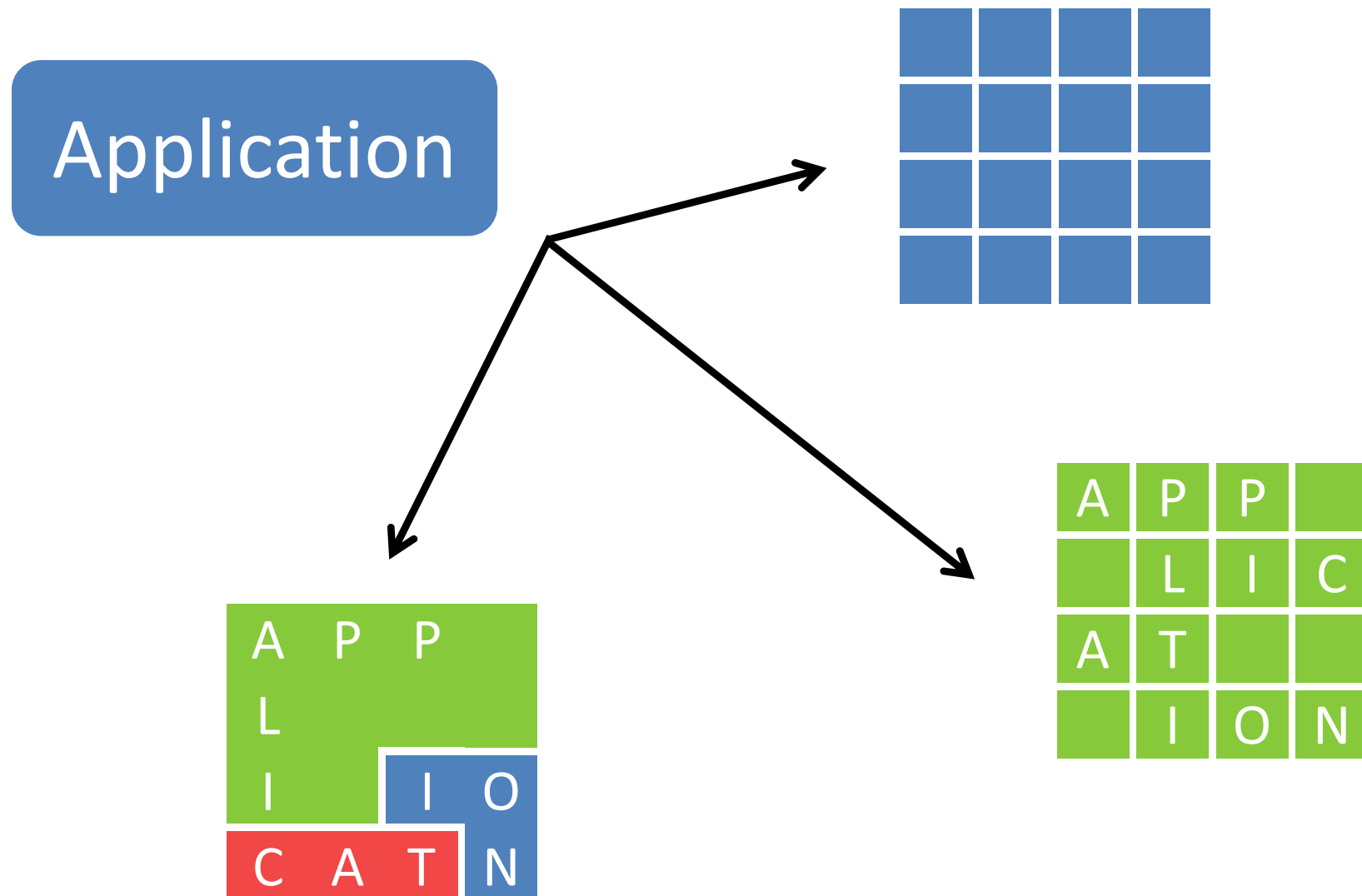


Measurement of Data Flow & Communication-Awareness

Peter Bertels, Wim Heirman and Dirk Stroobandt



Divide et impera: distributing the workload over multiple computational elements



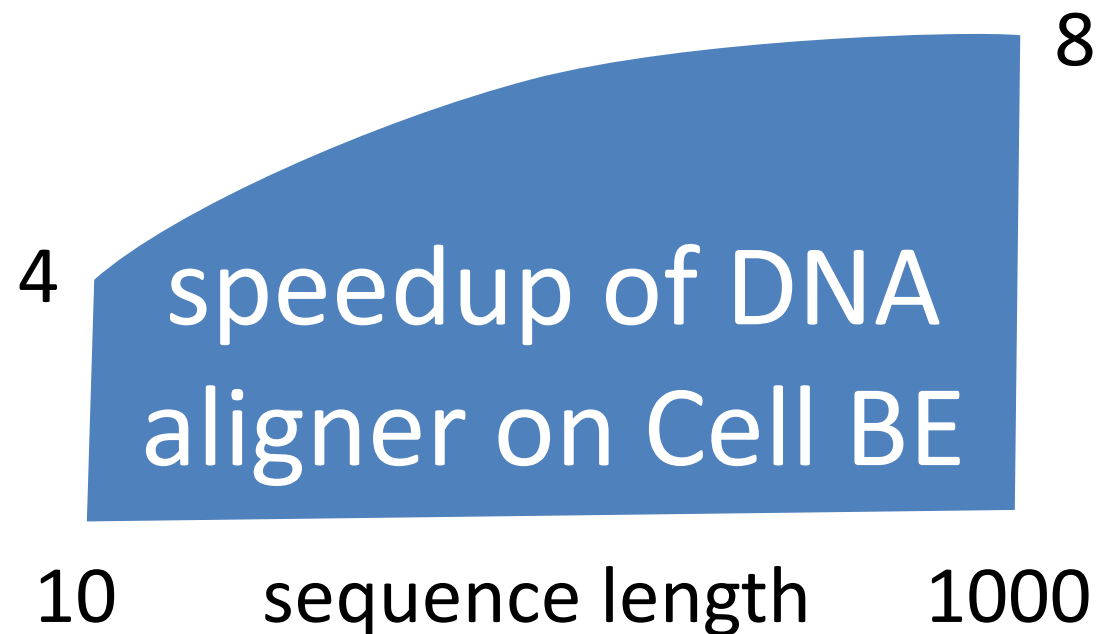
Distributing the workload **may**
leads ~~/~~ to faster program execution



only software

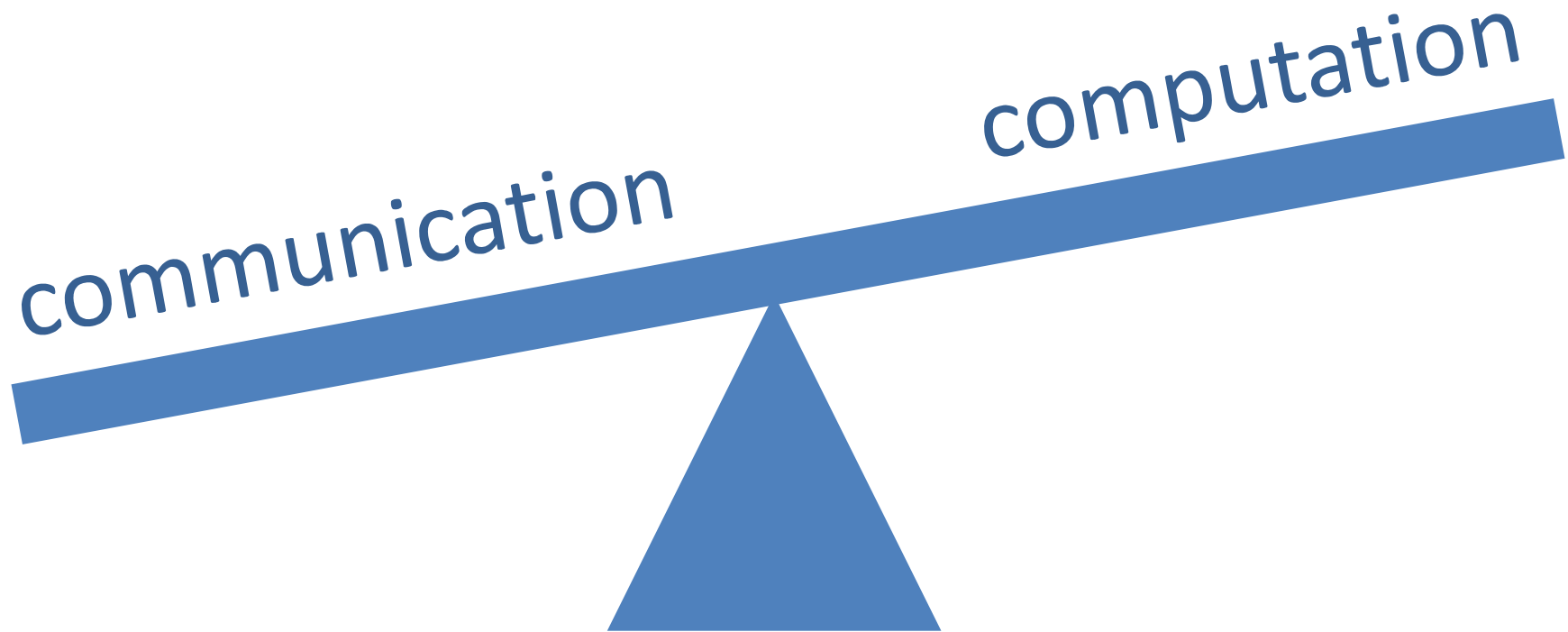


hw/sw codesign

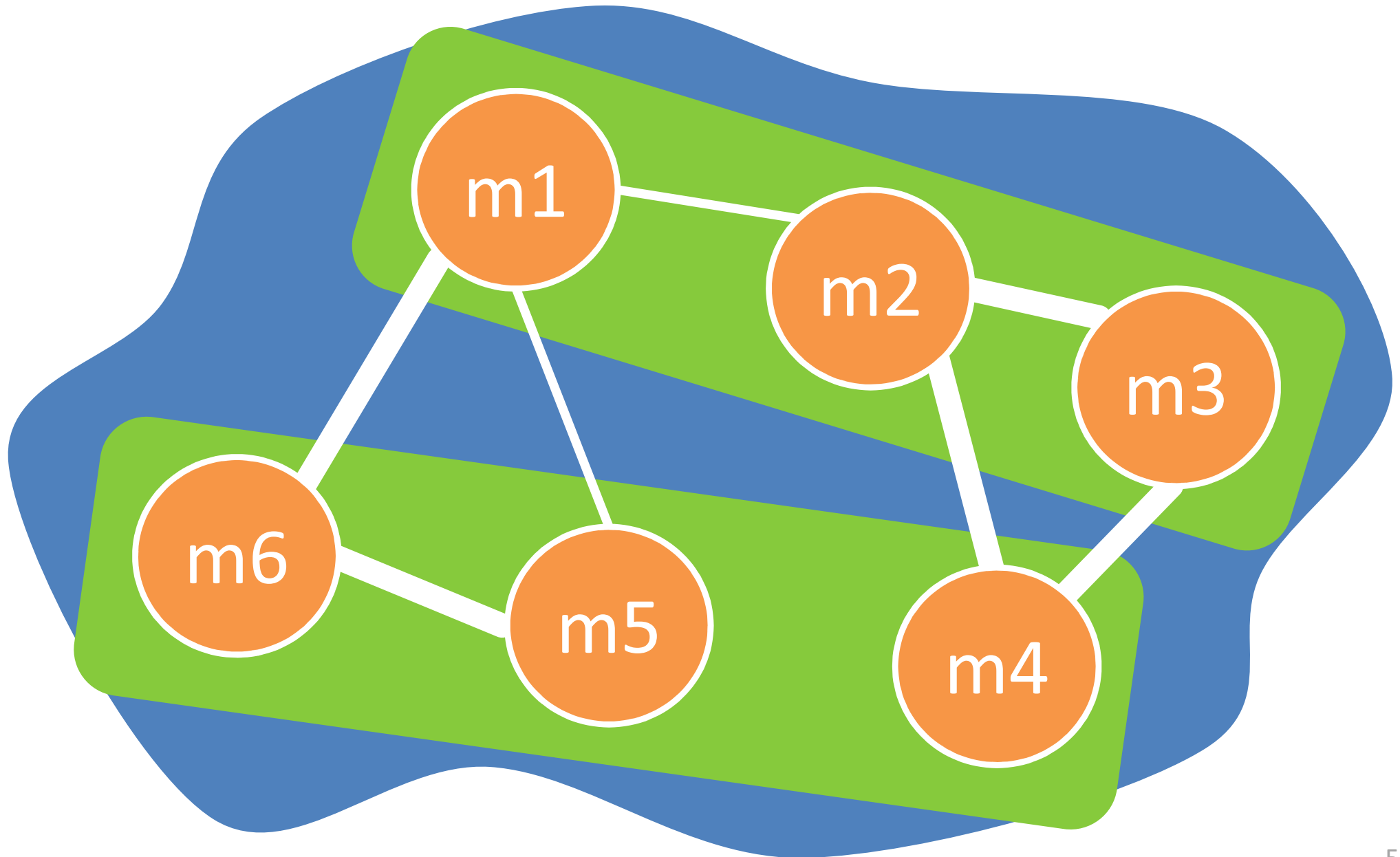


Communication often limits performance of distributed applications

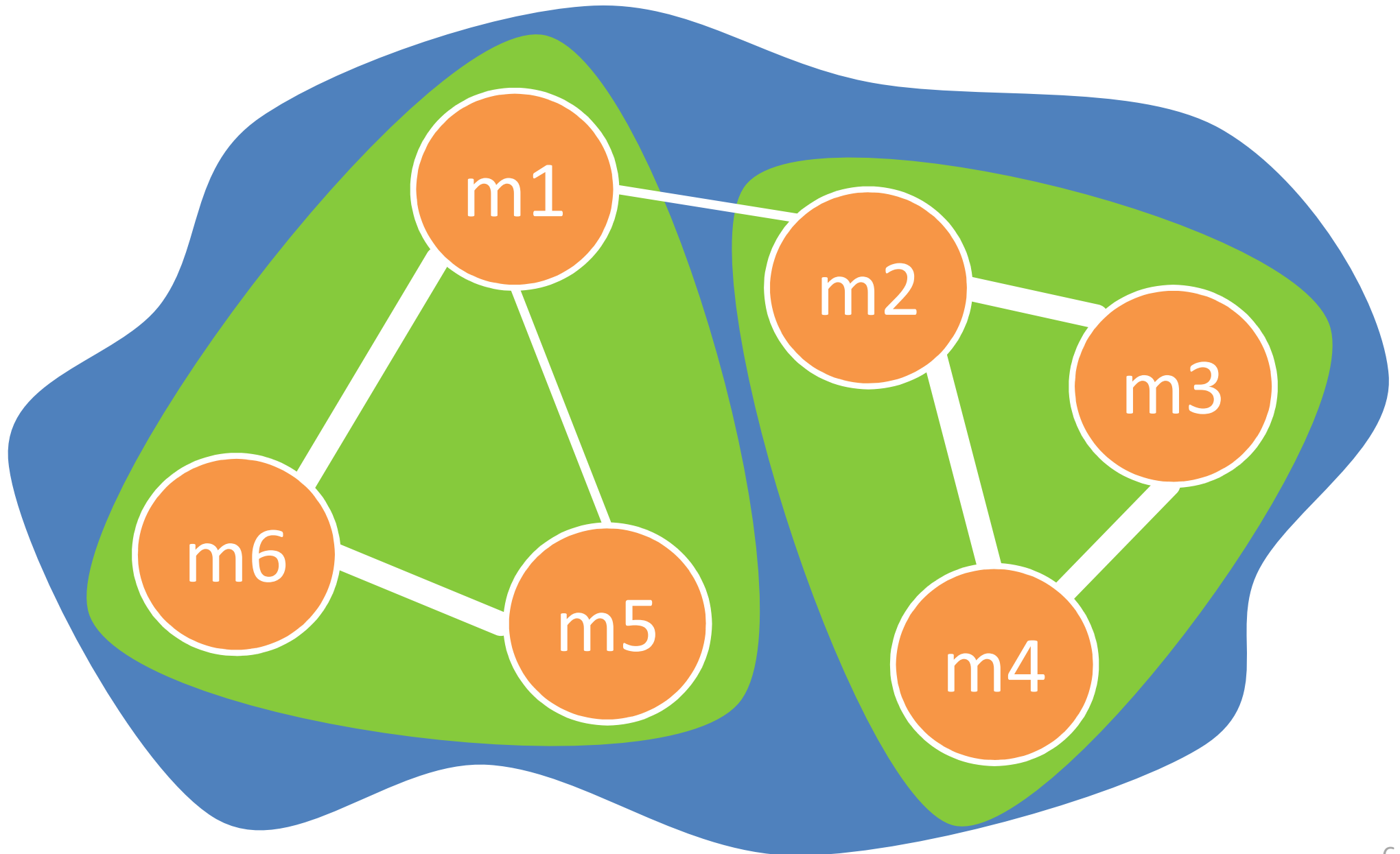
- Getting the data to the processor
- Accessing remote memory may be costly



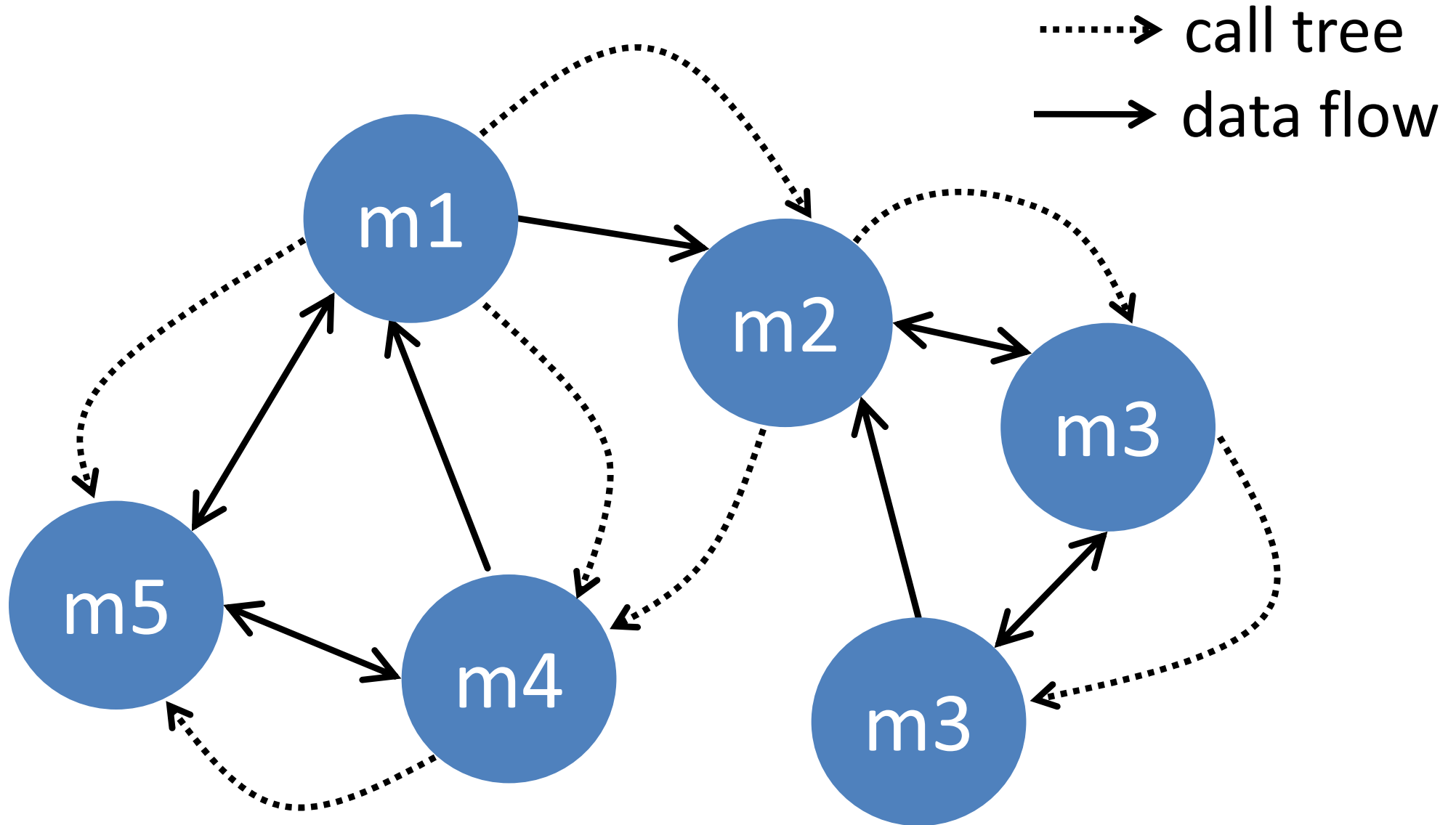
Data flow in an application becomes external communication after partitioning



Data flow in an application becomes external communication after partitioning



Communication graph combines a call tree with data flow information

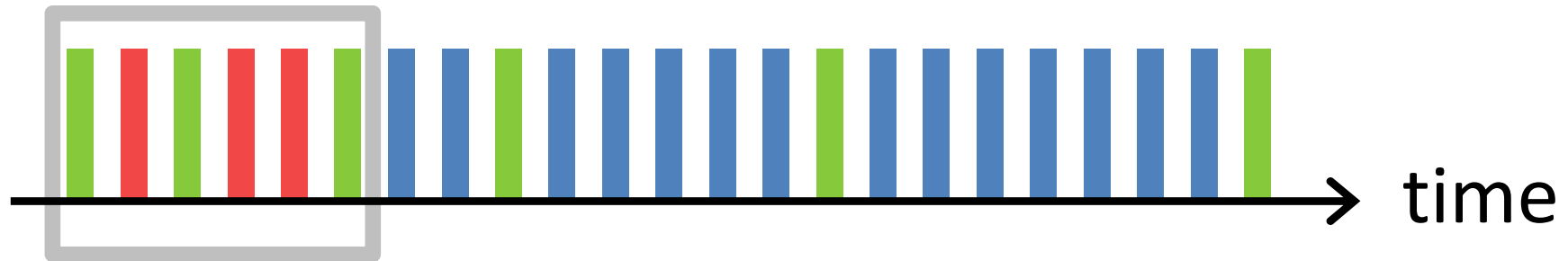


Building the communication graph is very slow and consumes a lot of memory

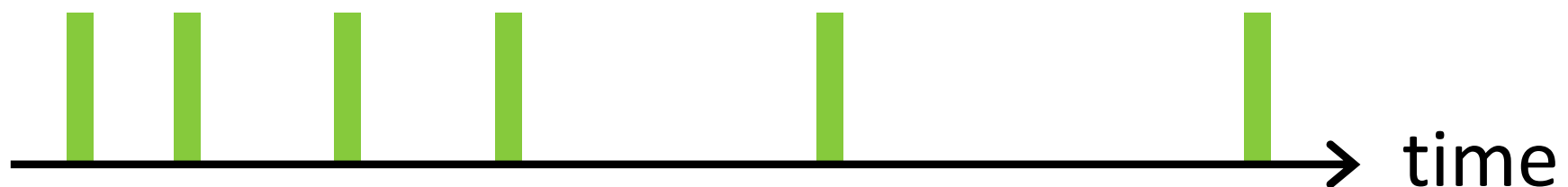
- Average over 6 Java benchmarks:
 - # memory accesses: 877 million
 - # method invocations: 155 million
- Time consuming:
 - Every load/store has to be instrumented
- Memory consuming:
 - Every Java object has a shadow object
 - Shadow objects keep track over producers of the data

Reservoir sampling enables faster measurement of the communication graph

- Estimating data flow in communication graph
- Reservoir sampling fills a reservoir of N samples with a random selection of all producer-consumer pairs



reservoir: N samples



Communication graph is built much faster but still very accurate

- Profiling overhead is reduced by x 15
- Relative error remains within predefined boundaries
- Reservoir size N is based on statistics

