EECS 583 – Class 20
Research Topic 2: Stream Compilation,
Stream Graph Modulo Scheduling

University of Michigan

November 30, 2011

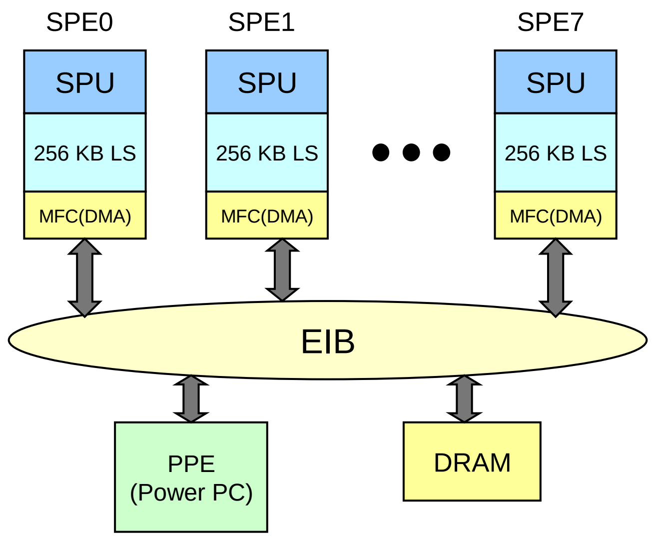
Guest Speaker Today: Daya Khudia

Announcements & Reading Material

- This class
  - "Orchestrating the Execution of Stream Programs on Multicore Platforms," M. Kudlur and S. Mahlke, Proc. ACM SIGPLAN 2008 Conference on Programming Language Design and Implementation, Jun. 2008.
- Next class – GPU compilation
  - "Program optimization space pruning for a multithreaded GPU," S. Ryoo, C. Rodrigues, S. Stone, S. Baghsorkhi, S. Ueng, J. Straton, and W. Hwu, Proc. Intl. Sym. on Code Generation and Optimization, Mar. 2008.

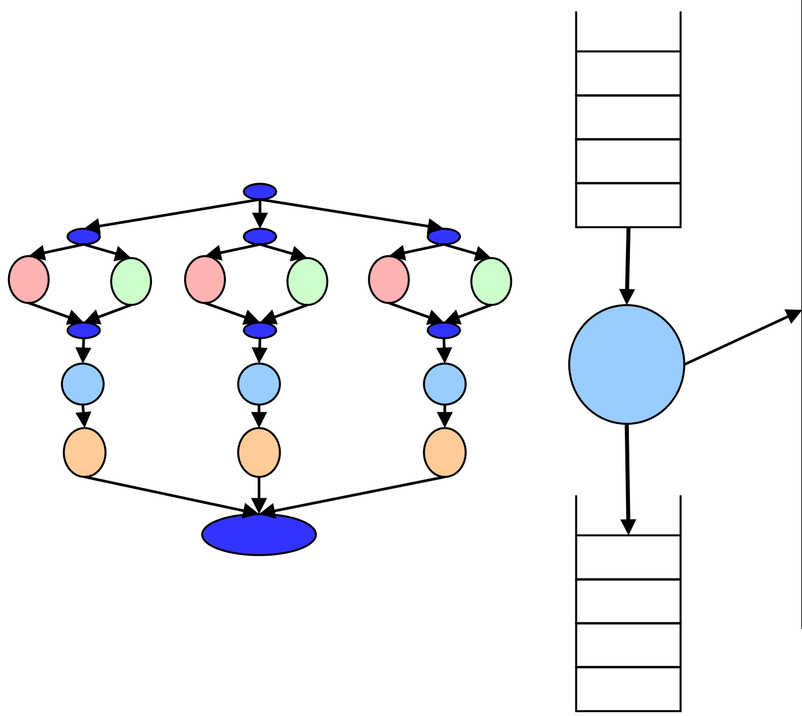
Stream Graph Modulo Scheduling (SGMS)

- Coarse grain software pipelining
  - Equal work distribution
  - Communication/computation overlap
  - Synchronization costs
- Target : Cell processor
  - Cores with disjoint address spaces
  - Explicit copy to access remote data
    - DMA engine independent of PEs
- Filters = operations, cores = function units

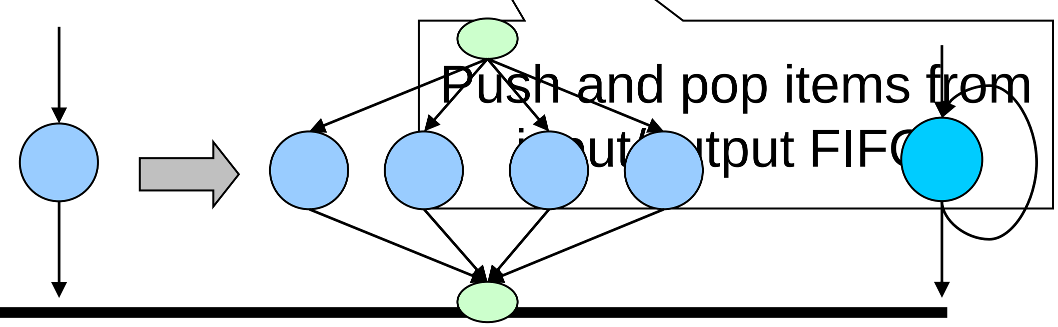


Preliminaries

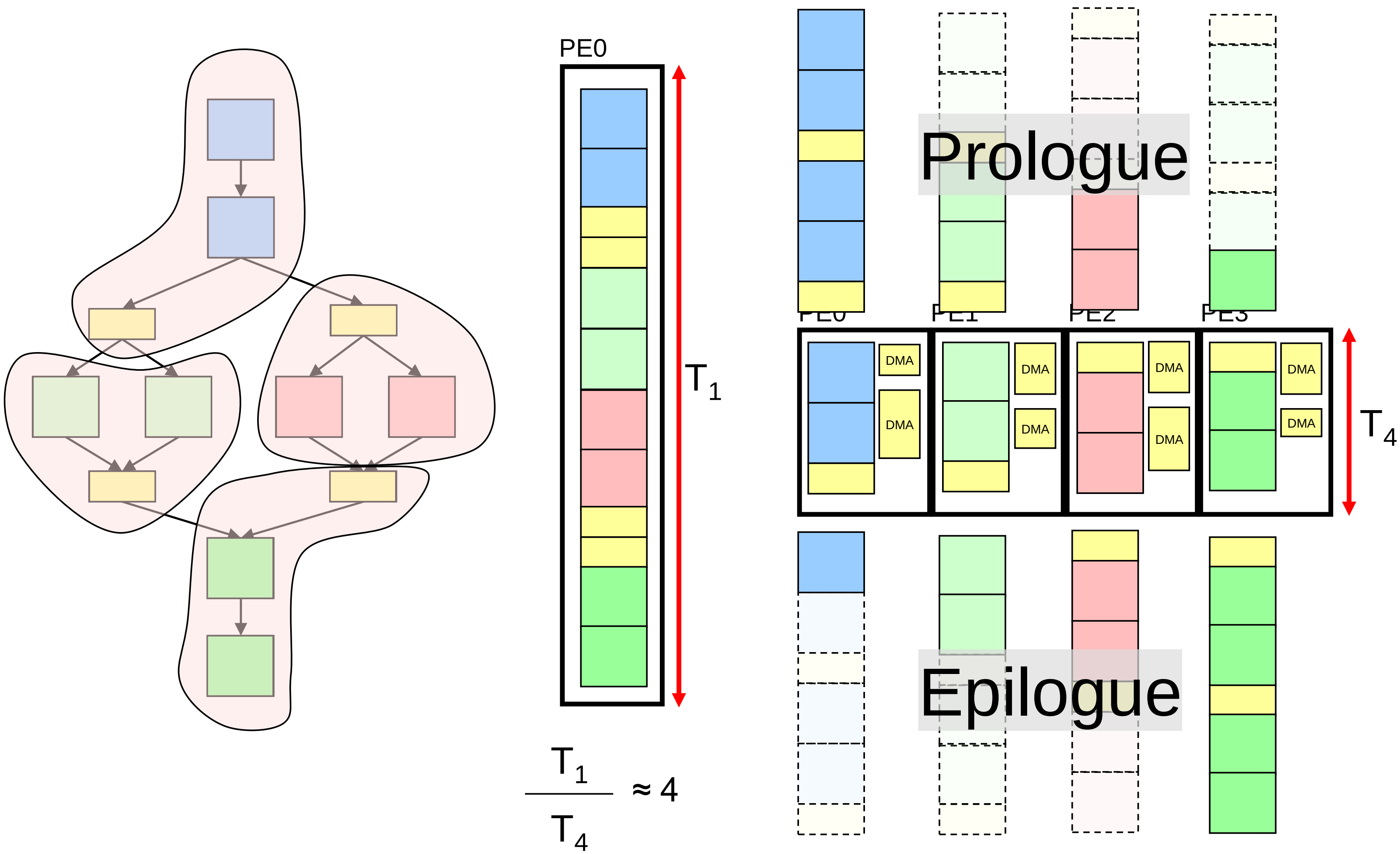
- Synchronous Data Flow (SDF) [Lee ’87]
- StreamIt [Thies ’02]



```
int->int filter FIR(int N, int wgts[N]) {
  int wgts[N];
  work pop 1 push 1 {
    int i, sum = 0;
    wgts = adapt(wgts);
    for(i=0; i<N; i++)
      sum += peek(i)*wgts[i];
    push(sum);
    pop();
  }
}
```

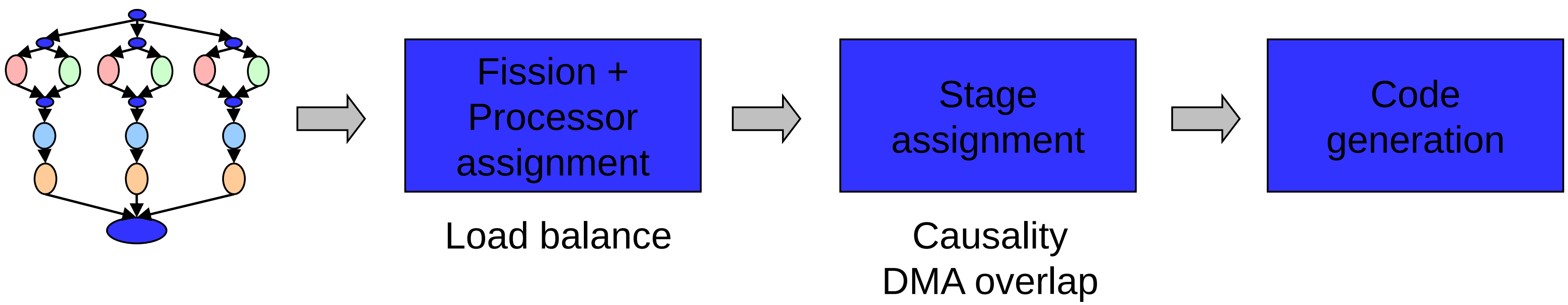


SGMS Overview



$$\frac{T_1}{T_4} \approx 4$$

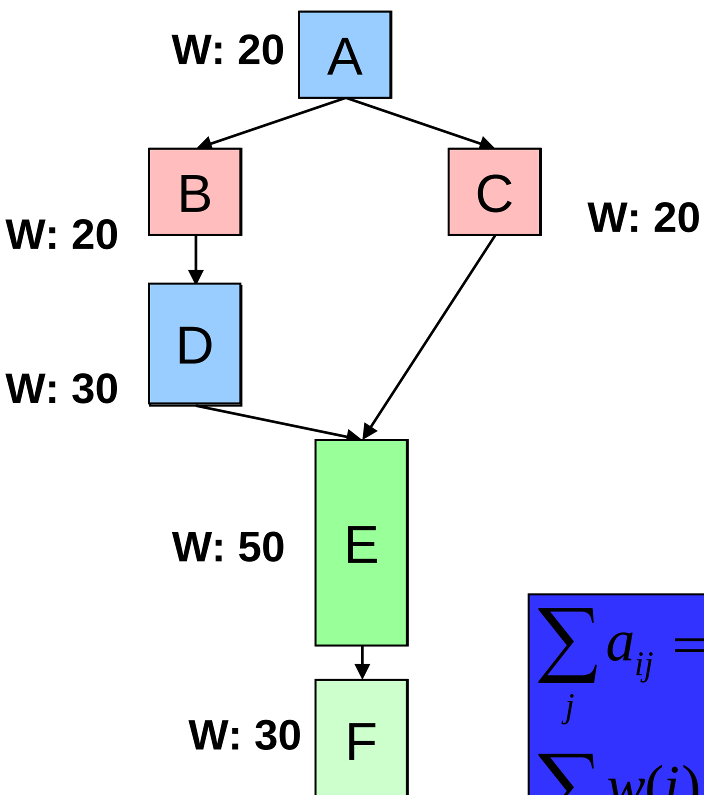
SGMS Phases


Fission +
Processor
assignment
Load balance
Stage
assignment
Causality
DMA overlap
Code
generation

Processor Assignment: Maximizing Throughputs

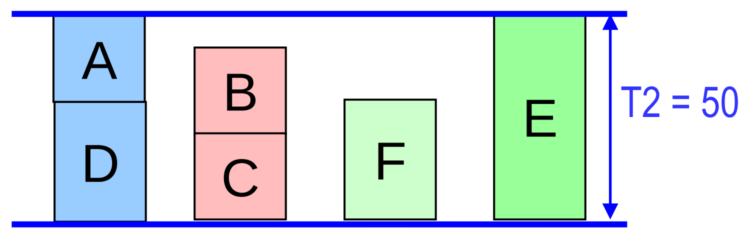
- Assigns each filter to a processor

W: workload

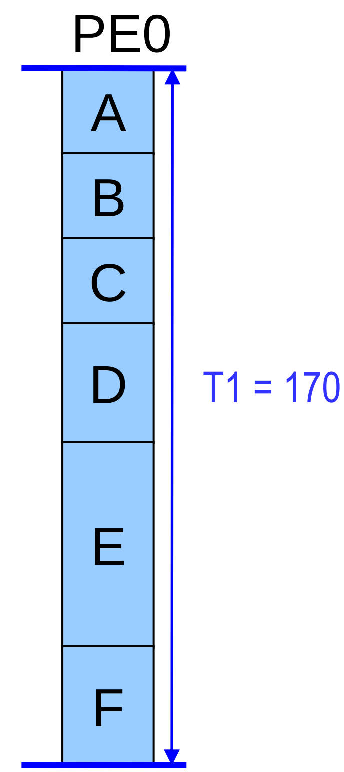


PE0 PE1 PE2 PE3

Four Processing Elements



Minimum II: 50
Balanced workload!
Maximum throughput



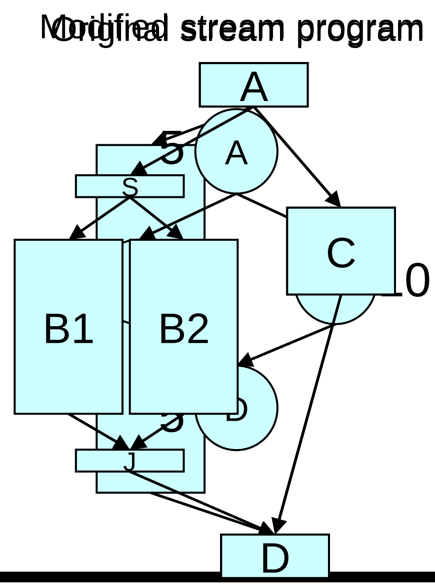
T1/T2 = 3. 4

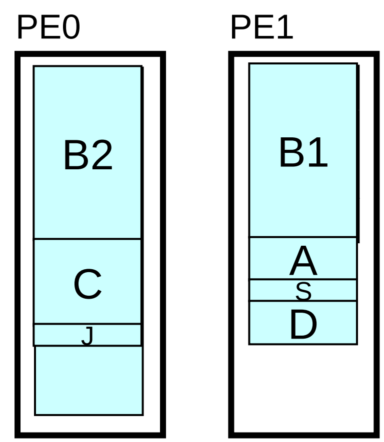
$$\sum_j a_{ij} = 1$$

$$\sum_i w(i) \cdot a_{ij} \le II$$

Need More Than Just Processor Assignment

- Assign filters to processors
  - Goal : Equal work distribution
- Graph partitioning?
- Bin packing?

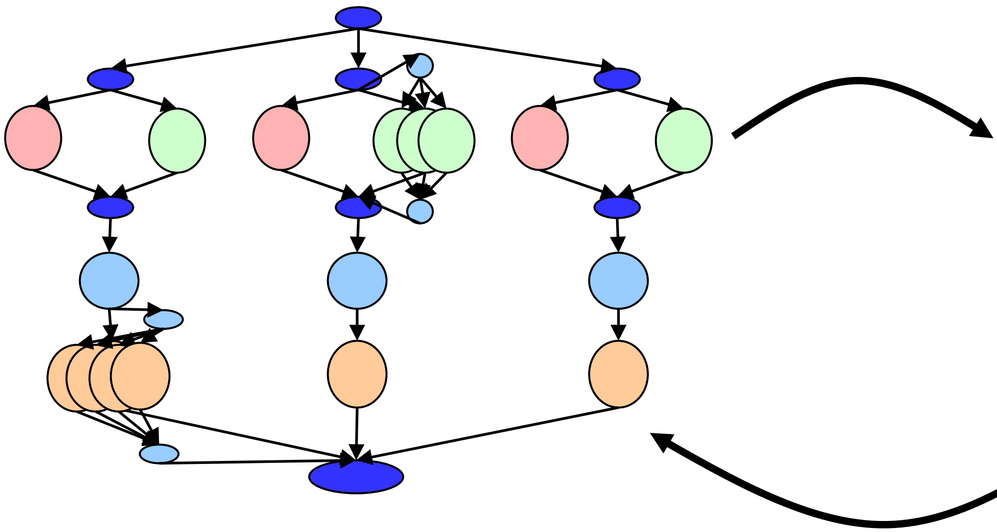


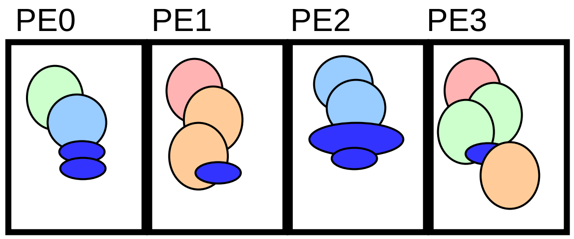


Speedup = 60/40 = 1.5

Speedup = 60/32 ~ 2

Filter Fission Choices

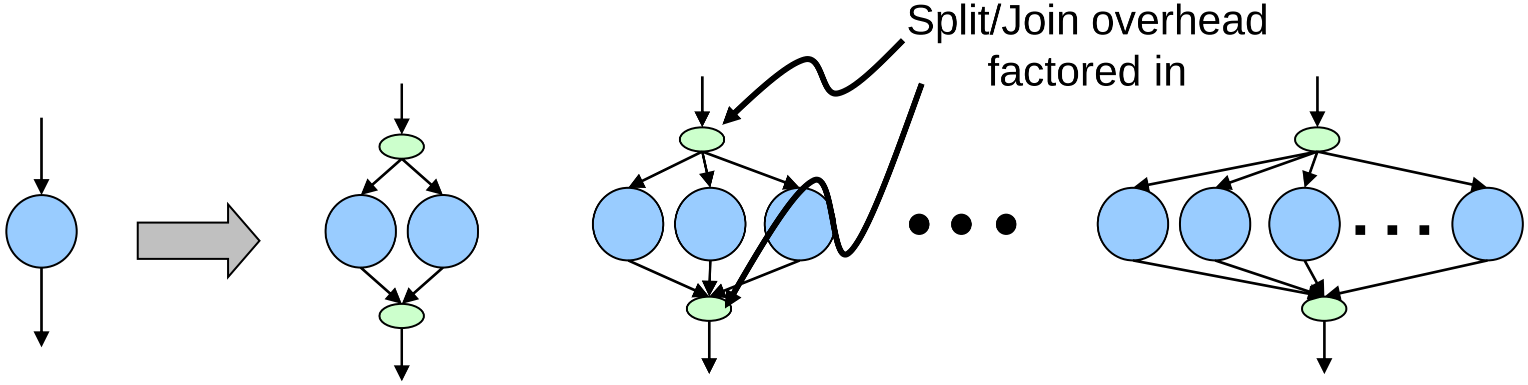


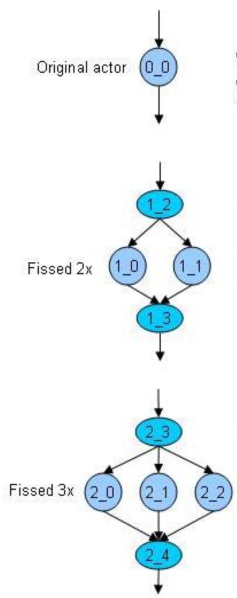


Speedup ~ 4 ?

Integrated Fission + PE Assign

- Exact solution based on Integer Linear Programming (ILP)





$$\sum_{i=1}^{P} a_{1,0,0,i} - \mathbf{b}_{1,0} = 0$$

$$\sum_{i=1}^{P}\sum_{j=0}^{3} a_{1,1,j,i} - \mathbf{b}_{1,1} \le M\mathbf{b}_{1,1}$$

$$\sum_{i=1}^{P}\sum_{j=0}^{3} a_{1,1,j,i} - \mathbf{b}_{1,1} - 2 \ge -M + M\mathbf{b}_{1,1}$$

$$\sum_{i=1}^{P}\sum_{j=0}^{4} a_{1,2,j,i} - \mathbf{b}_{1,2} \le M\mathbf{b}_{1,2}$$

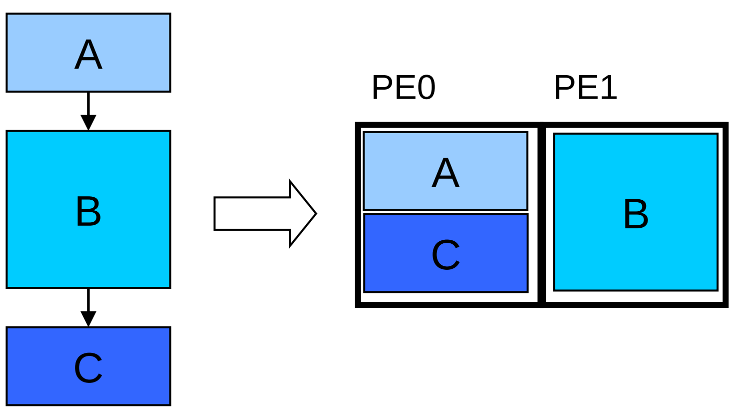
$$\sum_{i=1}^{P}\sum_{j=0}^{4} a_{1,2,j,i} - \mathbf{b}_{1,2} - 3 \ge -M + M\mathbf{b}_{1,2}$$

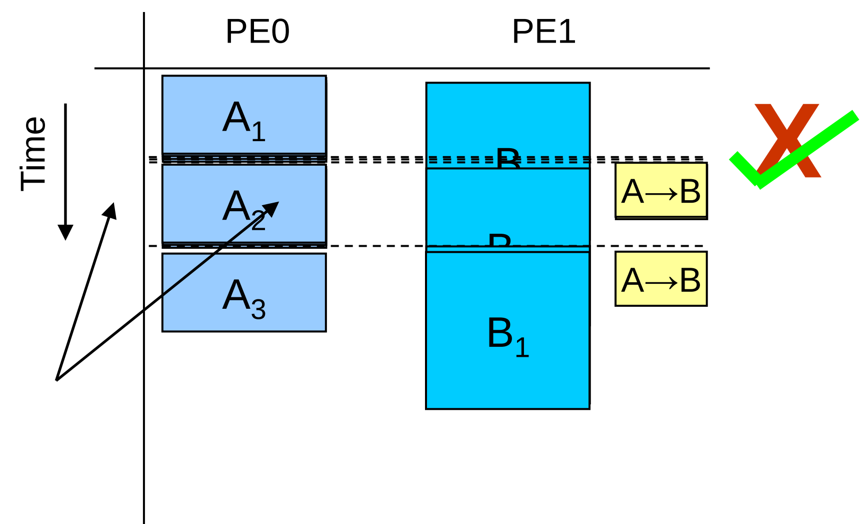
$$\boxed{b_{1,0} + b_{1,1} + b_{1,2} = 1}$$

- Objective function- Maximal load on any PE
  - » Minimize
- Result
  - » Number of times to “split” each filter
  - » Filter → processor mapping

Step 2: Forming the Software Pipeline

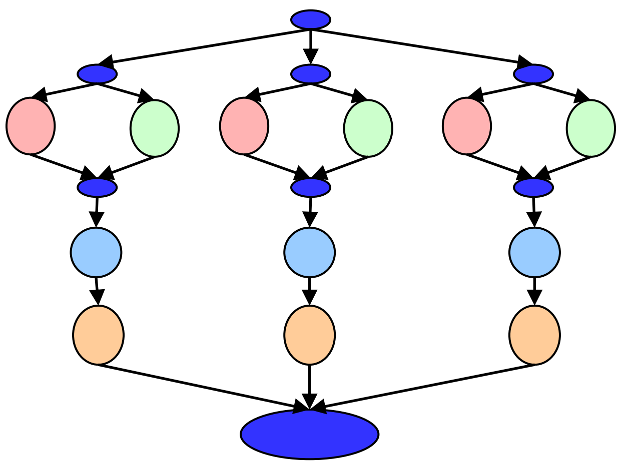
- To achieve speedup
  - All chunks should execute concurrently
  - Communication should be overlapped
- Processor assignment alone is insufficient information

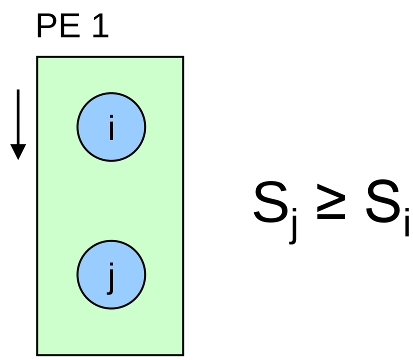




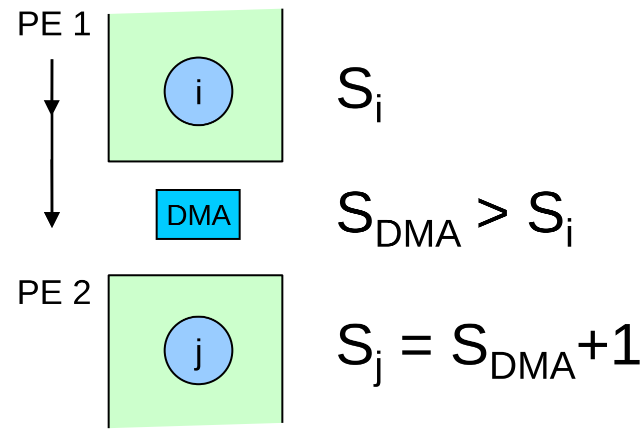
Overlap A_{i+2} with B_i

Stage Assignment





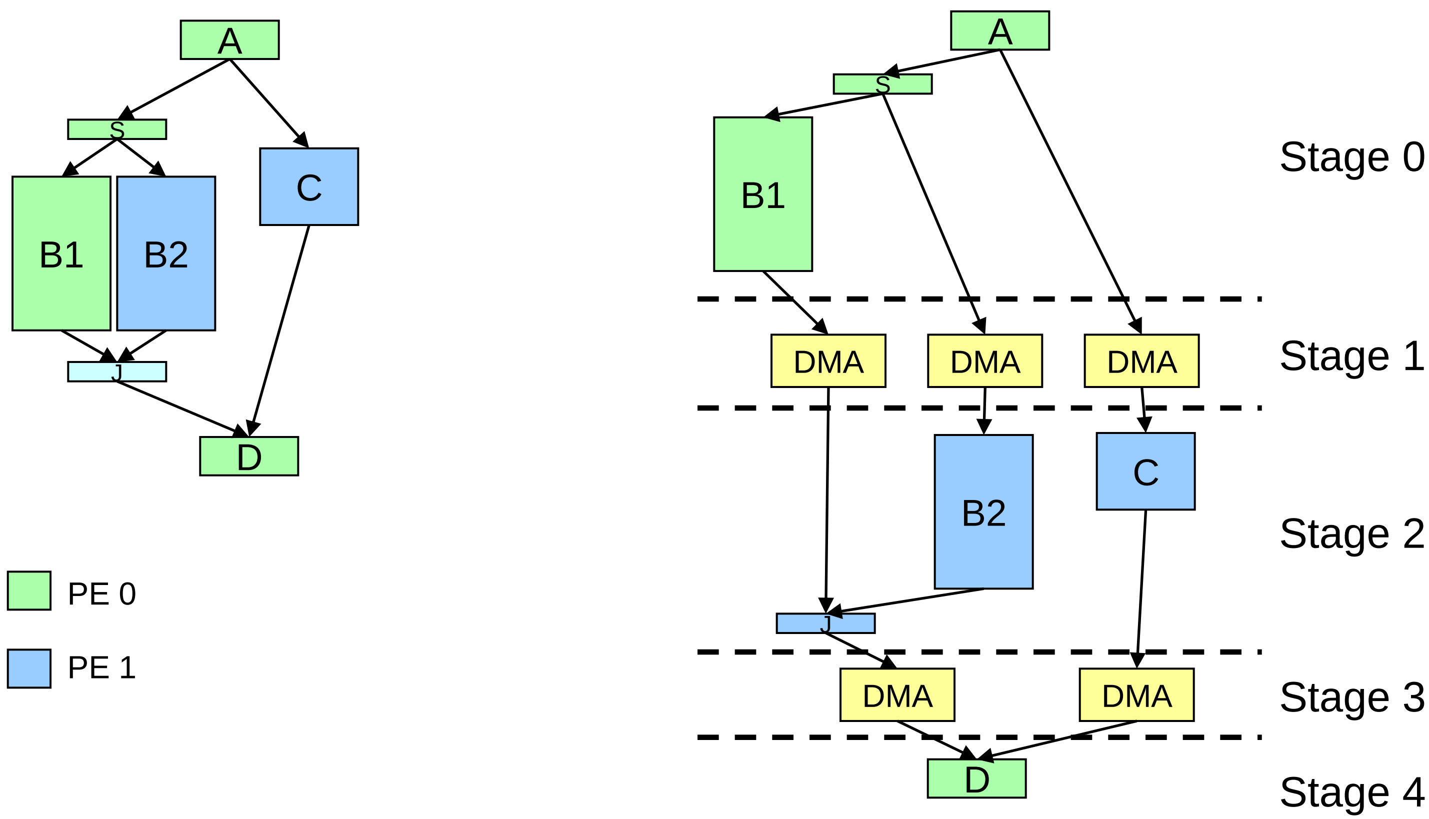
Preserve causality
(producer-consumer dependence)



Communication-computation
overlap

- Data flow traversal of the stream graph
  - Assign stages using above two rules

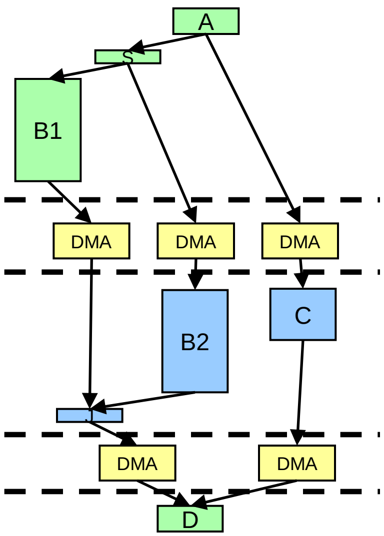
Stage Assignment Example



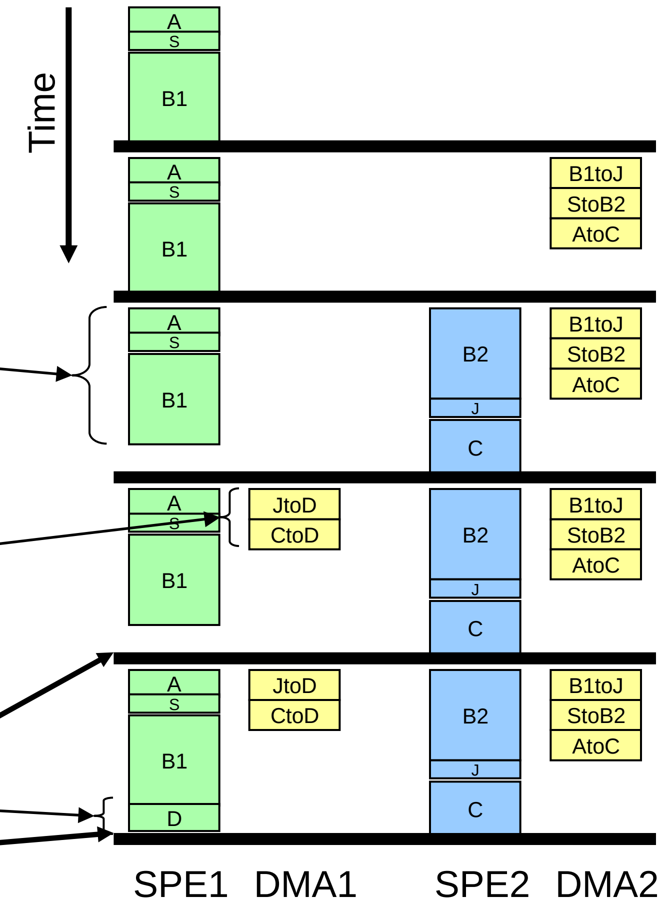
Step 3: Code Generation for Cell

- Target the Synergistic Processing Elements (SPEs)
  - » PS3 – up to 6 SPEs
  - » QS20 – up to 16 SPEs
- One thread / SPE
- Challenge
  - » Making a collection of independent threads implement a software pipeline
  - » Adapt kernel-only code schema of a modulo schedule

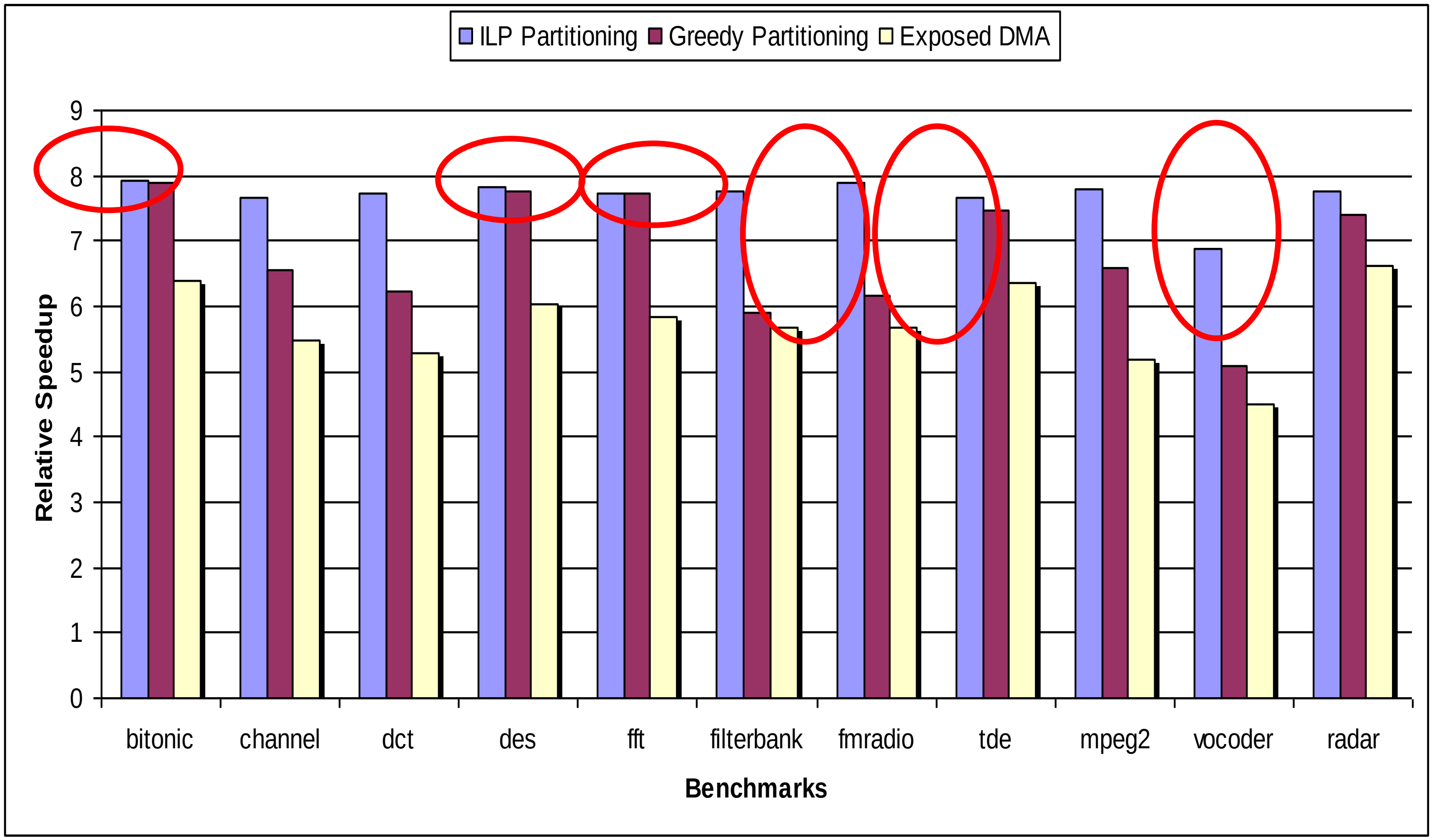
Complete Example




```
void spe1_work()
{
  char stage[5] = {0};
  stage[0] = 1;
  for(i=0; i<MAX; i++) {
    if (stage[0]) {
      A();
      S();
      B1();
    }
    if (stage[1]) {
    }
    if (stage[2]) {
      JtoD();
      CtoD();
    }
    if (stage[3]) {
    }
    if (stage[4]) {
      D();
    }
    barrier();
  }
}
```



SGMS(ILP) vs. Greedy (MIT method, ASPLOS'06)



- Solver time < 30 seconds for 16 processors

SGMS Conclusions

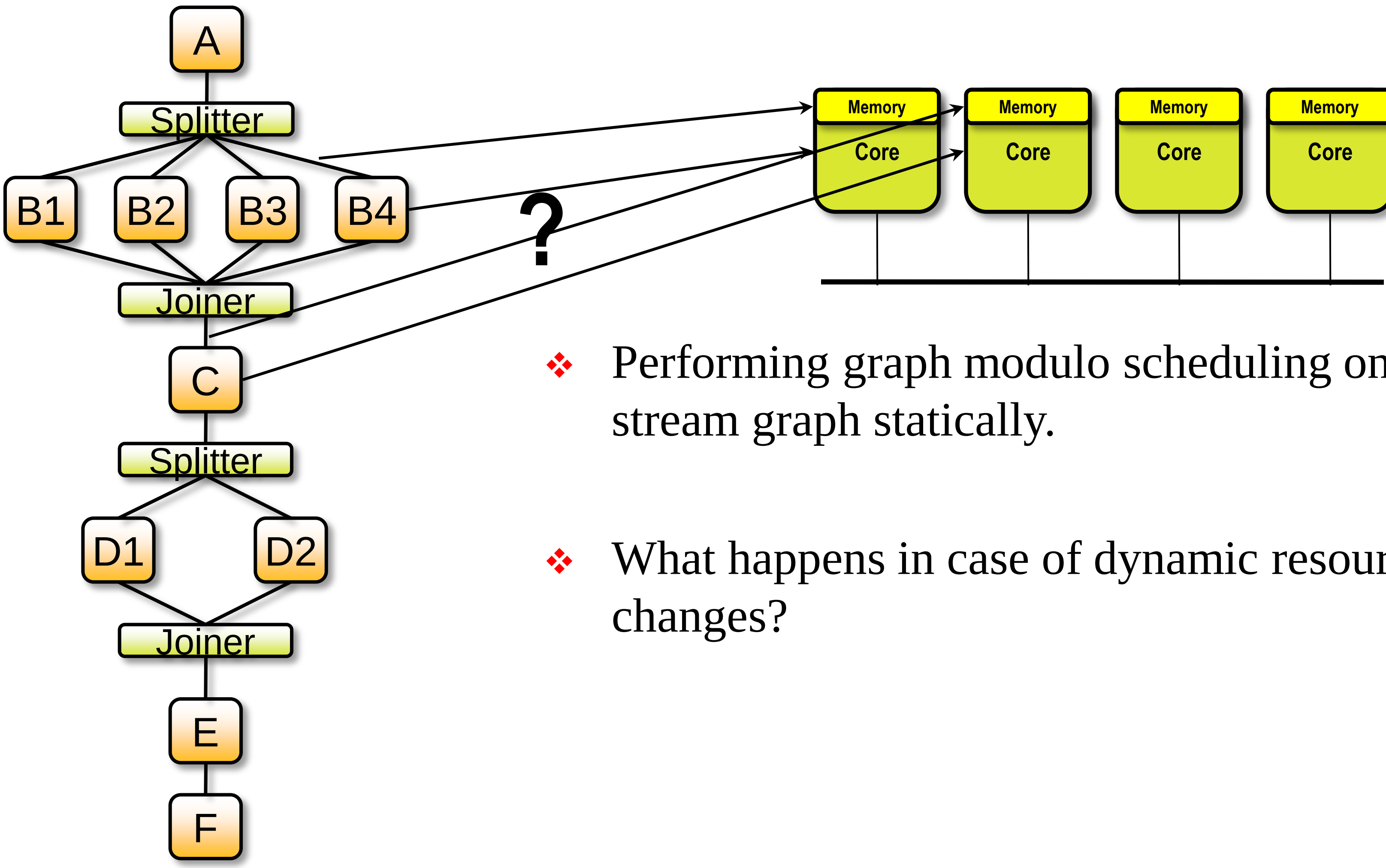
- Streamroller
  - Efficient mapping of stream programs to multicore
  - Coarse grain software pipelining
- Performance summary
  - 14.7x speedup on 16 cores
  - Up to 35% better than greedy solution (11% on average)
- Scheduling framework
  - Tradeoff memory space vs. load balance
    - Memory constrained (embedded) systems
    - Cache based system

Discussion Points

- Is it possible to convert stateful filters into stateless?
- What if the application does not behave as you expect?
  - Filters change execution time?
  - Memory faster/slower than expected?
- Could this be adapted for a more conventional multiprocessor with caches?
- Can C code be automatically streamized?
- Now you have seen 3 forms of software pipelining:
  - 1) Instruction level modulo scheduling, 2) Decoupled software pipelining, 3) Stream graph modulo scheduling
  - Where else can it be used?

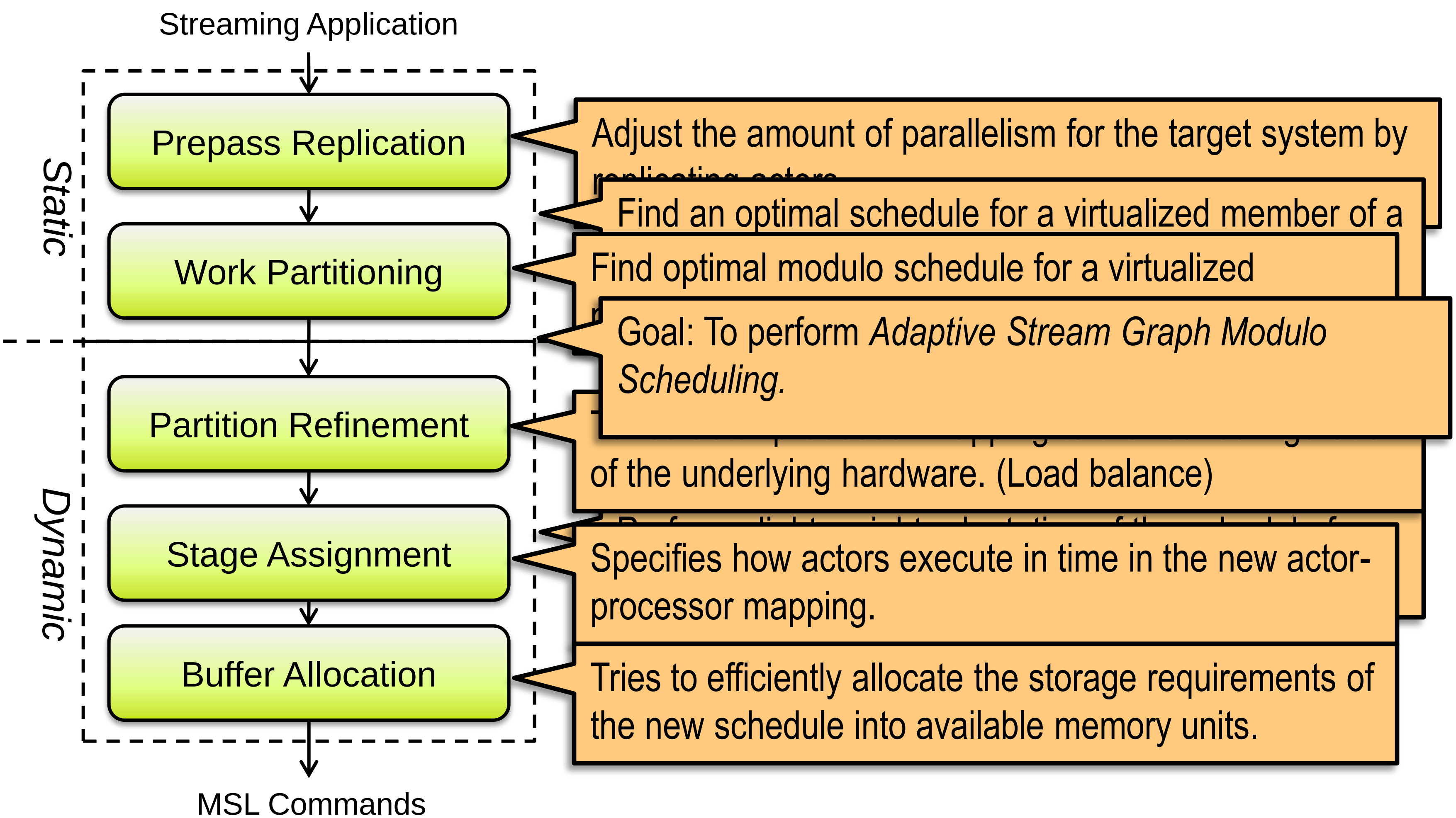
“Flextream: Adaptive Compilation of Streaming Applications for Heterogeneous Architectures,”

Static Verses Dynamic Scheduling



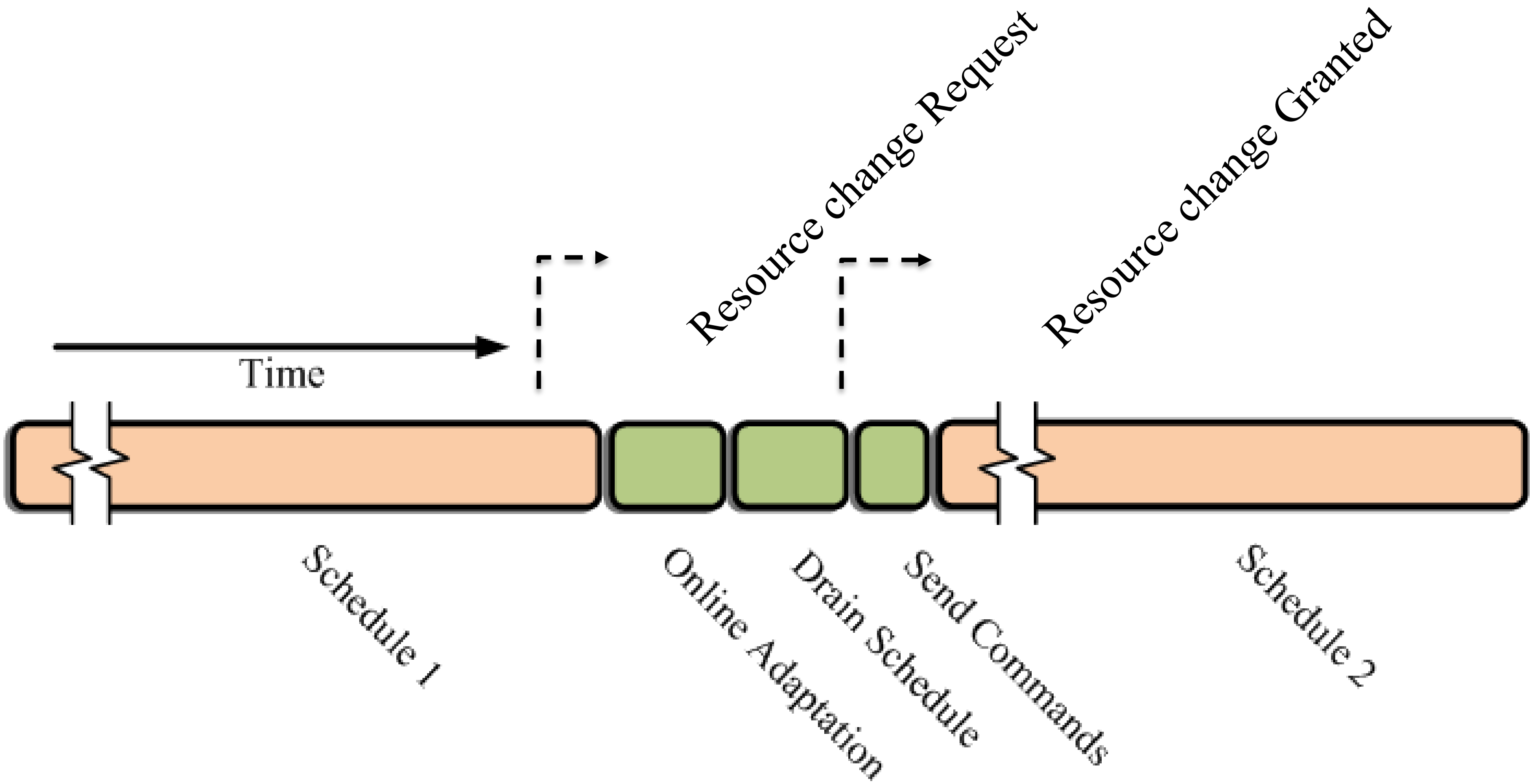
- Performing graph modulo scheduling on a stream graph statically.
- What happens in case of dynamic resource changes?

Overview of Flextream

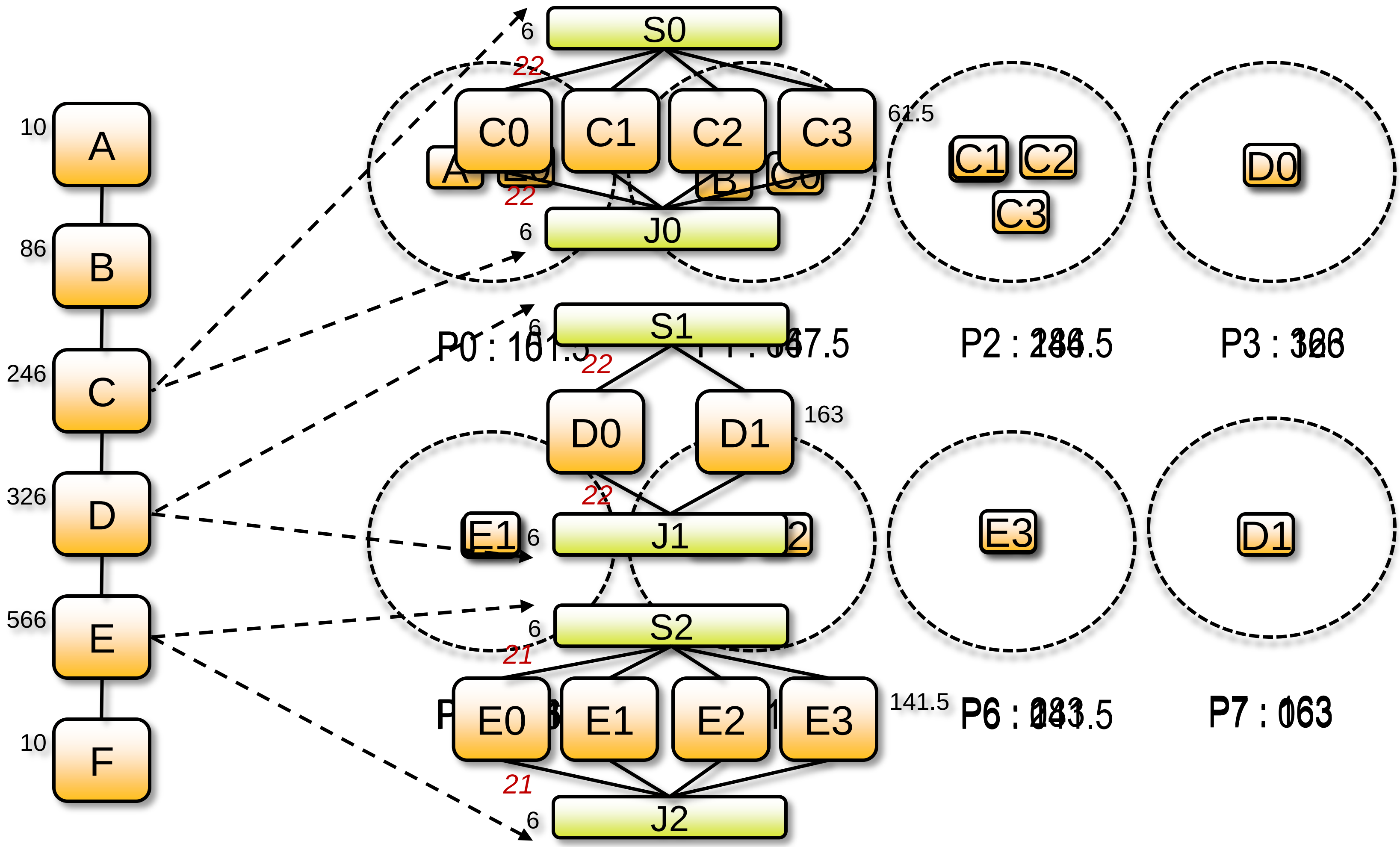


Overall Execution Flow

- For every application may see multiple iterations of:



Prepass Replication [static]

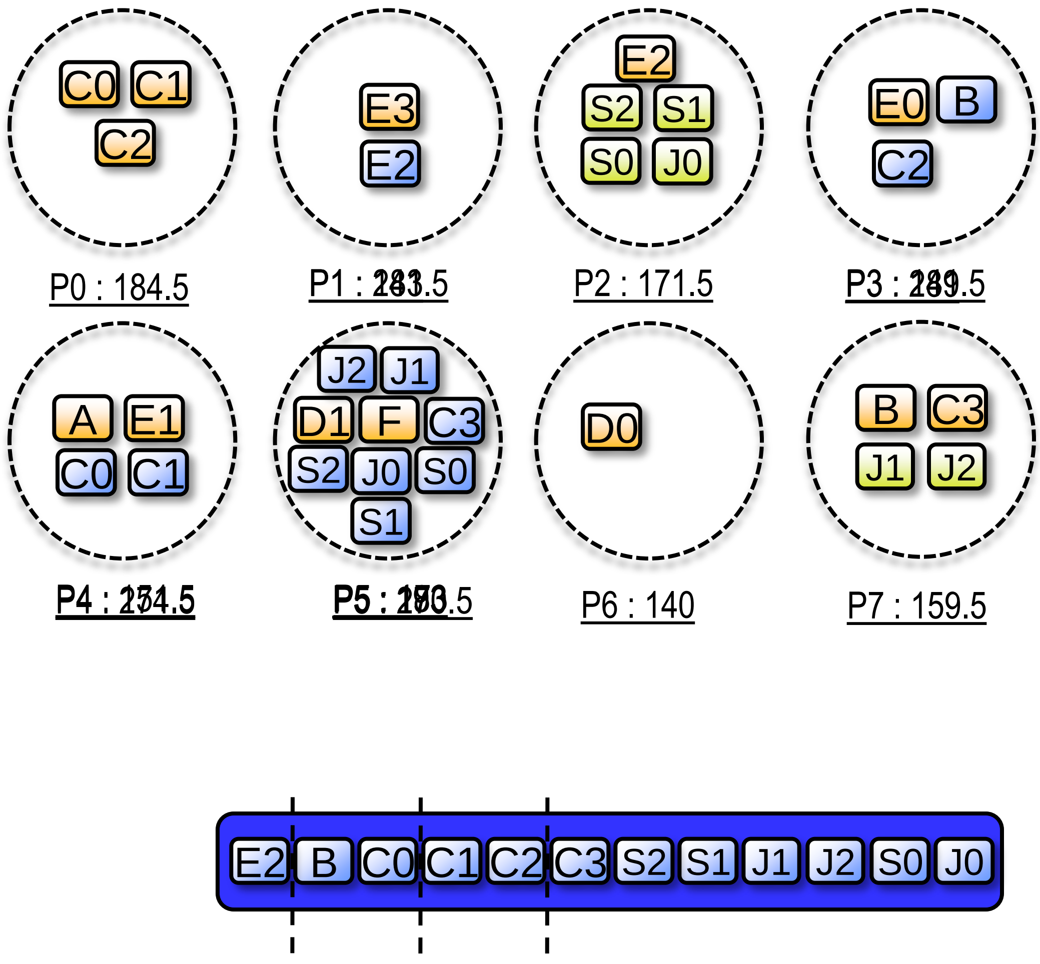


Partition Refinement ***[dynamic 1]***

- Available resources at runtime can be more limited than resources in static target architecture.
- Partition refinement tunes actor to processor mapping for the active configuration.
- A greedy iterative algorithm is used to achieve this goal.

Partition Refinement Example

- Pick processors with most number of actors.
- Sort the actors
- Find processor with max work
- Assign min actors until threshold



Stage Assignment ***[dynamic 2]***

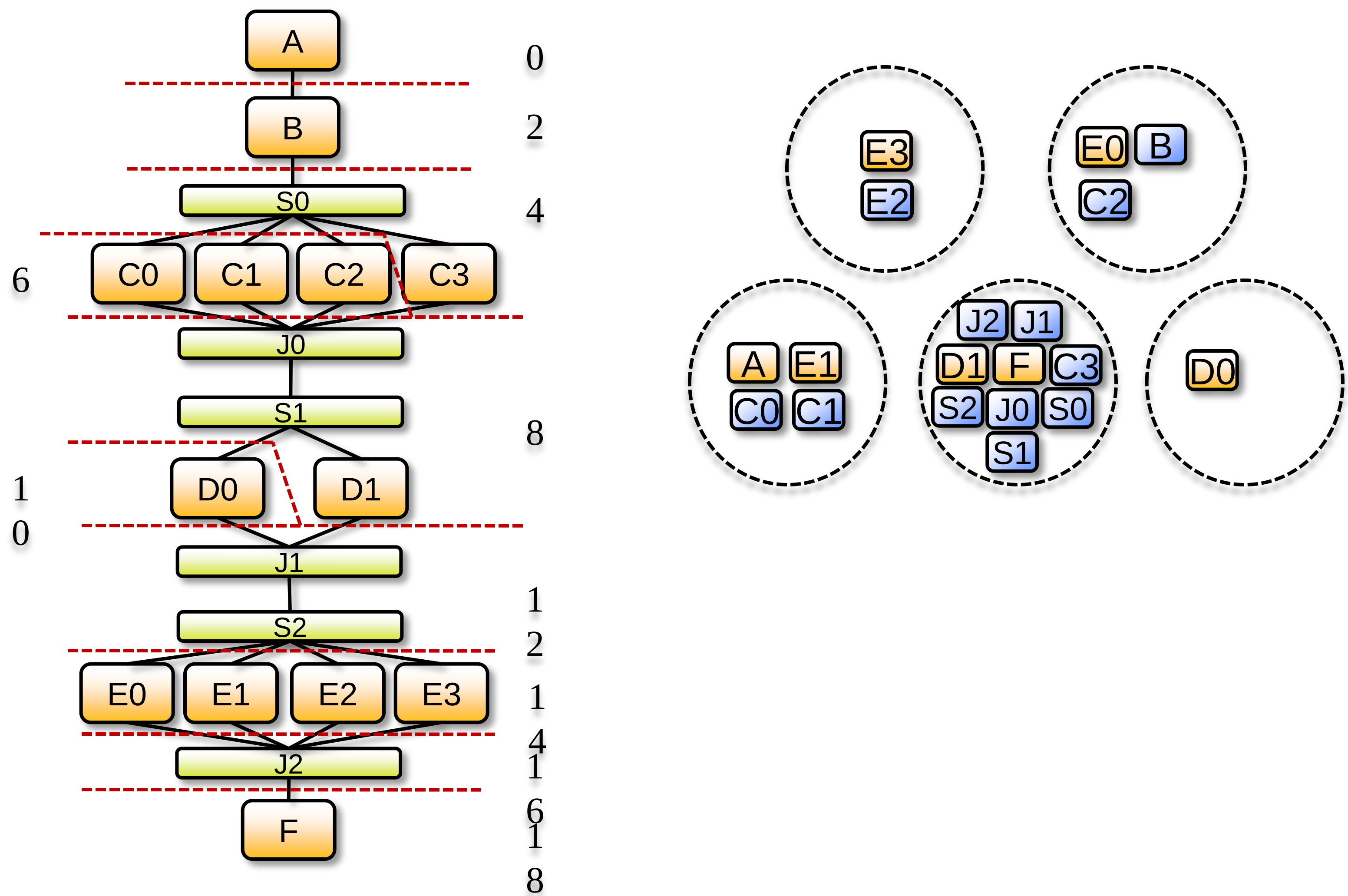
- Processor assignment only specifies how actors are overlapped across processors.

- Stage assignment finds how actors are overlapped in time.

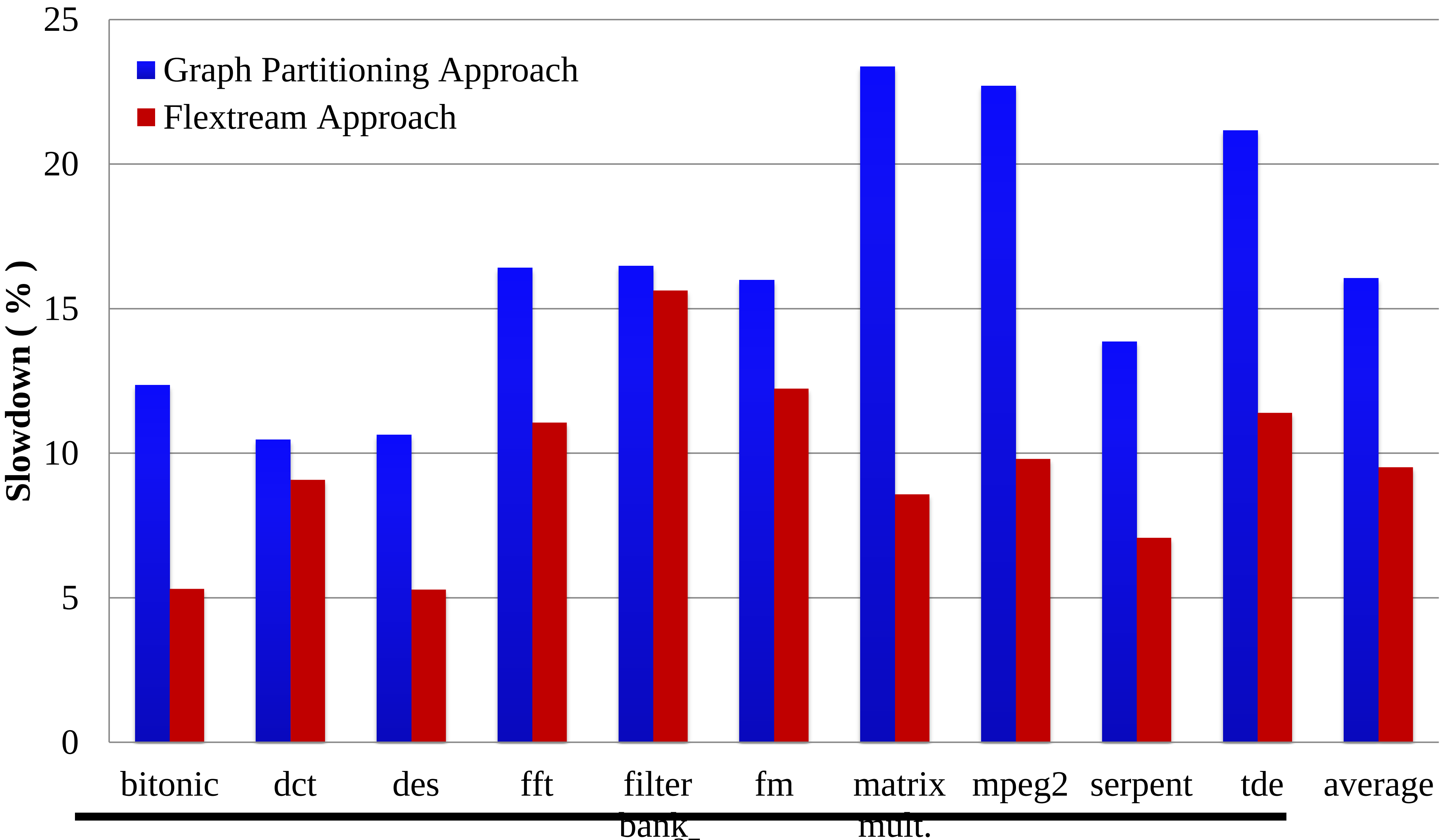
- Relative start time of the actors is based on stage numbers.

- DMA operations will have a separate stage.

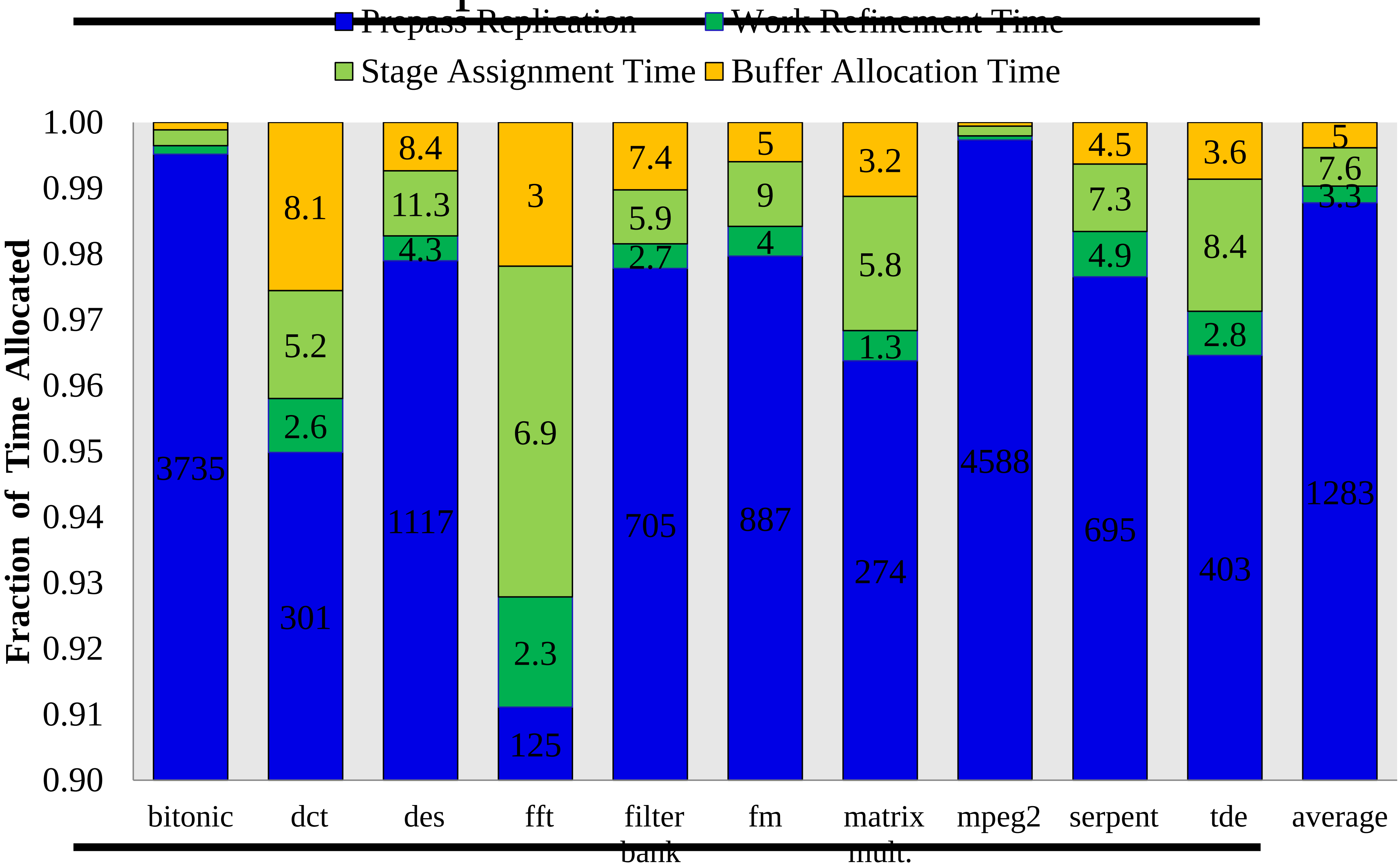
Stage Assignment Example



Performance Comparison



Overhead Comparison



Flextream Conclusions

- Static scheduling approaches are promising but not enough.
- Dynamic adaptation is necessary for future systems.
- Flextream provides a hybrid static/dynamic approach to improve efficiency.