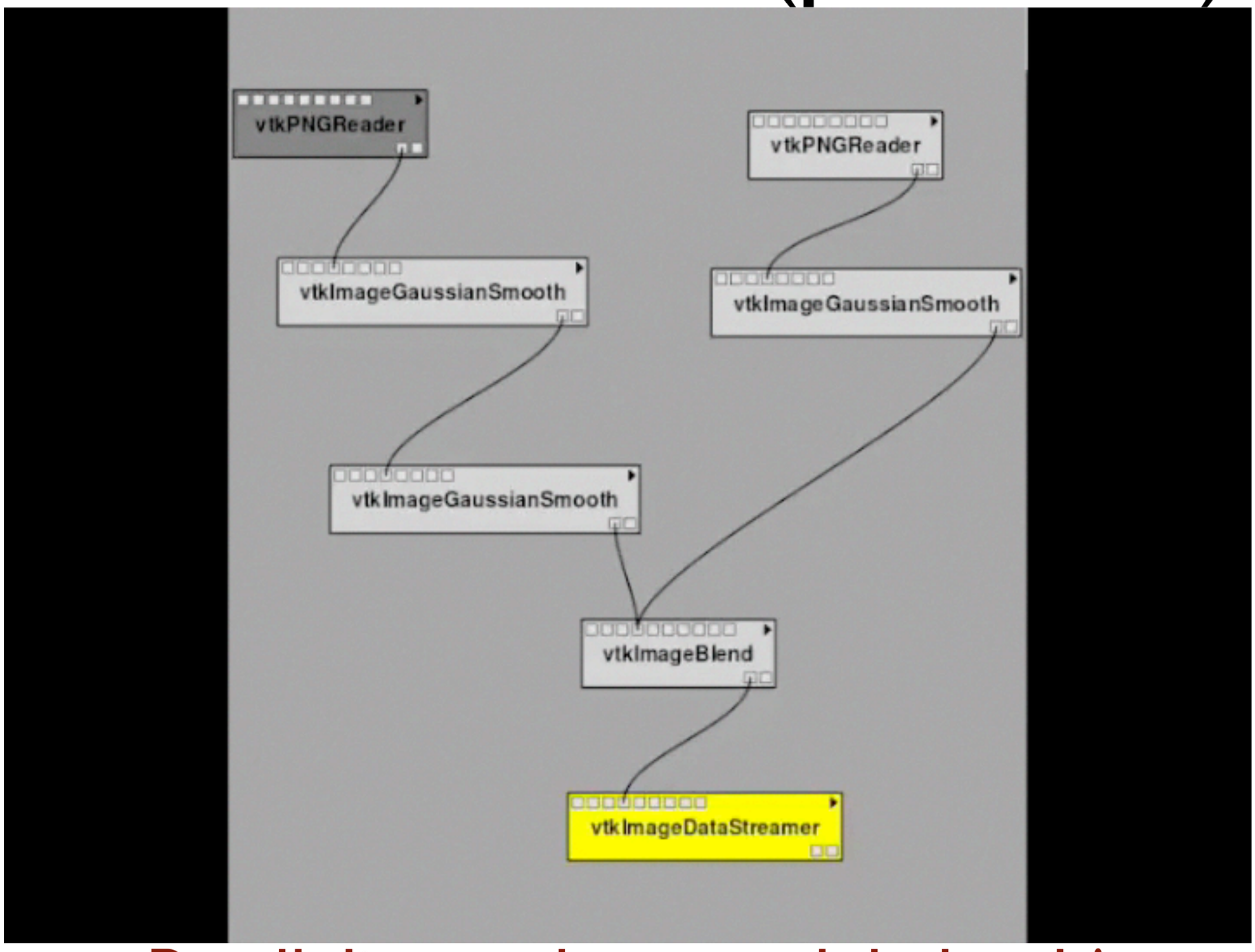
Exploring Streaming Dataflow Architecture for In Situ Visualization

Huy Vo, Daniel Osmari, Joao Comba, Peter Lindstrom, and Claudio Silva

Work in Progress Report

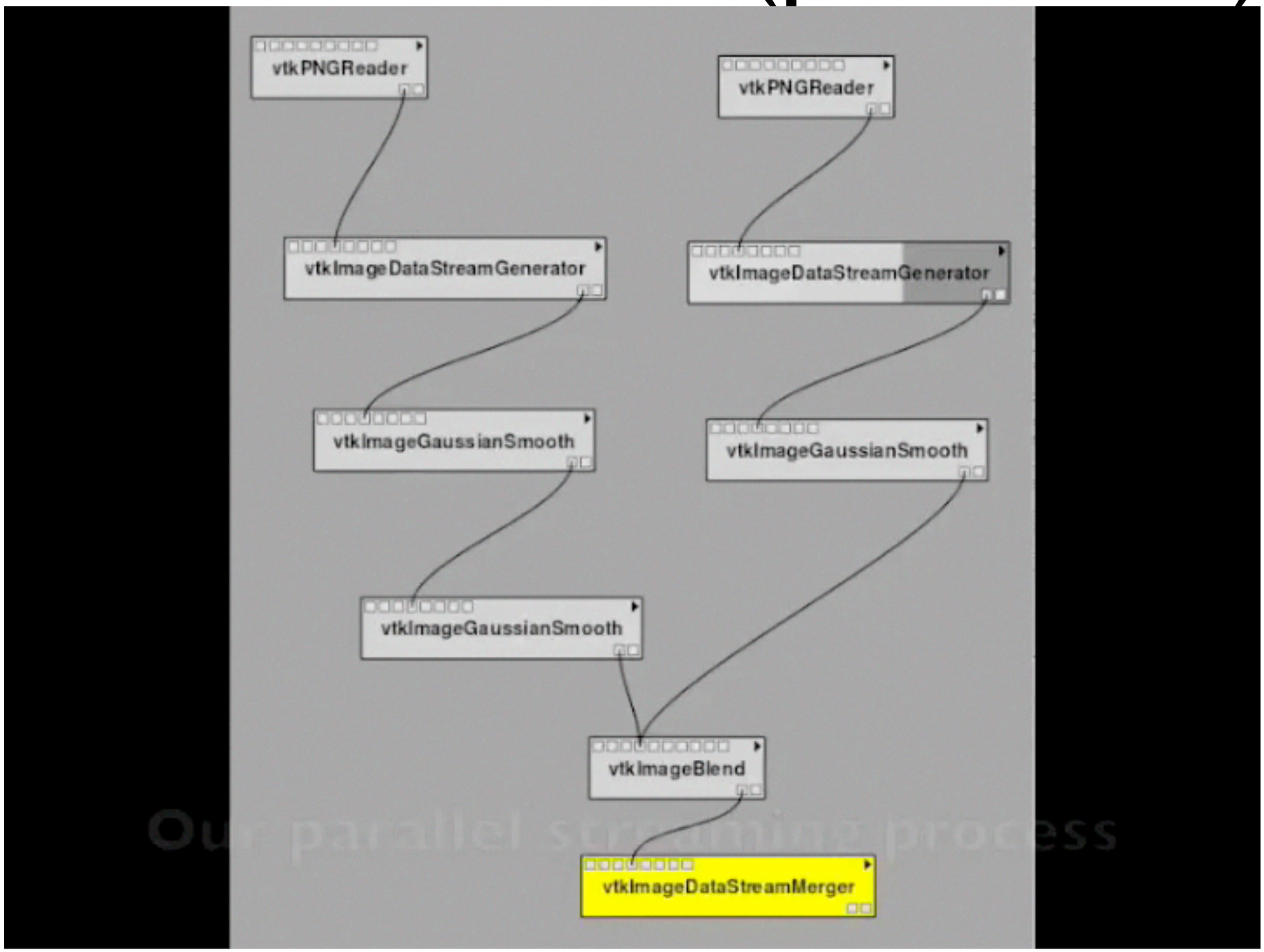
Funded by Sandia, LLNL, DOE, and NSF

VTK Parallelism (pre 2010)



Parallelism only at module level !

VTK Parallelism (post 2010)



Issues

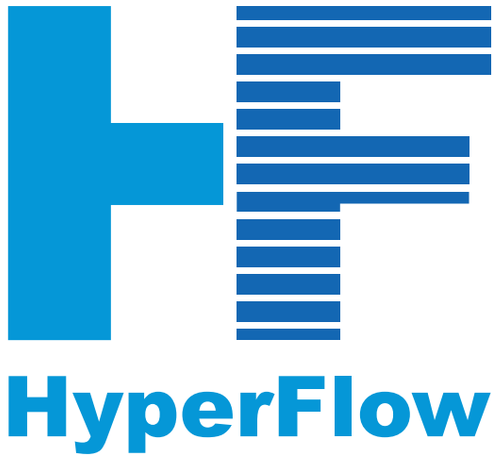
- Resources are maintained per each module
- Not suitable for GPU coordinations
- Conforming to VTK pipeline requests is making extension to the system to be over complicated

- Citations to detailed related work (e.g., VisIt, ParaView, DeVIDE, SCIRun, VisTrails, ...) are in our manuscript.

... then comes HyperFlow

- A true dataflow architecture
- Supporting both GPUs and CPUs
- Flexible to extend to other platforms

- Provide highly-parallel constructs

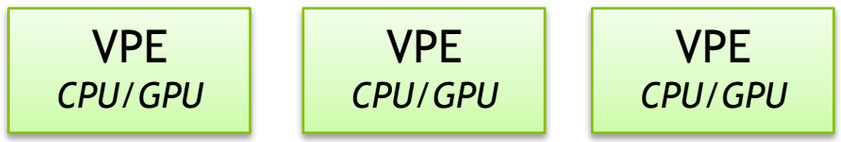


HyperFlow Architecture

- Task-Oriented Module (TOM)
  - A set of implementations
  - Each has its own resource specification

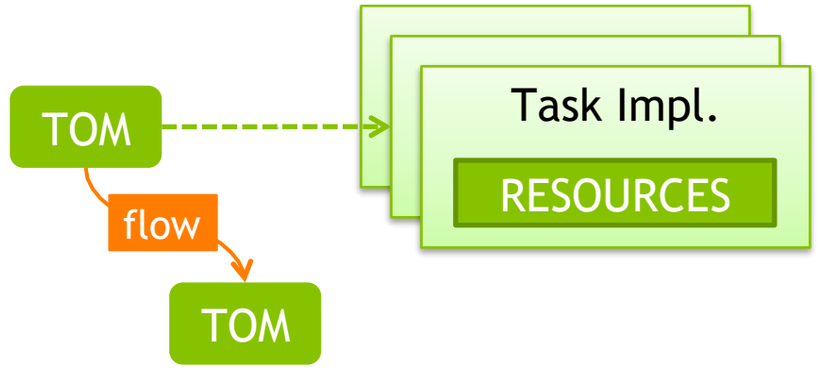
HyperFlow Architecture

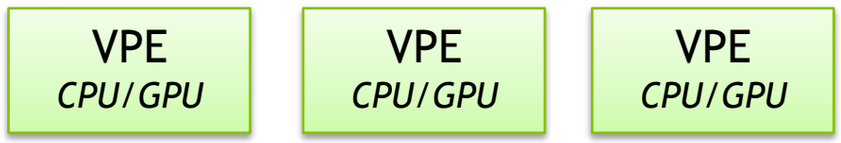
- Virtual Processing Element (VPE)
  - Abstract device driver
  - Setup context for execution and data transfer



HyperFlow Architecture

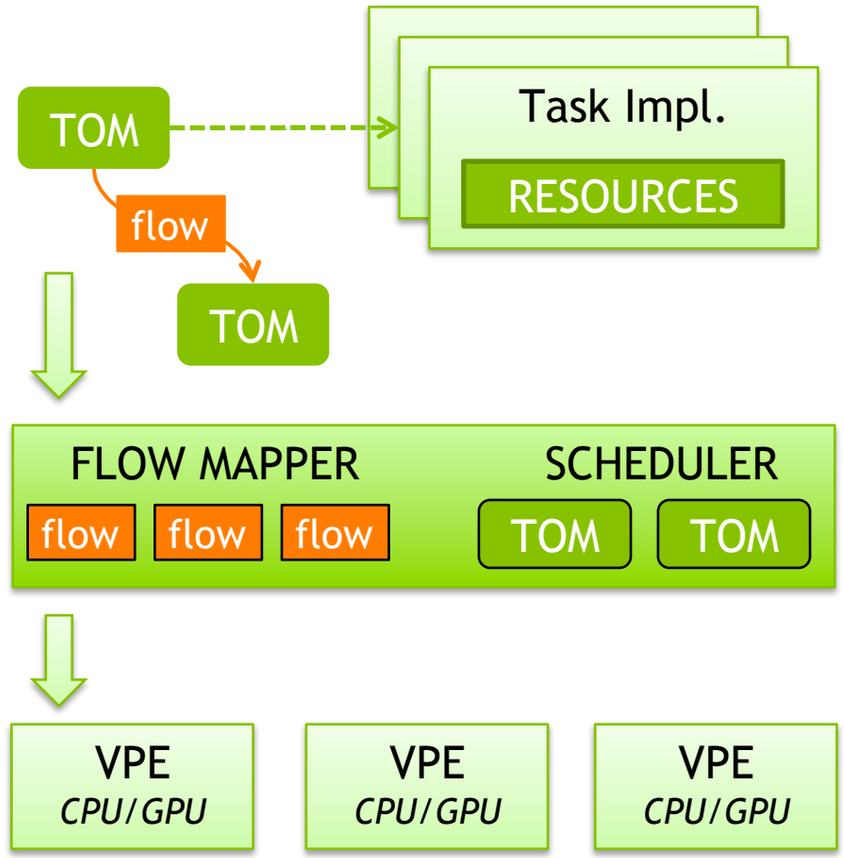
- Flows - Data Connections
  - Execution tokens
  - Wrapping actual data





HyperFlow Architecture

- Resource Manager and Execution Scheduler
  - Direct flows
  - Schedule data transfers
  - Map computation to VPEs



HyperFlow Library

- roughly 3000 lines of code
- 15 classes, but only need to know 5
  - hyperflow::TaskImplementation
  - hyperflow::TaskOrientedModule
  - hyperflow::Flow
  - hyperflow::Data
  - hyperflow::ExecutionEngine

HyperFlow API

- Easy to use: just send *flows* to the system

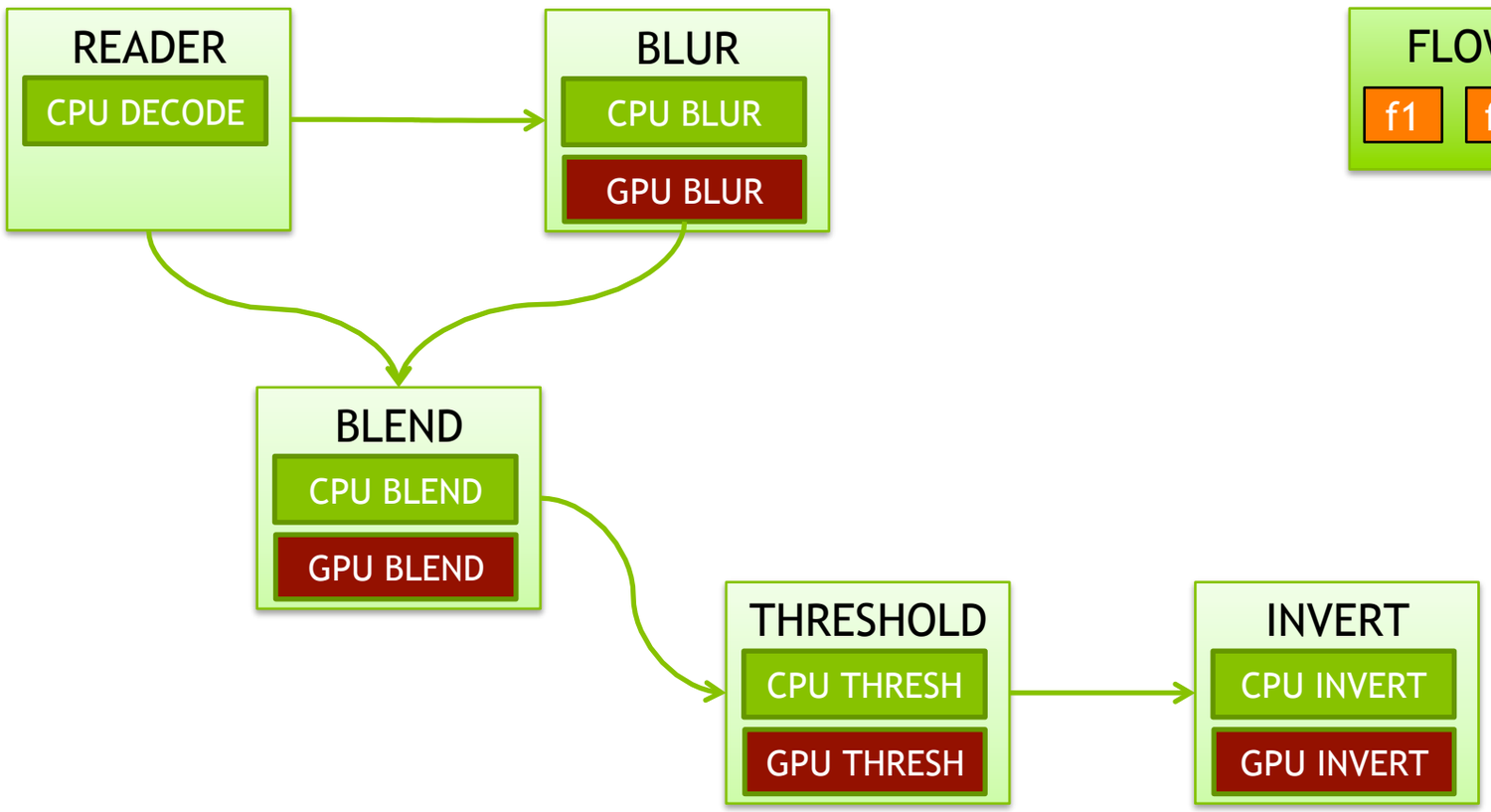
```
class TaskOrientedModule {
public:
  TaskOrientedModule(int nInput, int nOutput,
                     const char *moduleName);
  void addImplementation(TaskImplementation* impl);
  static void connect(
      TaskOrientedModule* srcModule, int srcPort,
      TaskOrientedModule* dstModule, int dstPort
    );
  ... // Additional parameters if subclassed
};
// Construction of the pipeline

TaskOrientedModule Source(0, 1, "Image Reader");
TaskOrientedModule Filter(1, 1, "Gaussian Blur");
TaskOrientedModule Sink(1, 0, "Viewer");
Source.addImplementation(new SourceImpl());
Filter.addImplementation(new FilterImpl());
Sink.addImplementation(new SinkImpl());
TaskOrientedModule::connect(&Source,0,&Filter,0);
TaskOrientedModule::connect(&Filter,0,&Sink,0);
```

```
while <data is coming> do {
  Flow *flow = engine->createFlow(data);
  engine->sendFlows(1, flow, RET_ON_DEPLOY, true);
}
```

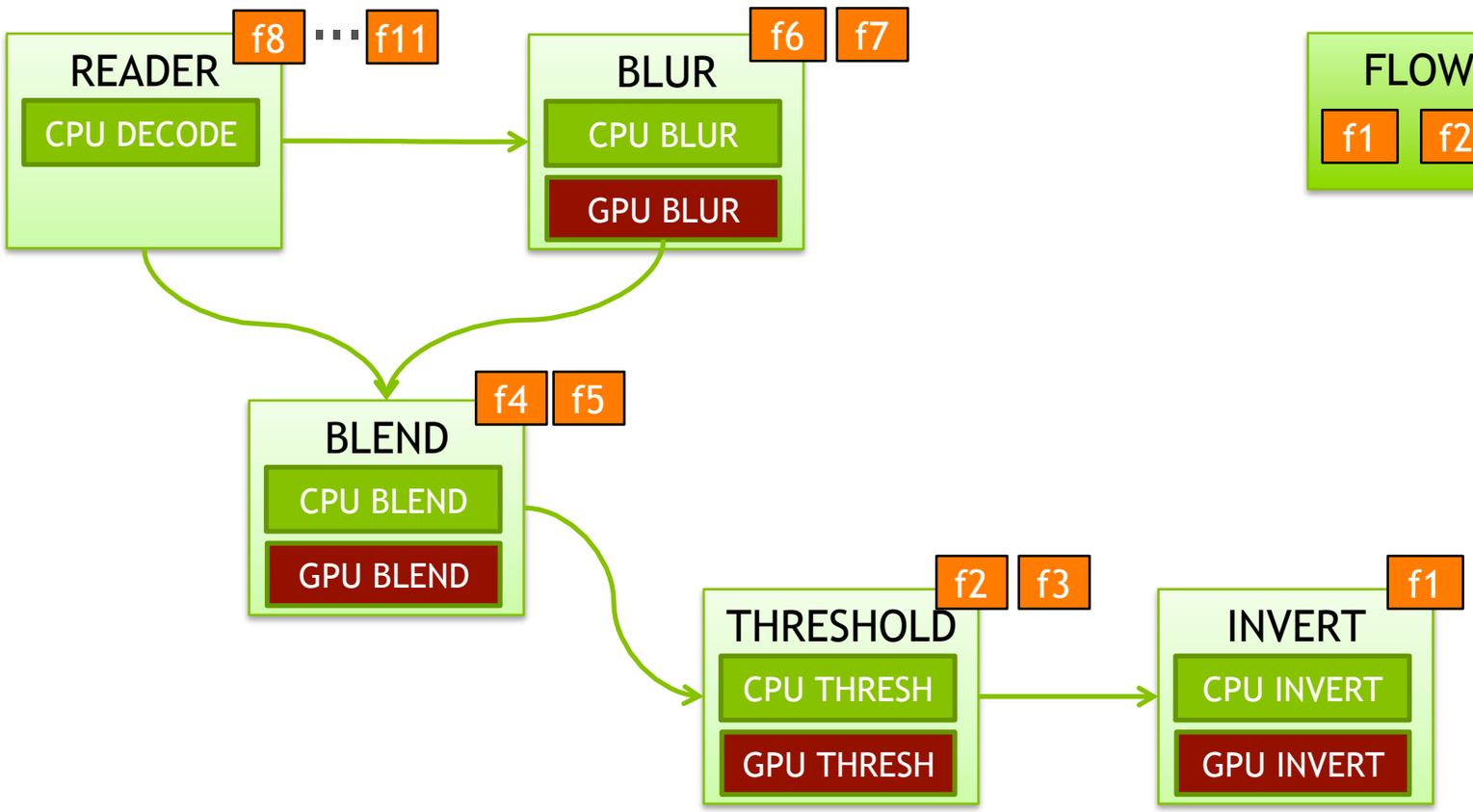
HyperFlow In Action

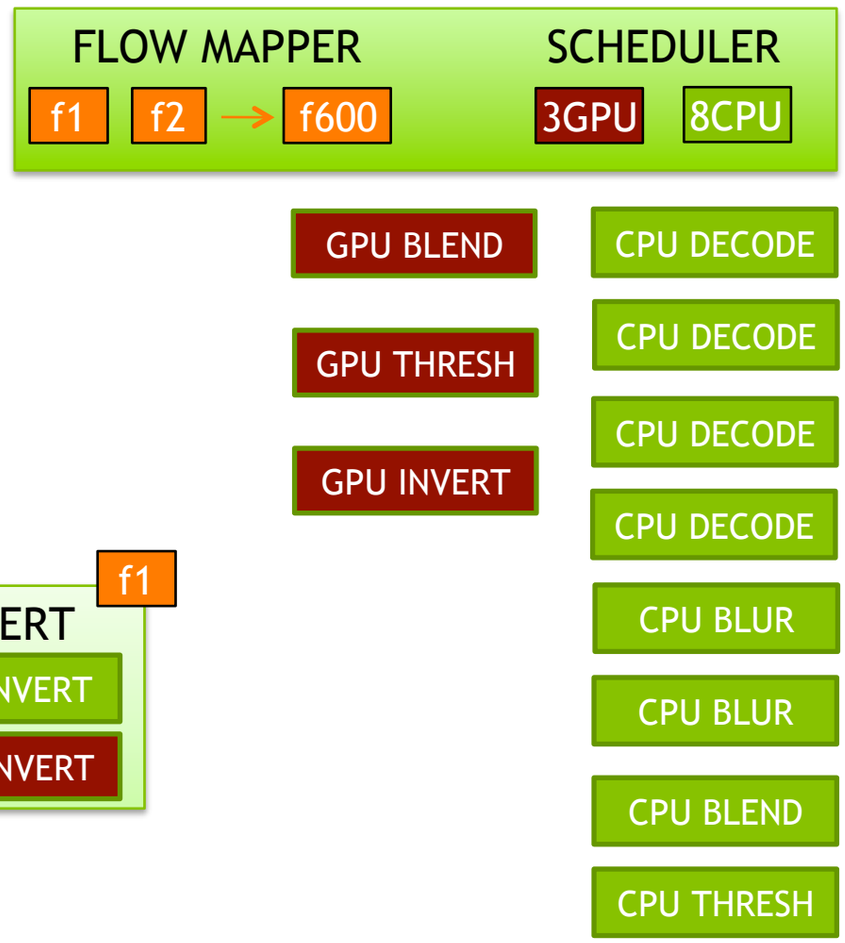
- An edge detection pipeline



HyperFlow In Action

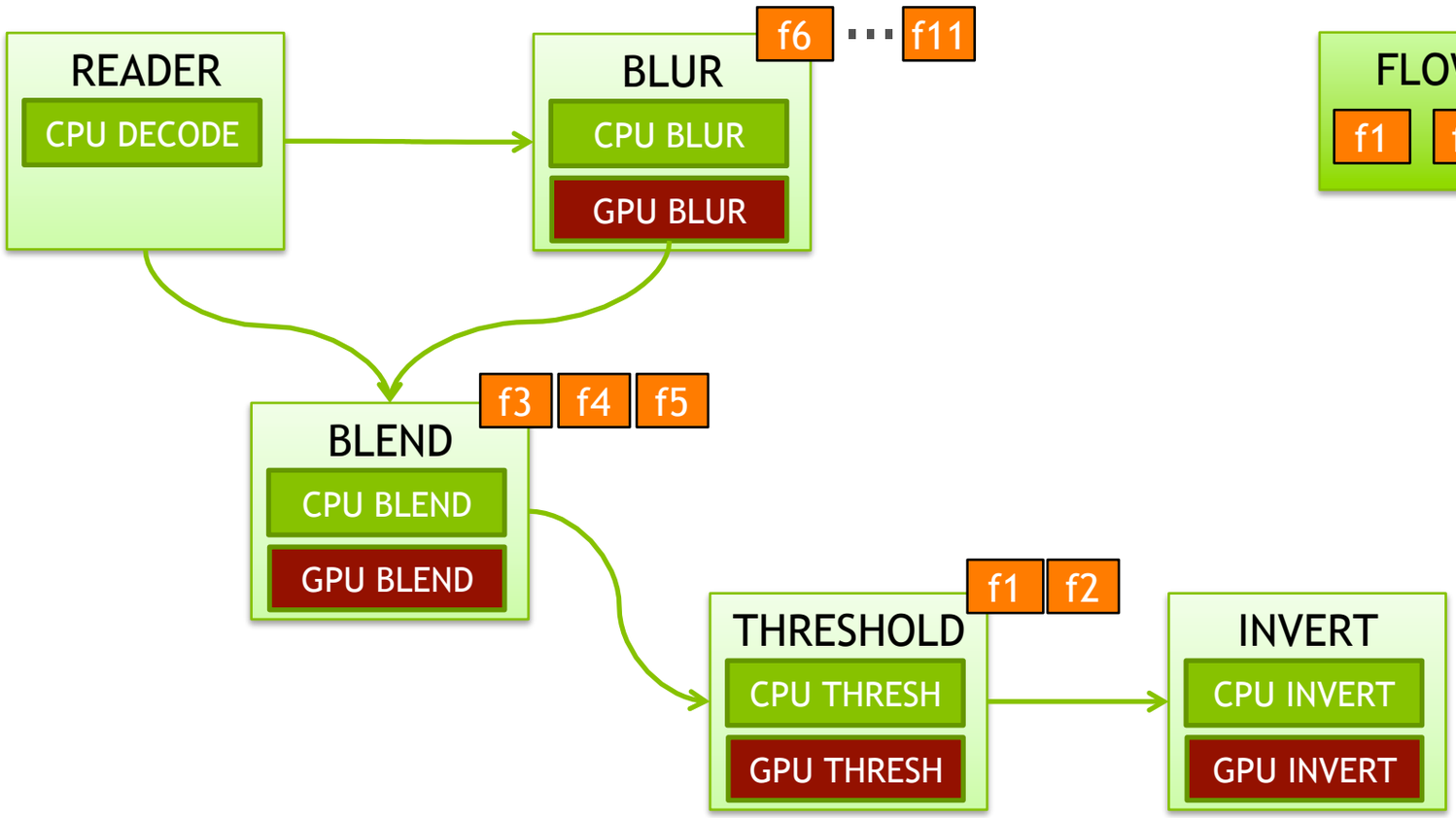
- Each flow carries a piece of image

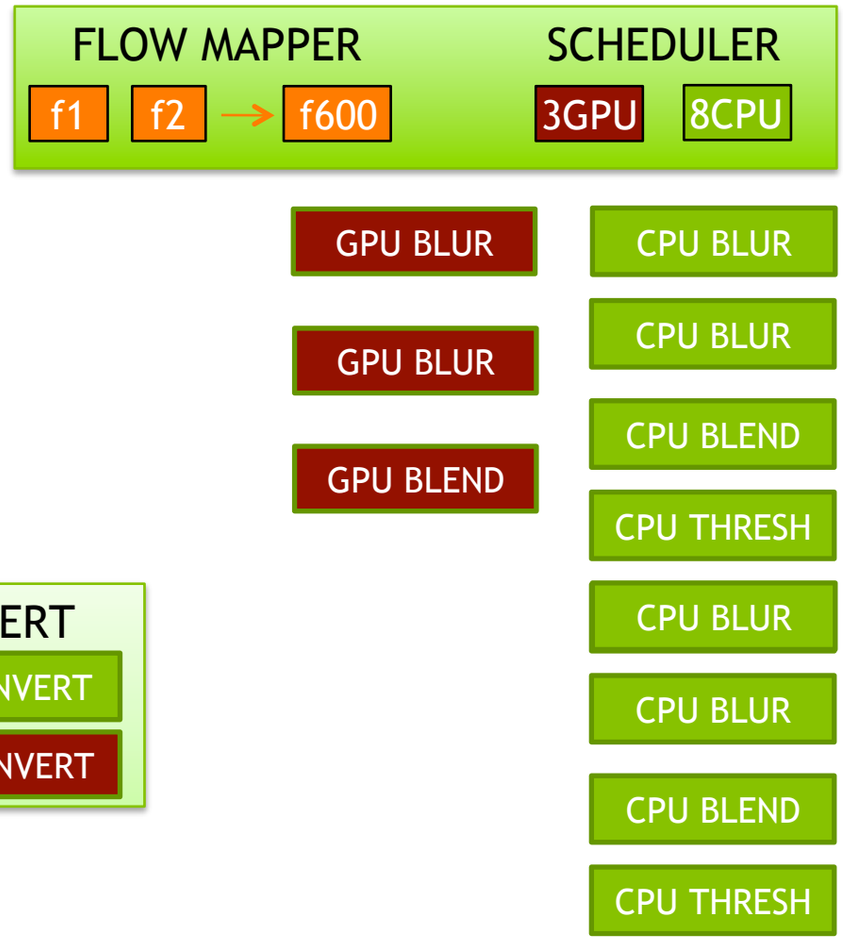




HyperFlow In Action

- Pairs of (module, flow) are mapped onto available resources





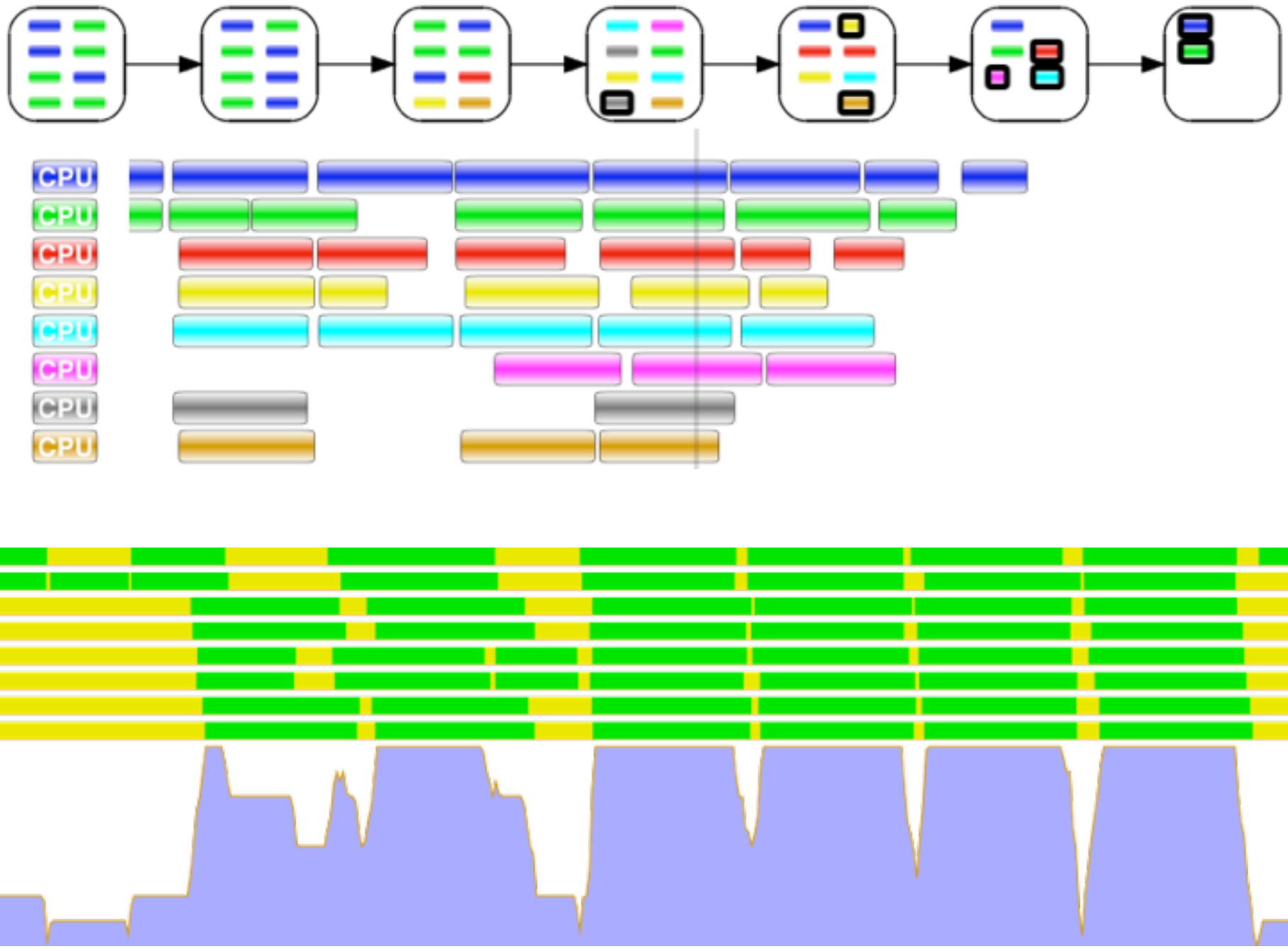
Experiments with HyperFlow

- Synthetic applications (micro-benchmarks)
  - Evaluating scheduling and data handling strategies
- Real applications
  - Evaluating scalability, performance and usability in practice

Micro-benchmarks

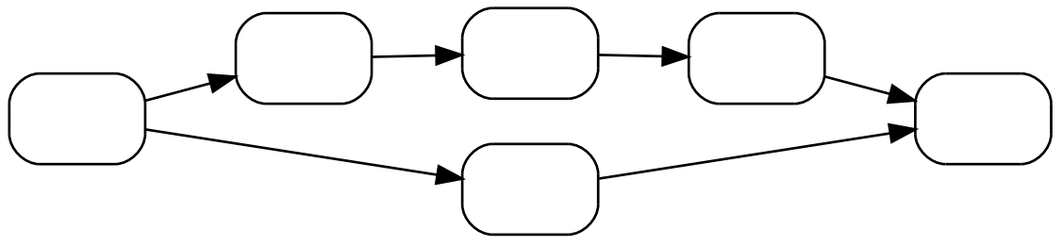
- Model actual pipelines without implementing real computation code
- Each task implementation is parameterized by execution time and input/output ratio
- Generated automatically by Python script
- Benchmark results can be visualized after runs with flow animation and Gantt charts of resources utilization.

Trace Visualization

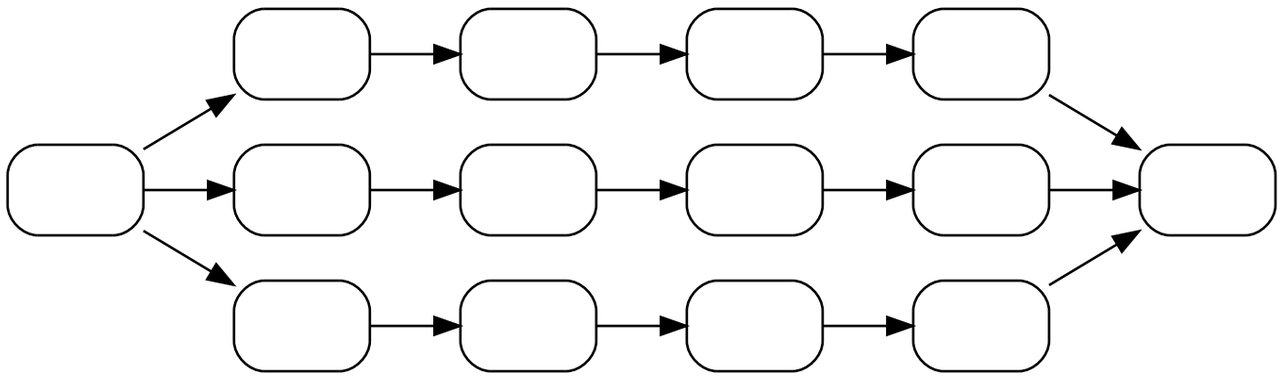


Micro-benchmarks Pipelines

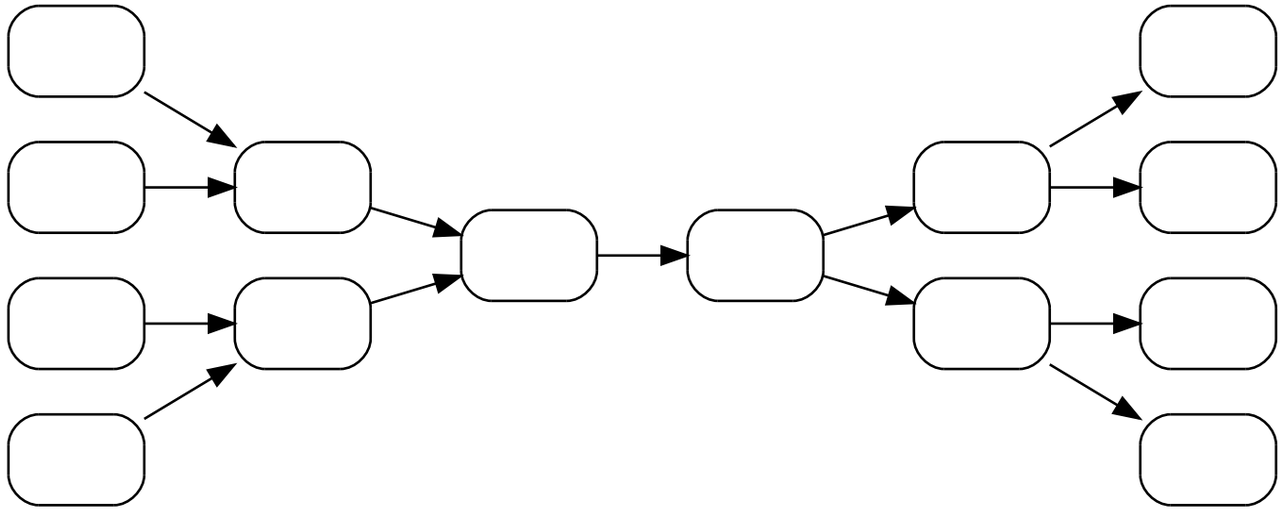
Asymmetric



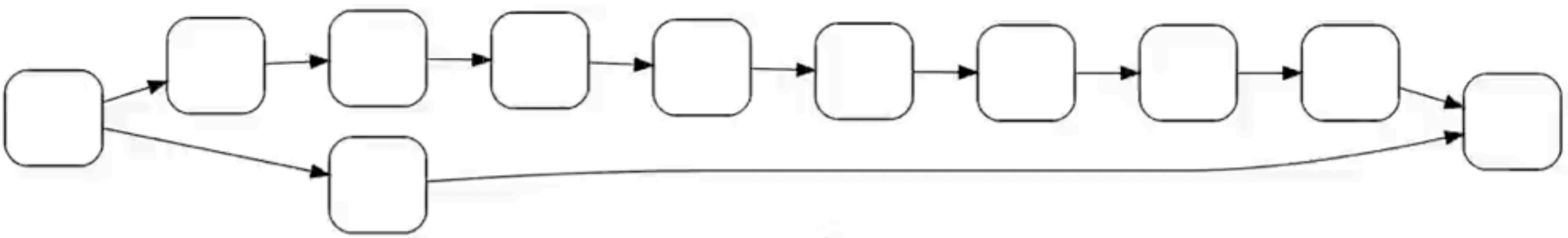
Split-Join



Scan

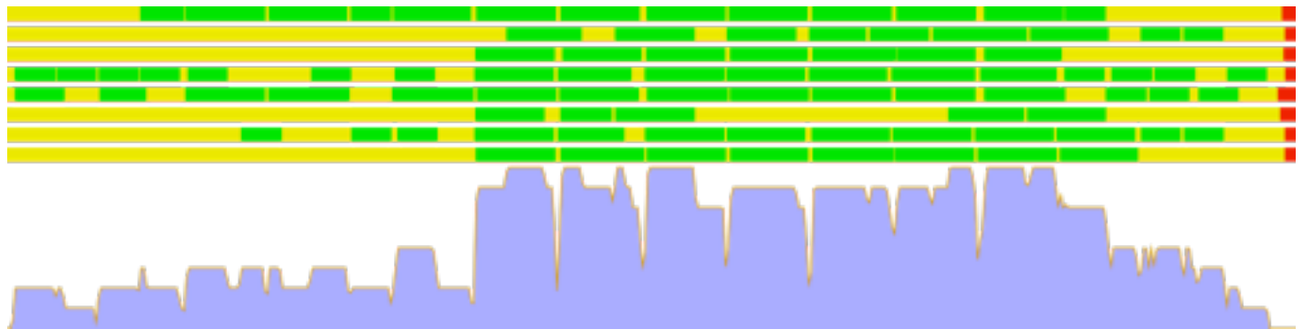


Asymmetric Flow Animation

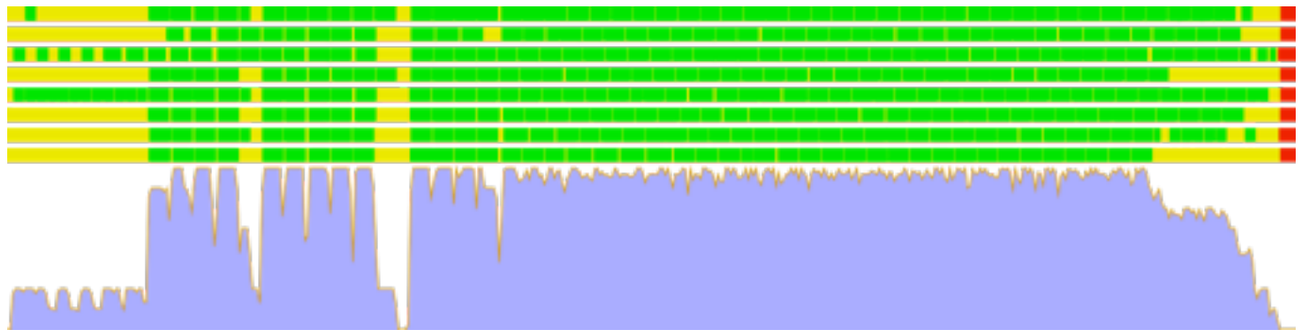


Micro-benchmarks Gantt Charts

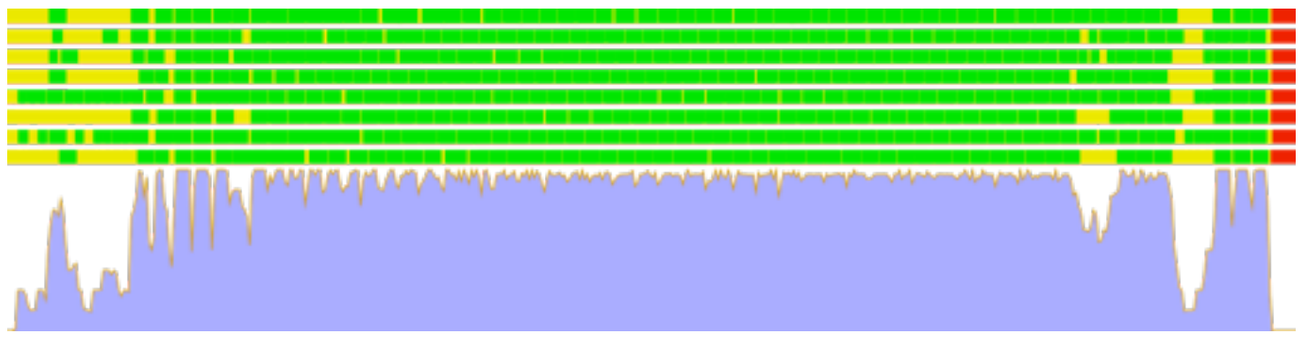
Asymmetric
(57.4% utilization)



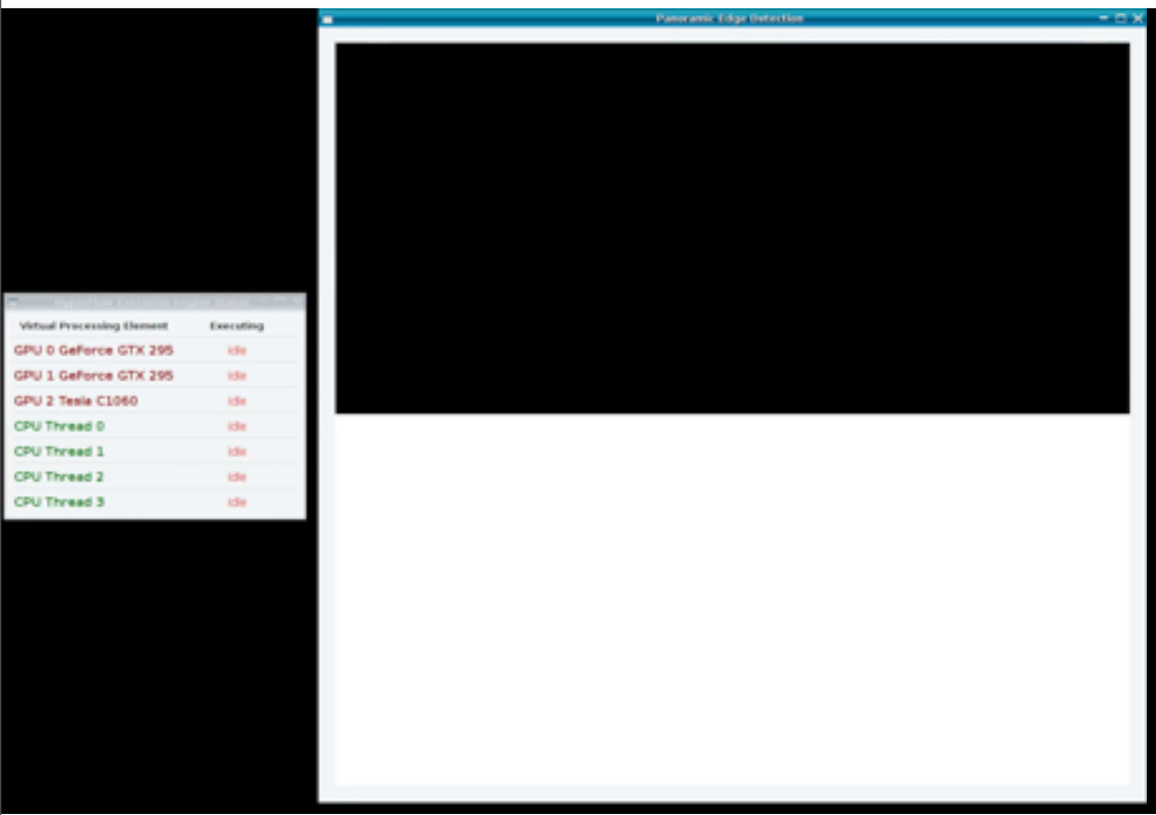
Split-Join
(78.9% utilization)

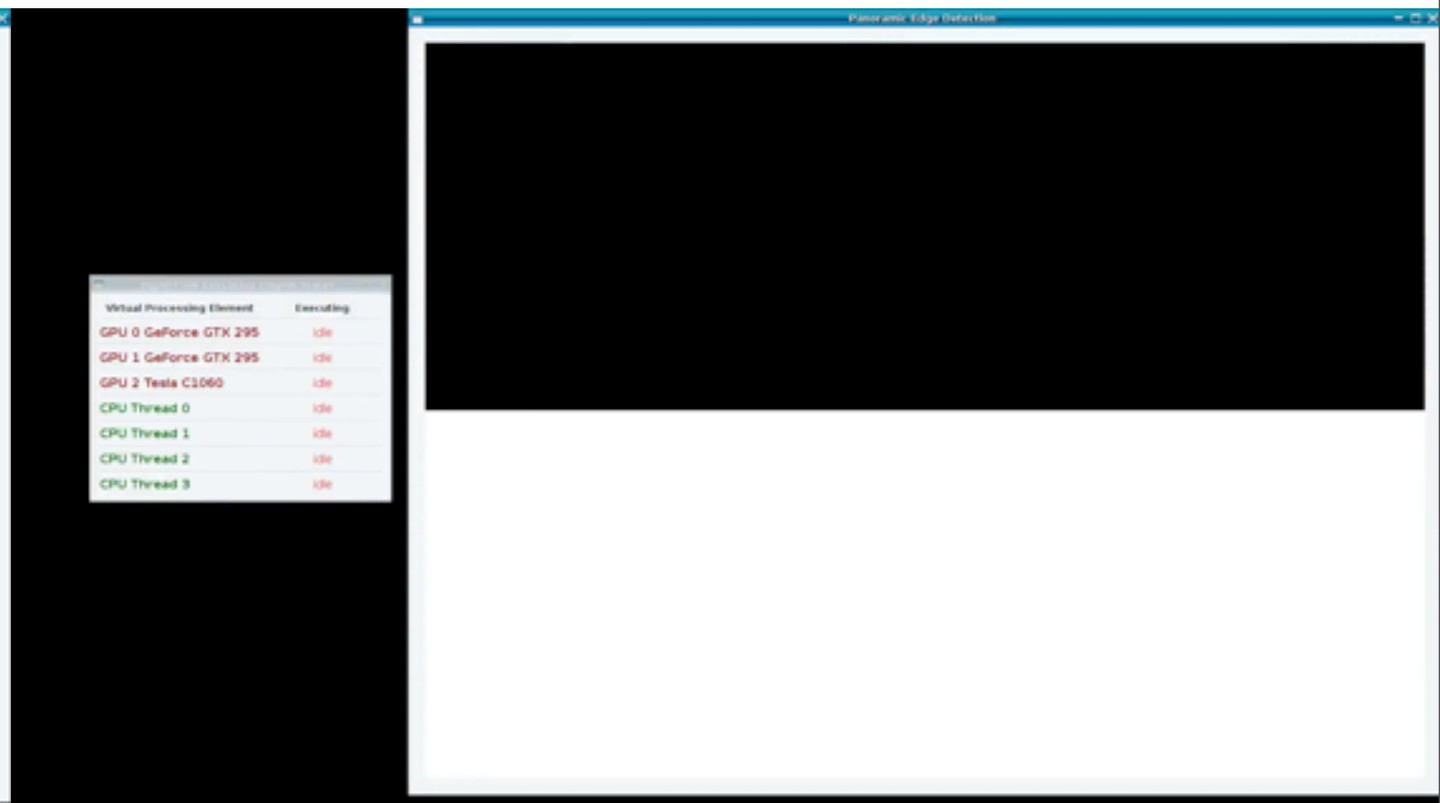


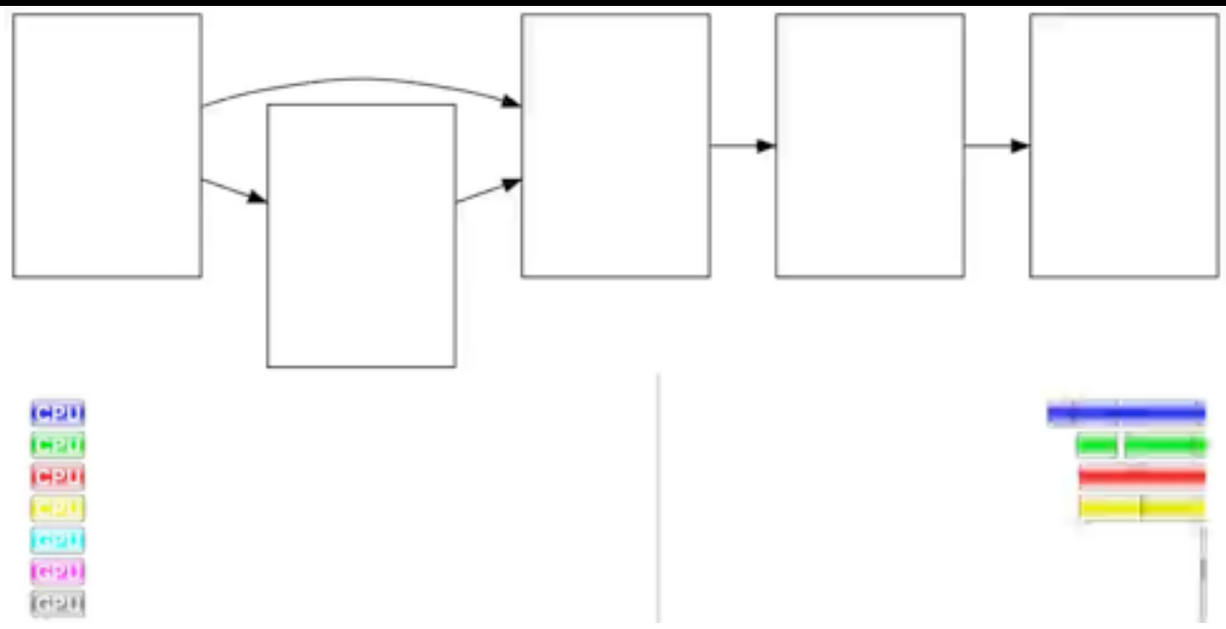
Scan
(83.4% utilization)



Streaming Edge Detection





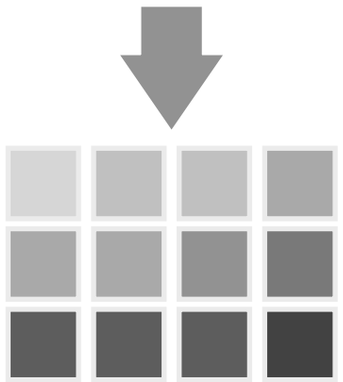


In Situ Pipeline

Simulation Code

In Situ Pipeline

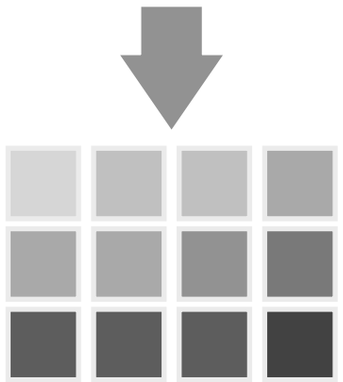
Simulation Code



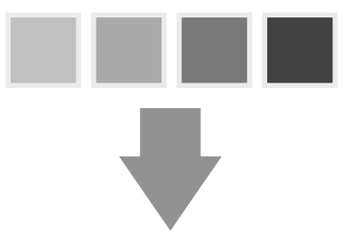
Visualization Code

In Situ Pipeline

Simulation Code

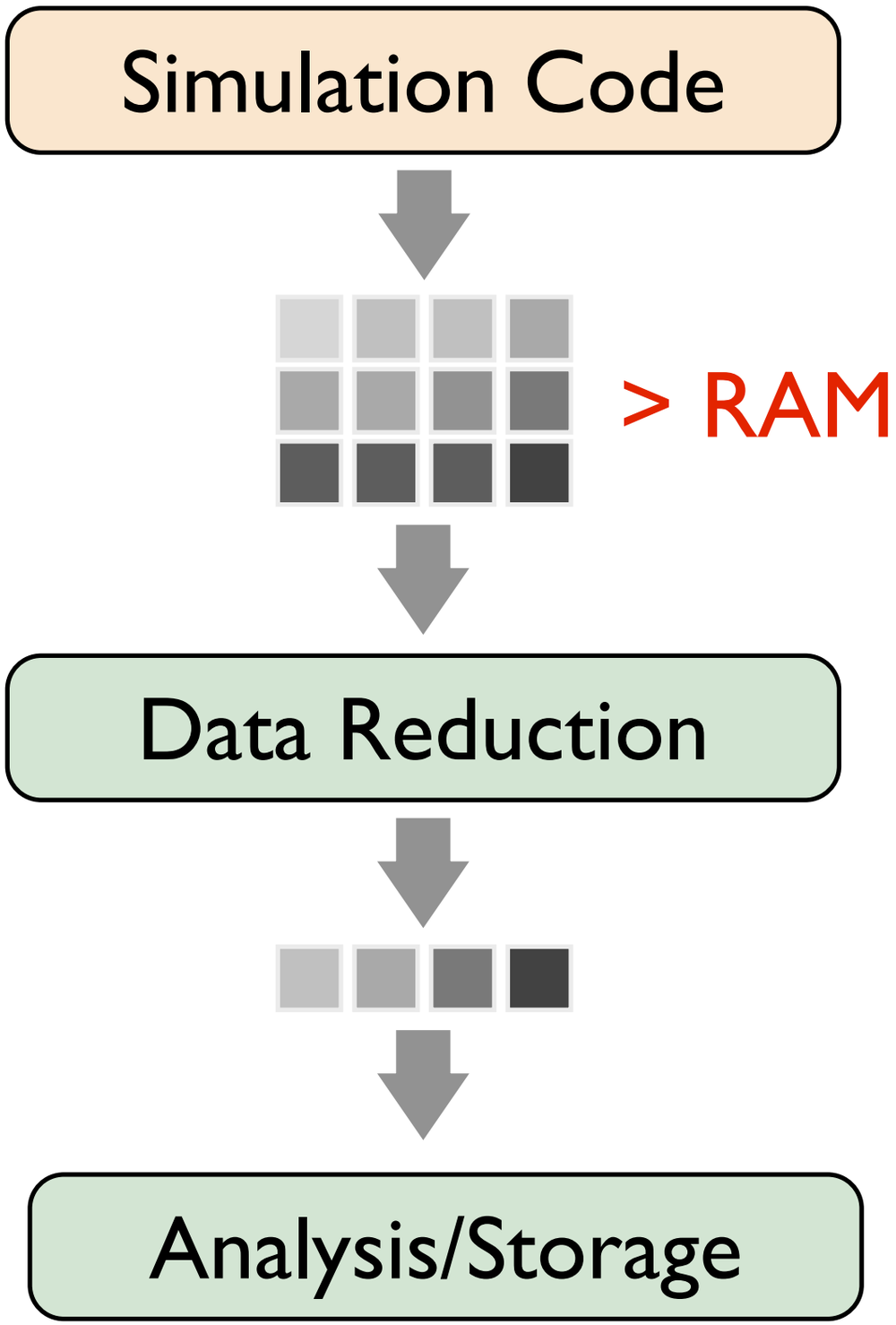


Data Reduction

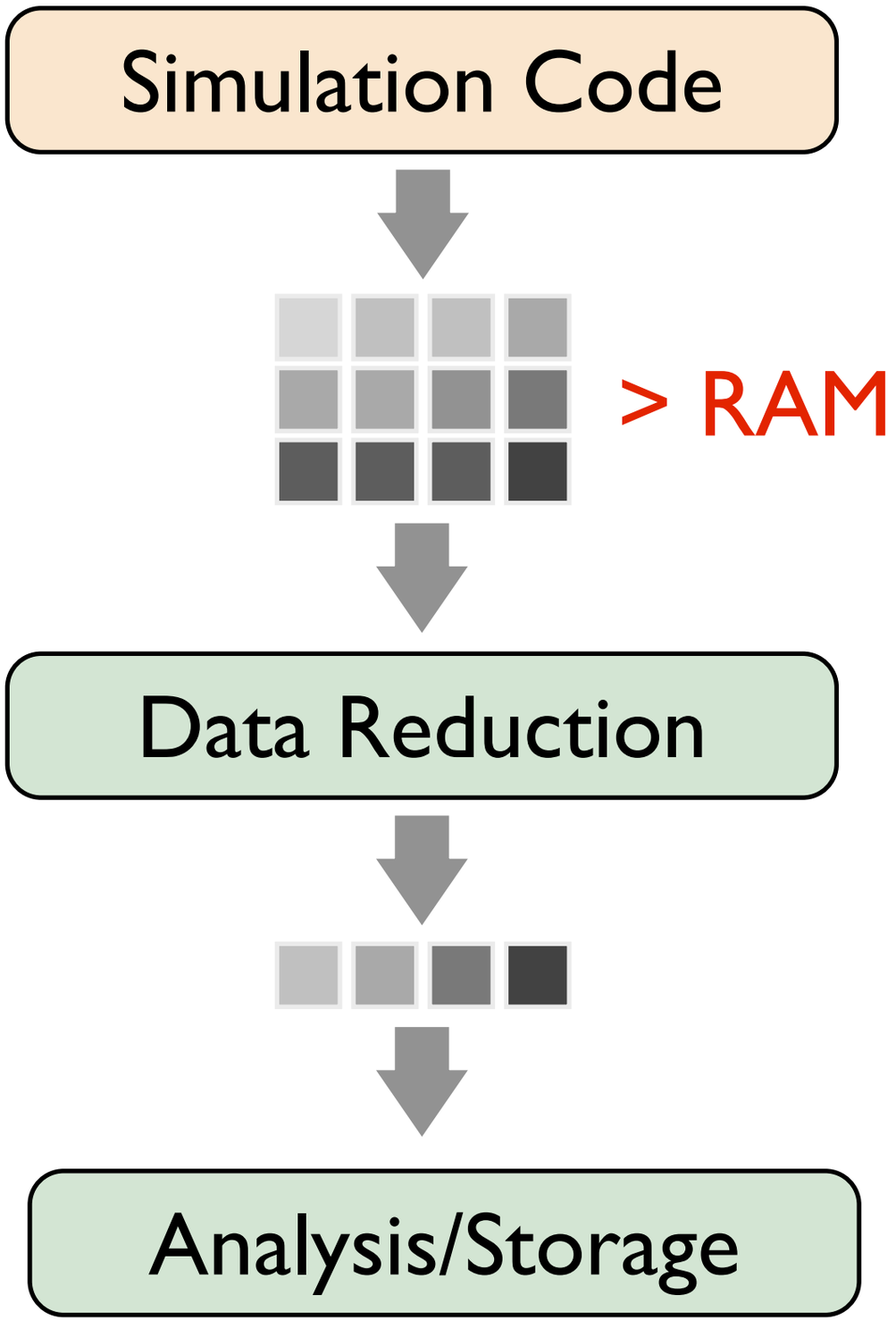


Analysis/Storage

In Situ Pipeline



In Situ Pipeline

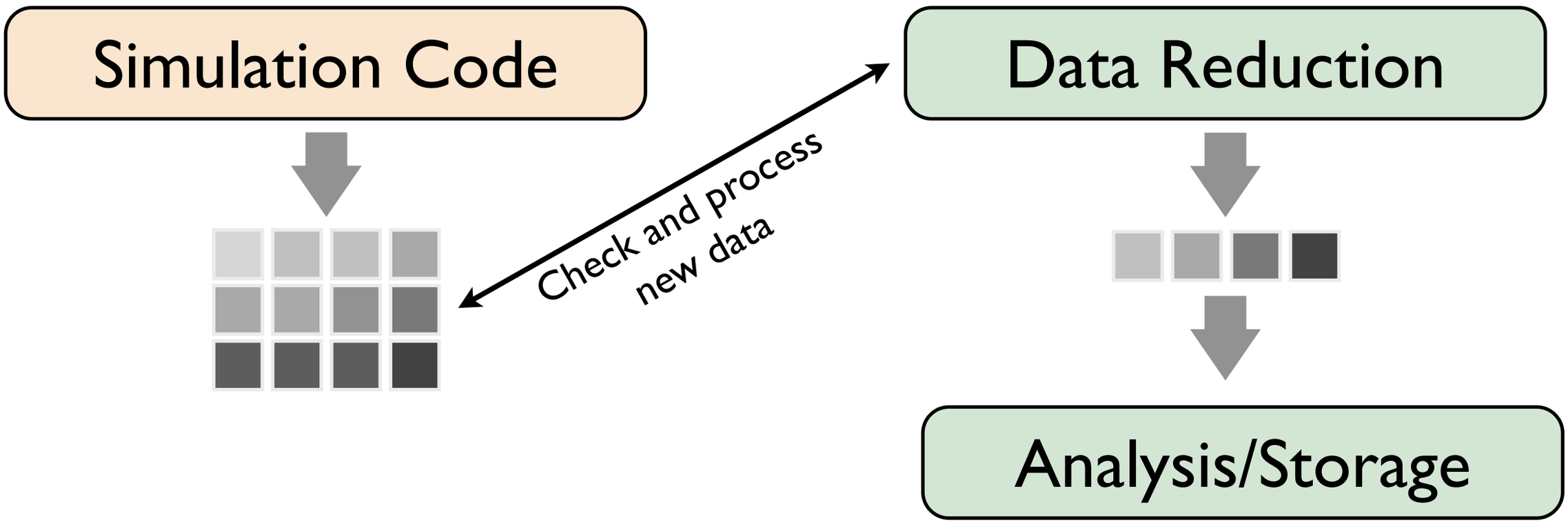


Sequential Execution:

```
Run all simulations
Run data reduction
```

✗ data spilled to disks

In Situ Pipeline

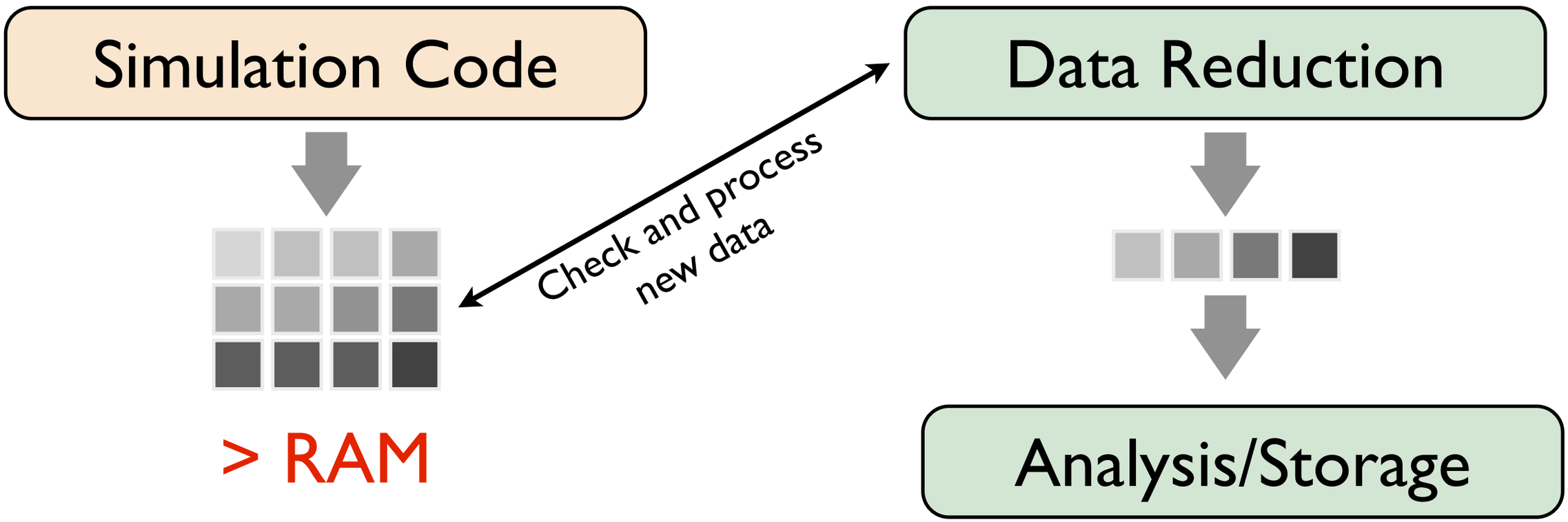


Concurrent Execution:

Visualization and Simulation are **indendent** tasks

performing visualization as soon as data are ready

In Situ Pipeline



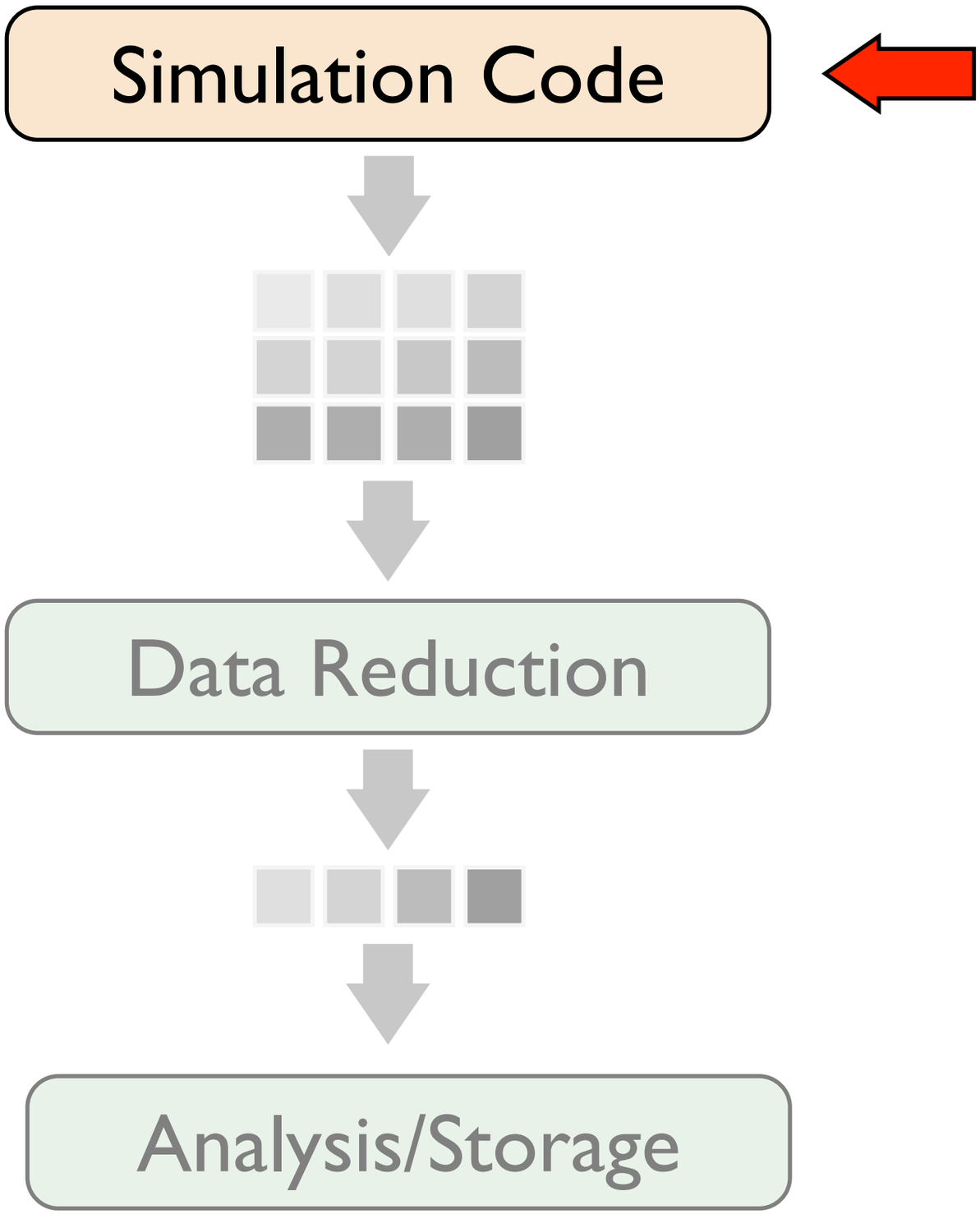
Concurrent Execution:

Visualization and Simulation are **indendent** tasks

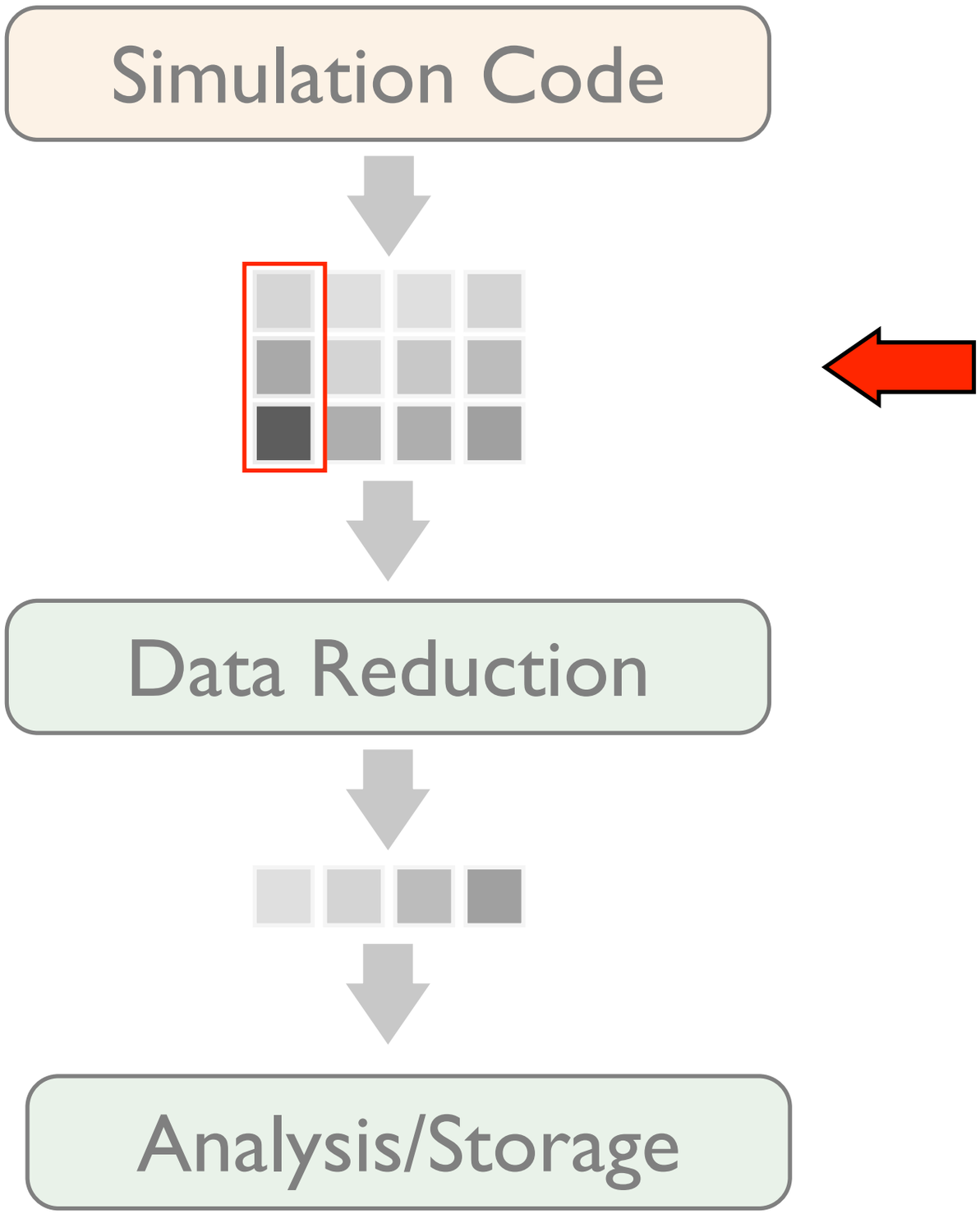
performing visualization as soon as data are ready

✗ still has the memory footprint problem if Simulation produces data faster than Visualization

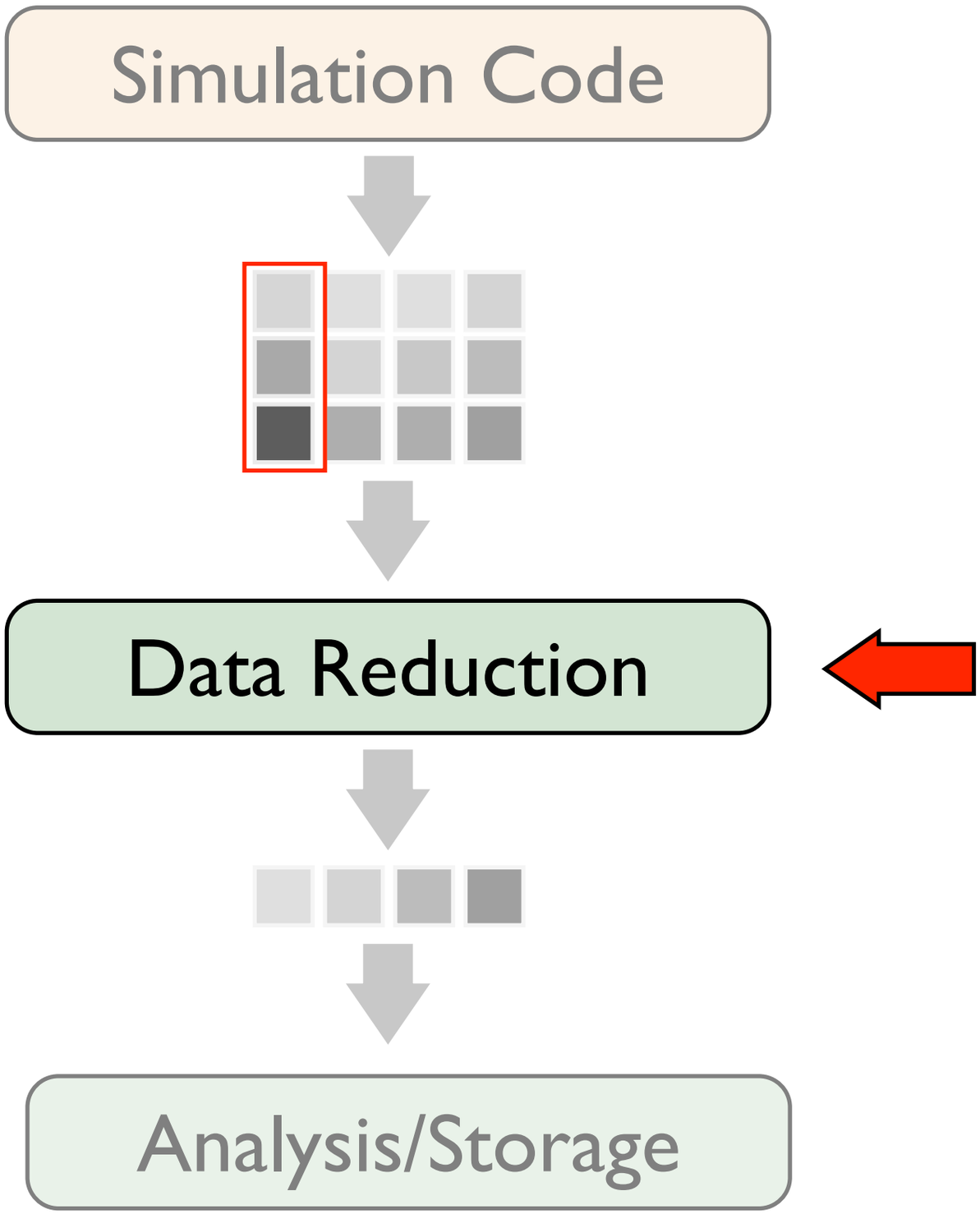
In Situ Streaming Pipeline



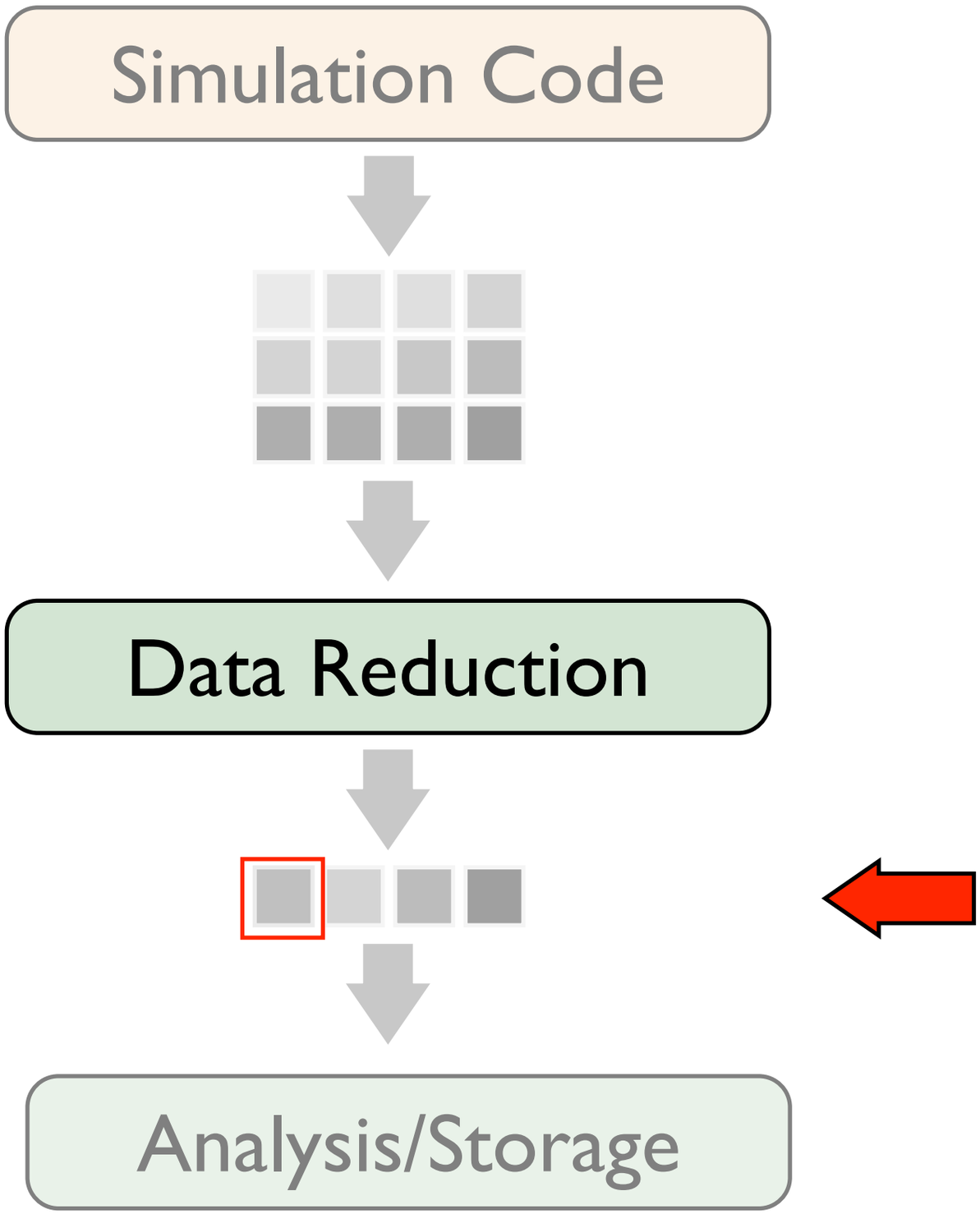
In Situ Streaming Pipeline



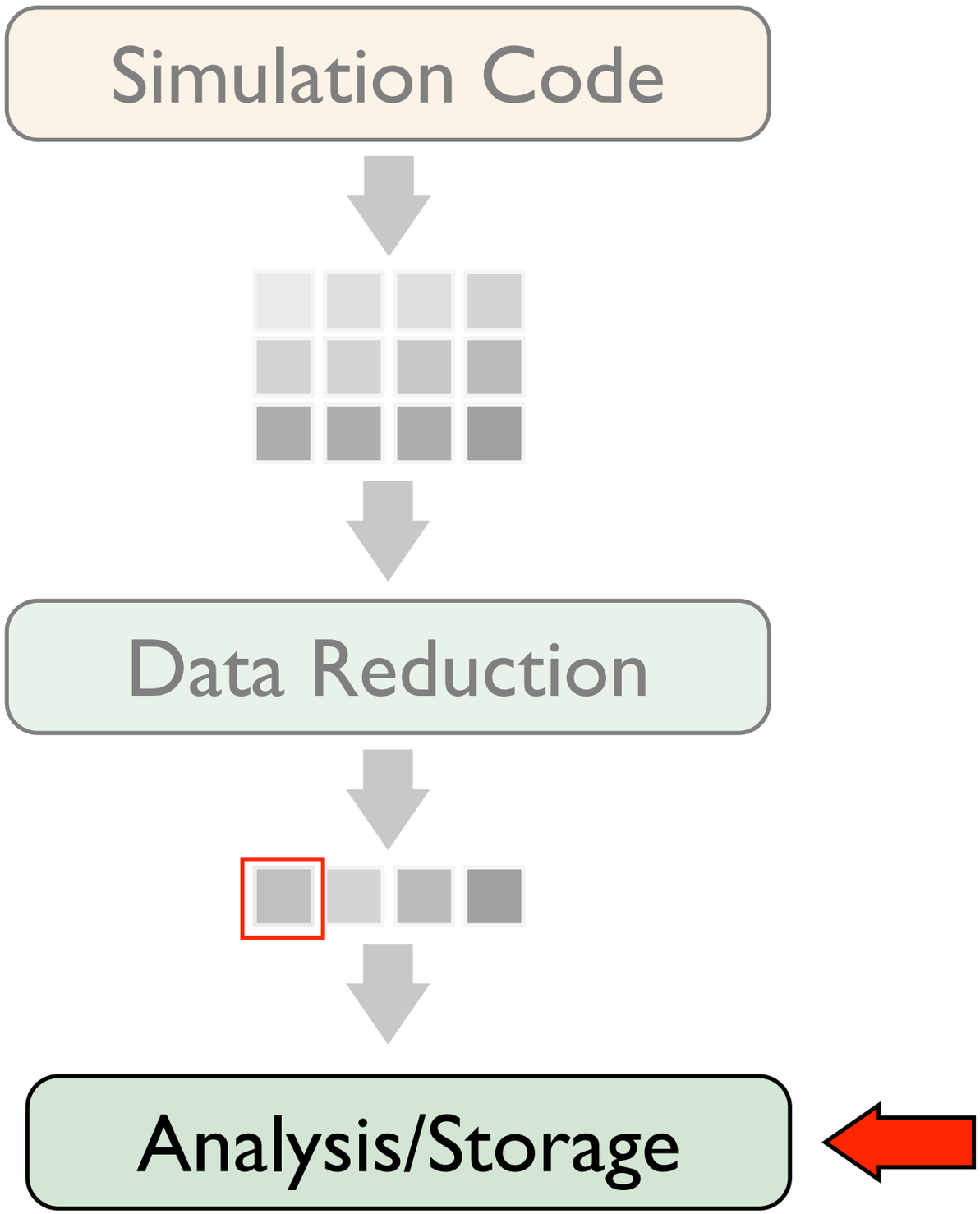
In Situ Streaming Pipeline



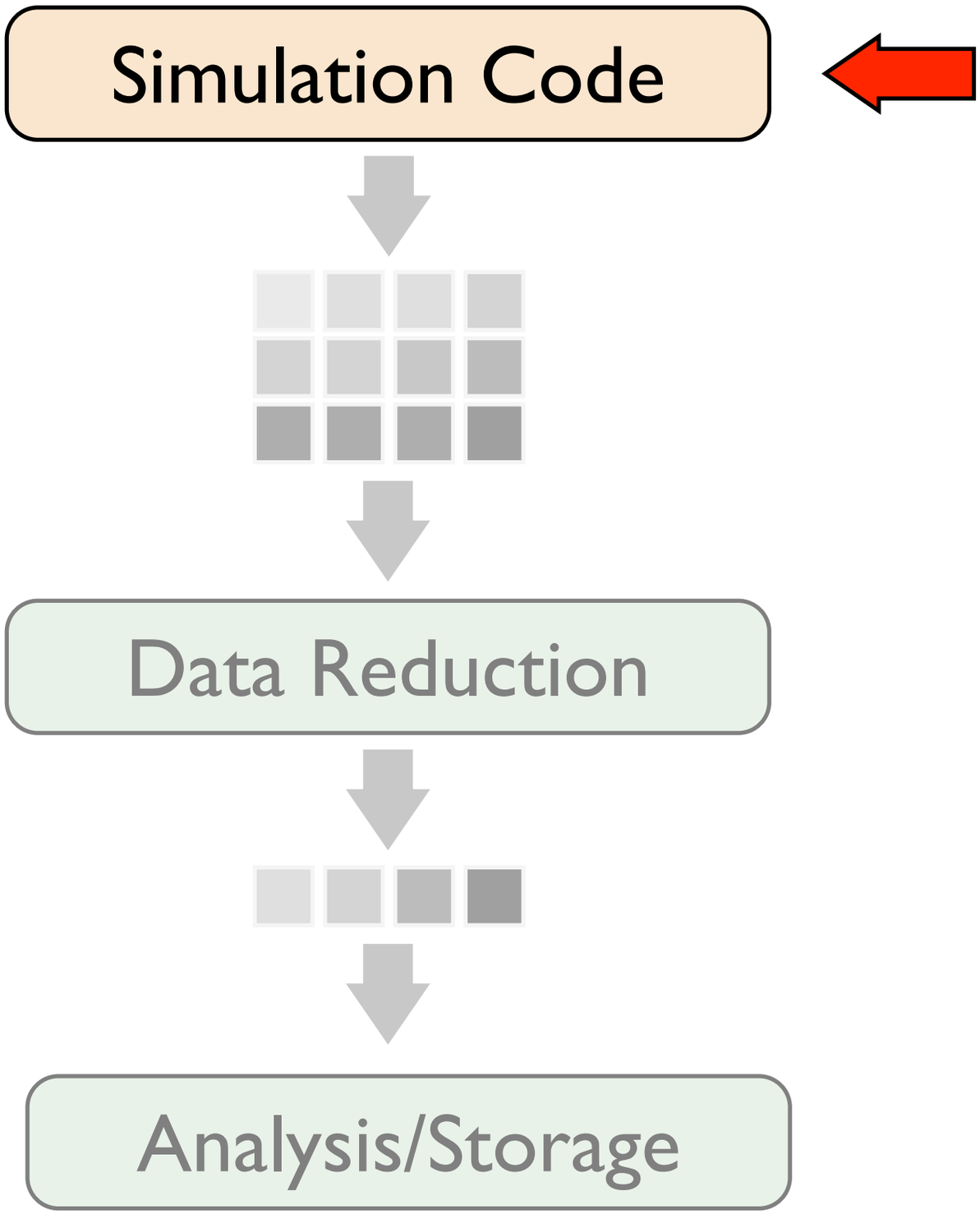
In Situ Streaming Pipeline



In Situ Streaming Pipeline

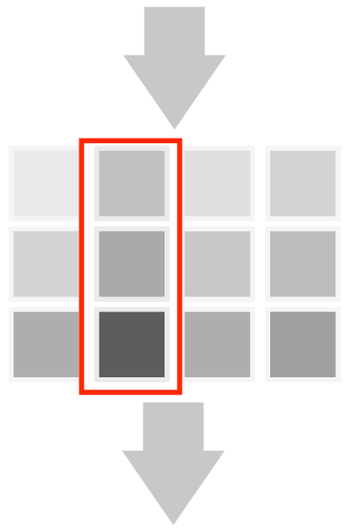


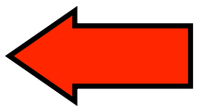
In Situ Streaming Pipeline



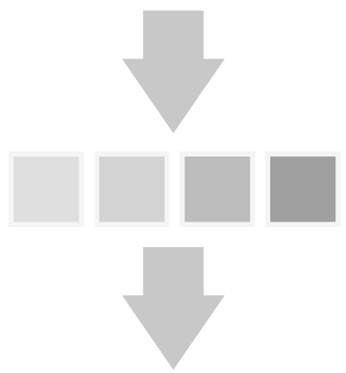
In Situ Streaming Pipeline

Simulation Code





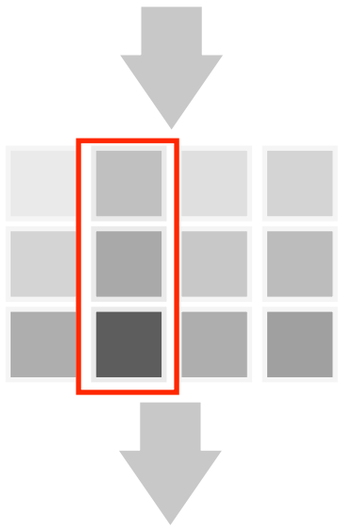
Data Reduction



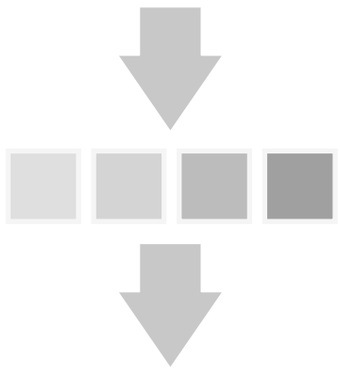
Analysis/Storage

In Situ Streaming Pipeline

Simulation Code



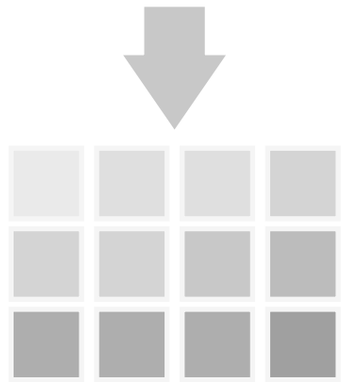
Data Reduction



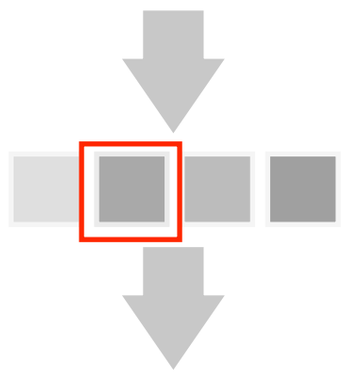
Analysis/Storage

In Situ Streaming Pipeline

Simulation Code

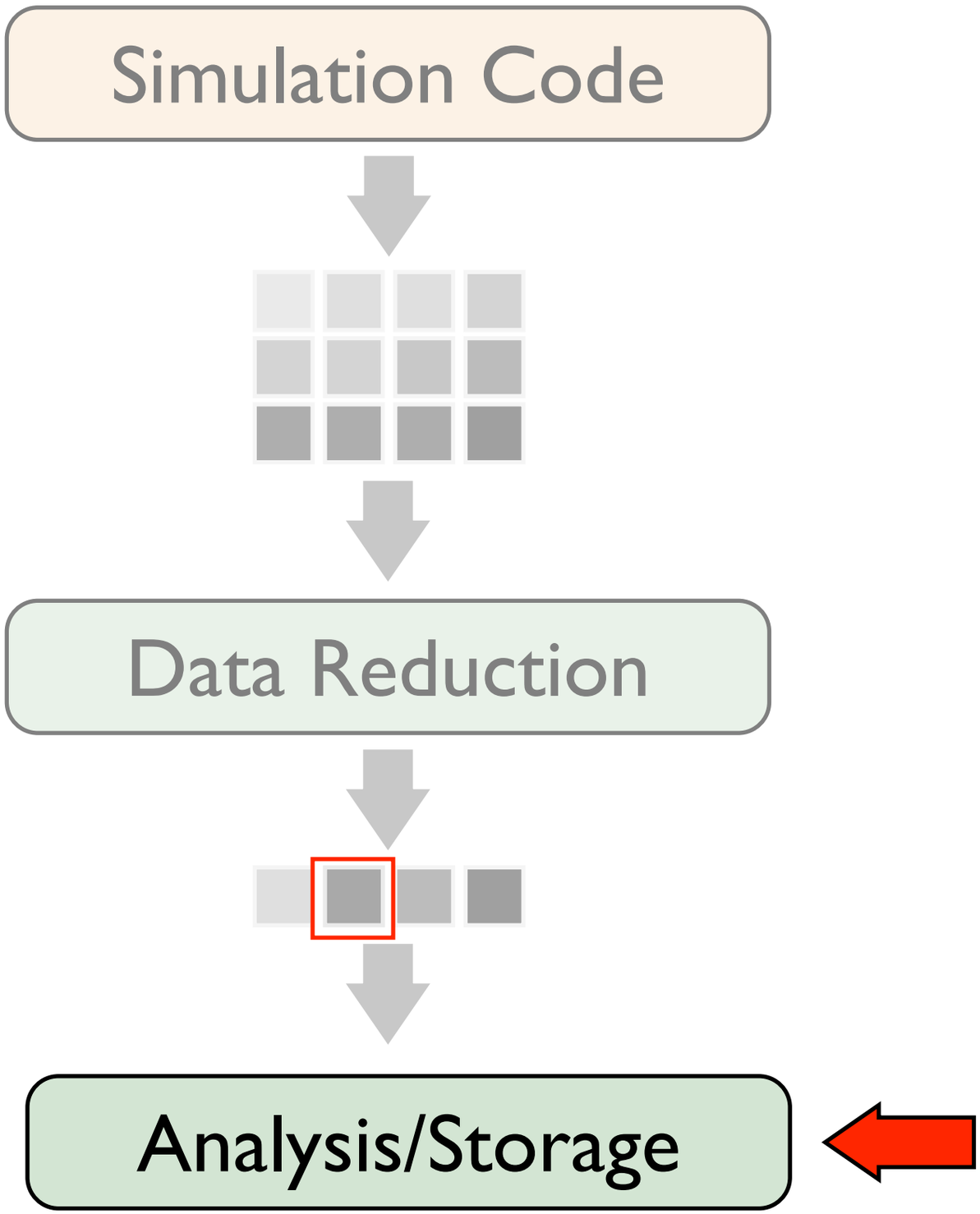


Data Reduction

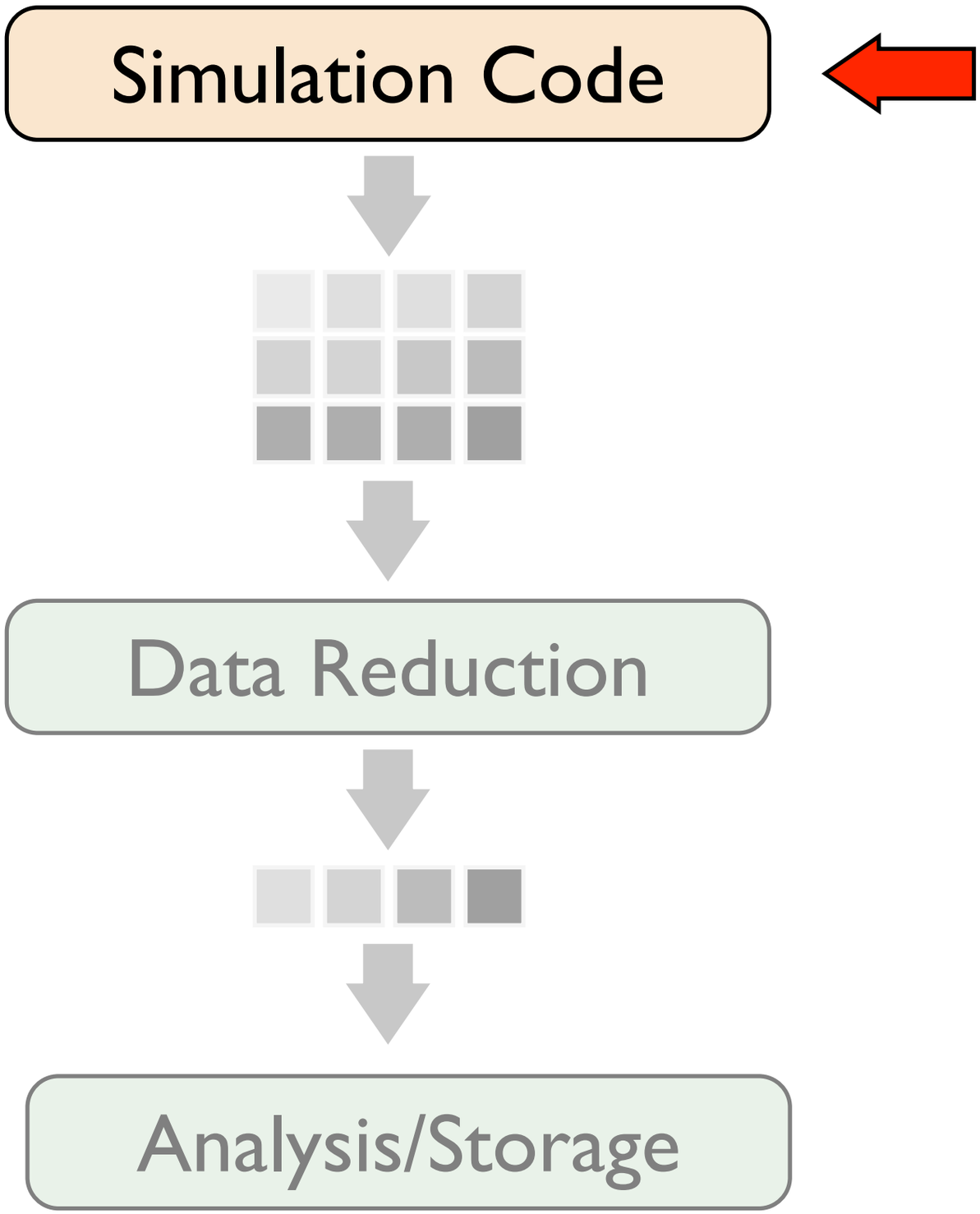


Analysis/Storage

In Situ Streaming Pipeline

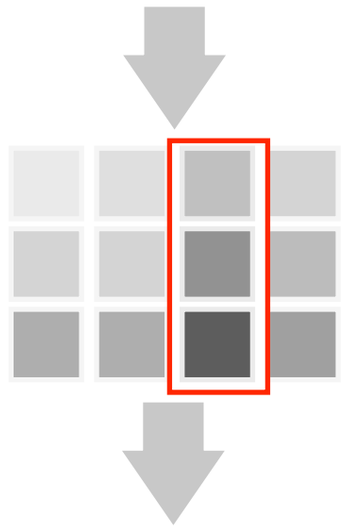


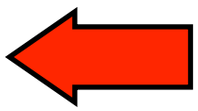
In Situ Streaming Pipeline



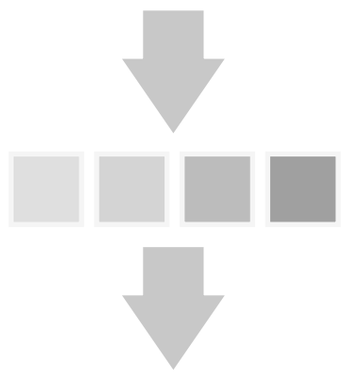
In Situ Streaming Pipeline

Simulation Code



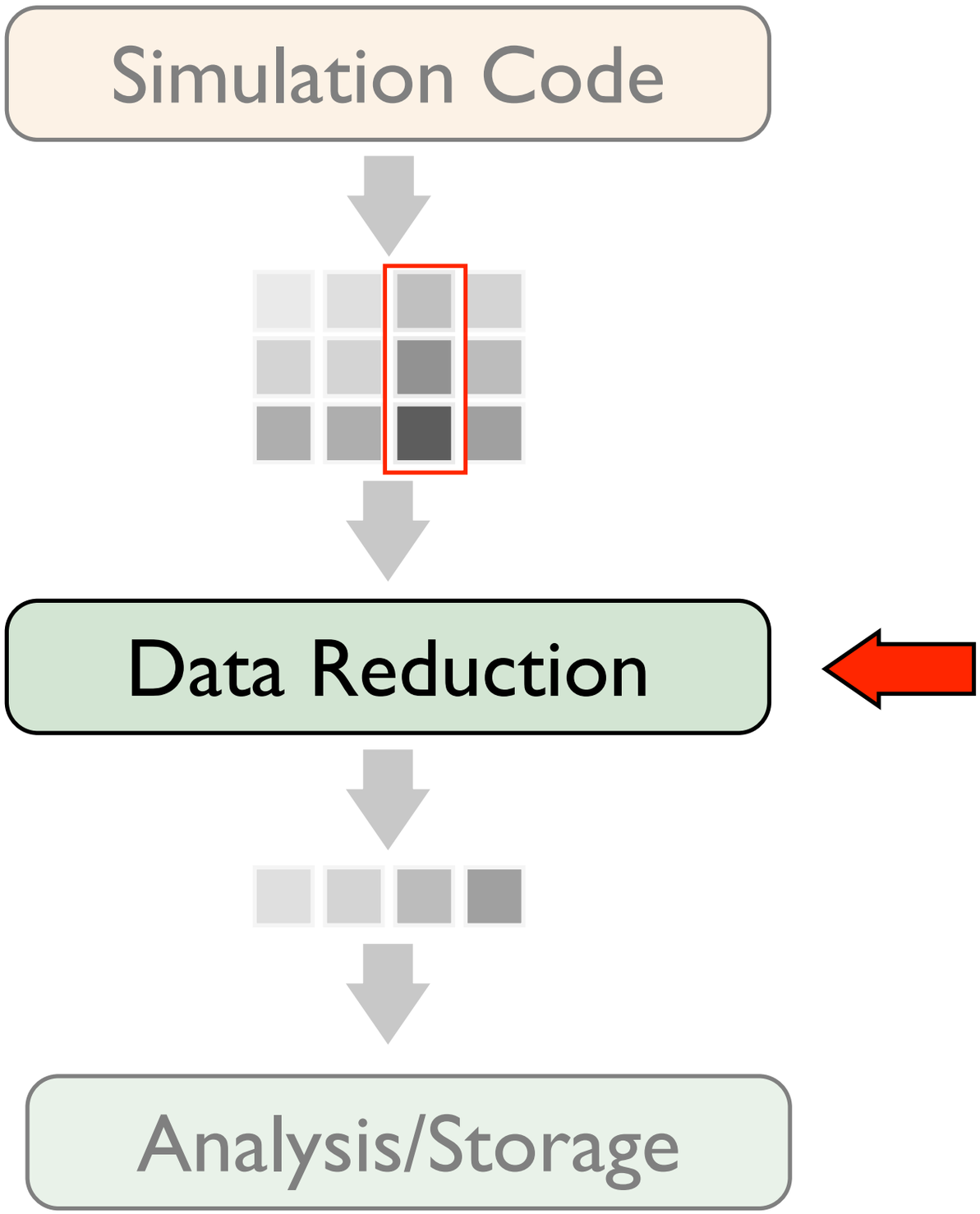


Data Reduction

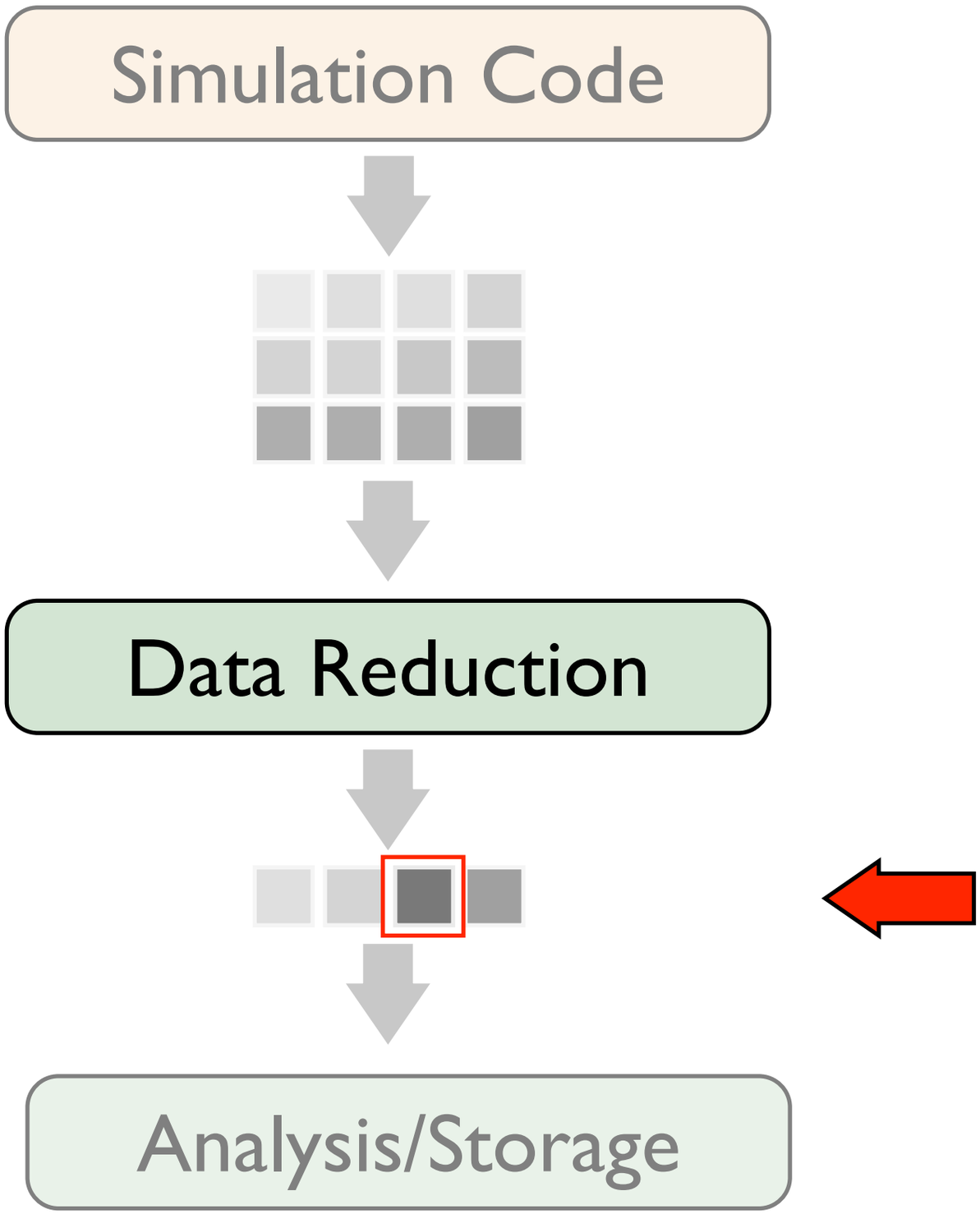


Analysis/Storage

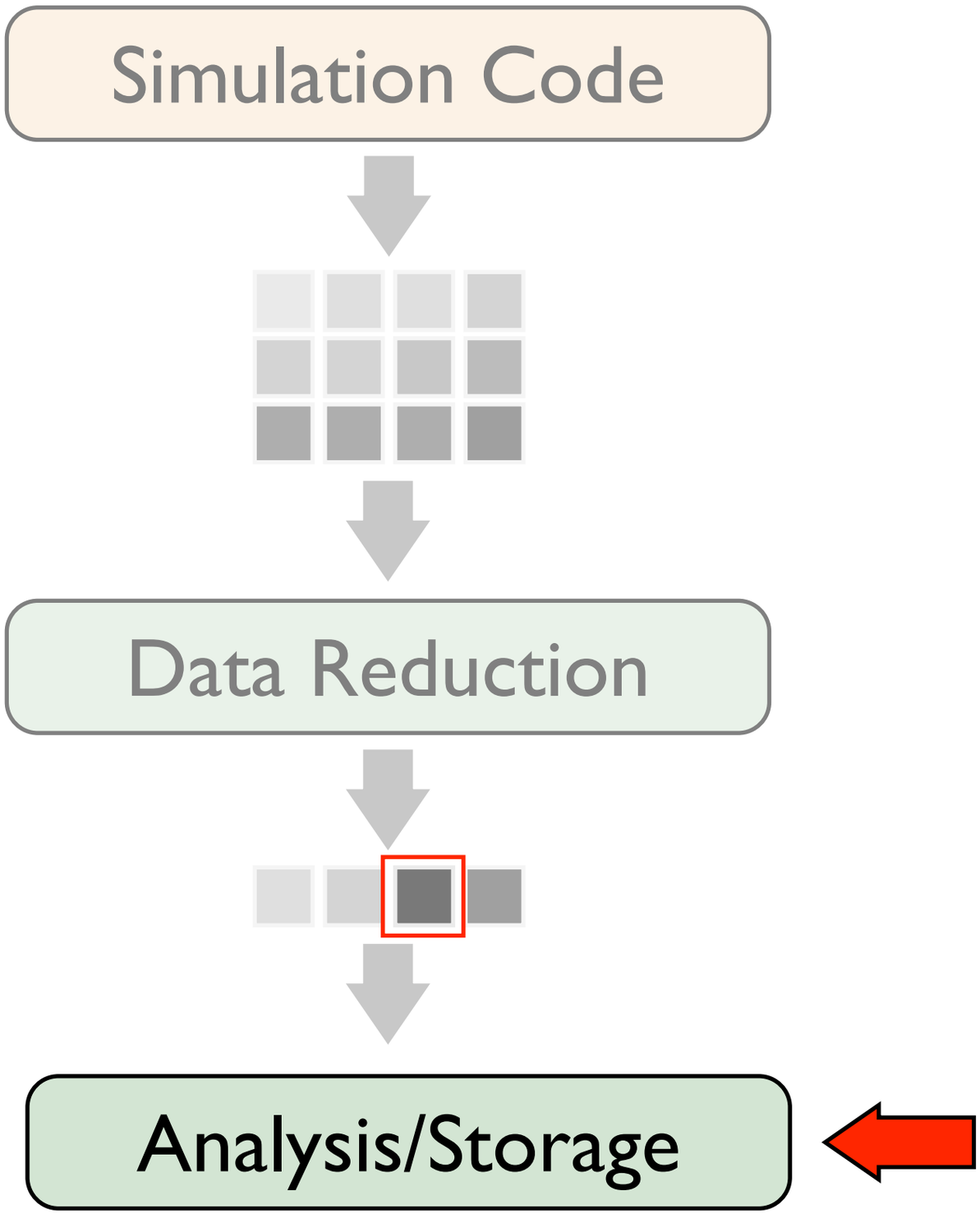
In Situ Streaming Pipeline



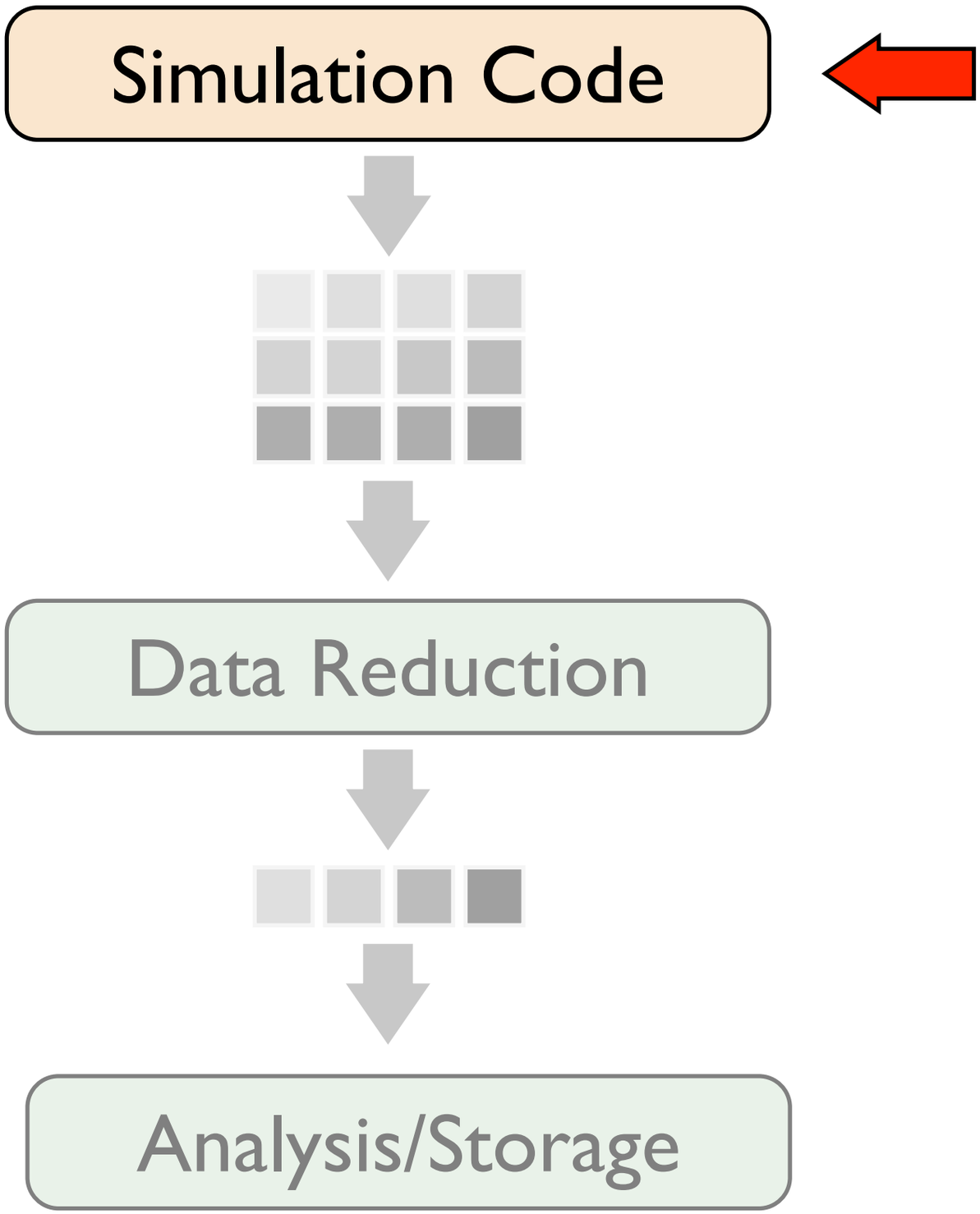
In Situ Streaming Pipeline



In Situ Streaming Pipeline

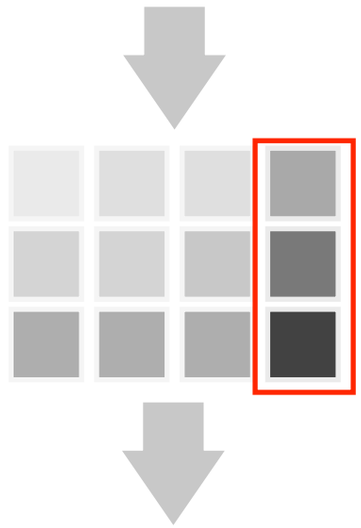


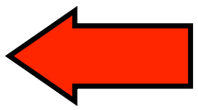
In Situ Streaming Pipeline



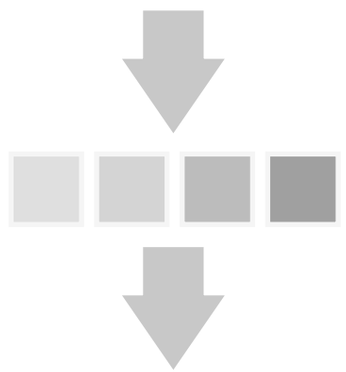
In Situ Streaming Pipeline

Simulation Code



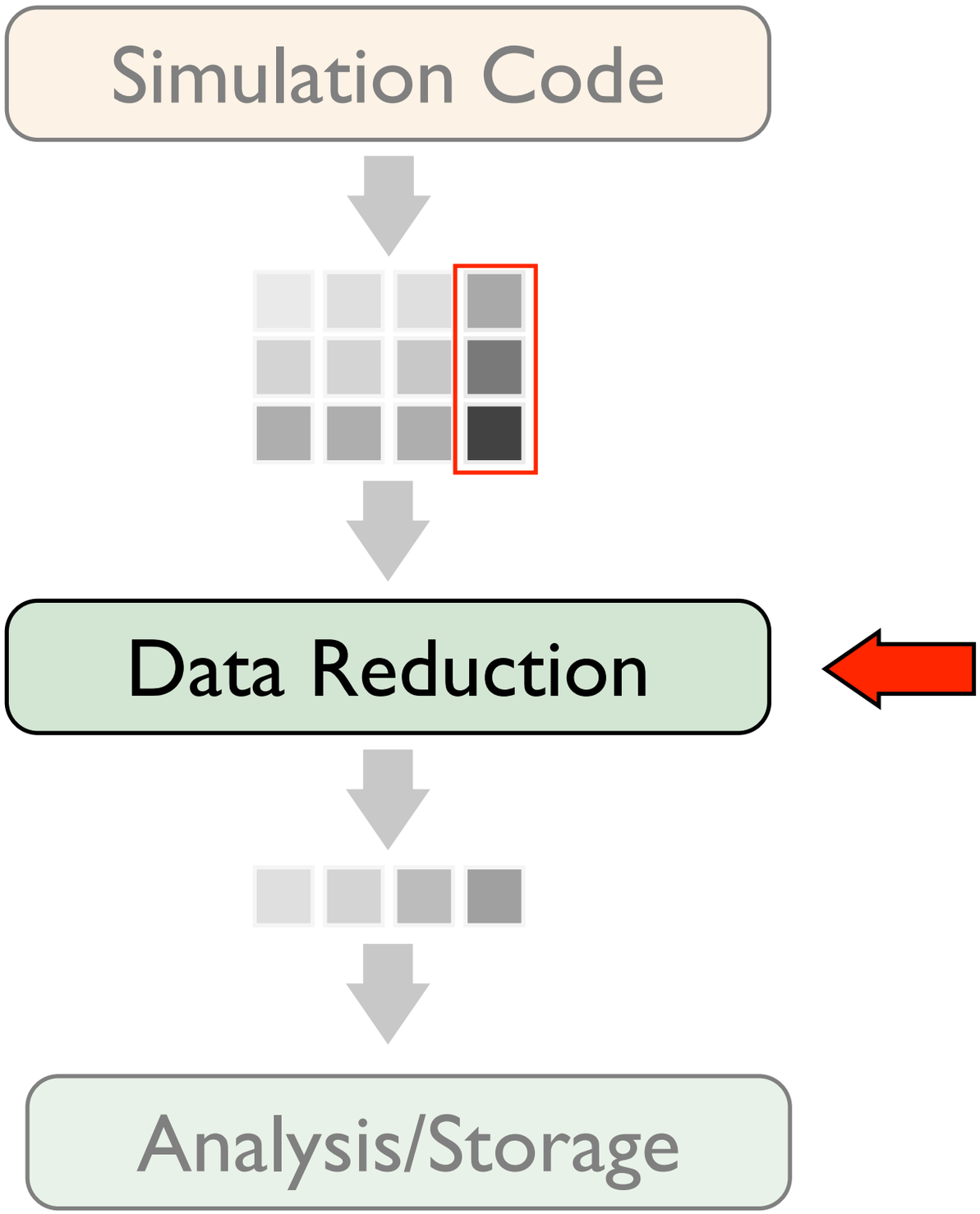


Data Reduction

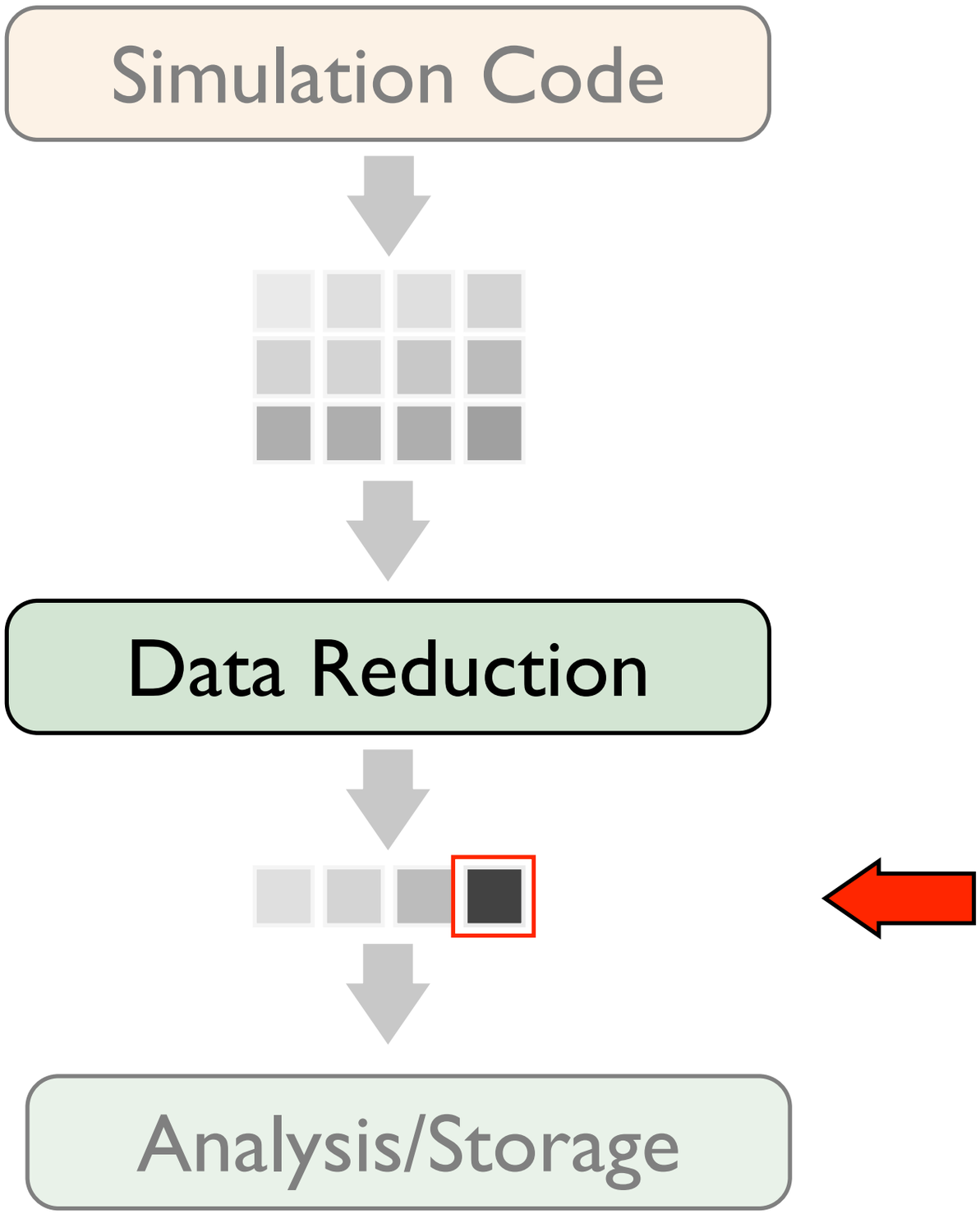


Analysis/Storage

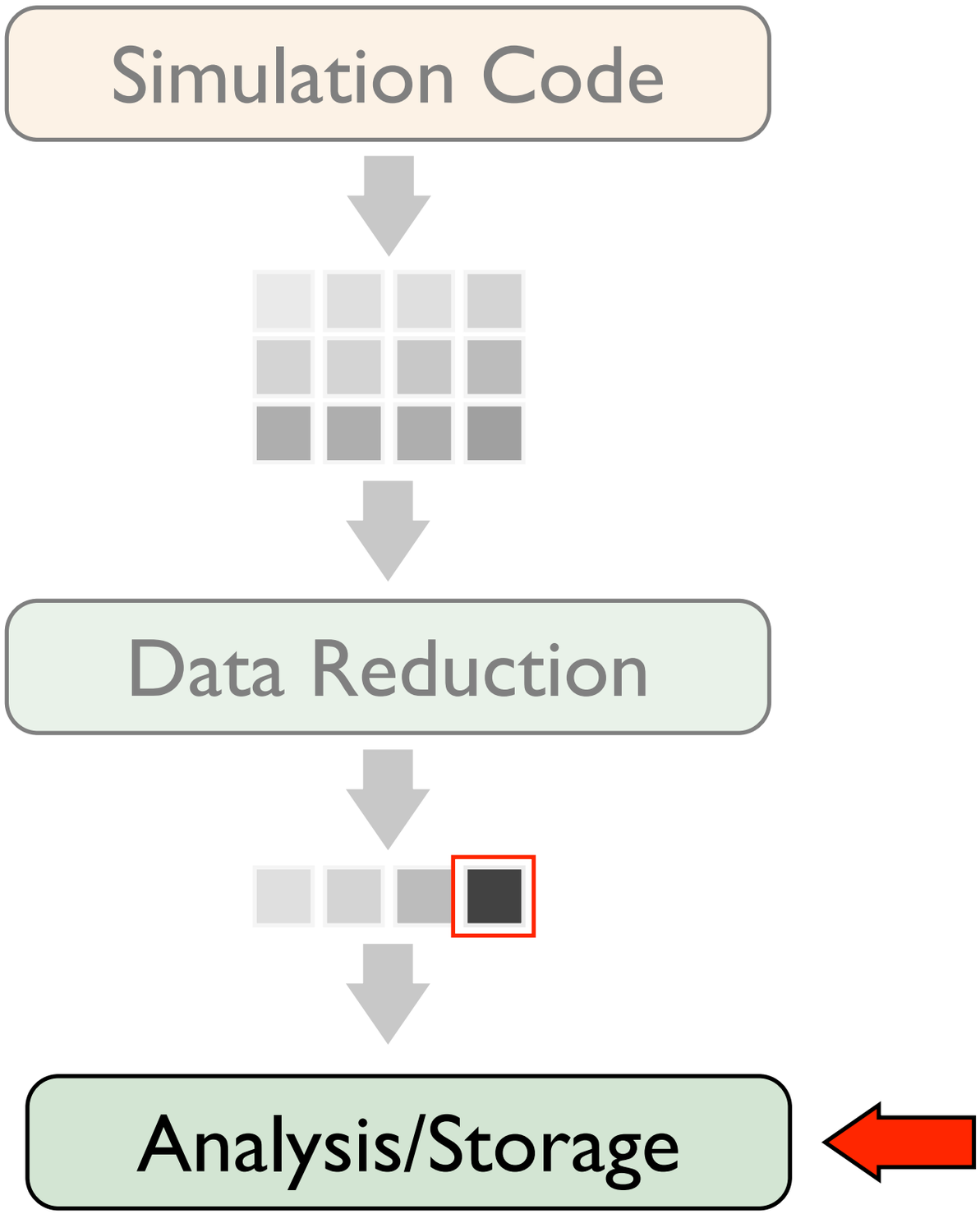
In Situ Streaming Pipeline



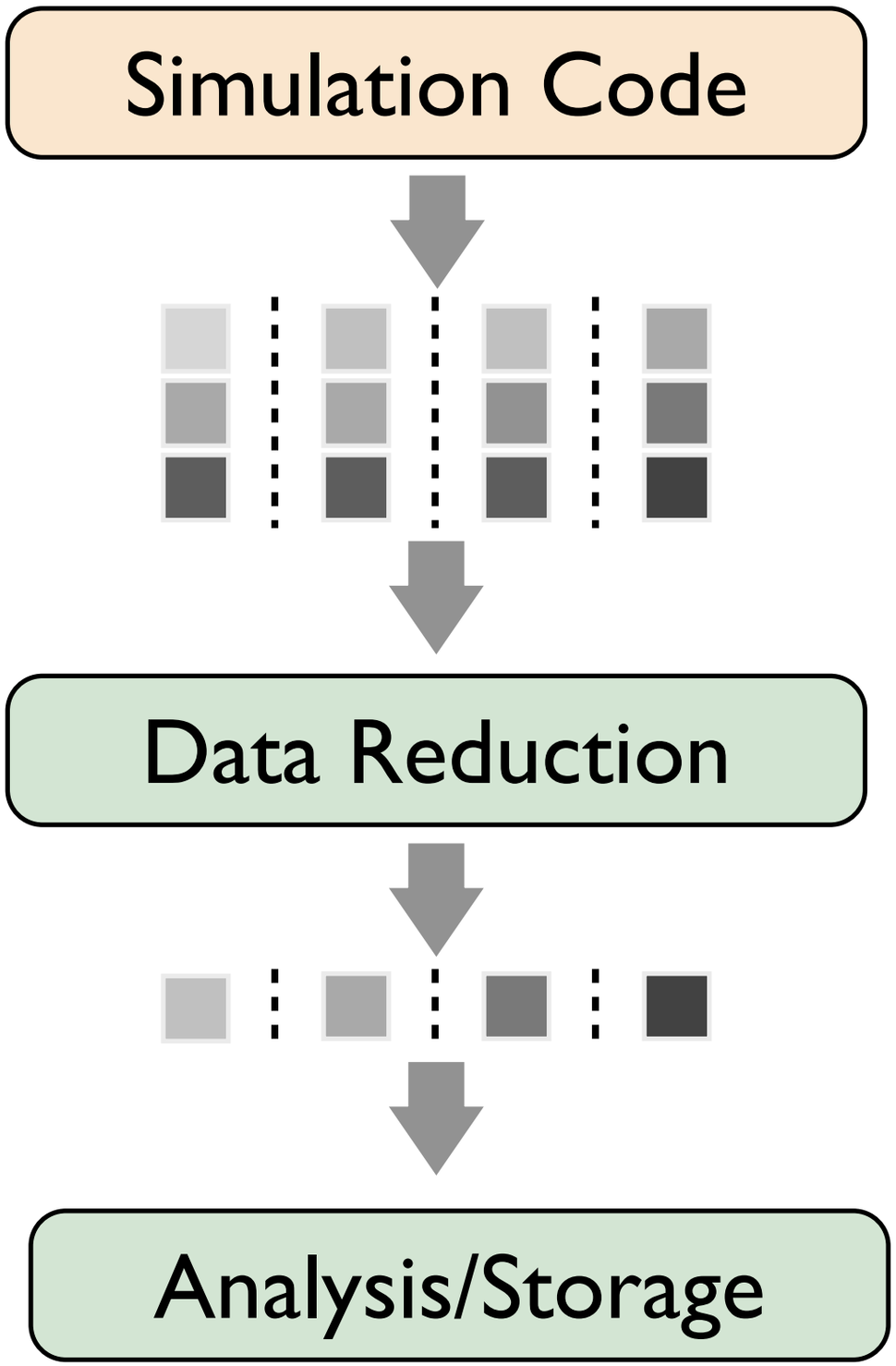
In Situ Streaming Pipeline



In Situ Streaming Pipeline



In Situ Streaming Pipeline

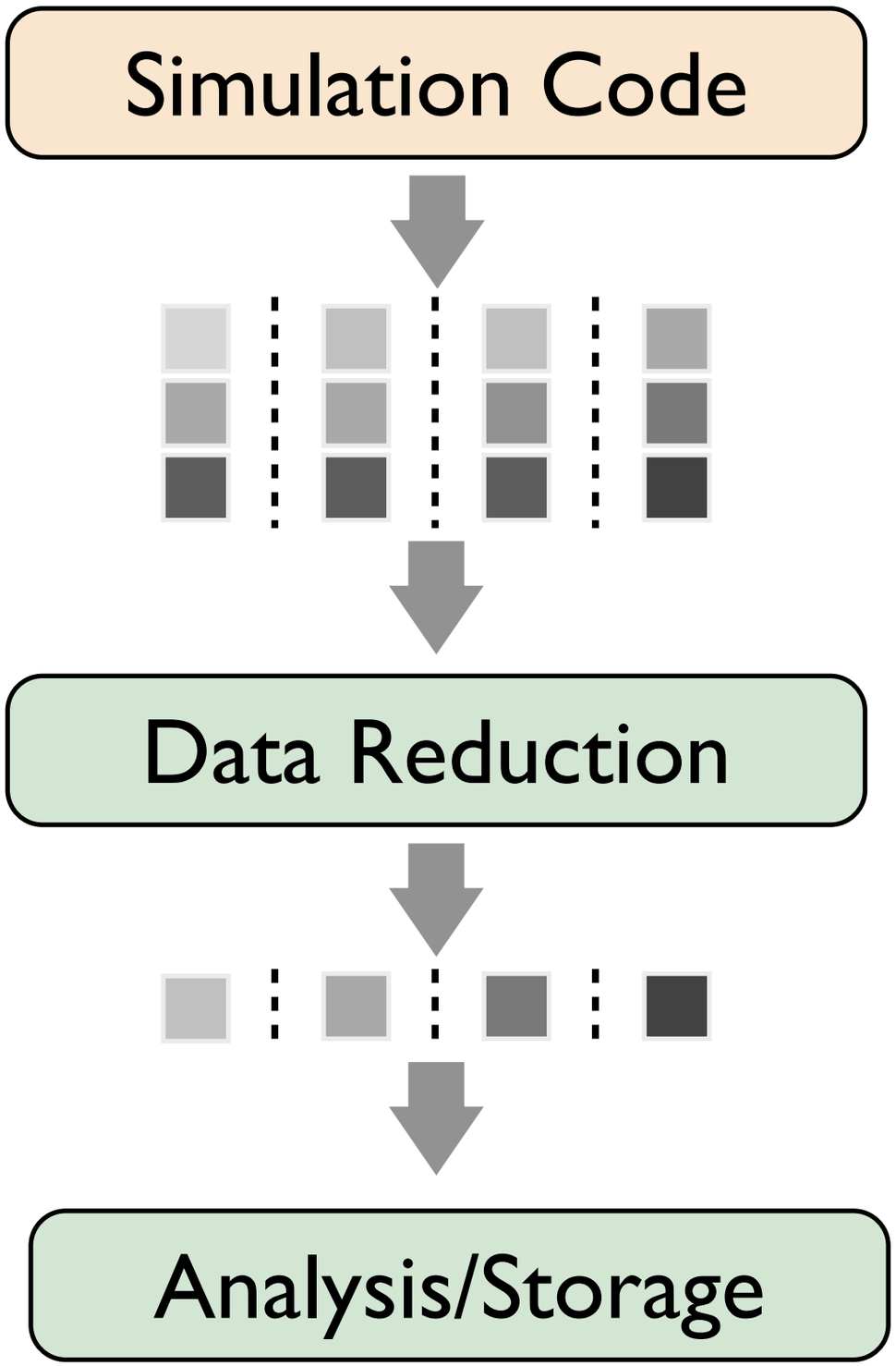


Interleaving Simulation and Visualization tasks

low memory footprint

cache-coherent

In Situ Streaming Pipeline



Interleaving Simulation and Visualization tasks

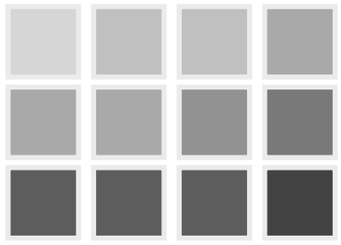
low memory footprint

cache-coherent

How to divide processing power between Simulation and Visualization for parallel execution?

- Too much Sim: high memory usage
- Too much Vis: low performance

Example with threaded VTK



Input: 600 compressed images

Simulation Code

Image decompression from disks

Visualization

Perform edge detection

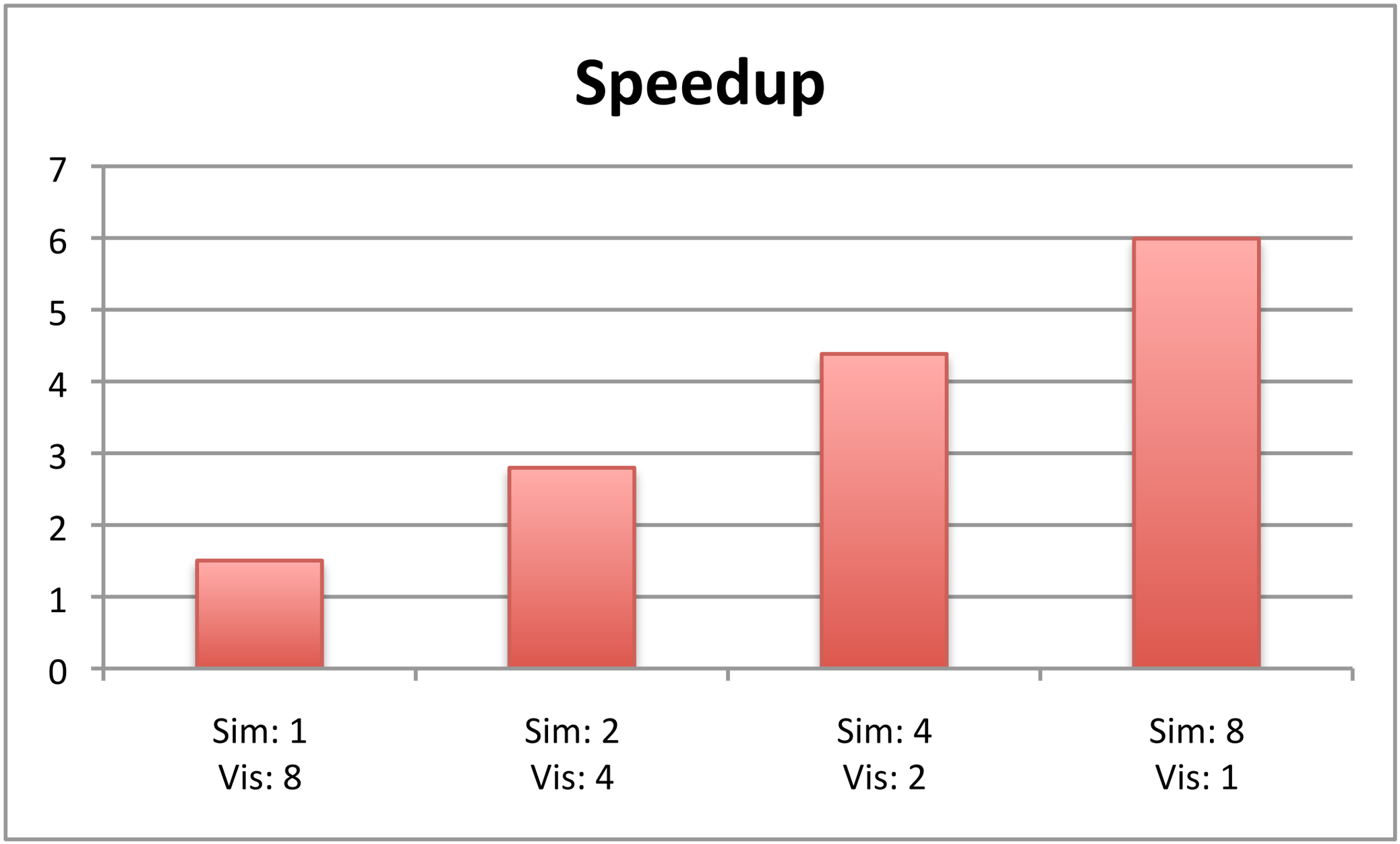
Data Reduction: resize images

Analysis/Storage

Store resized images to disk

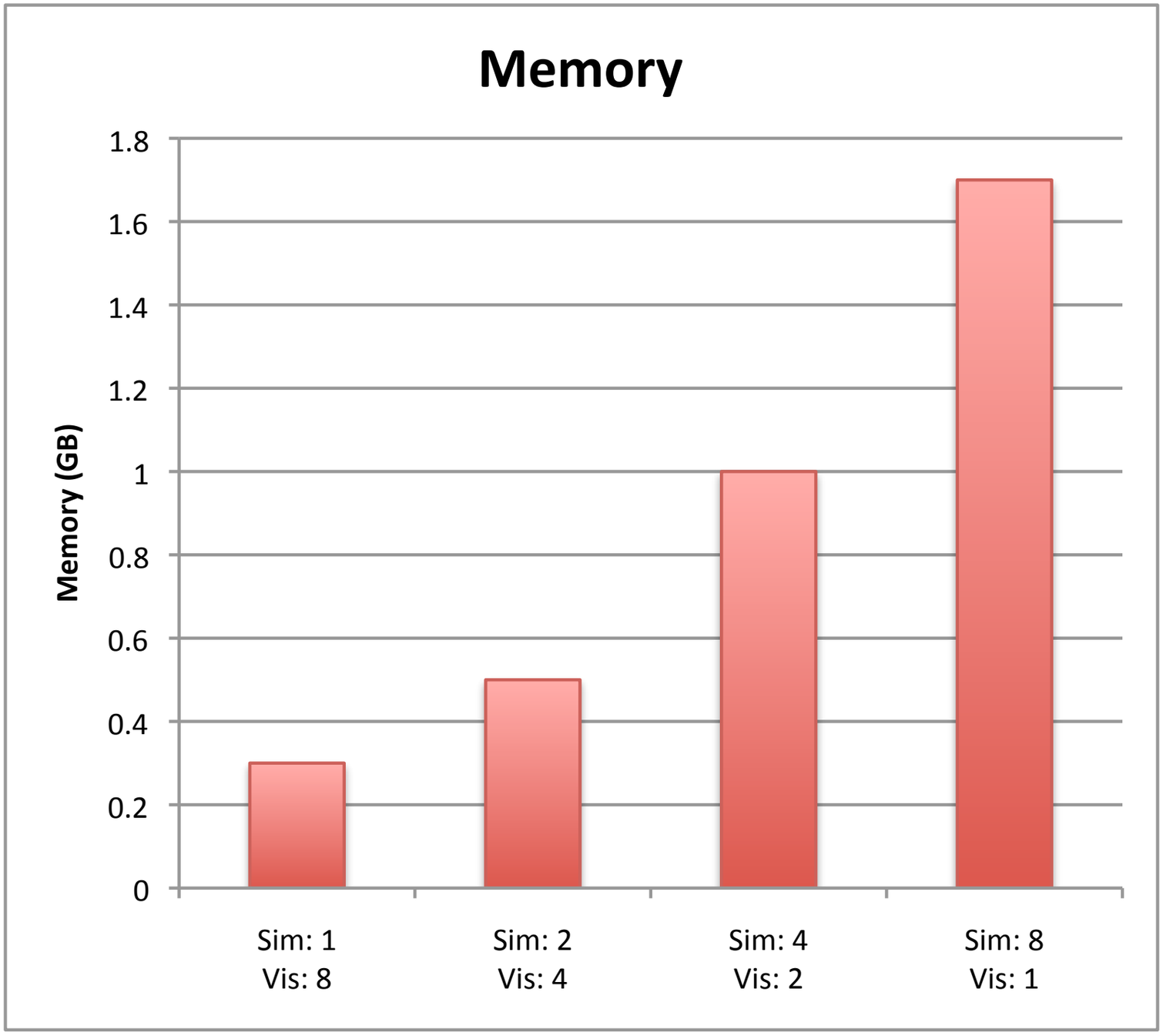
Example with threaded VTK

8-core machine



Simulation (image decoding/disk I/O) needs more processing power

Example with threaded VTK



Results in more memory usage

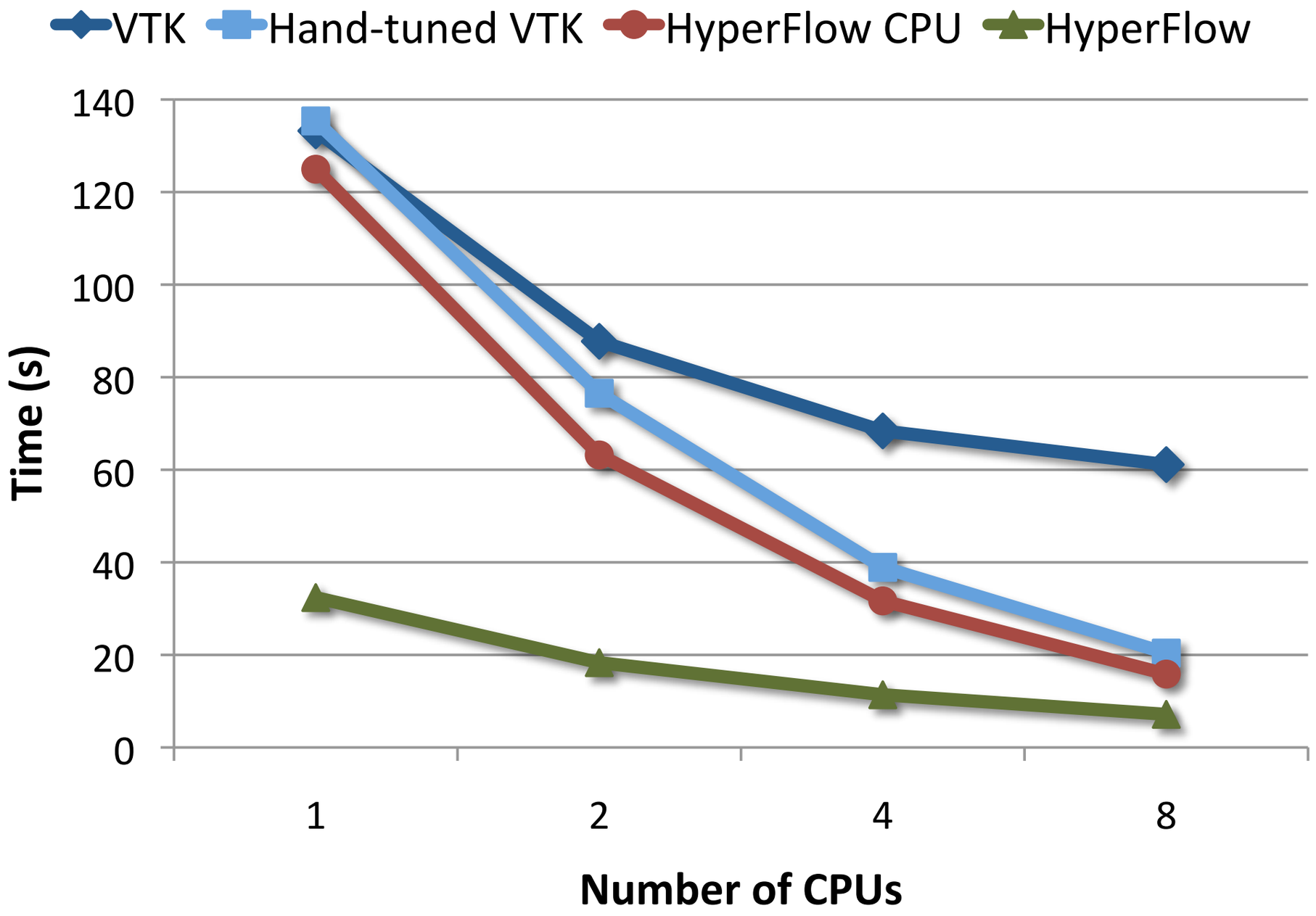
Example with threaded VTK

- Fixed resource allocation sacrifices memory footprint for performance
- Needs an adaptive scheduler to leverage the processing power between simulation and visualization

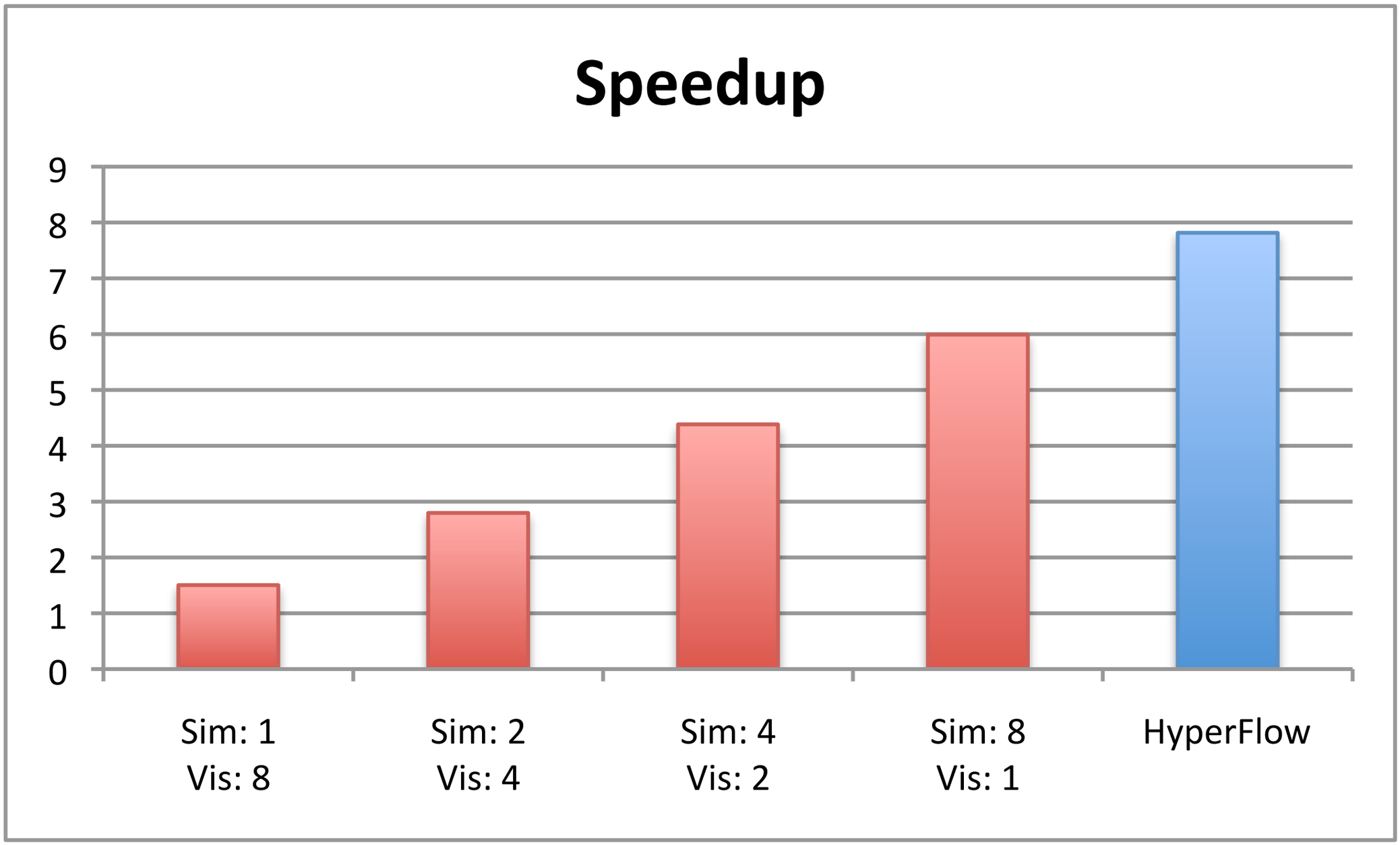
In Situ Dataflow Architecture -- Using HyperFlow

- Tie simulation and visualization together
- Support streaming
- Minimize memory footprint
- Efficient parallel execution
- Lightweight and straightforward integration

Threaded VTK Example

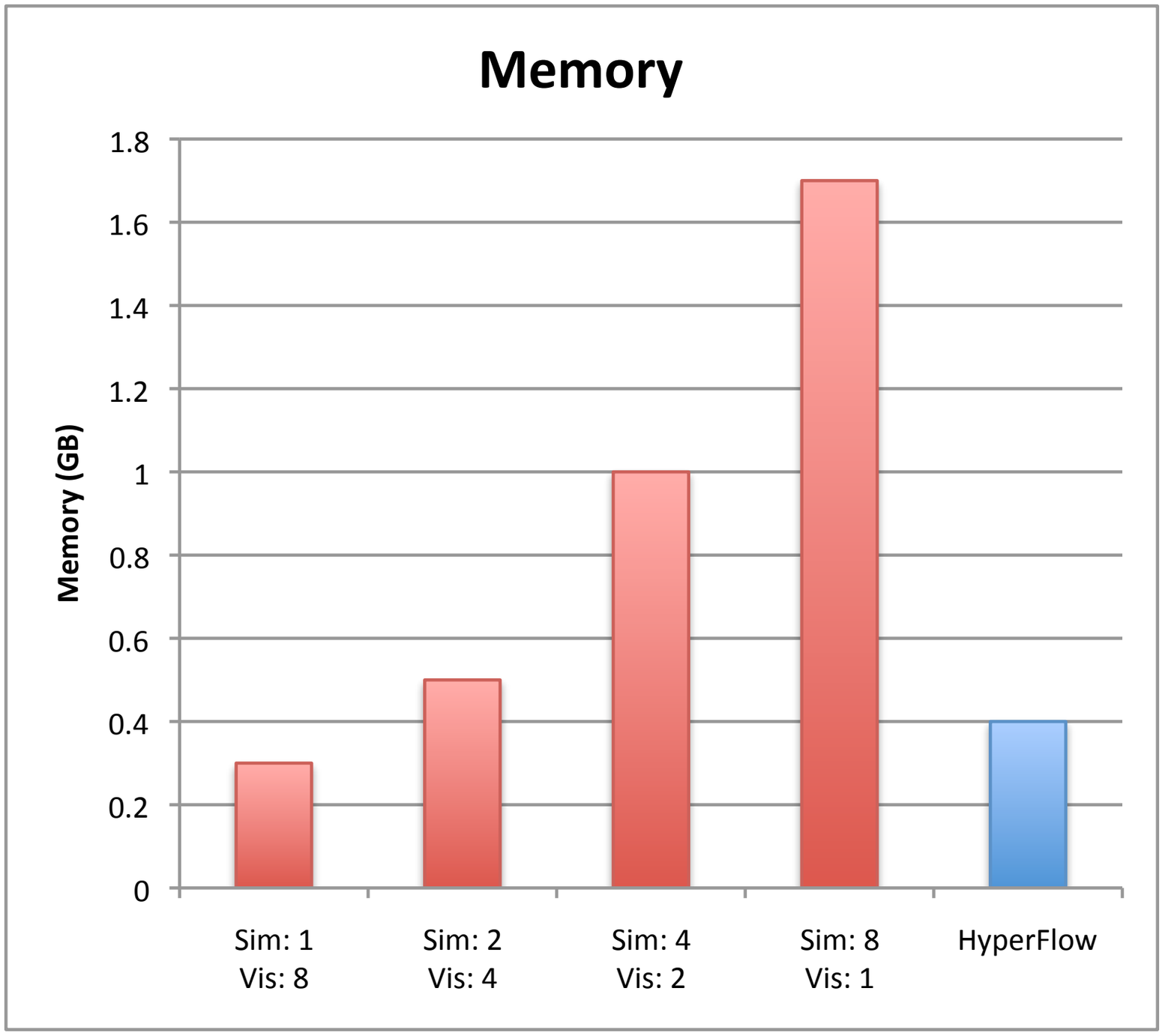


Threaded VTK Example



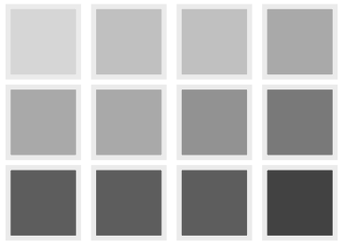
Better performance...

Threaded VTK Example



... and lower memory footprint!

Isosurface Example



Input: 2048x2048x1920 grid

Simulation Code

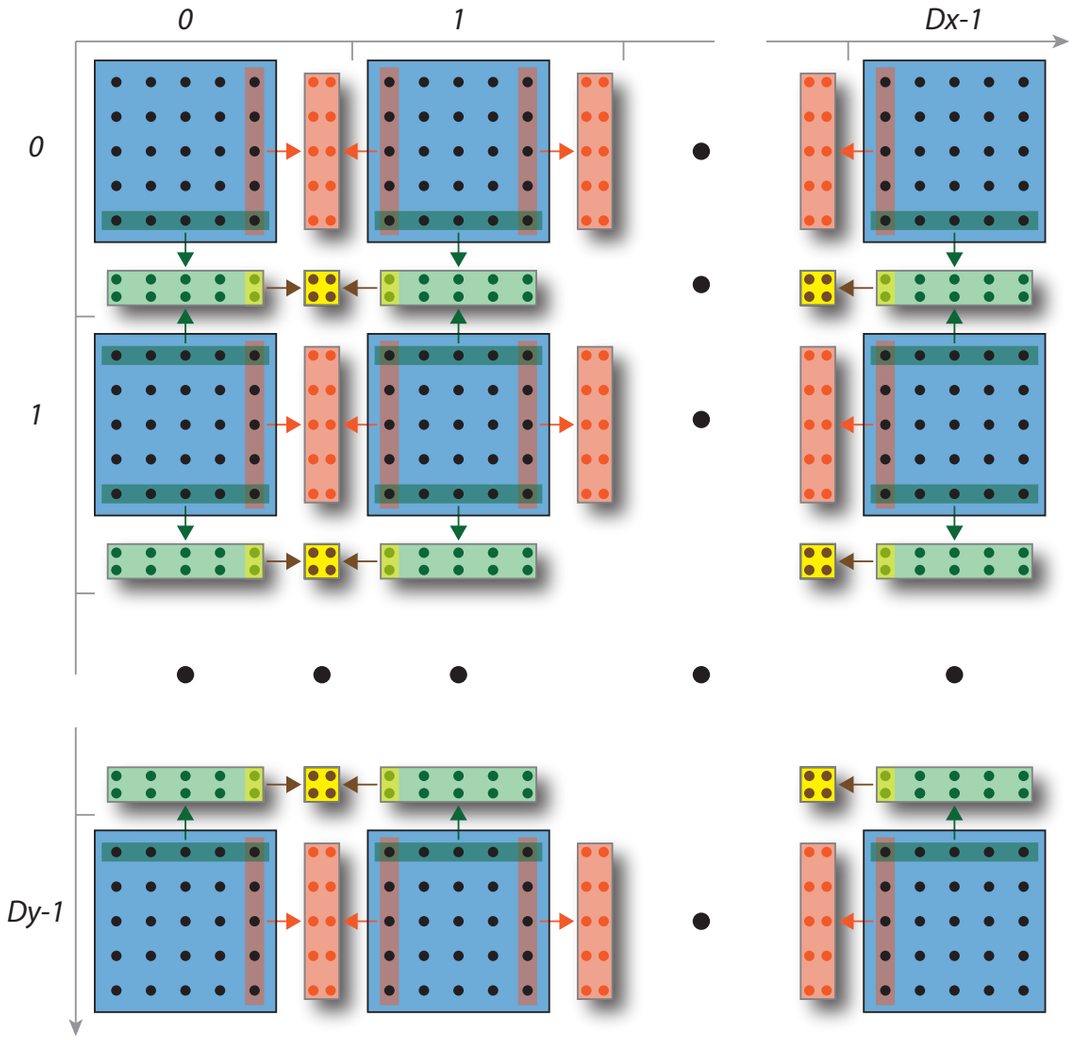
Streaming data from disks

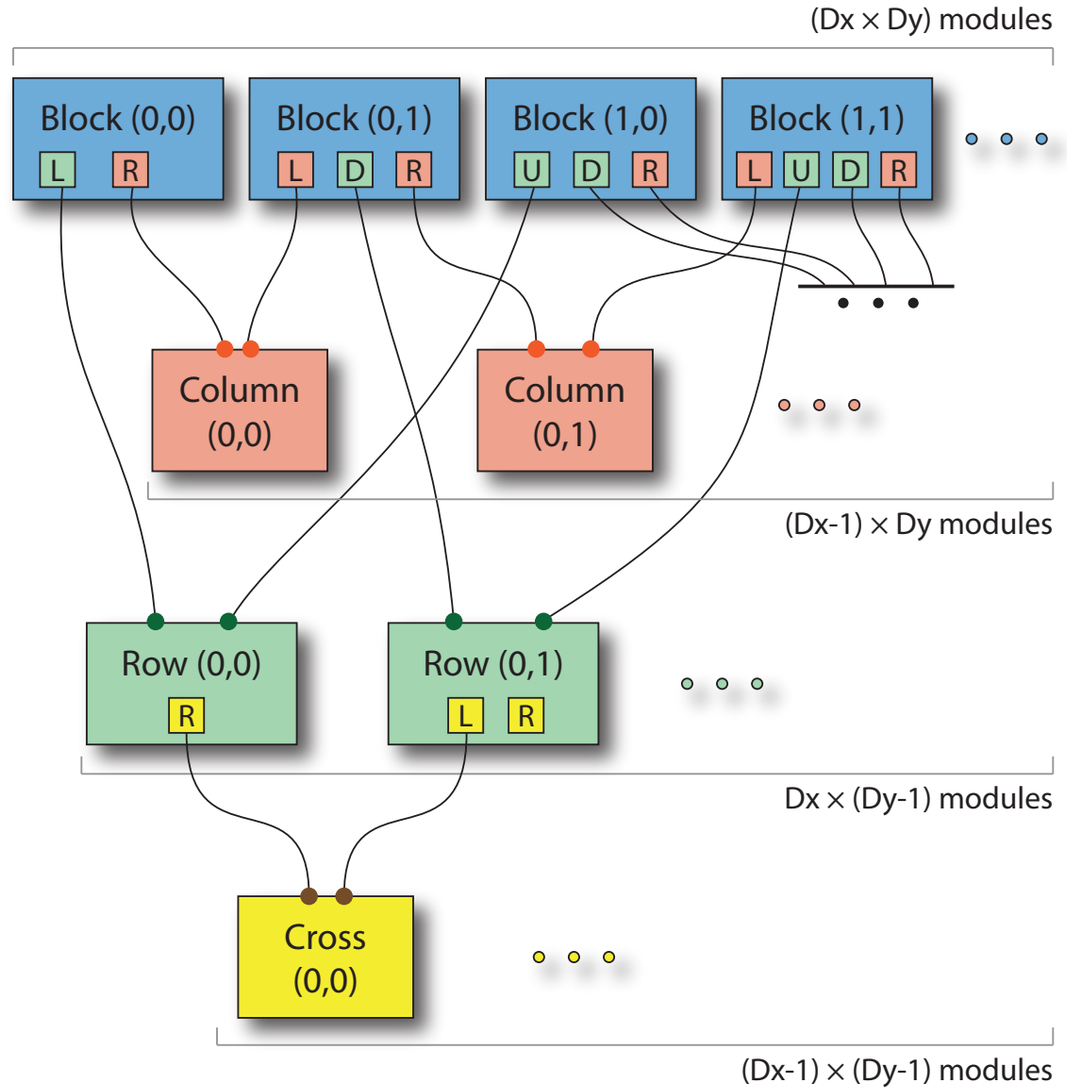
Visualization

Isosurface Extraction

Each visualization code processes data from several outputs of the Simulation code!

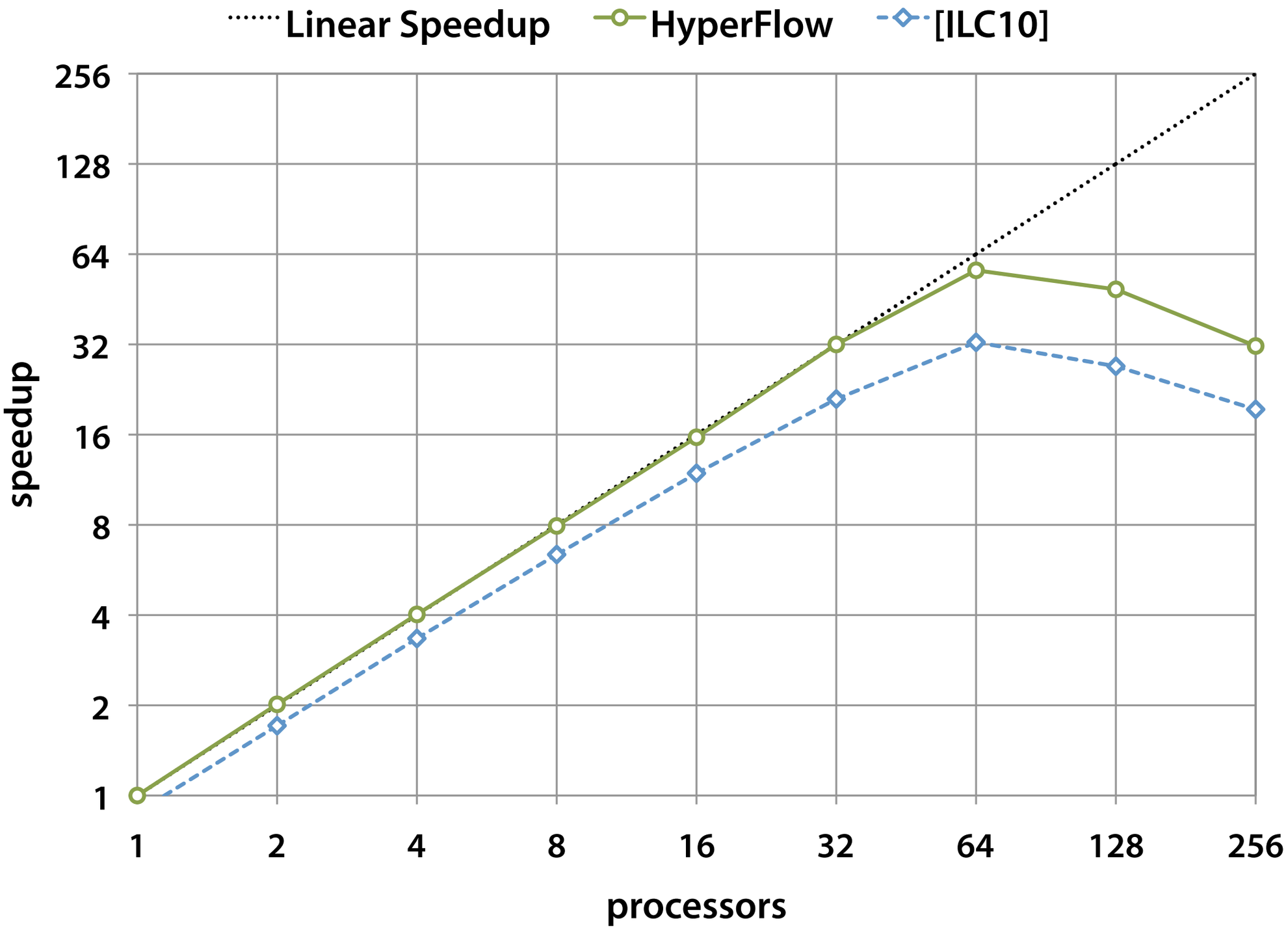
HyperFlow simplifies data dependency



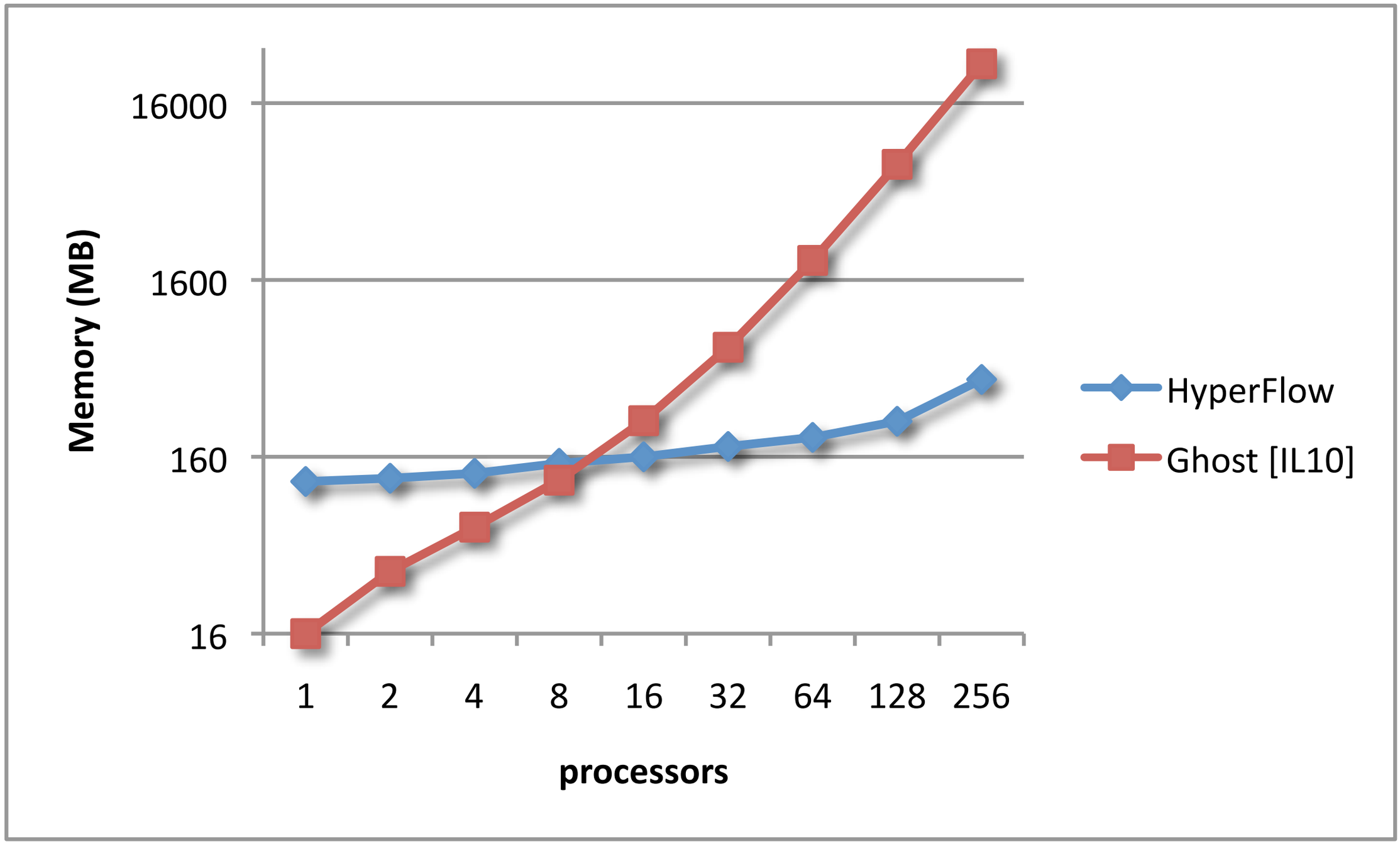


- Just specify the data communication between modules
- HyperFlow will optimize the data transfer scheduling strategy to best use the system resources (including memory usage)

Isosurface Performance



Isosurface Performance

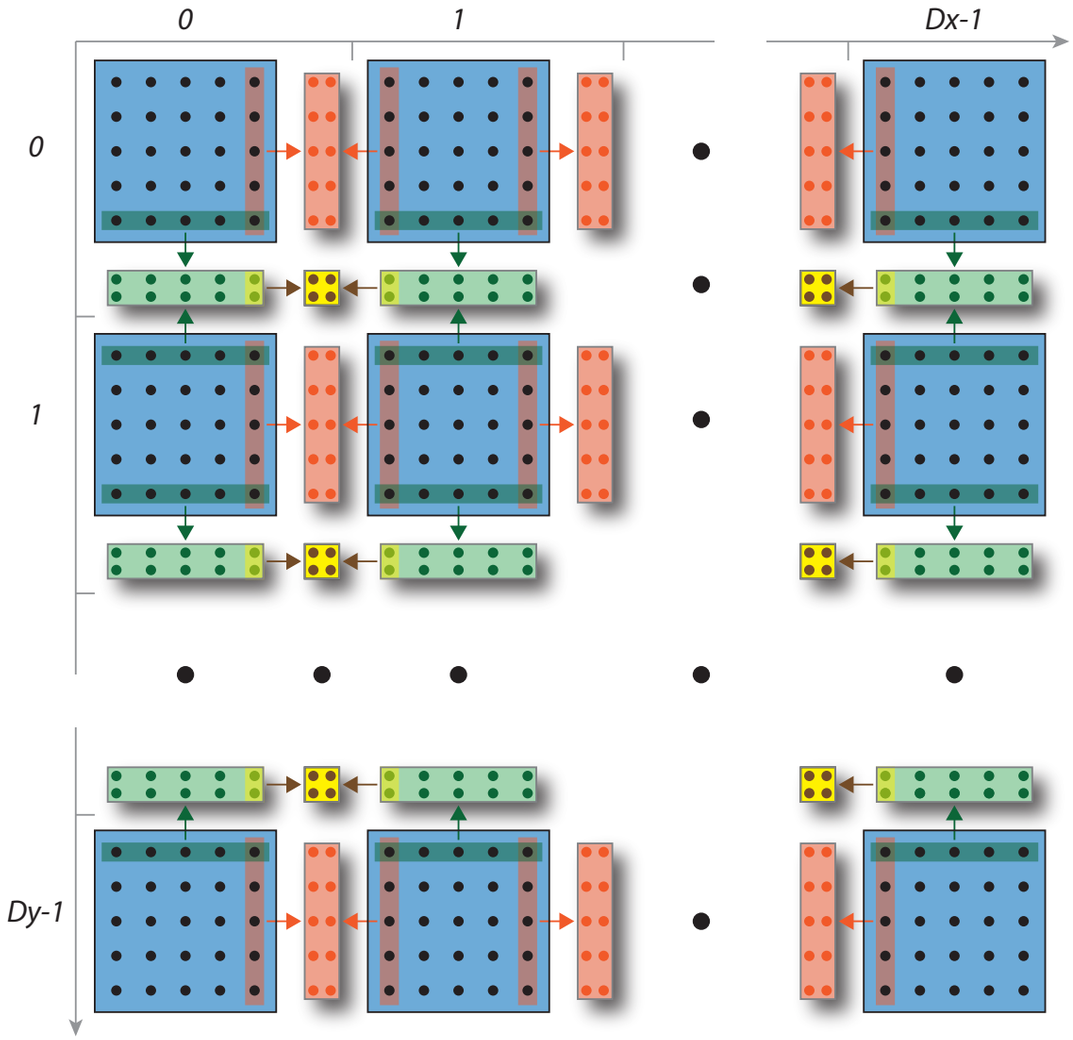


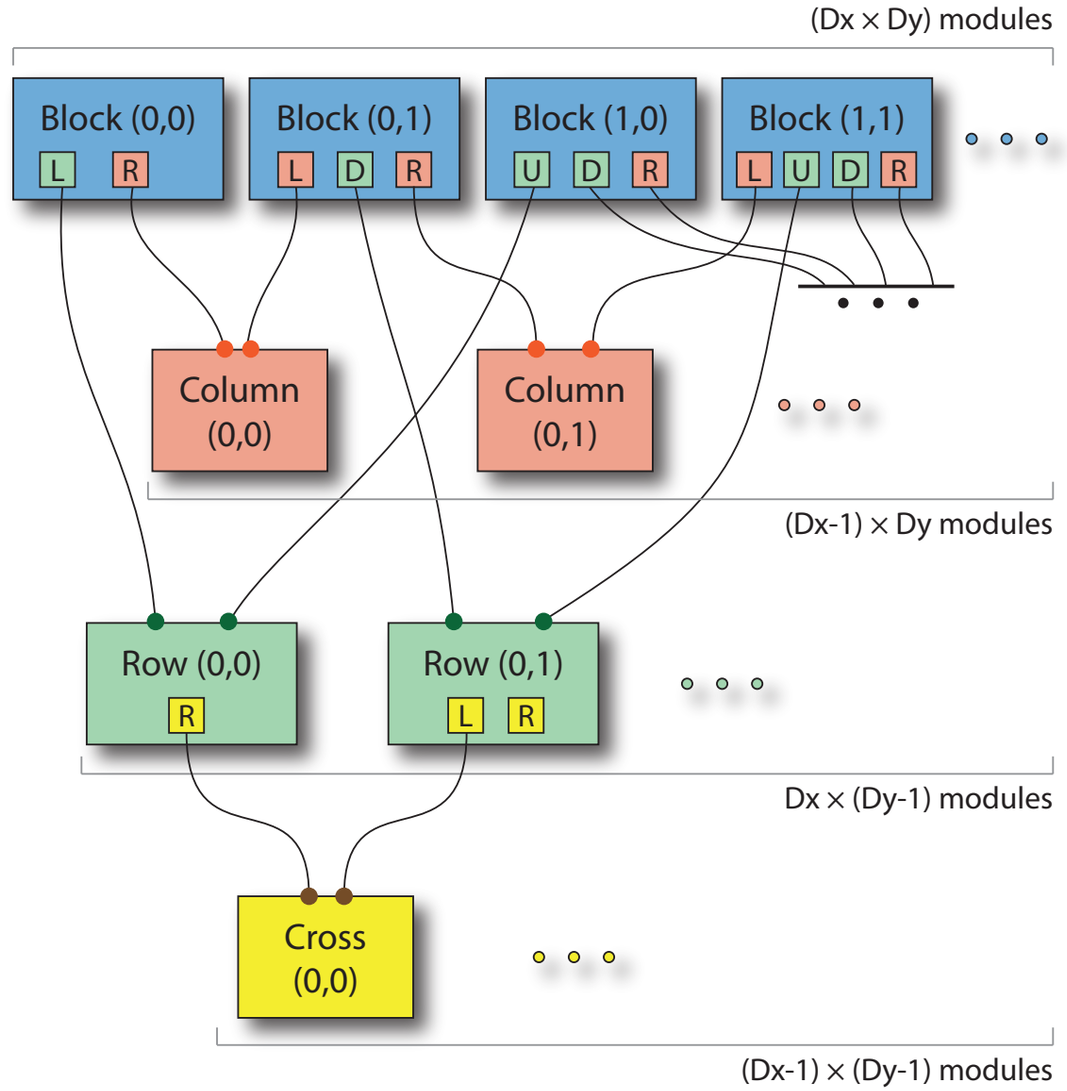
Higher base memory usage because of the unoptimized data dependency but has better scalability as well

Lessons Learned

- Streaming should be used to avoid the memory challenge of in situ vis
- Simulation and Visualization need to be closely integrated (in a dataflow archicture) to achieve efficient memory usage and performance

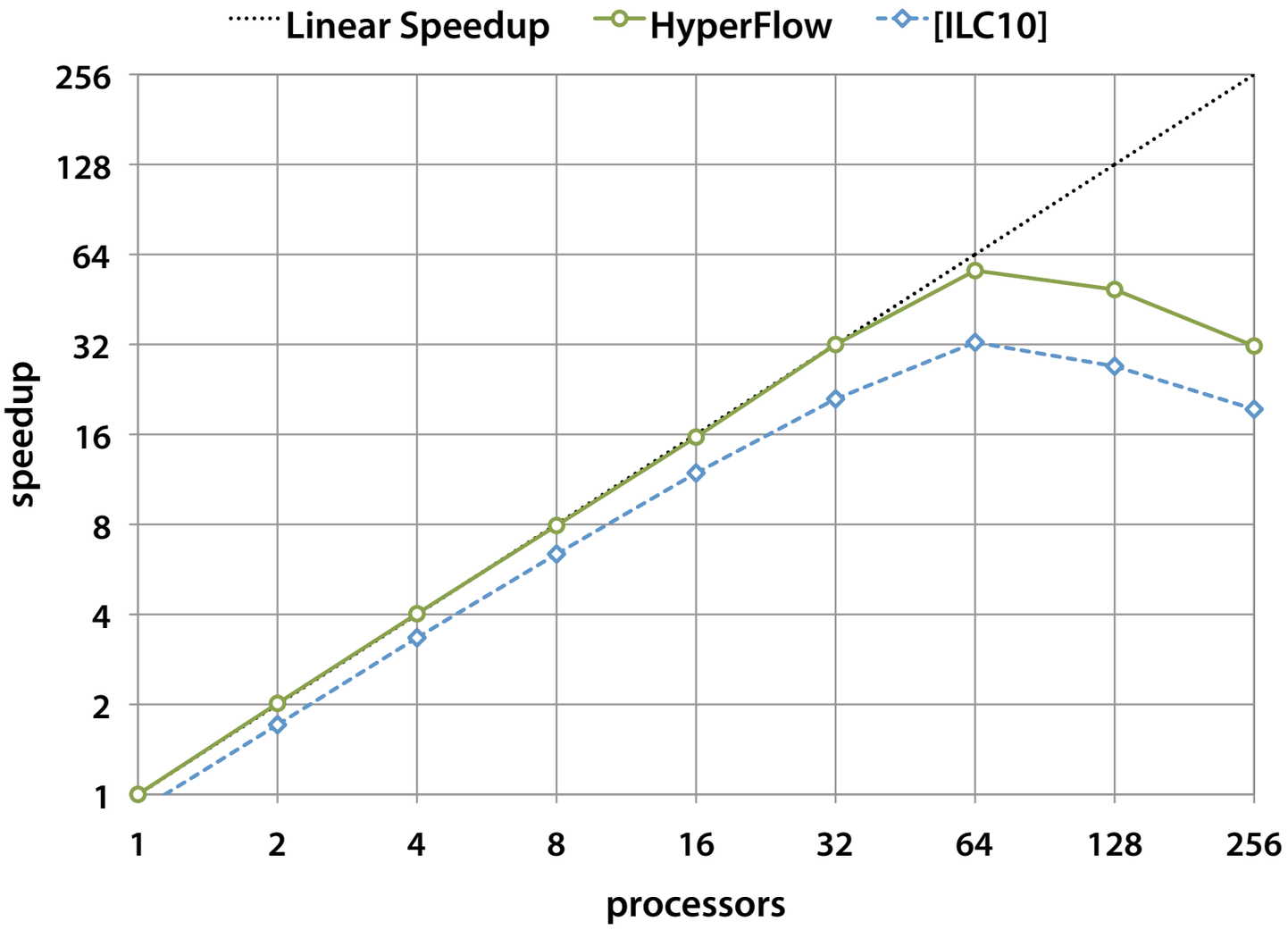
Isocontouring Structured Grid

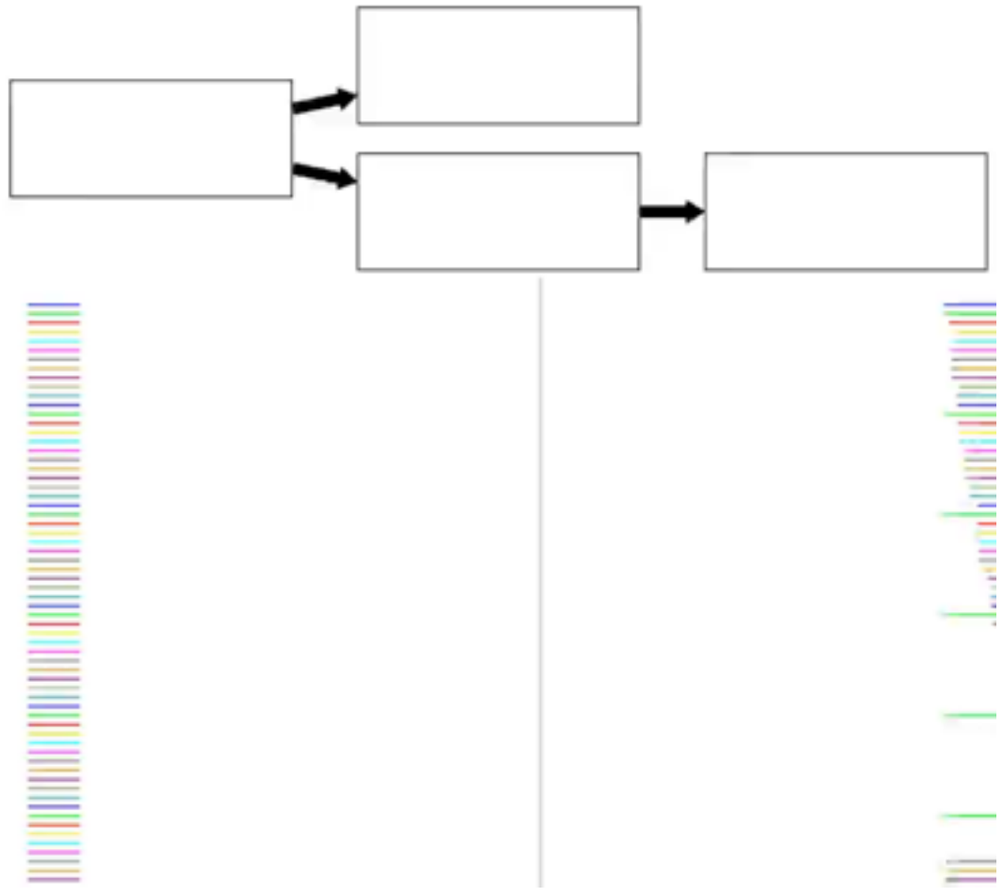




- Mix multiple data- and task-parallel phases
- Compared to an MPI, hand-tuned parallel streaming algorithm by Isenburg et al [2010].

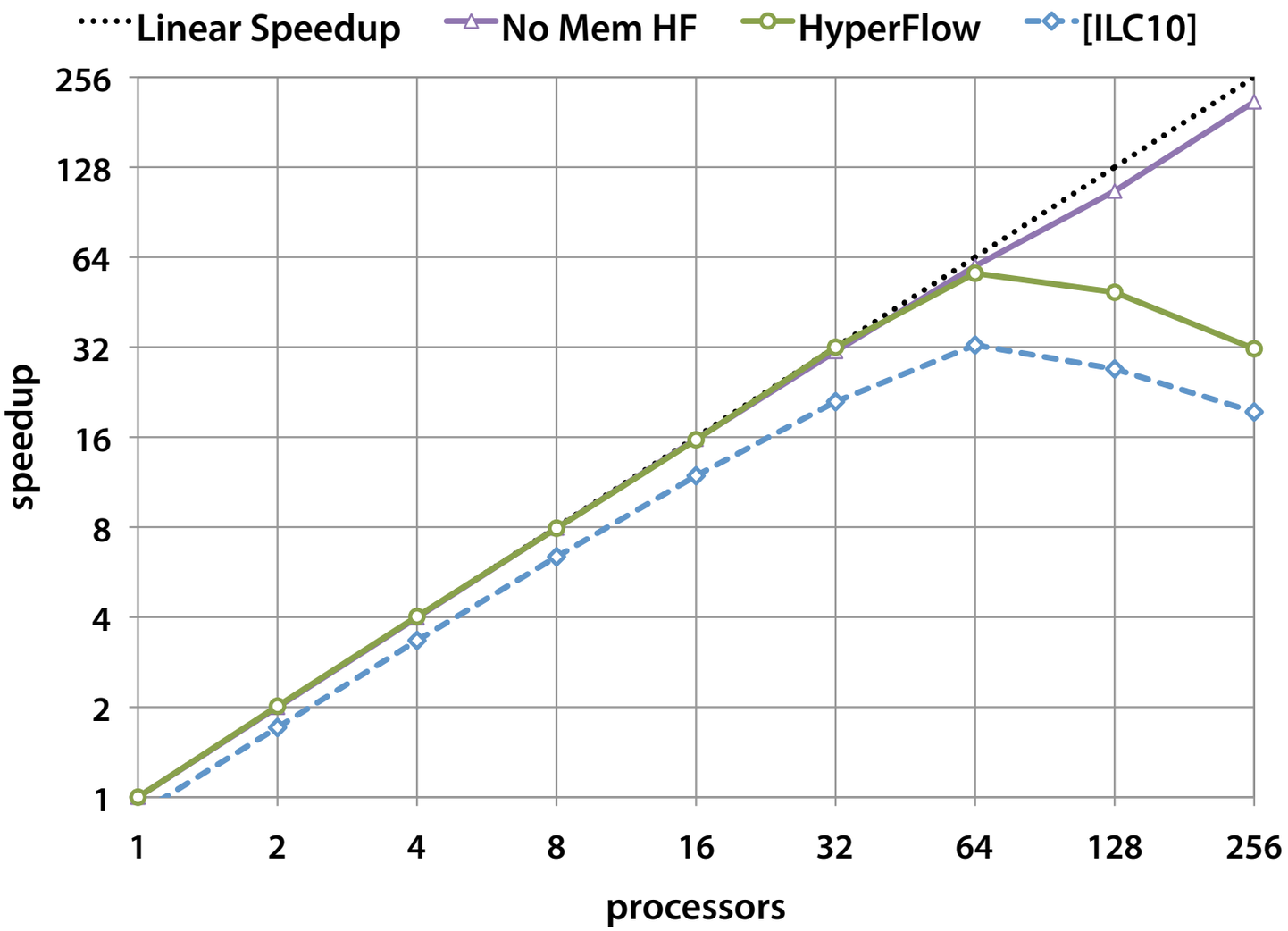
Isocontouring Structured Grid





- Ran on the UV machine with 264-core
- HyperFlow incurred less overhead than MPI
- Both methods only scale to 64-core

Scalability Issue



1. Remove computation

   bad scaling

2. Remove memory access

   good scaling

3. Measure memory access bandwidth

   sustained ~500GB/s after 64 nodes

Saturation of memory bandwidth is the cause