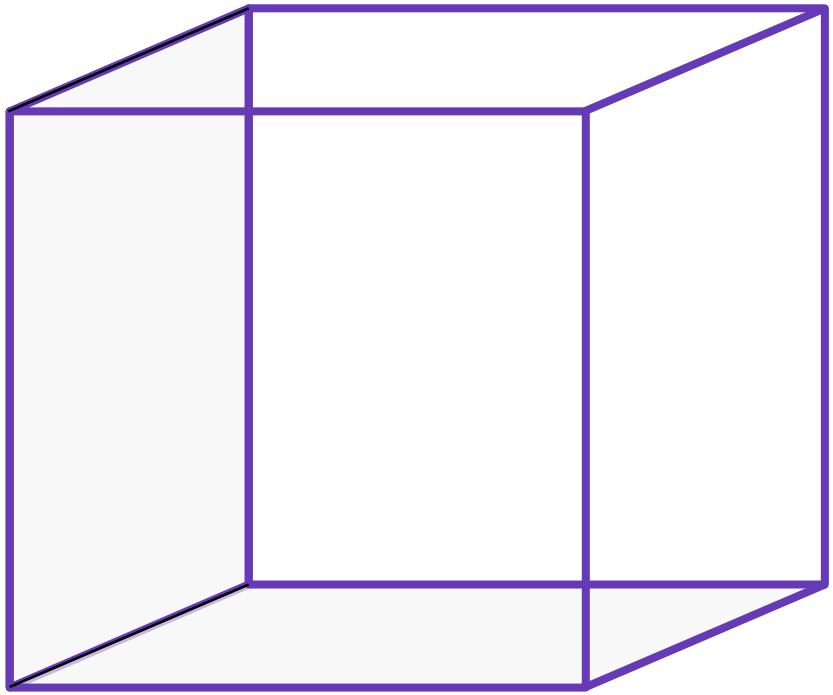


Flow-In Iterations: Data Needed to Execute One Tile

Bondh nance
Computing, Networking, Storage and Analysis, ACM, 2013

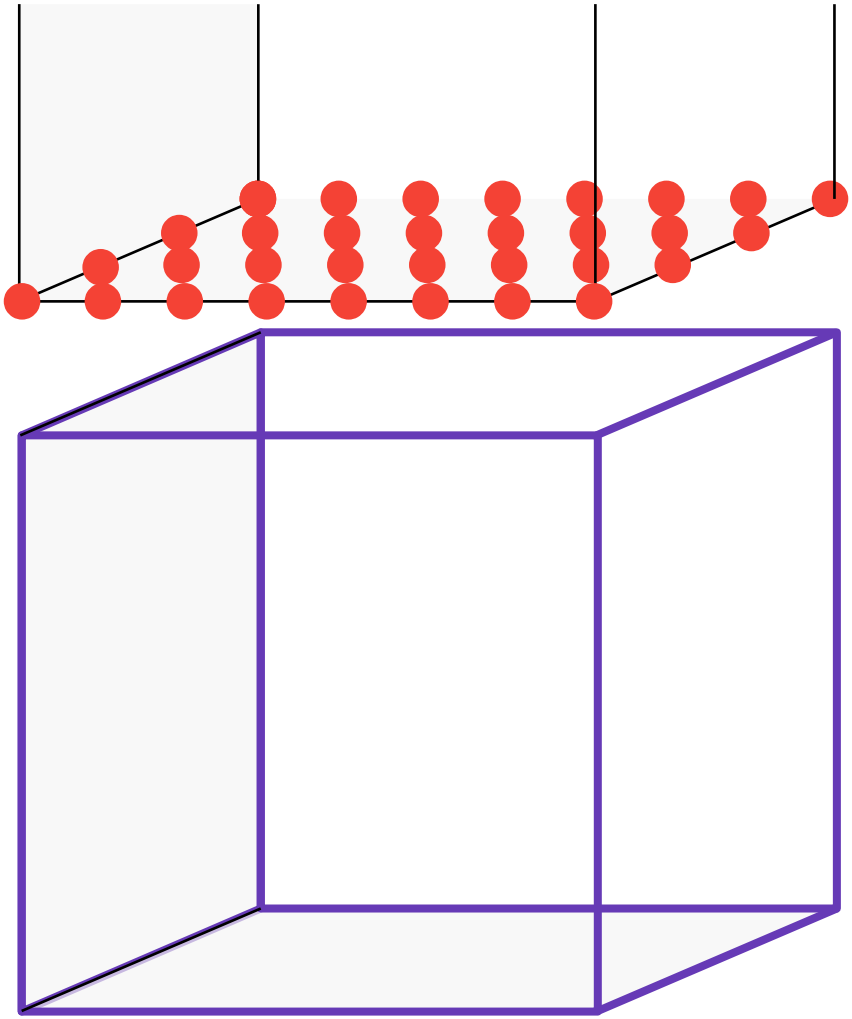
… and the Flow-Out Iterations

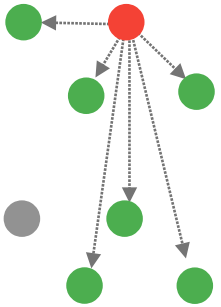




Bondhugula, U. *Compiling Affine Loop Nests for Distributed-Memory Parallel Architectures.* Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, ACM, 2013

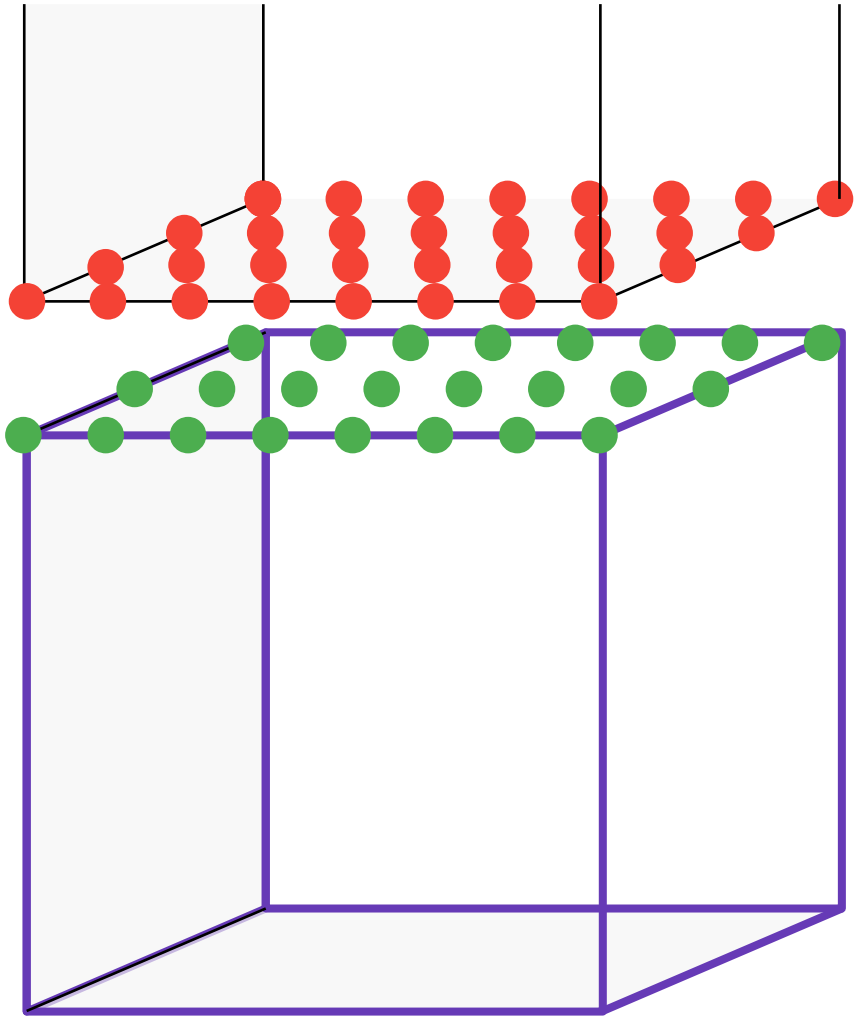
… and the Flow-Out Iterations

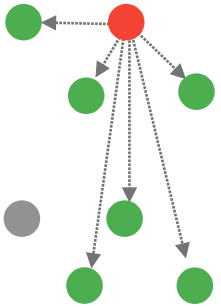




Bondhugula, U. *Compiling Affine Loop Nests for Distributed-Memory Parallel Architectures.* Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, ACM, 2013

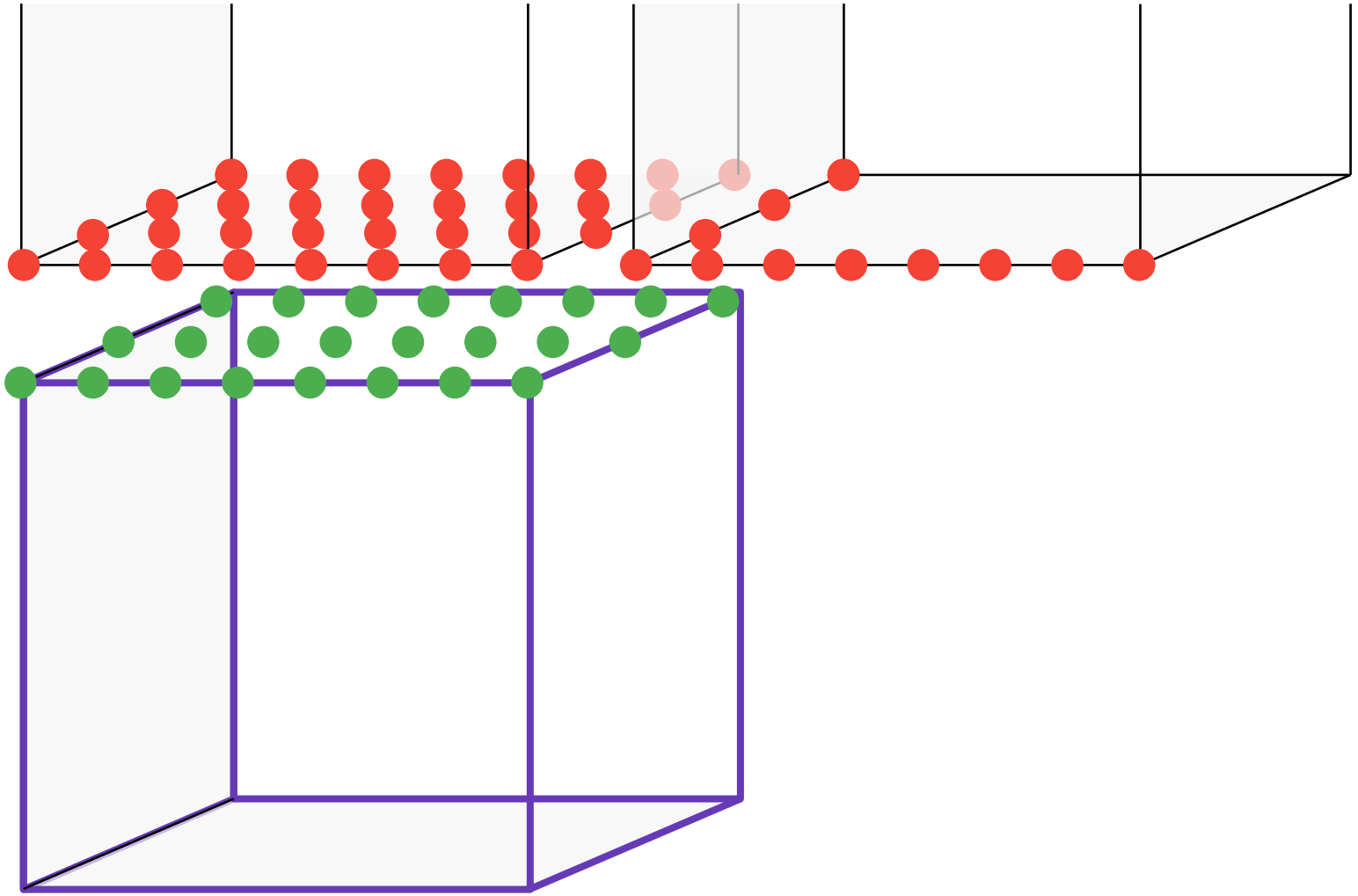
… and the Flow-Out Iterations

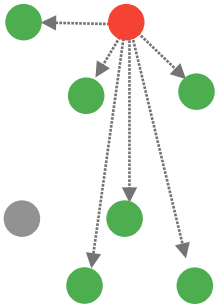




Bondhugula, U. *Compiling Affine Loop Nests for Distributed-Memory Parallel Architectures.* Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, ACM, 2013

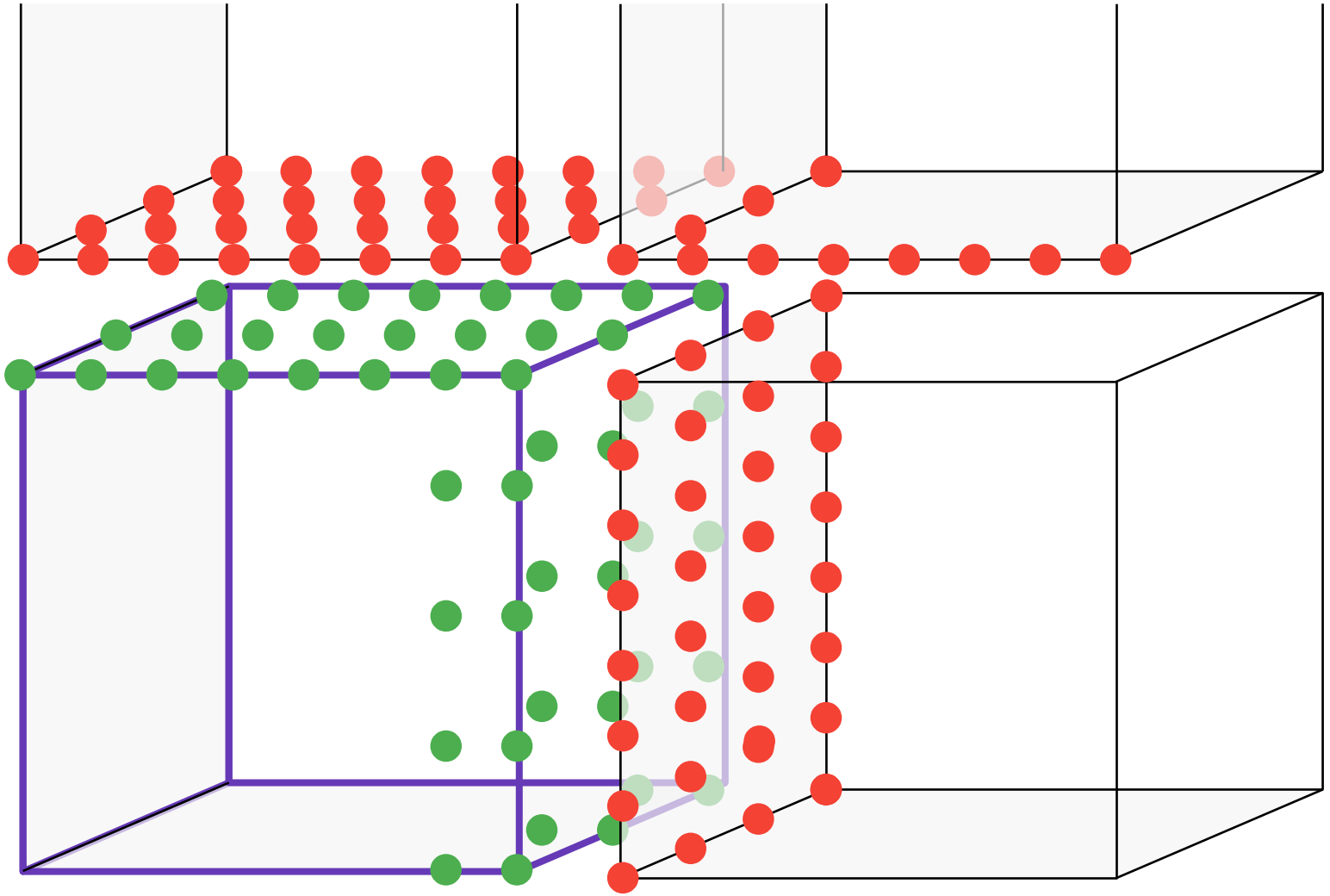
… and the Flow-Out Iterations

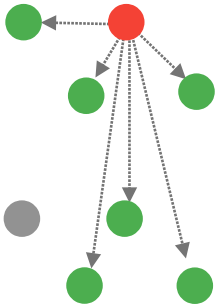




Bondhugula, U. *Compiling Affine Loop Nests for Distributed-Memory Parallel Architectures.* Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, ACM, 2013

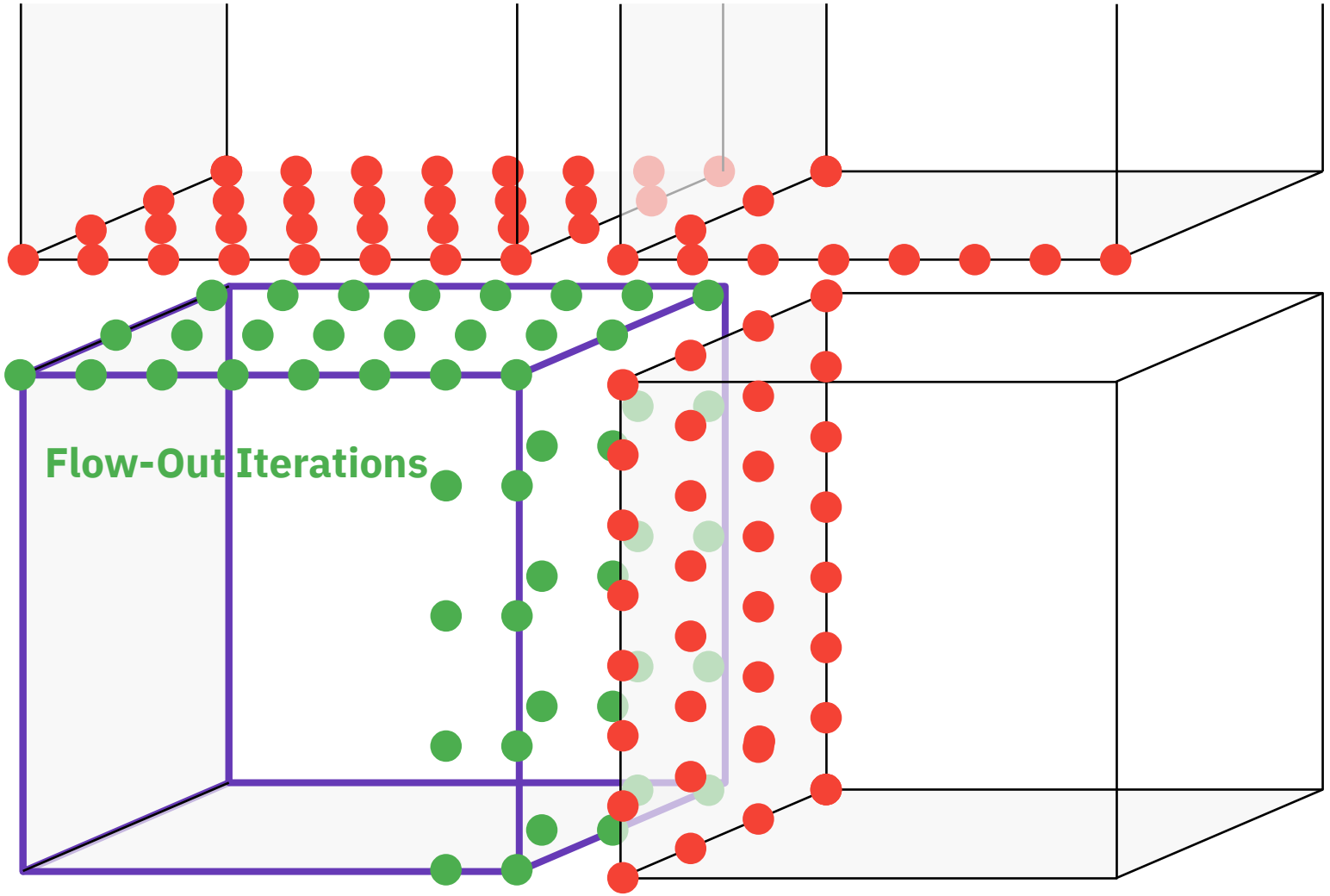
… and the Flow-Out Iterations

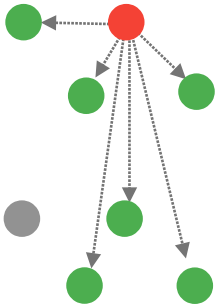




Bondhugula, U. *Compiling Affine Loop Nests for Distributed-Memory Parallel Architectures.* Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, ACM, 2013

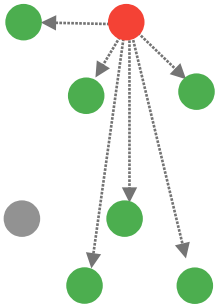
… and the Flow-Out Iterations

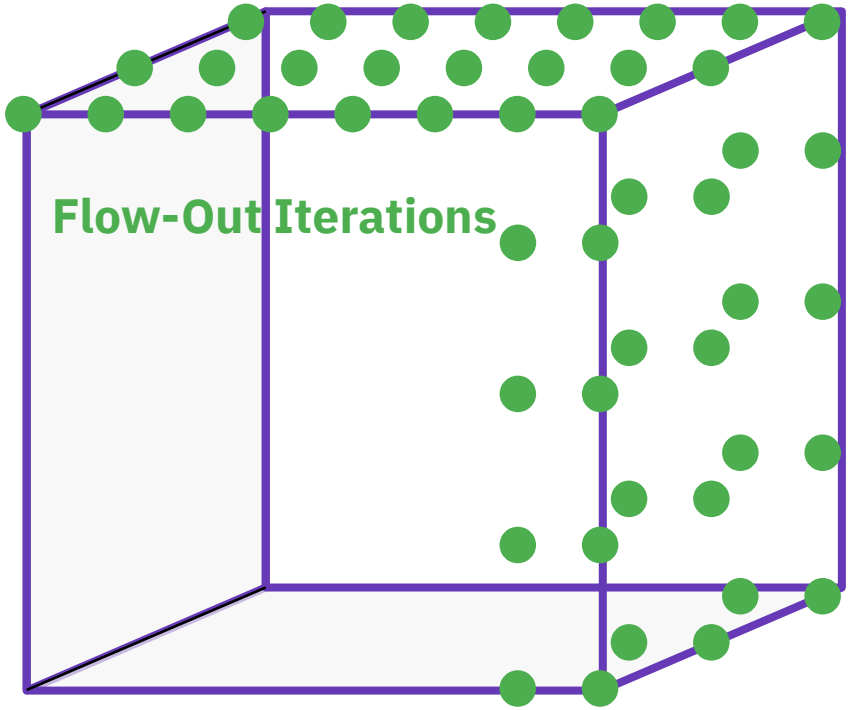




Bondhugula, U. *Compiling Affine Loop Nests for Distributed-Memory Parallel Architectures.* Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, ACM, 2013

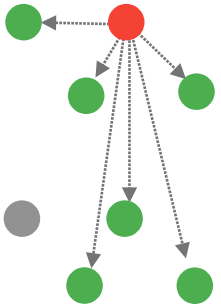
… and the Flow-Out Iterations

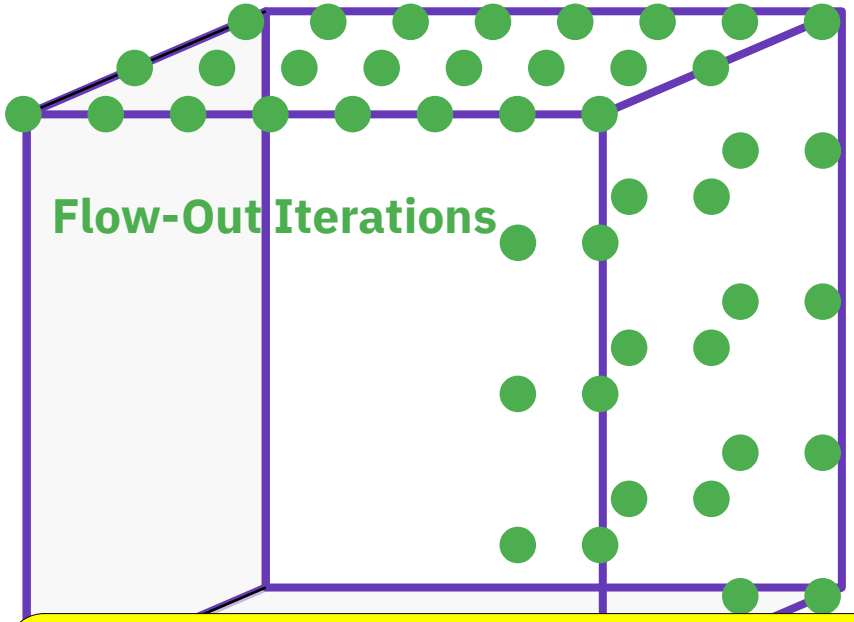




Bondhugula, U. *Compiling Affine Loop Nests for Distributed-Memory Parallel Architectures.* Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, ACM, 2013

… and the Flow-Out Iterations





Flow-Out Iterations: Data Needed To Execute Other Tiles

Most efficient allocation = allocation for flow-in & flow-out!

Bondh… …nance Computing, Networking, Storage and Analysis, ACM, 2013

Contributions of this work

- Create a memory allocation for intermediate results
  - ***Compute data sets produced by each tile to be made contiguous***
  - Allocate memory for each set

Contributions of this work

- Create a memory allocation for intermediate results
  - ***Compute data sets produced by each tile to be made contiguous***
  - Allocate memory for each set
- Increase data contiguity
  - Apply data tiling
  - ***Tweak the memory layout to get contiguity across data tiles***

Contributions of this work

- Create a memory allocation for intermediate results
  - ***Compute data sets produced by each tile to be made contiguous***
  - Allocate memory for each set
- Increase data contiguity
  - Apply data tiling
  - ***Tweak the memory layout to get contiguity across data tiles***
- Mechanize the process
  - ***Create a compiler pass automatically deriving I/O code***
  - Integrate the I/O code into an FPGA data-flow architecture

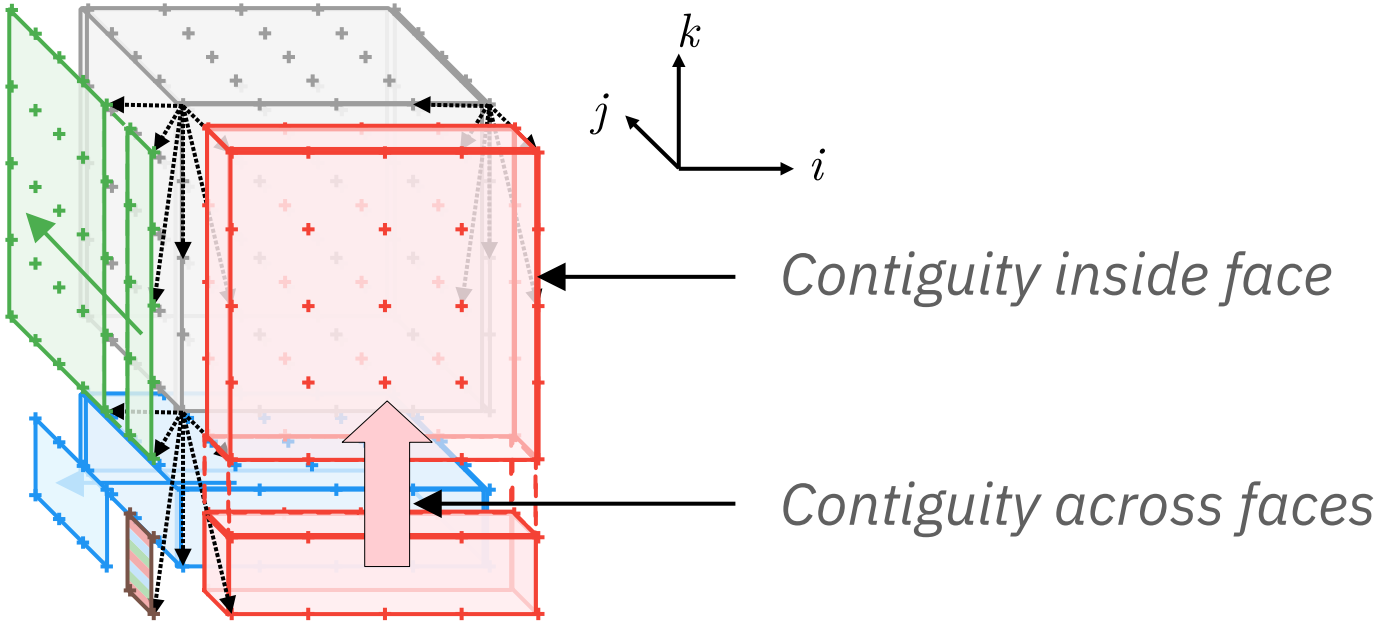
Contributions of this work

- Create a memory allocation for intermediate results
  - ***Compute data sets produced by each tile to be made contiguous***
  - Allocate memory for each set
- Increase data contiguity
  - Apply data tiling
  - ***Tweak the memory layout to get contiguity across data tiles***
- Mechanize the process
  - ***Create a compiler pass automatically deriving I/O code***
  - Integrate the I/O code into an FPGA data-flow architecture

Ferry, C.; Yuki, T.; Derrien, S. & Rajopadhye, S. *Increasing FPGA Accelerators Memory Bandwidth with a Burst-Friendly Memory Layout.* **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, Institute of Electrical and Electronics Engineers (IEEE), 2022, 1-1

Canonical Facet Allocation (CFA)

- Extension of Deest et al. [Deest FPL 2017, Deest PhD 2017]
  - Observation: flow-in data is made mostly of adjacent faces
  - Idea : Allocate contiguous memory for the flow-in/flow-out faces of each tile
- Our idea : tweak the data layout



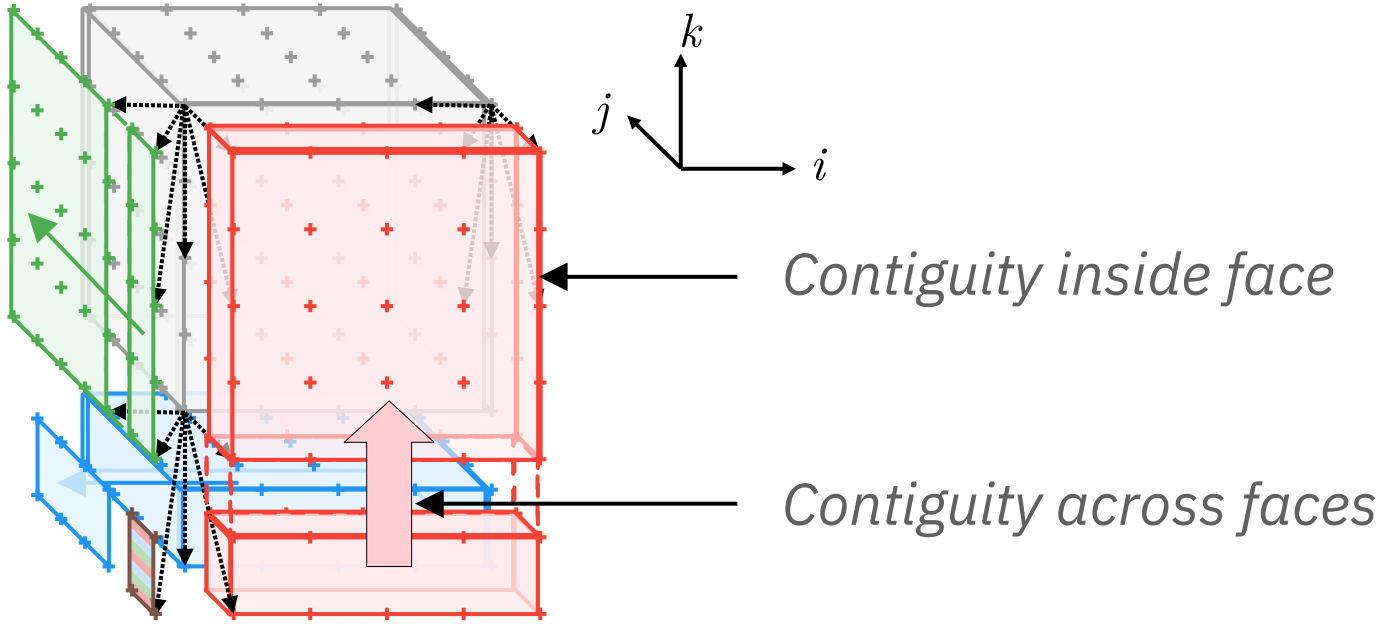
[Deest FPL 2017] Deest, G.; Yuki, T.; Rajopadhye, S. & and Derrien, S. "One size does not fit all: Implementation trade-offs for iterative stencil computations on FPGAs," 2017 27th International Conference on Field Programmable Logic and Applications (FPL), Ghent, Belgium, 2017, pp. 1-8, doi: 10.23919/FPL.2017.8056781.

[Deest PhD] Deest, G. “Implementation Trade-Offs for FPGA Accelerators.” PhD Thesis, Université de Rennes 1, 2017

Canonical Facet Allocation (CFA)

- Extension of Deest et al. [Deest FPL 2017, Deest PhD 2017]
  - Observation: flow-in data is made mostly of adjacent faces
  - Idea : Allocate contiguous memory for the flow-in/flow-out faces of each tile
- Our idea : tweak the data layout

Contiguity inside a face and across neighboring faces



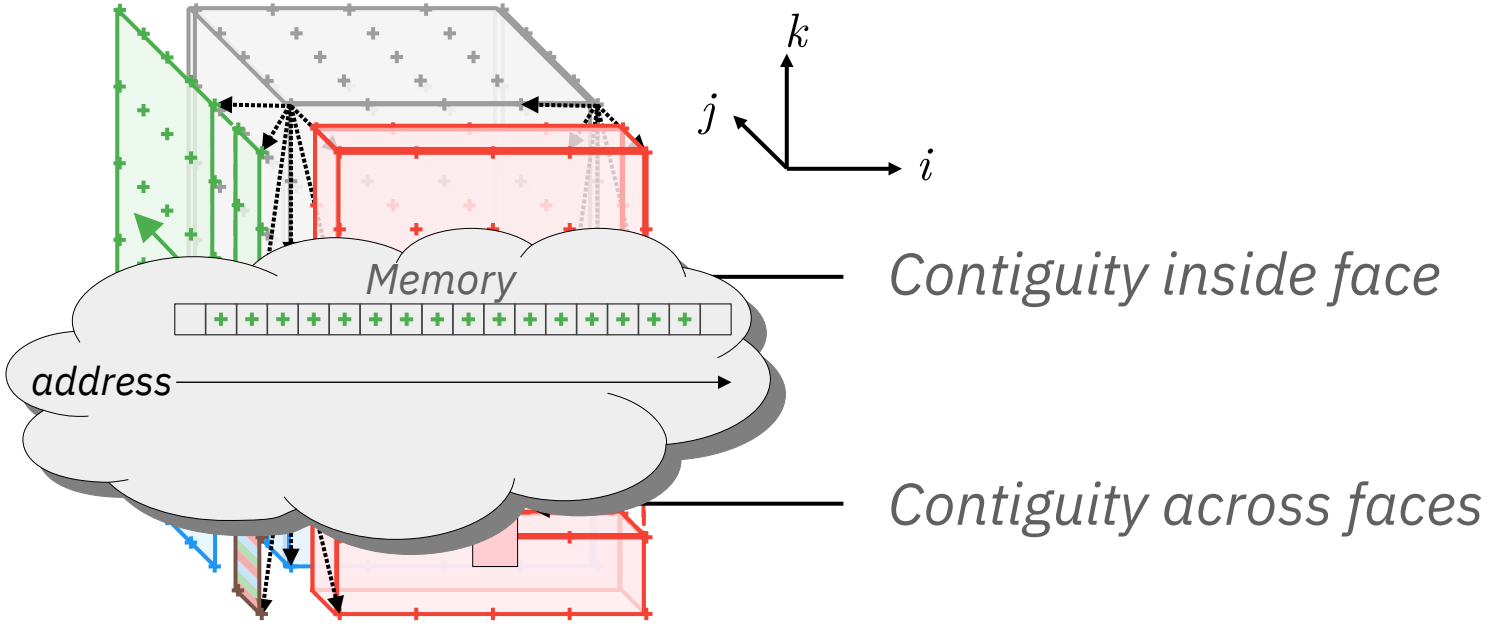
[Deest FPL 2017] Deest, G.; Yuki, T.; Rajopadhye, S. & and Derrien, S. "One size does not fit all: Implementation trade-offs for iterative stencil computations on FPGAs," 2017 27th International Conference on Field Programmable Logic and Applications (FPL), Ghent, Belgium, 2017, pp. 1-8, doi: 10.23919/FPL.2017.8056781.

[Deest PhD] Deest, G. “Implementation Trade-Offs for FPGA Accelerators.” PhD Thesis, Université de Rennes 1, 2017

Canonical Facet Allocation (CFA)

- Extension of Deest et al. [Deest FPL 2017, Deest PhD 2017]
  - Observation: flow-in data is made mostly of adjacent faces
  - Idea : Allocate contiguous memory for the flow-in/flow-out faces of each tile
- Our idea : tweak the data layout

Contiguity inside a face and across neighboring faces

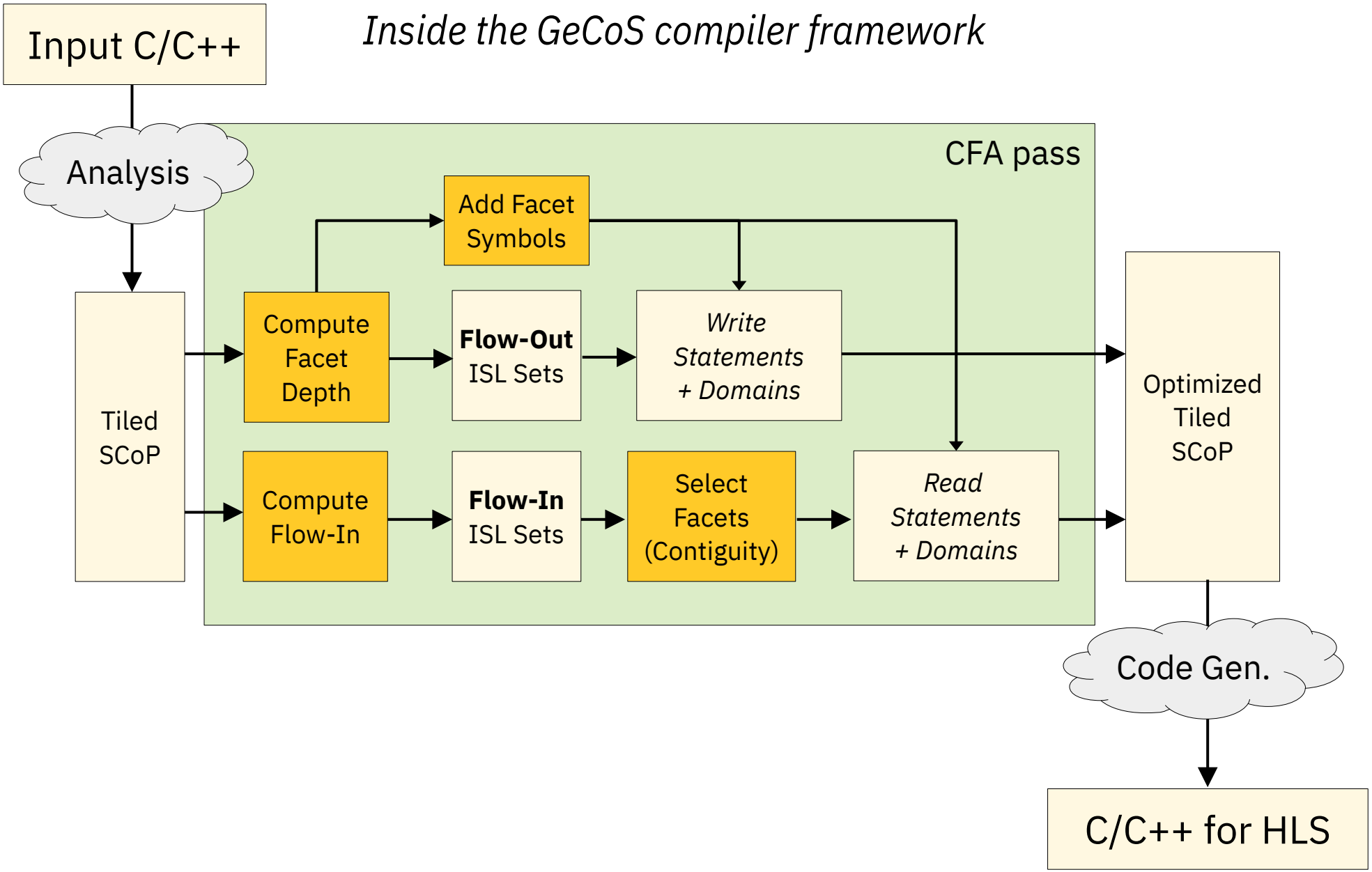


[Deest FPL 2017] Deest, G.; Yuki, T.; Rajopadhye, S. & and Derrien, S. "One size does not fit all: Implementation trade-offs for iterative stencil computations on FPGAs," 2017 27th International Conference on Field Programmable Logic and Applications (FPL), Ghent, Belgium, 2017, pp. 1-8, doi: 10.23919/FPL.2017.8056781.

[Deest PhD] Deest, G. “Implementation Trade-Offs for FPGA Accelerators.” PhD Thesis, Université de Rennes 1, 2017

Proof-of-Concept CFA Compiler Pass

Inside the GeCoS compiler framework



Validation on FPGA

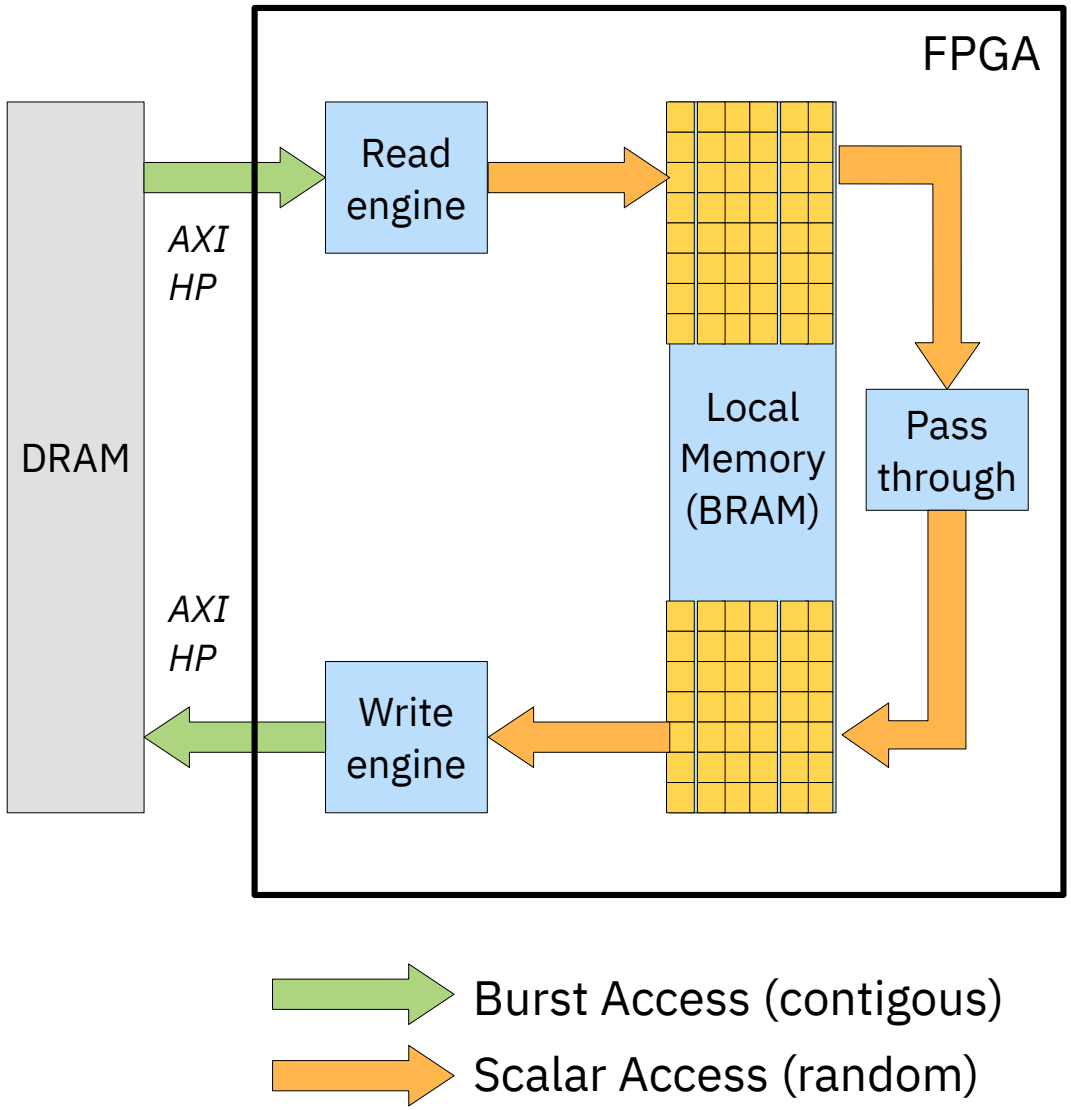
Does CFA yield a higher bandwidth utilization ?

- **Protocol:**
  - Execute I/O on FPGA
  - No execution engine
- CFA vs. 3 allocations:
  - Bounding box
  - Original (“UOV-like”)
  - Data Tiling
- Platform : Xilinx ZC706

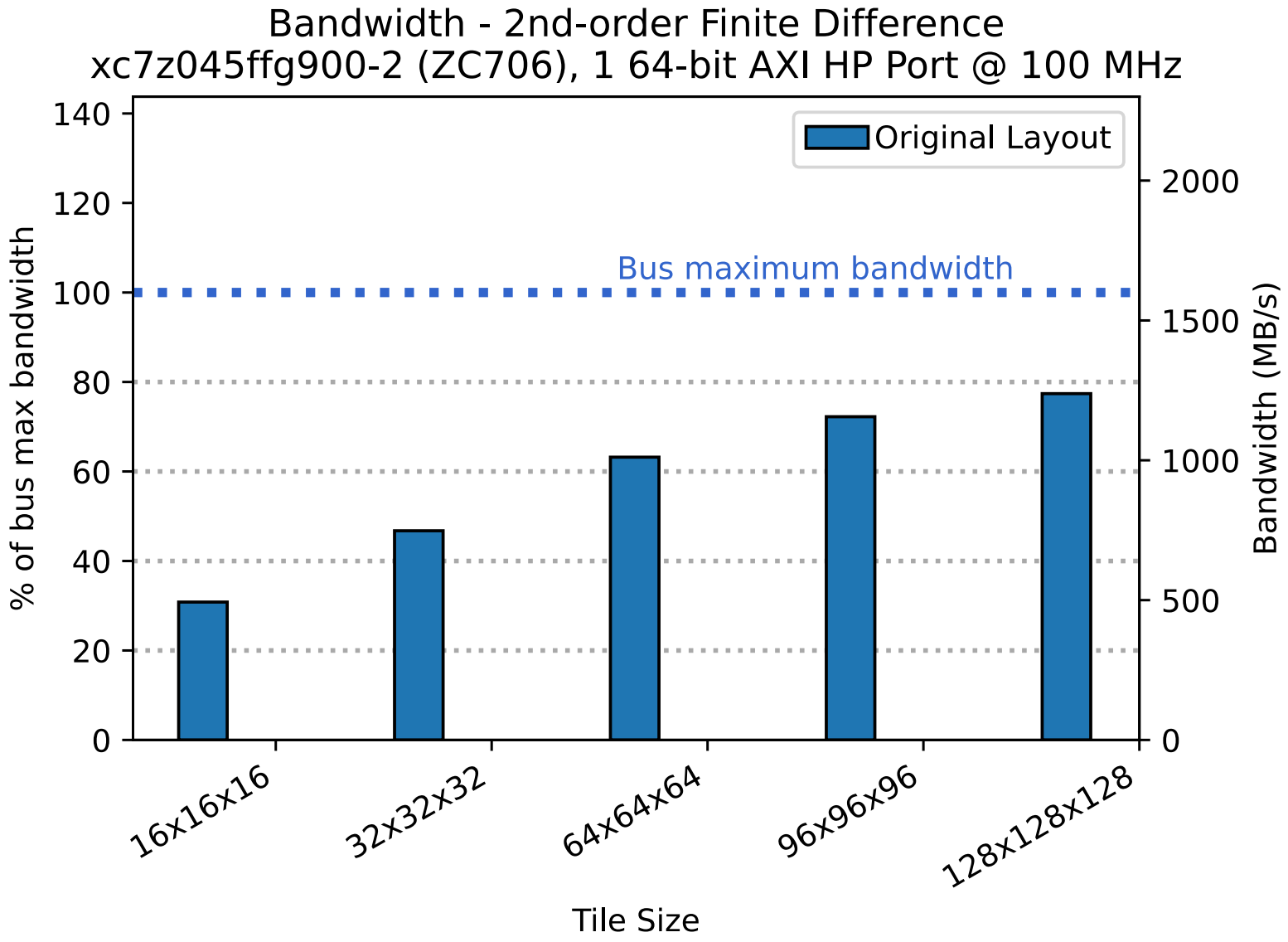
  Xilinx xc7z045ffg900-2 FPGA

  Frequency 100 MHz

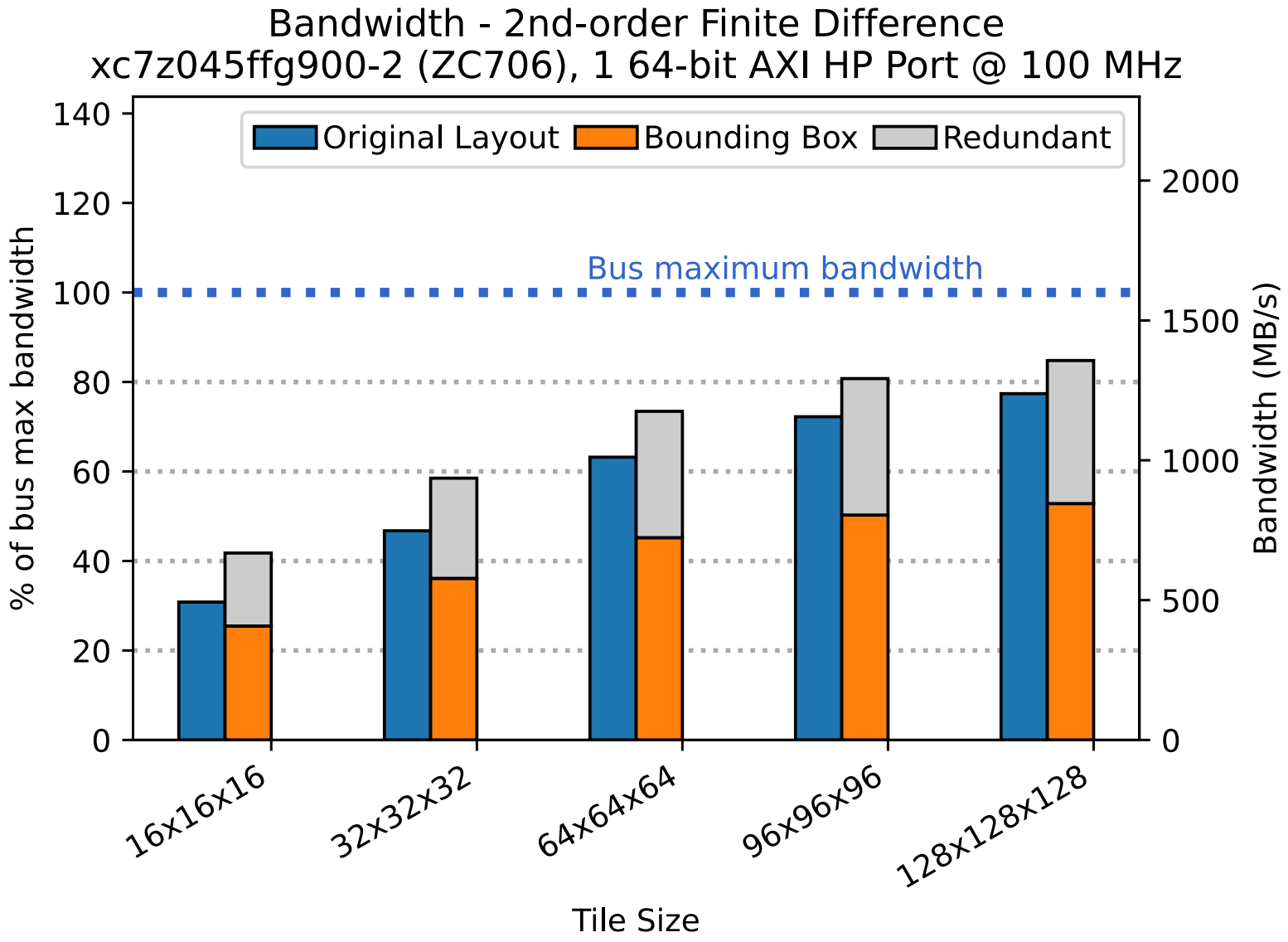
  1 access port, full-duplex 64-bit AXI



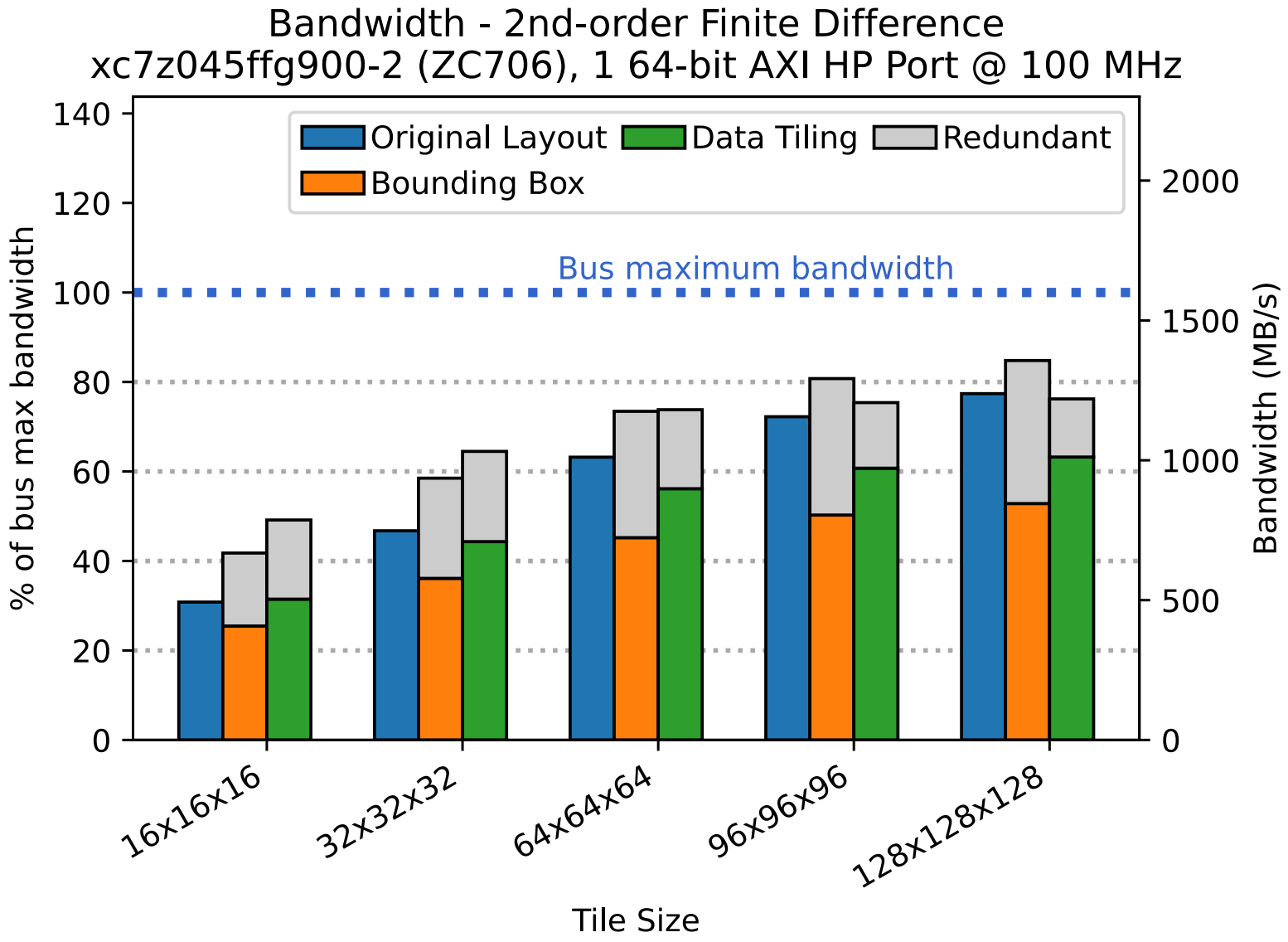
CFA → Higher Bandwidth Utilization



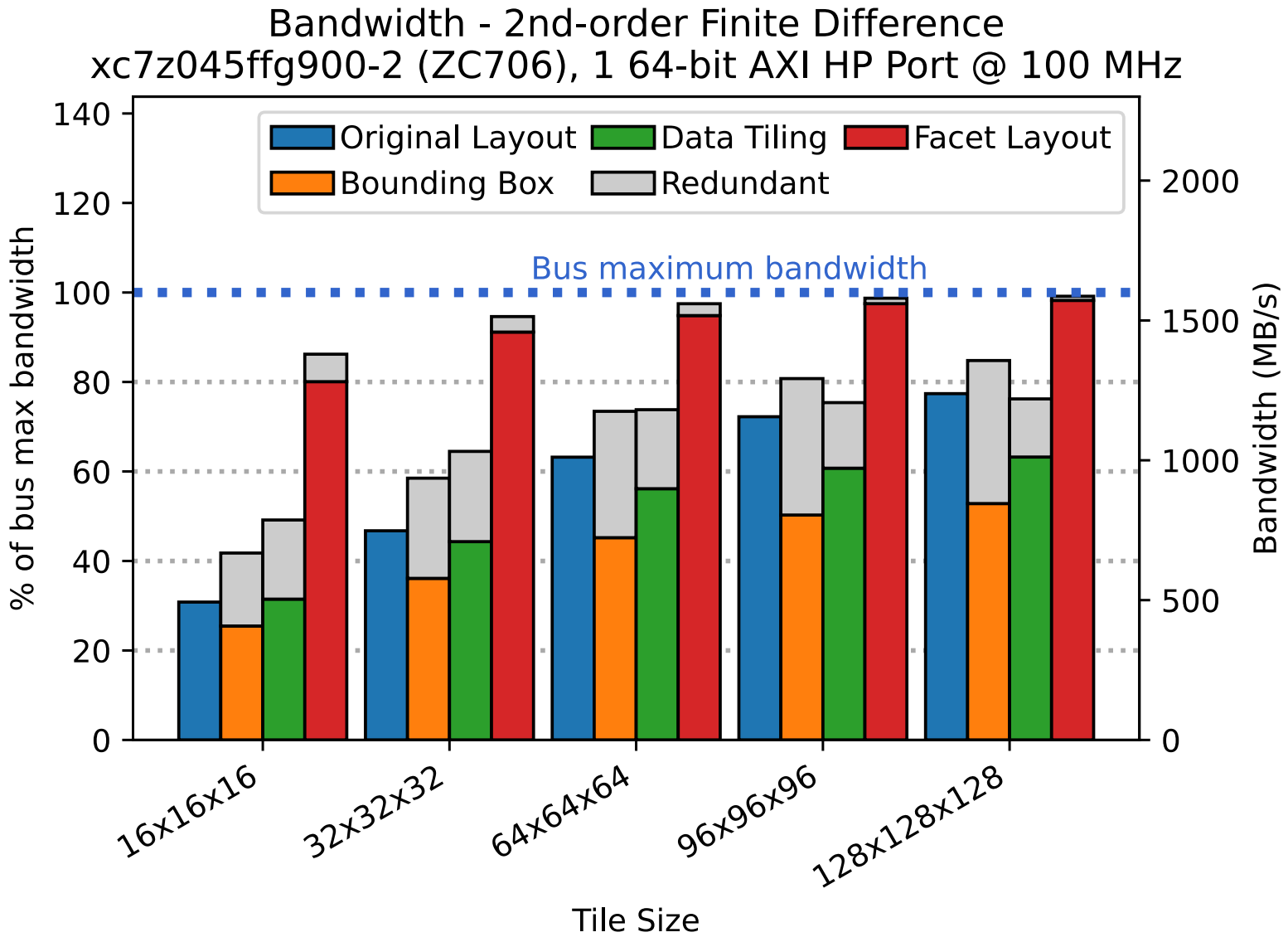
CFA → Higher Bandwidth Utilization



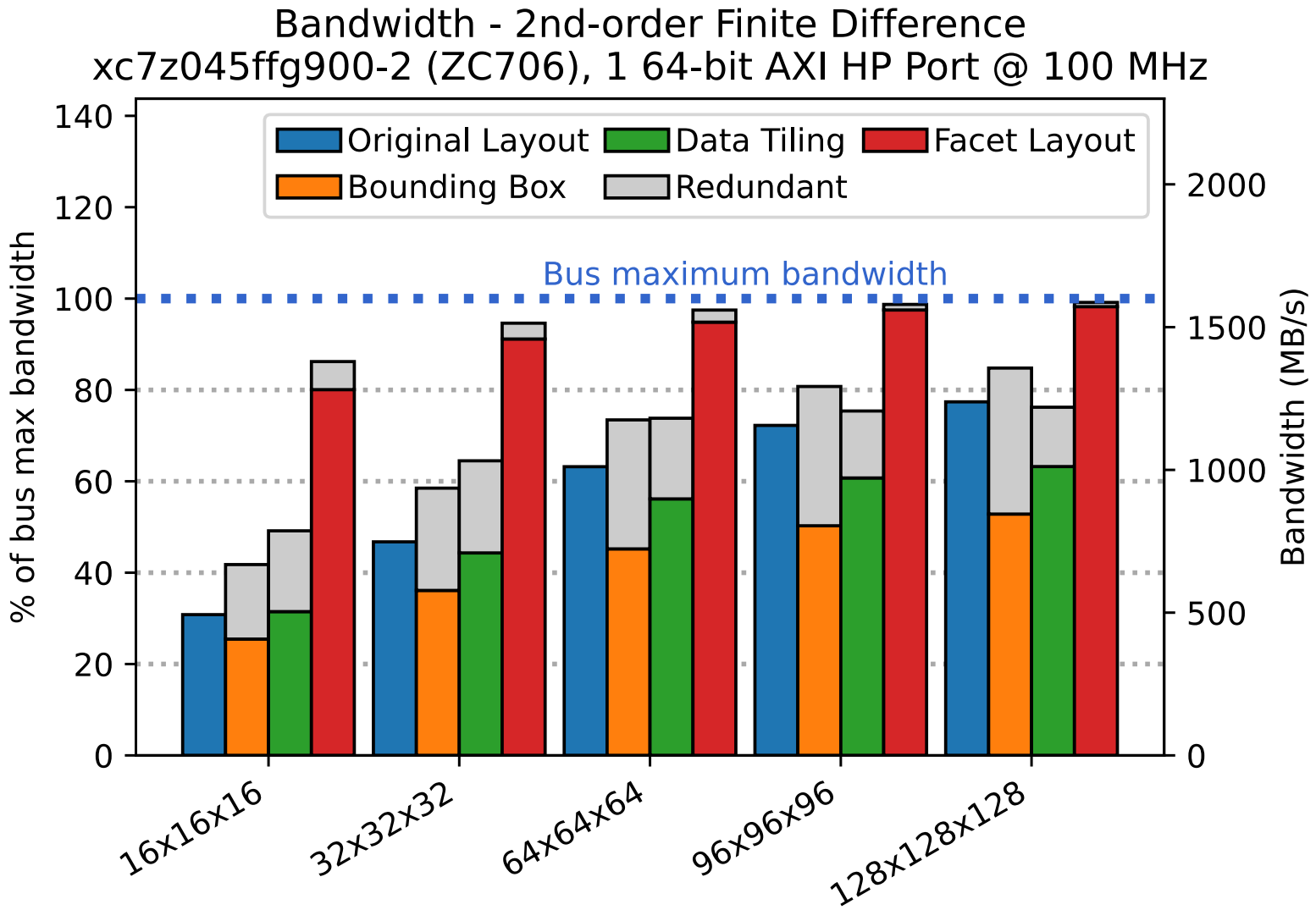
CFA → Higher Bandwidth Utilization



CFA → Higher Bandwidth Utilization



CFA → Higher Bandwidth Utilization



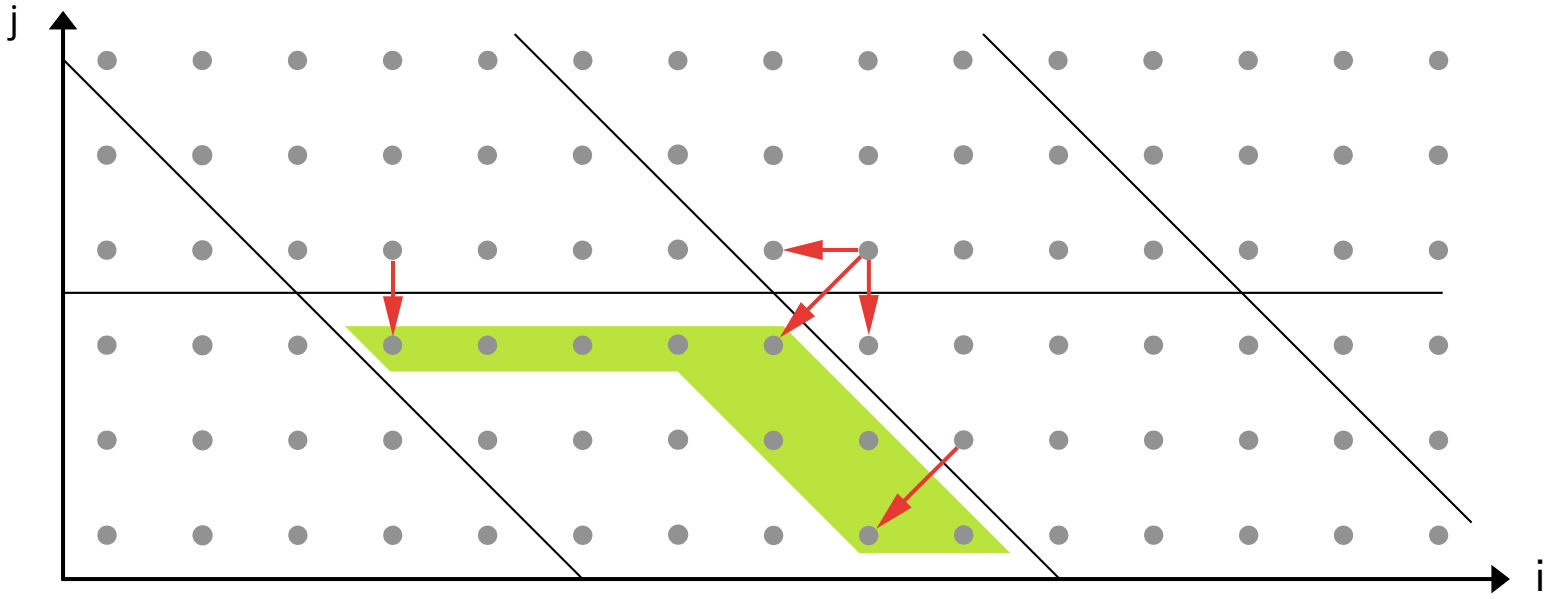
CFA brings contiguity & low redundancy → high *effective* bandwidth

Outline

- Introduction and Motivation
- Background
- Proposed Solutions
  - Contiguous Allocation for Rectangular Tiles
  - **Partitioning the Data Flow of Programs with Uniform Dependences**
  - **Contiguous, Compressed Memory Allocation for Uniform Dependence Programs**
  - Partitioning the Data Flow of Programs with Affine Dependences
- Conclusions

Inter-tile communications : flow-in/out

- Iterations consumed in another tile = **Flow-out**
- Bondhugula (2013) [2] : communicated sets = ***flow-in / flow-out* sets**
- **Only a part of flow-out is needed by every consumer tile**

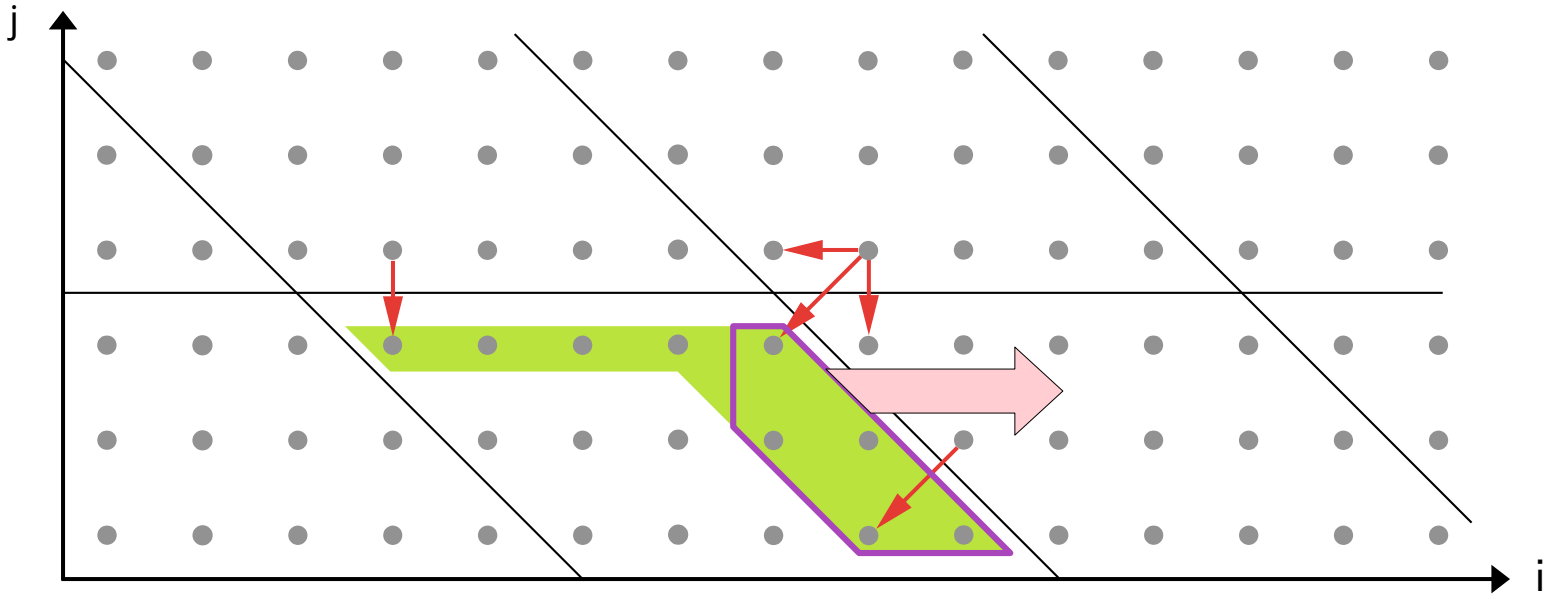


Flow-out set of a tile of iterations with a Smith-Waterman kernel

[2] Bondhugula, U. *Compiling Affine Loop Nests for Distributed-Memory Parallel Architectures.* Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, ACM, 2013

Inter-tile communications : flow-in/out

- Iterations consumed in another tile = **Flow-out**
- Bondhugula (2013) [2] : communicated sets = ***flow-in / flow-out* sets**
- **Only a part of flow-out is needed by every consumer tile**

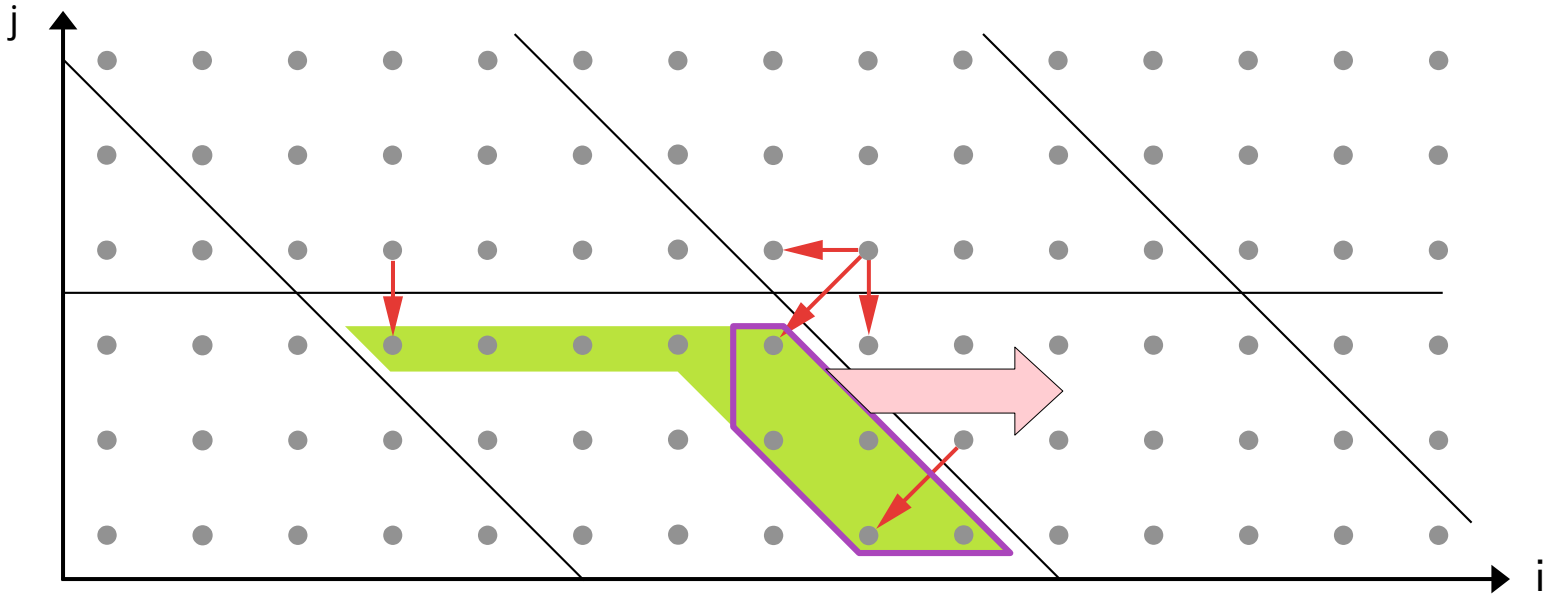


Flow-out set of a tile of iterations with a Smith-Waterman kernel

[2] Bondhugula, U. *Compiling Affine Loop Nests for Distributed-Memory Parallel Architectures.* Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, ACM, 2013

Inter-tile communications : flow-in/out

- Iterations consumed in another tile = **Flow-out**
- Bondhugula (2013) [2] : communicated sets = ***flow-in / flow-out* sets**
- **Only a part of flow-out is needed by every consumer tile**



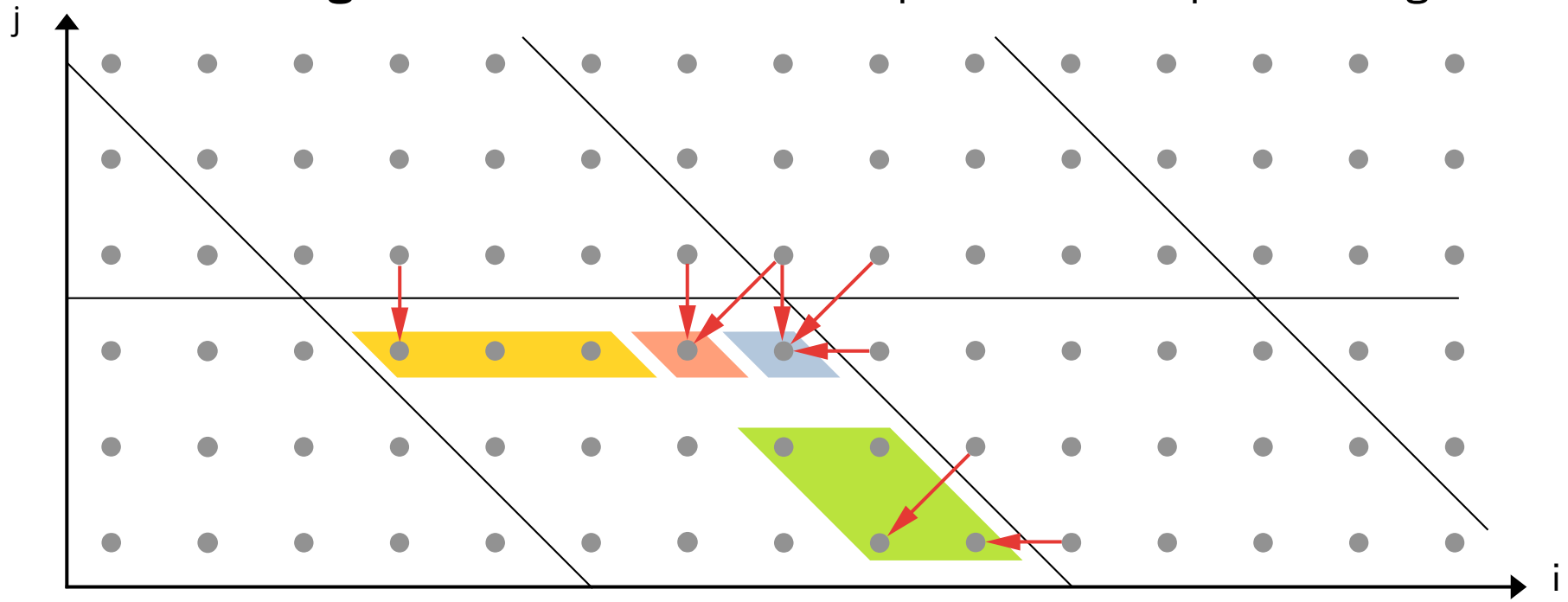
Flow-out set of a tile of iterations with a Smith-Waterman kernel

Our Idea: Split the flow-out based on consumption

[2] Bondhugula, U. *Compiling Affine Loop Nests for Distributed-Memory Parallel Architectures.* Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, ACM, 2013

Maximal Atomic irRedundant Sets (MARS)

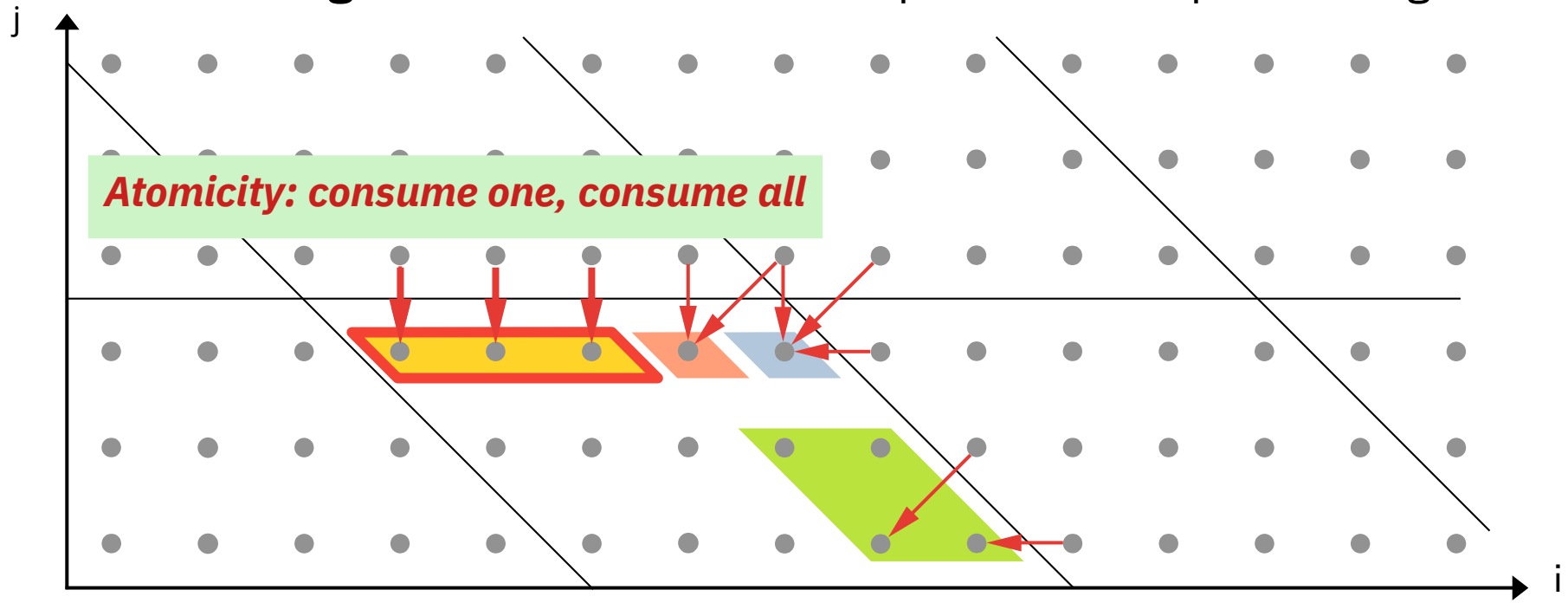
- **Partition the flow-in and flow-out of iteration tiles**
  - MARS = result of the partition
- Provide an **algorithm** and mathematical proofs for the partitioning



Ferry, C.; Derrien, S. & Rajopadhye, S. *Maximal Atomic irRedundant Sets: a Usage-based Dataflow Partitioning Algorithm.* **13th International Workshop on Polyhedral Compilation Techniques** (IMPACT'23), 2023

Maximal Atomic irRedundant Sets (MARS)

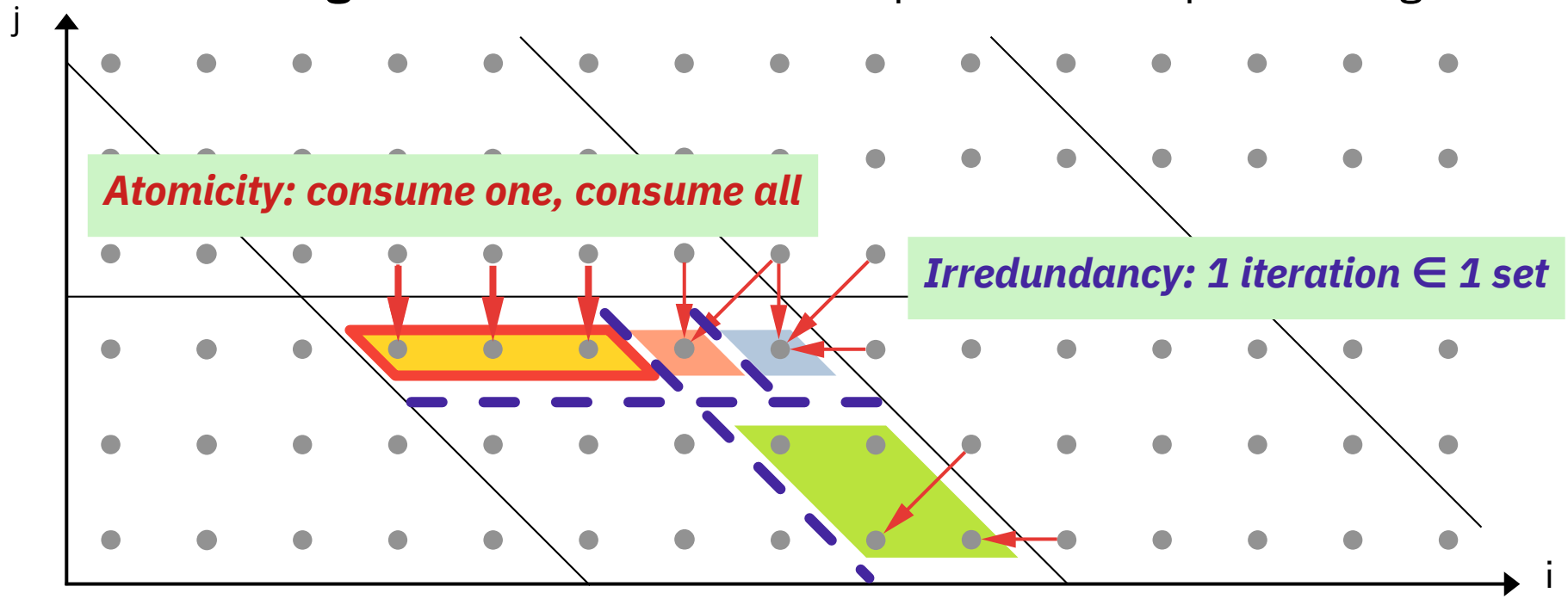
- **Partition the flow-in and flow-out of iteration tiles**
  - MARS = result of the partition
- Provide an **algorithm** and mathematical proofs for the partitioning



Ferry, C.; Derrien, S. & Rajopadhye, S. *Maximal Atomic irRedundant Sets: a Usage-based Dataflow Partitioning Algorithm.* **13th International Workshop on Polyhedral Compilation Techniques** (IMPACT'23), 2023

Maximal Atomic irRedundant Sets (MARS)

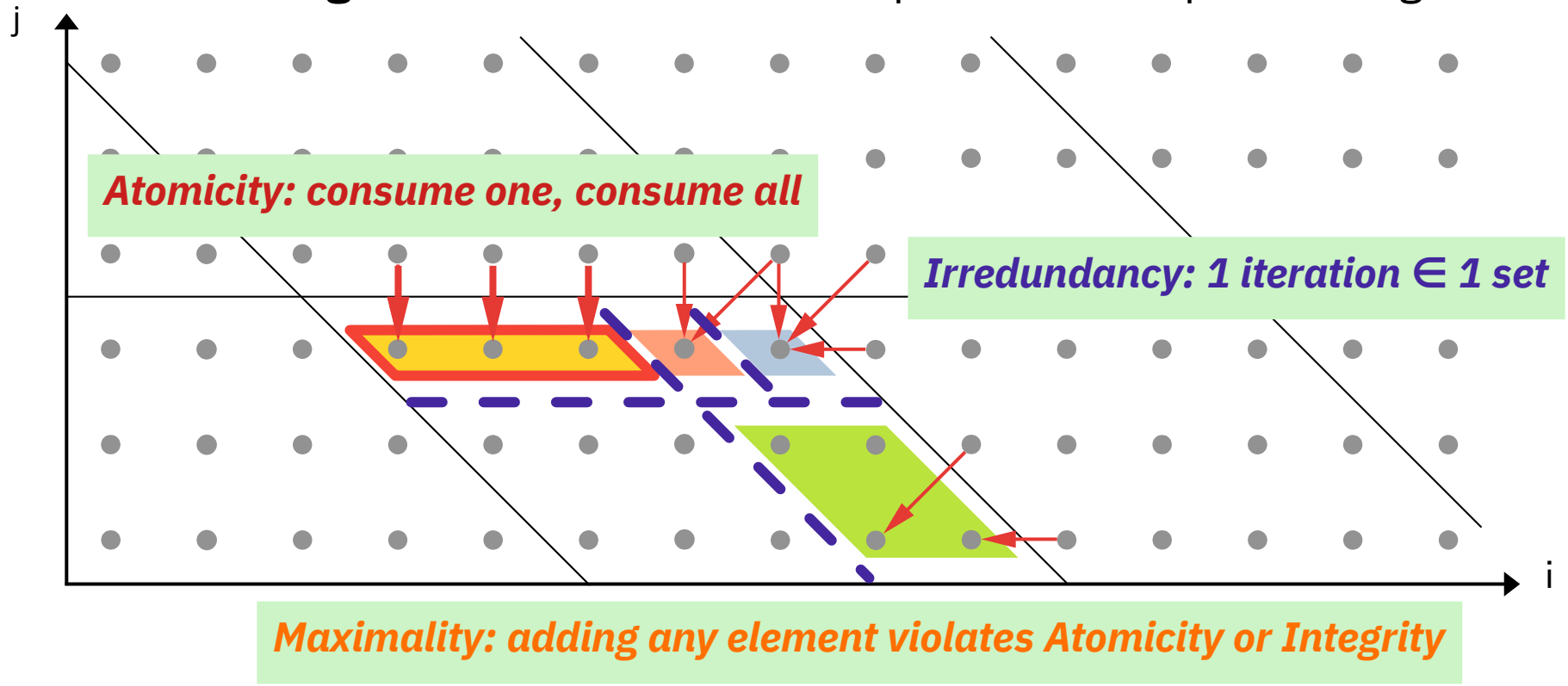
- **Partition the flow-in and flow-out of iteration tiles**
  - MARS = result of the partition
- Provide an **algorithm** and mathematical proofs for the partitioning



Ferry, C.; Derrien, S. & Rajopadhye, S. *Maximal Atomic irRedundant Sets: a Usage-based Dataflow Partitioning Algorithm.* **13th International Workshop on Polyhedral Compilation Techniques** (IMPACT'23), 2023

Maximal Atomic irRedundant Sets (MARS)

- **Partition the flow-in and flow-out of iteration tiles**
  - MARS = result of the partition
- Provide an **algorithm** and mathematical proofs for the partitioning

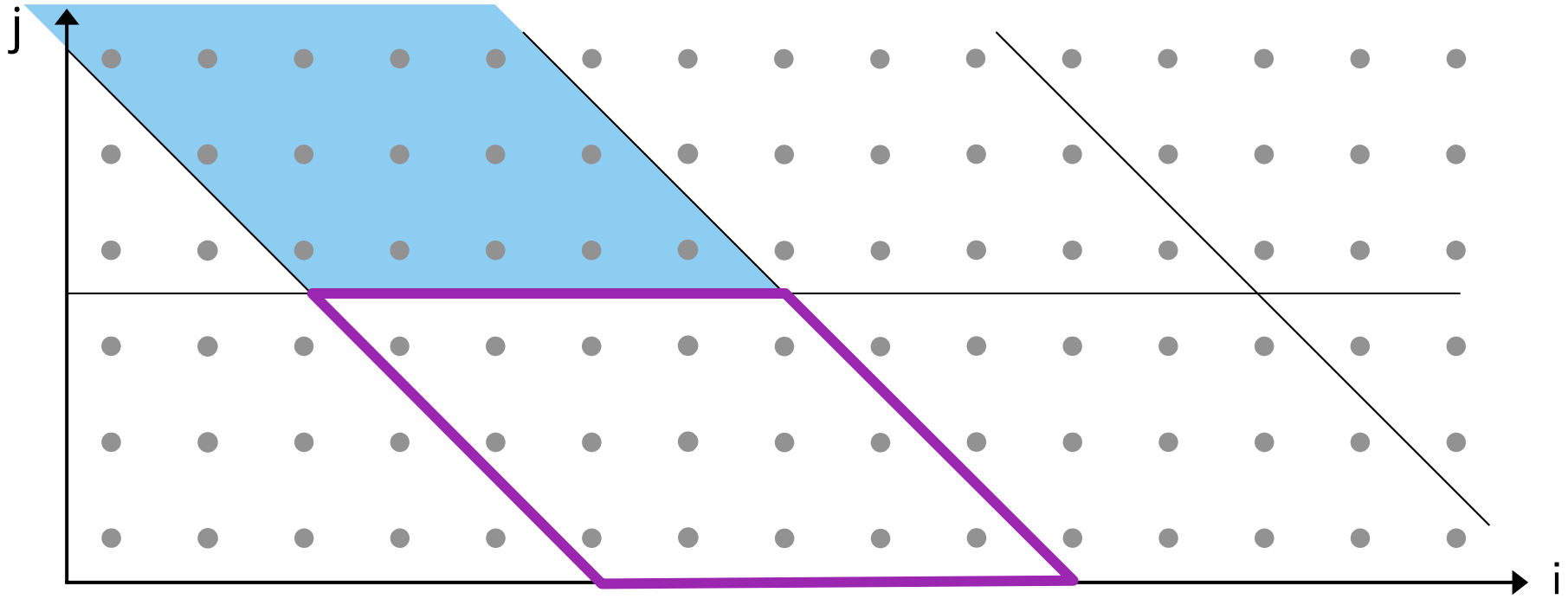


Ferry, C.; Derrien, S. & Rajopadhye, S. *Maximal Atomic irRedundant Sets: a Usage-based Dataflow Partitioning Algorithm.* **13th International Workshop on Polyhedral Compilation Techniques** (IMPACT'23), 2023

How to construct the MARS?

- Sought: Iterations consumed **exclusively** by specific consumer tiles
- Constructed by iterating over **tuples of consumers**

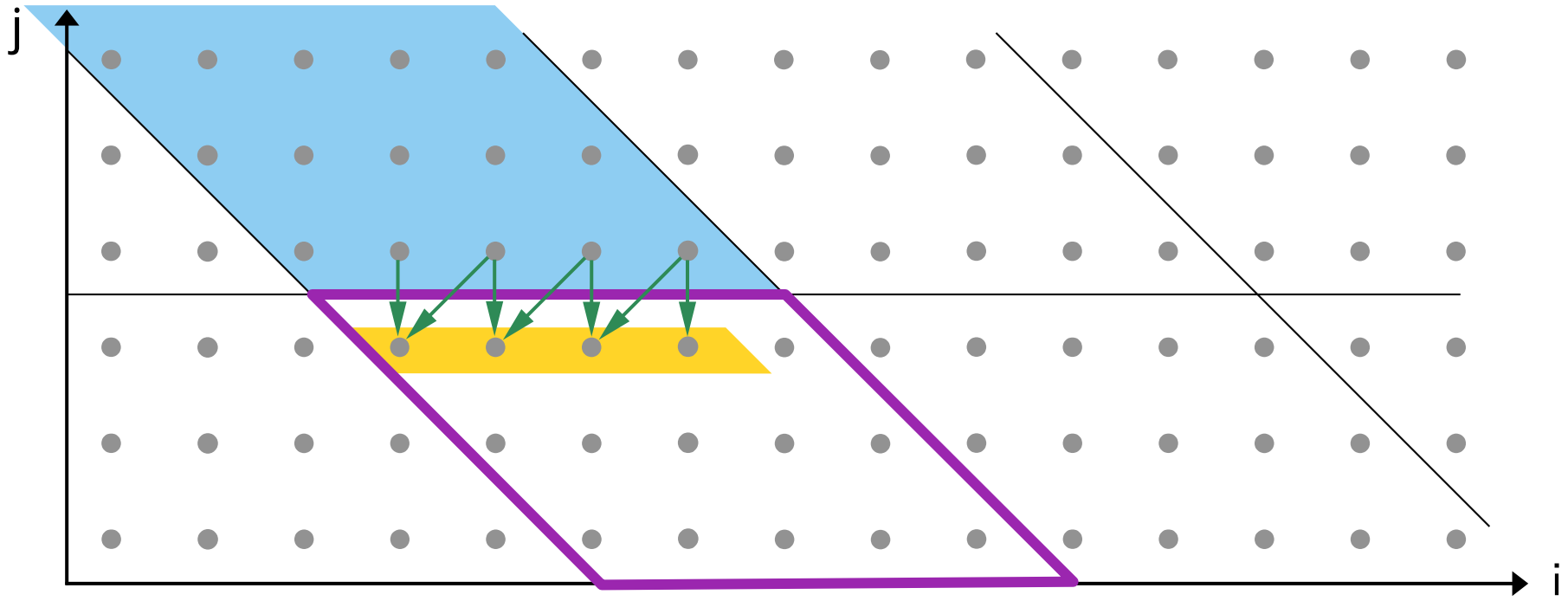
Iterations consumed by tuple less Iterations consumed by other tiles



How to construct the MARS?

- Sought: Iterations consumed **exclusively** by specific consumer tiles
- Constructed by iterating over **tuples of consumers**

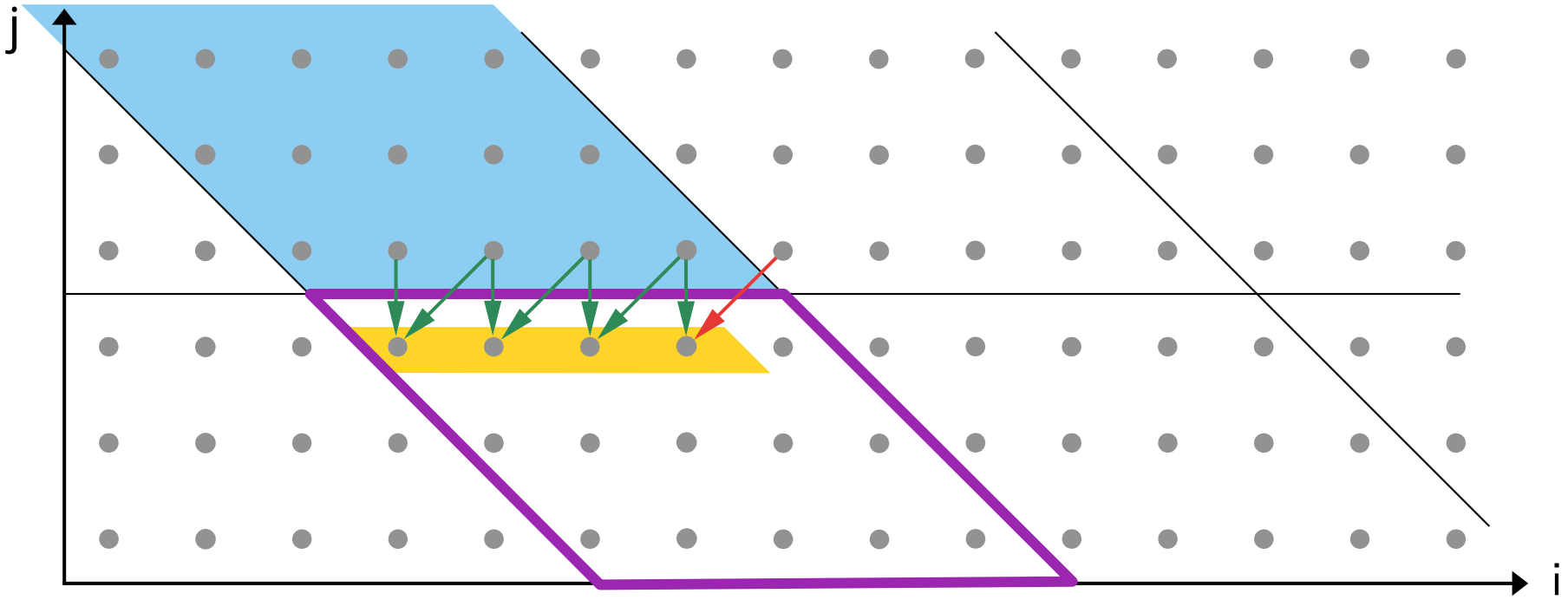
Iterations consumed by tuple less Iterations consumed by other tiles



How to construct the MARS?

- Sought: Iterations consumed **exclusively** by specific consumer tiles
- Constructed by iterating over **tuples of consumers**

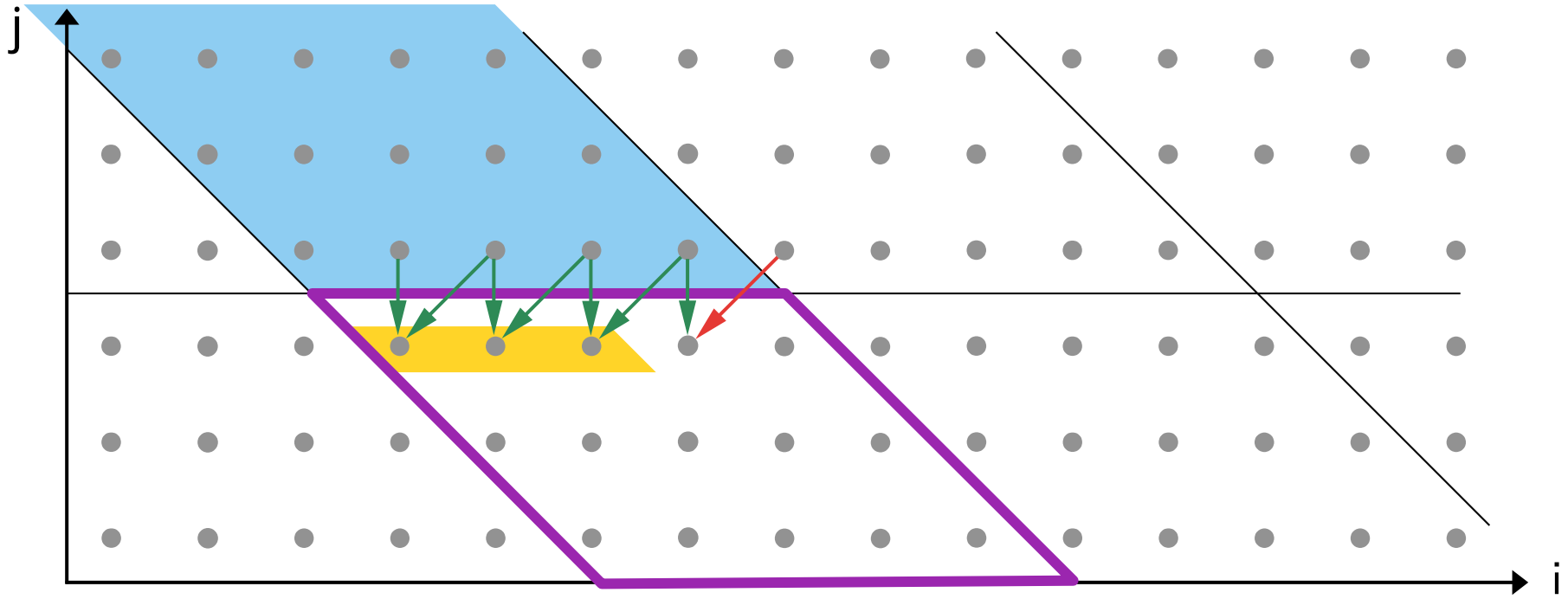
Iterations consumed by tuple less Iterations consumed by other tiles



How to construct the MARS?

- Sought: Iterations consumed **exclusively** by specific consumer tiles
- Constructed by iterating over **tuples of consumers**

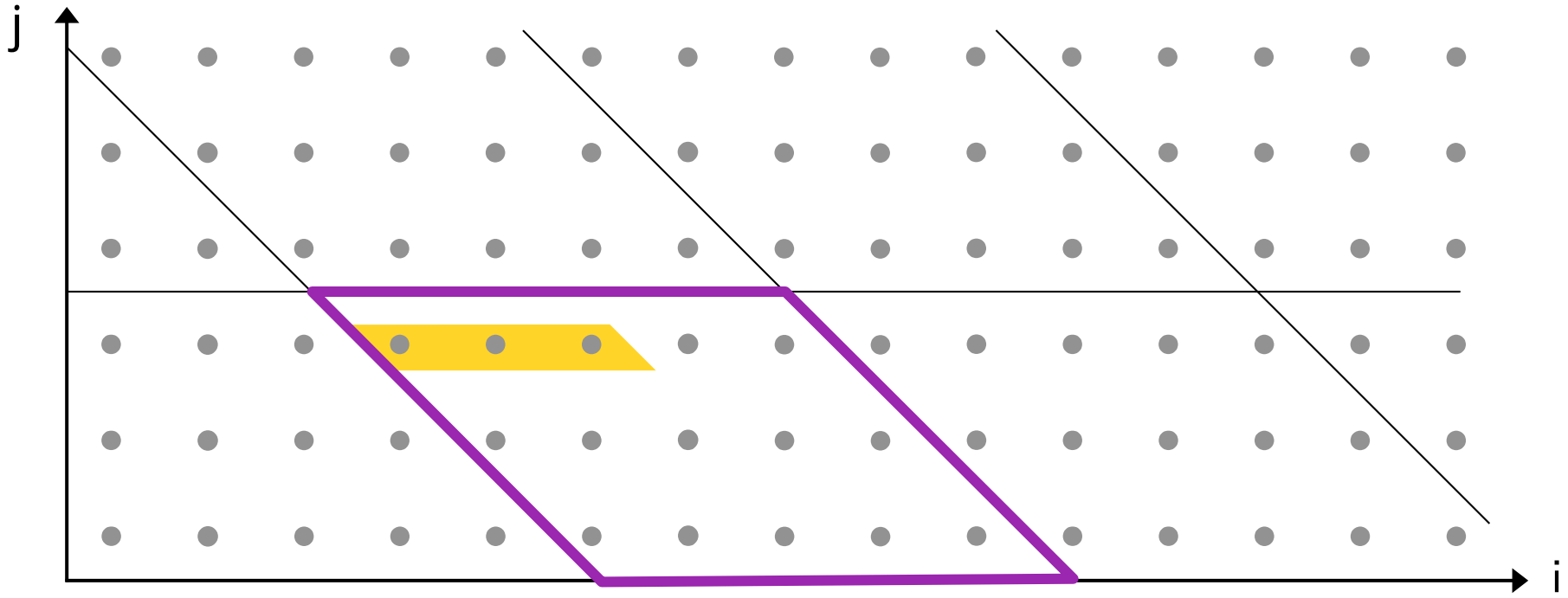
Iterations consumed by tuple less Iterations consumed by other tiles



How to construct the MARS?

- Sought: Iterations consumed **exclusively** by specific consumer tiles
- Constructed by iterating over **tuples of consumers**

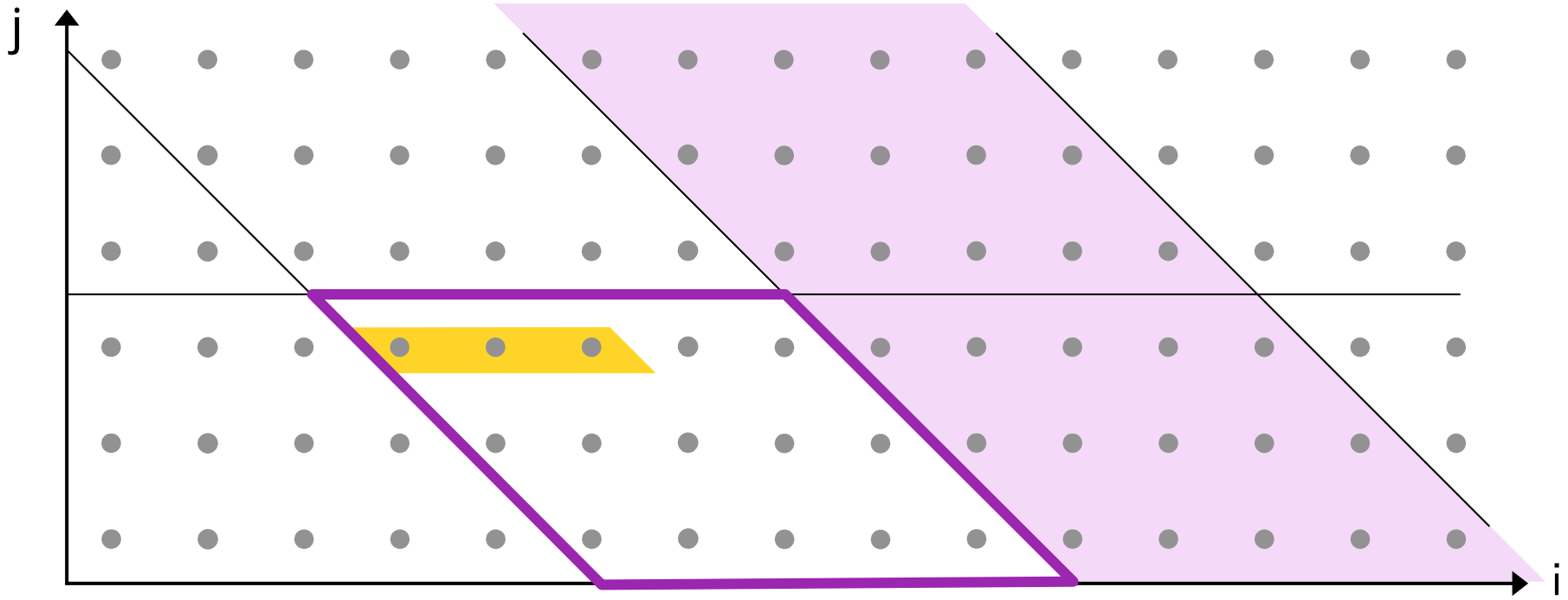
Iterations consumed by tuple less Iterations consumed by other tiles



How to construct the MARS?

- Sought: Iterations consumed **exclusively** by specific consumer tiles
- Constructed by iterating over **tuples of consumers**

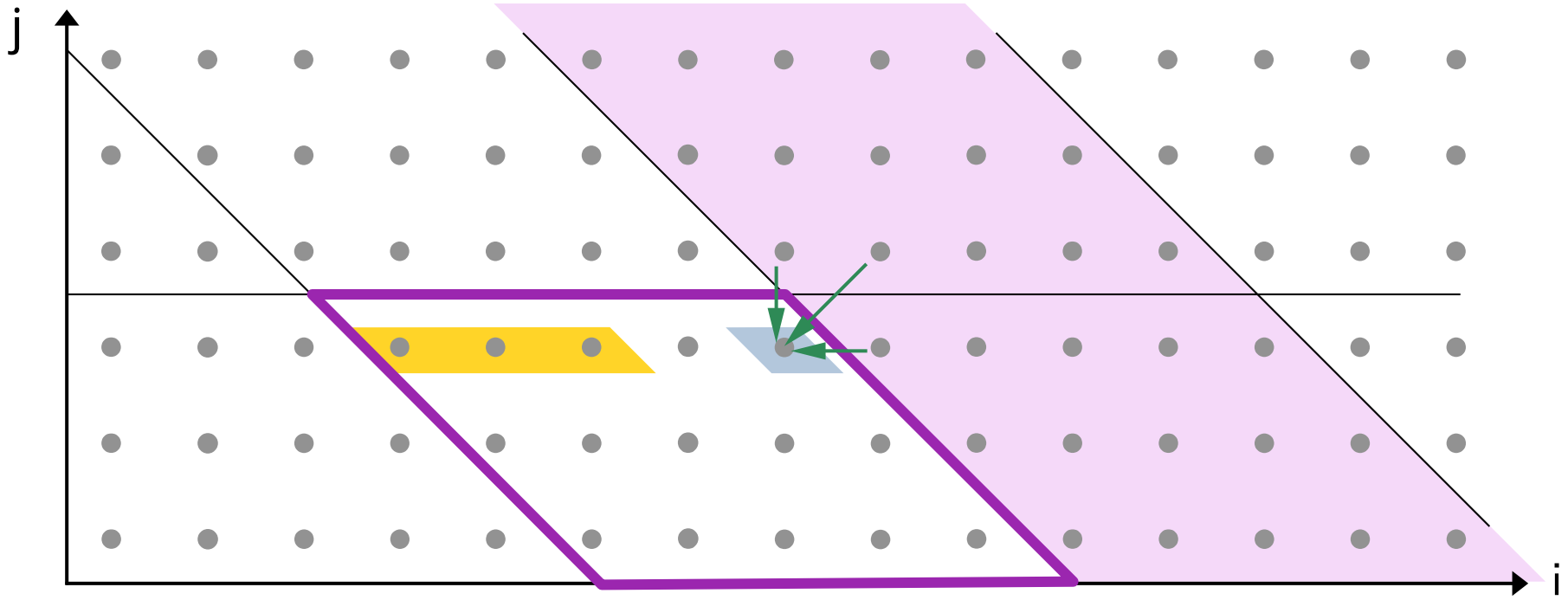
Iterations consumed by tuple less Iterations consumed by other tiles



How to construct the MARS?

- Sought: Iterations consumed **exclusively** by specific consumer tiles
- Constructed by iterating over **tuples of consumers**

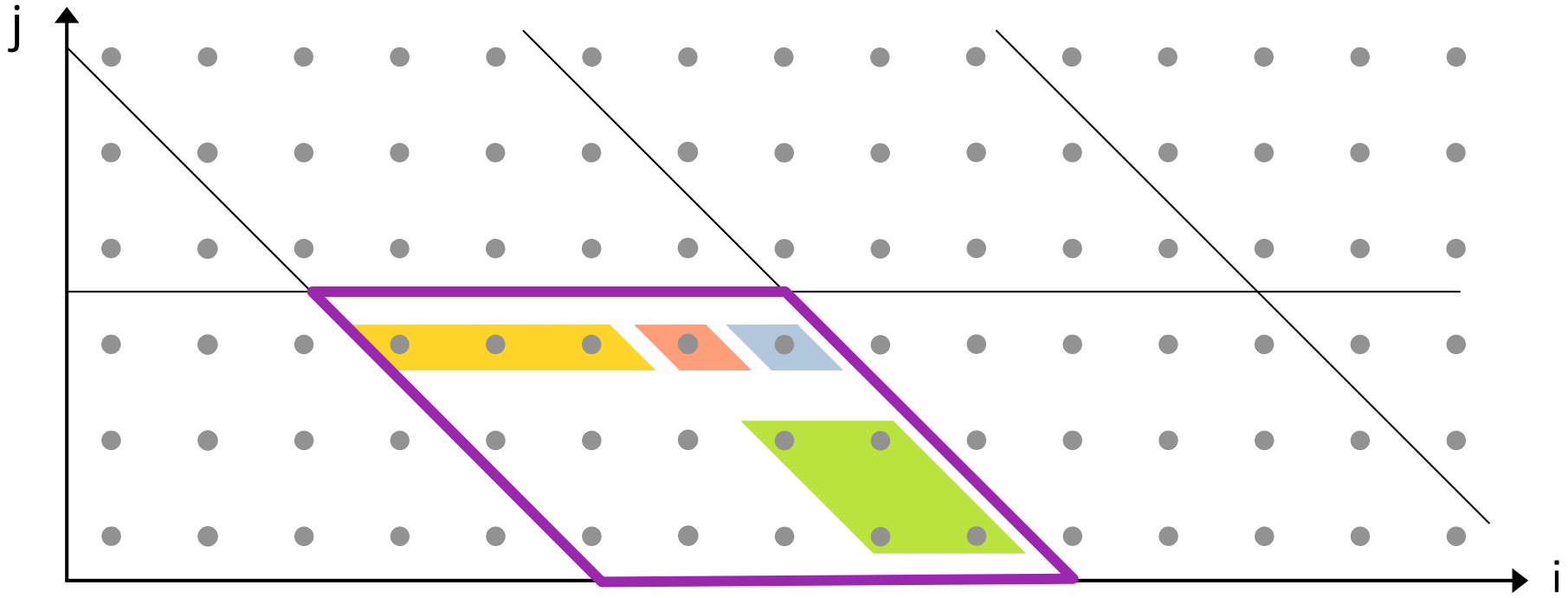
Iterations consumed by tuple less Iterations consumed by other tiles



How to construct the MARS?

- Sought: Iterations consumed **exclusively** by specific consumer tiles
- Constructed by iterating over **tuples of consumers**

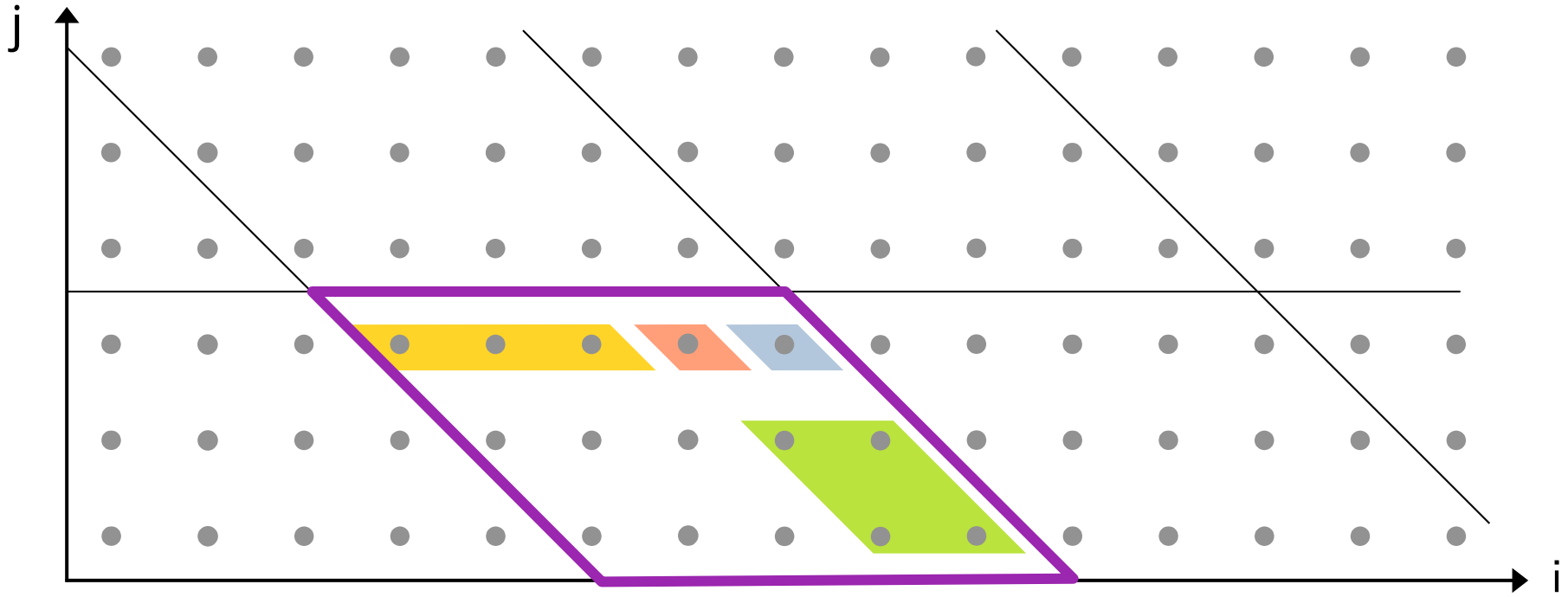
Iterations consumed by tuple less Iterations consumed by other tiles



How to construct the MARS?

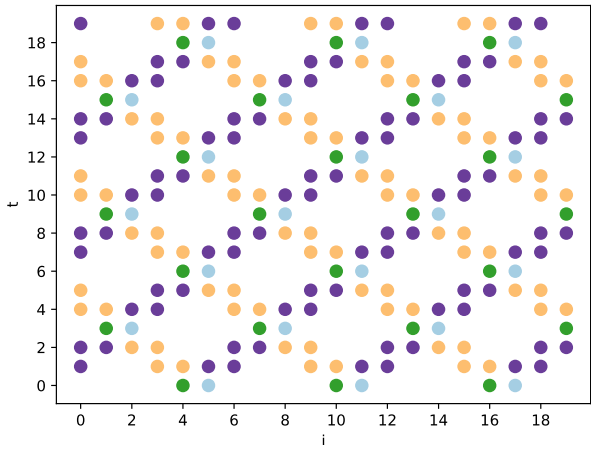
- Sought: Iterations consumed **exclusively** by specific consumer tiles
- Constructed by iterating over **tuples of consumers**

Iterations consumed by tuple less Iterations consumed by other tiles



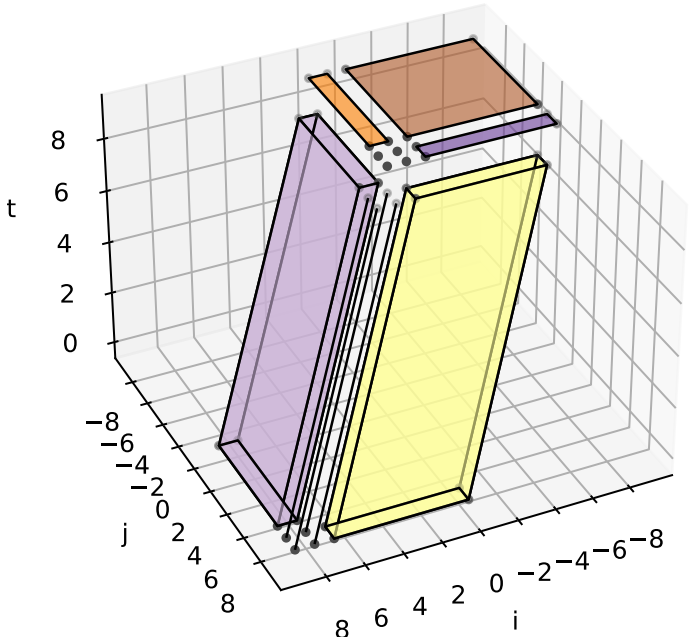
Union of MARS = Flow-Out (complete!)

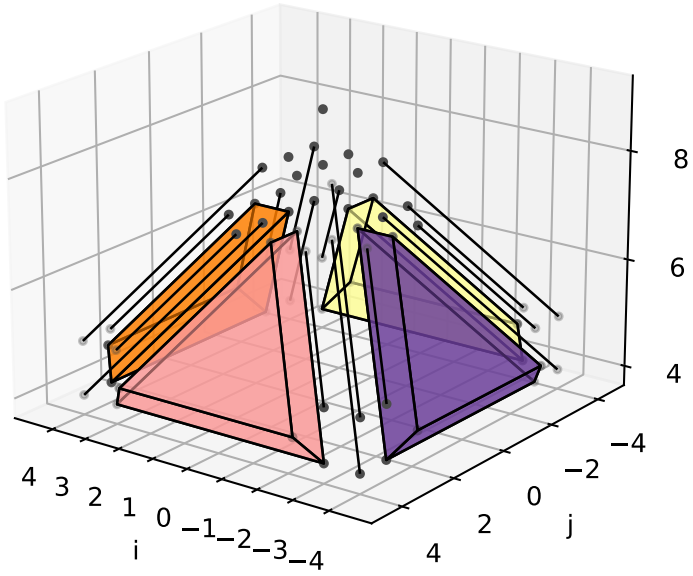
Some examples of MARS



Jacobi 1D, diamond tiling

Jacobi 2D, skewed tiling





Jacobi 2D, “diamond” tiling

Applications of MARS

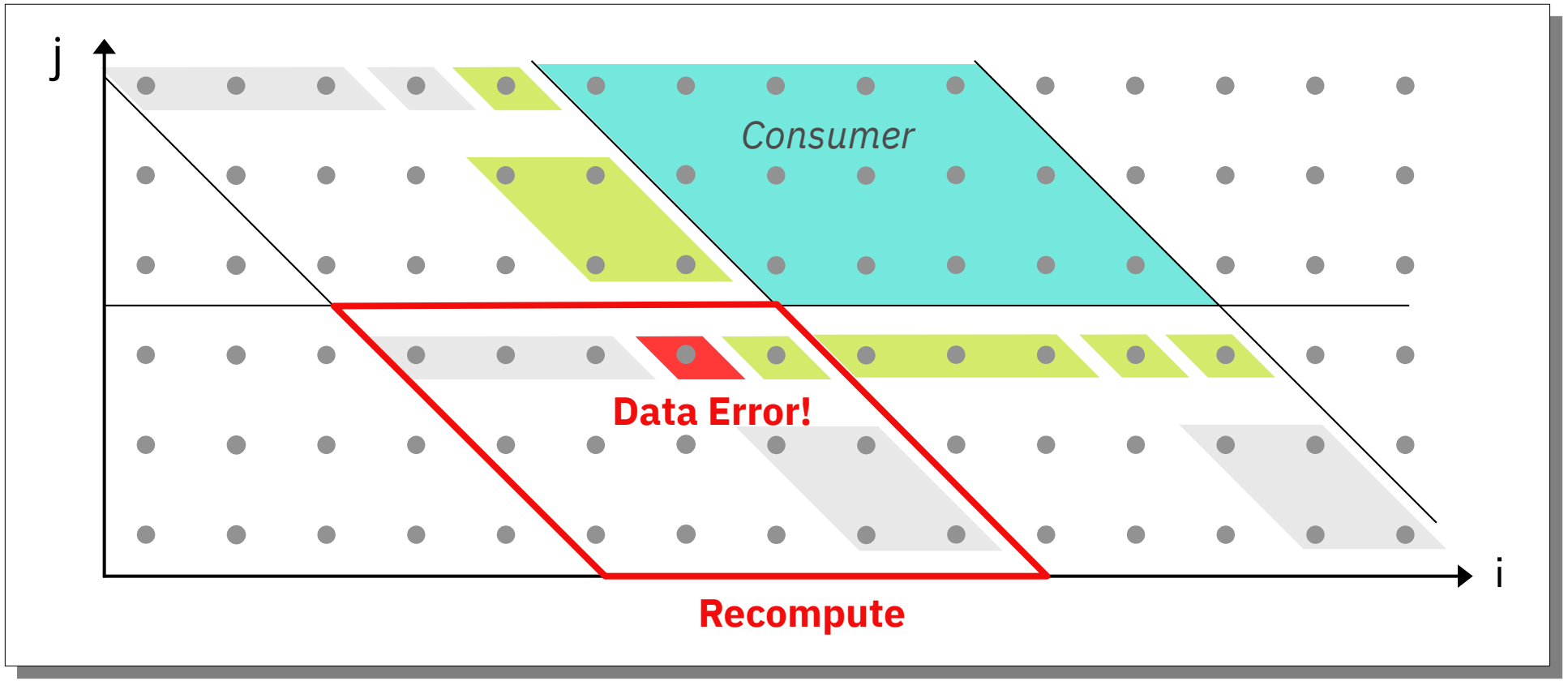
- **Memory allocation for FPGA accelerators**
  - Exploit atomicity of the MARS
  - Derive a data layout using MARS minimizing read transactions

Applications of MARS

- **Memory allocation for FPGA accelerators**
  - Exploit atomicity of the MARS
  - Derive a data layout using MARS minimizing read transactions

- **Compression**
  - Along with data layout → increase the effective bandwidth (amount of useful data transmitted over the bus) thanks to MARS' irredundancy

Applications of MARS



- **Fault tolerance**
  - Compute a checksum on each MARS. If error → the producer tile (known) is to be re-executed.

Application of MARS: Contributions

- Use the MARS decomposition to exhibit contiguity
  - Derive MARS from program, allocate data space for MARS
  - ***Find a layout maximizing the contiguity opportunities***

Application of MARS: Contributions

- Use the MARS decomposition to exhibit contiguity
  - Derive MARS from program, allocate data space for MARS
  - ***Find a layout maximizing the contiguity opportunities***

- Exploit MARS atomicity with data compression
  - ***Build a compression engine on FPGA for MARS data***

Application of MARS: Contributions

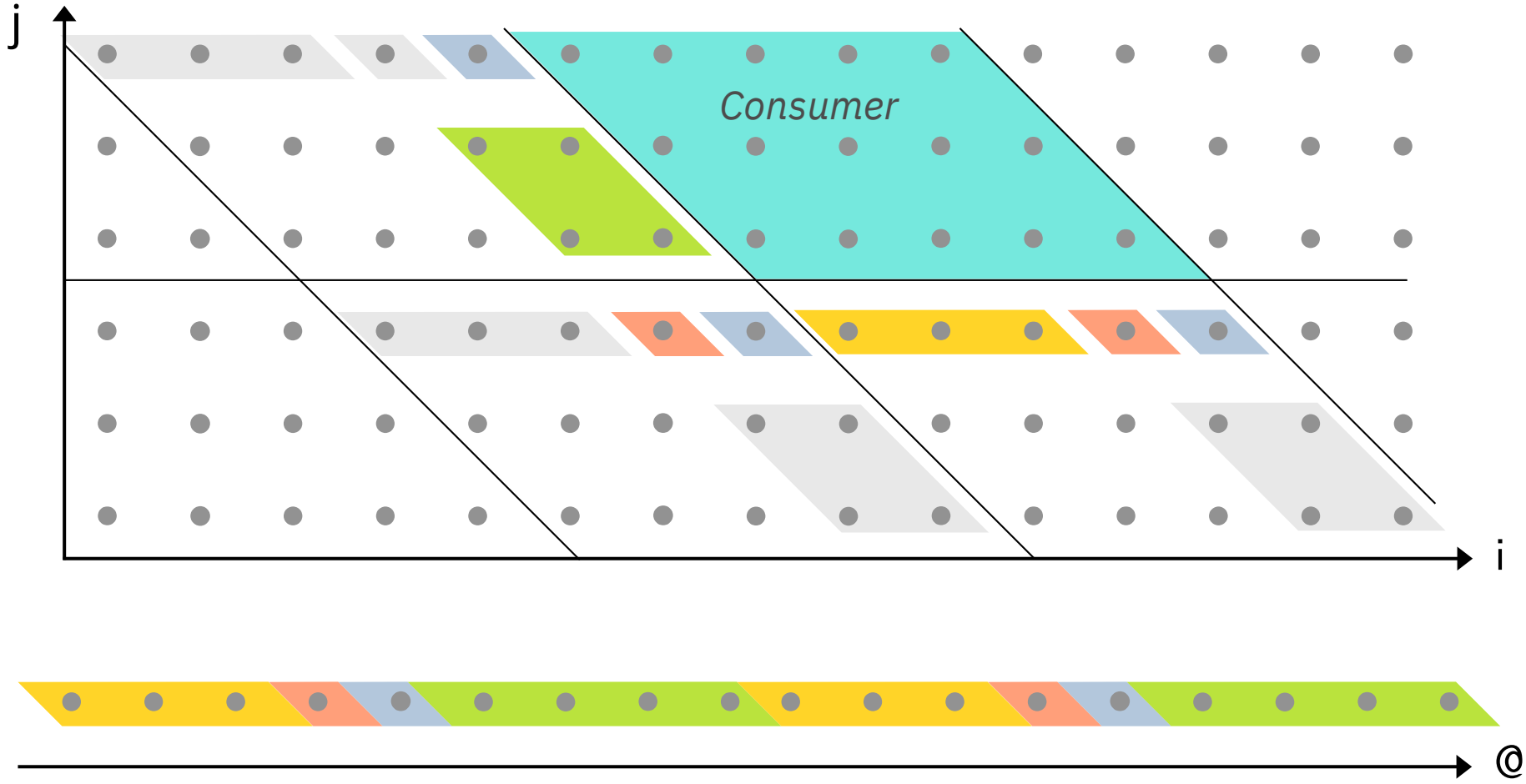
- Use the MARS decomposition to exhibit contiguity
  - Derive MARS from program, allocate data space for MARS
  - ***Find a layout maximizing the contiguity opportunities***

- Exploit MARS atomicity with data compression
  - ***Build a compression engine on FPGA for MARS data***

- Automate the process
  - ***Integrate the compressed MARS into existing FPGA acceleration code***

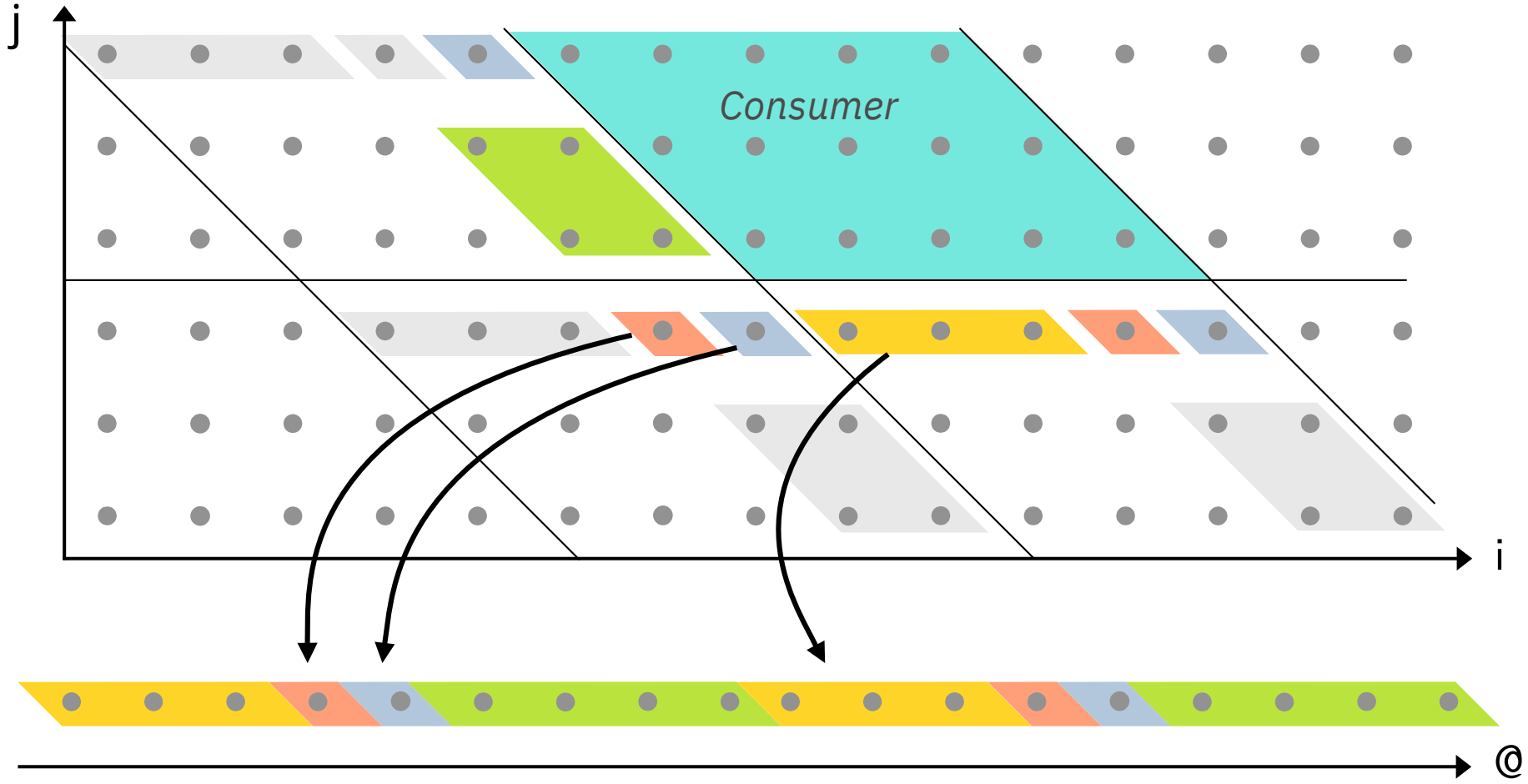
Optimizing Memory Layout of MARS

- Find MARS layout in memory **minimizing read time**
- **Remember: Memory is Linear (1 MARS between at most 2 MARSes)**



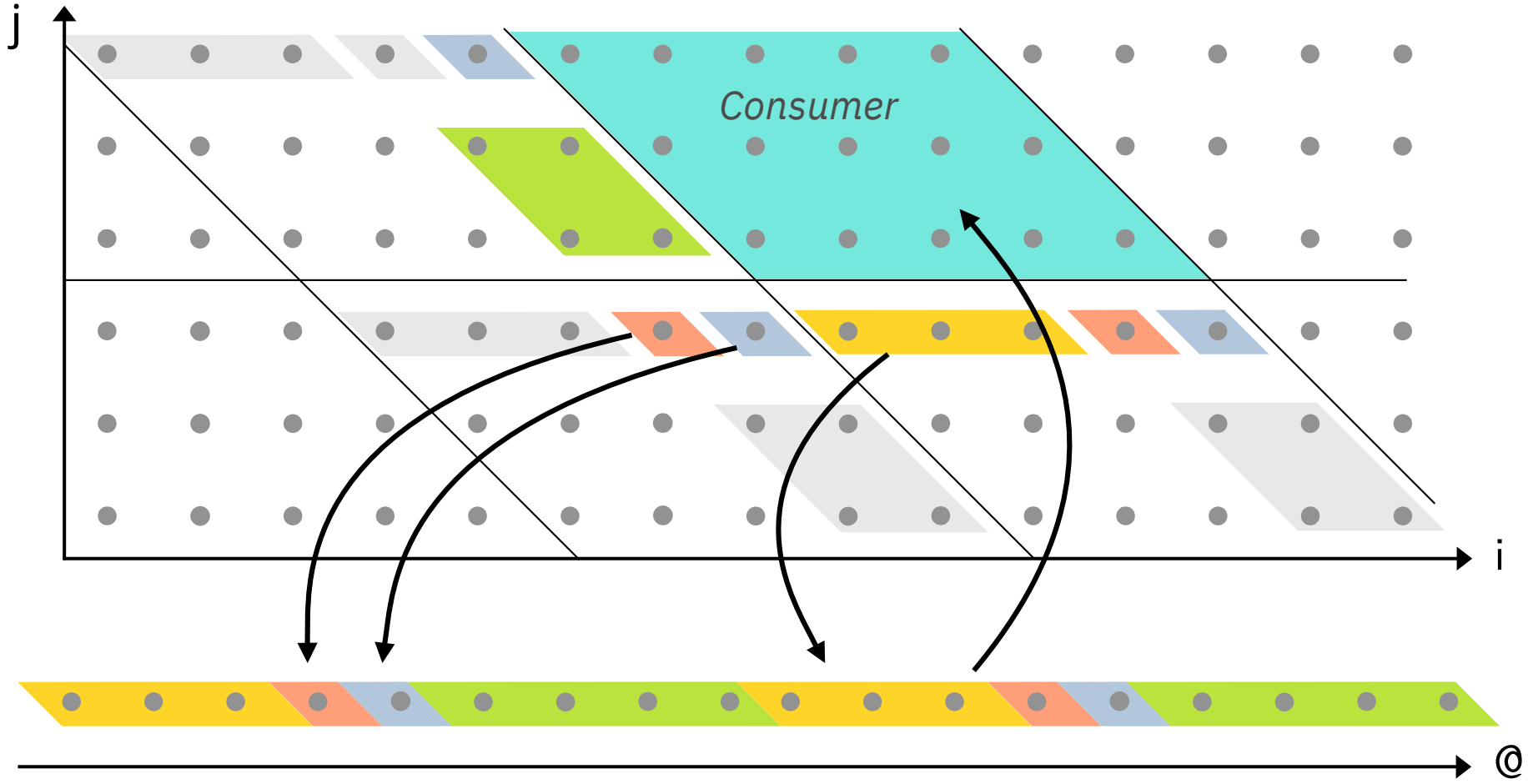
Optimizing Memory Layout of MARS

- Find MARS layout in memory **minimizing read time**
- **Remember: Memory is Linear (1 MARS between at most 2 MARSes)**



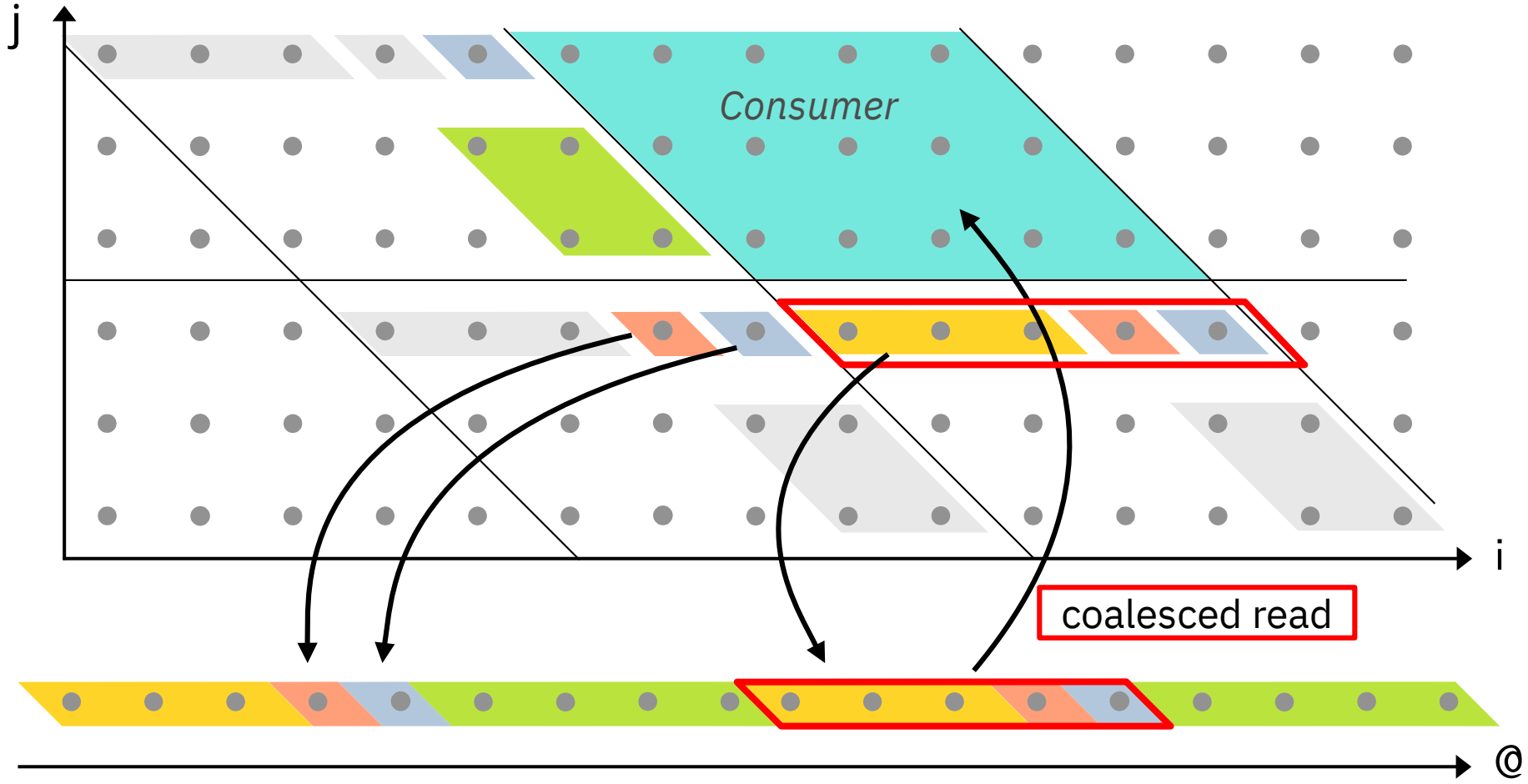
Optimizing Memory Layout of MARS

- Find MARS layout in memory **minimizing read time**
- **Remember: Memory is Linear (1 MARS between at most 2 MARSes)**



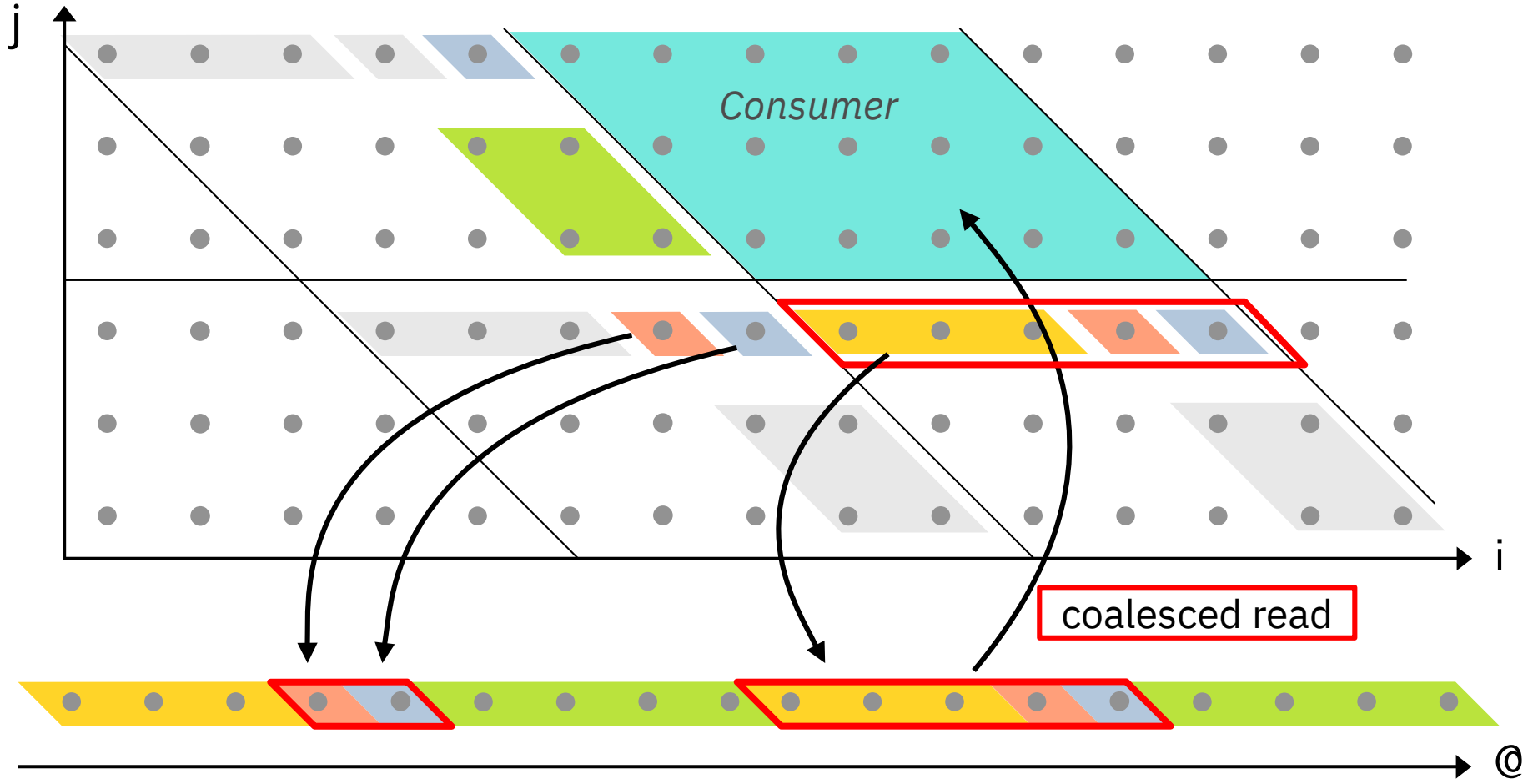
Optimizing Memory Layout of MARS

- Find MARS layout in memory **minimizing read time**
- **Remember: Memory is Linear (1 MARS between at most 2 MARSes)**



Optimizing Memory Layout of MARS

- Find MARS layout in memory **minimizing read time**
- **Remember: Memory is Linear (1 MARS between at most 2 MARSes)**



Optimizing Memory Layout of MARS

- Find MARS
- **Remember**

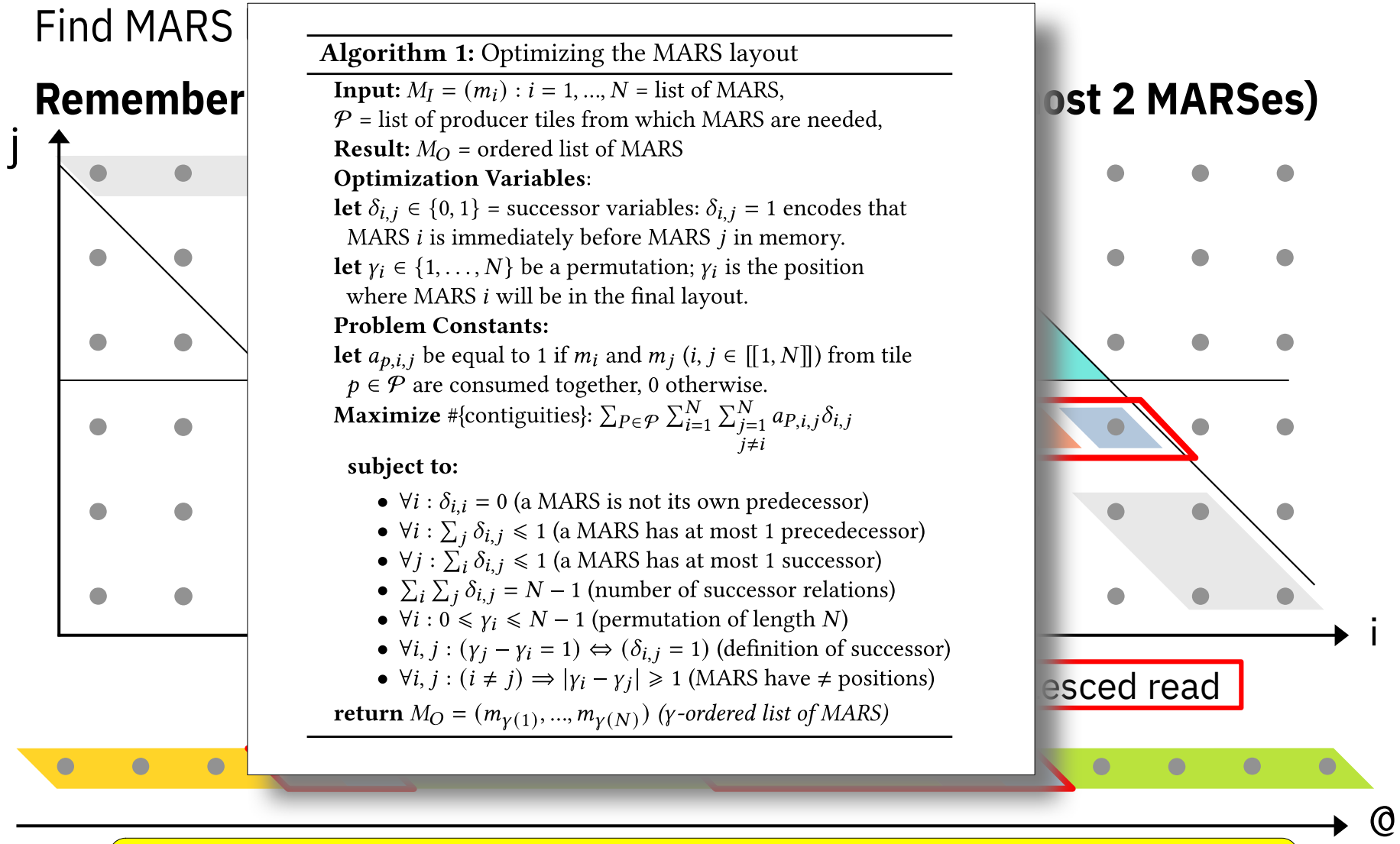
Algorithm 1: Optimizing the MARS layout

Input: $M_I = (m_i) : i = 1, ..., N$ = list of MARS,
$\mathcal{P}$ = list of producer tiles from which MARS are needed,
Result: M_O = ordered list of MARS
Optimization Variables:
let $\delta_{i,j} \in \{0, 1\}$ = successor variables: $\delta_{i,j} = 1$ encodes that MARS i is immediately before MARS j in memory.
let $\gamma_i \in \{1, \ldots, N\}$ be a permutation; γ_i is the position where MARS i will be in the final layout.
Problem Constants:
let $a_{p,i,j}$ be equal to 1 if m_i and m_j $(i, j \in [[1, N]])$ from tile $p \in \mathcal{P}$ are consumed together, 0 otherwise.
Maximize #{contiguities}: $\sum_{P \in \mathcal{P}} \sum_{i=1}^{N} \sum_{\substack{j=1 \\ j \neq i}}^{N} a_{P,i,j} \delta_{i,j}$

subject to:

- $\forall i : \delta_{i,i} = 0$ (a MARS is not its own predecessor)
- $\forall i : \sum_j \delta_{i,j} \leqslant 1$ (a MARS has at most 1 precedecessor)
- $\forall j : \sum_i \delta_{i,j} \leqslant 1$ (a MARS has at most 1 successor)
- $\sum_i \sum_j \delta_{i,j} = N - 1$ (number of successor relations)
- $\forall i : 0 \leqslant \gamma_i \leqslant N - 1$ (permutation of length N)
- $\forall i, j : (\gamma_j - \gamma_i = 1) \Leftrightarrow (\delta_{i,j} = 1)$ (definition of successor)
- $\forall i, j : (i \neq j) \Rightarrow |\gamma_i - \gamma_j| \geqslant 1$ (MARS have $\neq$ positions)

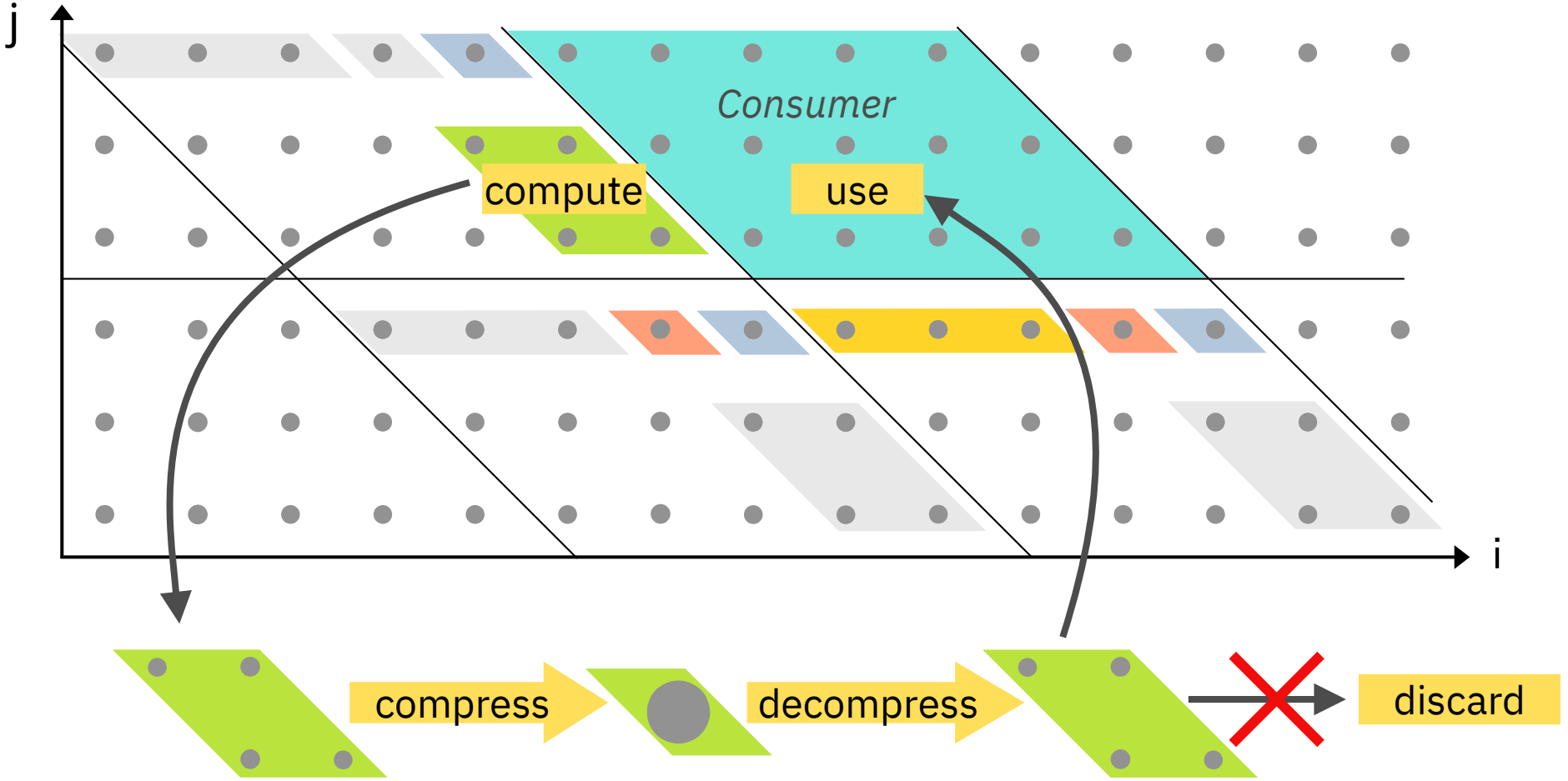
return $M_O = (m_{\gamma(1)}, ..., m_{\gamma(N)})$ *(γ-ordered list of MARS)*



Formulated as an LP optimization problem

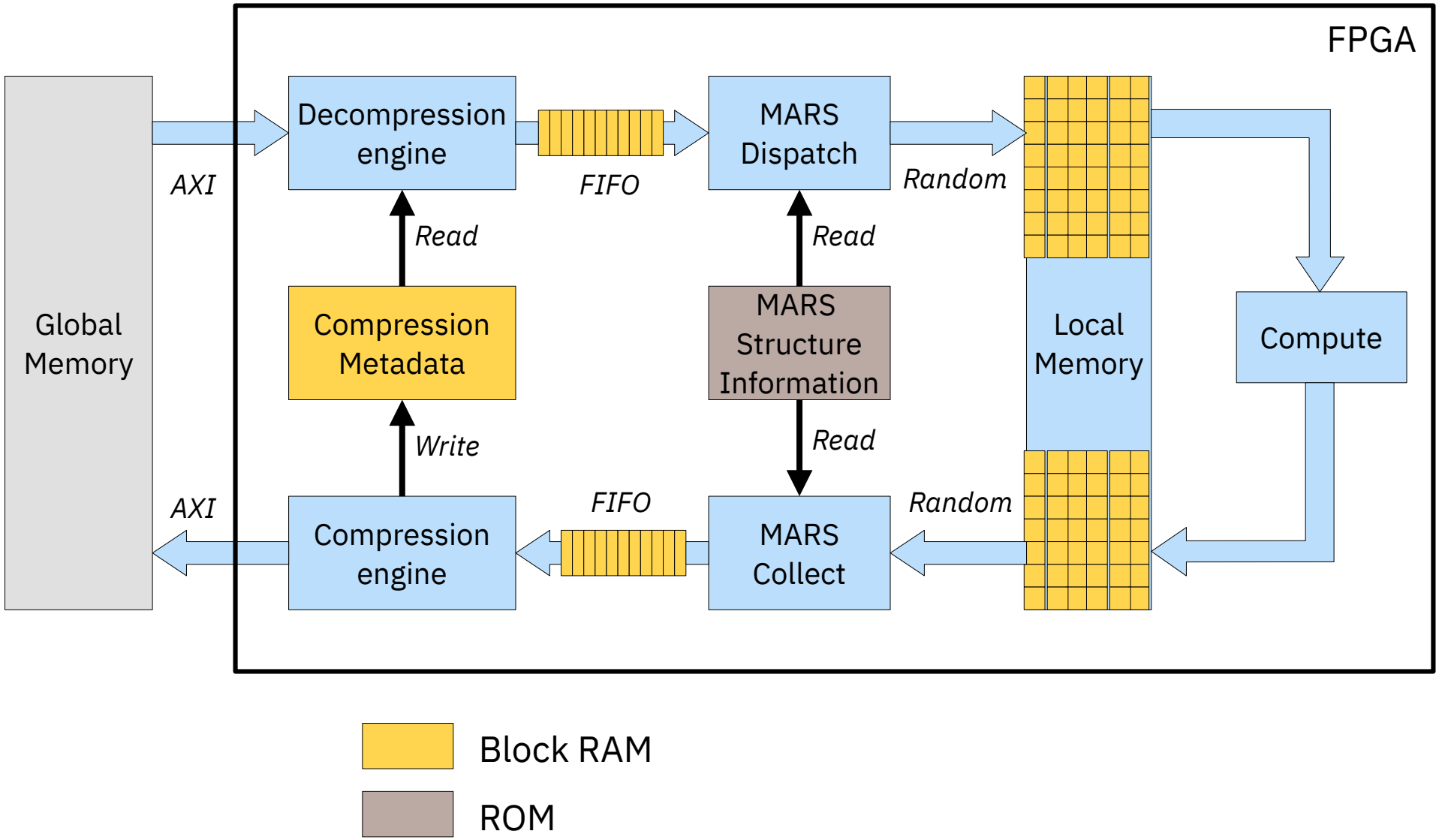
Exploiting Atomicity for Compression

- Compressing data creates atomic blocks
- **MARS are already atomic → leverage this property**
- Efficient compression = **no redundancy**
  - e.g. dynamic compression [Ozturk 2009] : decompress one full tile



Compressed MARS FPGA Architecture

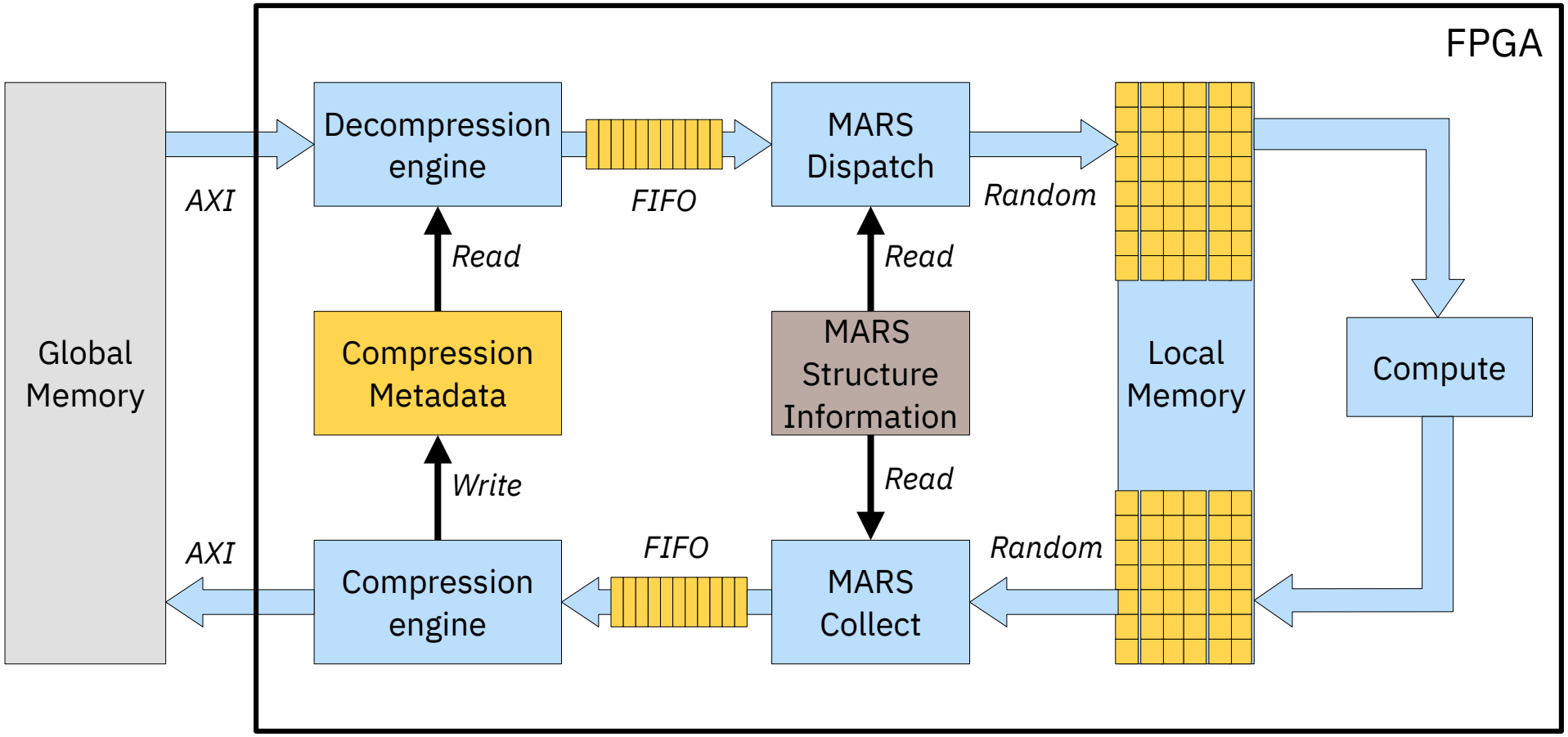
Combines MARS (random ↔ contiguous) + compression to save bandwidth



Validation on FPGA

Validation on **PolyBench benchmarks** on FPGA (ZCU104)

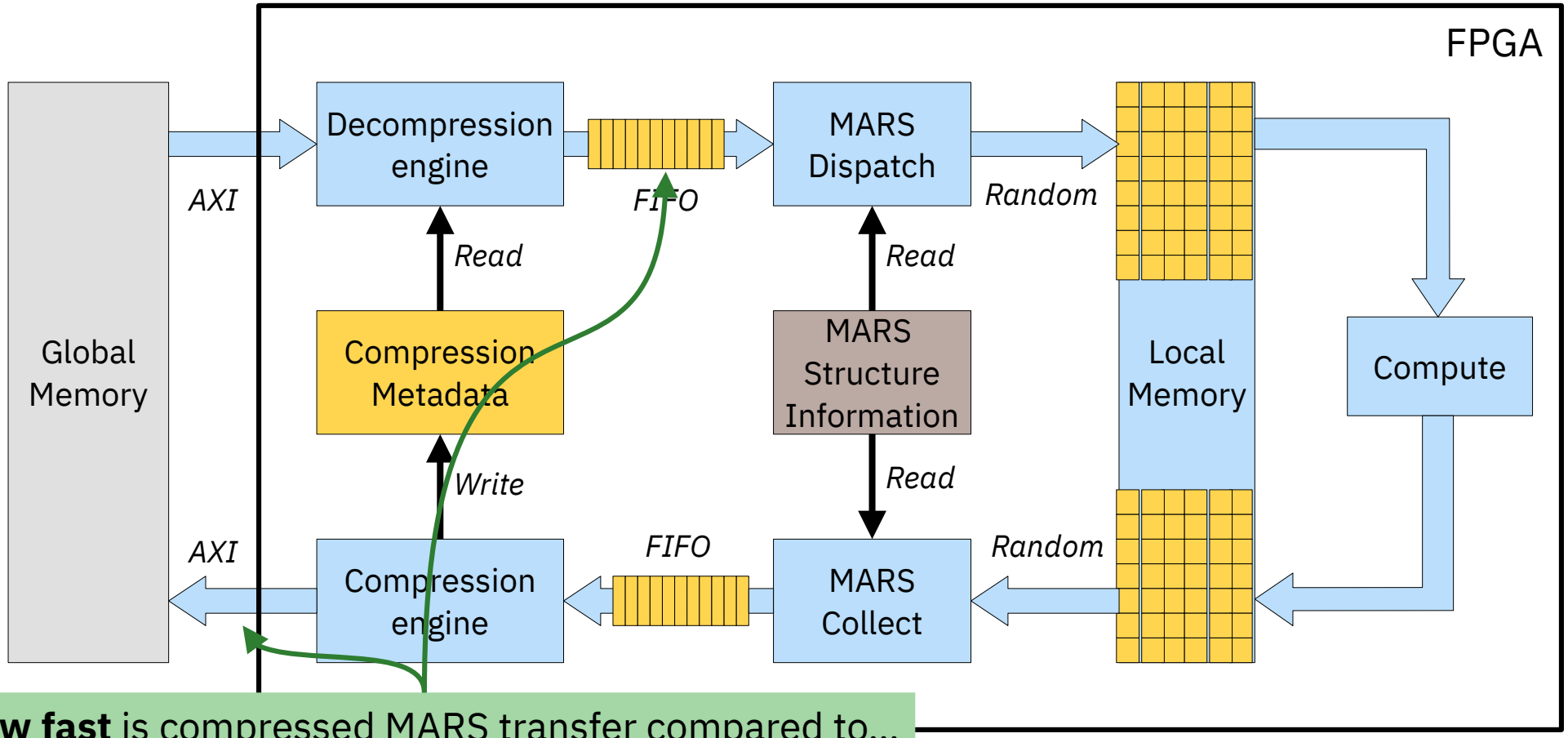
jacobi-1d, jacobi-2d, seidel-2d



Validation on FPGA

Validation on **PolyBench benchmarks** on FPGA (ZCU104)

jacobi-1d, jacobi-2d, seidel-2d



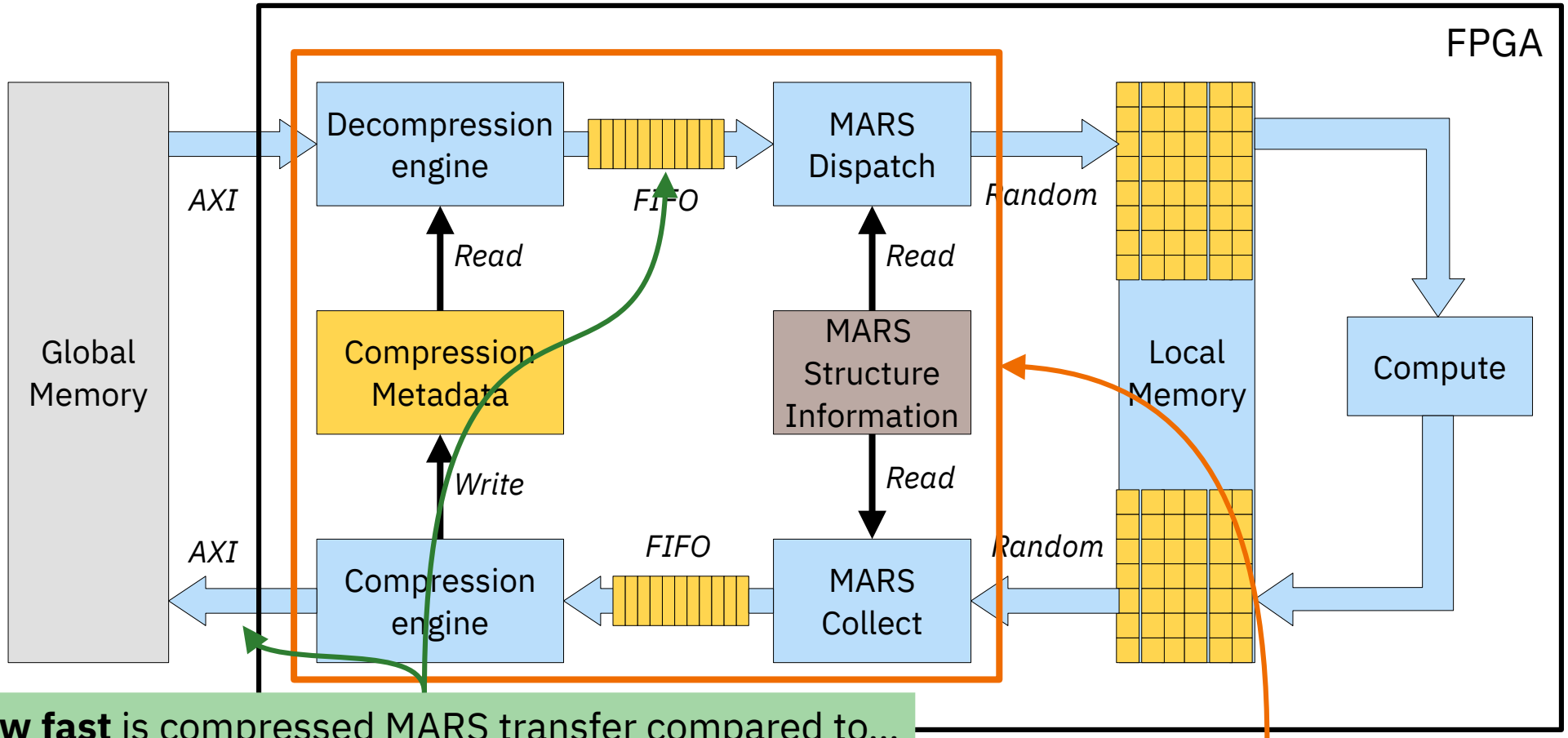
How fast is compressed MARS transfer compared to...

- A rectangular bounding box around the data?
- The original allocation for CPU program?
- MARS without compression?

Validation on FPGA

Validation on **PolyBench benchmarks** on FPGA (ZCU104)

jacobi-1d, jacobi-2d, seidel-2d



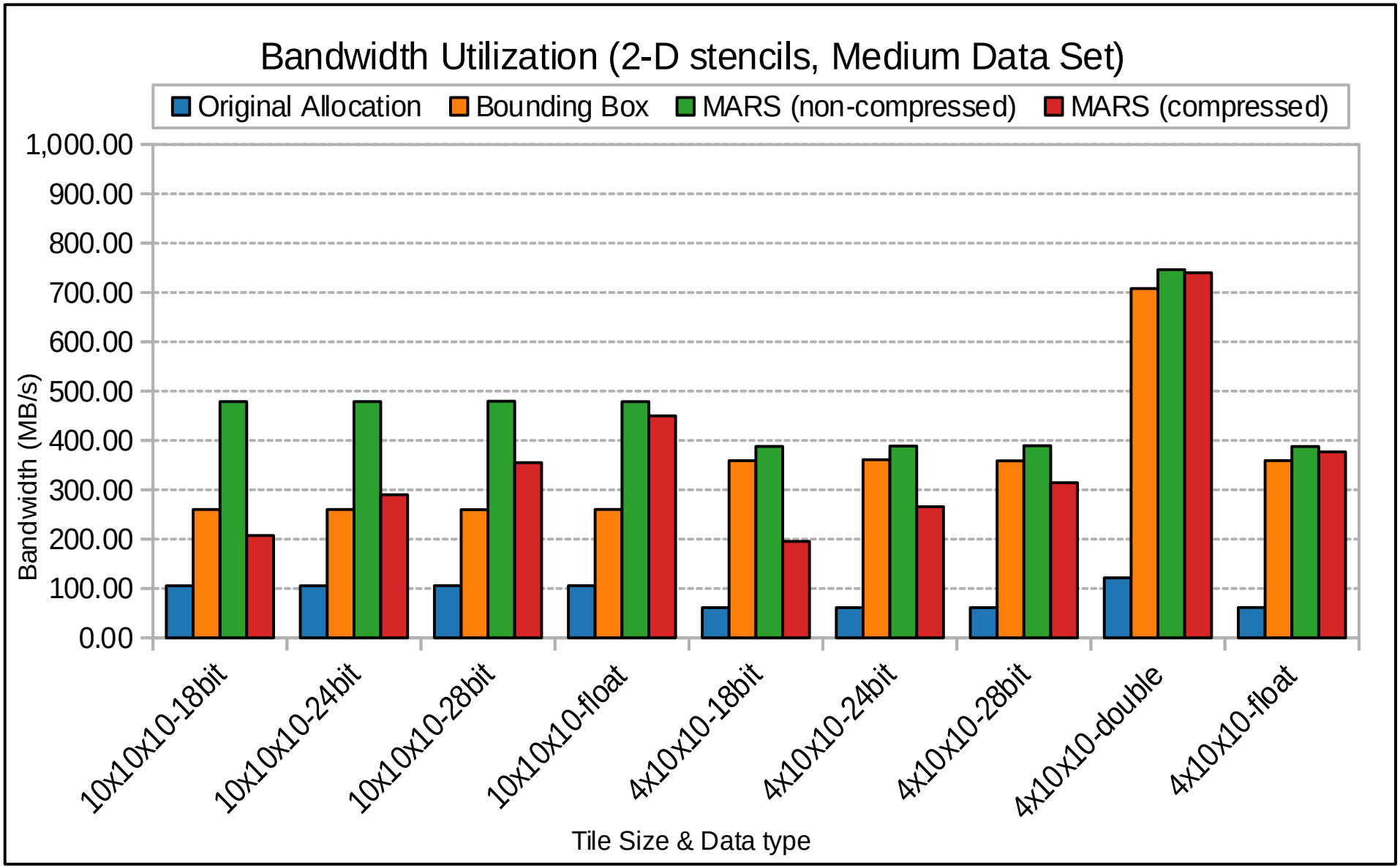
How fast is compressed MARS transfer compared to...

- A rectangular bounding box around the data?
- The original allocation for CPU program?
- MARS without compression?

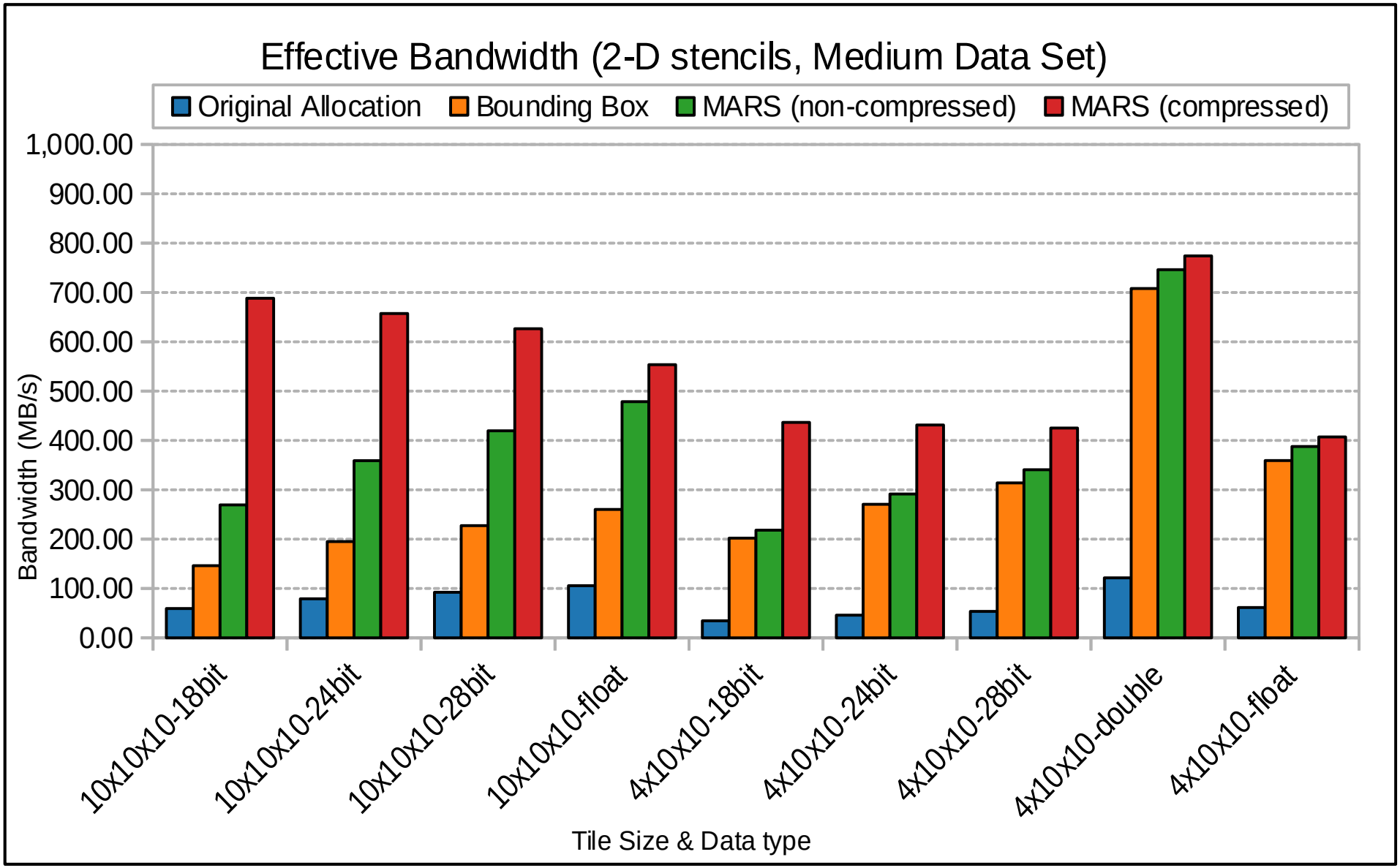
How large is the transfer machinery

- In terms of logic/state machines?
- In terms of extra storage?

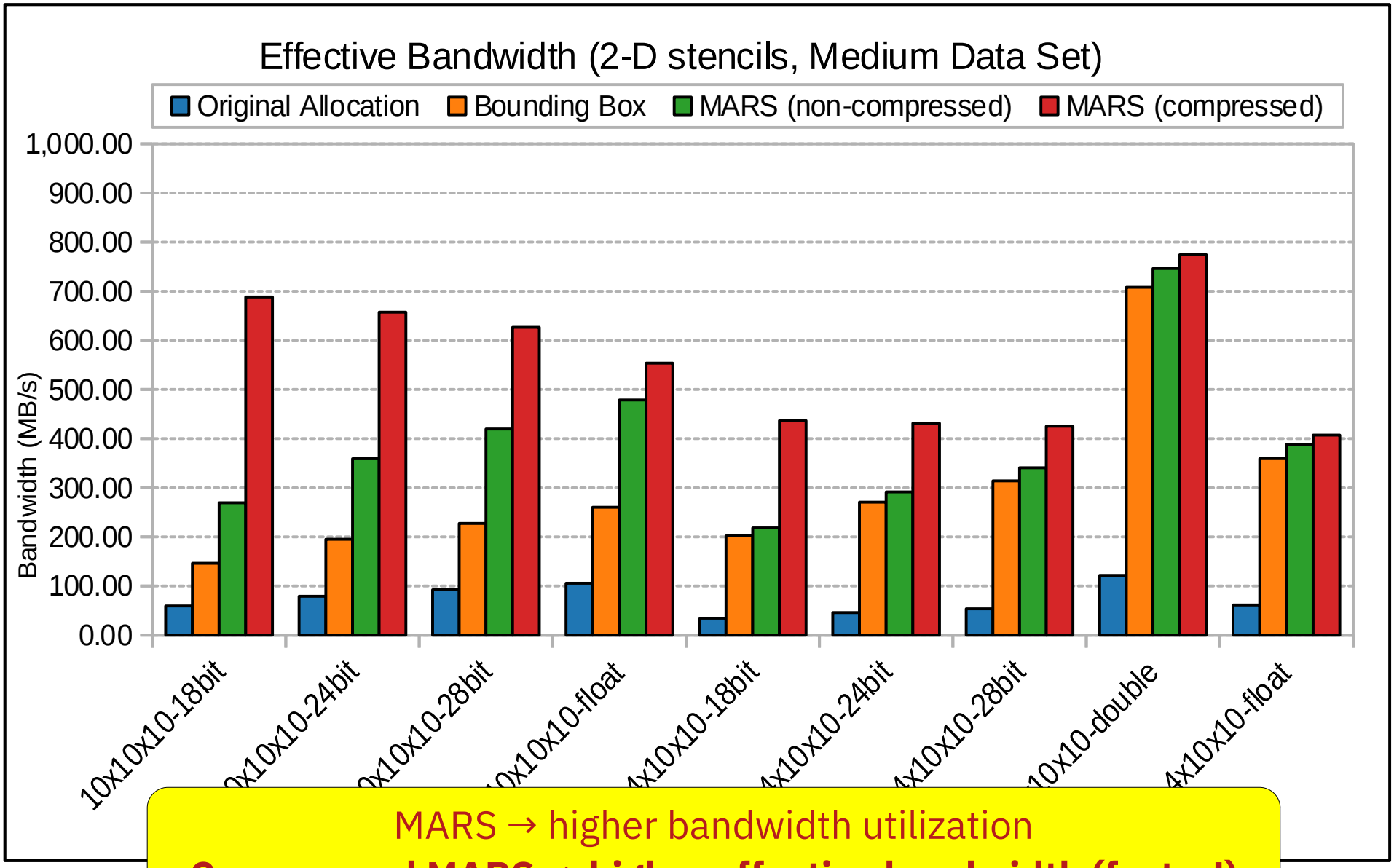
MARS Enables Higher Bandwidth



MARS Enables Higher Bandwidth



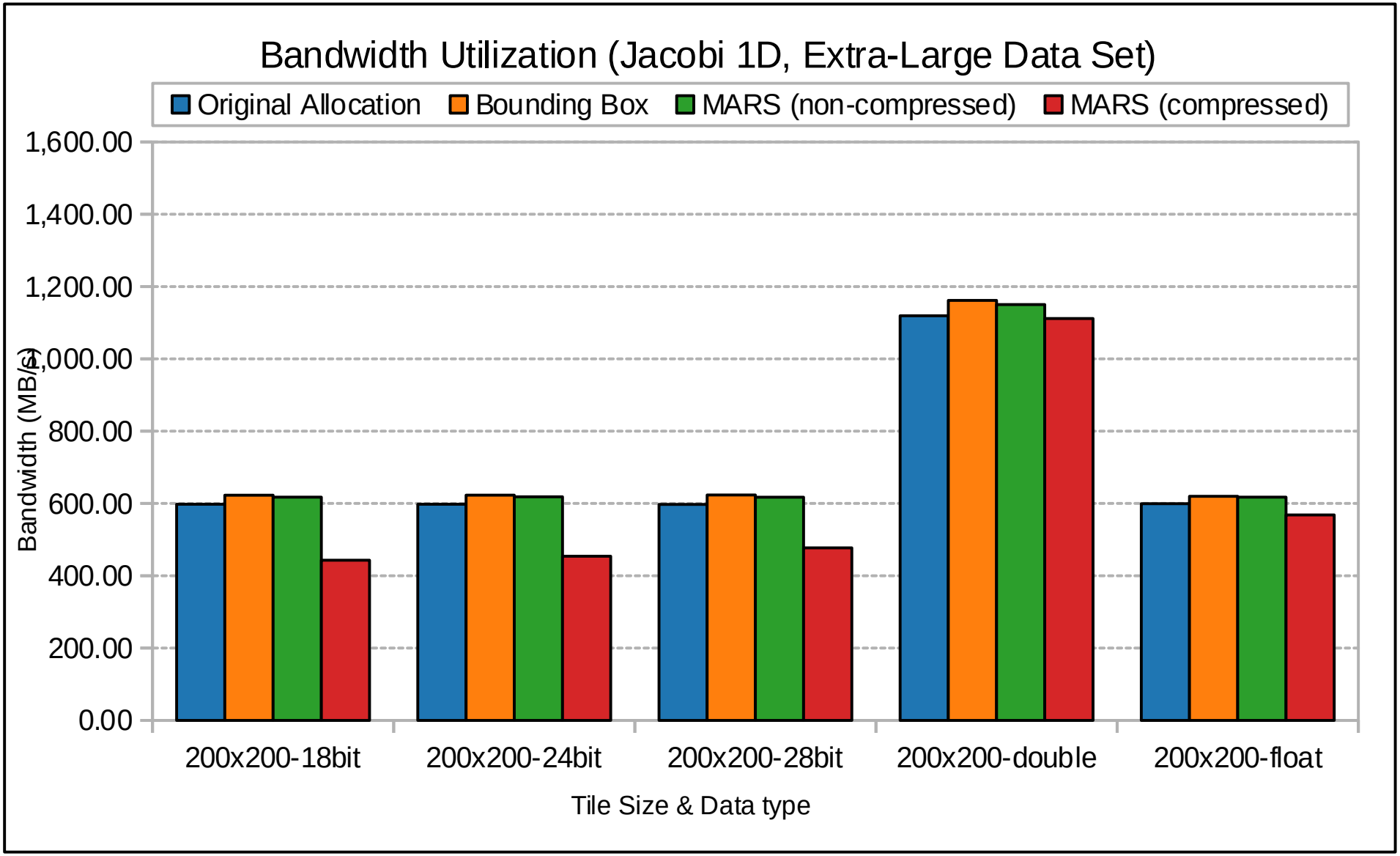
MARS Enables Higher Bandwidth



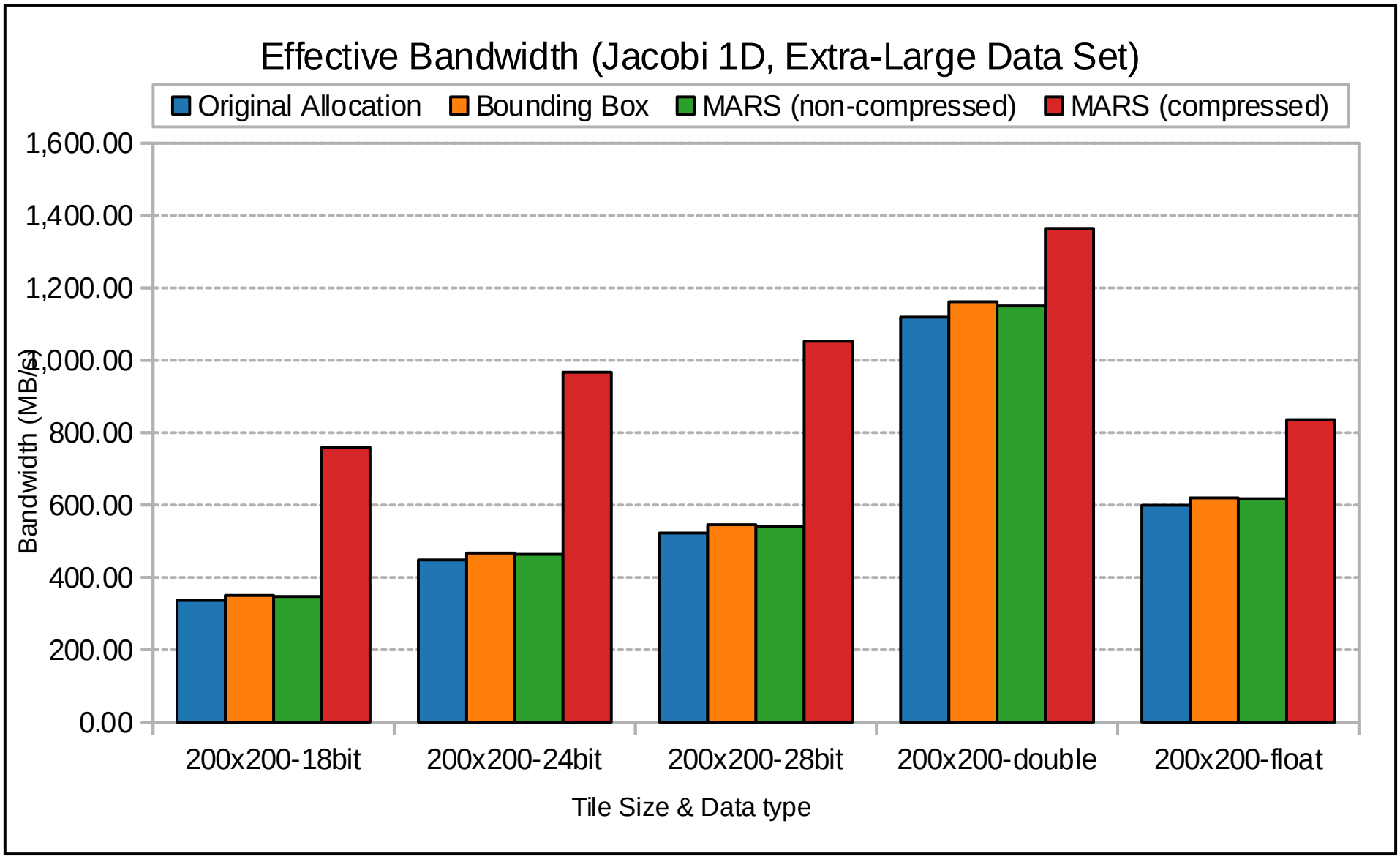
MARS → higher bandwidth utilization

Compressed MARS → higher effective bandwidth (faster!)

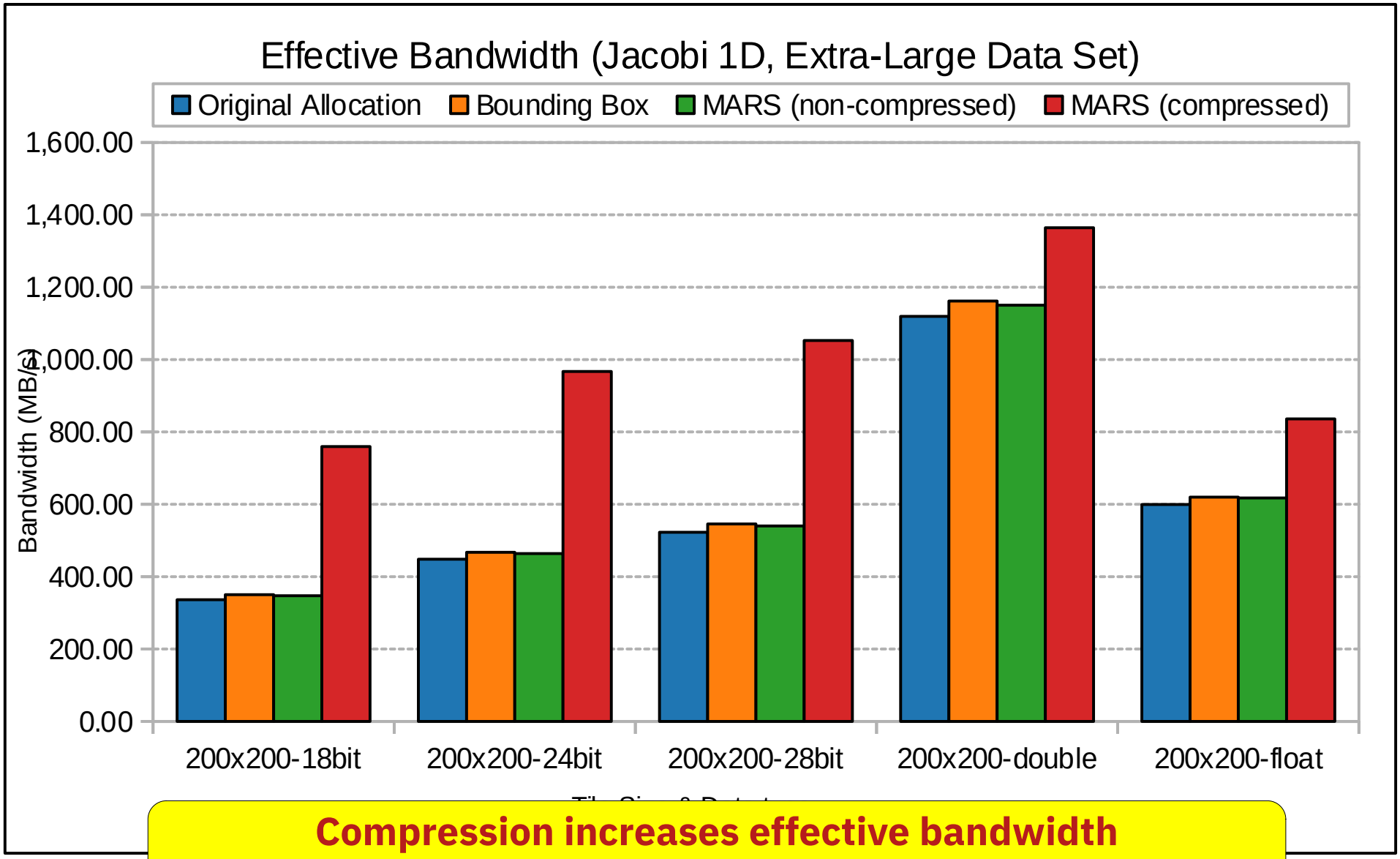
Compression Enables Higher Bandwidth



Compression Enables Higher Bandwidth

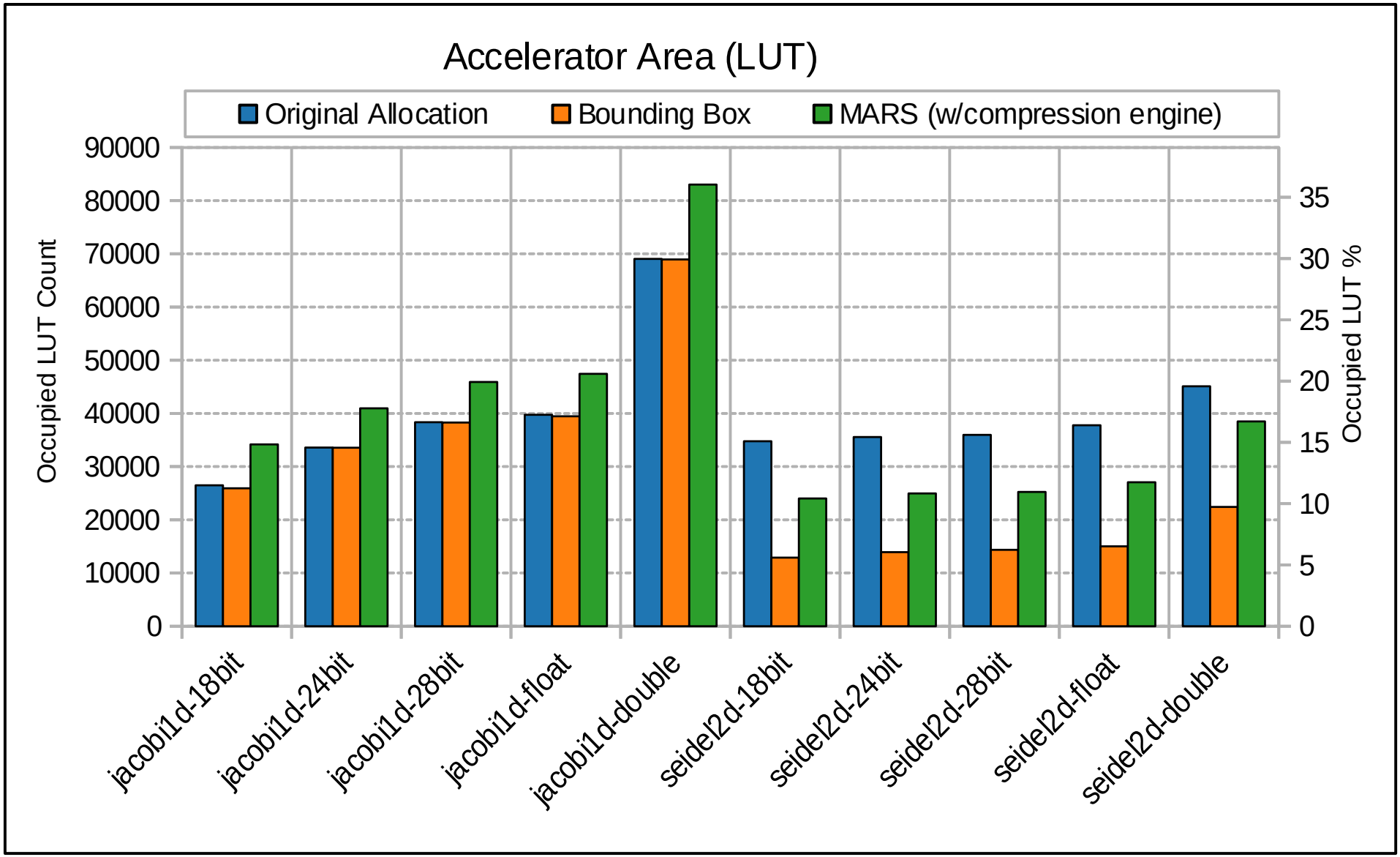


Compression Enables Higher Bandwidth

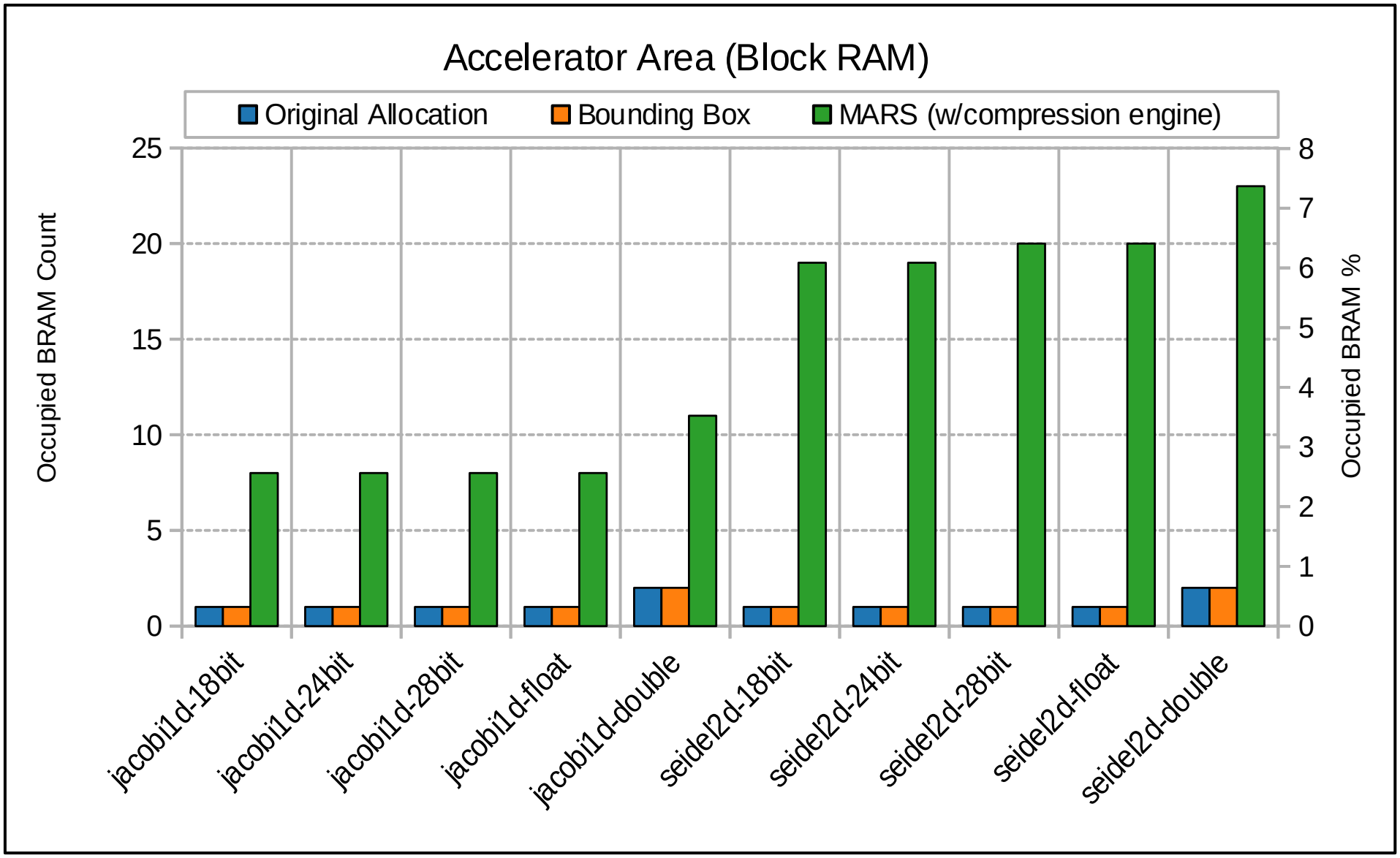


Compression increases effective bandwidth
even in 1-D cases

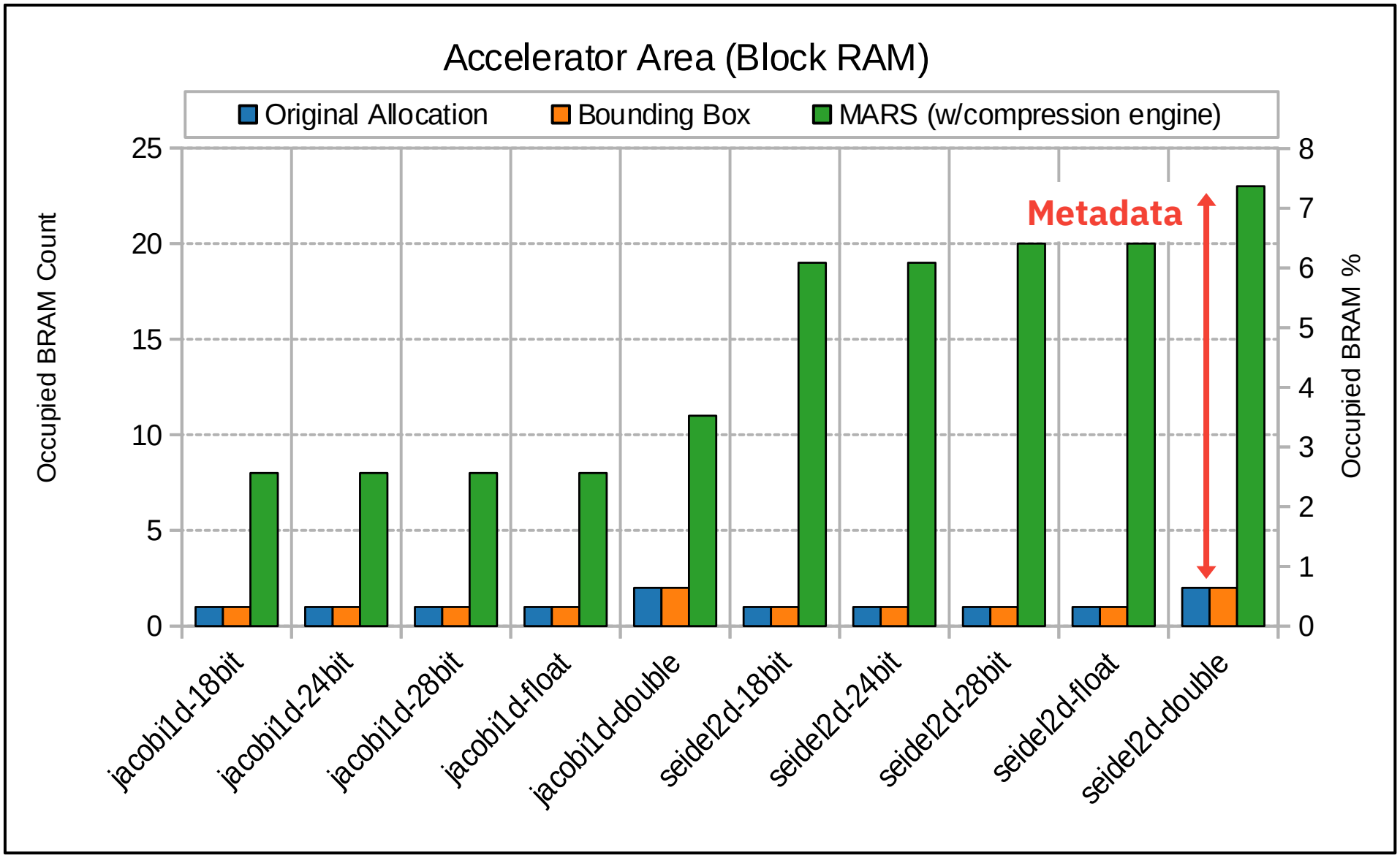
High bandwidth has a cost!



High bandwidth has a cost!



High bandwidth has a cost!



MARS take-aways

Compression + Data Contiguity = High Effective Bandwidth

Open doors…

- Limited to a very **specific class of programs** (uniform dependences)
- How about a **fine-grain layout** inside MARS?

  → Would prevent on-chip port contention
- Add **data persistence** across tiles?

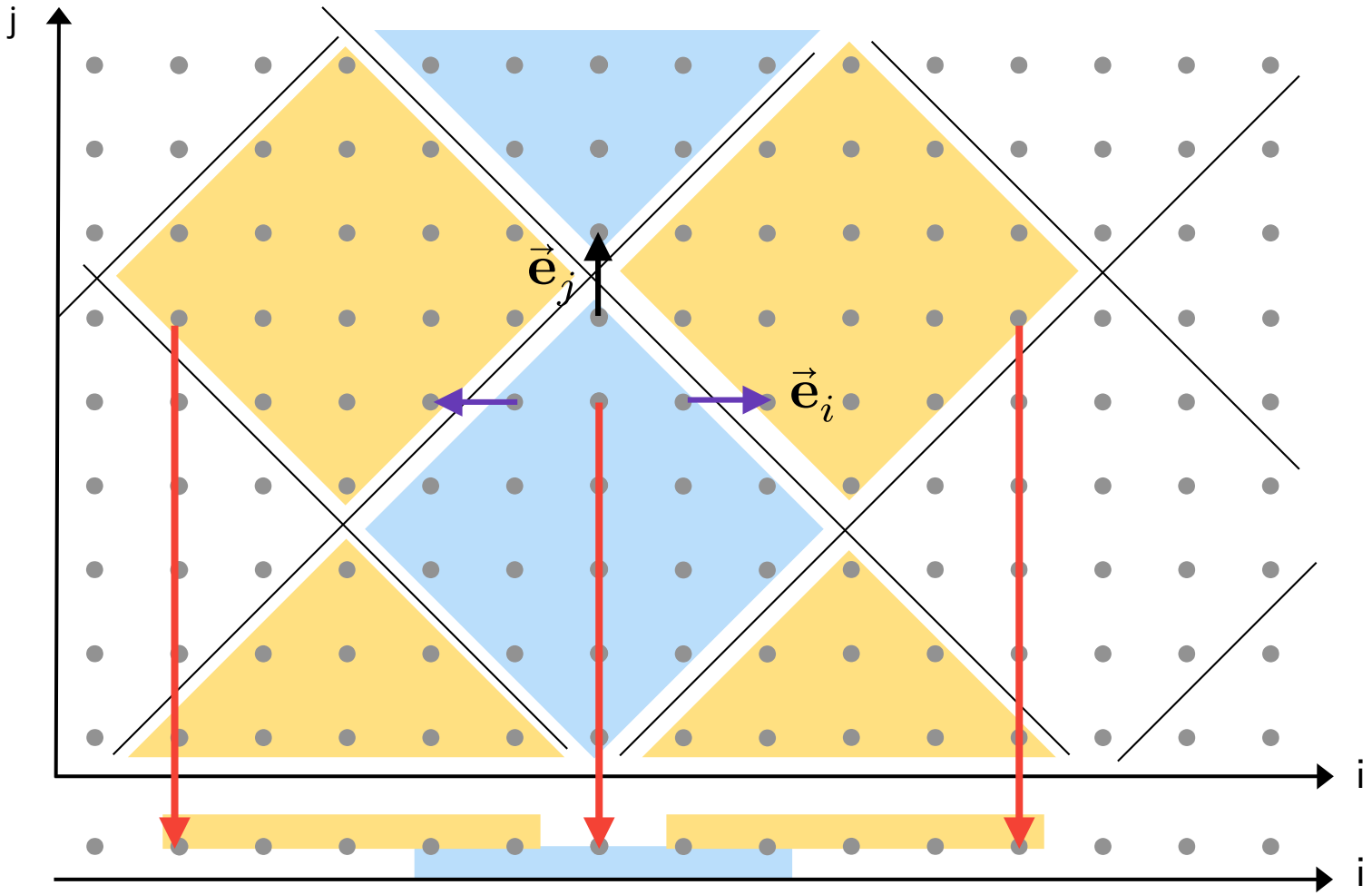
  → Would reduce the number of MARS (further bandwidth savings)

Outline

- Introduction and Motivation
- Background
- Proposed Solutions
  - Contiguous Allocation for Rectangular Tiles
  - Partitioning the Data Flow of Programs with Uniform Dependences
  - Contiguous, Compressed Memory Allocation for Uniform Dependence Programs
  - **Partitioning the Data Flow of Programs with Affine Dependences**
- Conclusions

Study on affine dependences and MARS

MARS does not handle affine dependences, e.g. broadcasts

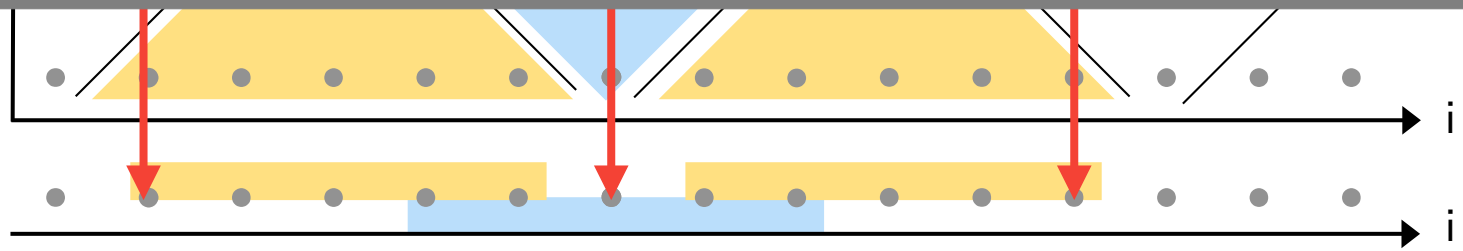


Broadcast dependence

Study on affine dependences and MARS

MARS does not handle affine dependences, e.g. broadcasts

Dependences	Consumers Enumerable	Partition Invariant
Uniform (≥ 1)	Yes	Yes
Single Affine	Yes	Yes
Multiple Uniformly Intersecting	Yes	Yes
Multiple Affine Same null space	Unknown	Unknown
Multiple Affine Multiple null spaces	No	No



Broadcast dependence

Study on affine dependences and MARS

MARS does not handle affine dependences, e.g. broadcasts

Dependences	Consumers Enumerable	Partition Invariant
Uniform (≥ 1)	Yes	Yes
Single Affine	Yes	Yes
Multiple Uniformly Intersecting	Yes	Yes
Multiple Affine Same null space	Unknown	Unknown
Multiple Affine Multiple null spaces	No	No

Flagship application: Deep Neural Nets (ML)

Broadcast dependence

Study on affine dependences and MARS

MARS does not handle affine dependences, e.g. broadcasts

Dependences	Consumers Enumerable	Partition Invariant
Uniform (≥ 1)	Yes	Yes
Single Affine	Yes	Yes
Multiple Uniformly Intersecting	Yes	Yes
Multiple Affine Same null space	Unknown	Unknown
Multiple Affine Multiple null spaces	No	No

Flagship application: Deep Neural Nets (ML)

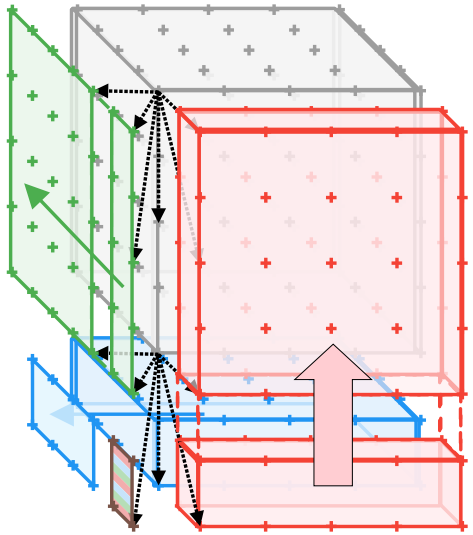
Ferry, C.; Derrien, S. & Rajopadhye, S. *An Irredundant Decomposition of Data Flow with Affine Dependences.* **14th International Workshop on Polyhedral Compilation Techniques** (IMPACT'24), 2024

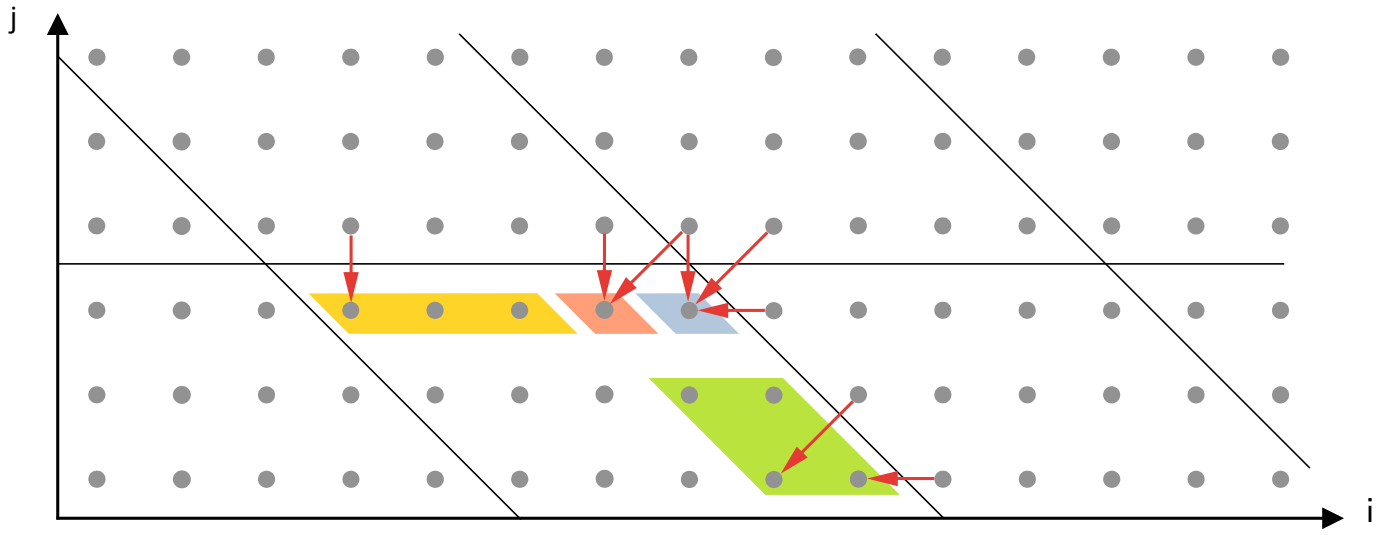
Outline

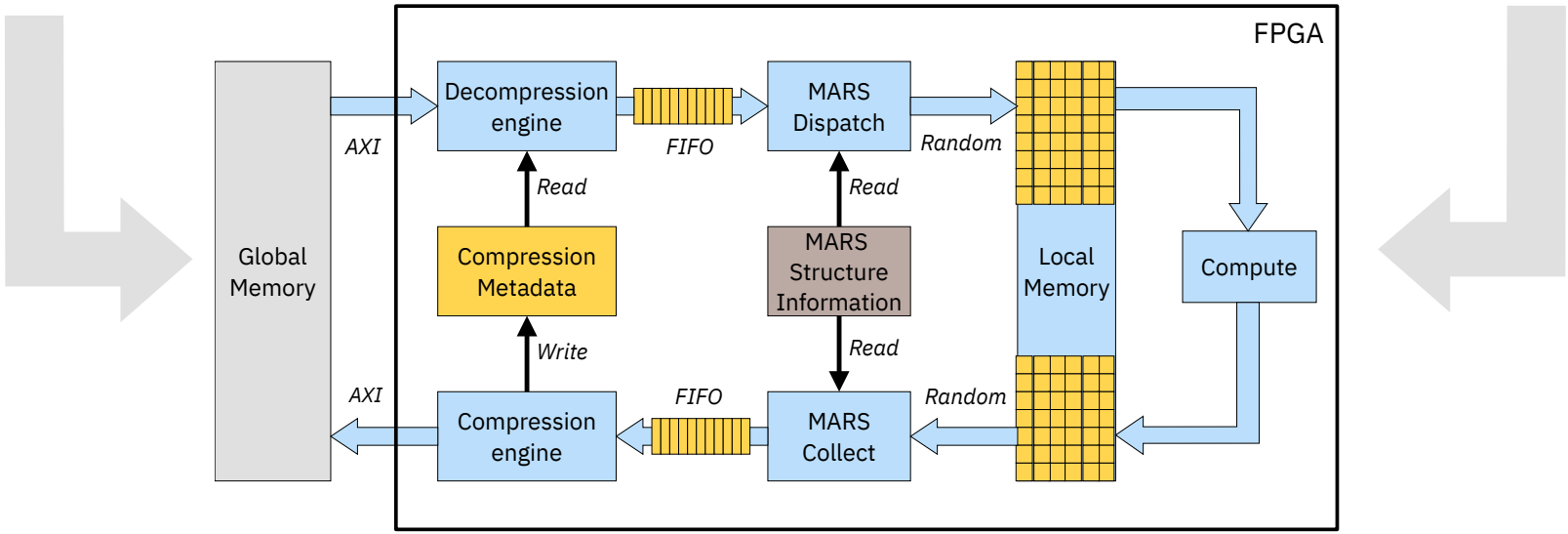
- Introduction and Motivation
- Background
- Proposed Solutions
  - Contiguous Allocation for Rectangular Tiles
  - Partitioning the Data Flow of Programs with Uniform Dependences
  - Contiguous, Compressed Memory Allocation for Uniform Dependence Programs
  - Partitioning the Data Flow of Programs with Affine Dependences
- **Conclusions**

Conclusions

Improving FPGAs Memory Performance via Polyhedral Compilation Methods

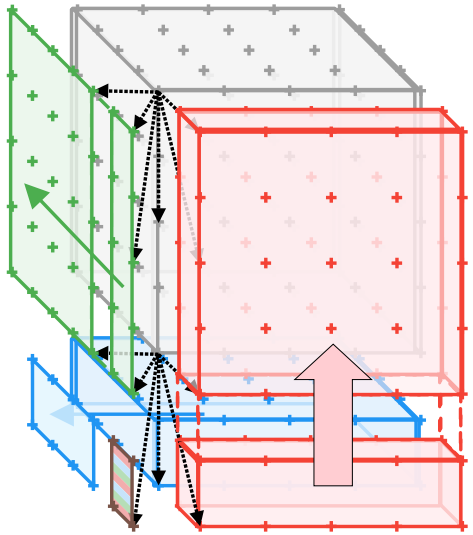


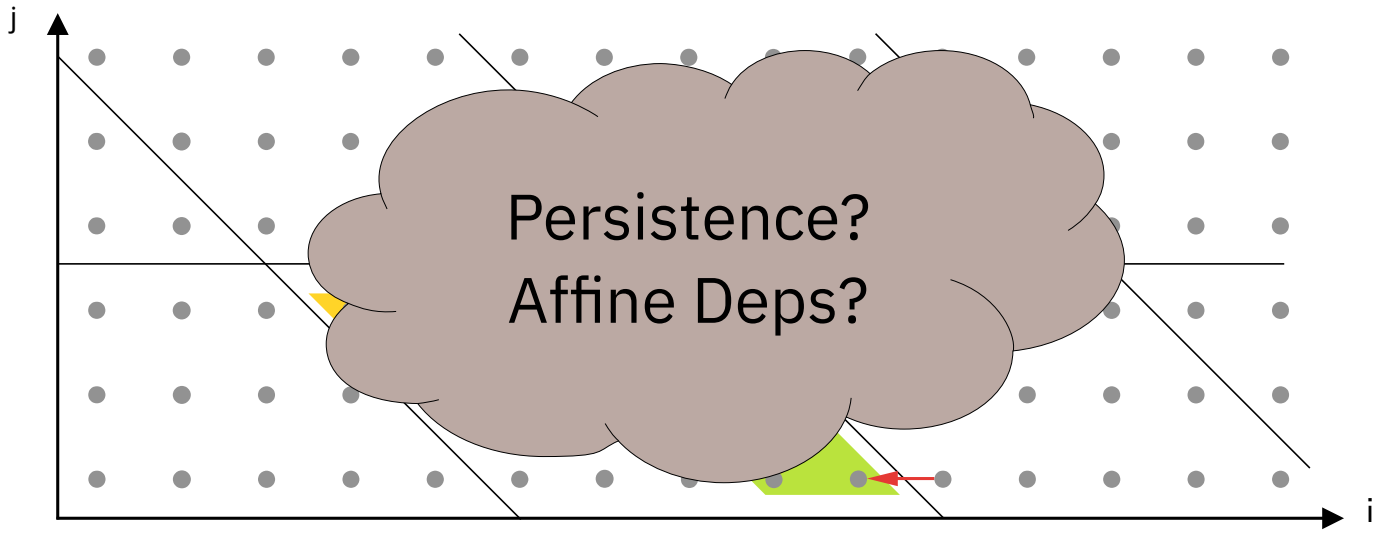


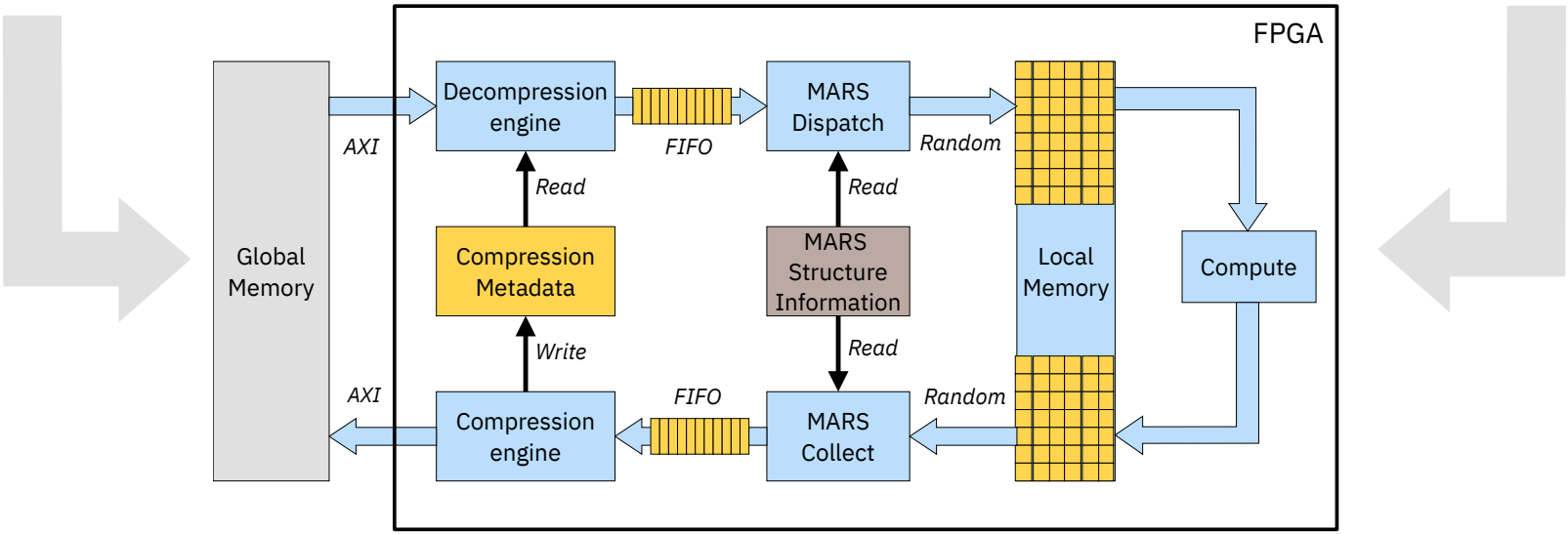


Conclusions

Improving FPGAs Memory Performance via Polyhedral Compilation Methods

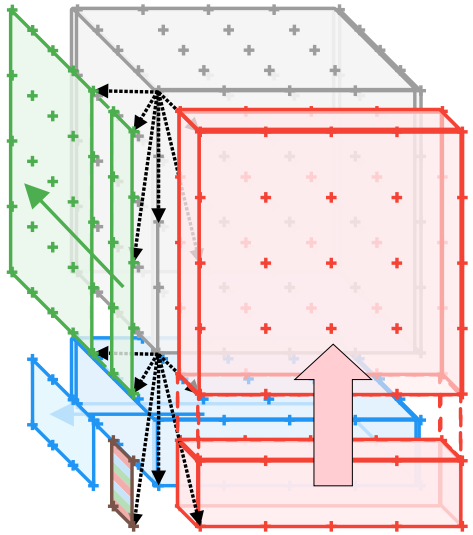


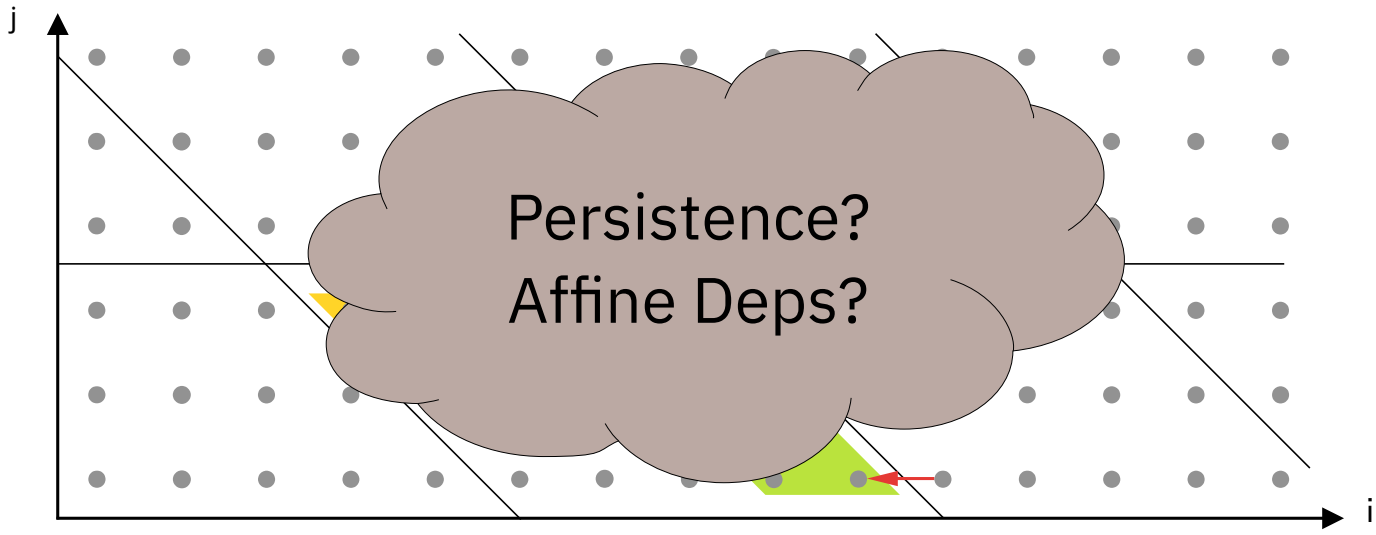


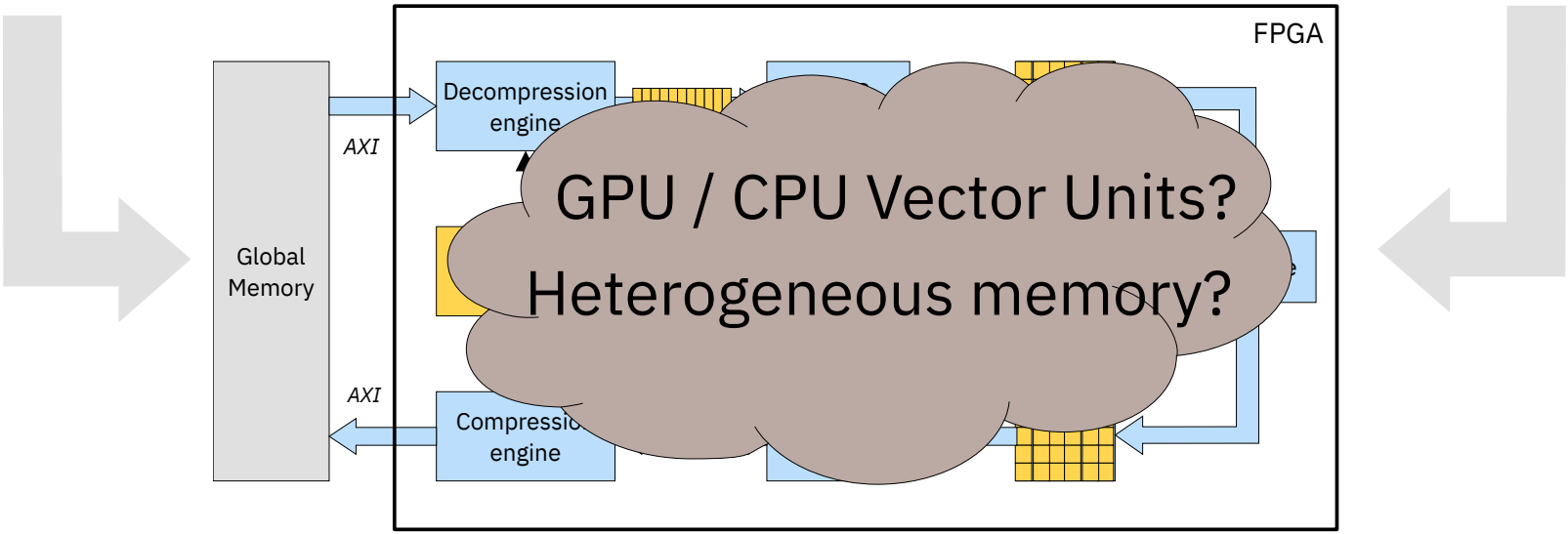


Conclusions

Improving FPGAs Memory Performance via Polyhedral Compilation Methods







Thank you