MXDAG: A Hybrid Abstraction for Emerging Applications

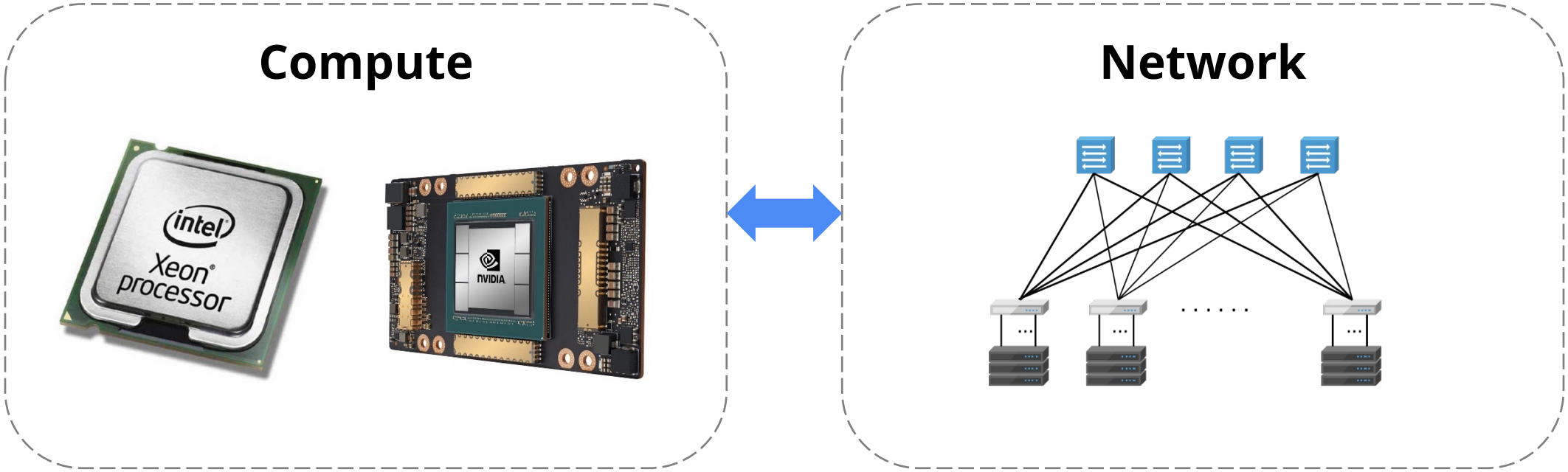
Weitao Wang, Sushovan Das, Xinyu Crystal Wu, Zhuang Wang, Ang Chen, T. S. Eugene Ng

RICE
UNCONVENTIONAL WISDOM

Trending Cloud Applications/Frameworks

Amazon
Lambda

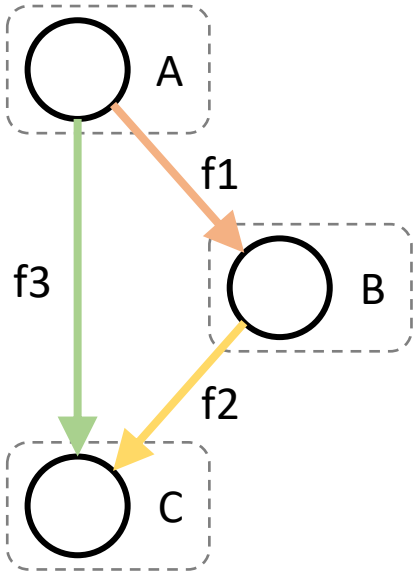
Compute and Network Resources are Both Critical

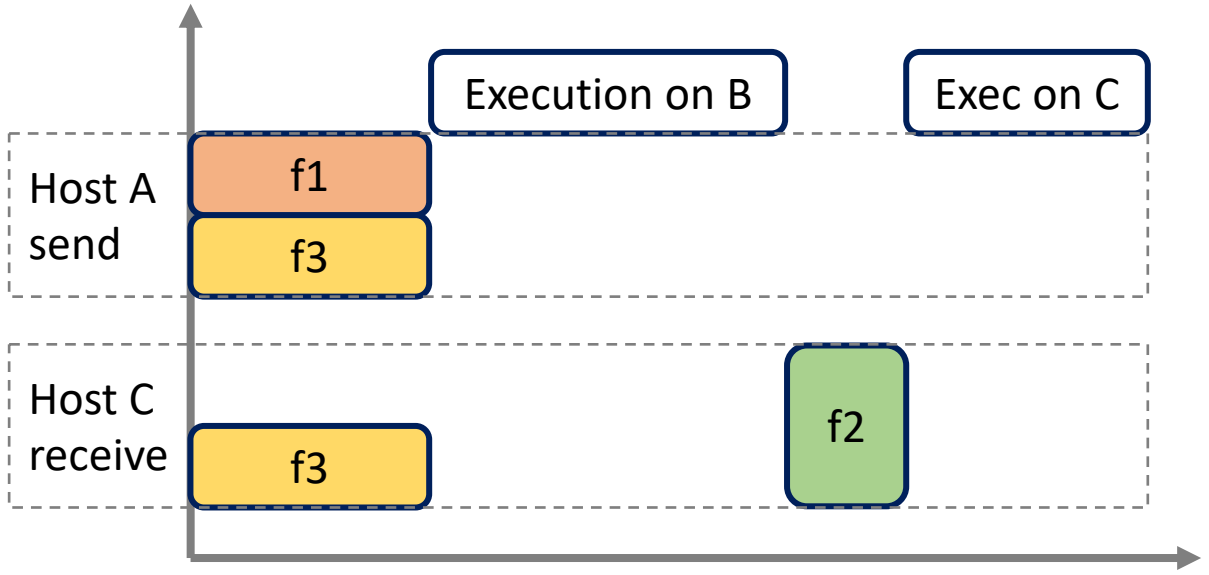


Questions:

- How to schedule the resources to improve application performance?
- How to share the resources among multiple applications?

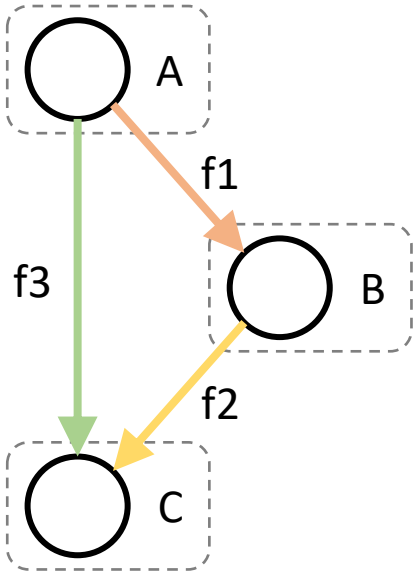
1. Previous DAGs Lack Explicit Network Co-scheduling

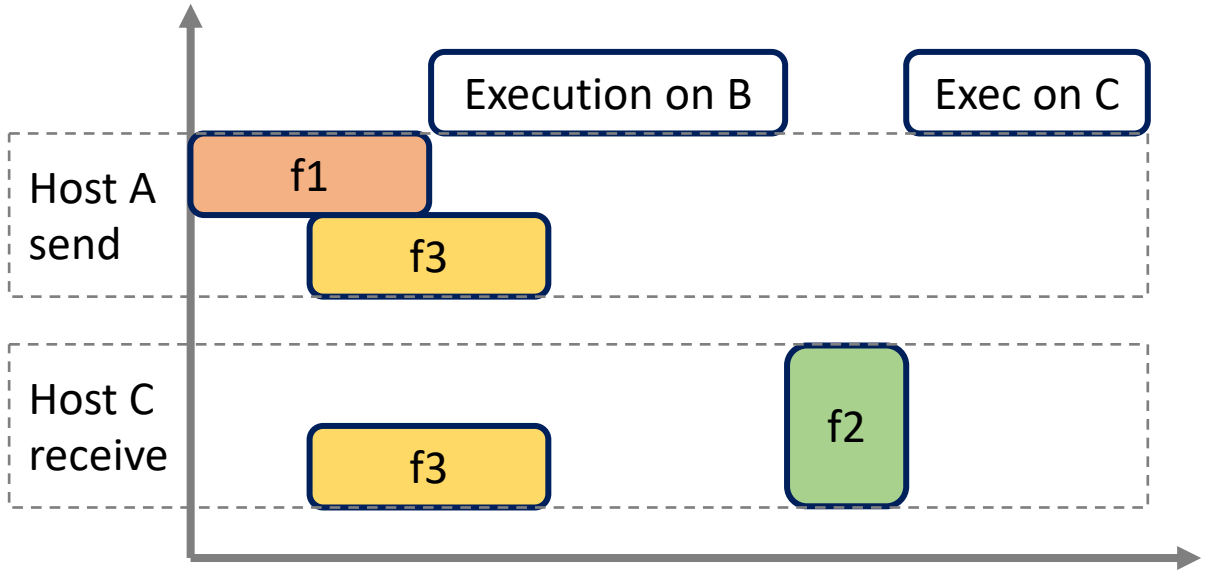




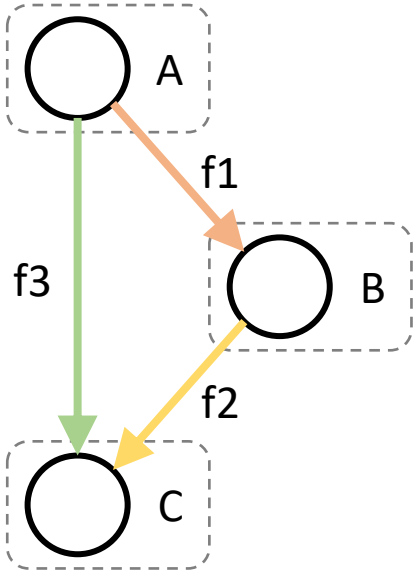
Without co-scheduling, flows 1&3 start at the same time and share the NIC bandwidth equally

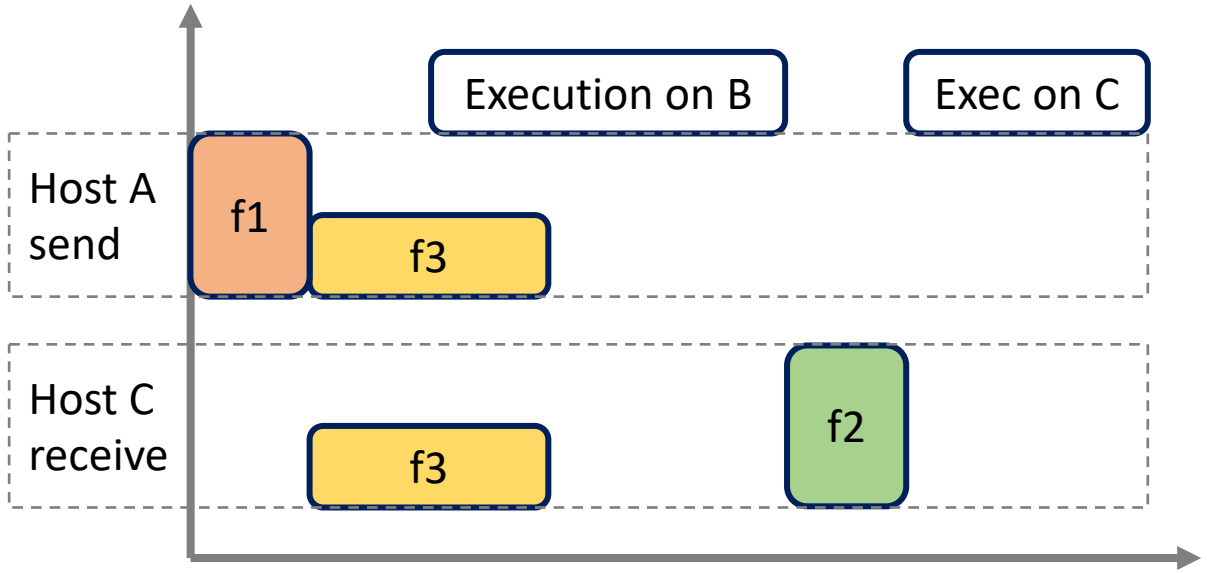
1. Previous DAGs Lack Explicit Network Co-scheduling



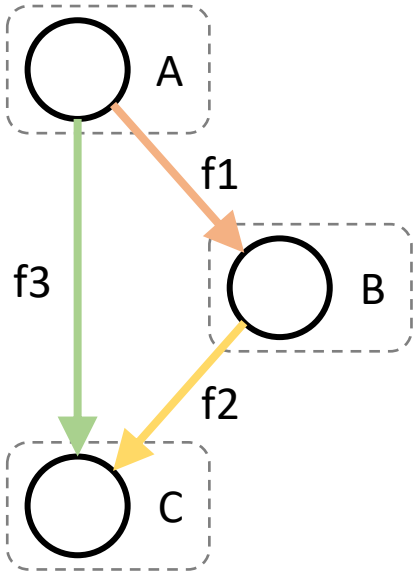


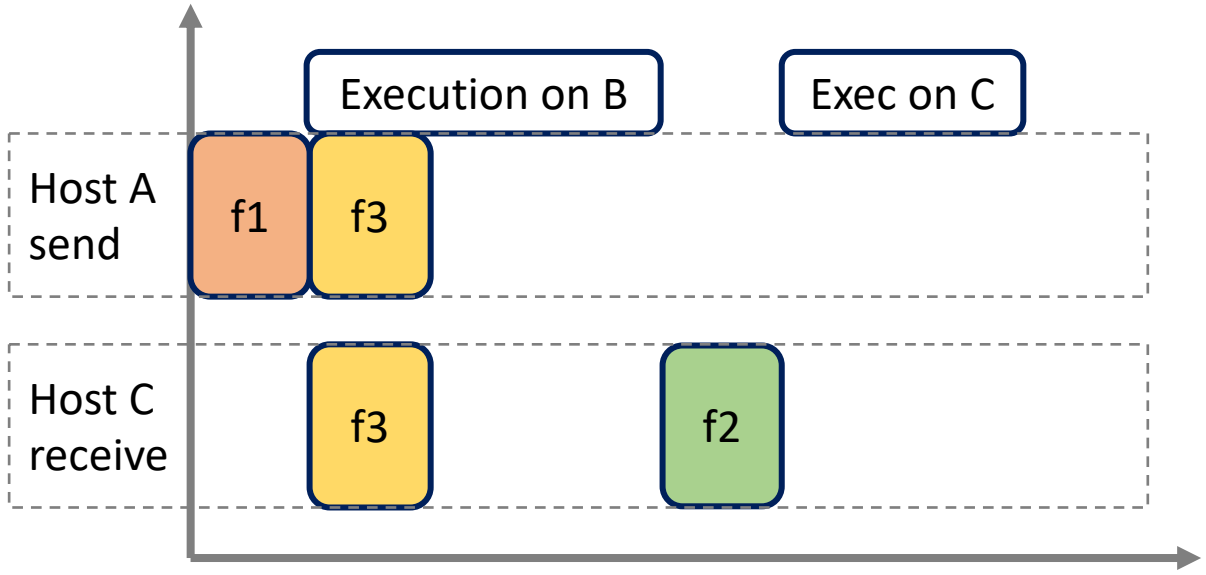
1. Previous DAGs Lack Explicit Network Co-scheduling





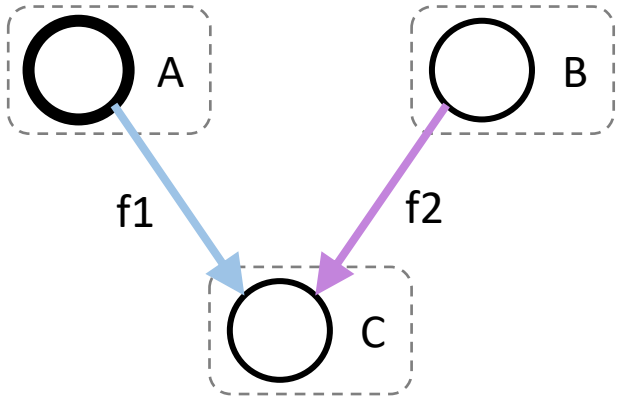
1. Previous DAGs Lack Explicit Network Co-scheduling

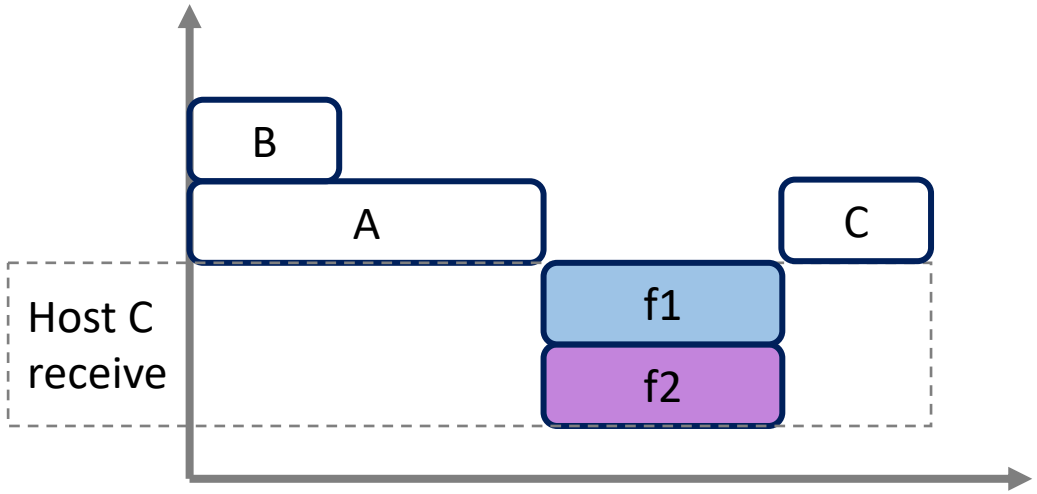




With co-scheduling, flow 1 is prioritized over flow 3 and allows task B to start earlier, reducing overall completion time

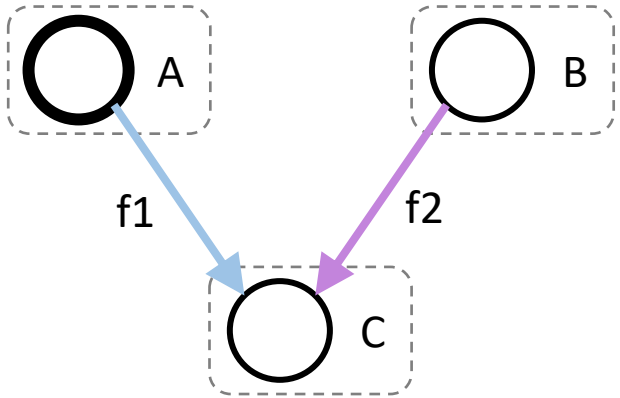
2. Coflow Abstraction Lacks Global View

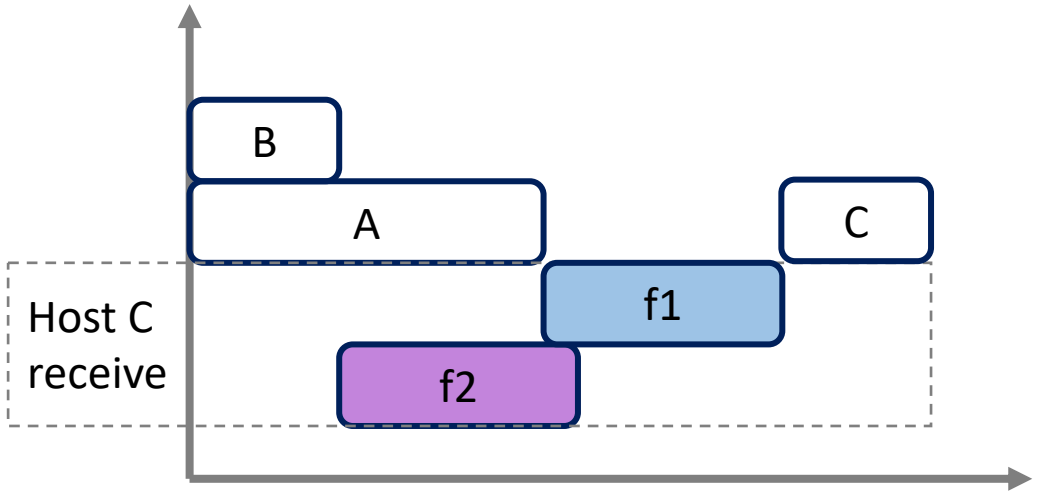




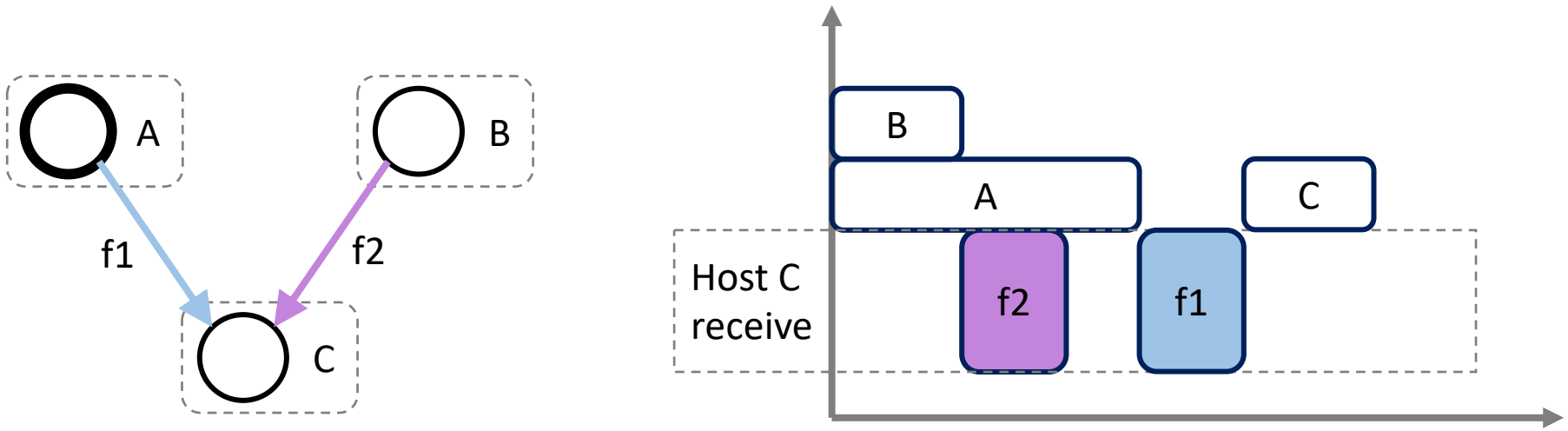
Coflow can obscure the critical path information and lead to sub-optimal performance

2. Coflow Abstraction Lacks Global View



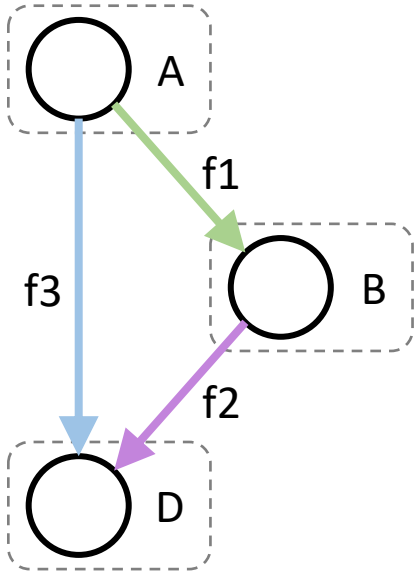


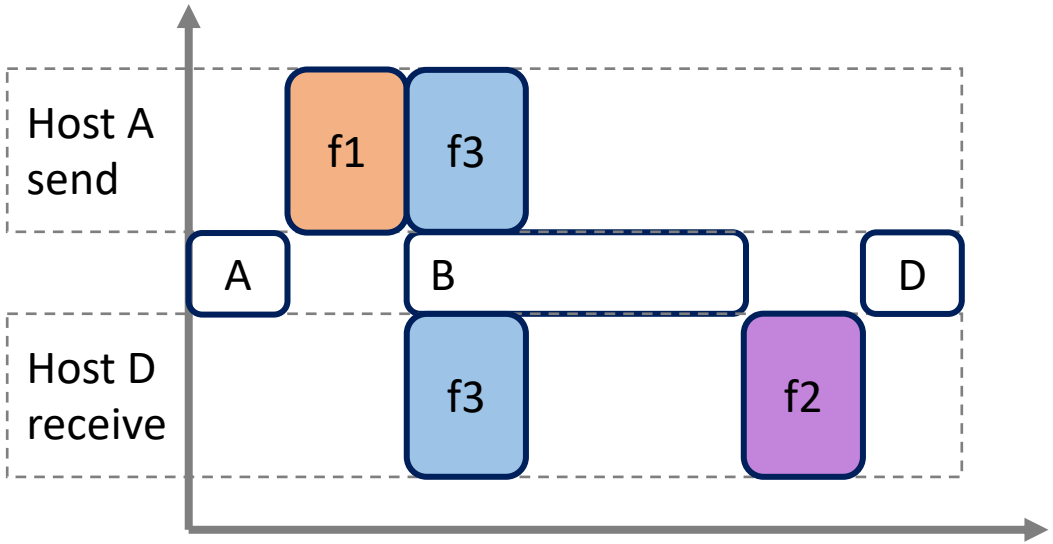
2. Coflow Abstraction Lacks Global View



Asymmetric DAG can be better scheduled without Coflow abstraction

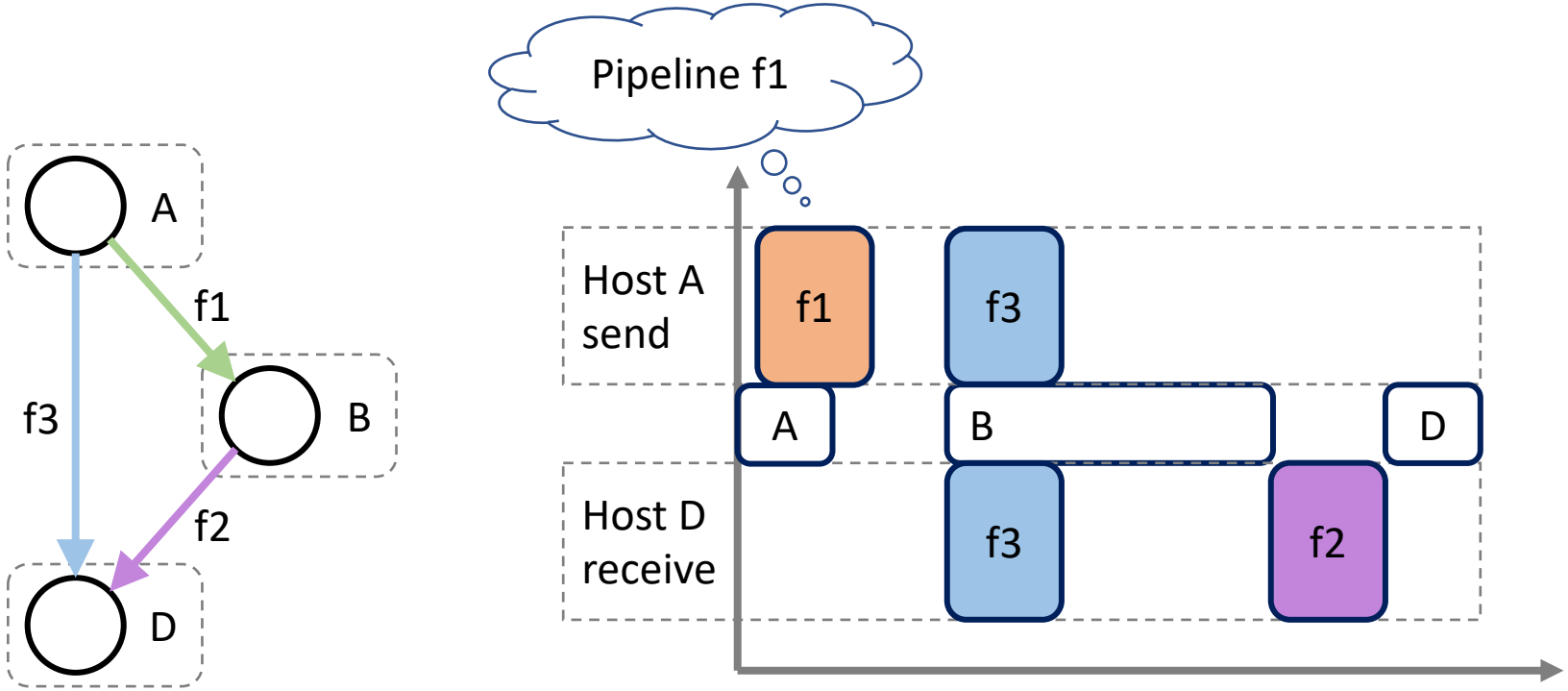
3. Both Abstractions Lack Pipelineability Analysis



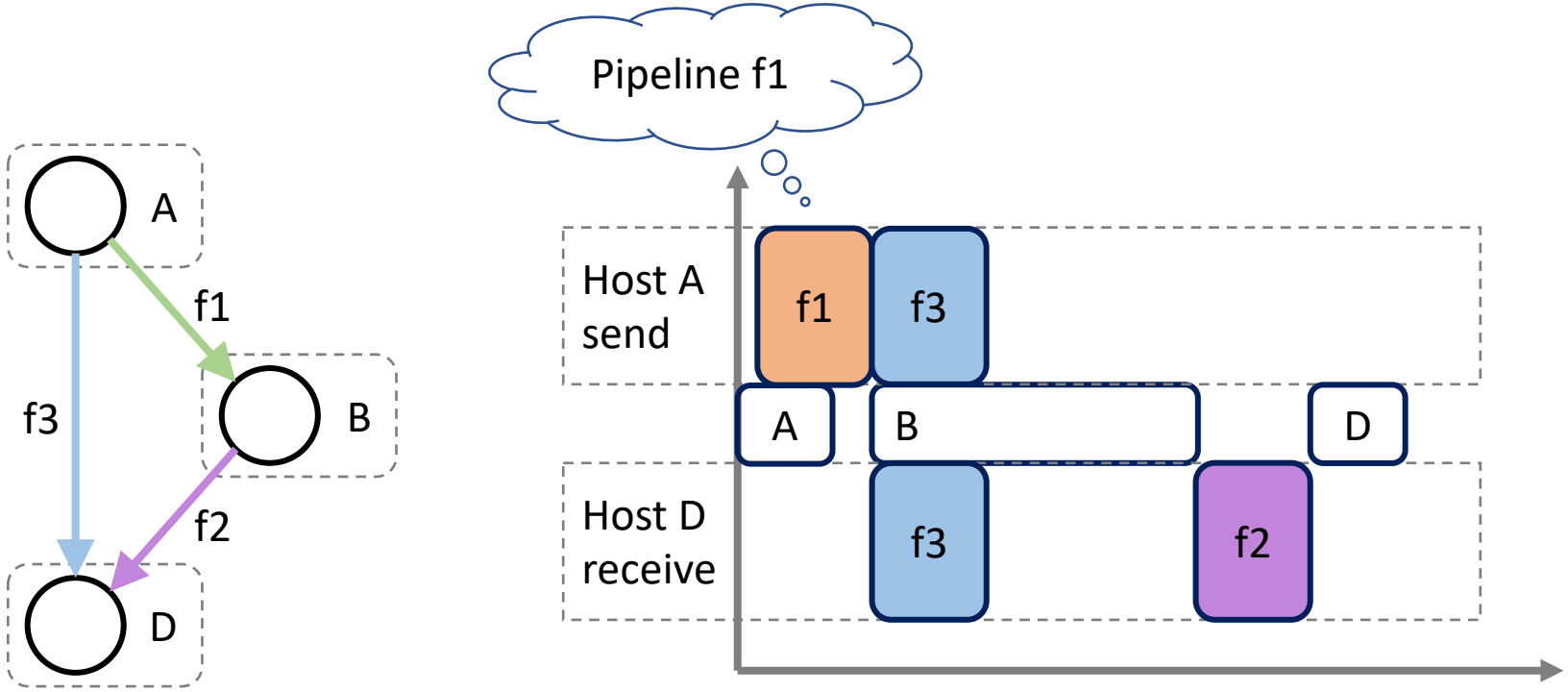


Optimal scheduling without pipeline

3. Both Abstractions Lack Pipelineability Analysis

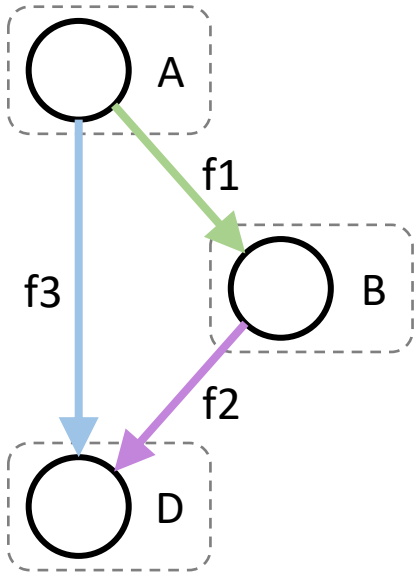


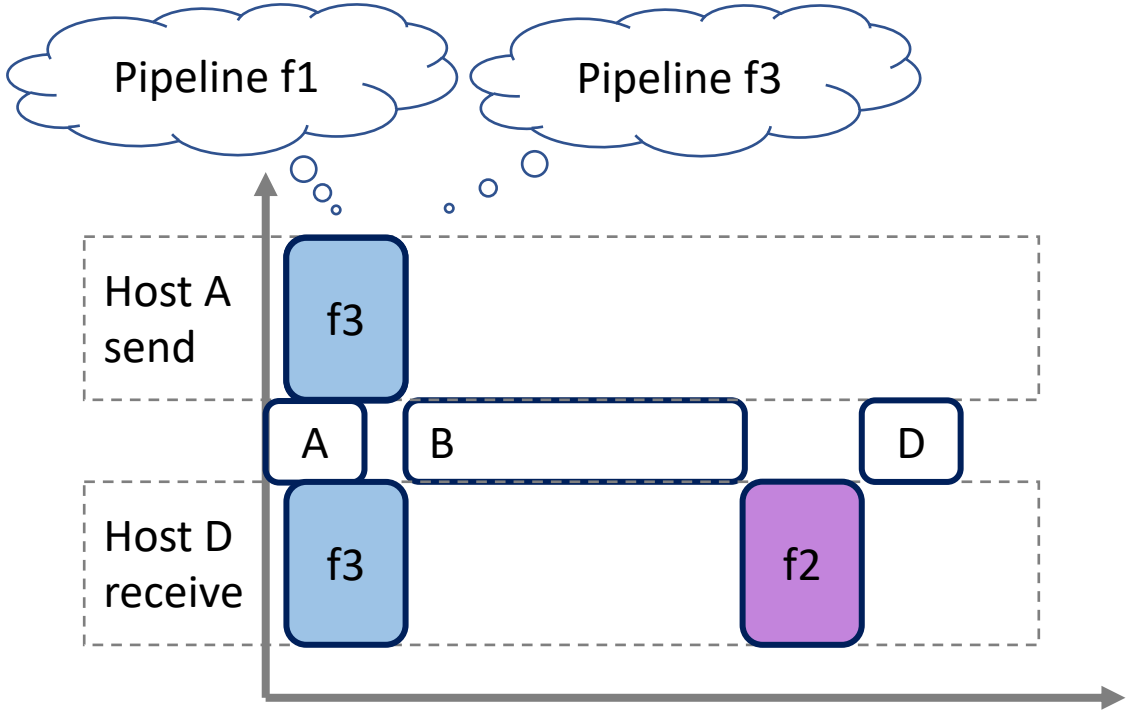
3. Both Abstractions Lack Pipelineability Analysis



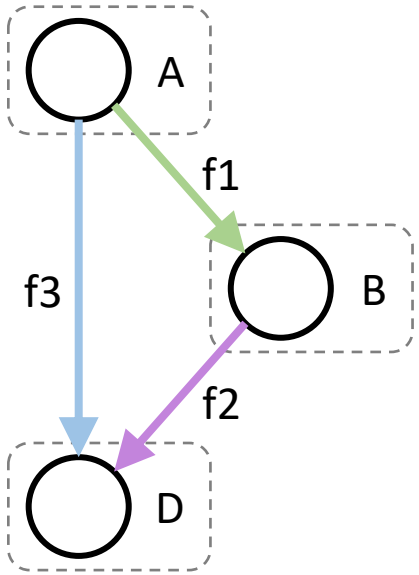
Good pipeline leads to improved performance

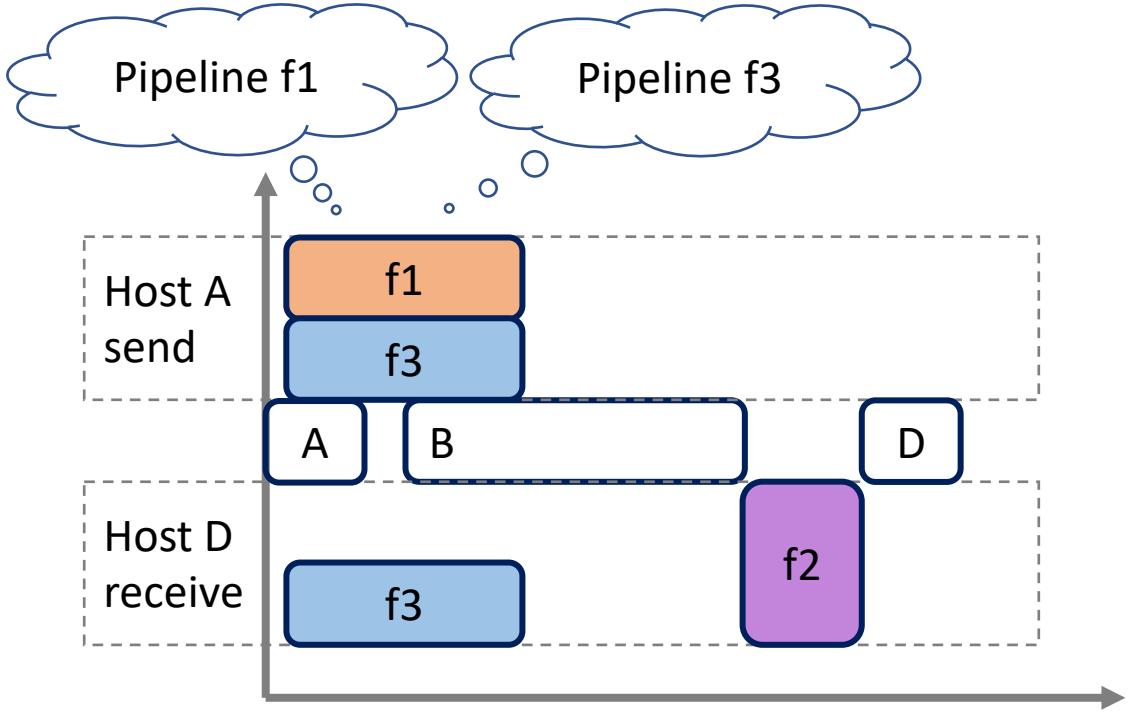
3. Both Abstractions Lack Pipelineability Analysis



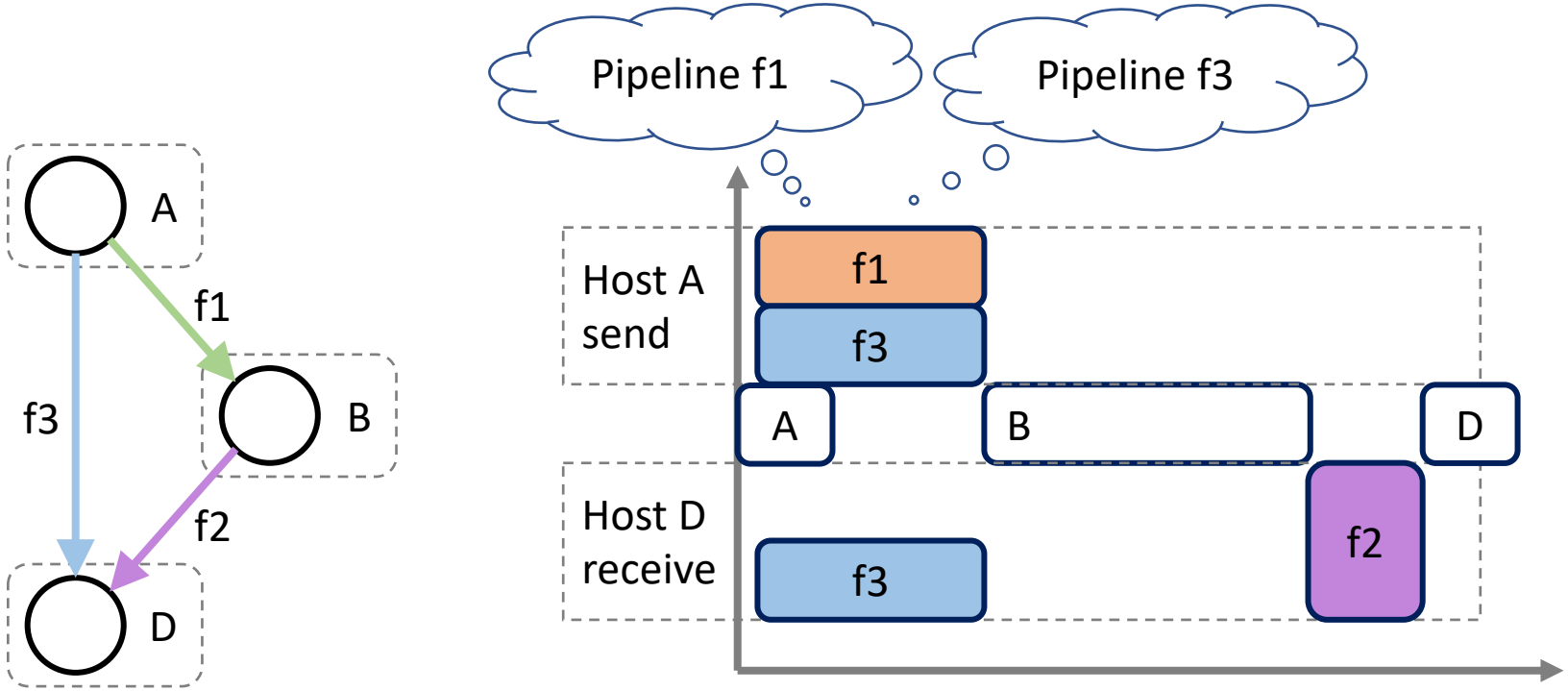


3. Both Abstractions Lack Pipelineability Analysis





3. Both Abstractions Lack Pipelineability Analysis



Inappropriate pipeline leads to performance degradation

Better Abstraction is Required for Co-scheduling

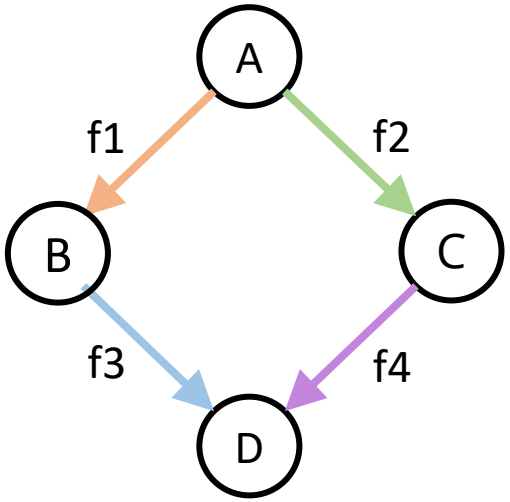
- Explicitly consider both **compute** and **network tasks** to find the **end-to-end critical path**
- Allocate the resources to optimize the performance of the critical path
- Schedule the pipeline while taking resource sharing into consideration

Benefits

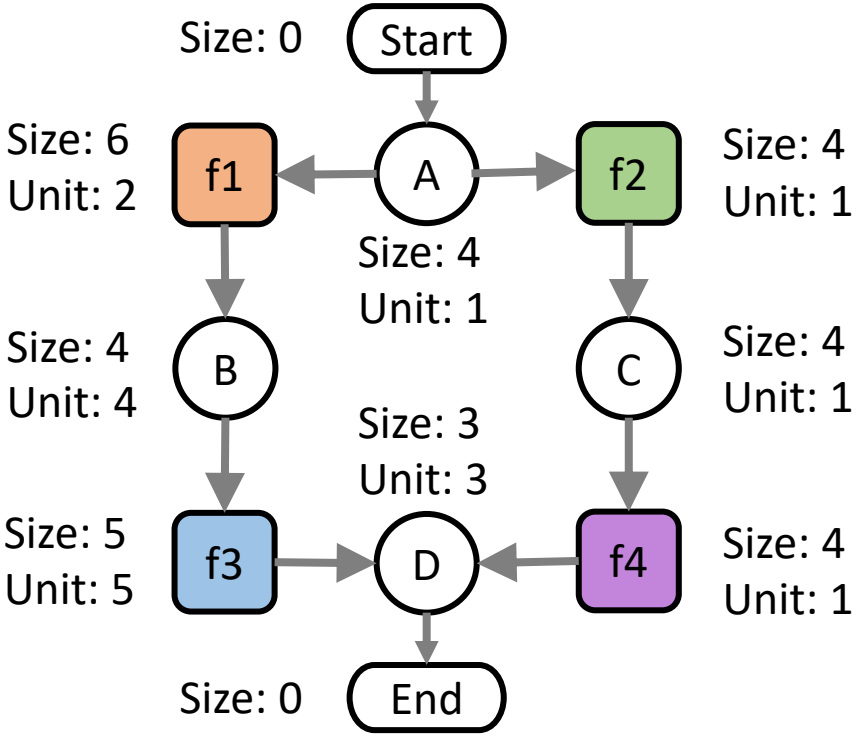
- Finer-grained view of the application
- Better critical path analysis
- Improve application performance and resource utilization

MXDAG: A Compute-Network Hybrid Abstraction

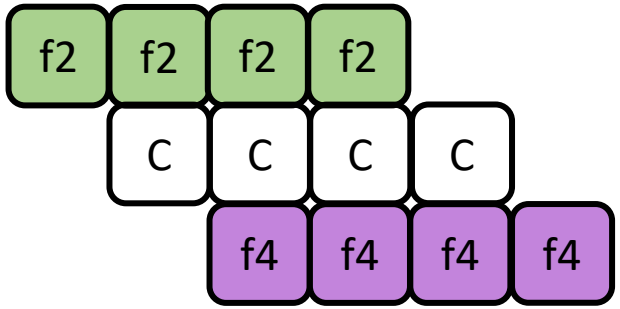
- **Nodes:** Both Compute and Network tasks (MXTasks)
- **Edges:** Dependencies between MXTasks
- **Size:** Ideal completion time of an individual MXTask
- **Unit:** Smallest unit size under pipelining



Traditional DAG



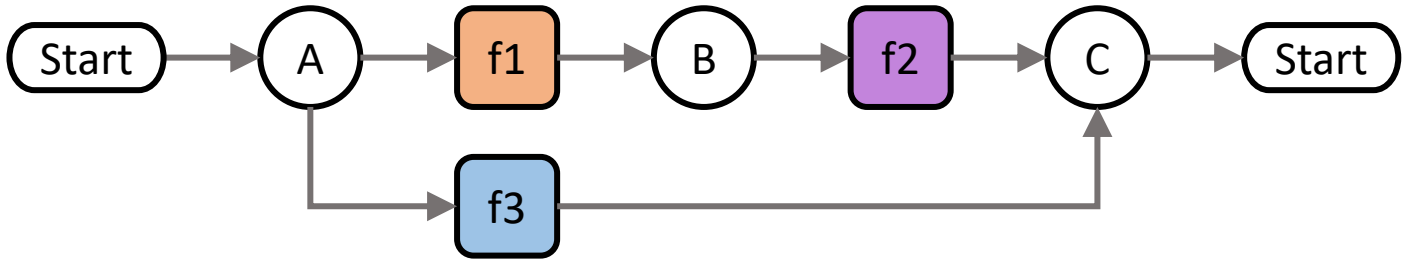
MXDAG

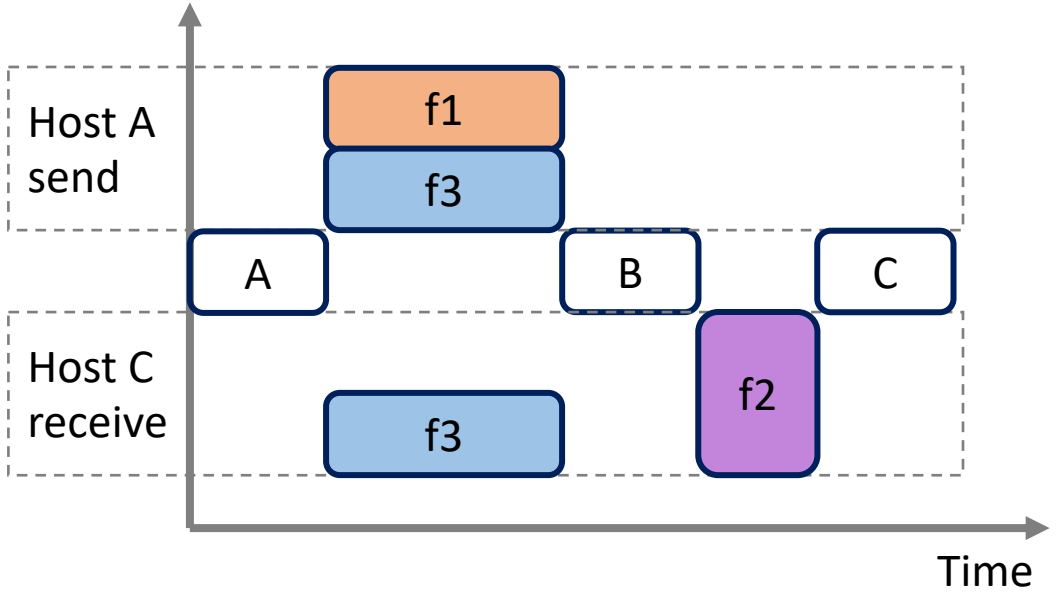


Pipelineability

Schedule A Single MXDAG

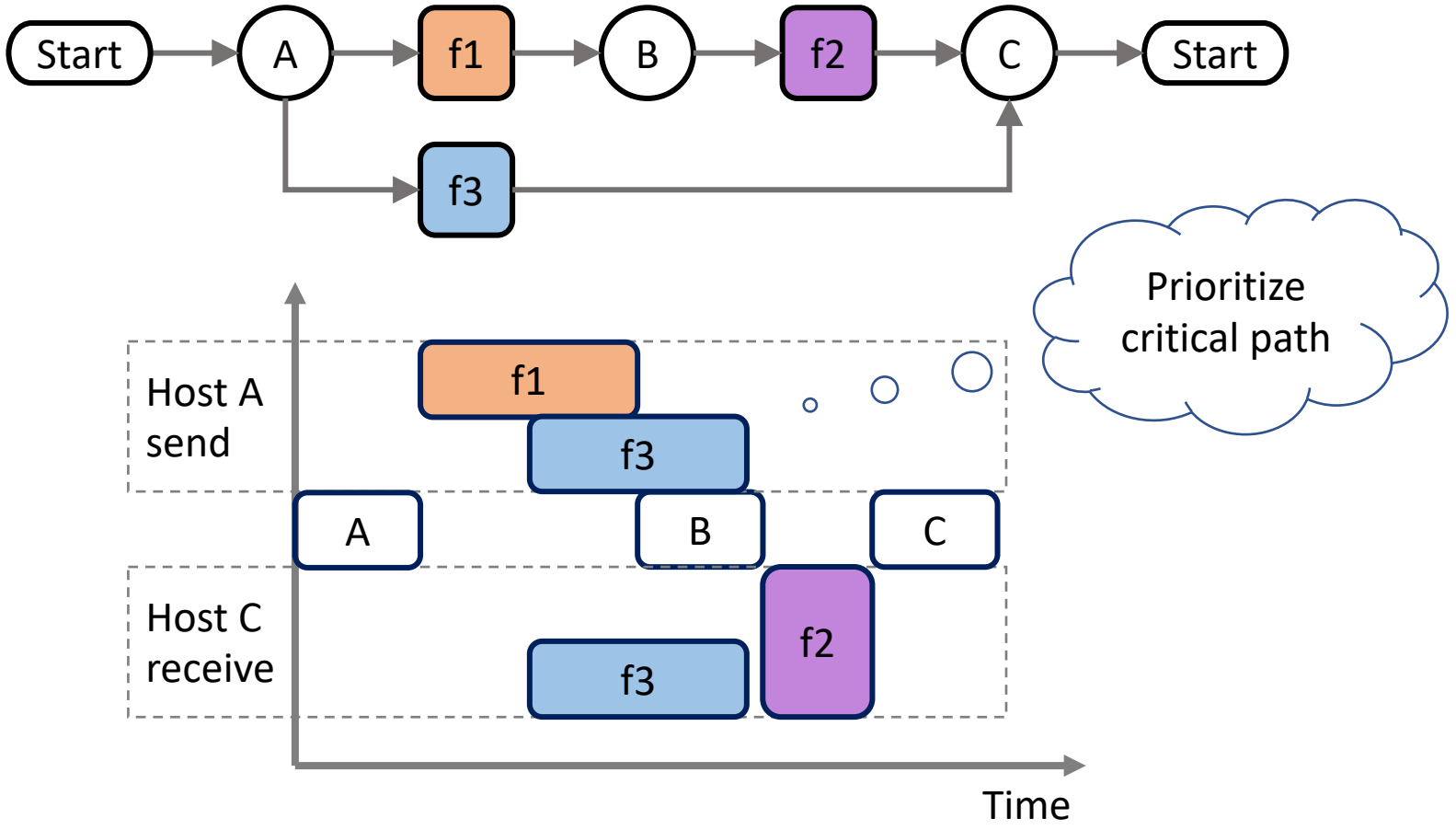
Key Principle: **Prioritize** the **end-to-end critical path** since it dominates the completion time.





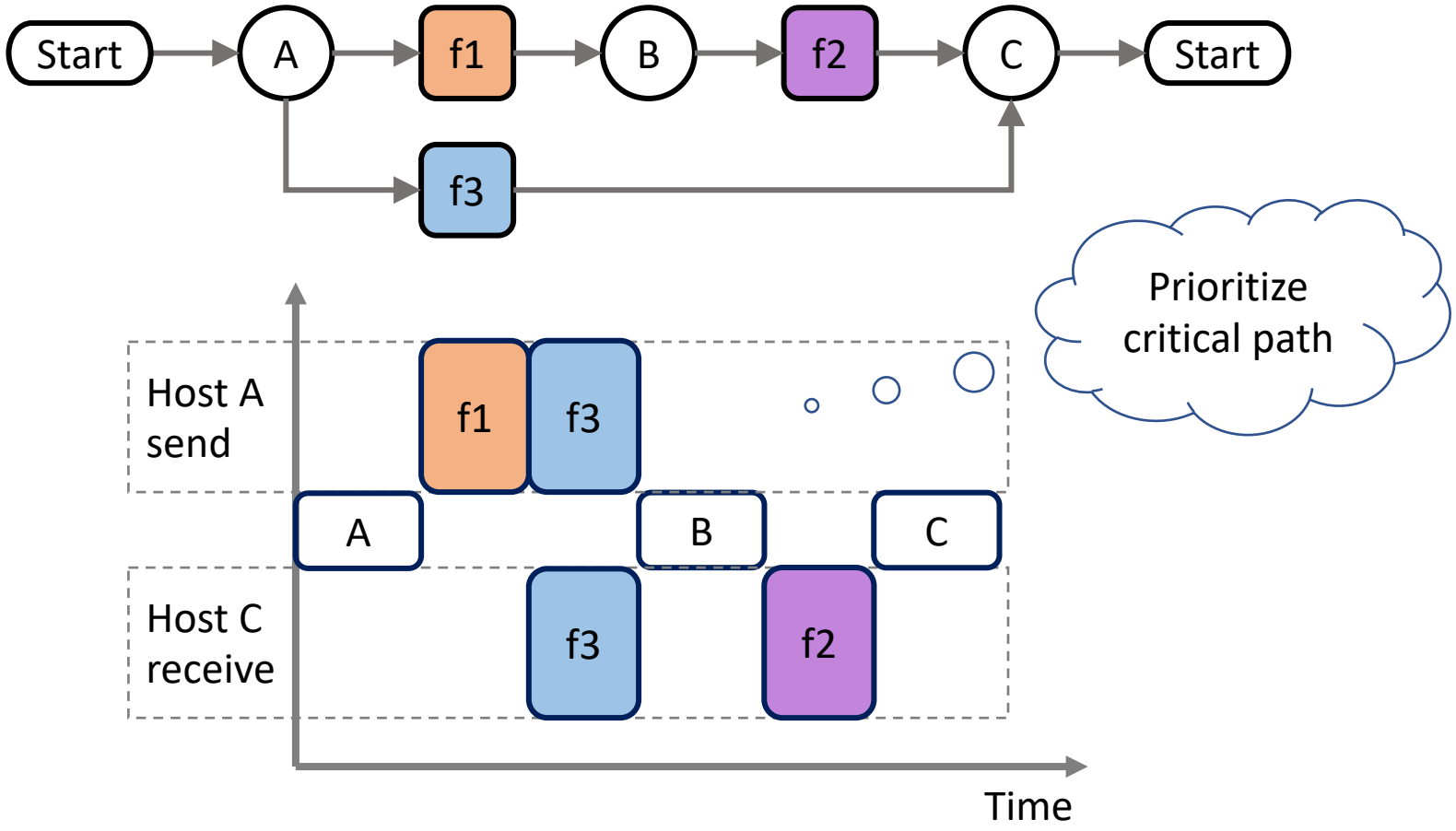
Schedule A Single MXDAG

Key Principle: **Prioritize** the **end-to-end critical path** since it dominates the completion time.



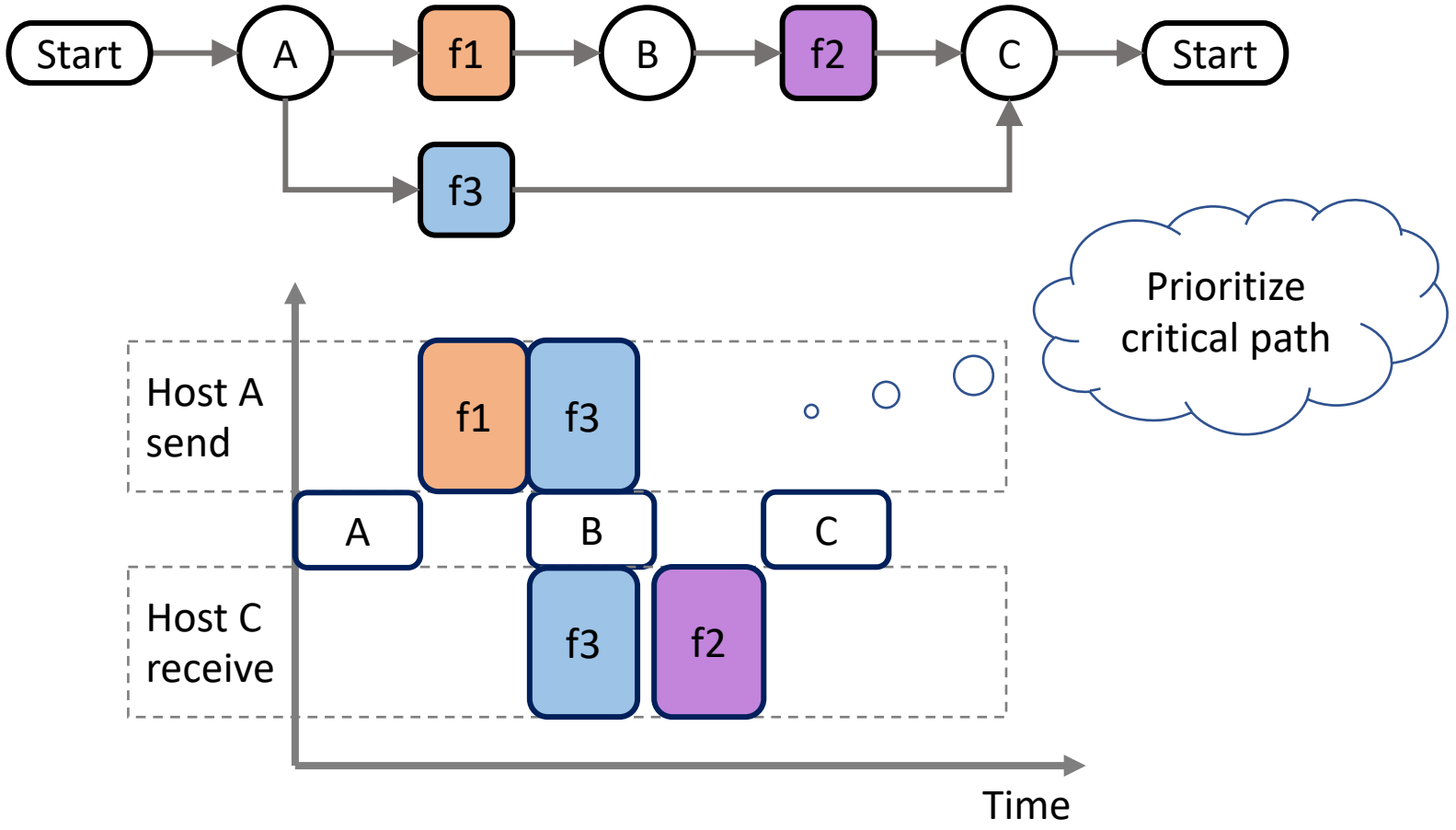
Schedule A Single MXDAG

Key Principle: **Prioritize** the **end-to-end critical path** since it dominates the completion time.



Schedule A Single MXDAG

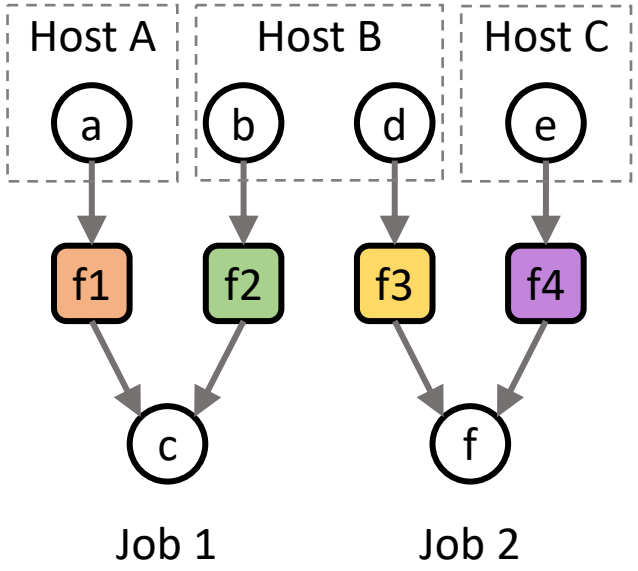
Key Principle: **Prioritize** the **end-to-end critical path** since it dominates the completion time.

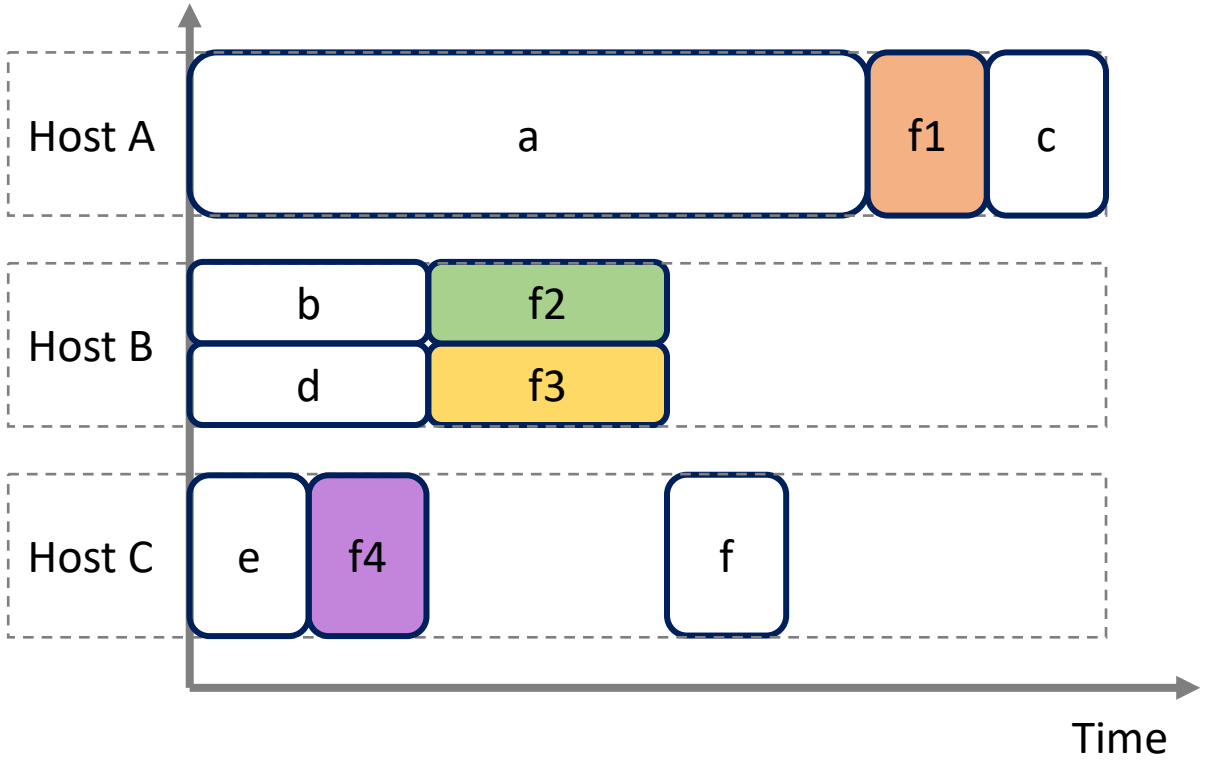


Schedule Multiple MXDAGs

Key Principle: **be altruistic,** namely, try to benefit others without hurting its own performance

Method: **delay the execution of the non-critical path** until necessary

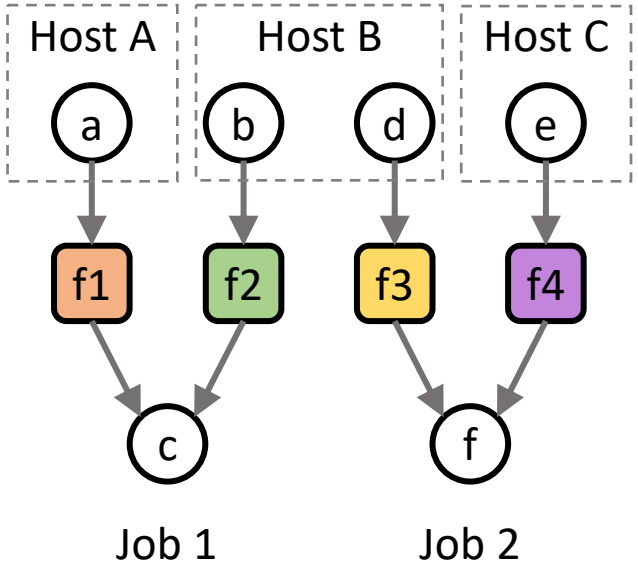


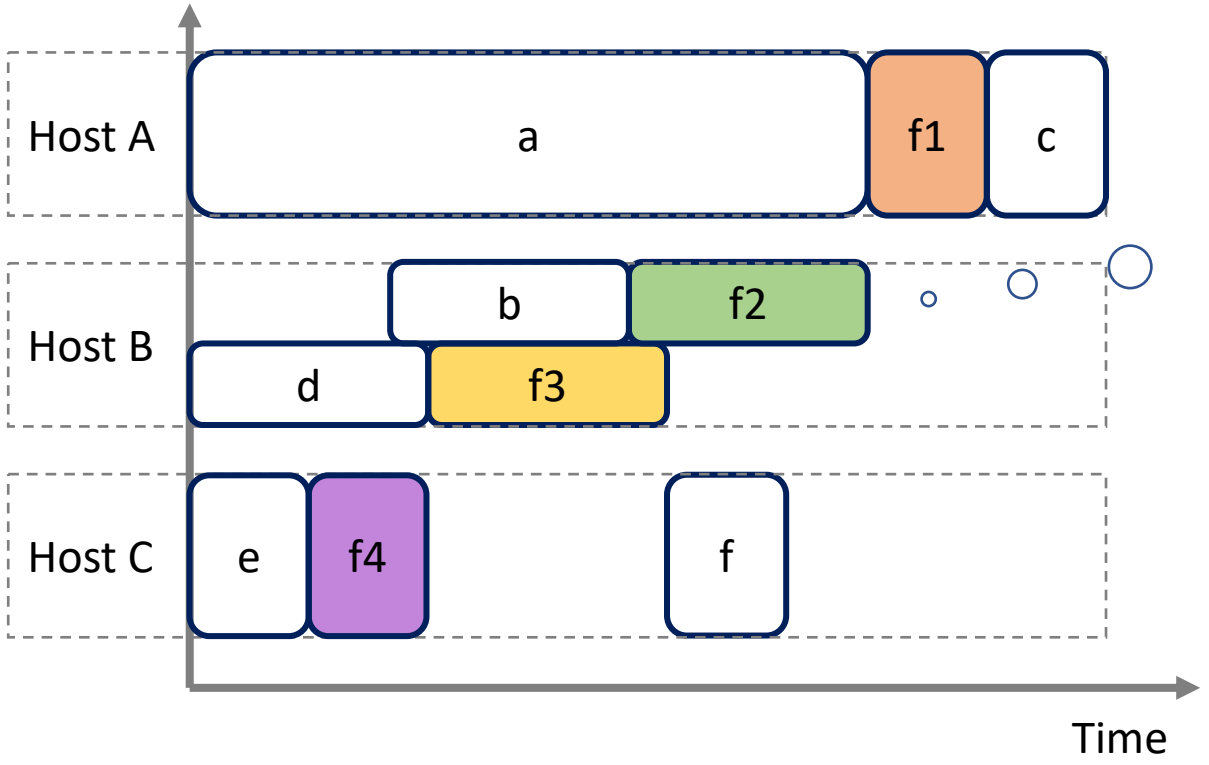


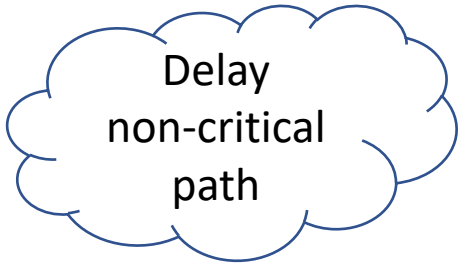
Schedule Multiple MXDAGs

Key Principle: **be altruistic,** namely, try to benefit others without hurting its own performance

Method: **delay the execution of the non-critical path** until necessary



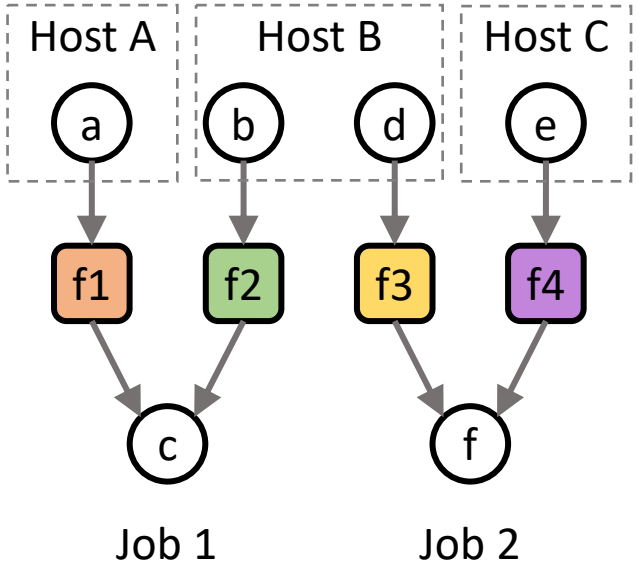


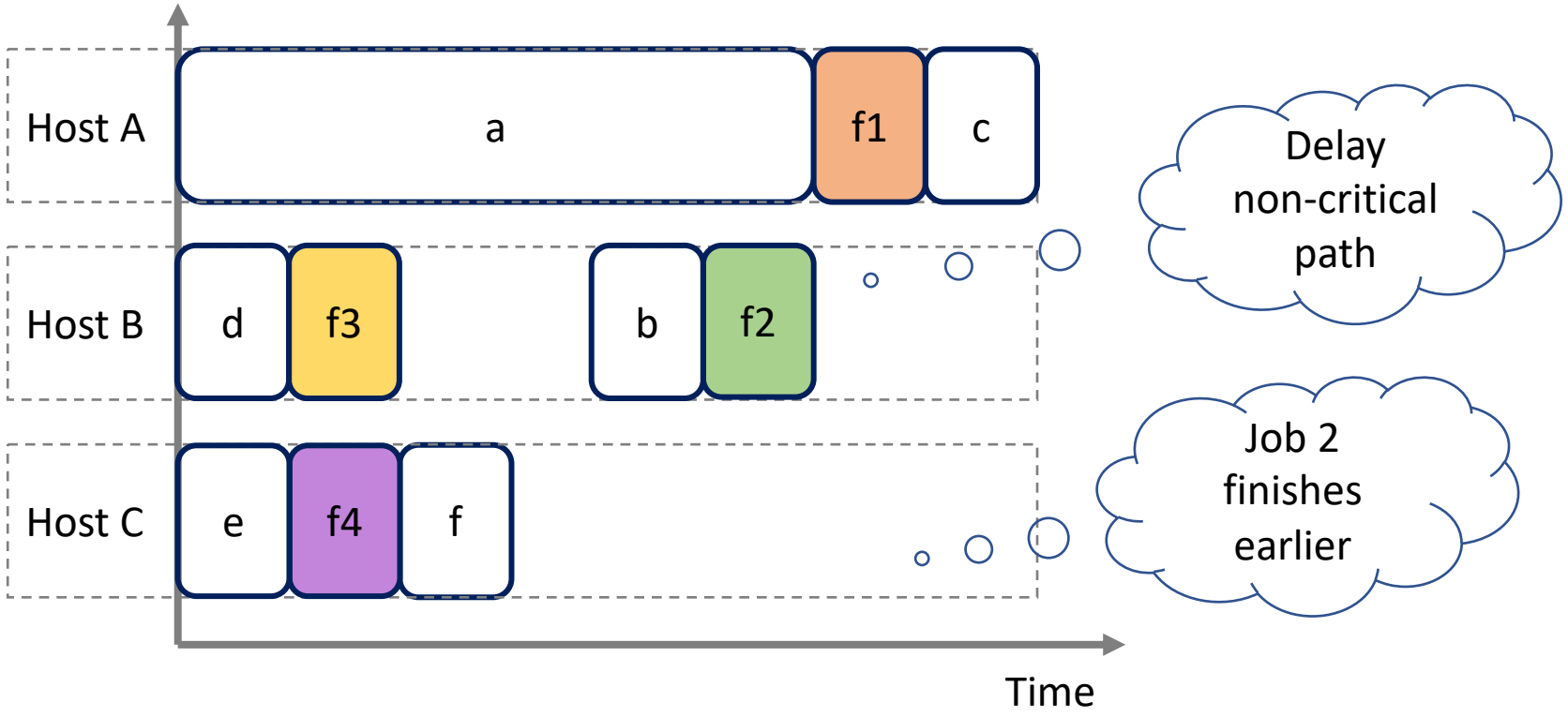


Schedule Multiple MXDAGs

Key Principle: **be altruistic,** namely, try to benefit others without hurting its own performance

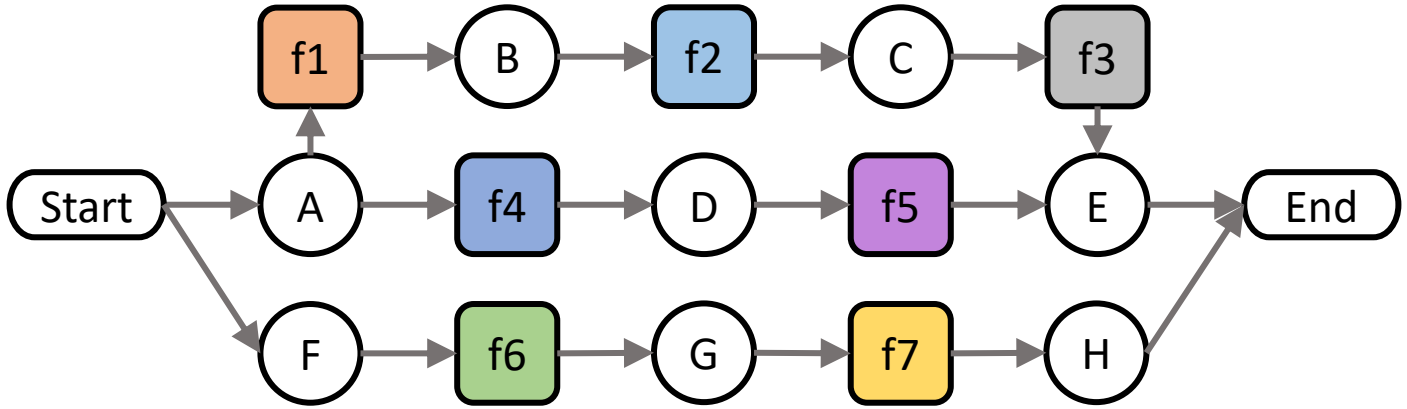
Method: **delay the execution of the non-critical path** until necessary





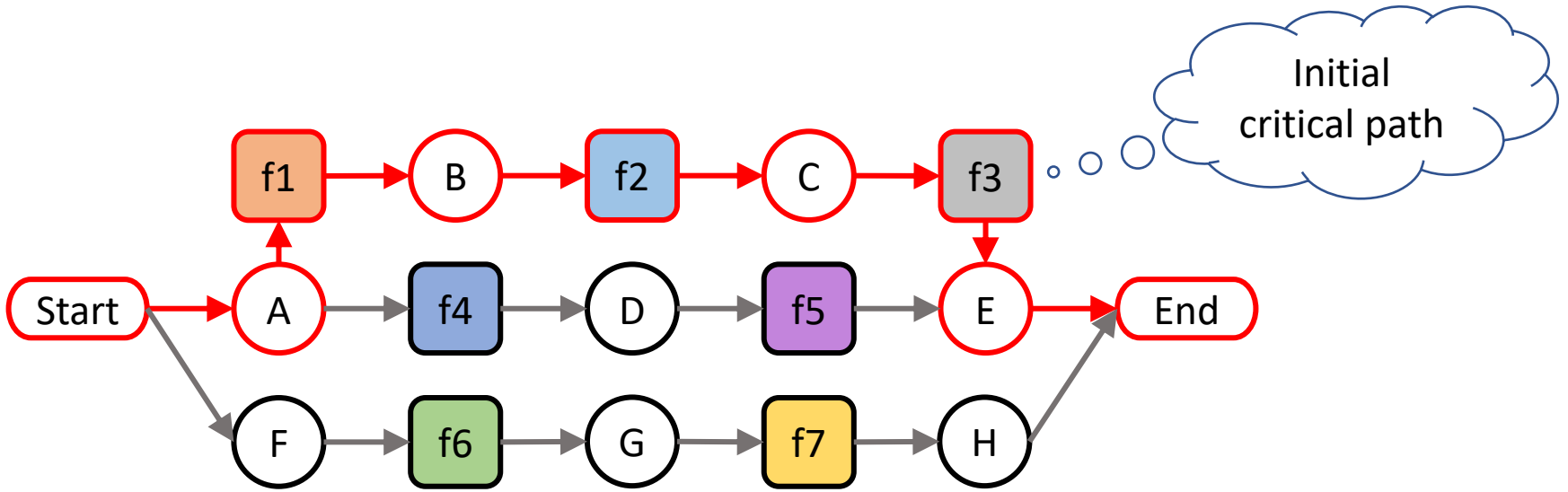
Other Usages of MXDAG

Runtime close-loop control:



Other Usages of MXDAG

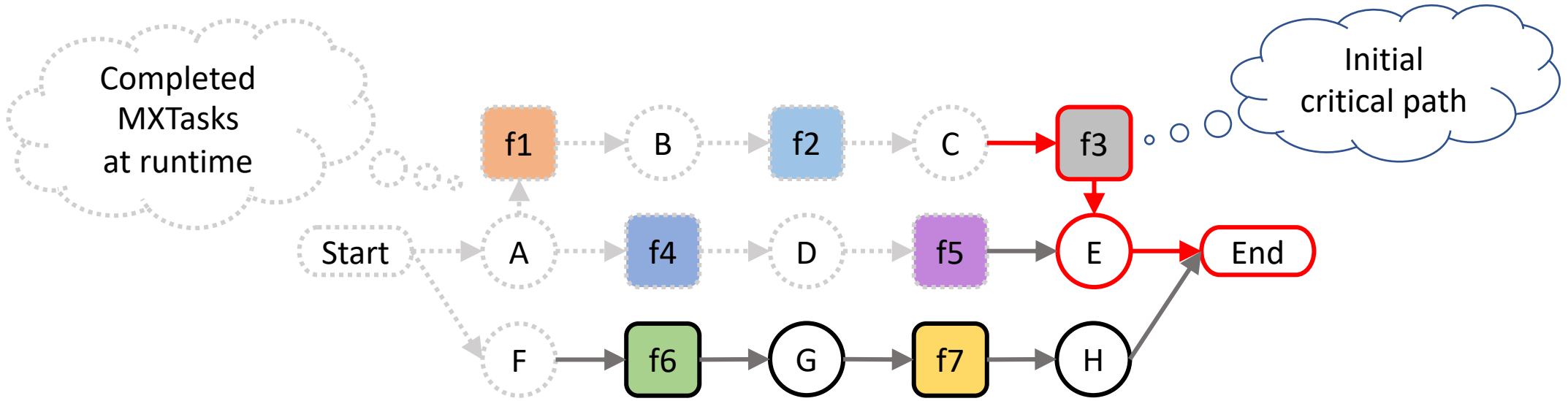
Runtime close-loop control:



Other Usages of MXDAG

Runtime close-loop control:

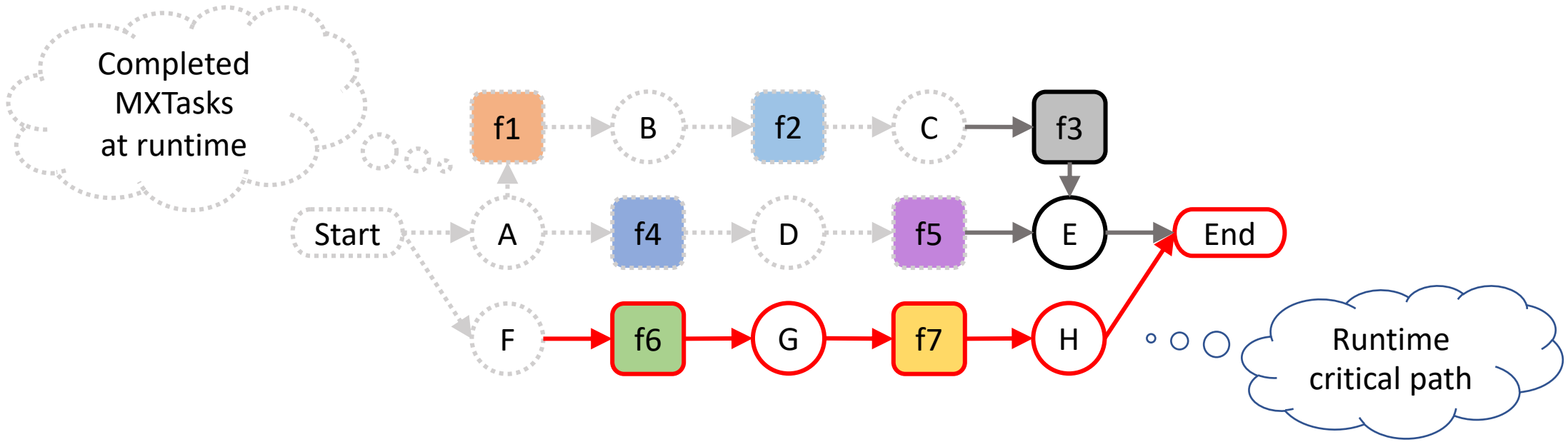
1. Identify the network or compute **stragglers at runtime** with monitoring;



Other Usages of MXDAG

Runtime close-loop control:

1. Identify the network or compute **stragglers at runtime** with monitoring;
2. **Reschedule** the resources depending on the **runtime critical path**.



Conclusion

- MXDAG provides the first comprehensive abstraction for emerging applications
  - Treat compute and network tasks equally as MXTasks
  - Introduce pipelineability in abstraction and make the pipelines runtime reconfigurable
- Provide principles for the compute-network co-scheduler
  - Prioritize the **critical path** within a single MXDAG
  - Be **altruistic** for multiple MXDAGs
- Other usages:
  - Monitor the application's path-wise progress at runtime
  - Mitigate the global stragglers among the application's tasks
  - …

Q & A!

Schedule Multiple MXDAGs

Key Principle: **be altruistic,** namely, try to benefit others without hurting its own performance

Method: **delay the execution of the non-critical path** until necessary