A Case Study in Tightly Coupled Multiparadigm Parallel Programming

Sayantan Chakravorty
Aaron Becker (abecker3@uiuc.edu)
Terry Wilmarth
Laxmikant V. Kalé

Parallel Programming Lab (charm.cs.uiuc.edu)
University of Illinois, Urbana-Champaign

LCPC '08

There is no shortage of parallel programing models

Global Arrays

BSP

X10

OpenMP

High Performance Fortran

NESL

MPI

Parallel Matlab

Charm++

StreaMIT

STAPL

DPJ

Unified Parallel C

HTA

Chapel

Why so many?

- Each is good at something different
- Some aim for maximum performance,
  others emphasize productivity and effective abstractions
- Some models are especially well-suited for particular problem domains
  - Cilk: state-space search
  - Co-Array Fortran: linear algebra
  - MapReduce: data mining

- Many models, coexisting happily
- Easy interoperation and reuse (especially with MPI)
- Choose right level of abstraction, based on performance requirements
- Shared resource management

Related Work

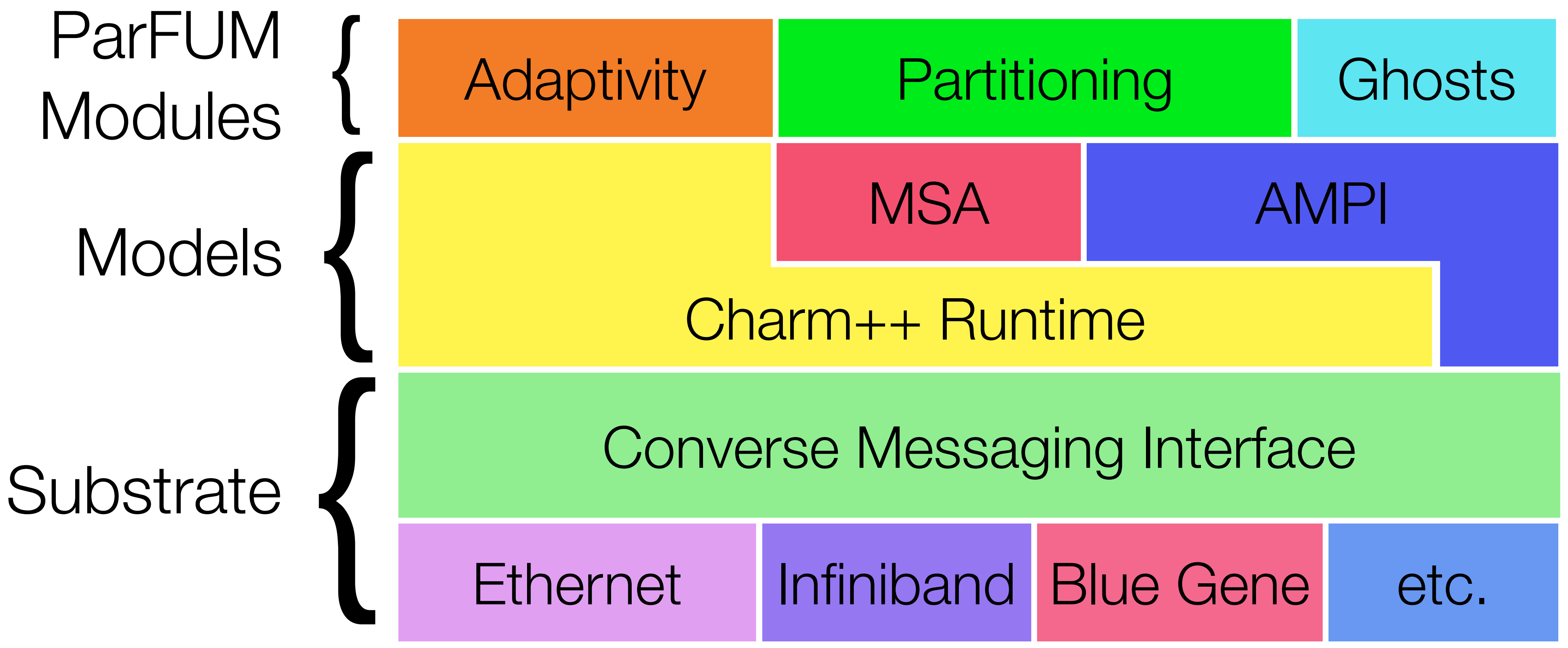
- Symponents
- MPI+OpenMP, Extended OpenMP
- TPVM
- Fortran M
- Lots of serial multi-language systems, e.g. .NET

ParFUM: a Multiparadigm Library

ParFUM

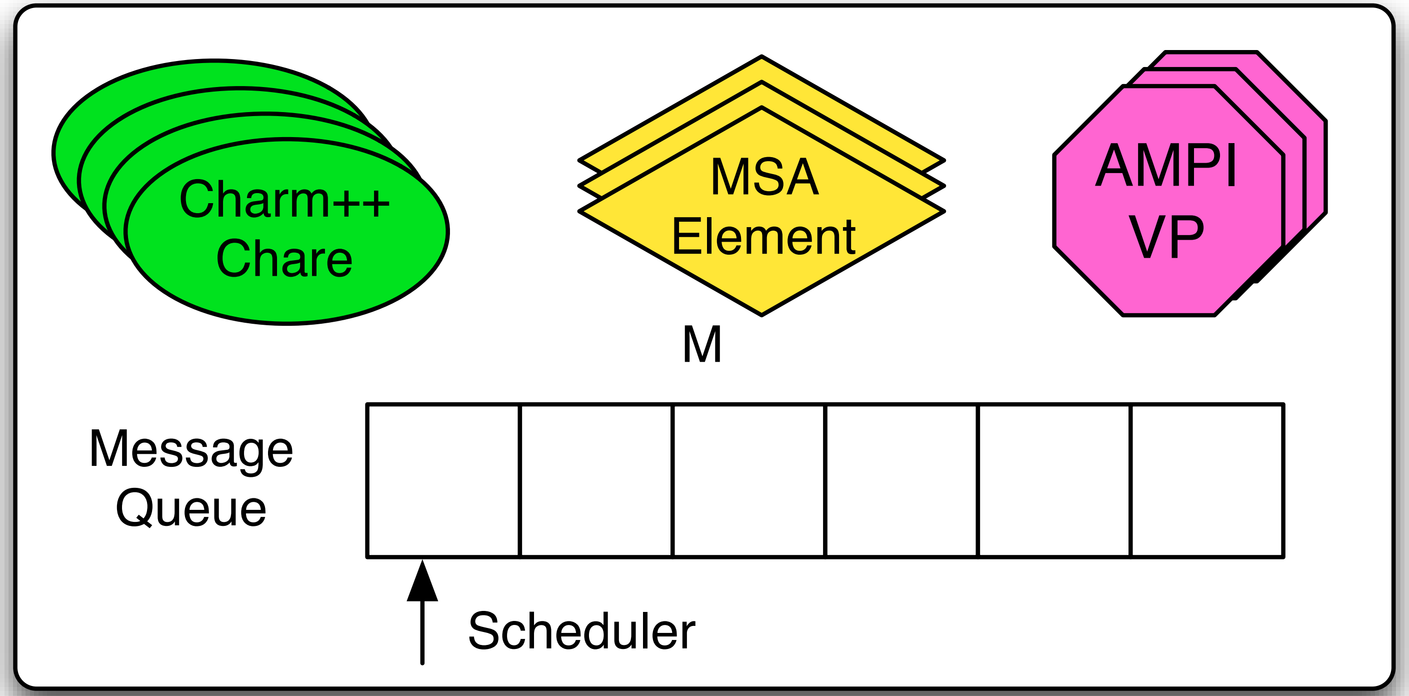
- **Par**allel **F**ramework for **U**nstructured **M**eshing
- Goal: simplify common tasks for parallel unstructured meshing apps
  - partitioning
  - data distribution
  - ghost generation and communication
  - adaptivity
  - collision detection
  - etc.
- Implemented in Charm++ (message driven), AMPI (message passing), and MSA (shared memory)

ParFUM Architecture



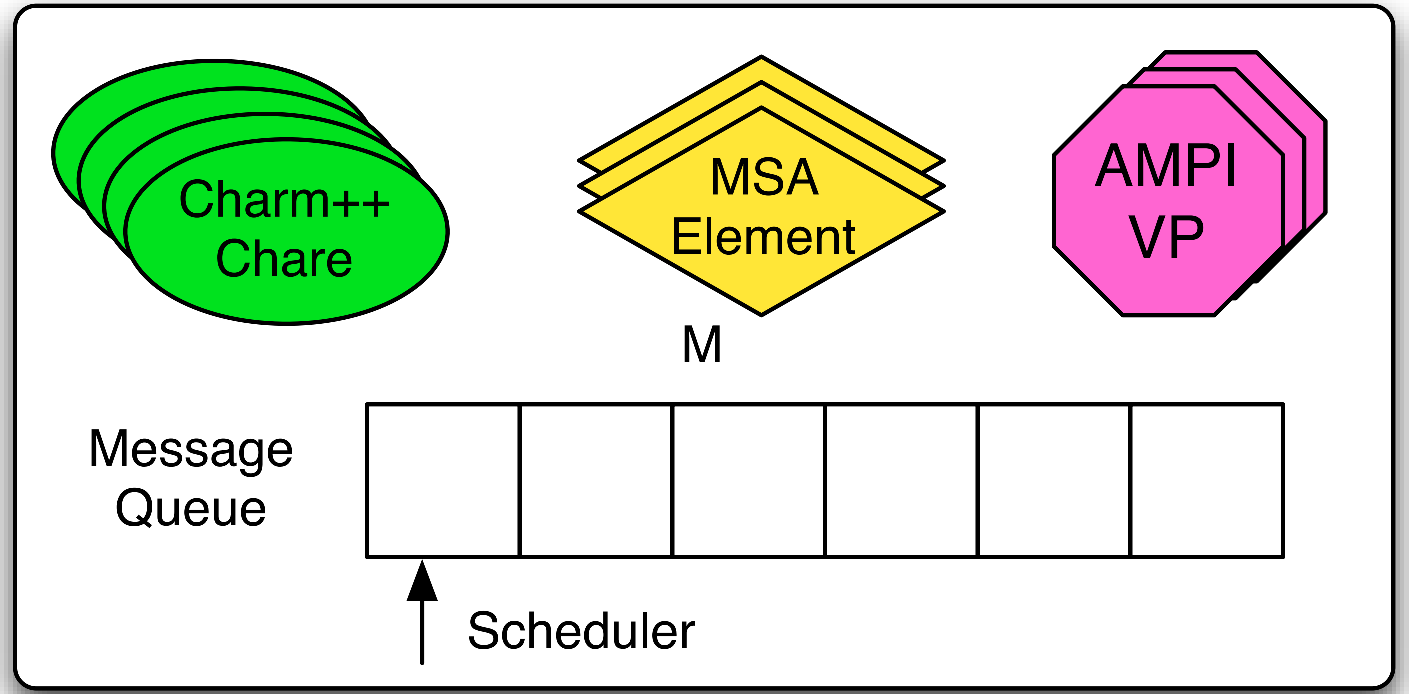
Charm RTS

- On each processor, there is a collection of parallel objects, each associated with a lightweight thread
- Incoming messages are placed in a queue
- A scheduler looks at the queue and chooses which object will run next



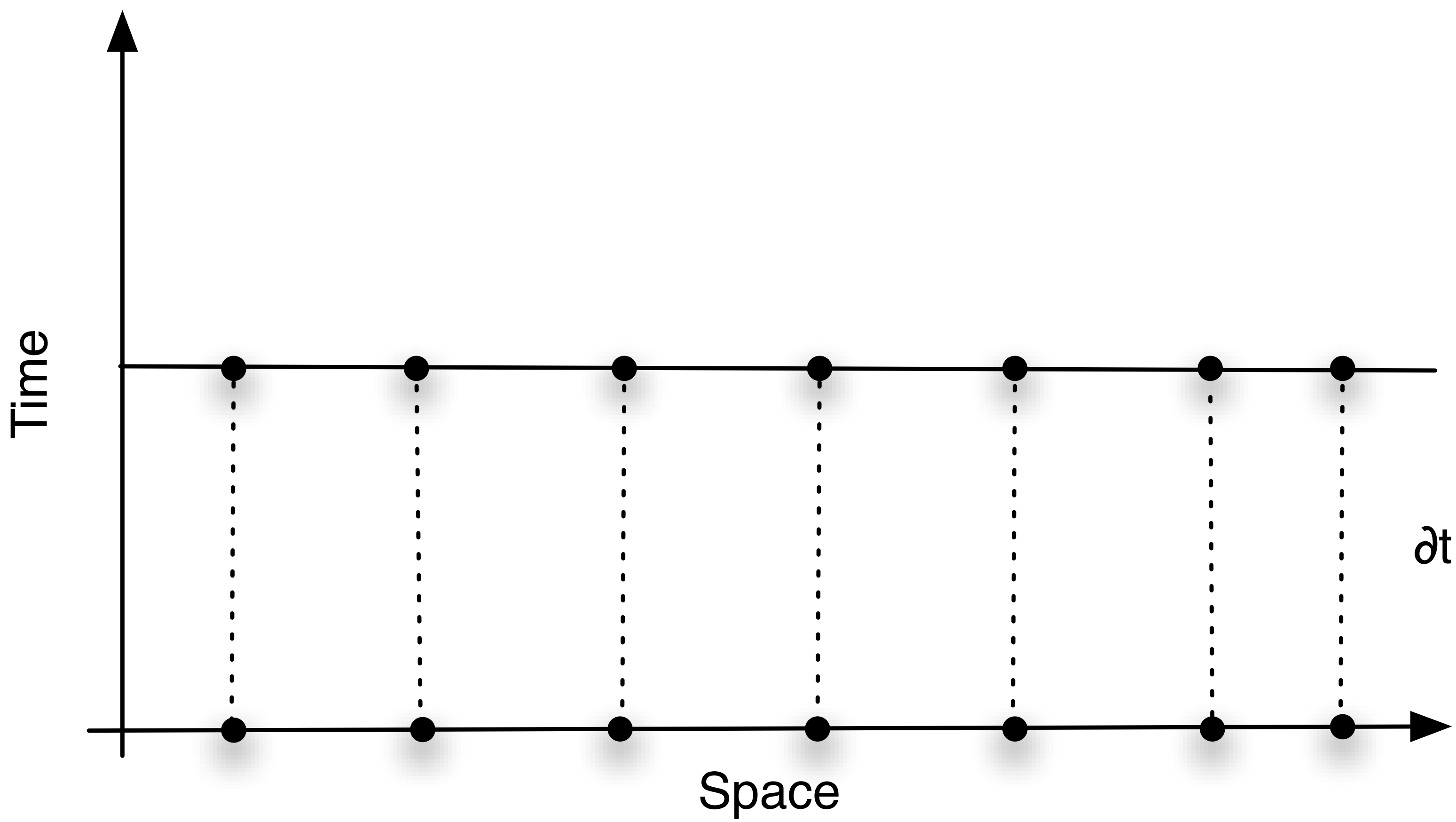
Charm RTS

- Virtualization: overdecomposition (many objects per processor)
  - overlap of communication and computation
  - control over working set size by varying level of decomposition
- Common resource management and instrumentation
- Load balancing based on object migration

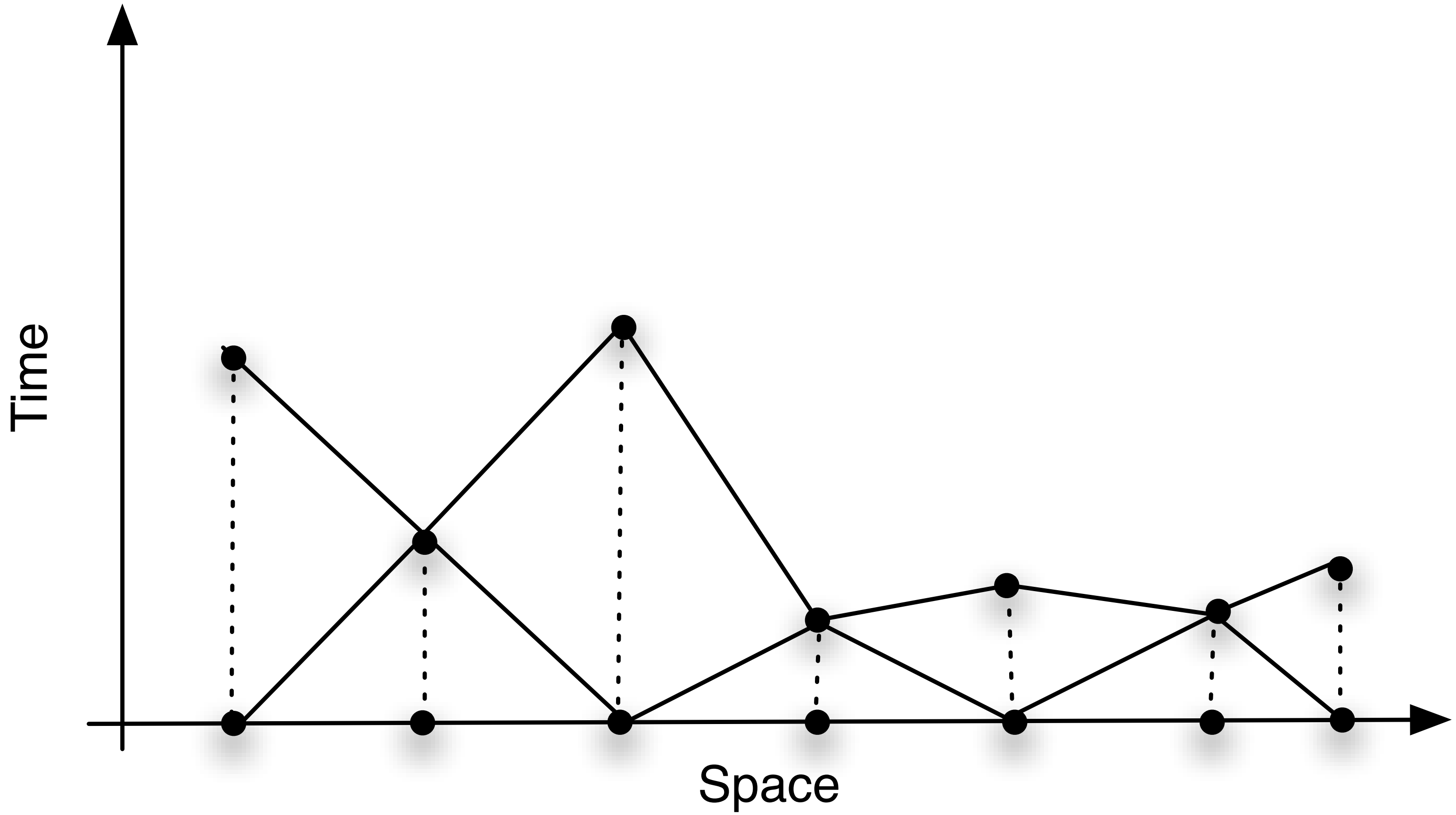


Example Application:
Spacetime Discontinuous Galerkin Mesh

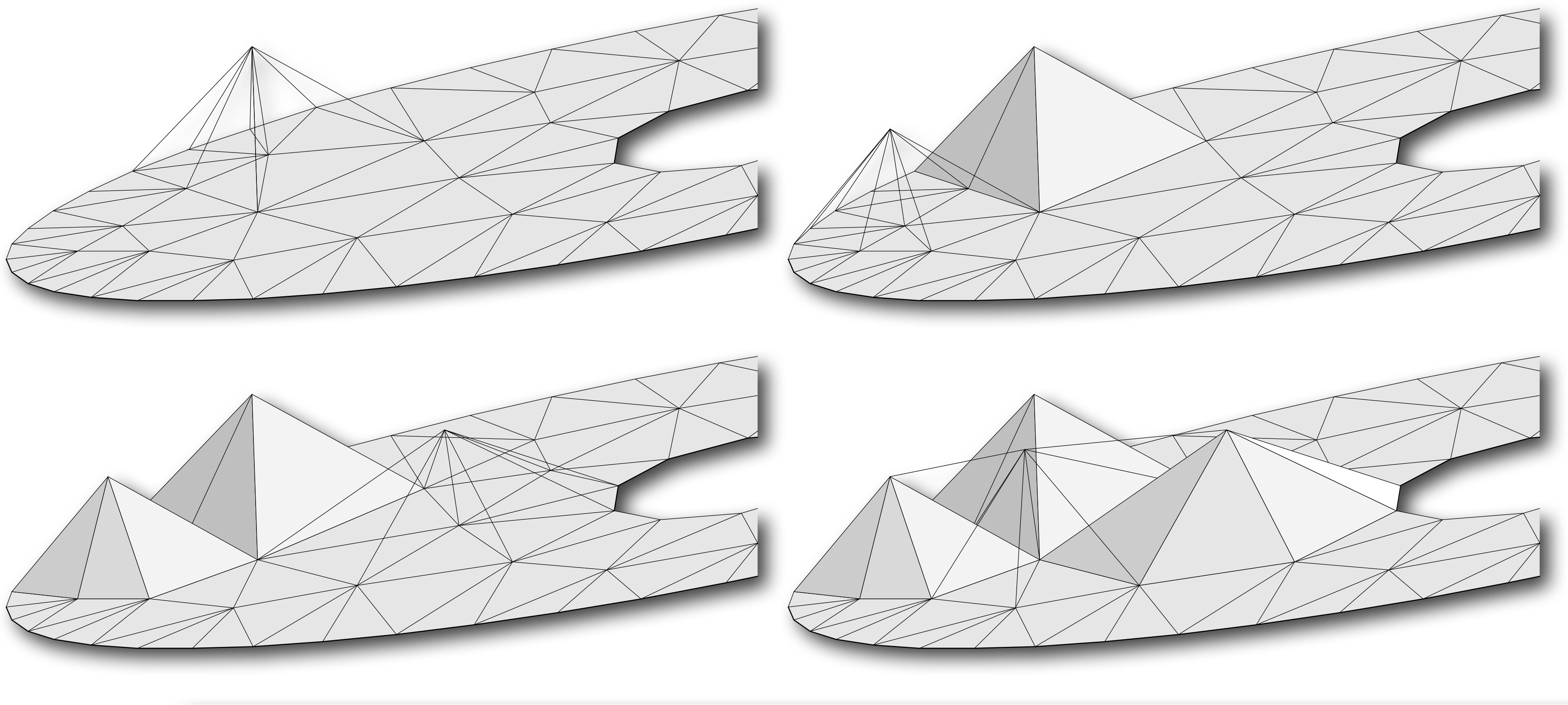
Typical 1D Finite Element Code

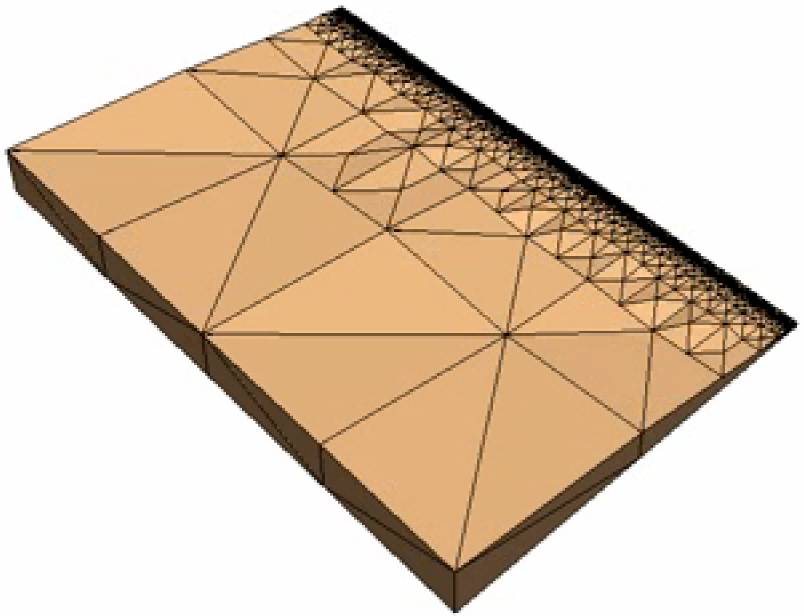


Spacetime Discontinuous Galerkin Code

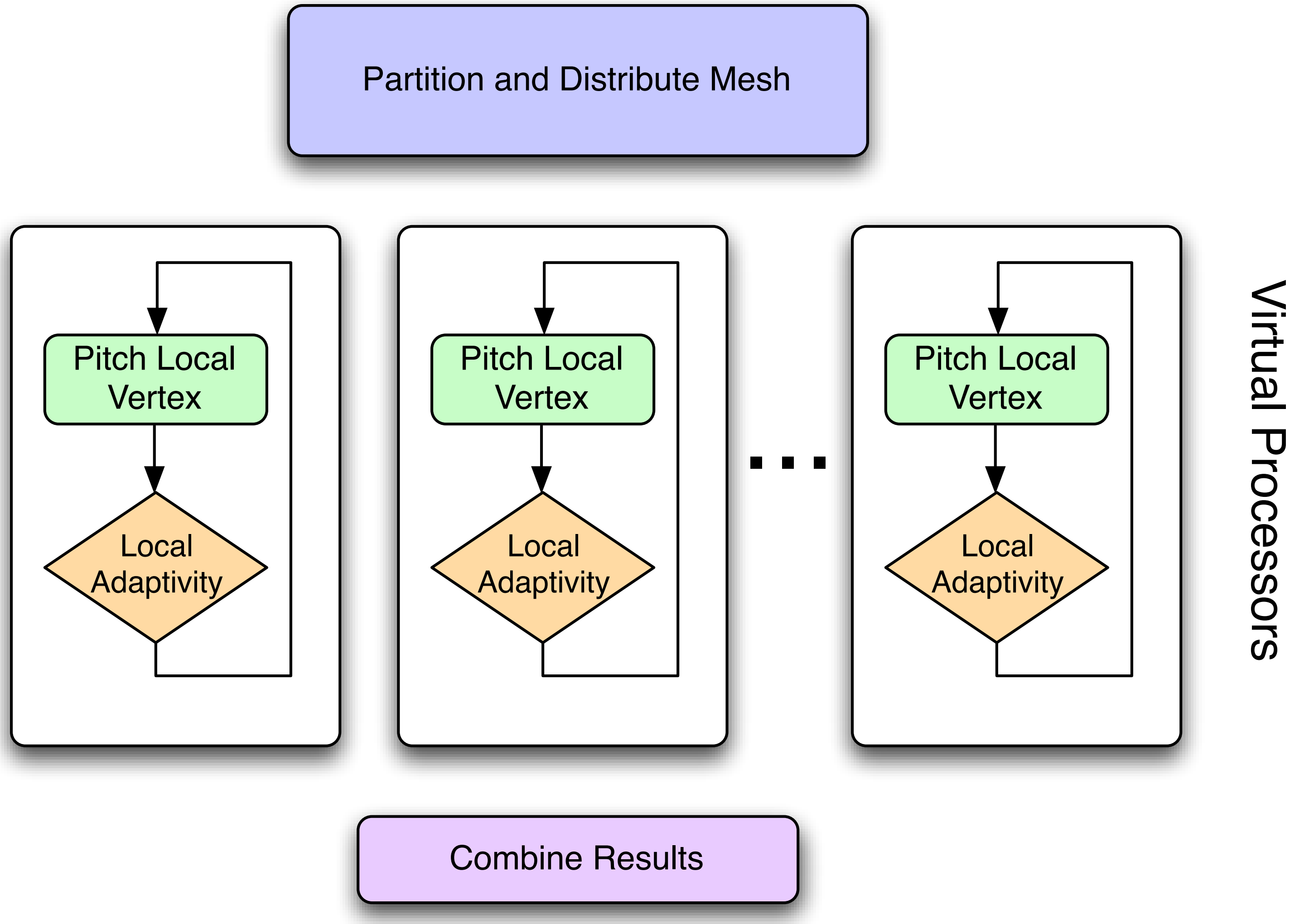


CPSD

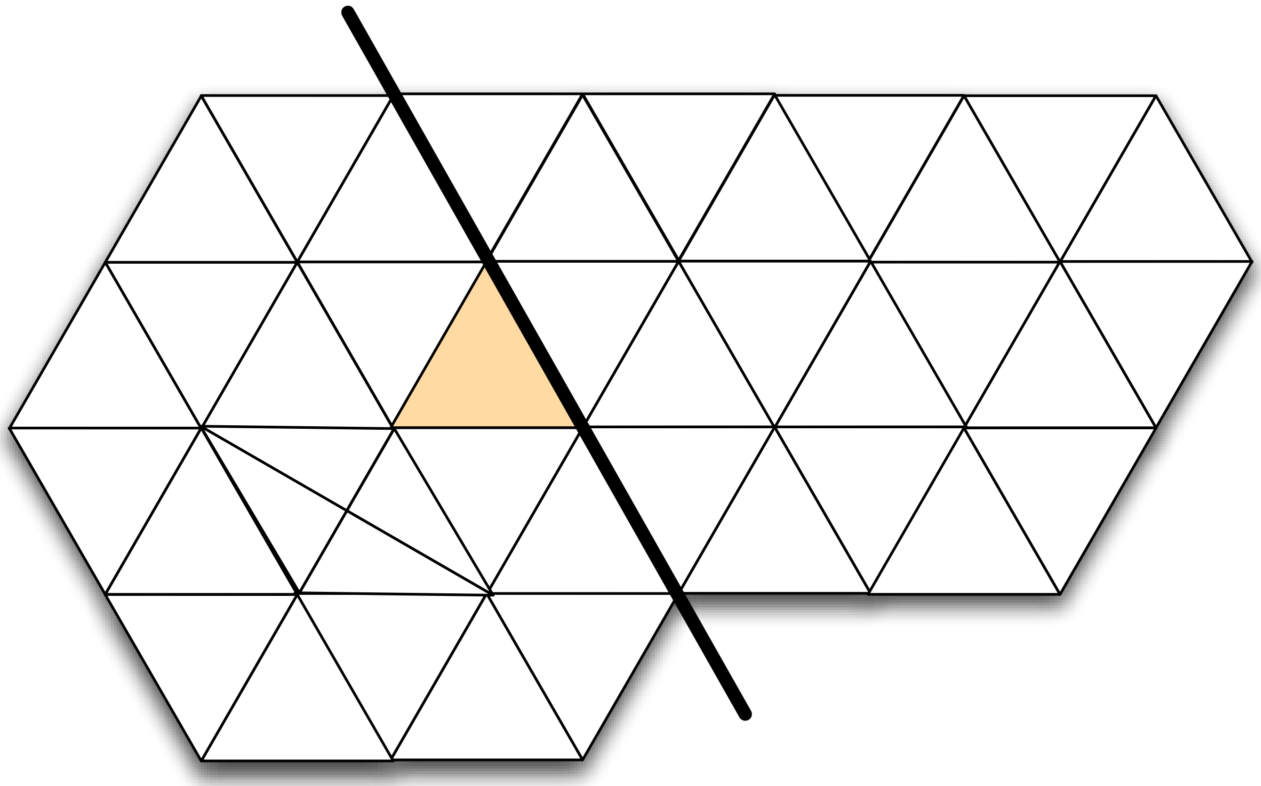




SDG Code Structure

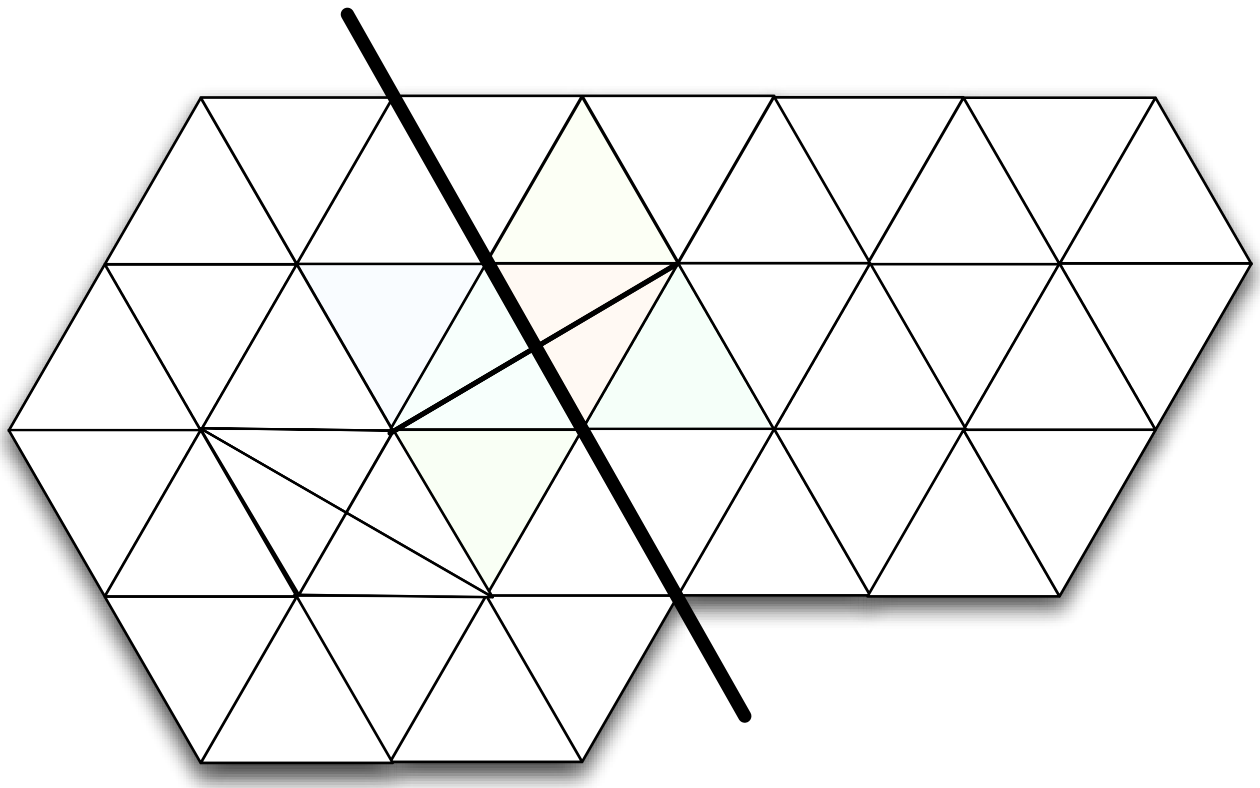


Incremental Adaptivity



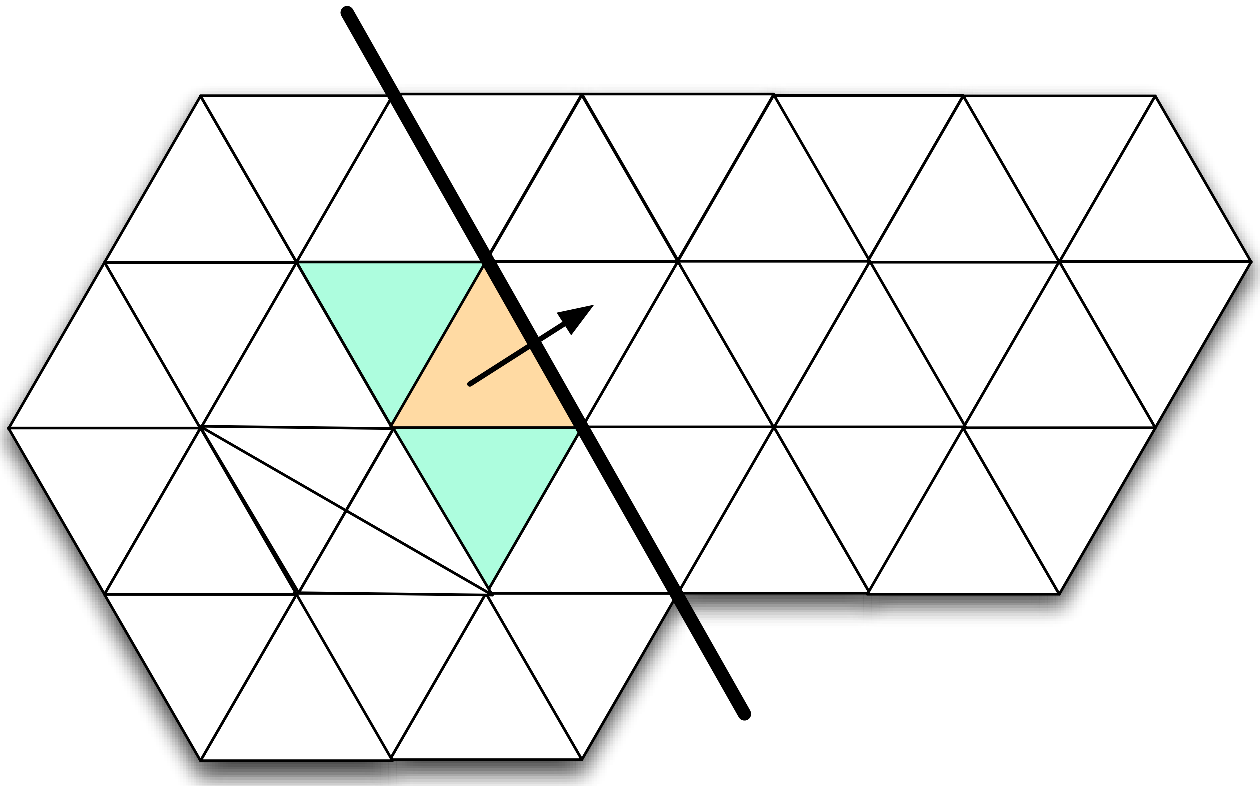
Example: edge bisection
on a processor boundary

Incremental Adaptivity



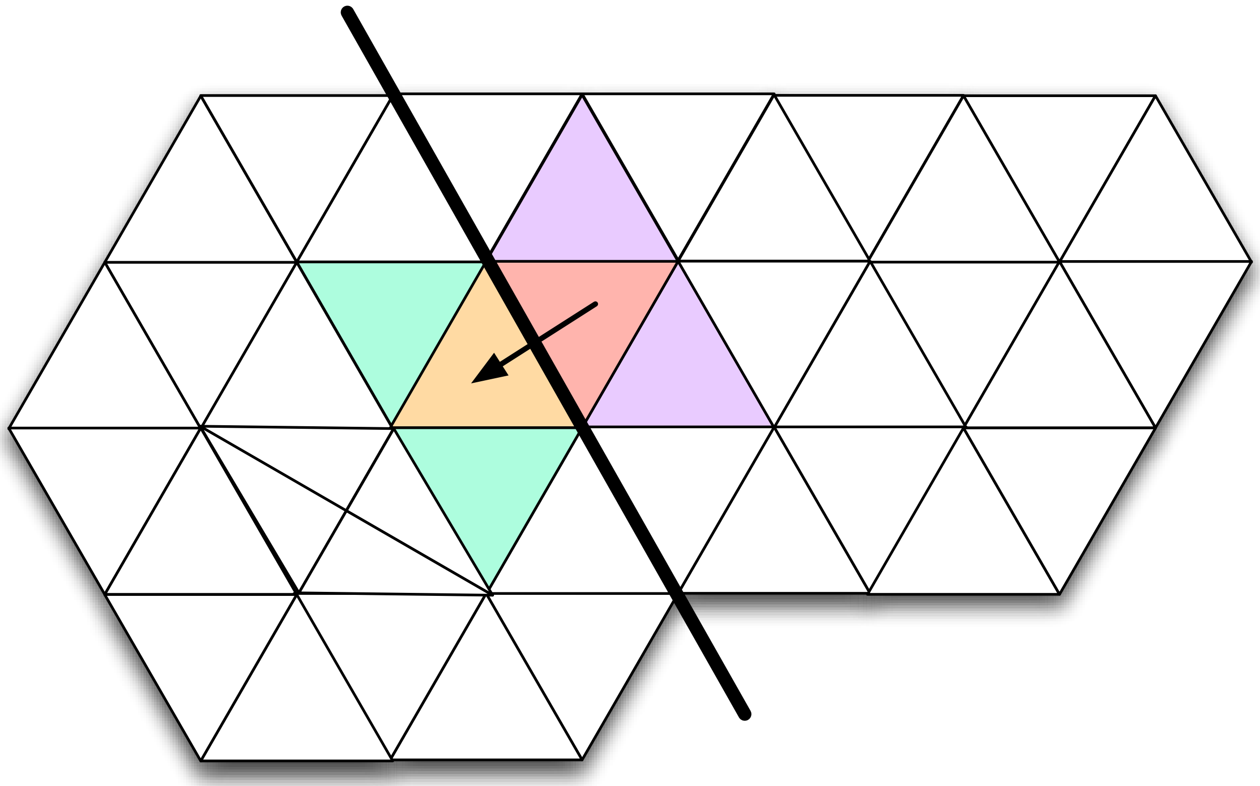
Goal State

Incremental Adaptivity



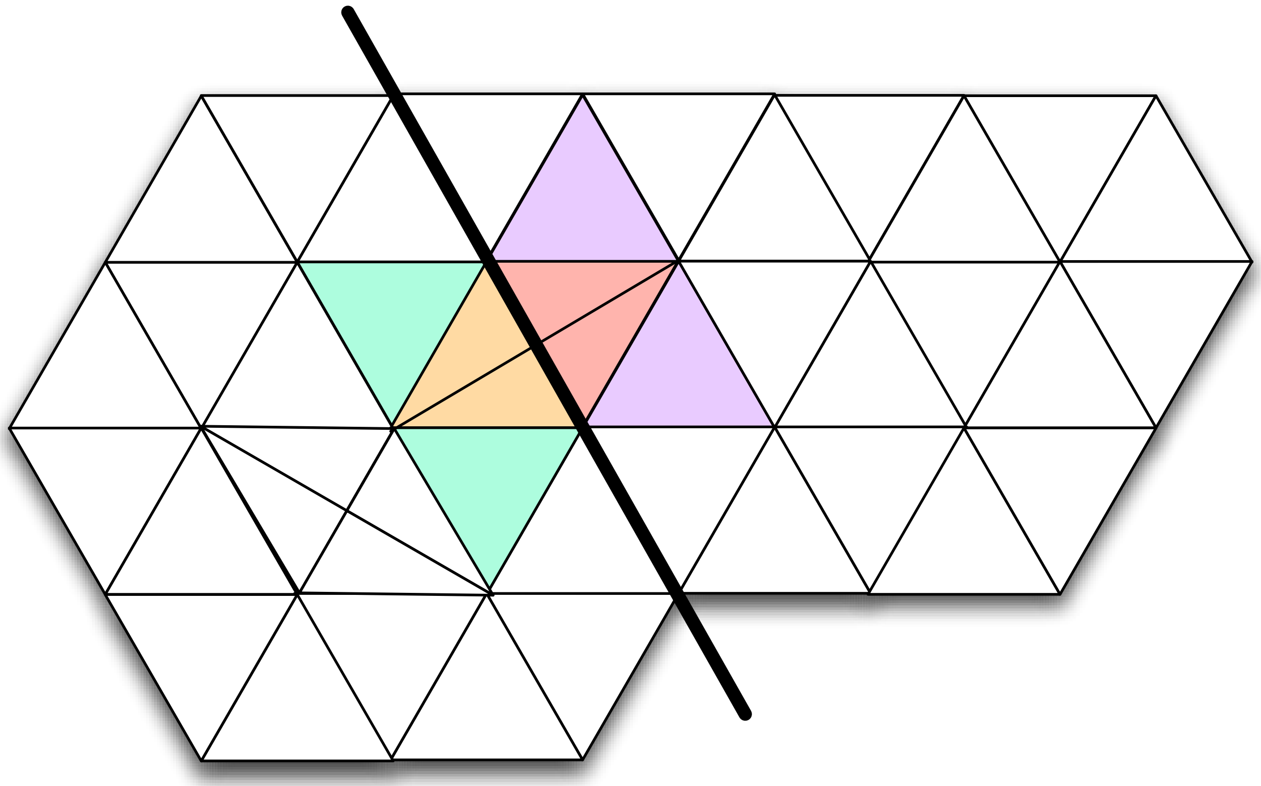
Lock local neighbors, request bisect from neighbor

Incremental Adaptivity



Receive request, lock local elements

Incremental Adaptivity



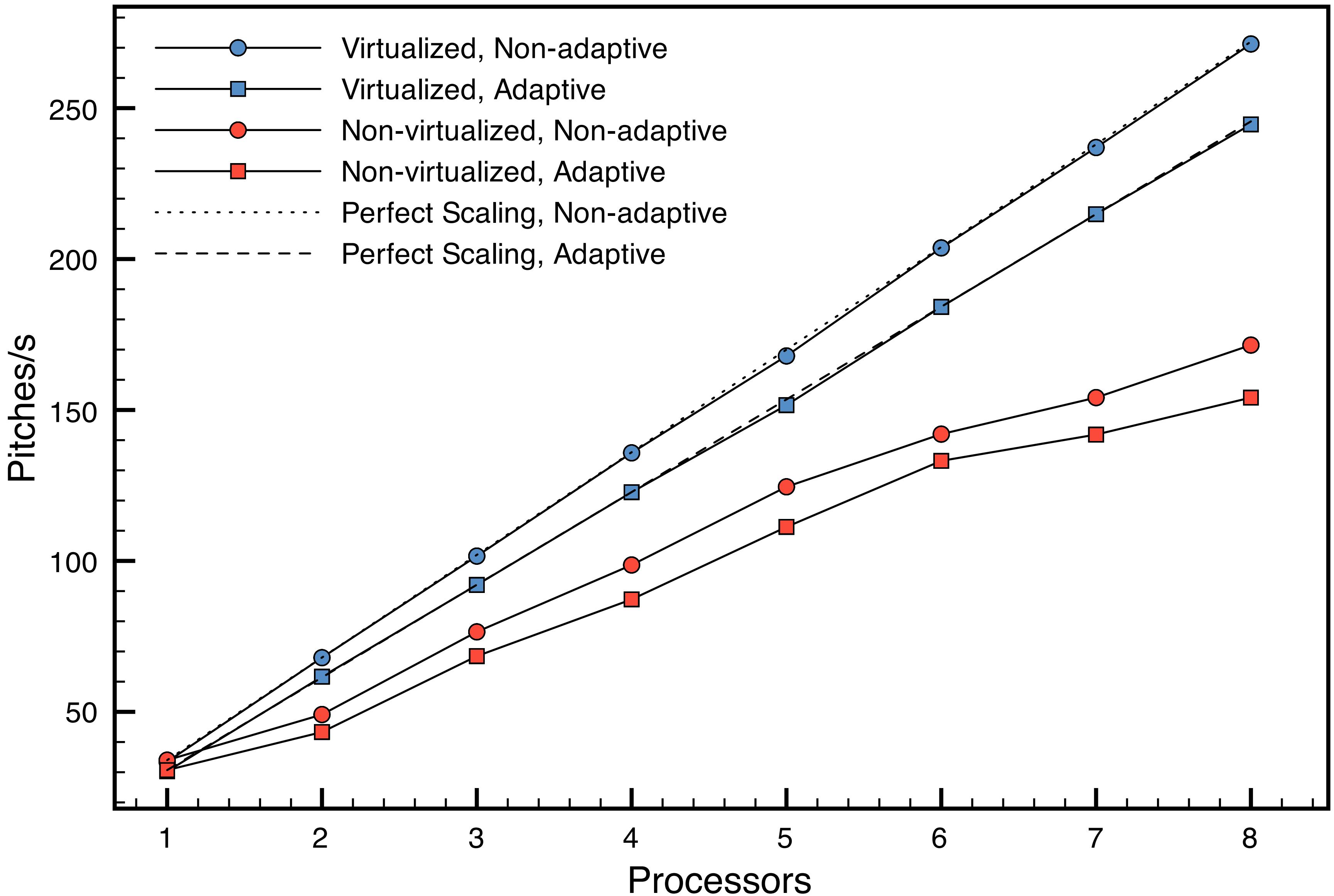
Example: edge bisection on a processor boundary

Performance

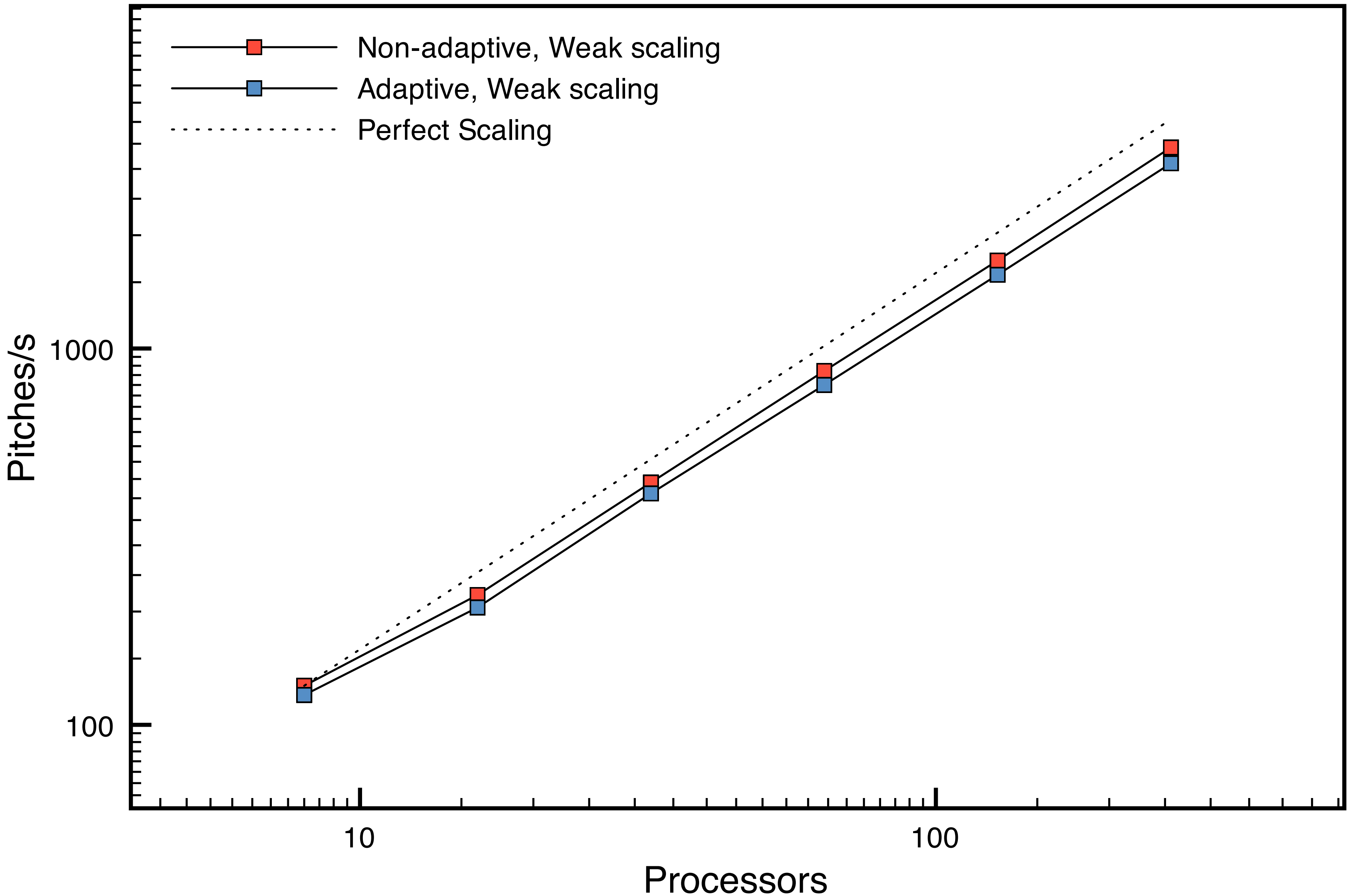
Benchmarking

- Unfortunately, existing benchmark suites do not lend themselves well to testing multiparadigm systems
  - too simple
  - often designed with one particular paradigm in mind
- What are good examples of very small, realistic benchmarks for which a multiparadigm approach makes sense?
- Since I don't have benchmarks, I will present some results from the SDG application

SDG Workstation Performance



SDG Cluster Performance



Summary

- Multiparadigm programs potentially offer advantages in terms of level of abstraction, compatibility, and reuse
- Modules written using different parallel models can be effectively combined

- Application performance in ParFUM has been good, but still need better multiparadigm benchmarking to identify and quantify overheads
- Number of models available when using Charm is still limited

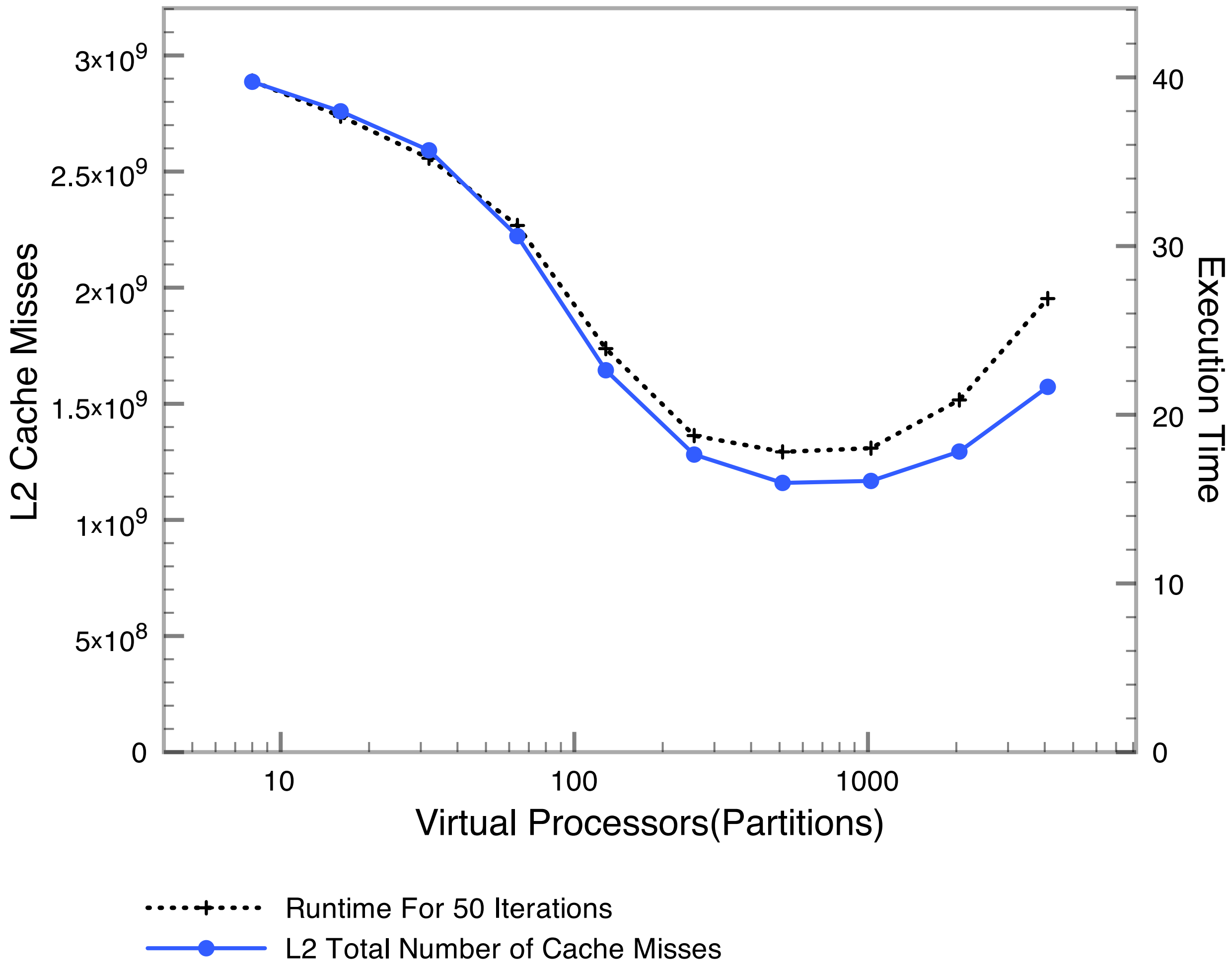
A Case Study in Tightly Coupled Multiparadigm Parallel Programming

Sayantan Chakravorty
Aaron Becker (abecker3@uiuc.edu)
Terry Wilmarth
Laxmikant V. Kalé

Parallel Programming Lab (charm.cs.uiuc.edu)
University of Illinois, Urbana-Champaign

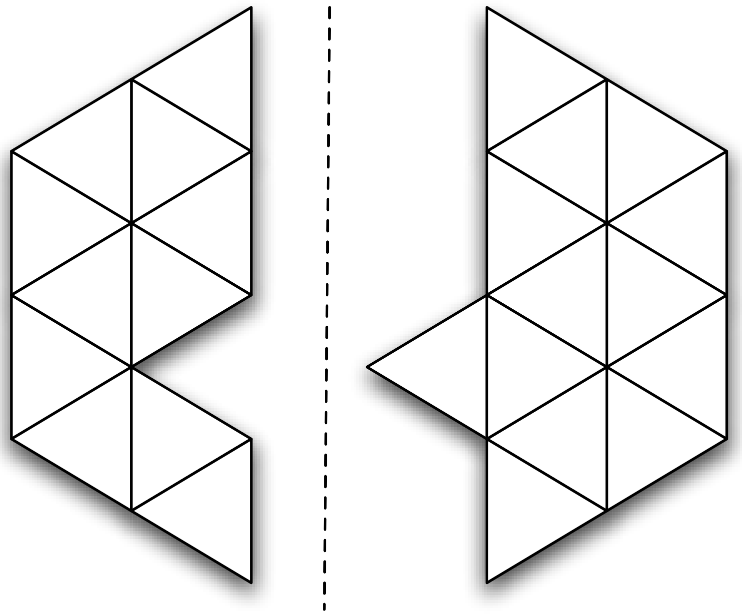
LCPC ‘08

Virtualization and Cache Effects

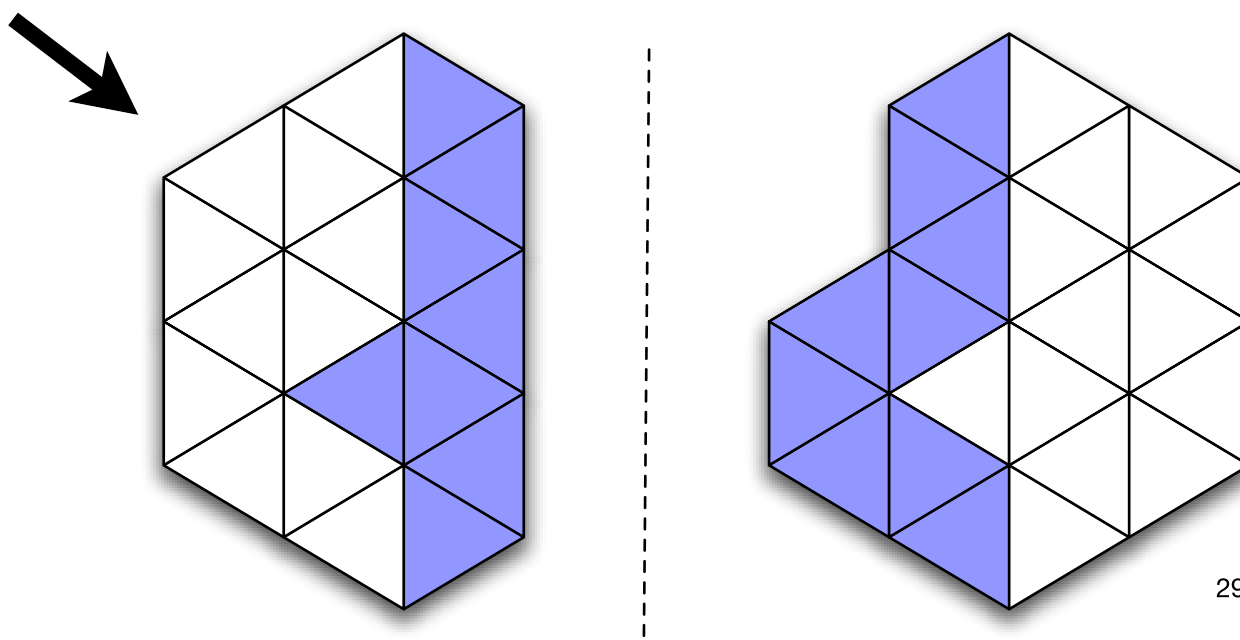


32 Processors, 1.2M Elements

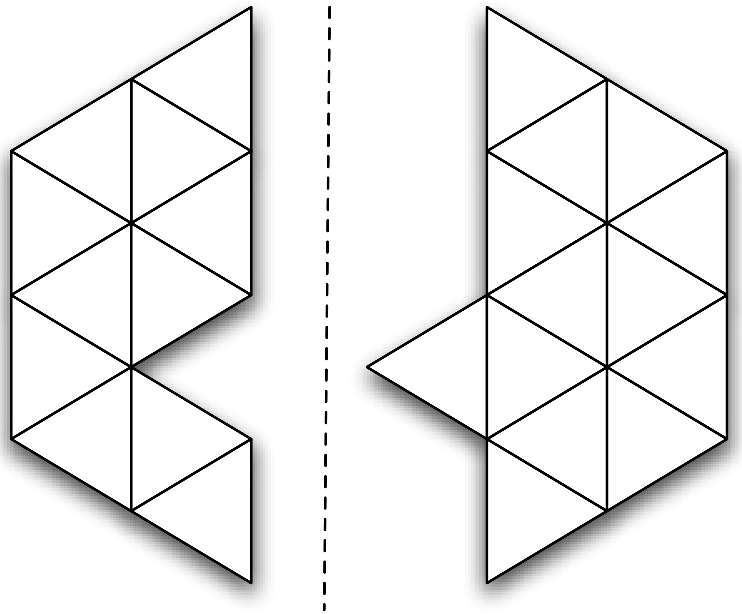
A data distribution problem



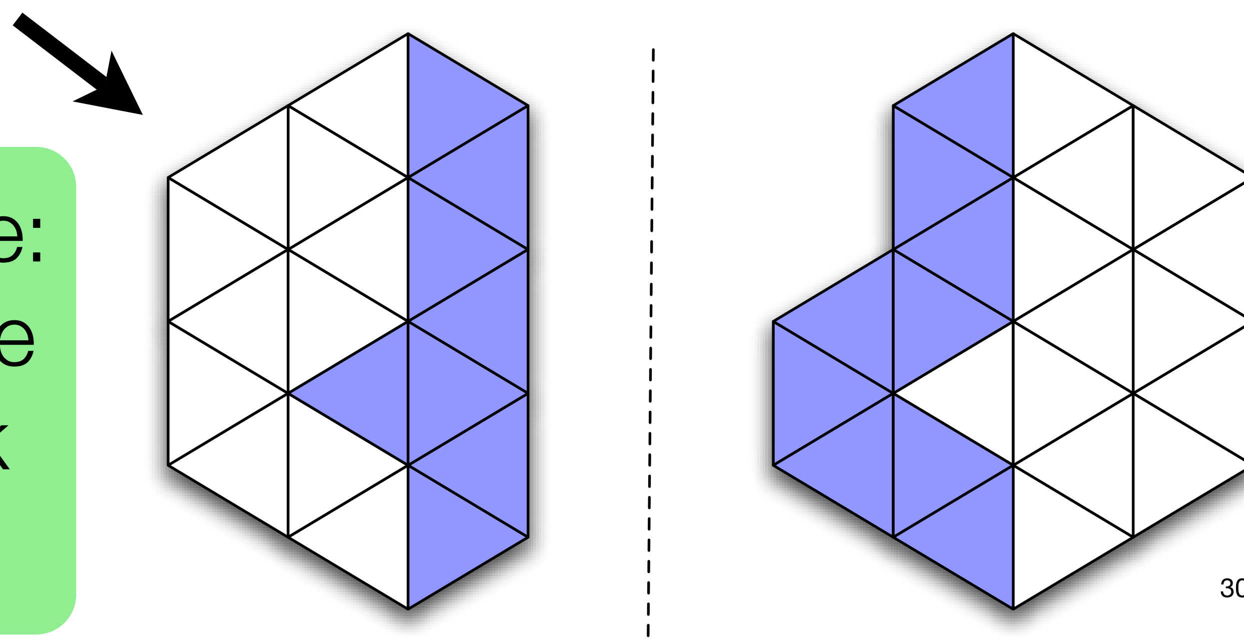
After initial partitioning, we need to determine which boundary elements must be exchanged.



A data distribution problem



After initial partitioning, we need to determine which boundary elements must be exchanged.



What we would like: an easily accessible global table to look up shared edges

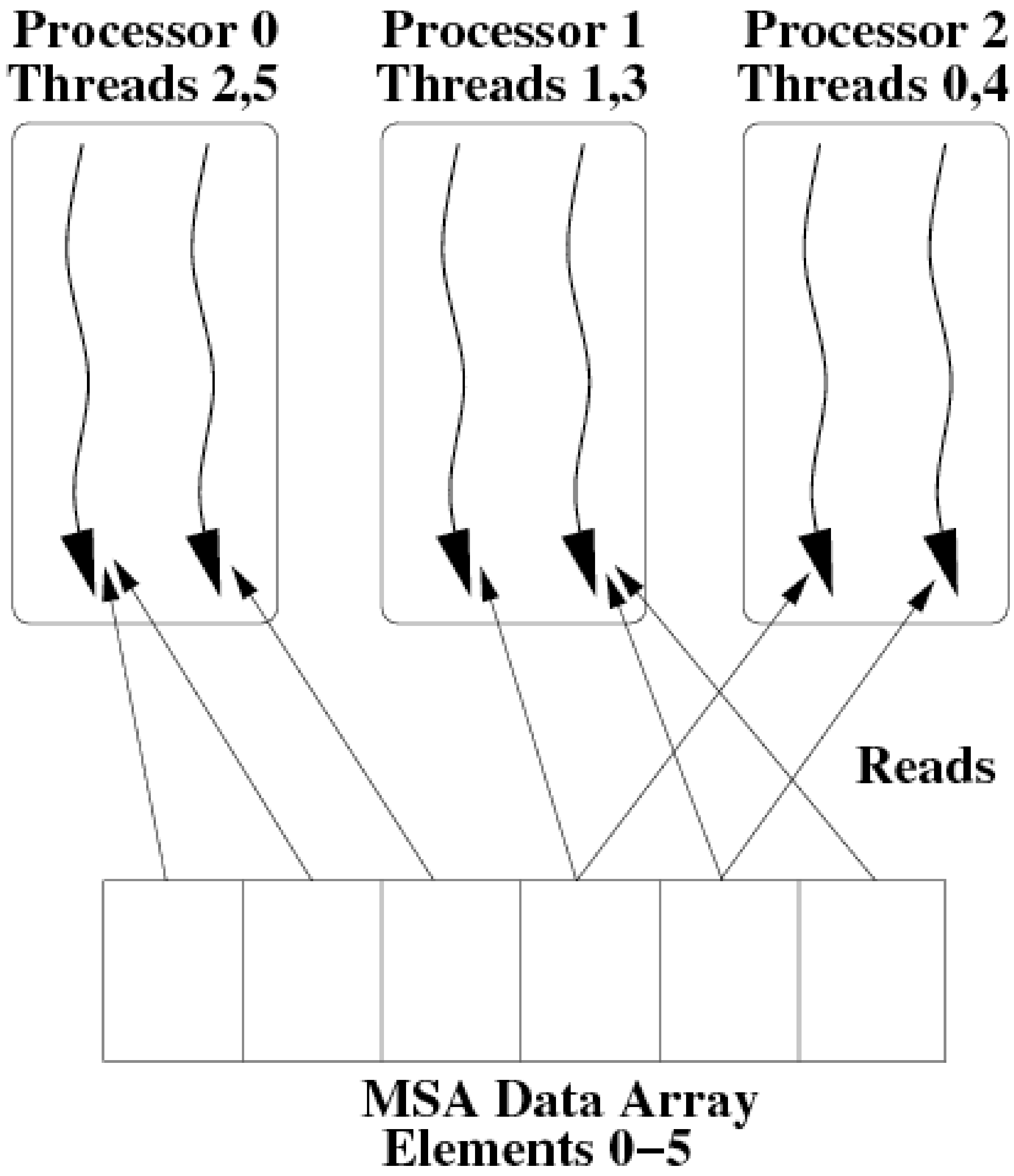
What is MSA?

Idea: shared arrays, where only one type of access is allowed at a time

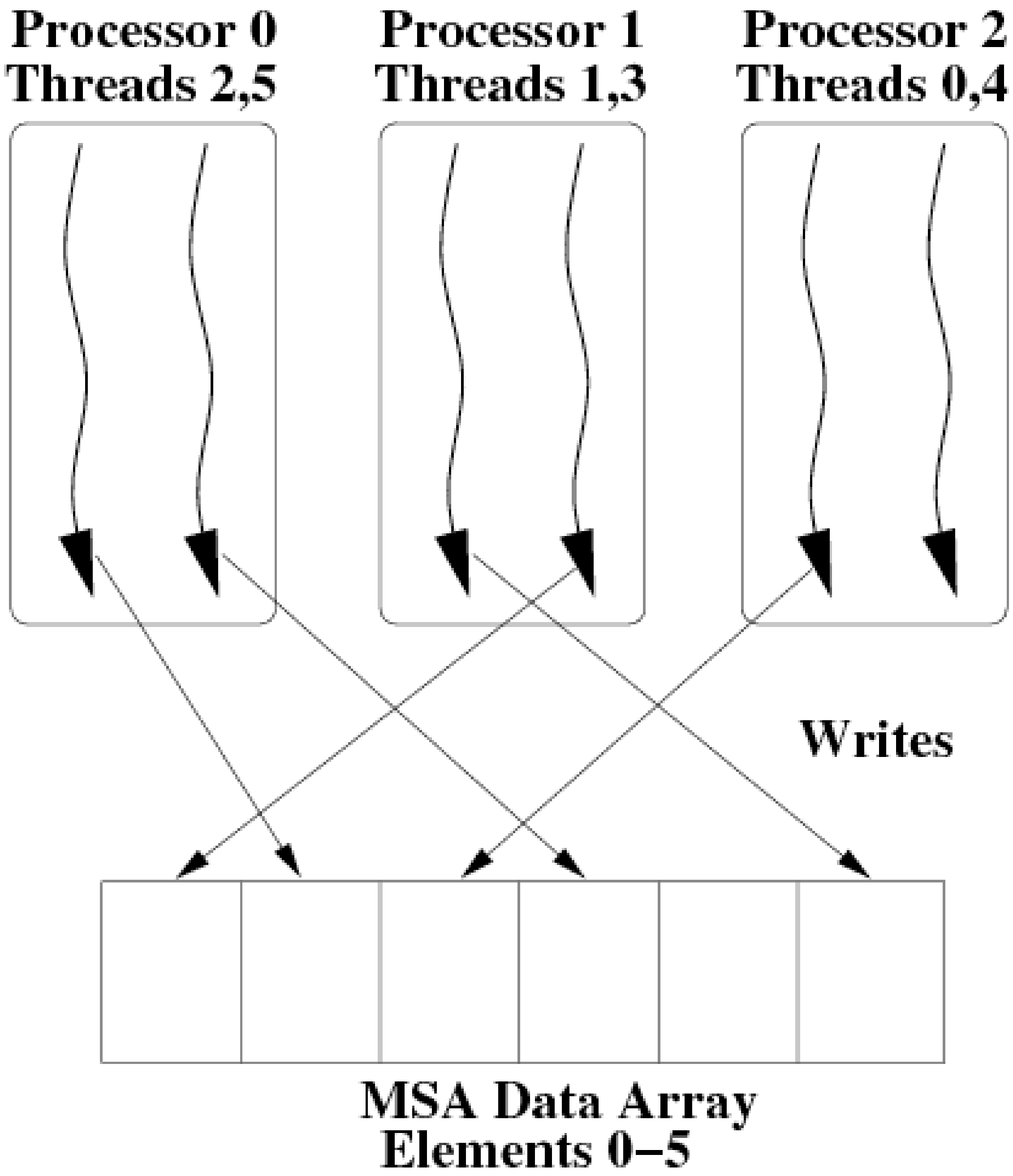
Access type is controlled by the array's *phase*

Phases include:

- read-only
- write-by-one
- accumulate

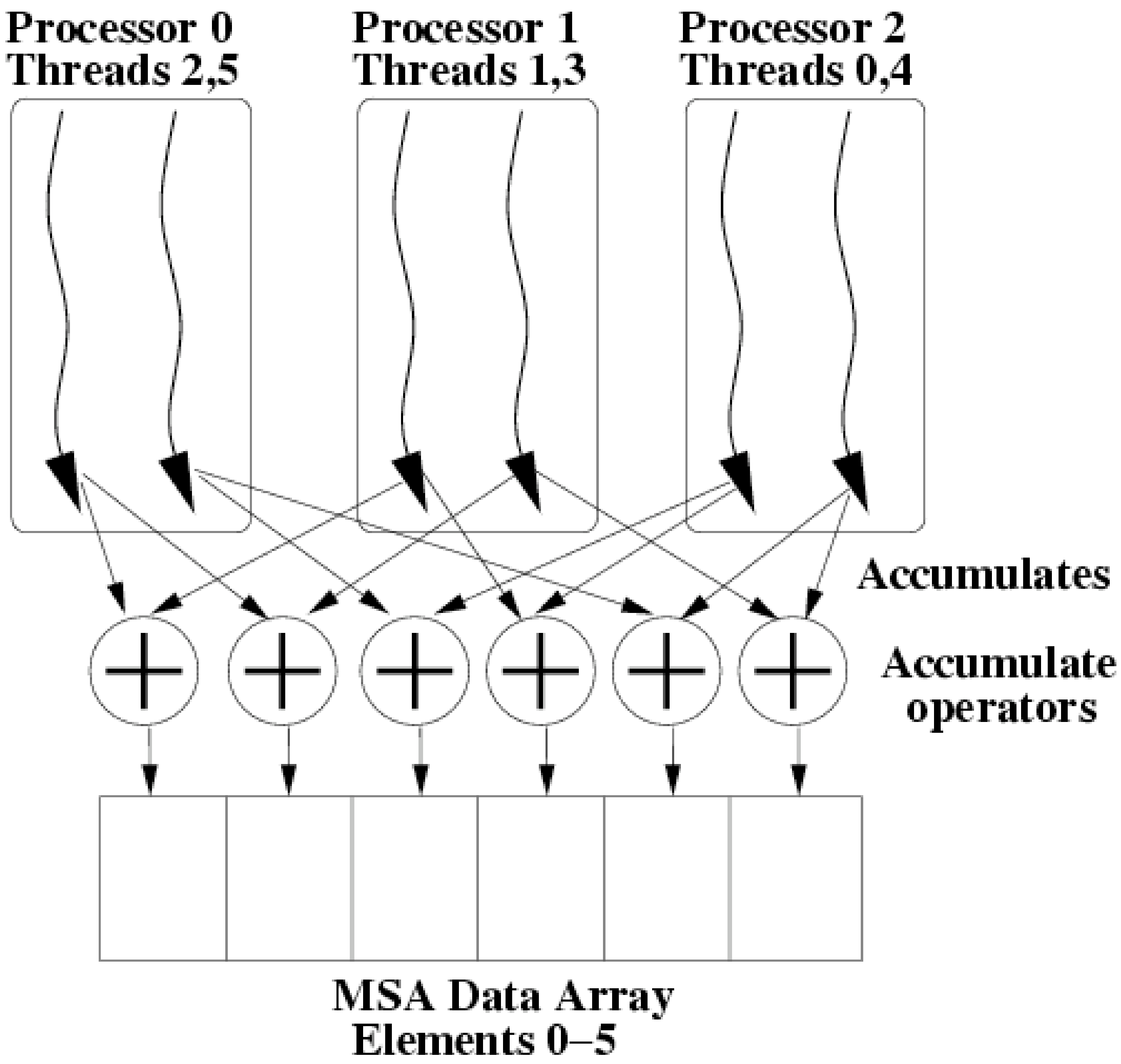


Read-only mode



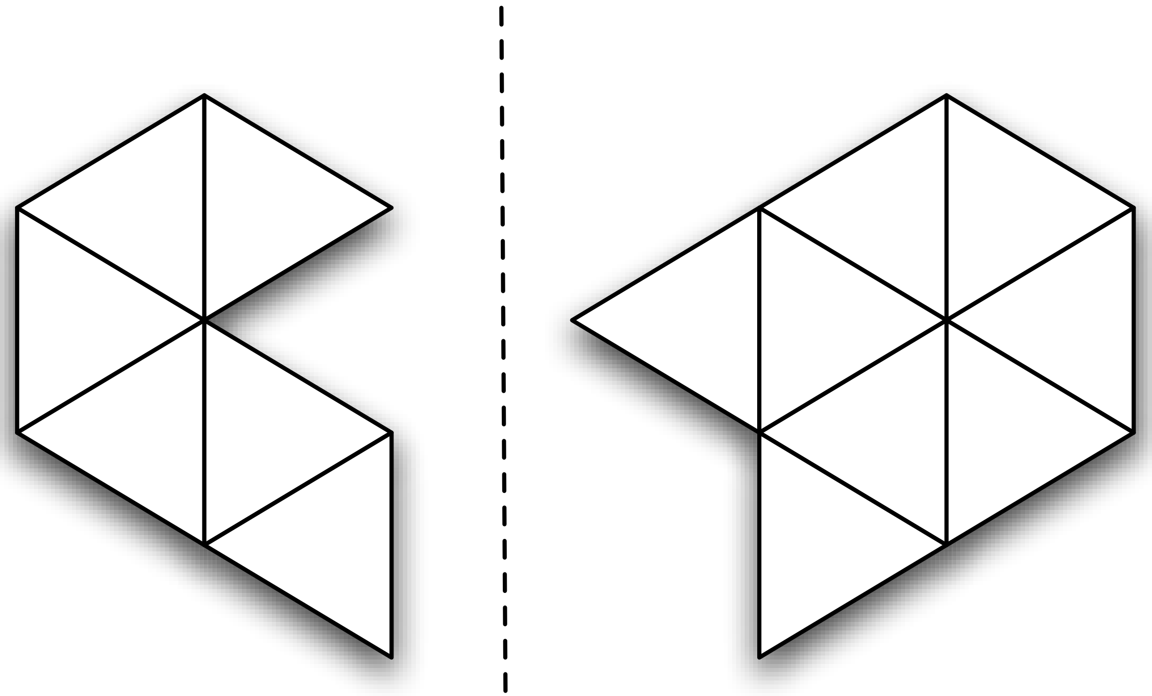
Write-by-one mode

note: one thread could write to many elements

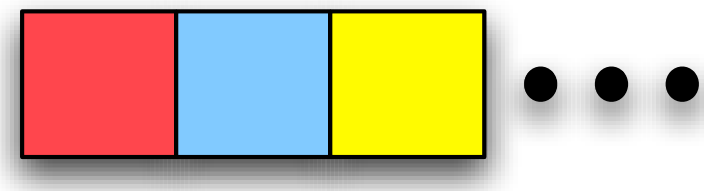


Accumulate mode

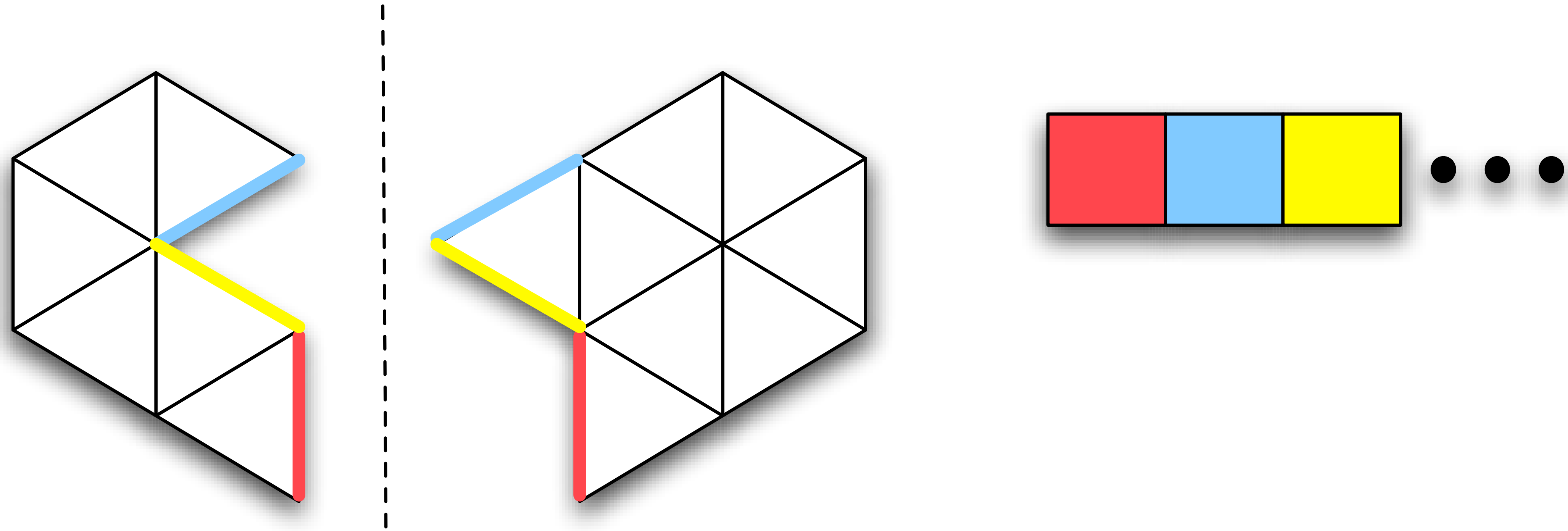
note: accumulation operator must be associative and commutative



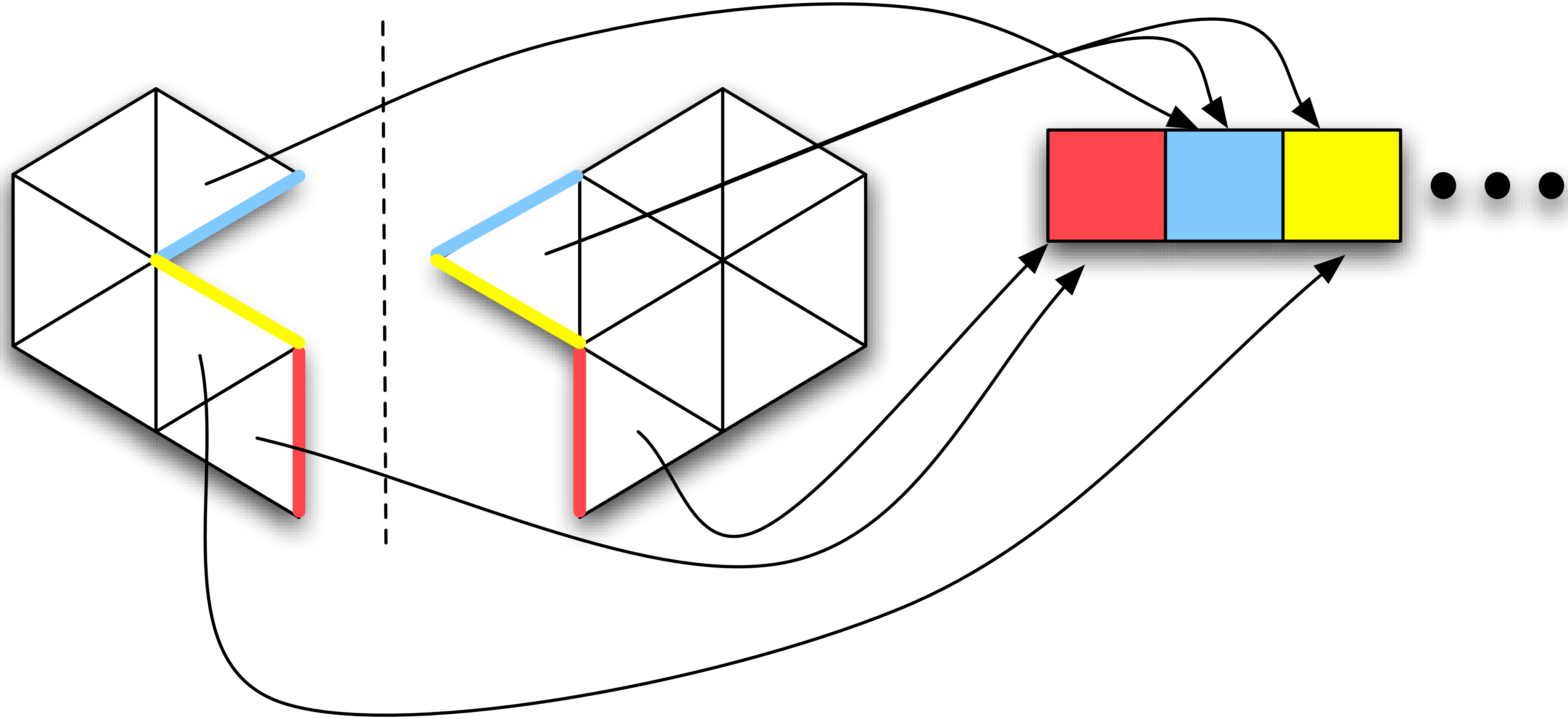
Partitioned Mesh



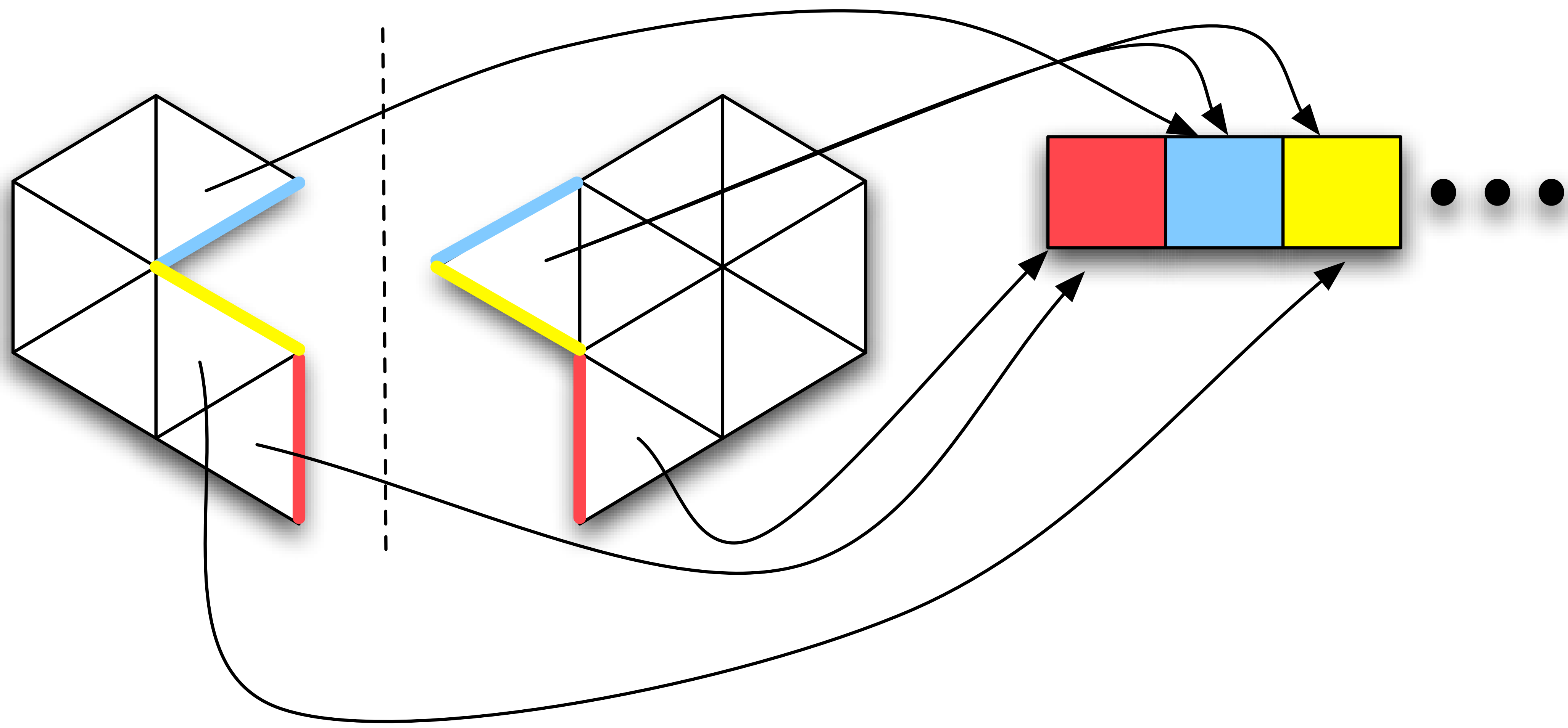
Distributed MSA
Hash Table



Each shared edge is hashed



Entries are added to the table in accumulate mode



Now elements which collide in the table probably share an edge