Defining supersteps for BSP

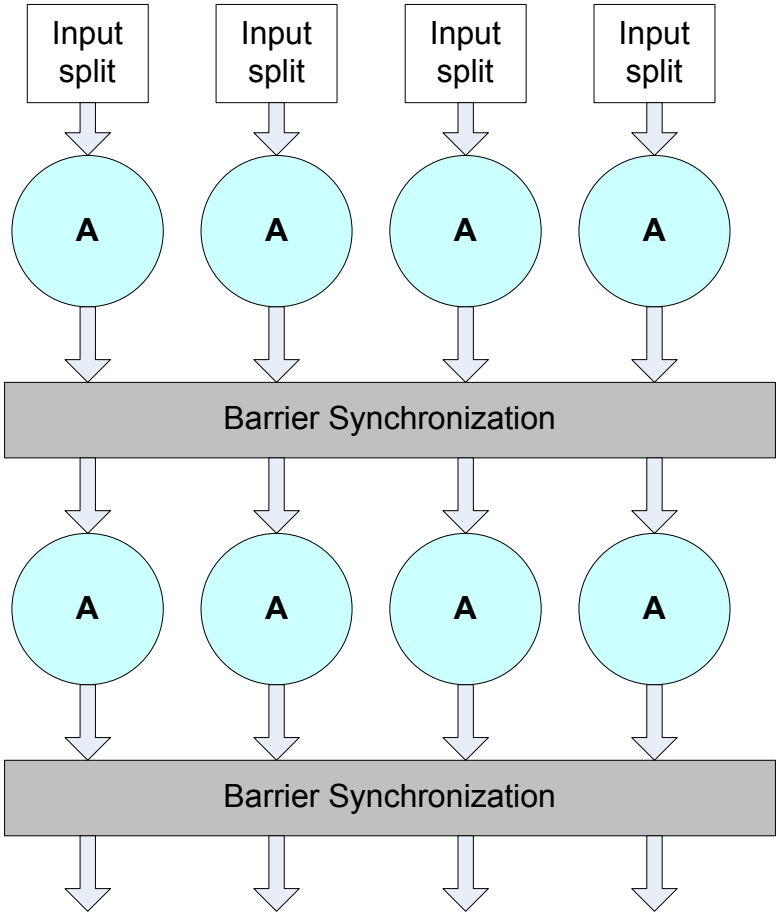

Figure 1 Superstep with same nature of computation executed in parallel today. (The arrows denote just the temporal progress of task and not the transfer of messages between tasks.)

Support for different genres of computation in parallel

In today's HAMA design, we assume that at an instant, the supersteps running in parallel have the same nature of computation (bsp function implementation in the same BSP class). As shown in Figure 1, A resembles the execution of bsp() function defined in class A. We are bound to have usages of Hama where a single job would require different genres of computation running in parallel. With current design where a single bsp function defines the implementation of all supersteps, we would have to implement them with multiple if/else or switch case conditional statements. There is a scope/need for improvement here. Such a case is described in Figure 2. It shows how different genres of task would be required to be executed in parallel and Hama should support this in job submission and task scheduling in cluster.

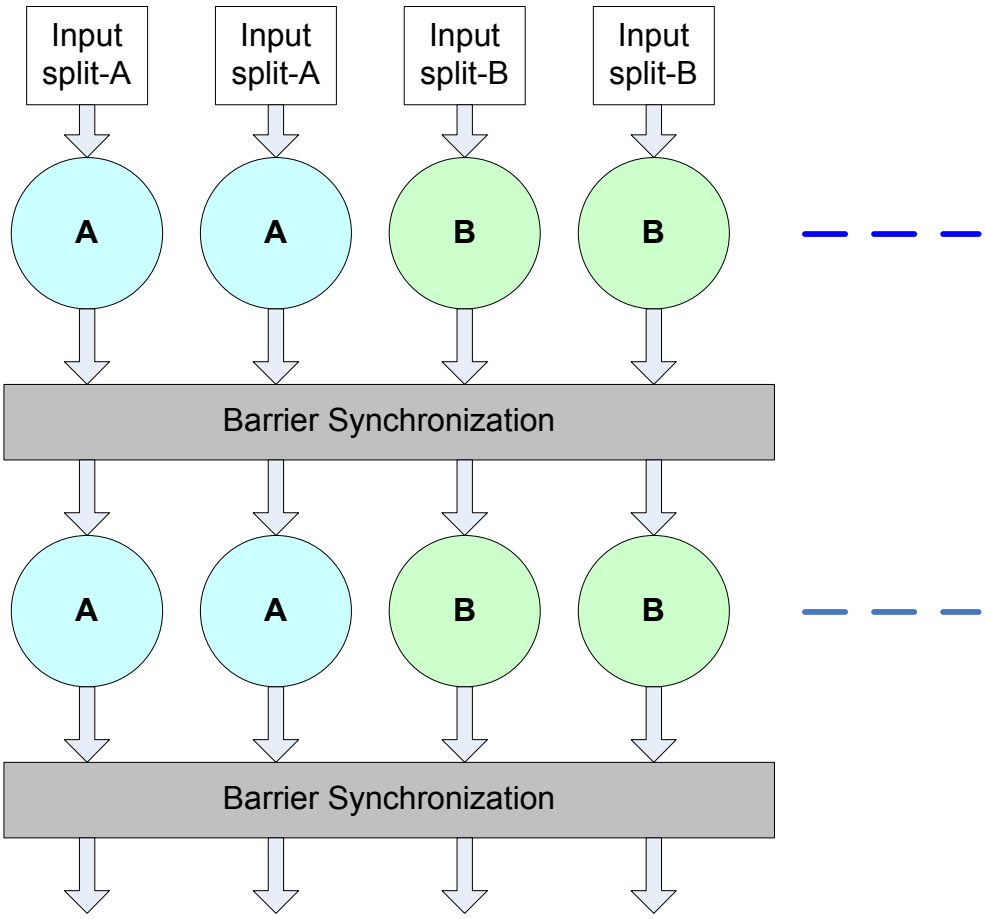

Figure 2 An example task where different nature of computations would be executed in parallel that is defined and scheduled to run in one BSP class. . (The arrows denote just the temporal progress of task and not the transfer of messages between tasks.)

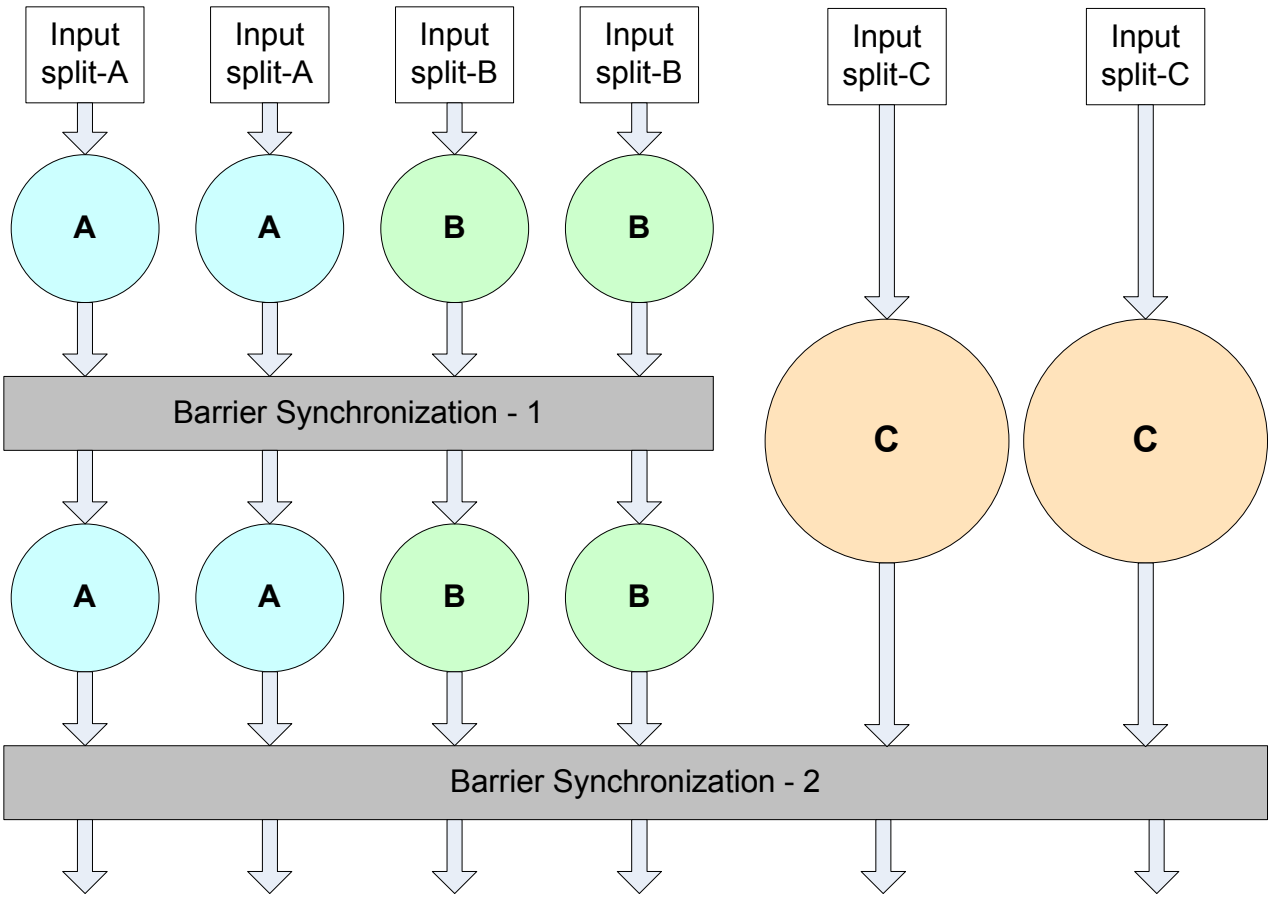

Figure 3 Selective superstep synchronization.The first synchronization period does not involve superstep synchronization of C. (The arrows denote just the temporal progress of task and not the transfer of messages between tasks.)

Support for selective superstep synchronization

Figure 3 shows the situation explained in [1] where not all supersteps should be synchronized. In this figure, task C takes either too long to complete execution or expects inputs from A and B after their 2 iterations. In either case, BSP model does not encourage slowing down A and B for synchronization with C in every superstep. The Hama job submission API should thus support configuring the points at which a particular set of tasks should enter synchronization or skip them.

Task precedence

Today we have a way to define with only single line of flow of computation. Consider a task execution graph as shown in DAG in Figure 4. The BSP execution model for the same is shown in Figure 5. Hama job submission API should support this.

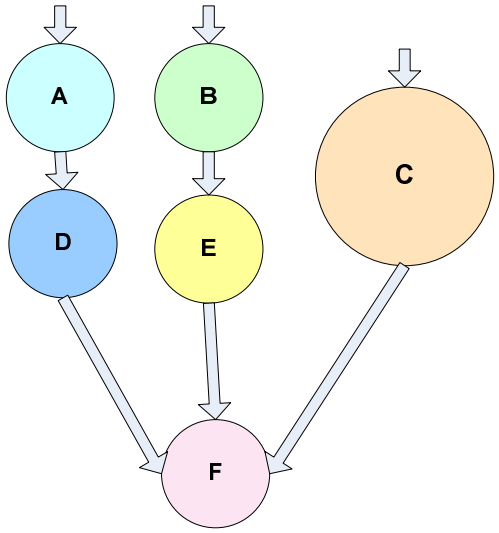

Figure 4 An example task precedence graph

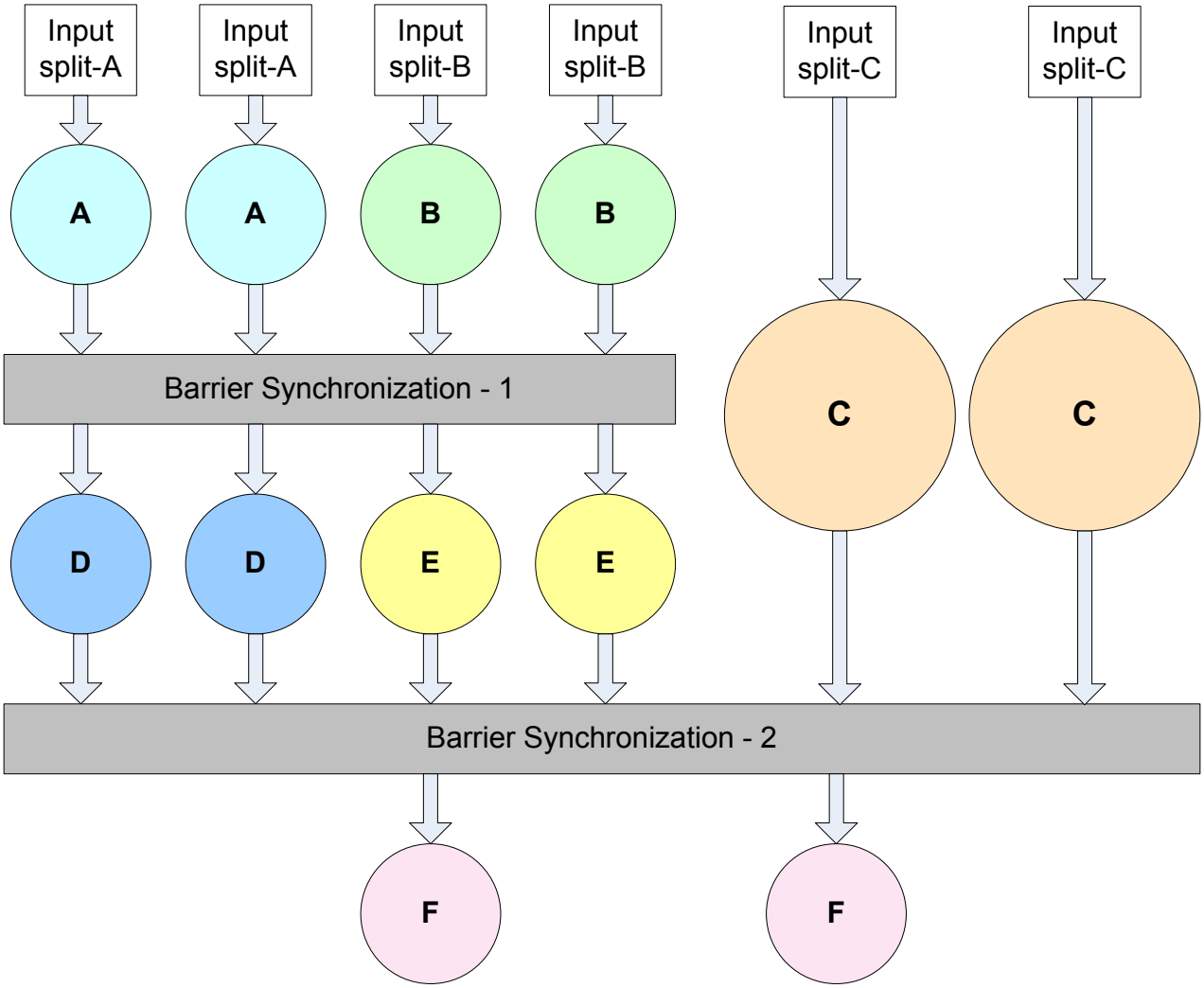

Figure 5 BSP task execution of tasks with precedence explained in Figure 4. (The arrows denote just the temporal progress of task and not the transfer of messages between tasks.)

Job Submission

So from the above observations, we can conclude that our input is a graph that defines the precedence of tasks and thereby the points in the computation supersteps where synchronization should be performed. This means a user should be able to provide complete information on the task structure which is a graph in Hama job submission. Graphs could be expressed as adjacency list or adjacency matrix. This could be distributed to all the BSPPeers. Thus all the peers would know which superstep to sync and which checkpoint to recover from on failure.

SuperstepA , 1	SuperstepB , 1	SuperstepC , 2
SuperstepD , 1	SuperstepE , 1	null
SuperstepF , 1	null	null

Figure 6 Adjacency Matrix for the job defined in Figure 5. Every entry in the matrix is a tuple of Superstep class and the supersteps count the task would take to execute.

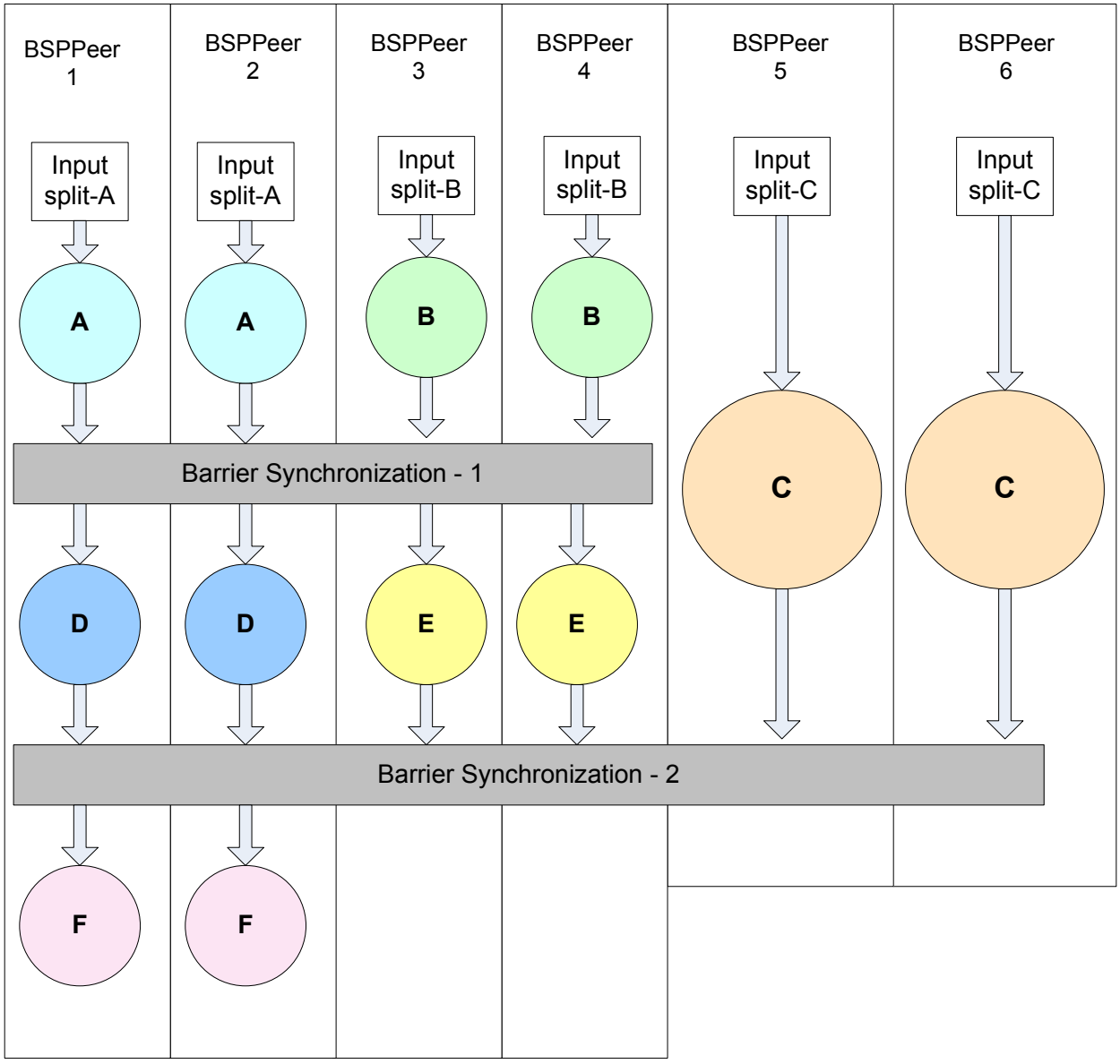

Figure 7 Task execution divided between BSPPeers

The matrix in Figure 6 contains the information for task execution. Each entry in the matrix explains what task to execute and how many superstep each takes in their computation model. Figure 7 shows how the task execution gets divided among BSP peers. Every superstep should define the number of global supersteps that task should take to execute the current BSP task. This superstep number is used while syncing with other peers.

Points to ponder:

- **Should this be implemented?**
- **If yes.**
    - **Should iterative jobs be submitted with the matrix and the total sum of supersteps to be computed. We cannot have too huge matrices.**
    - **This complies well with the nascent design that Thomas Jungblut has for fault tolerance on his github.**
    - **Check how similar this system would be to Dryad.**
    - **Effect on code**
        - **Job Submission**
        - **SyncClient.sync**
        - **BSPPeer**
        - **Send Message logic. Tasks might have to be grouped per superstep type.**
        - **Fault tolerance**
        - **Task Progress and Web UI can have information per superstep.**

-

References:

[1] Leslie G. Valiant, A bridging model for parallel computation, Communications of the ACM, Volume 33 Issue 8, Aug. 1990