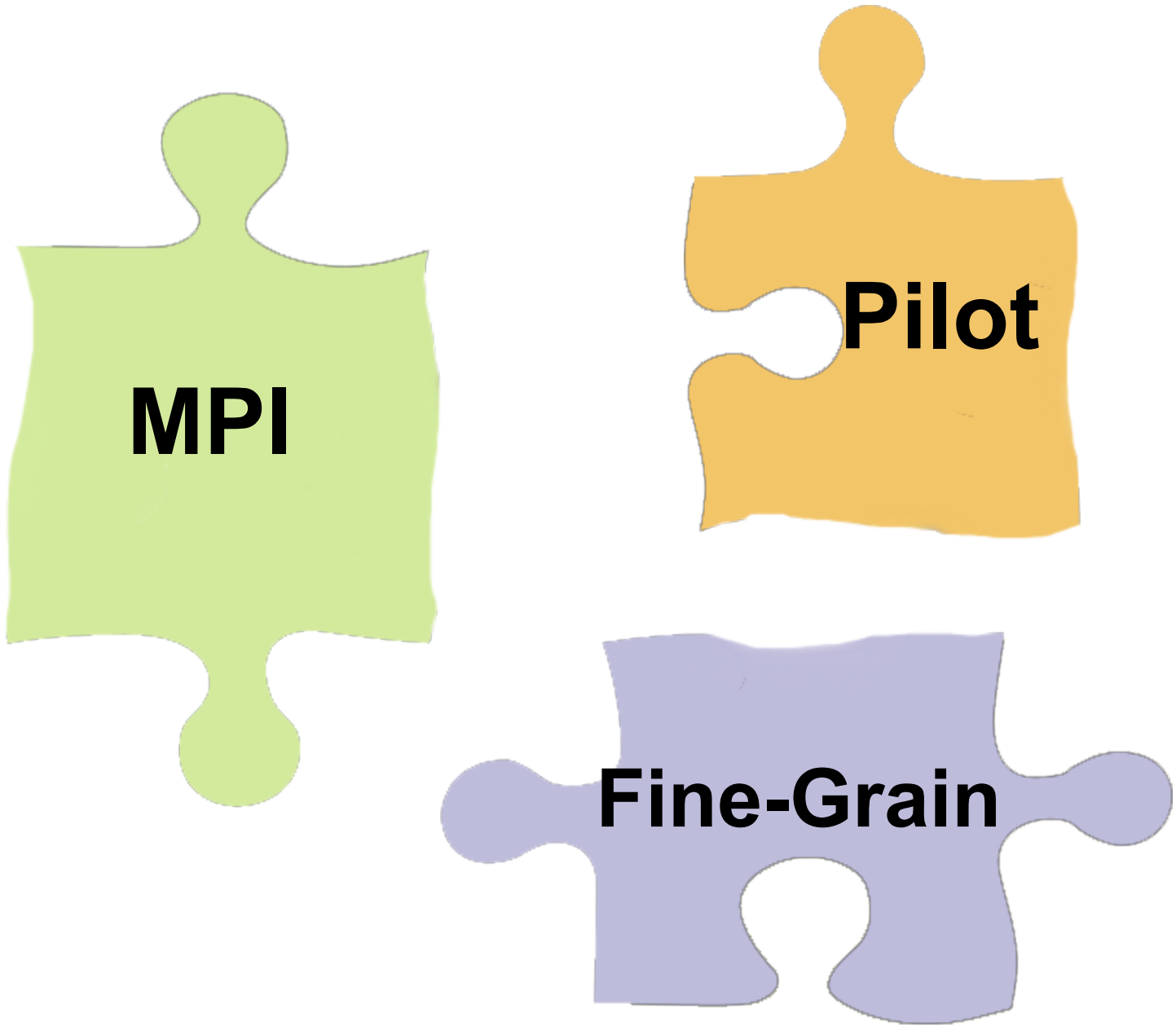
Supporting a process-oriented model in MPI through fine-grain mapping

Master Thesis Presentation
The University of British Columbia

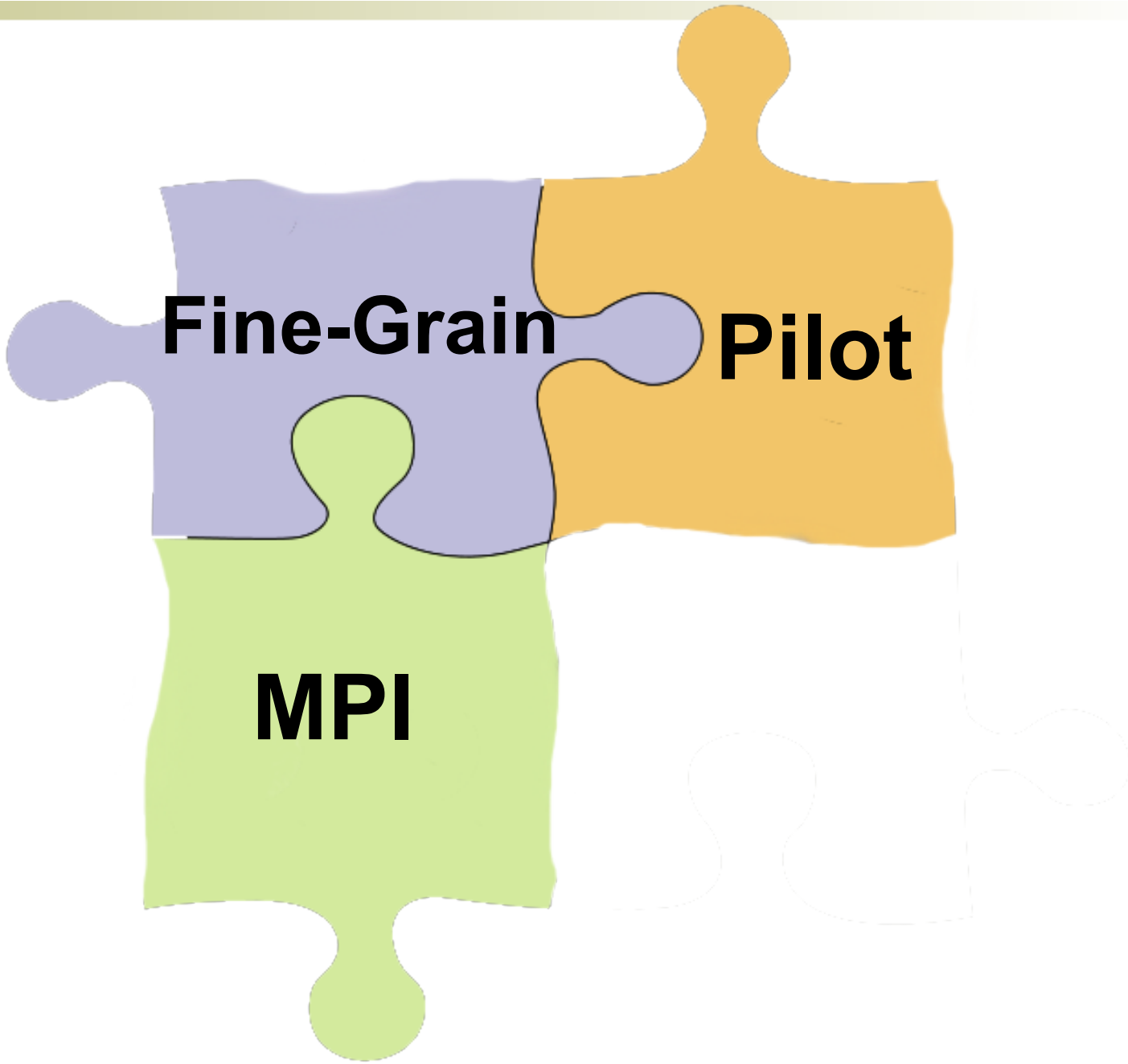
Cody R. Brown
Friday, May 25, 2012
ICCS 238, 2366 Main Mall
Supervisor: Alan Wagner

THE UNIVERSITY OF BRITISH COLUMBIA

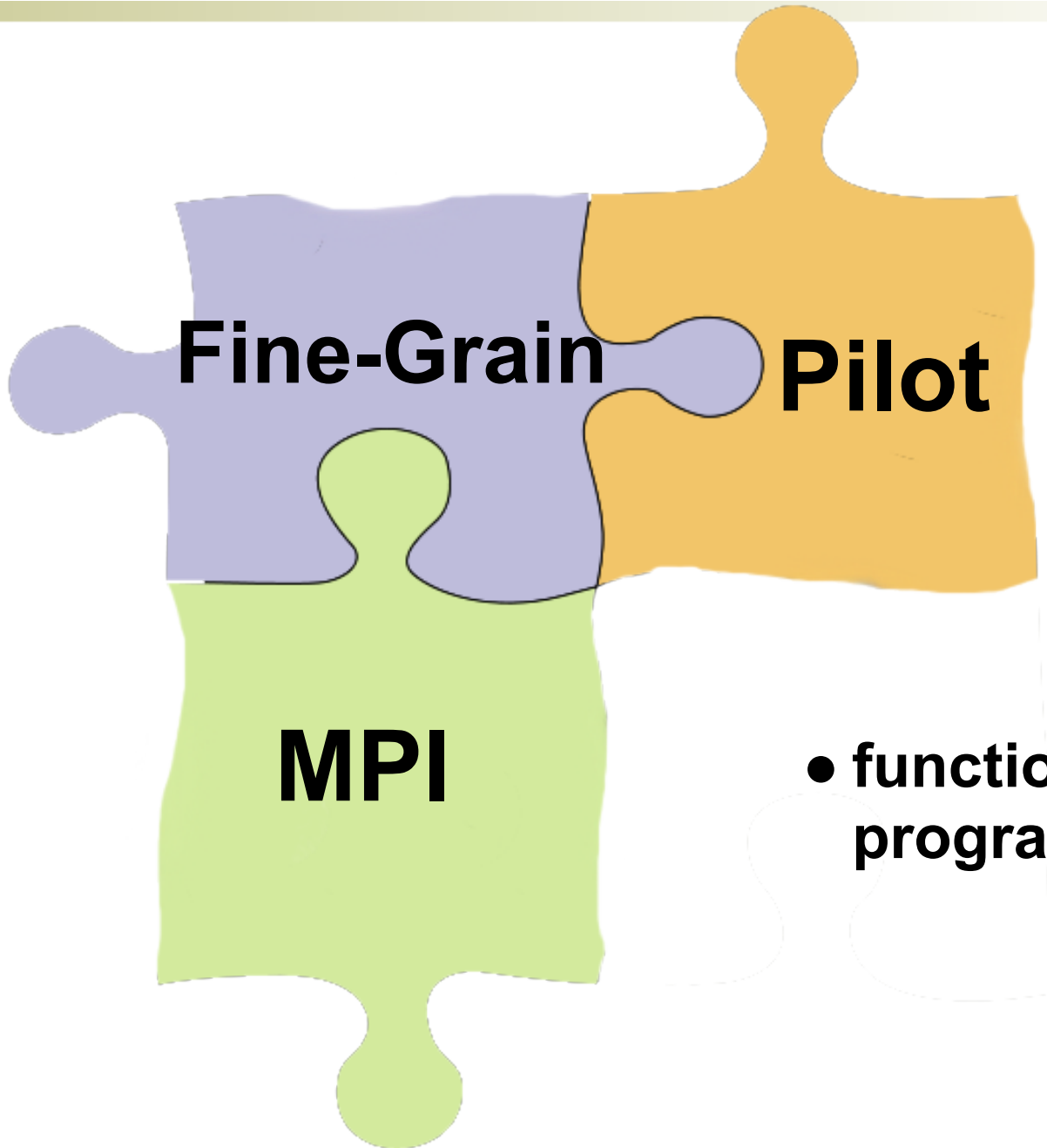
Motivation



Motivation

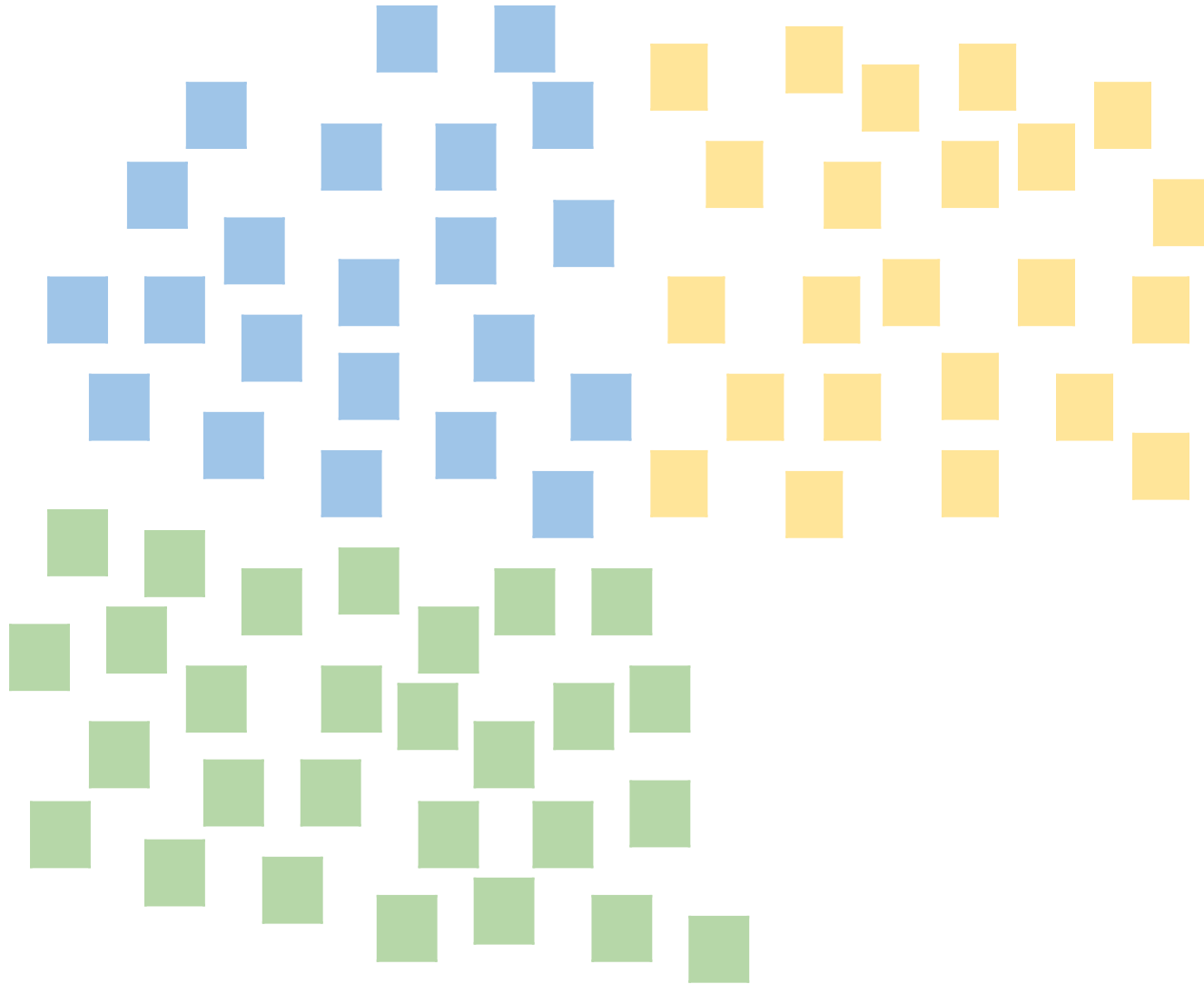


Motivation

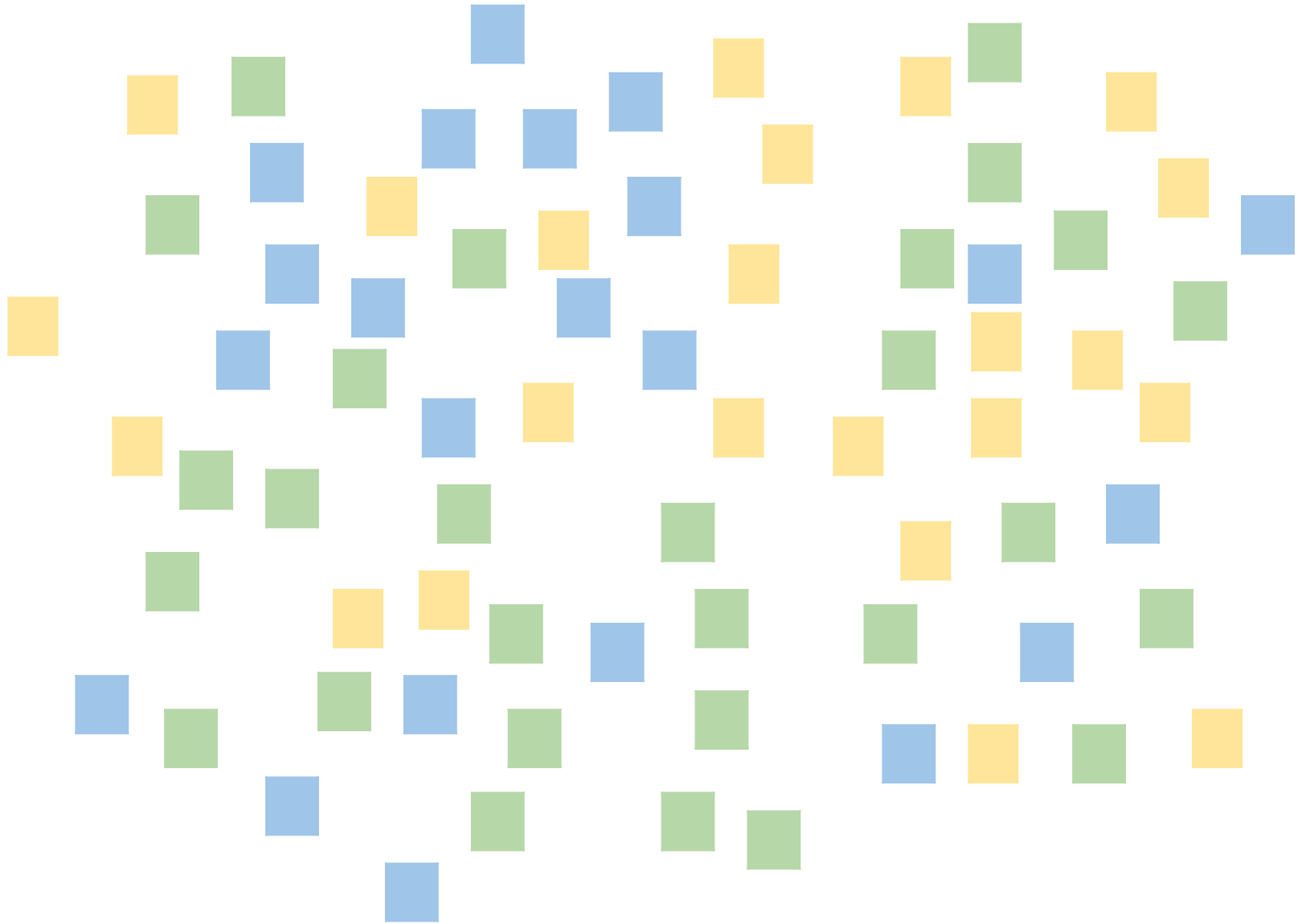


- function-level programming

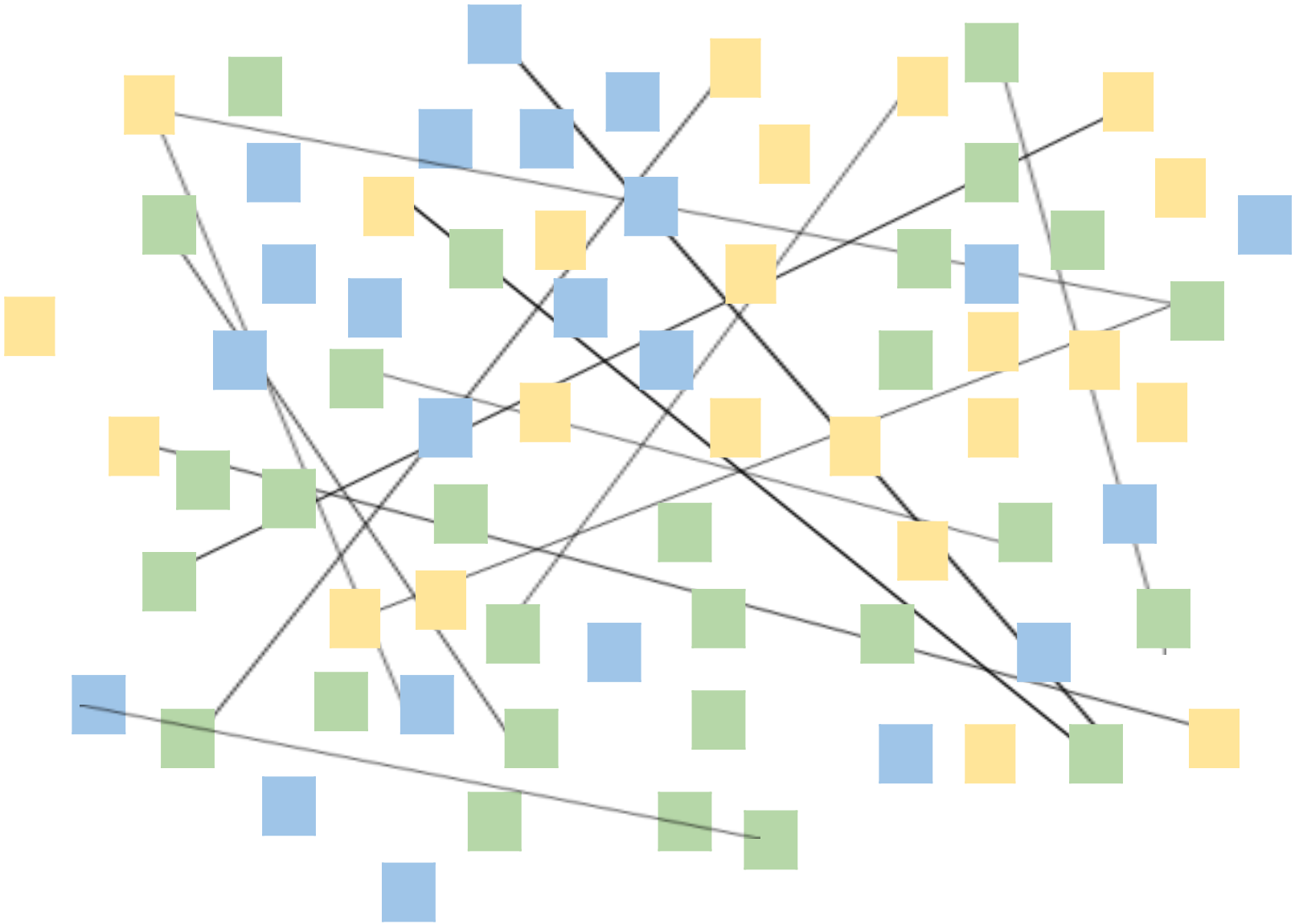
Motivation



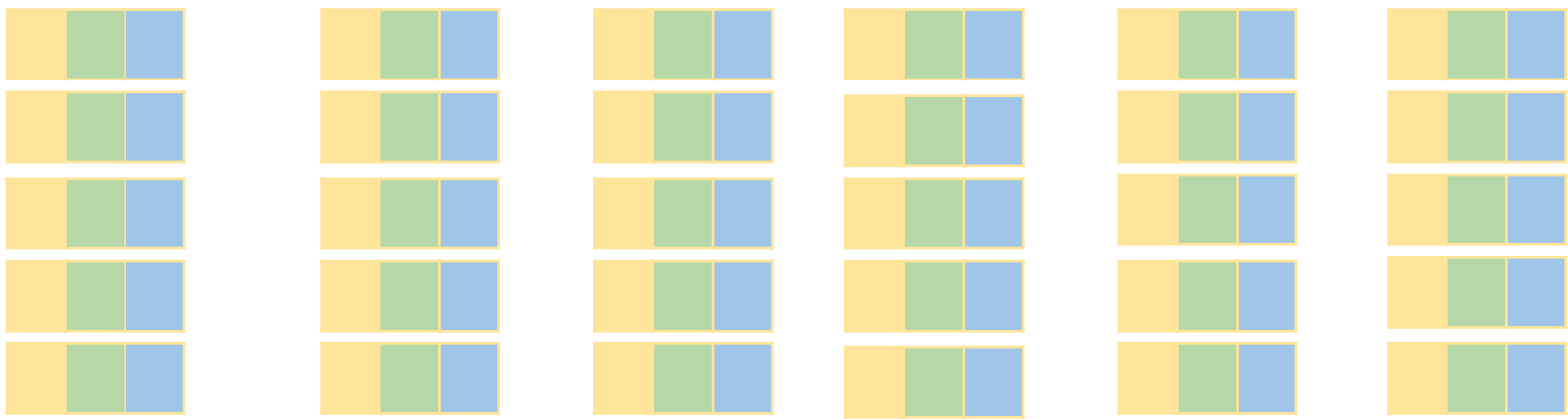
Motivation



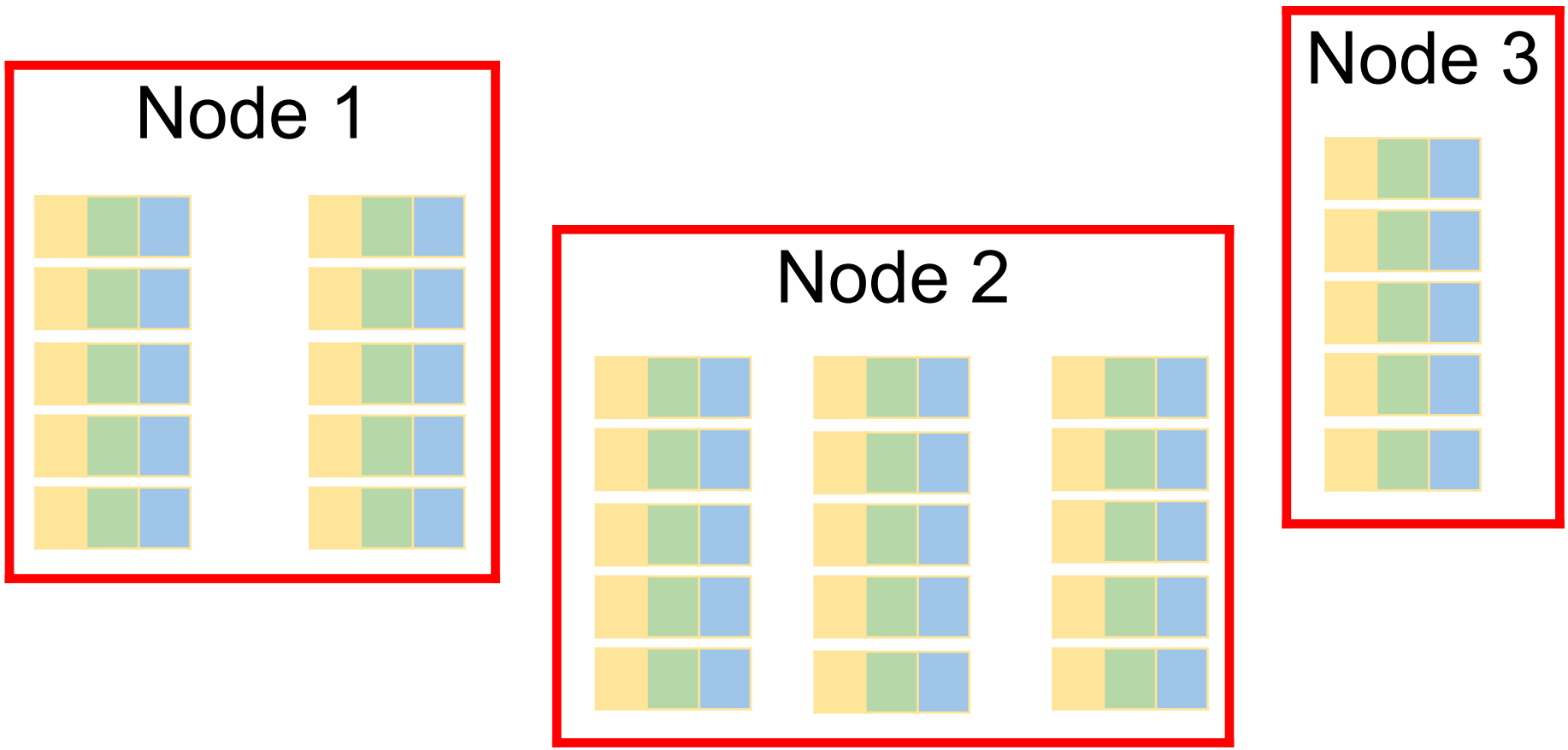
Motivation



Motivation



Motivation



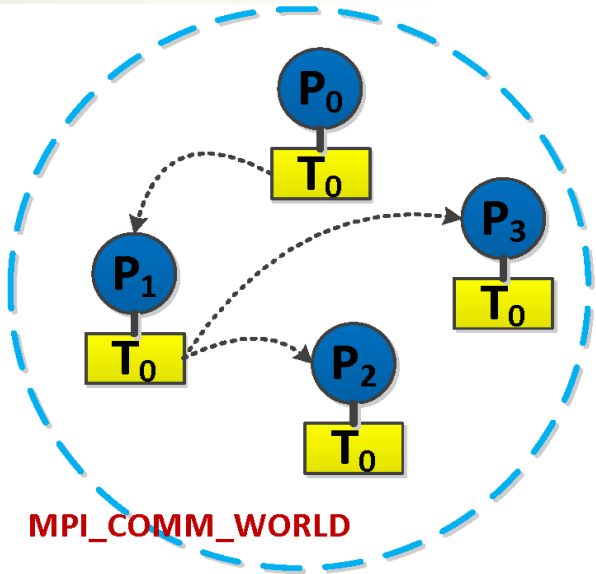
Overview

- Motivation
- Background and Tools Used
  - MPI, FG-MPI and Pilot
- Representation of the Graphical Format
  - Kahn Process Networks
- Implementation
  - Tool Architecture
  - A Distributed Example
  - A Pilot Example
- Evaluation
- Conclusion

Tools: MPICH2

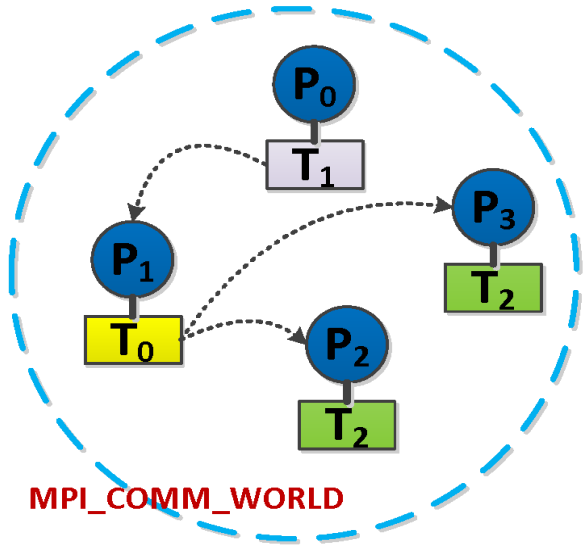
MPICH2:

- well used *send-receive mechanism*
- simple process management
  - through a *hostfile*
- supports *multiple* parallel programs communicating together



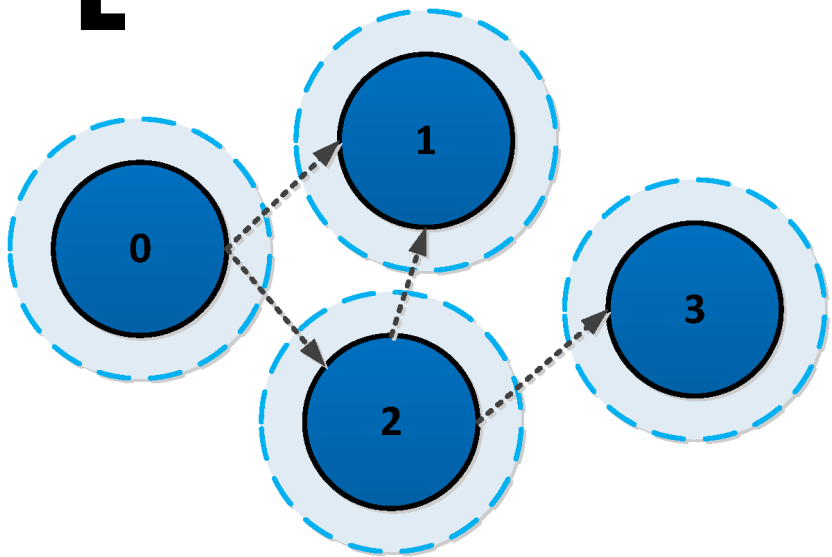
(a) SPMD

- process management too *coarse-grain*
- communication is too *free*
  - prone to error, or deadlocks

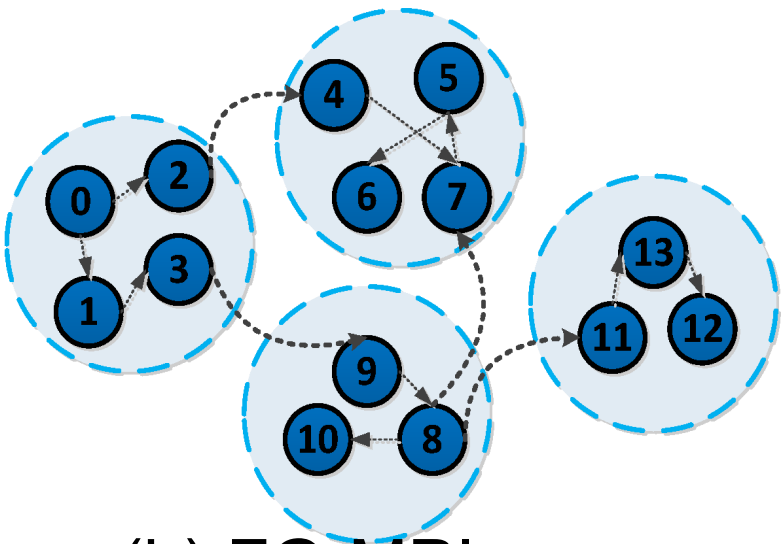


(b) MPMD

Tools: FG-MPI



(a) MPICH2



(b) FG-MPI

FG-MPI:

- developed at UBC
- reduces the *unit of parallelism*
  - *uses co-routines*
  - *supports function-level*
- decouples the MPI rank from hardware

Tools: FG-MPI

FG-MPI: Global variable issue

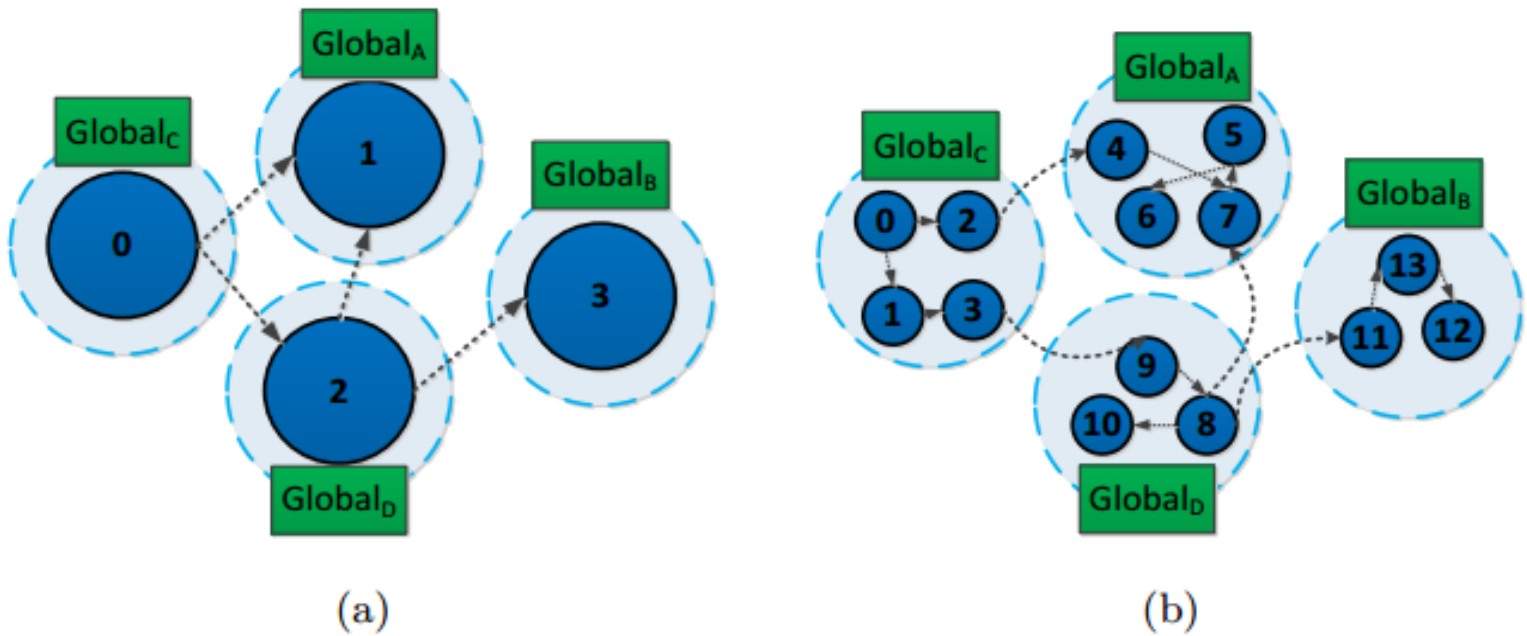
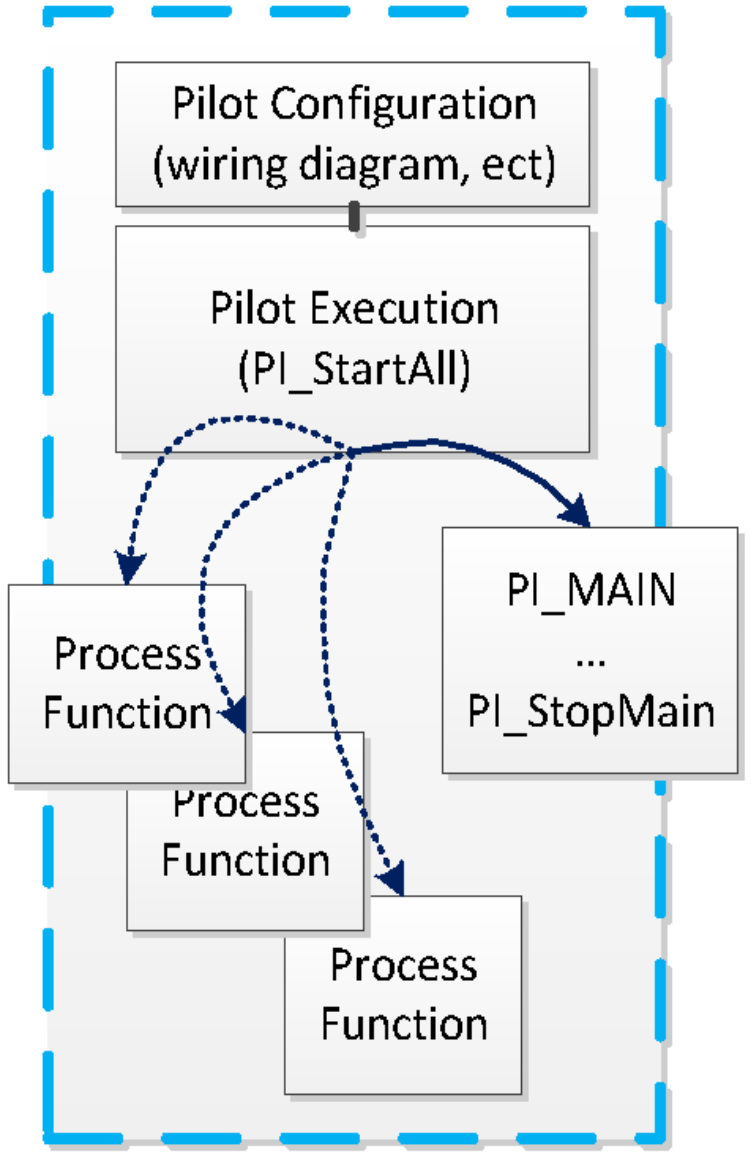


Figure 4.1: The shared global variable issue. Dotted circles are OS-processes, solid circles are MPI processes, and squares are shared global variables. (a) is the MPICH2 case, where each MPI process has its own global writable variable, while (b) is the FG-MPI collocated case, where several MPI processes share the same global variable; care must be made when writing to these variables.

Tools: Pilot

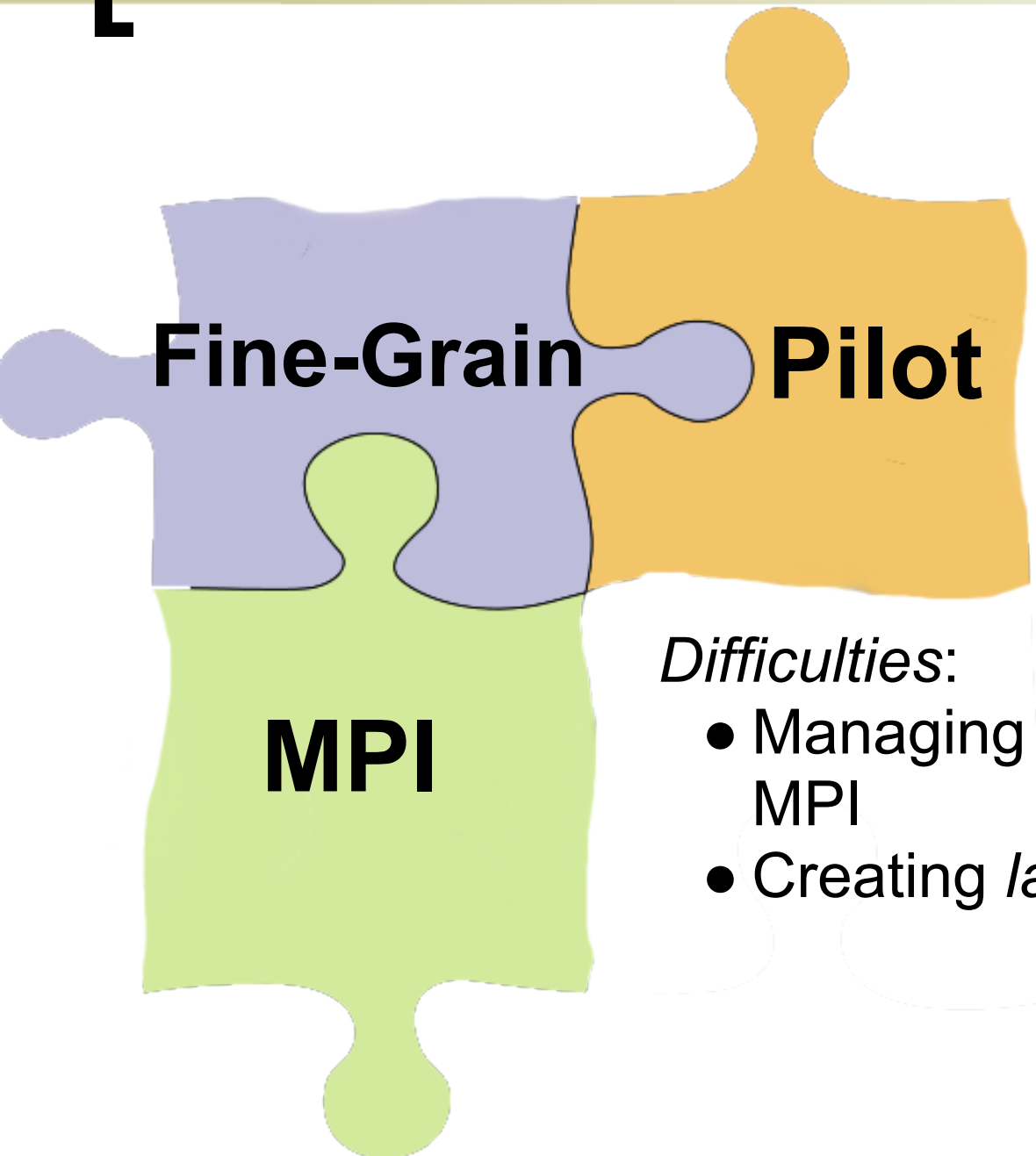
Pilot:

- developed at *University of Guelph*
- MPI library that provides **CSP properties**
  - avoids *communication errors* and *deadlocks*
  - creation of a *wiring diagram* necessary



- Not **fine-grain**
- wiring diagrams **difficult** to create with lots of elements

Tools: Together



- Pilot use MPI processes
- *Combining* Pilot with FG-MPI allows a large number of elements

Difficulties:

- Managing *many* small elements with MPI
- Creating *large* wiring diagrams for CSP

The Idea

Extending MPI to use a *Graphical* format instead of a *hostfile.*

- easy to *represent*, lots of tools
- easy to *manage* large groups of elements
  - clusters of nodes can be group together
  - isolating elements and repeating patterns intuitive
- easy to *construct* a CSP wiring diagram
  - use the same structure for different tasks
  - CSP diagrams can be auto-generate and updated

Simplify code

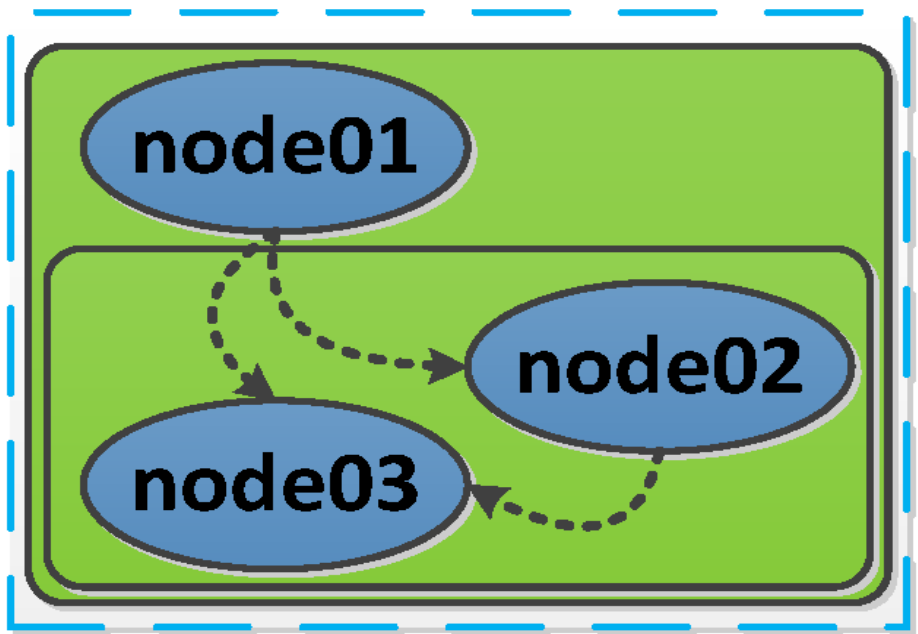
- module design to exploit graph information
- portable to many MPI implementations

flexible, manageable, scalable and extensible

Graphical Format

Kahn Process Networks (KPNs)

- used due to simplistic model
- graphically representable
- deterministic

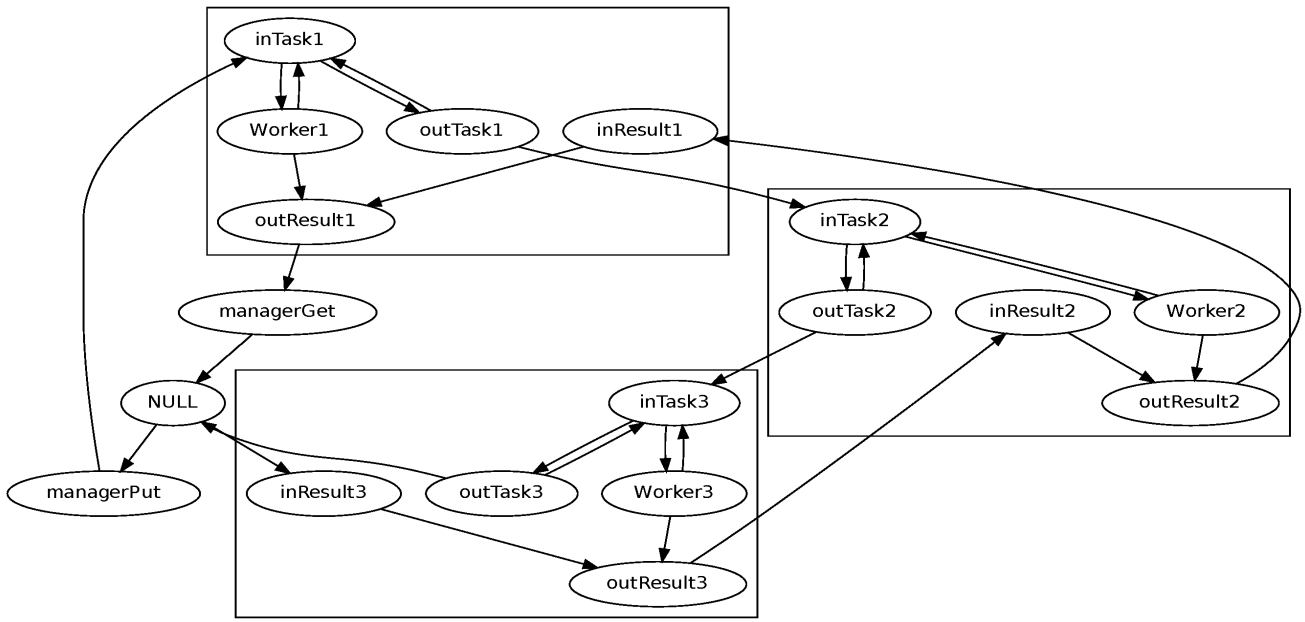


Graphical Format

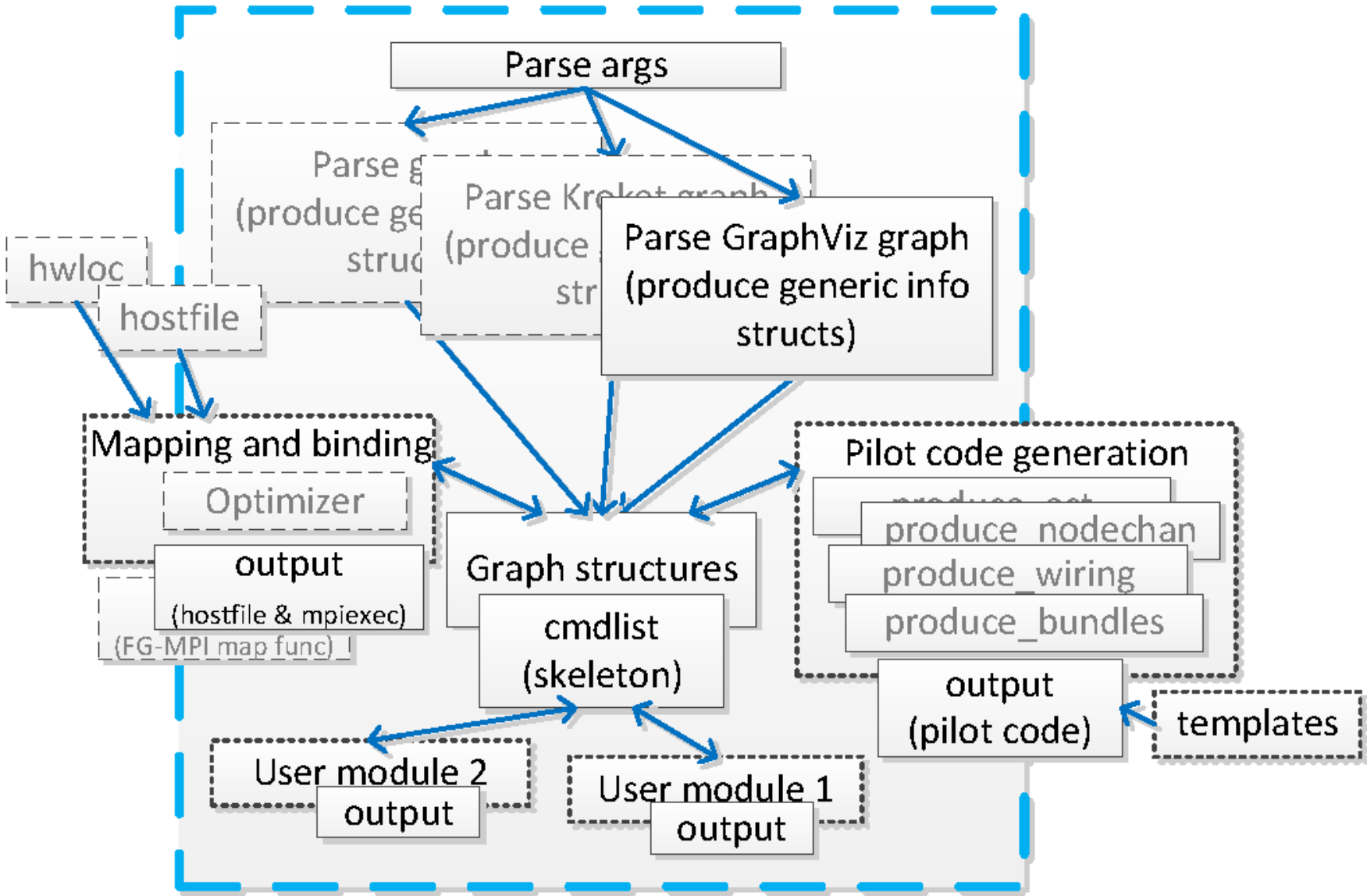
Represented as a *GraphViz file*

- nodes represent *processes*
- edges represent *communication channels*
- subgraphs are used to cluster elements together as *modules*

- *portable* format
- generic front-ends can be used to construct the graph
- modules or graph attributes use to provide *hostfile* info



Tool Architecture



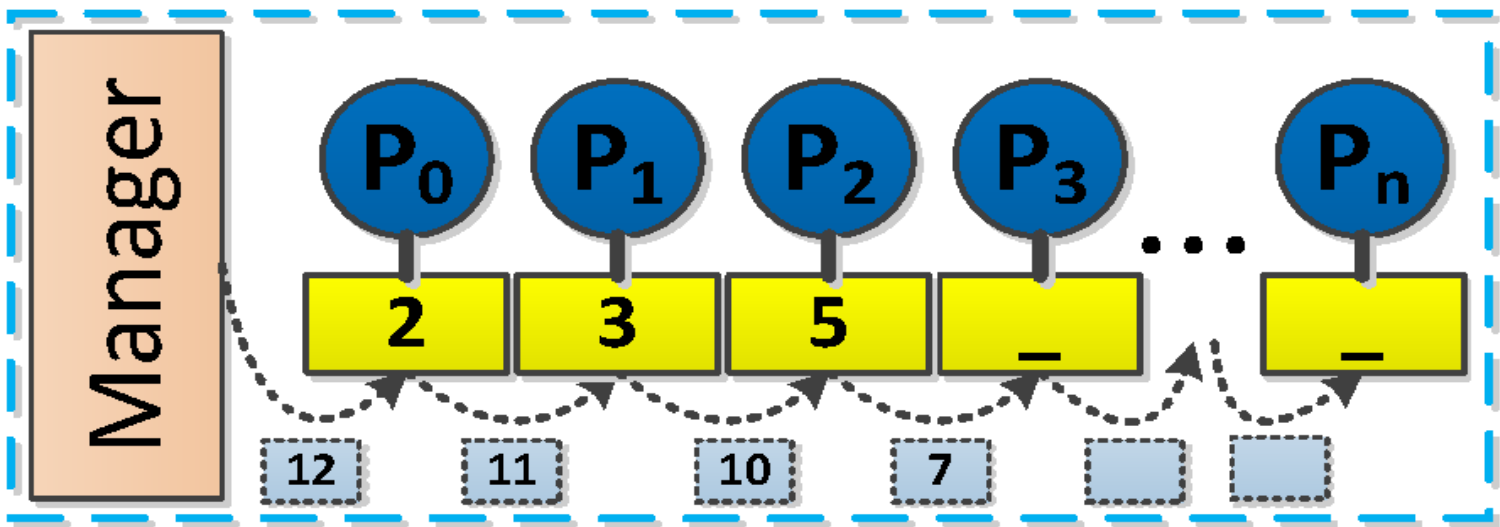
A Distributed Example

Why function-level parallelism:

- "scaling starts at 1"
- parallelism fundamental in the model

Distributed prime number sieve:

- application not easily coded in normal MPI



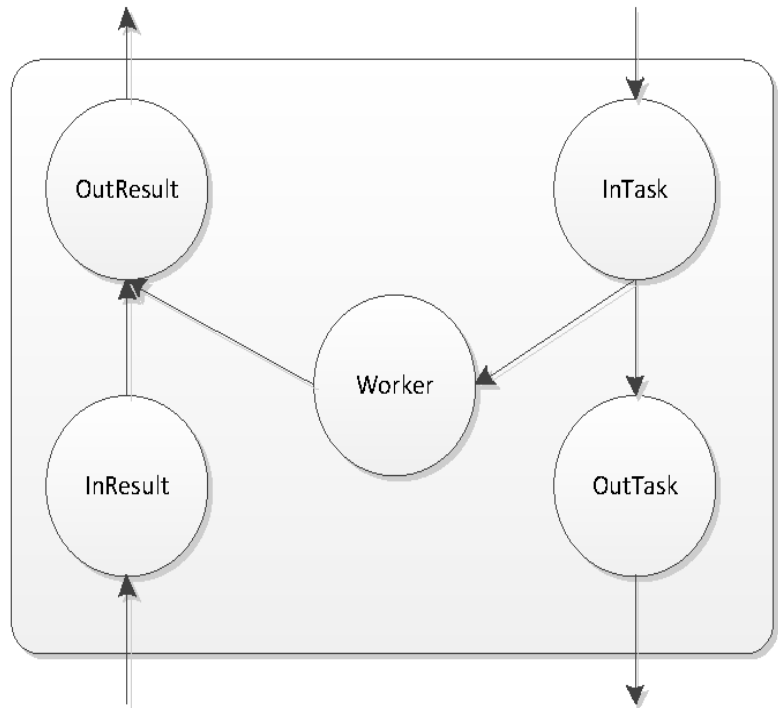
A Pilot Example

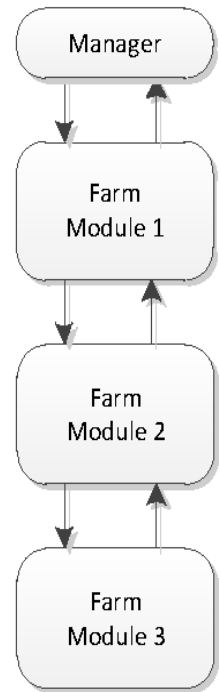
Pilot ported to FG-MPI:

- mainly remove global variables
- consideration of structures processes use
  - large repeatable structures should be shared/avoided

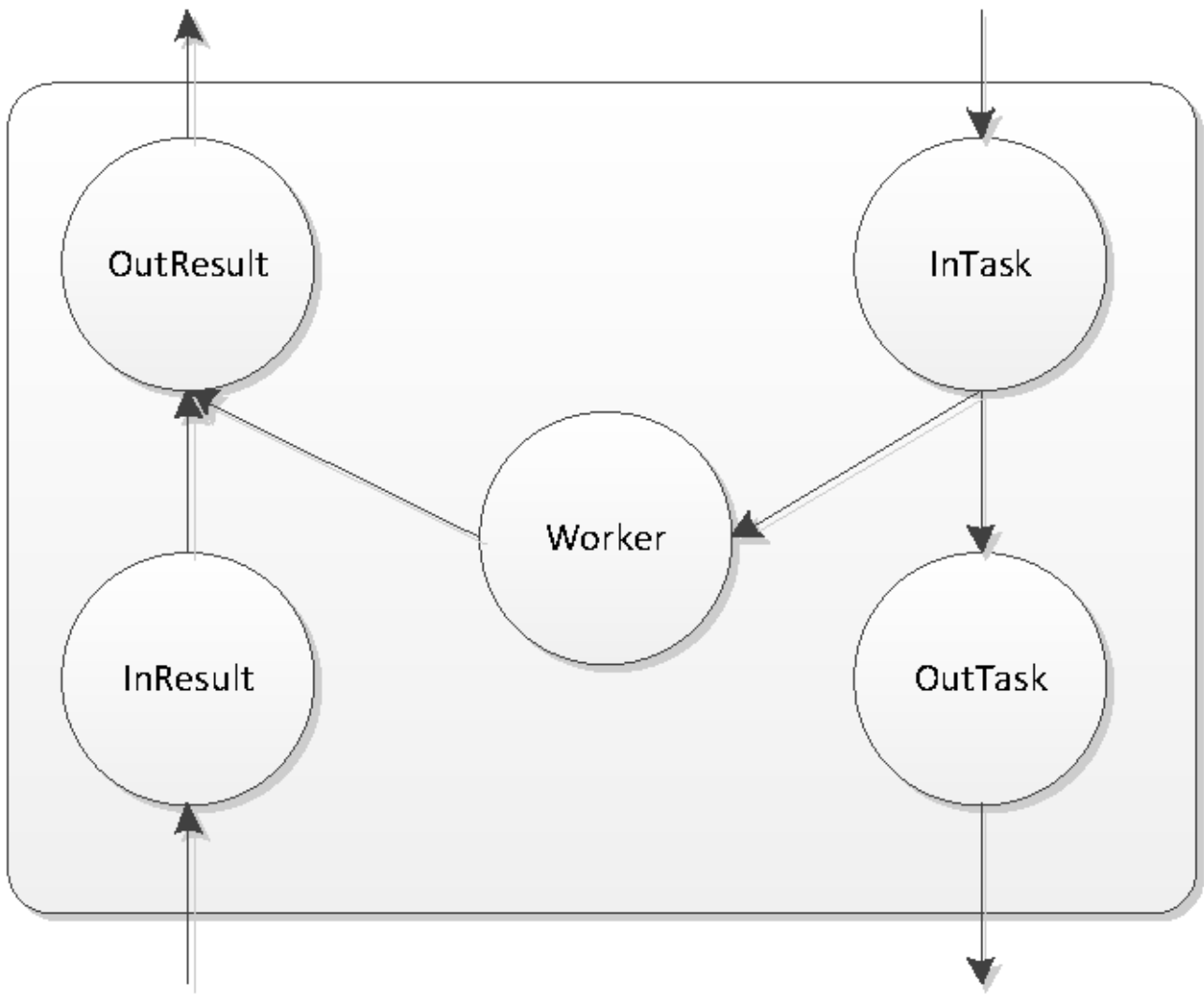
A Pilot farm was constructed:

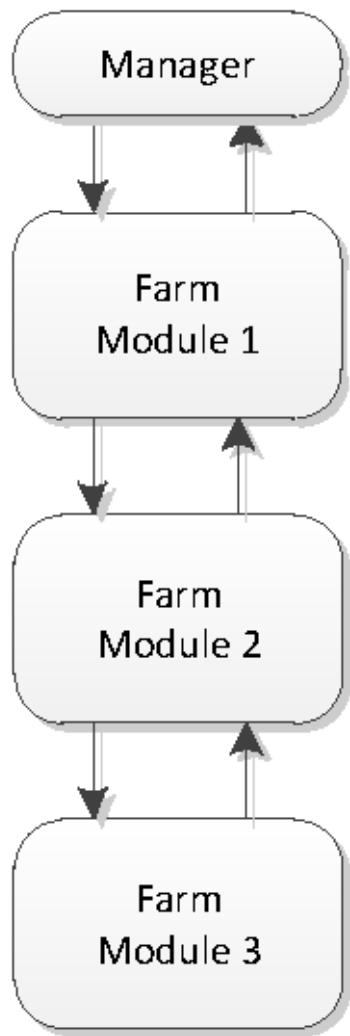
- a chain was used to provide easy communication patterns to analyse



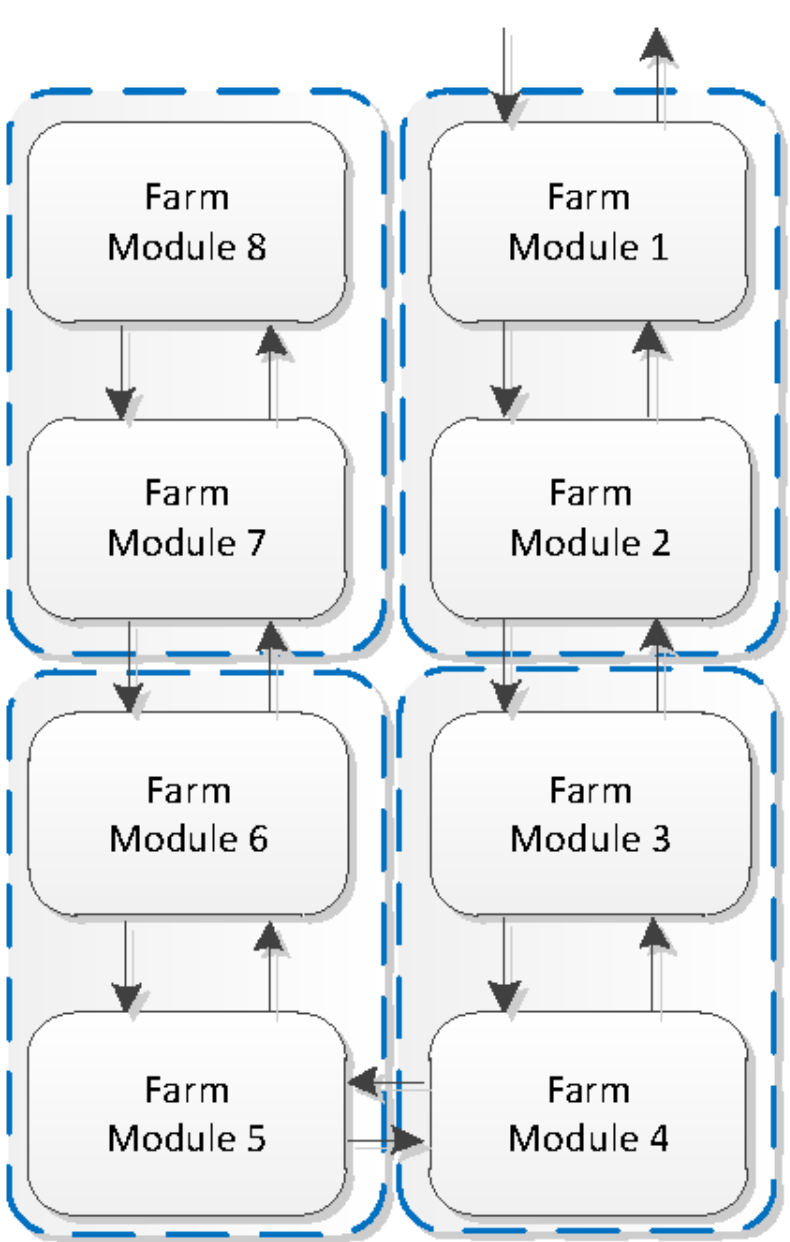


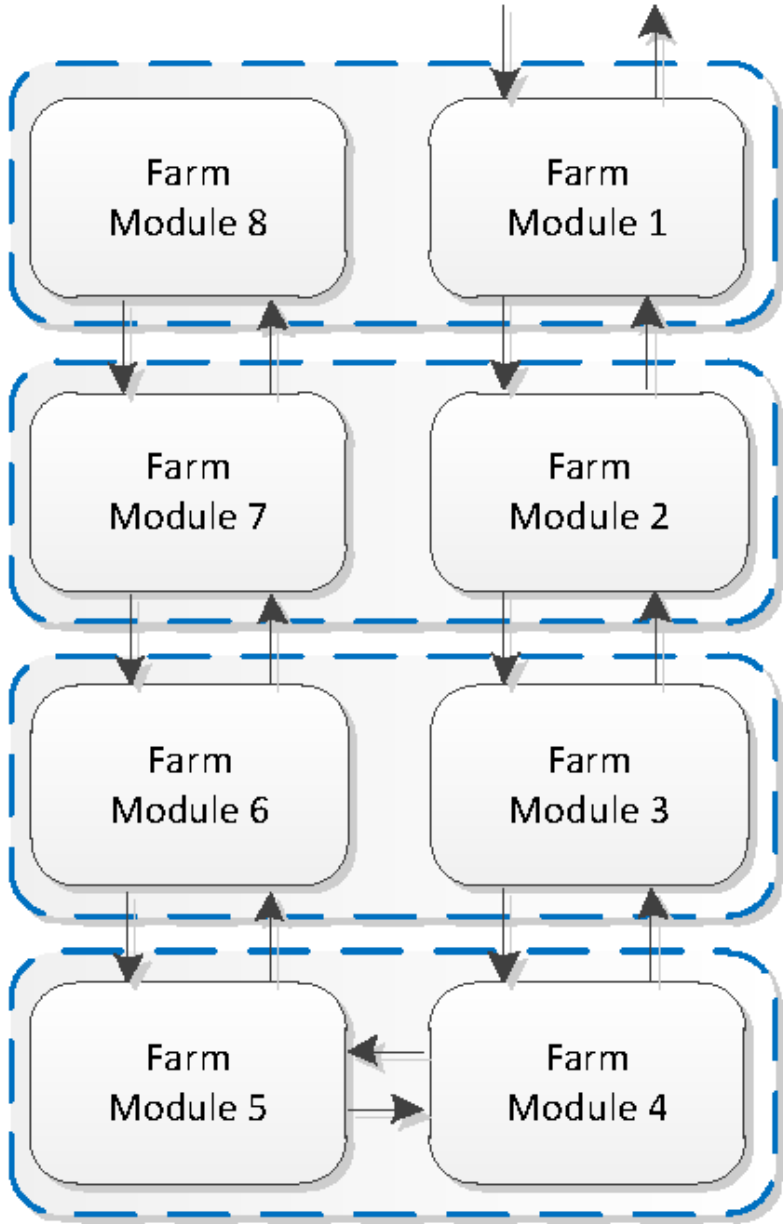
A Pilot Example: Farm Module





A Pilot Example: Stacking





Evaluation

To support this model in MPI some conditions must be shown

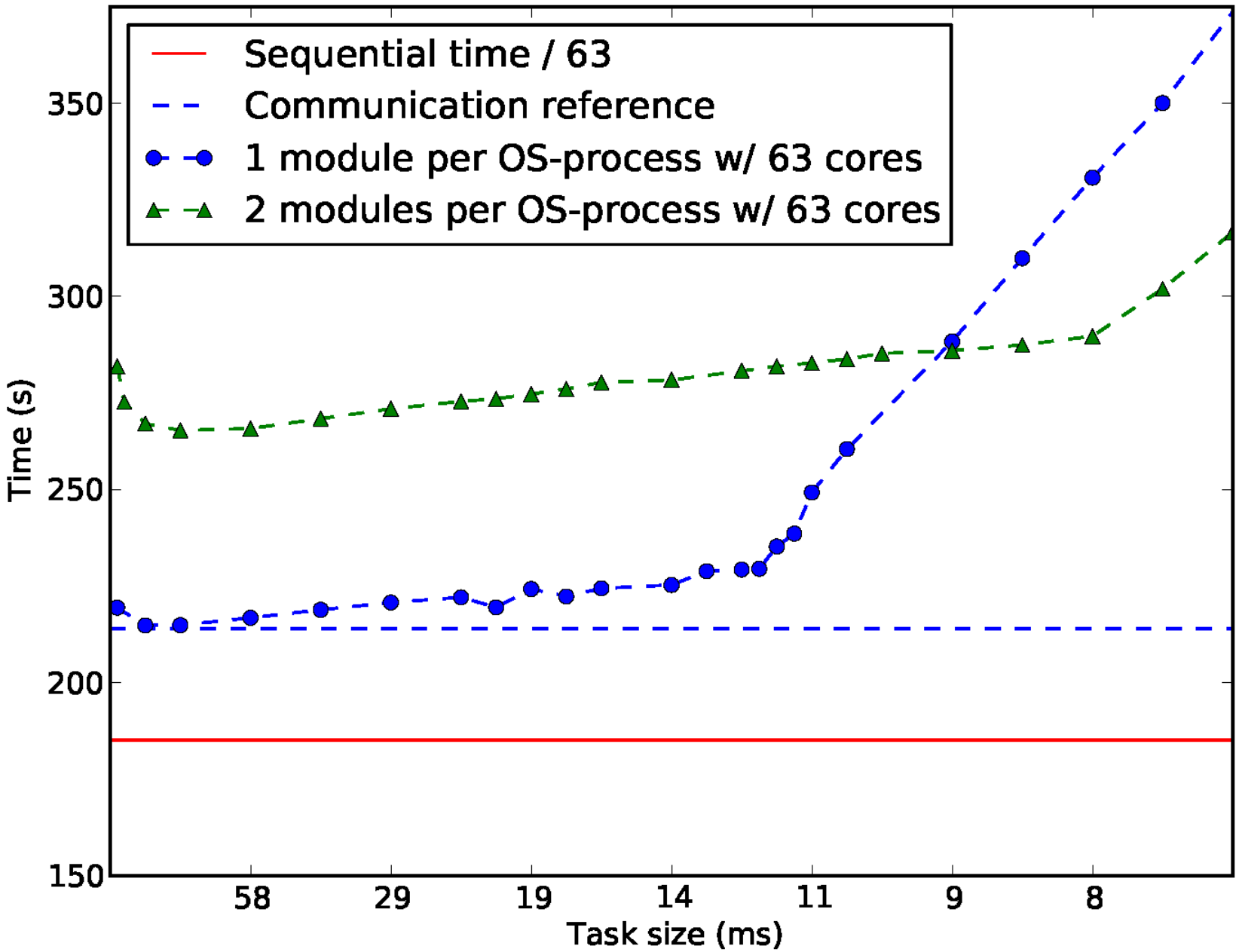
- *low communication overheads* must exist
- must be *scalable* to a large number of tasks
- a small *unit of parallelism* (task size) must be achievable

Evaluation: Cost of an MPI_Send

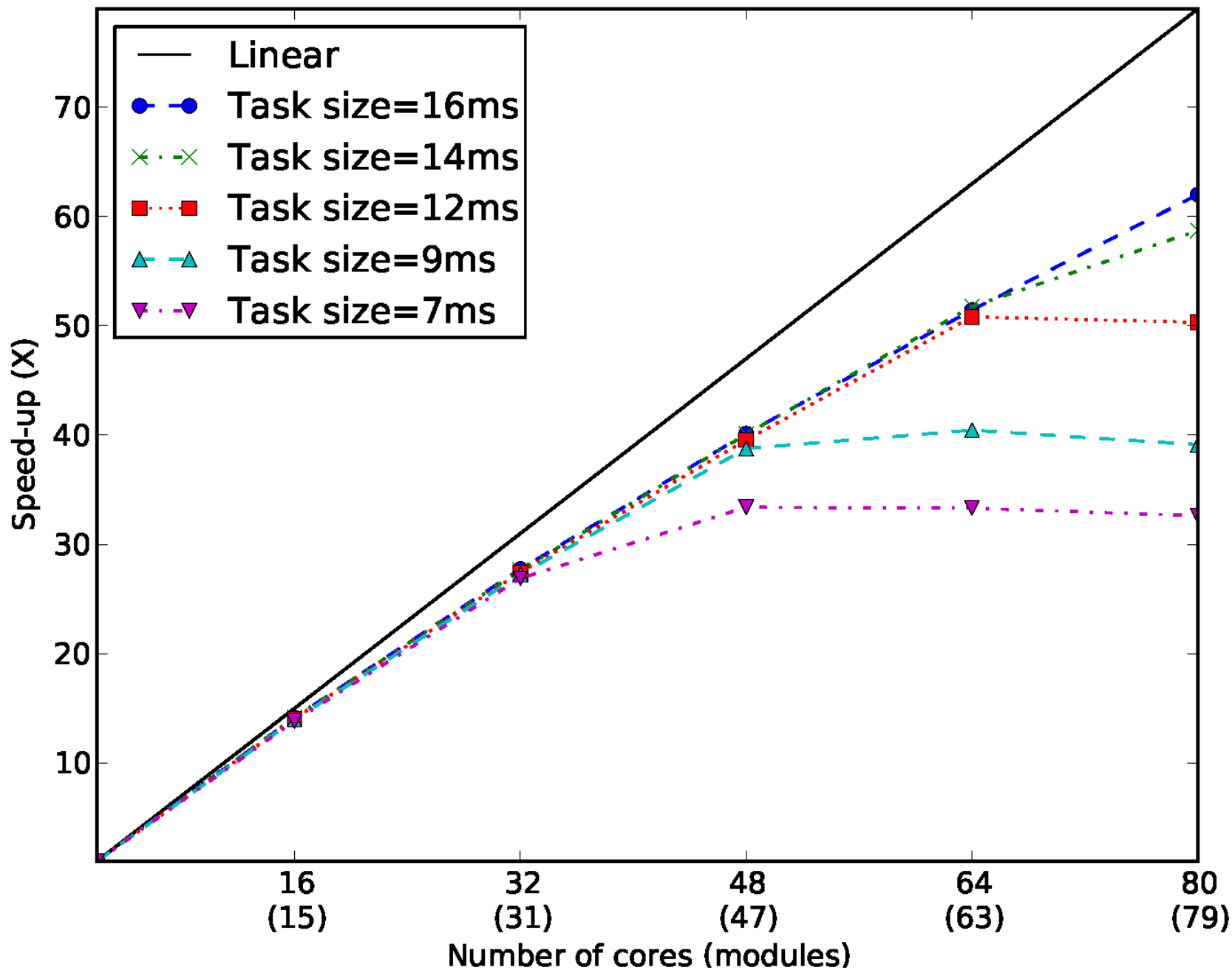
Situation	Compute	Communication	Time
Between Nodes	Parallel	TCP	62.47μs
Between OS-Processes	Parallel	Shared memory	3.13μs
Within Collocated Processes	Shared	`memcpy`	1.41μs

Table 5.1: Communication costs for a single one-way `MPI_Send` in FG-MPI for a message size of 8192 bytes on the test setup.

Evaluation


Sequential time / 63
Communication reference
1 module per OS-process w/ 63 cores
2 modules per OS-process w/ 63 cores
Time (s)
350
300
250
200
150
58
29
19
14
11
9
8
Task size (ms)

Evaluation


Linear
Task size=16ms
Task size=14ms
Task size=12ms
Task size=9ms
Task size=7ms
Speed-up (X)
70
60
50
40
30
20
10
16
(15)
32
(31)
48
(47)
64
(63)
80
(79)
Number of cores (modules)

Evaluation

Task size	Total time	Amount of comm.	Average time per comm.
14.6ms	1648.0s	2,000,040	1.52μs
2.9ms	1660.2s	9,999,920	1.52μs
1.8ms	1667.8s	16,000,040	1.43μs

Table 5.2: A seven module chain with 35 collocated processes within one OS-process. Task size and number of tasks were varied to maintain a constant amount of work. Average time per intra-process communication is shown.

Conclusions

A graph specification was developed, extending MPI to represent *hostfile* information, and other attributes graphically.

A modular tool to use this graph file, and create complicated mapping and binding information was created:

- the tool is generic and is usable by many implementations
- user created modules can auto-generate code, such as Pilot wiring diagrams, from the graph data

Evaluation was done to calculate the smallest unit of parallelism in this test setup on a chain structure.

- this information can be further used in optimizations
- other structures are possible, like trees

Acknowledgements

More information is available in the thesis:
http://hdl.handle.net/2429/42152

Thanks goes to **Humaira Kamal** for managing FG-MPI:
http://wiki.nss.cs.ubc.ca/FineGrainMPI

And to **William Gardner** for providing the Pilot library:
http://carmel.cis.uoguelph.ca/pilot/

Questions?

Title

Future Work

Pilot:
Memory optimizations
backward compatible Pilot

Graph:
Specific front end GUI
Abstract graph definition

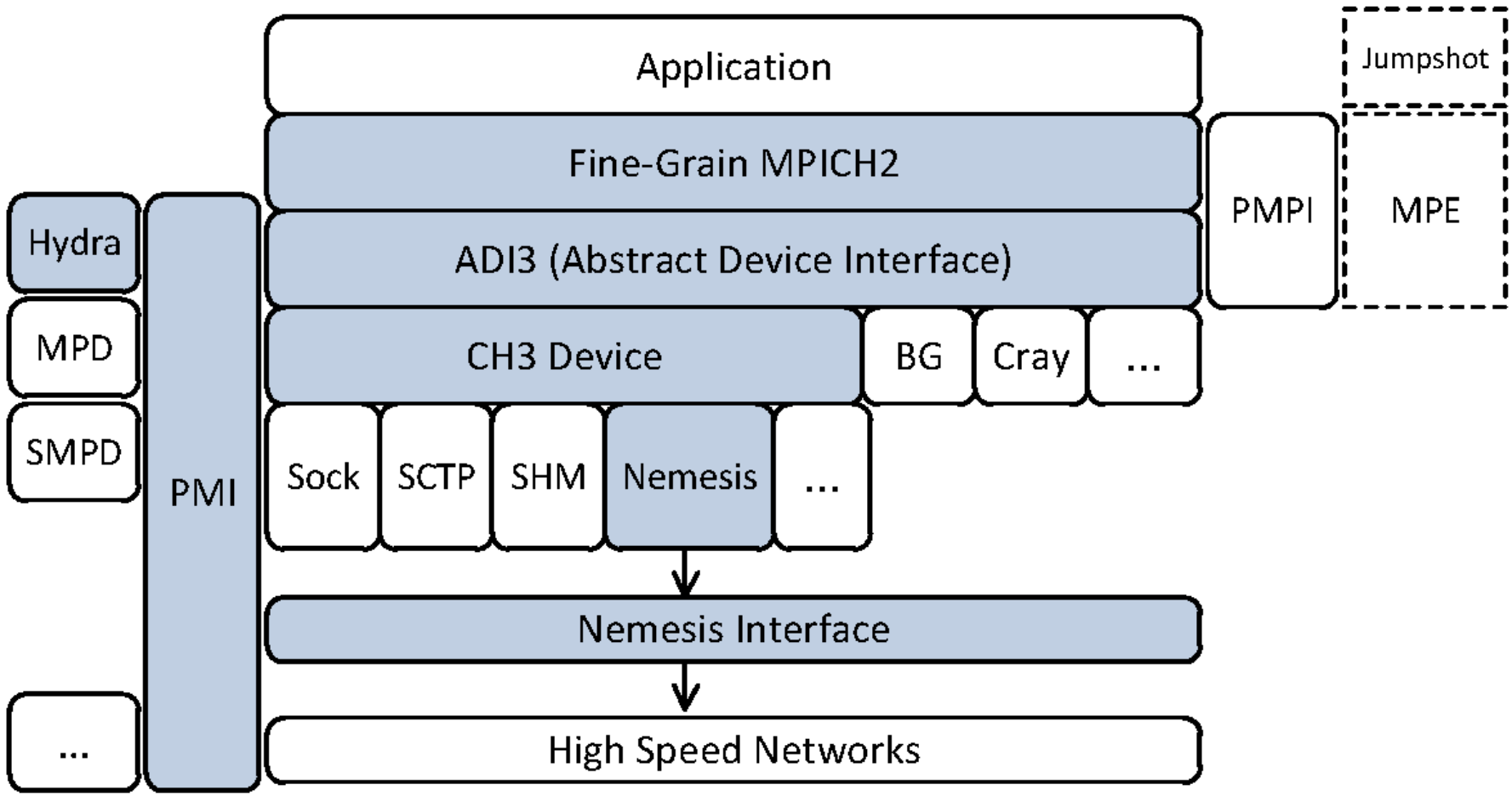
More in-depth optimization modules, use of heavy graph algorithms

Reducing unit of parallelism

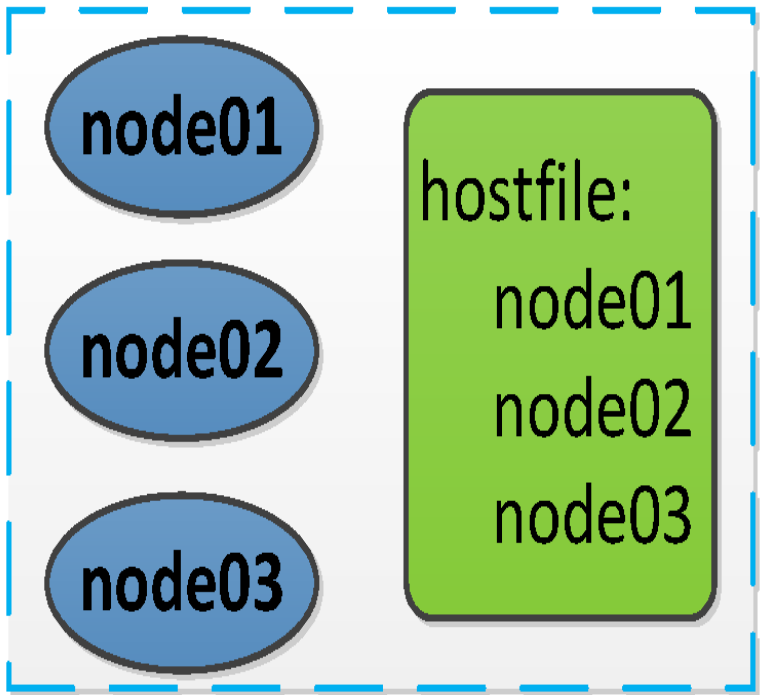
(a) (b) (c)

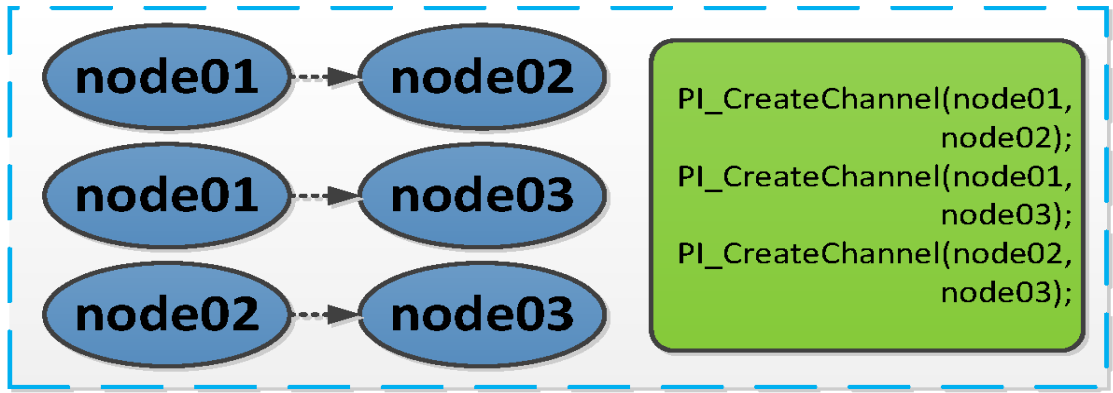
Figure 5.1: Showing a reducing unit of parallelism from (a), to (b), to (c), for each consecutive OS-process. Large dotted circles represent OS-processes, while small solid circles represent MPI processes.

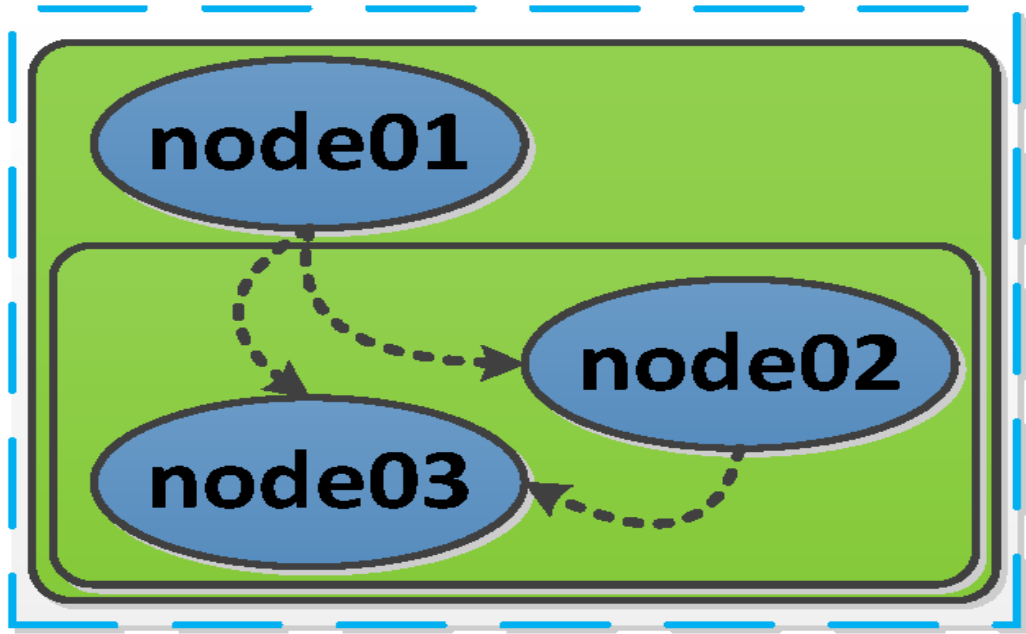
FG-MPI Architecture



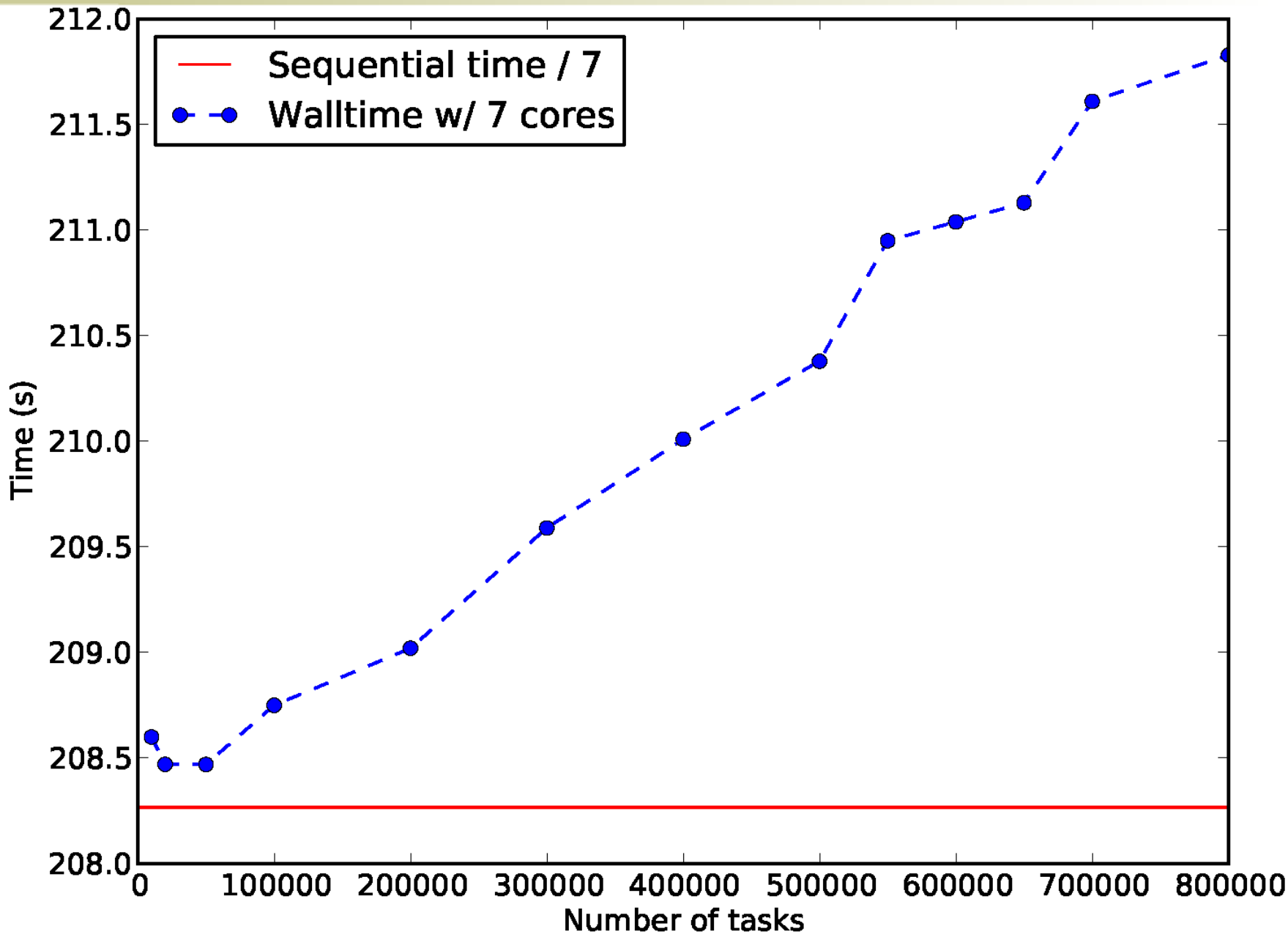
Grpahical Formalization







Evaluation


Sequential time / 7
Walltime w/ 7 cores
Time (s)
212.0
211.5
211.0
210.5
210.0
209.5
209.0
208.5
208.0
0
100000
200000
300000
400000
500000
600000
700000
800000
Number of tasks