Maximal Atomic irRedundant Sets: a Usage-Based Dataflow Partitioning Algorithm

Corentin Ferry – Univ Rennes, CNRS, Inria, IRISA

Steven Derrien – Univ Rennes, CNRS, Inria, IRISA

Sanjay Rajopadhye – Colorado State University

Context of this work

- HPC Applications: Large volumes of data, low number of operations

  → **I/O intensive, low operational intensity**

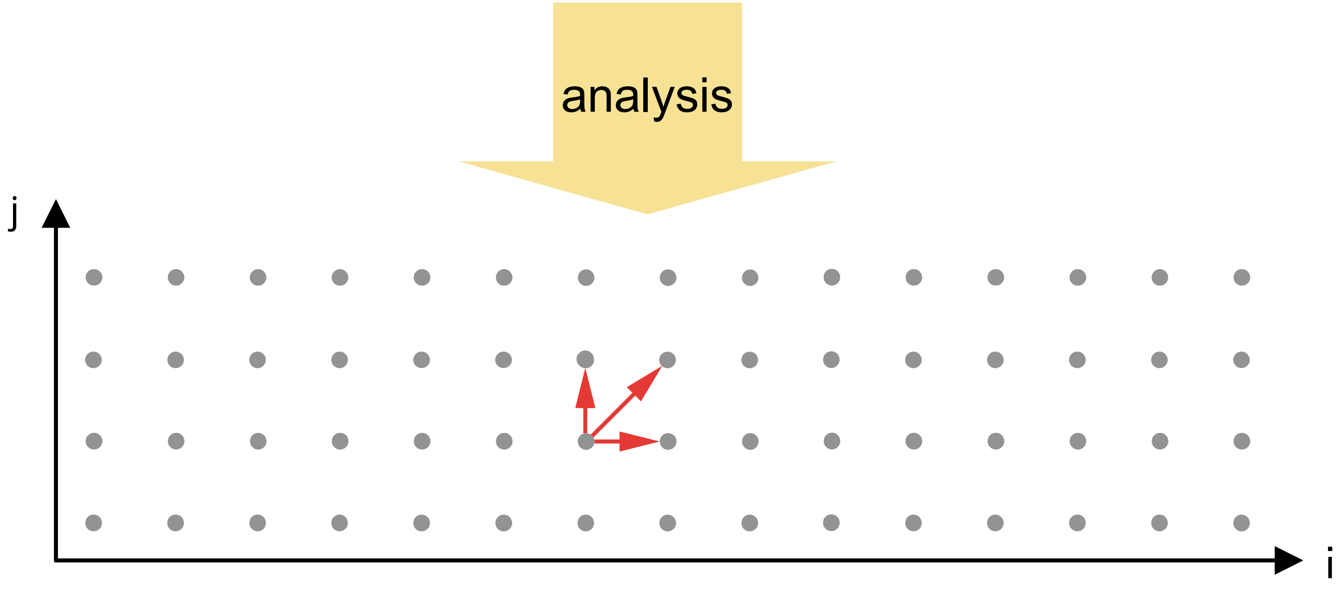
  - Significant *intrinsic* parallelism
  - « Splitting » the workload between nodes or accelerators
- **Communications** are extremely **expensive** (time + energy)
  - GPU / FPGA to host (PCIe) : thousands of cycles per transaction
  - MPI (network) : millions of cycles per transaction
  - Each transaction = **penalty**
- **Under-utilization** of transfer resource = **bottleneck**

Need to optimize host-accelerator communications

Programs we target

- **Computational kernels that admit a *polyhedral model***
  - 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);
  }
}
```

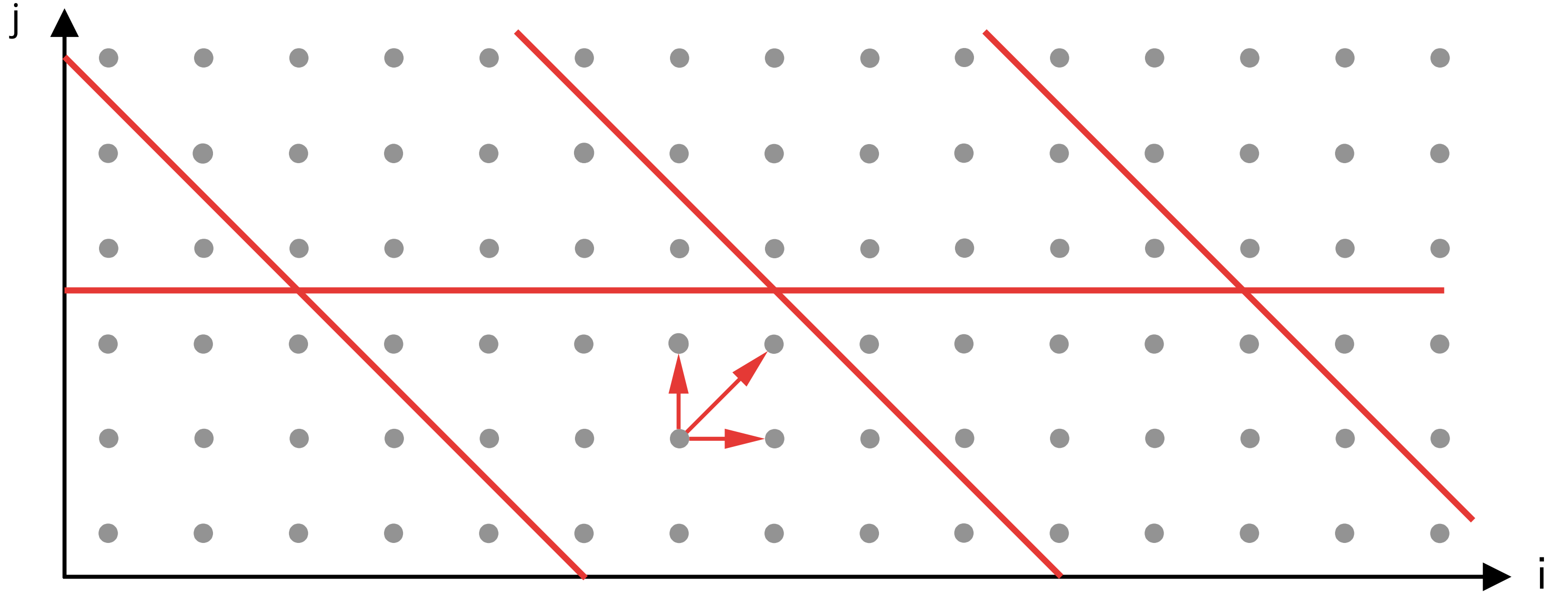


Smith-Waterman kernel iteration space and dependences

[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

Benefits of Loop Tiling

- **Tiling**
  - Already applied, e.g. for *temporal* locality & parallelism
- **Uniform dependences (vectors)**
  - Inter-tile communications = same across all tiles

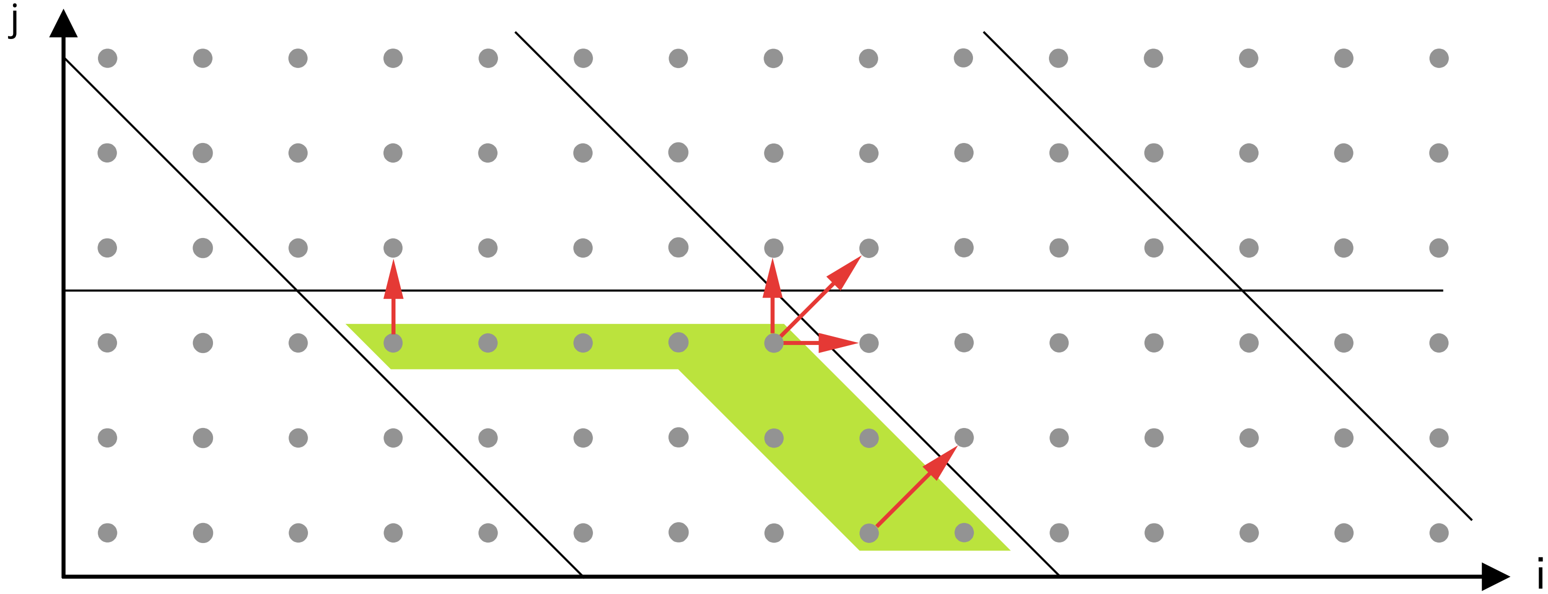


Tiling of Smith-Waterman kernel iteration space

We seek to improve spatial locality as well

Inter-tile communications : flow-in/out

- Intermediate **values used by other tiles** → *communications*
- Bondhugula (2013) [2] : communicated sets = ***flow-in / flow-out* sets**
- Iterations consumed in another tile = **Flow-out**



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

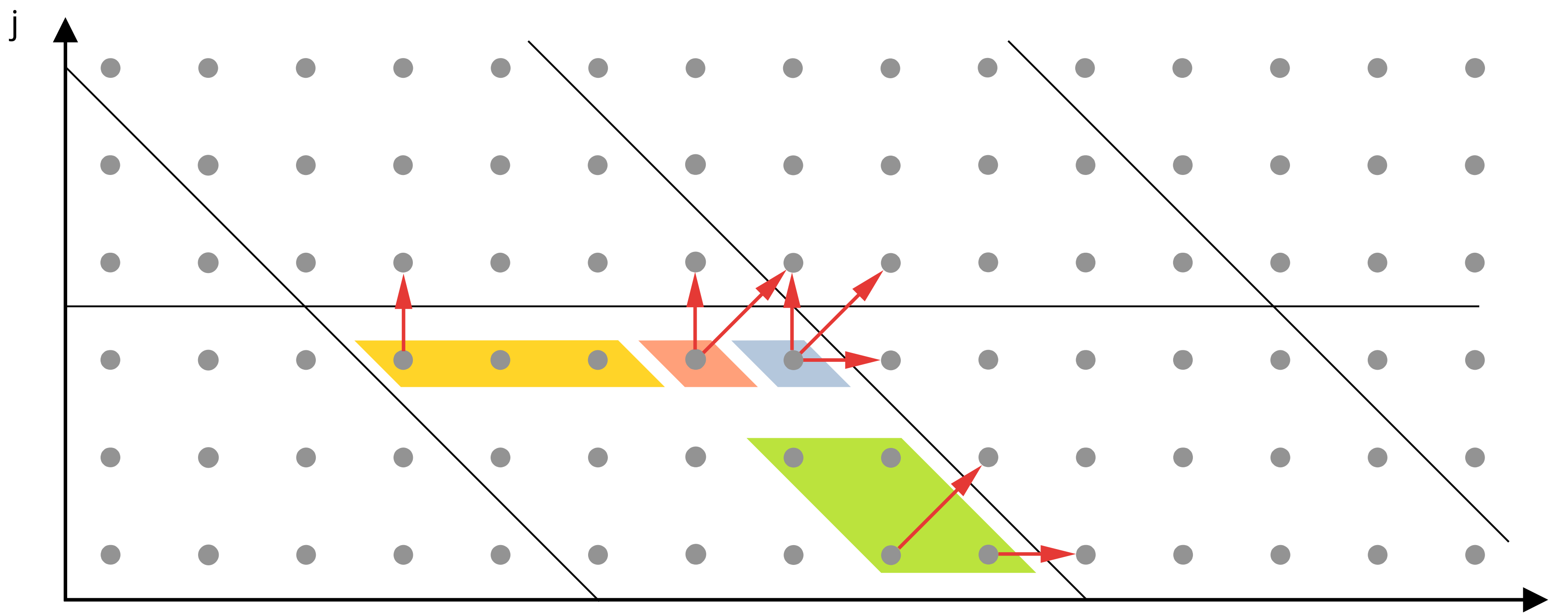
Each Consumer Tile Needs Parts of Flow-out

[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

What is a  ?

Maximal Atomic irRedundant Sets

- Single-Producer (SP) : all iterations in a MARS come from *one* tile
- All-Consumed (AC) : MARS are *entirely* consumed by every consumer tile



MARS guarantee the absence of read + write redundancy

State of the Art

- Decomposition of **Inter-Tile Communications**:
  - Datharthri et al., 2013 [3]: Flow-In/Flow-Out partitioning
    → MARS = one specific case, **static determination at compile time**
  - Zhao et al, 2021 [4]: partitioning + layout
    → MARS = **« generalization » to uniform dependences**
- Memory Layout for **Host-Accelerator Communications**:
  - Ozturk et al., 2009 [5]: data tiling + compression
    → MARS = **finer-grain data breakdown** amenable to compression
- Allocation from a Polyhedral Model:
  - Yuki and Rajopadhye, 2013 [6]: lower memory footprint with Uniform Dependences
    → MARS = **trade footprint for 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

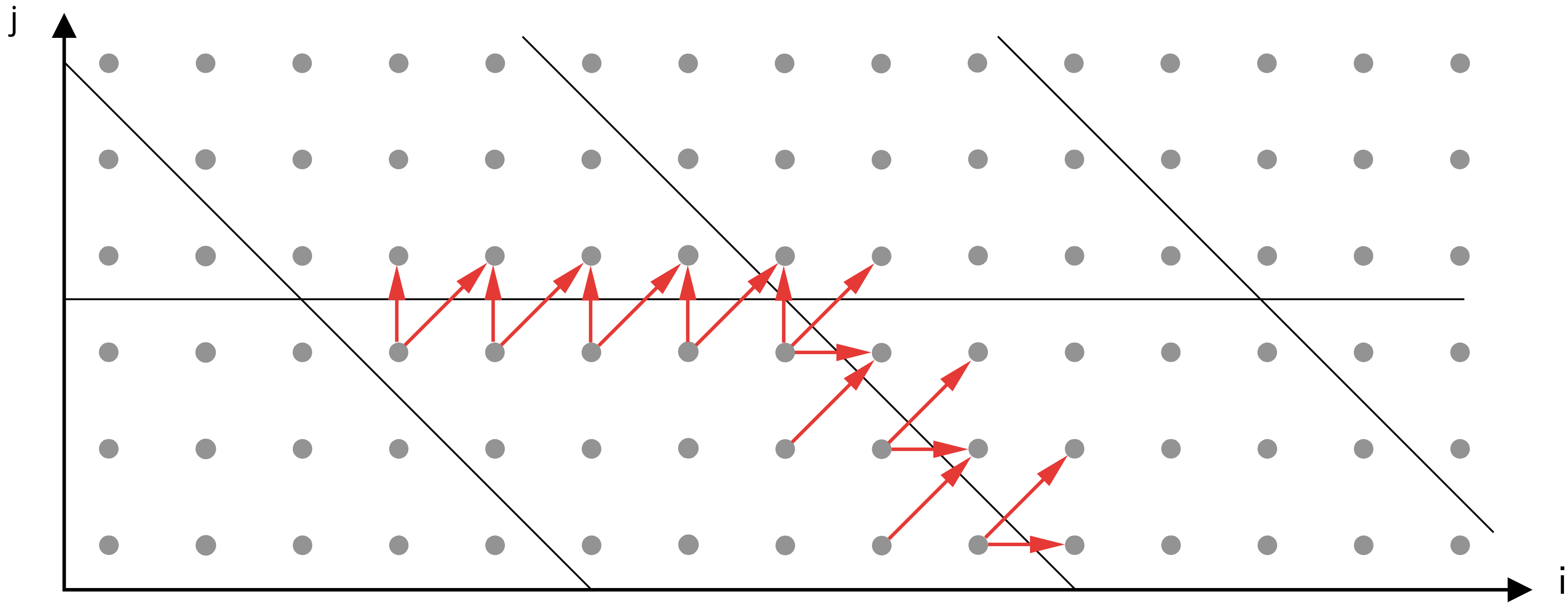
[6] Yuki, T. & Rajopadhye, S. *Memory allocations for tiled uniform dependence programs* IMPACT 2013, 2013, 13

Outline

- **Construction of MARS**
  - Core Notion : « dependence crossing hyperplane »
  - Illustrated Construction
- **Implementation results**: Examples and Analysis
- **Discussion**: Possible uses

Constructing the flow-out

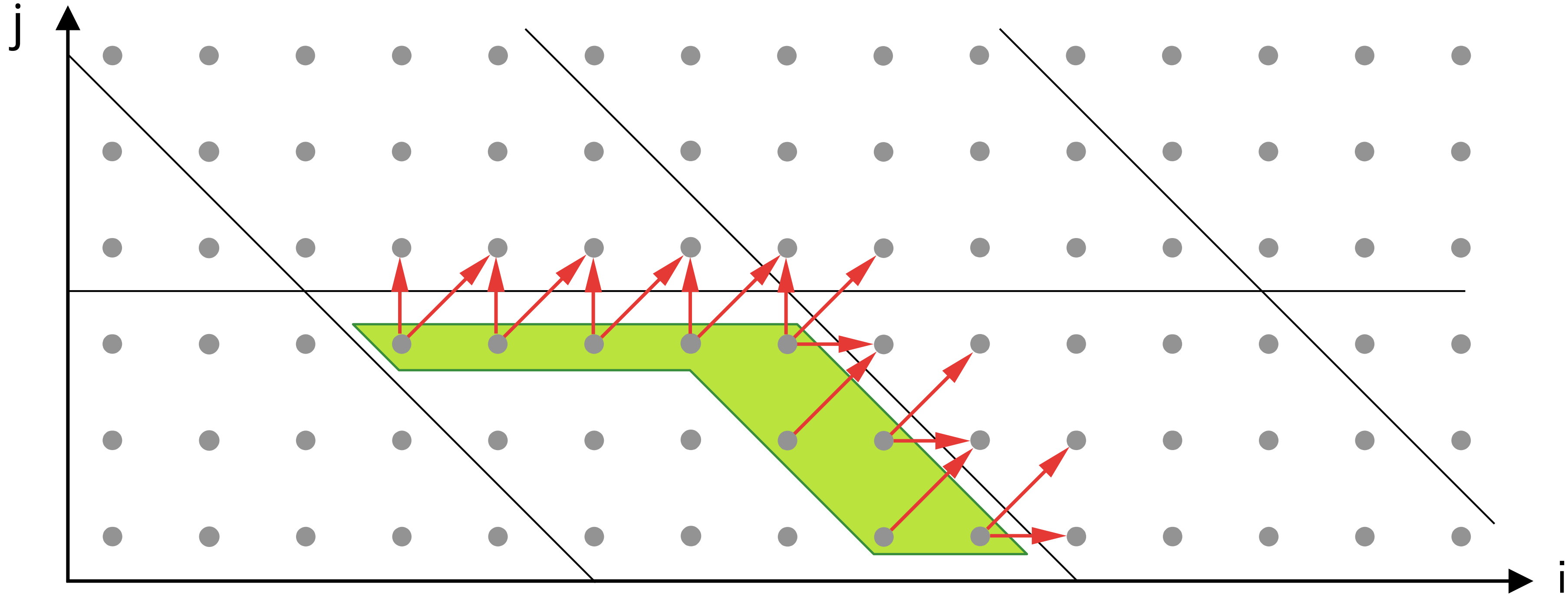
- Inter-tile dependence : *some* dependence **crosses *some* tiling hyperplane**



Expressible using hyperplane standard equation

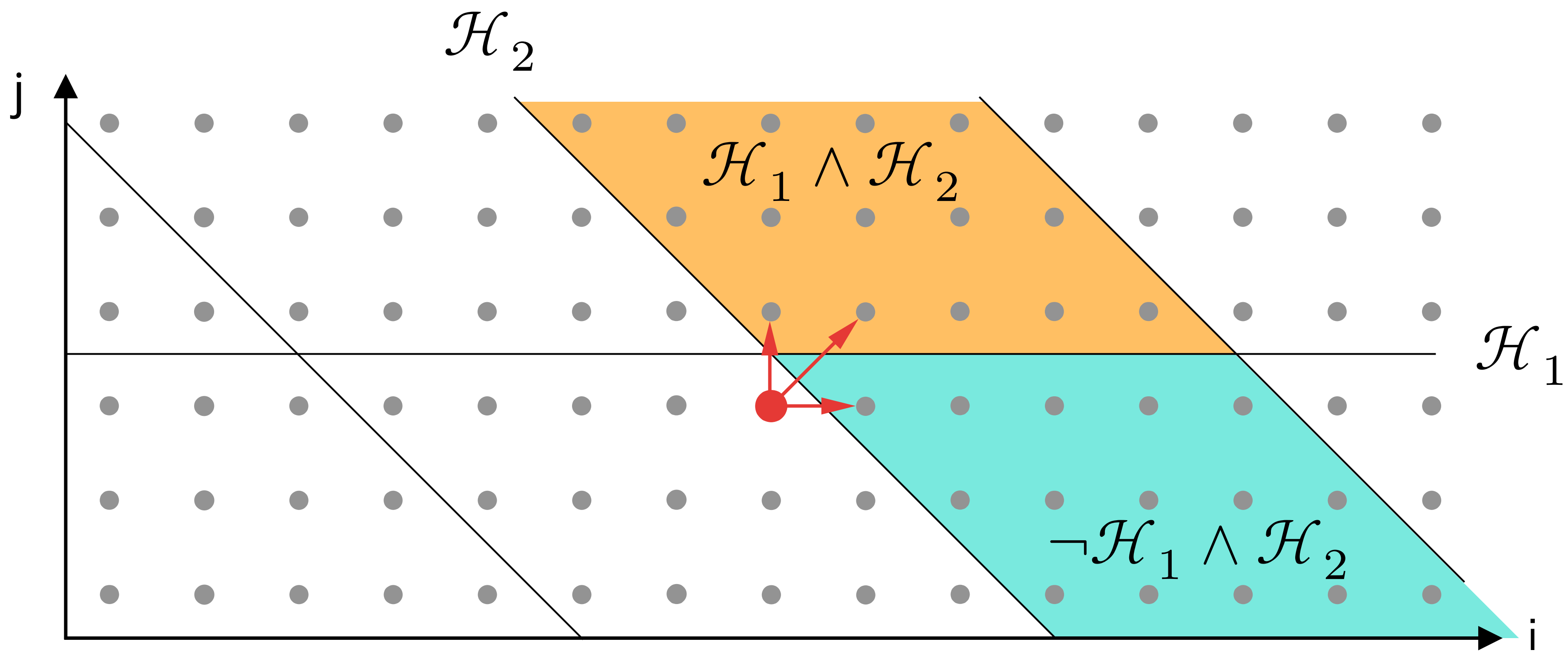
Constructing the flow-out

- **Flow-out :** *iteration points* such that translating by *some* dependence vector crosses *some* tiling hyperplane



Consumer Tiles

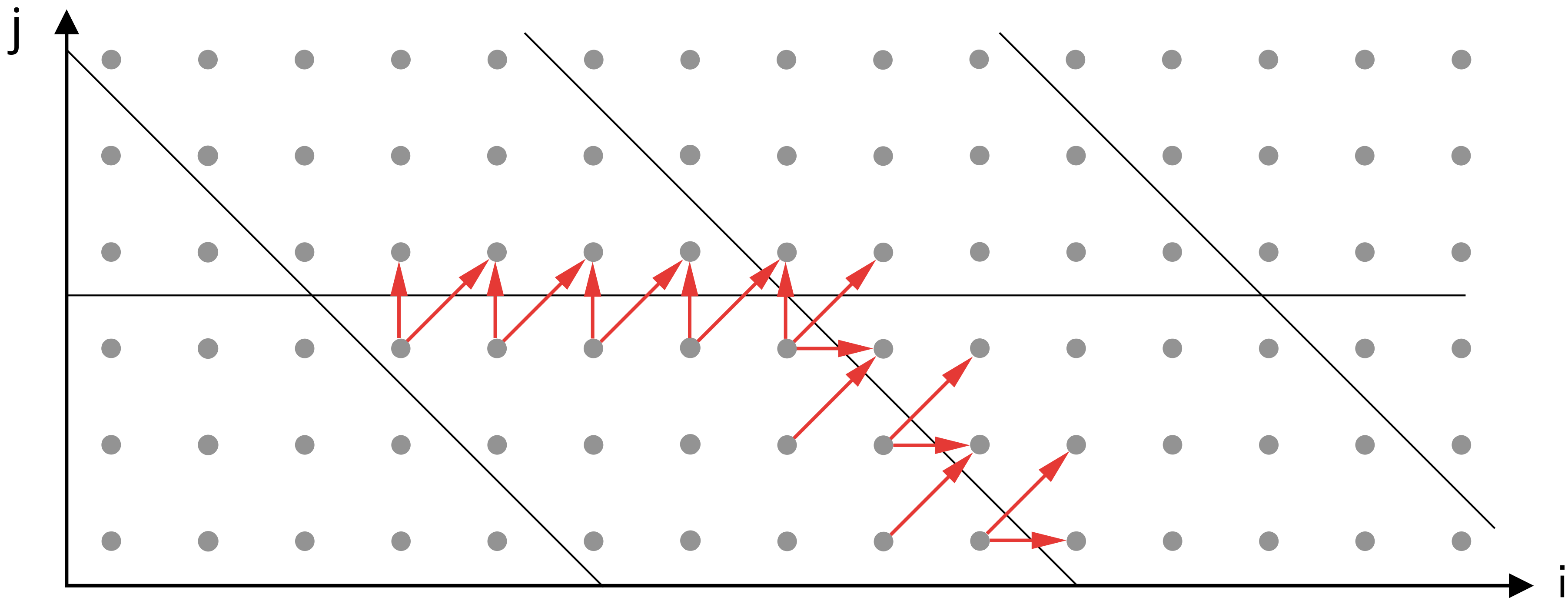
- Defined per iteration point
- Some dependence crosses **exactly select hyperplanes** to consumer tile



Iteration point in red has 2 consumer tiles

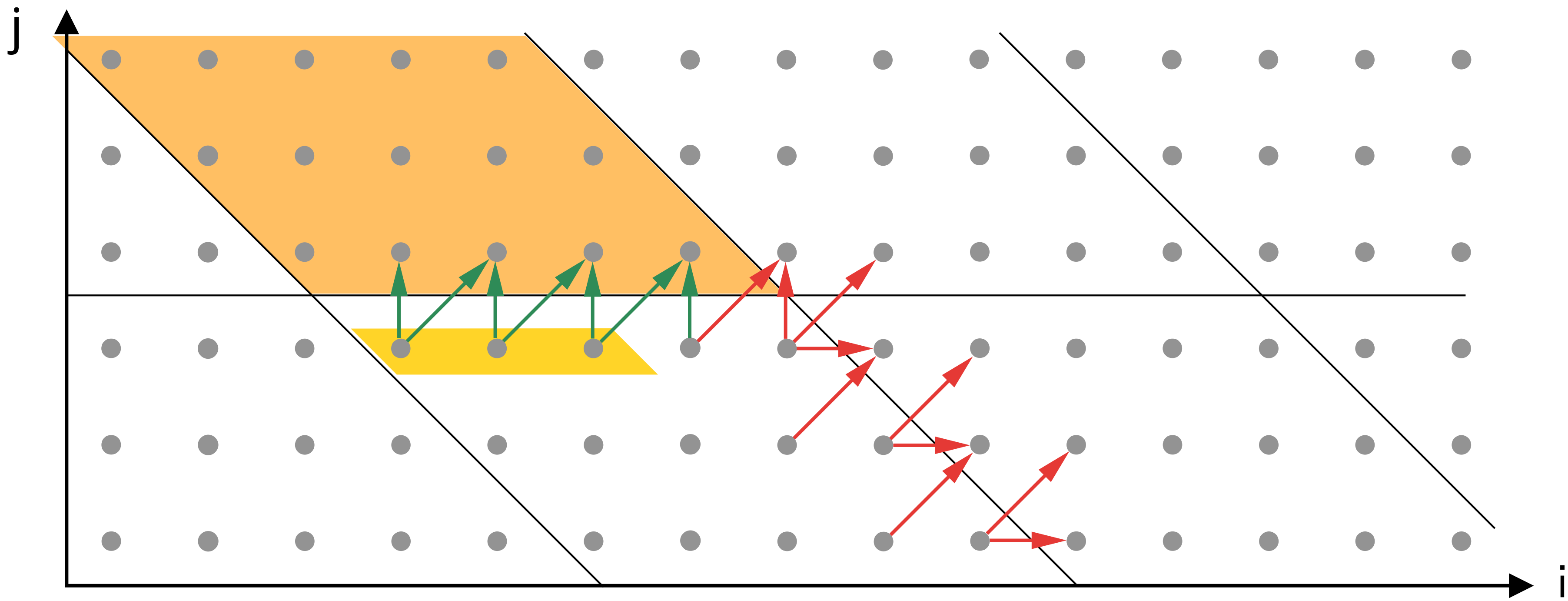
Constructing MARS

- MARS : *iterations* consumed **exclusively *by specific* consumer tiles**



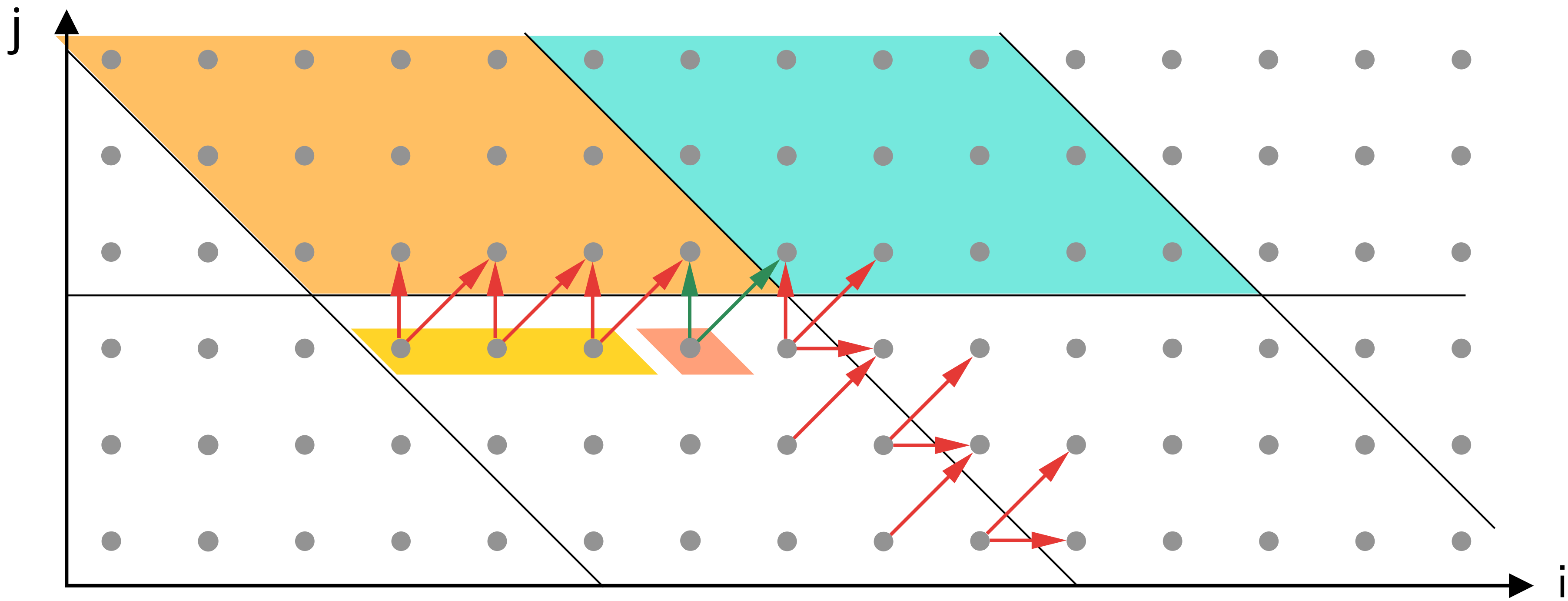
Constructing MARS

- MARS : *iterations* consumed **exclusively *by specific* consumer tiles**



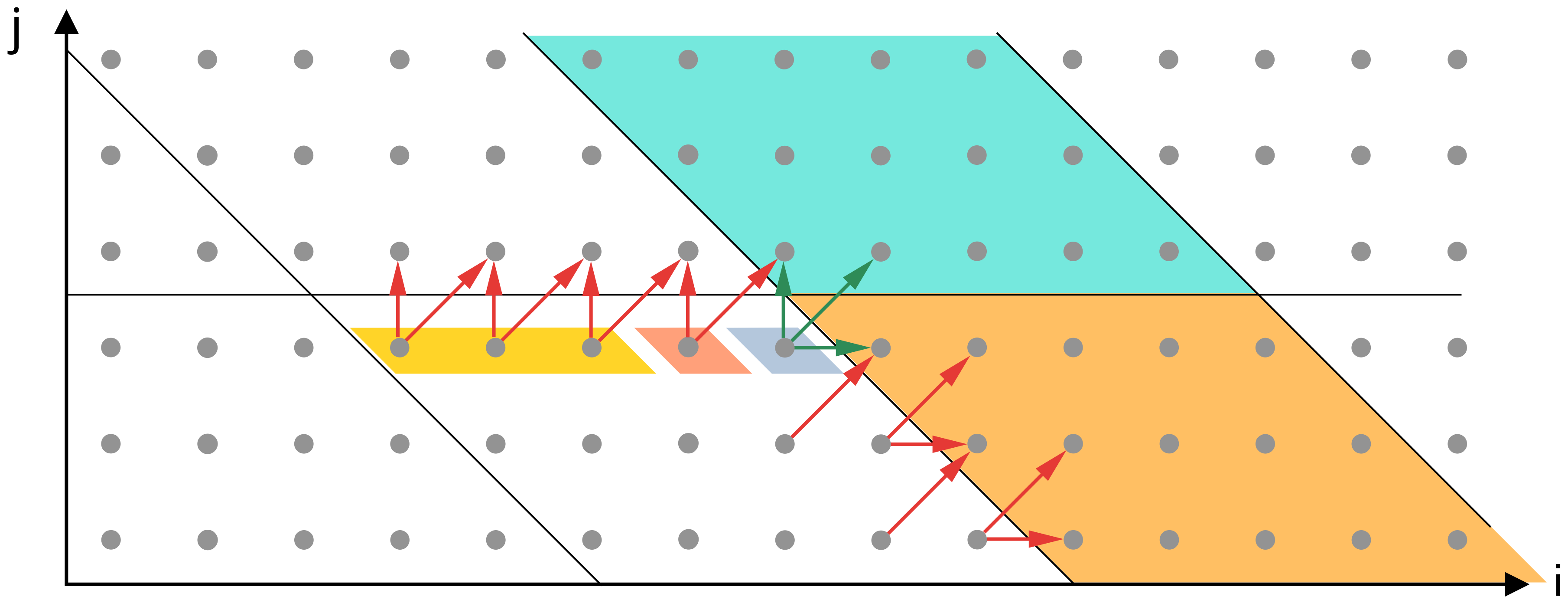
Constructing MARS

- MARS : *iterations* consumed **exclusively *by specific* consumer tiles**



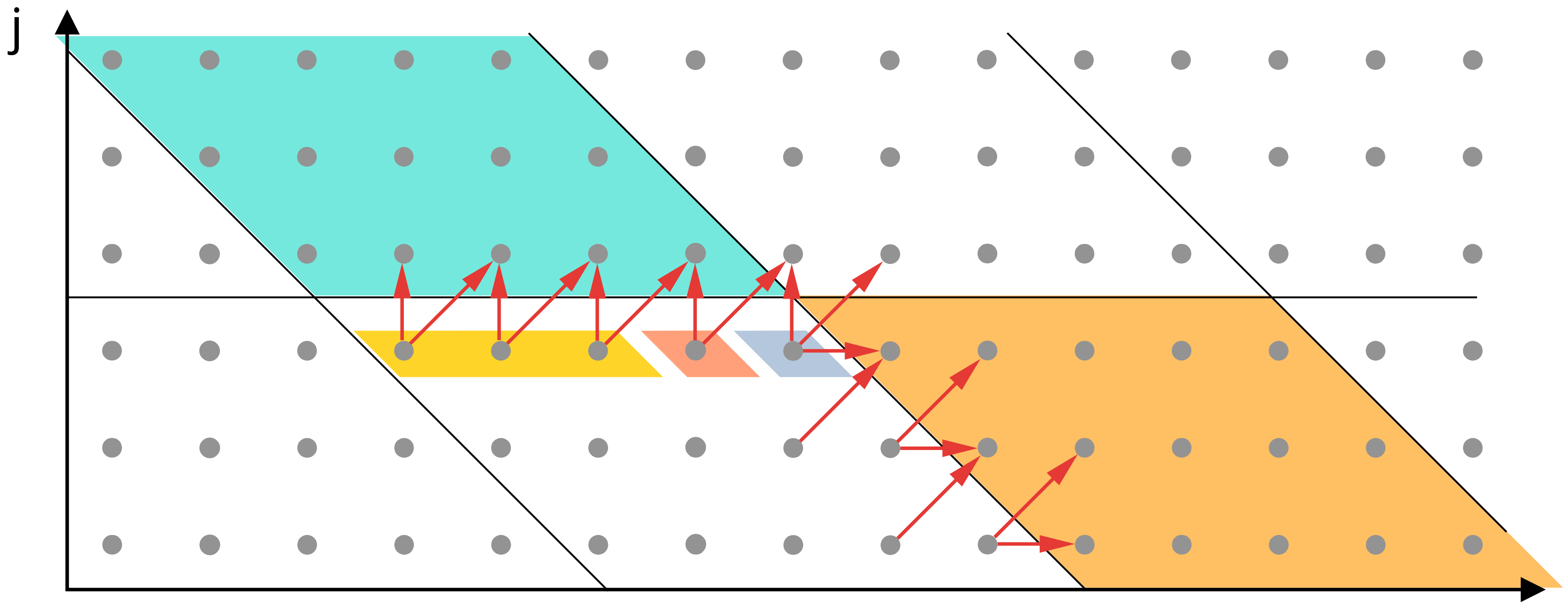
Constructing MARS

- MARS : *iterations* consumed **exclusively *by specific* consumer tiles**



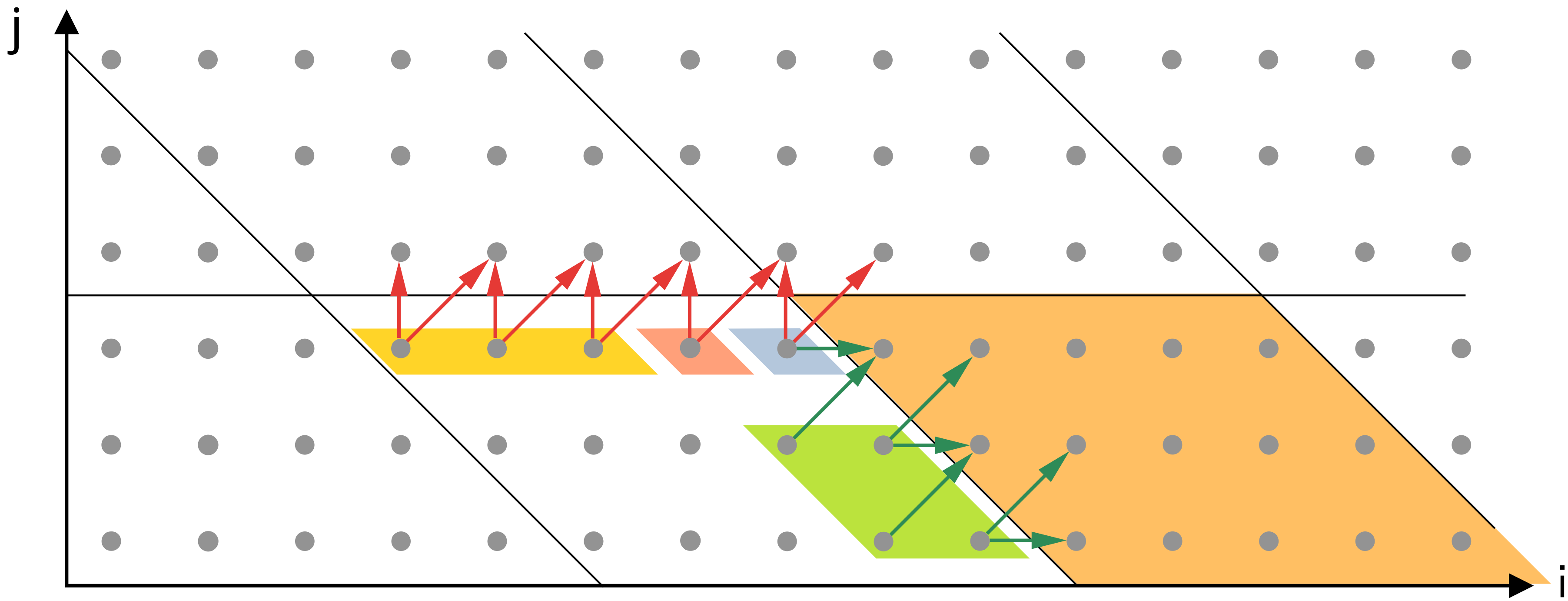
Constructing MARS

- MARS : *iterations* consumed **exclusively *by specific* consumer tiles**



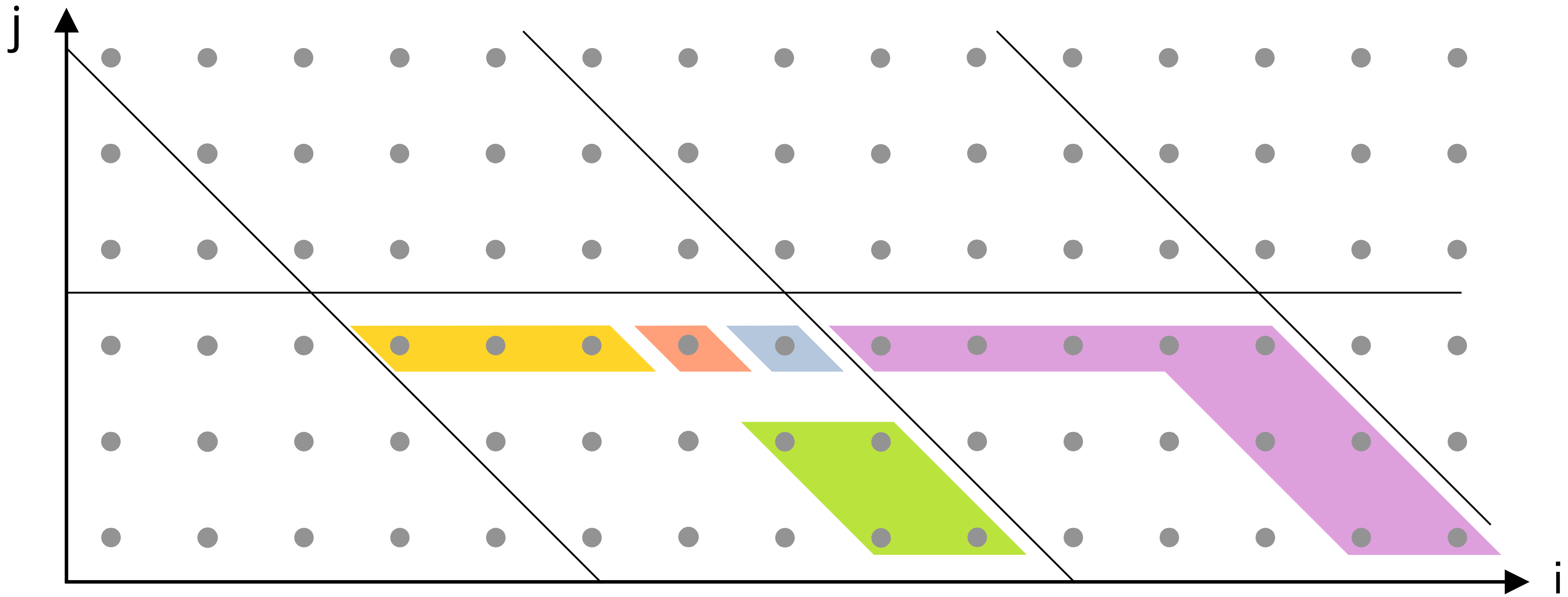
Constructing MARS

- MARS : *iterations* consumed **exclusively *by specific* consumer tiles**



Constructing MARS

- MARS : *iterations* consumed **exclusively *by specific* consumer tiles**



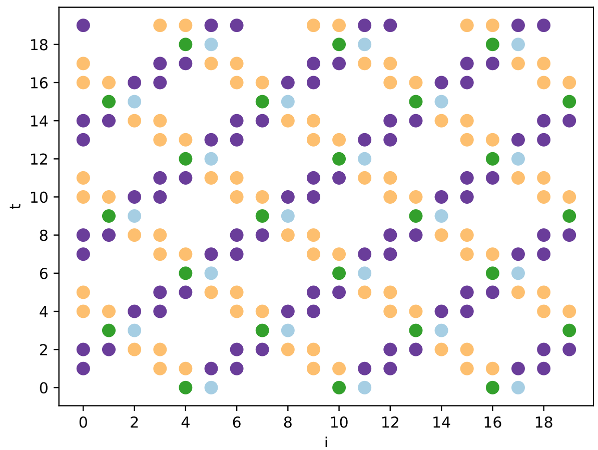
Union of MARS = Flow-Out

Computation Performance

- MARS calculator **available online** (https://github.com/cferr/mars.git)
  - Jupyter notebook using ISLPy
- Runtime :
  - 1-5 seconds if ≤ 3 tiling hyperplanes
  - **2 hours** for 4 tiling hyperplanes
  - Cause : exhaustive exploration of power set *of power set* (**2^(2^n)** elements)
- Recursive implementation : **~10 seconds** for 4 tiling hyperplanes
  - Not yet publicly released

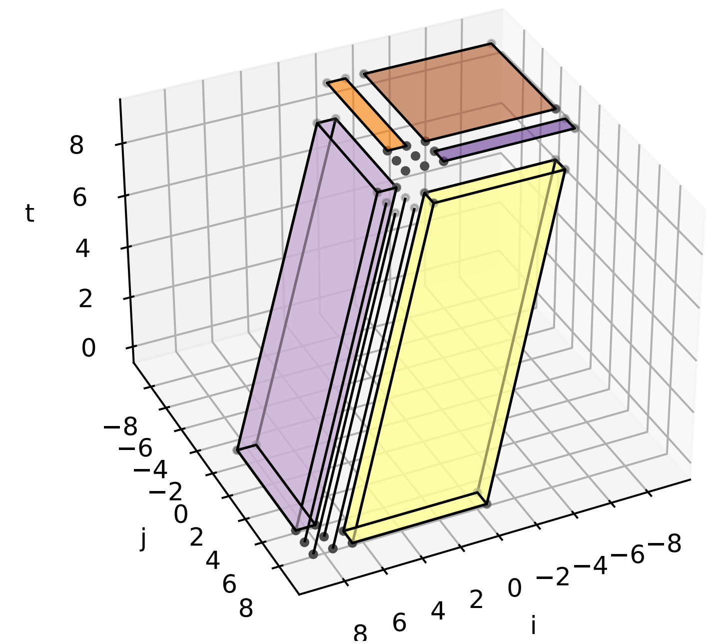
We have alleviated MARS calculation complexity

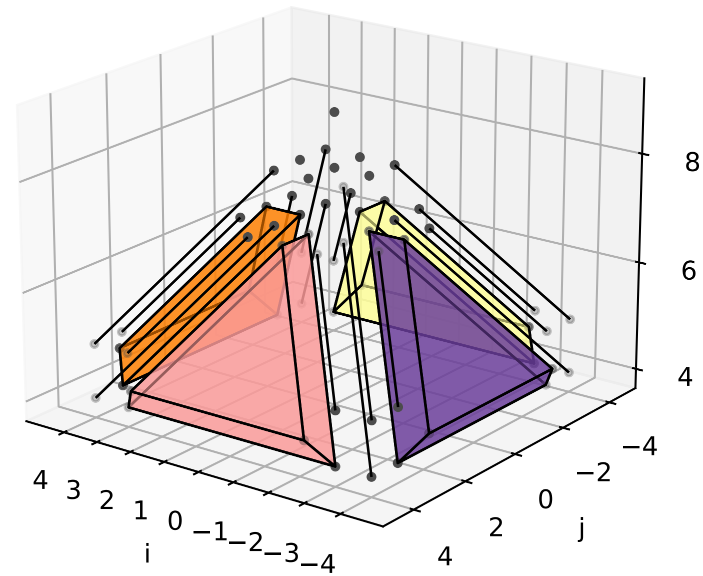
MARS supports 1D, 2D, 3D spaces…



Jacobi 1D, diamond tiling

Jacobi 2D, skewed tiling





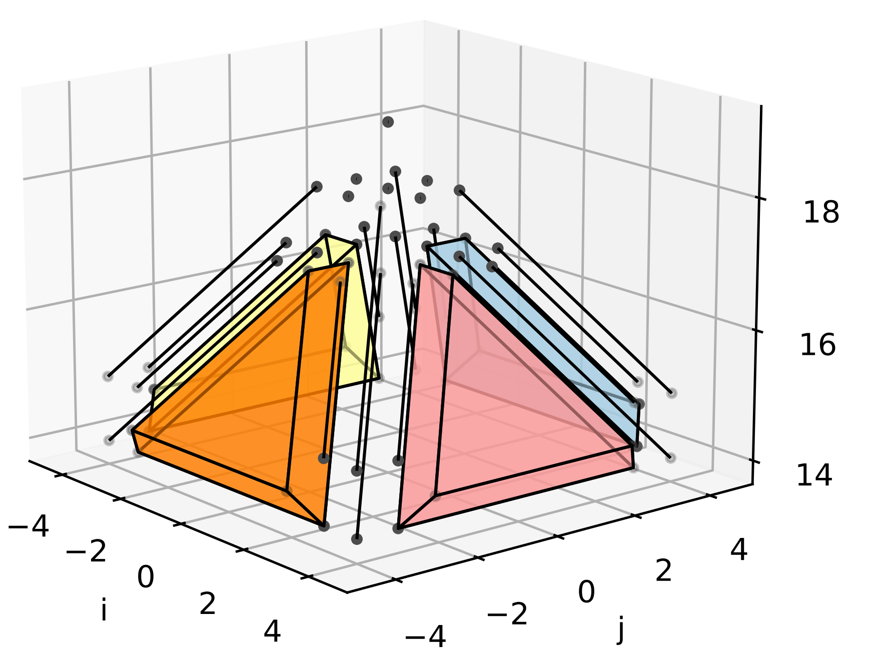
Jacobi 2D, « diamond » tiling

… and non-canonical tiling hyperplanes !

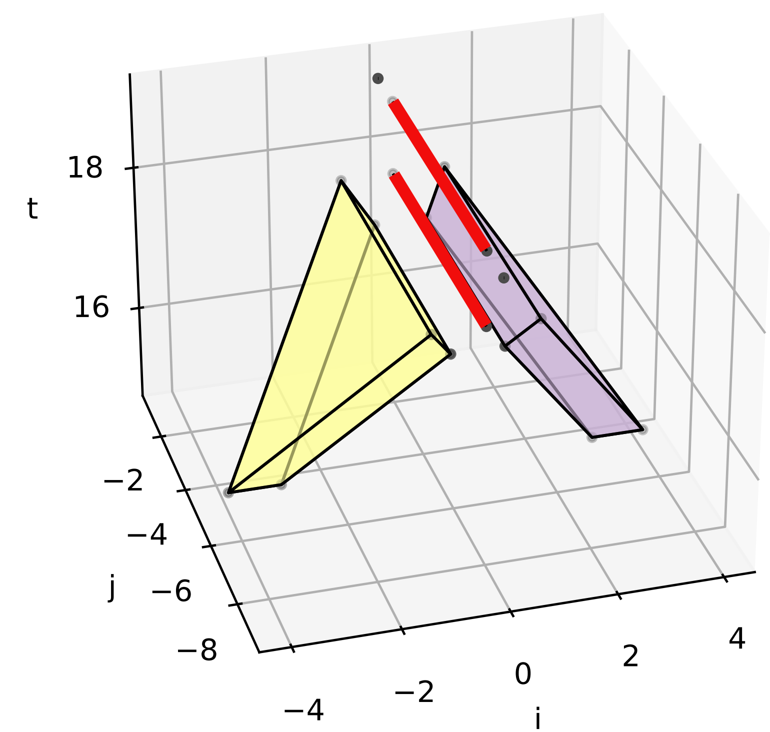
MARS supports non-LI hyperplanes

- Tiling hyperplanes aren't **L**inearly **I**ndependent (4 hyperplanes for a 3D space)
- Example : Jacobi 2D – Diamond, **3 tile shapes**

MARS Flow-Out (jacobi2d_d) : k4 = k1 - k2 + k3

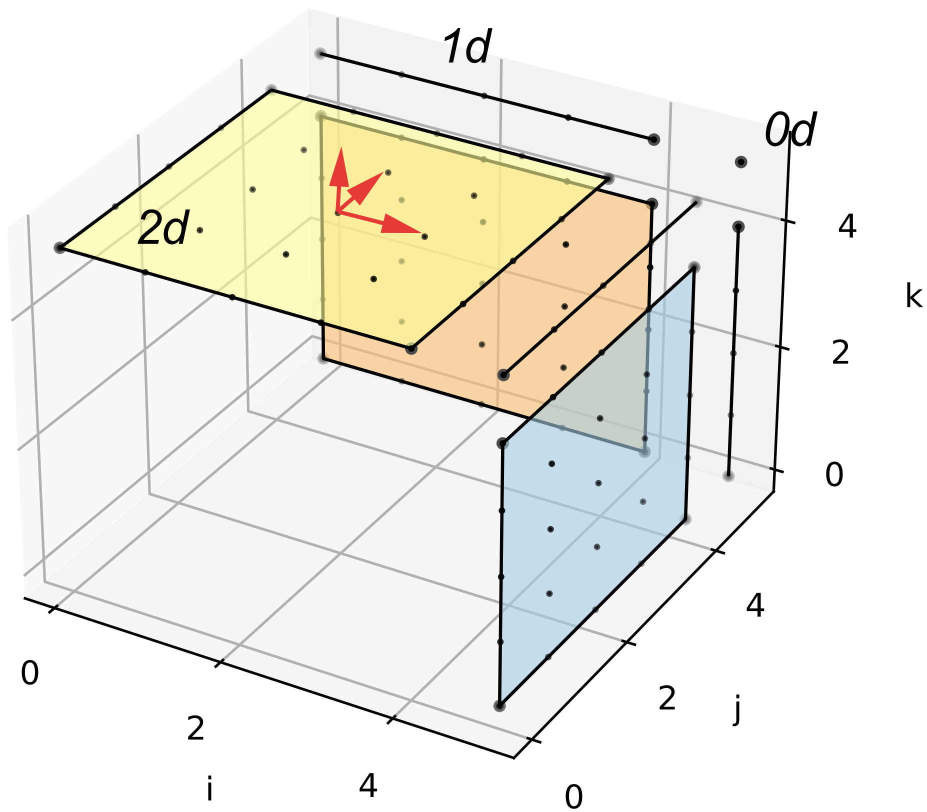


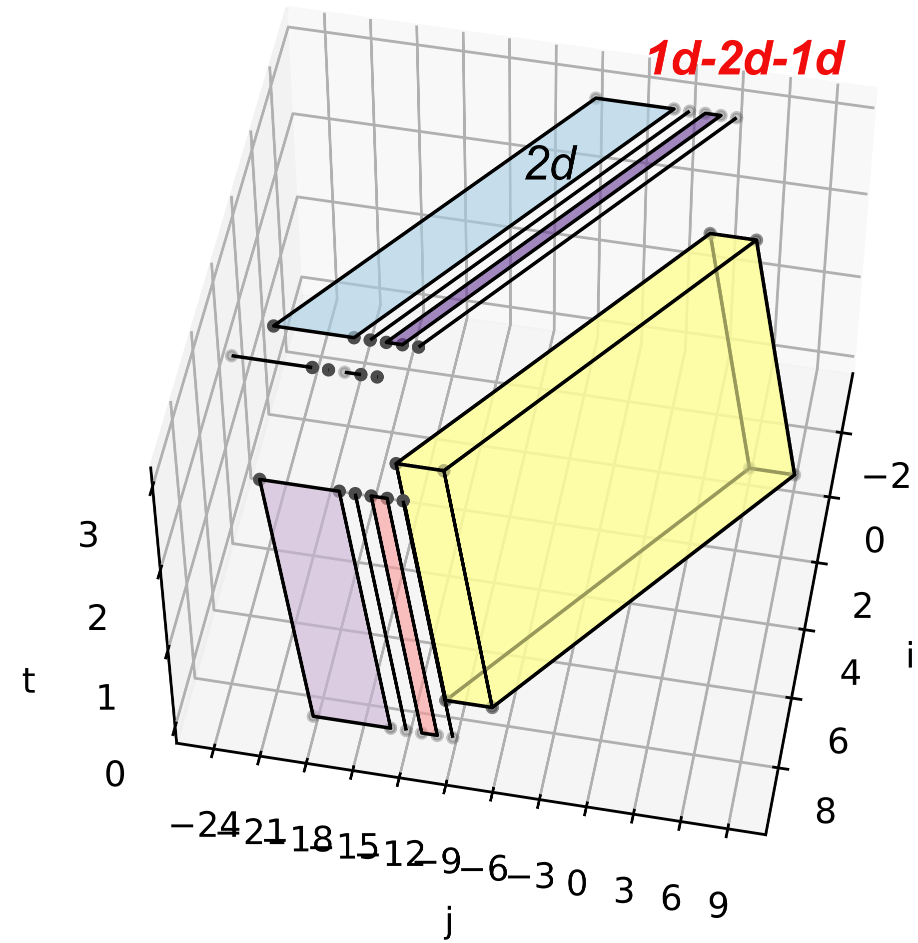
MARS Flow-Out (jacobi2d_d) : k4 = 1 + k1 - k2 + k3



Each tile shape produces different MARS

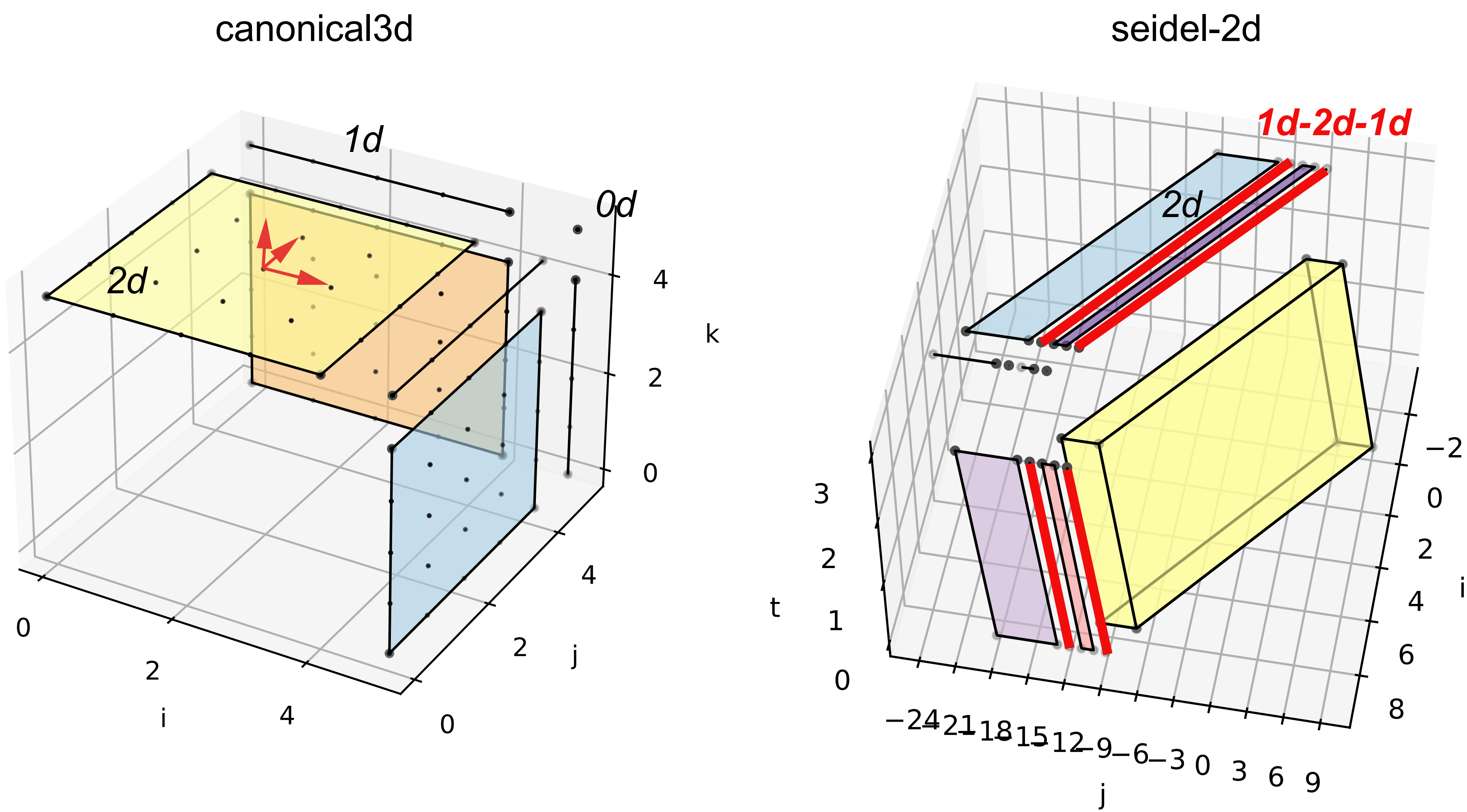
Counterintuitive Observation...





MARS dimensionality does not necessarily decrease as we get closer to intersections of edges

Counterintuitive Observation...



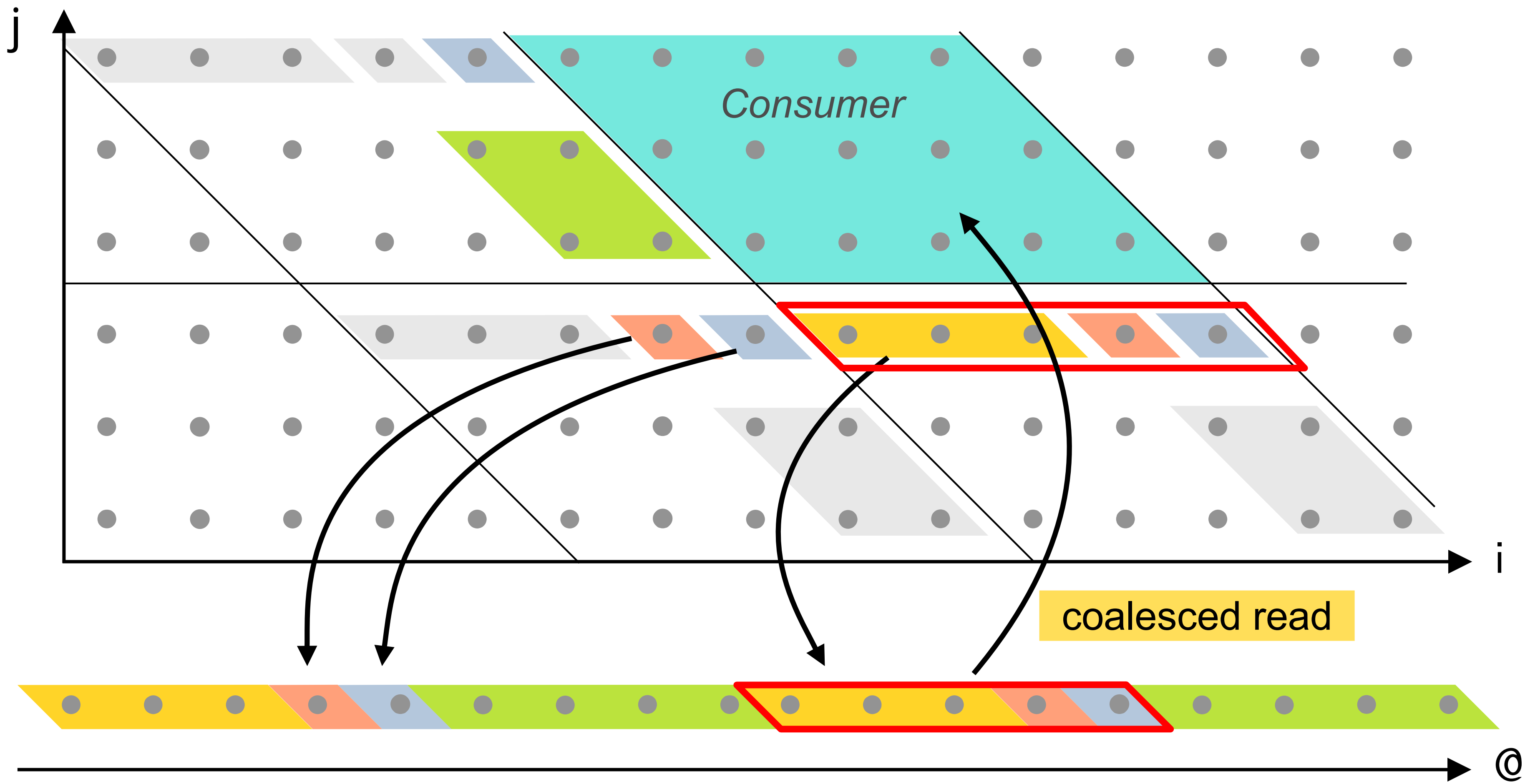
MARS dimensionality does not necessarily decrease as we get closer to intersections of edges

Applications of MARS

- **Memory allocation for FPGA accelerators**
  - Work In Progress : automatically derive a data layout minimizing read transactions
- **Compression**
  - Along with data layout → increase the effective bandwidth (amount of useful data transmitted over the bus) thanks to MARS' irredundancy
- **Fault tolerance**
  - Compute a checksum on each MARS. If error → the producer tile (known) is to be re-executed.

MARS for FPGA-Host communications

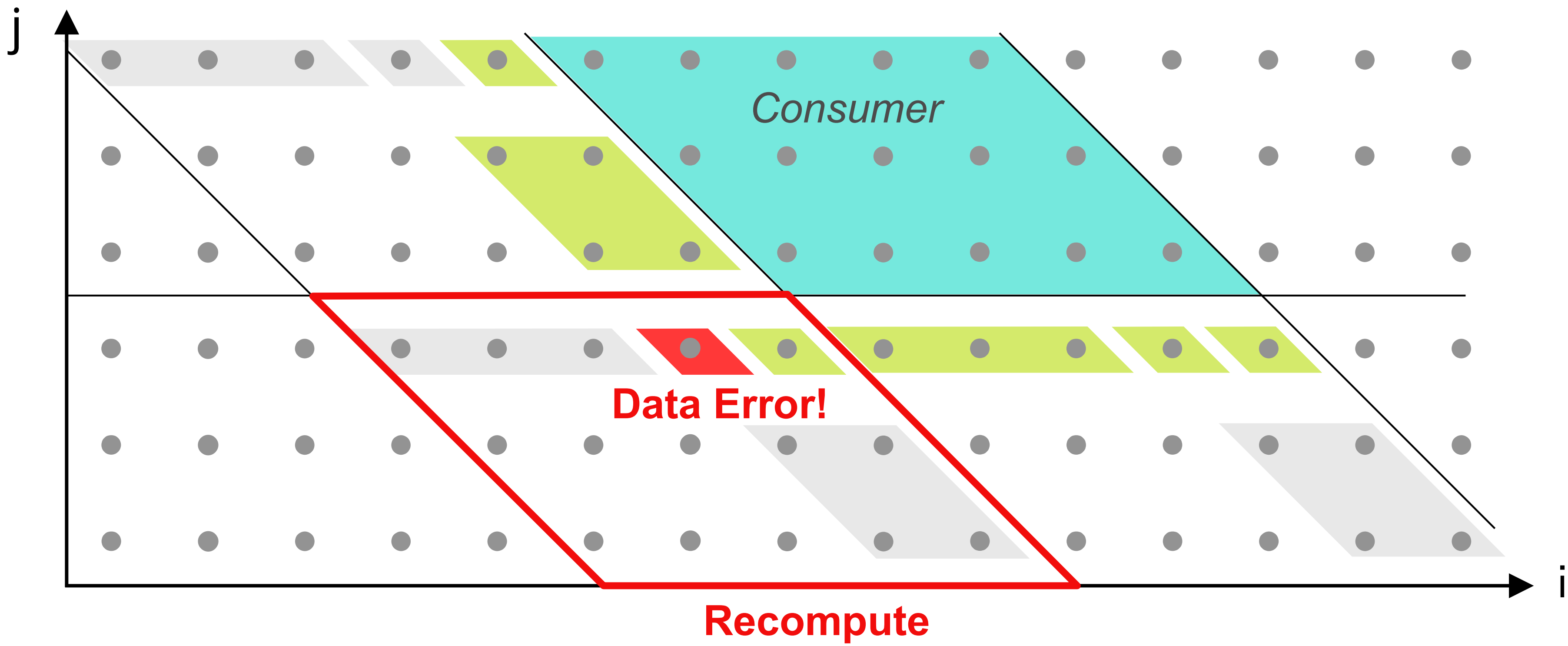
- Find MARS layout in memory **minimizing read time**



Formulated as an LP optimization problem - WIP

MARS for Error Detection

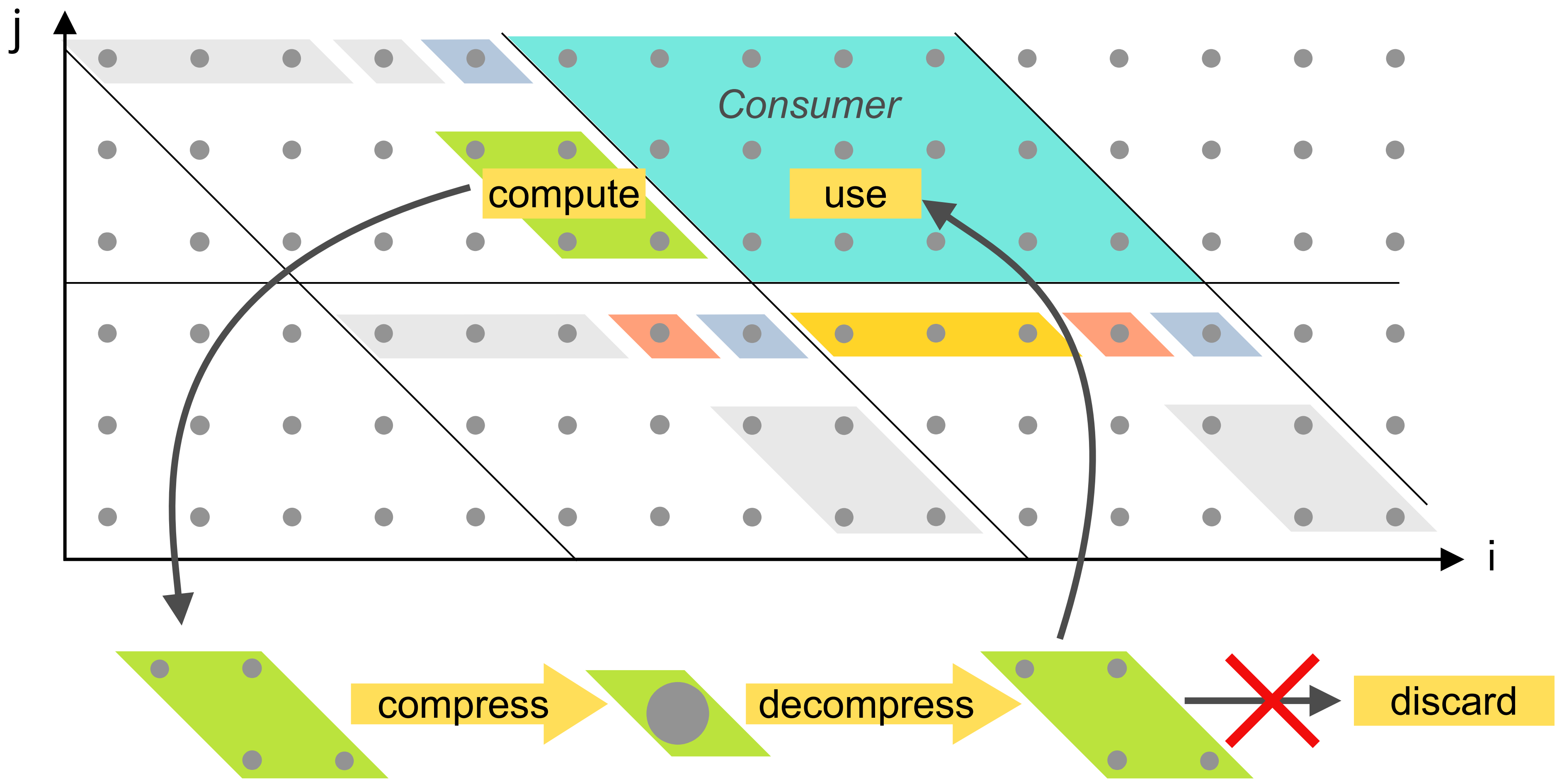
- Checksum MARS to determine if **data from producer tile has errors**



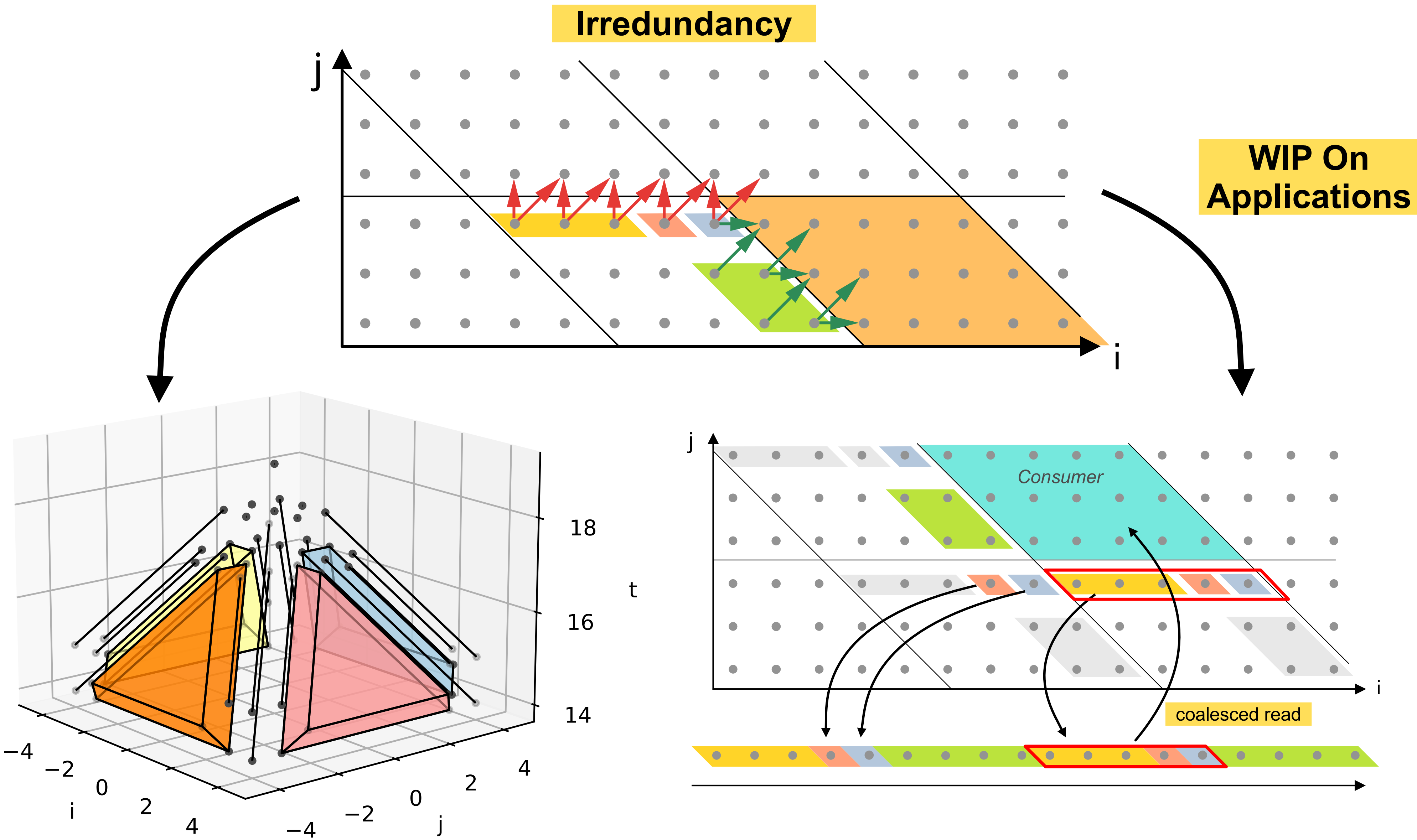
We can use MARS for overclocking, undervolting

MARS for Irredundant Compression

- Compression **without readback redundancy**



Conclusion - Take-Home



Thank you

Questions?