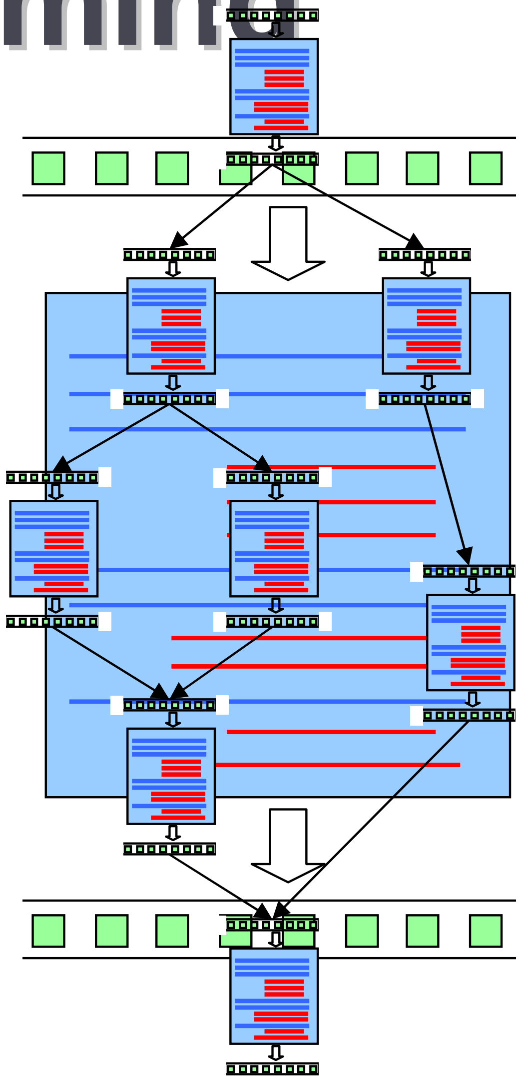
Adaptive Streaming for Dealing with Dynamic Heterogeneity

Amir Hormati and Scott Mahlke
Advanced Computer Architecture Lab.
University of Michigan

Stream Programming

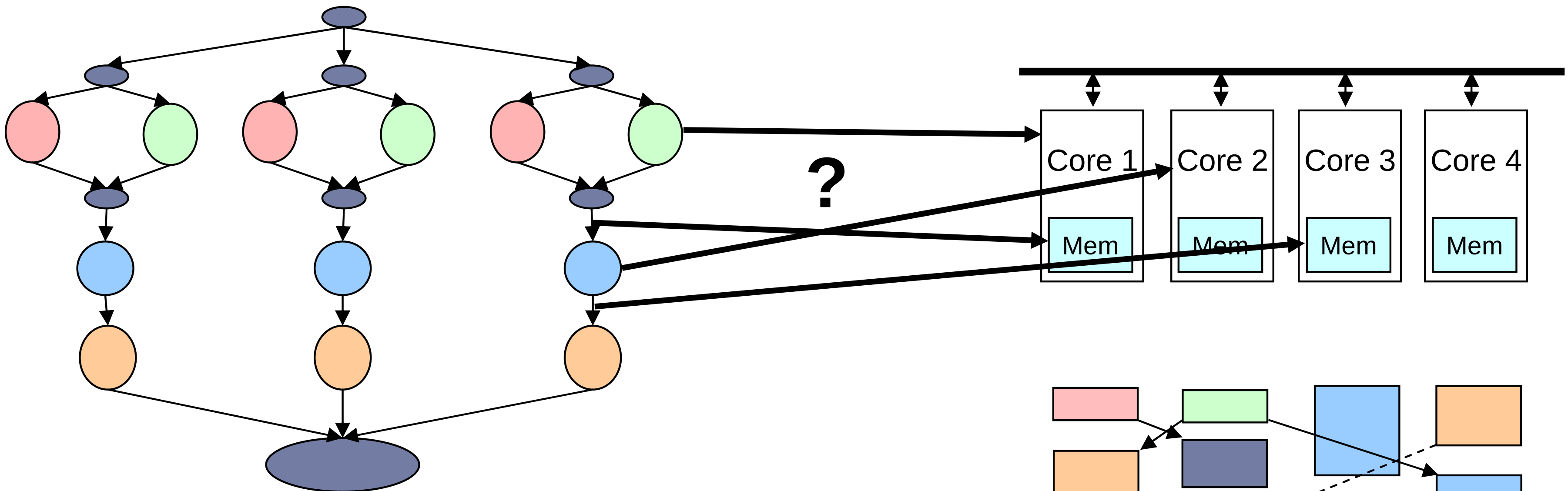
- Programming style
  - Embedded domain
    - Audio/video (H.264), wireless (WCDMA)
  - Mainstream
    - Continuous query processing (IBM SystemS), Search (Google Sawzall)
- Stream
  - Collection of data records
- Kernels/Filters
  - Functions applied to streams
  - Input/Output are streams
  - Coarse grain dataflow
  - Amenable to aggressive compiler optimizations [ASPLOS'02, '06, PLDI '03, PLDI '08]

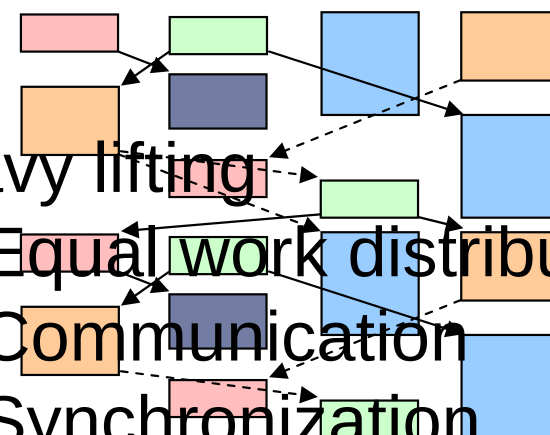


Compiling Stream Programs

Stream Program

Multicore System

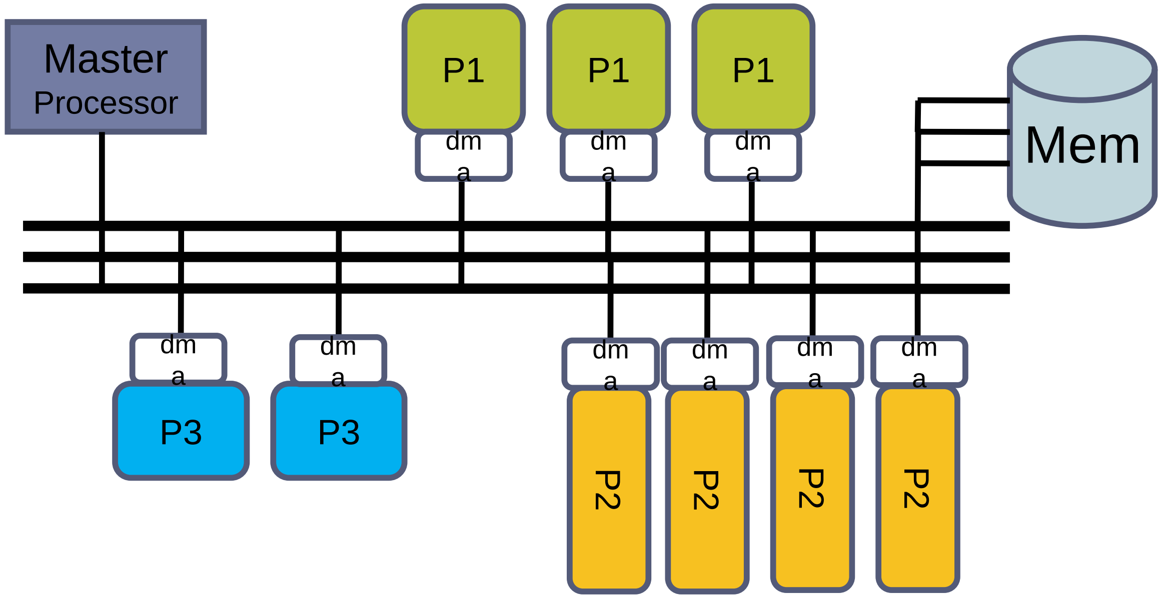




- Heavy lifting
  - Equal work distribution
  - Communication
  - Synchronization

Target Architecture

- Cores with disjoint address spaces
- Explicit copy to access remote data
- DMA engine independent of Processors

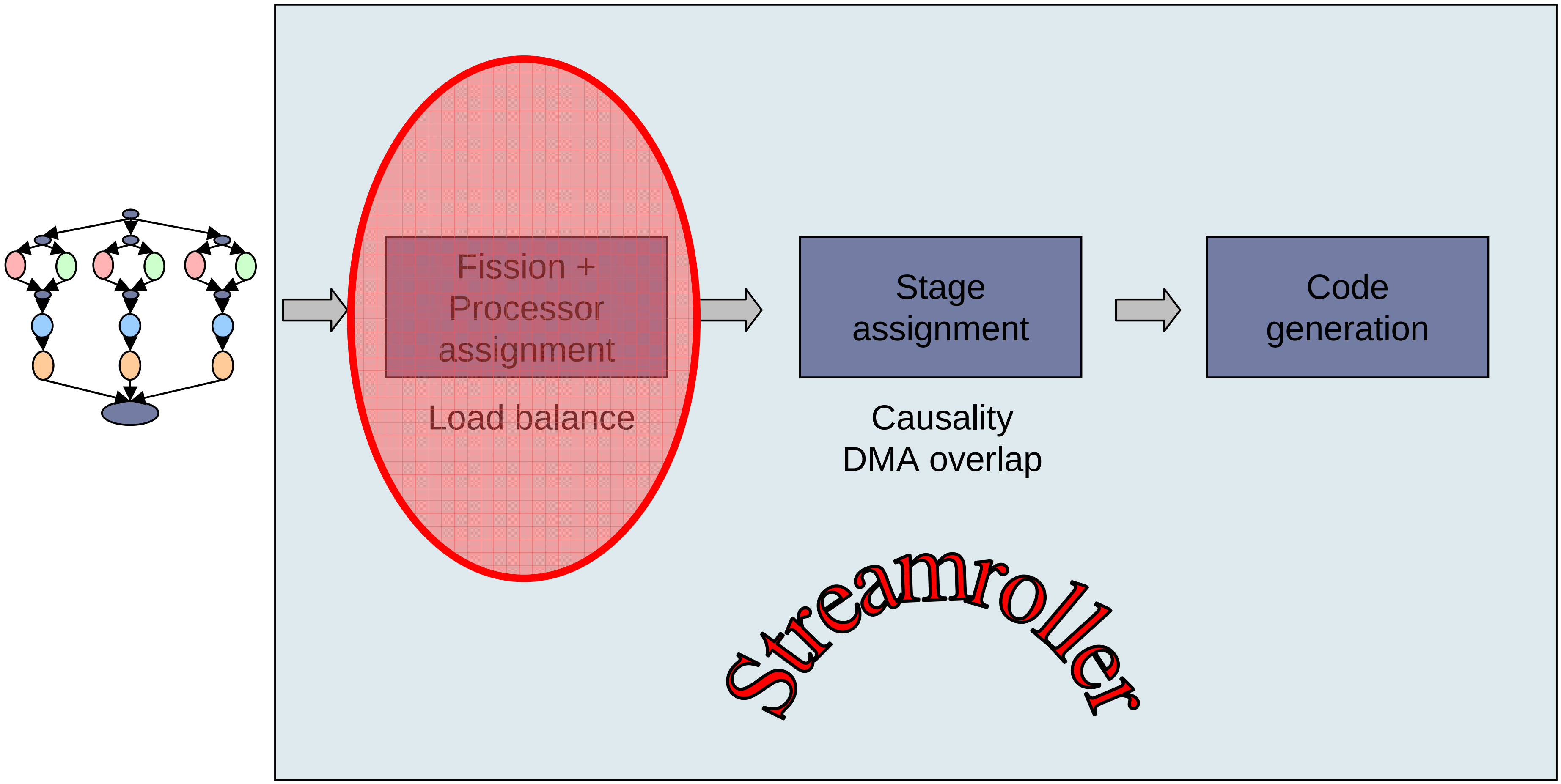


Orchestrating Stream Graphs

- Common phases:
  - Rate Matching
  - Graph Refinement
  - Scheduling
  - Mapping

- The phase ordering varies in different approaches.

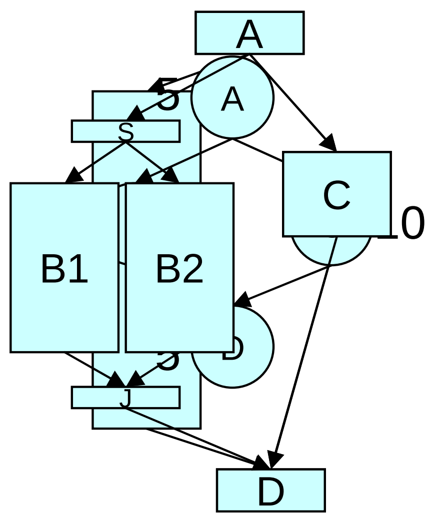
Static Stream Compilation

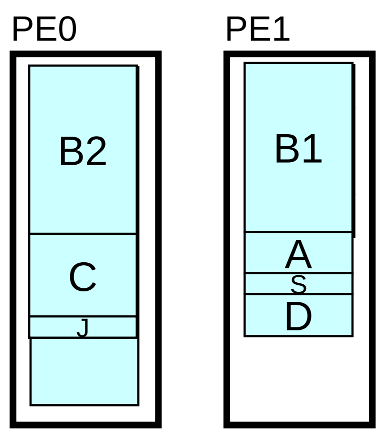


Kudlur, PLDI 2008

Processor Assignment

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

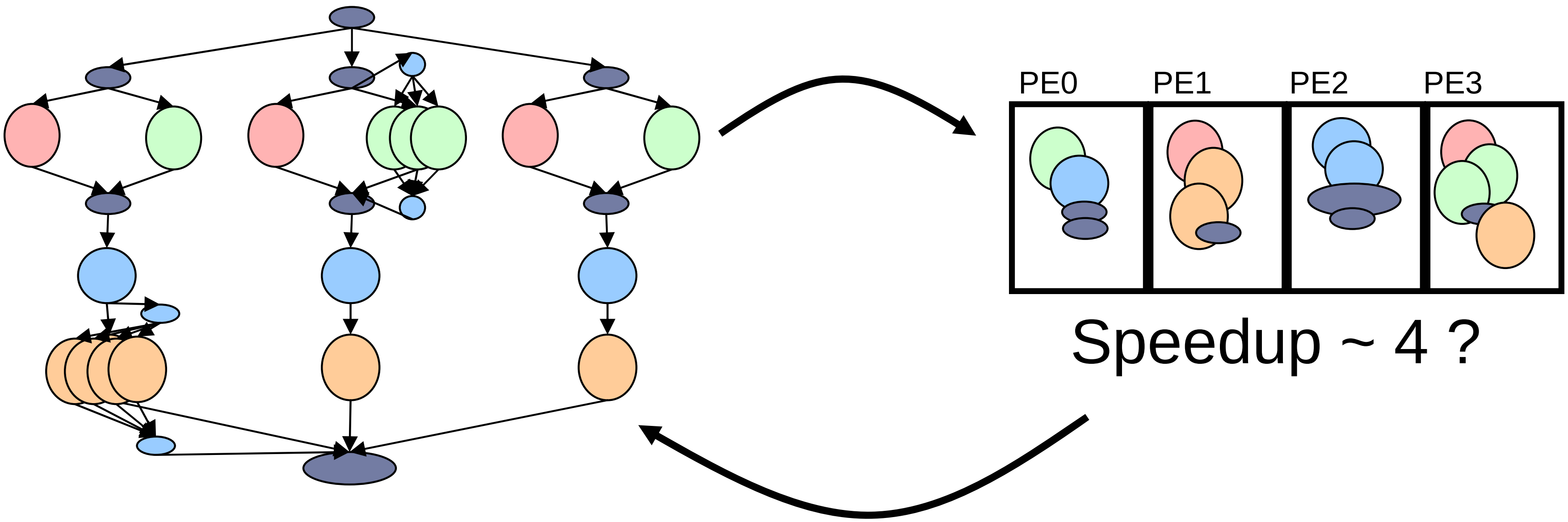


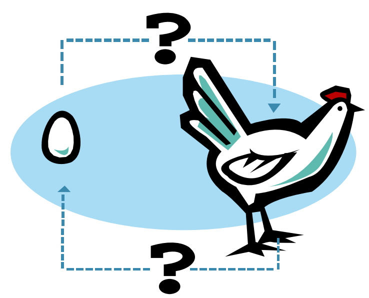


Speedup = 60/40 = 1.5

Speedup = 60/32 ~ 2

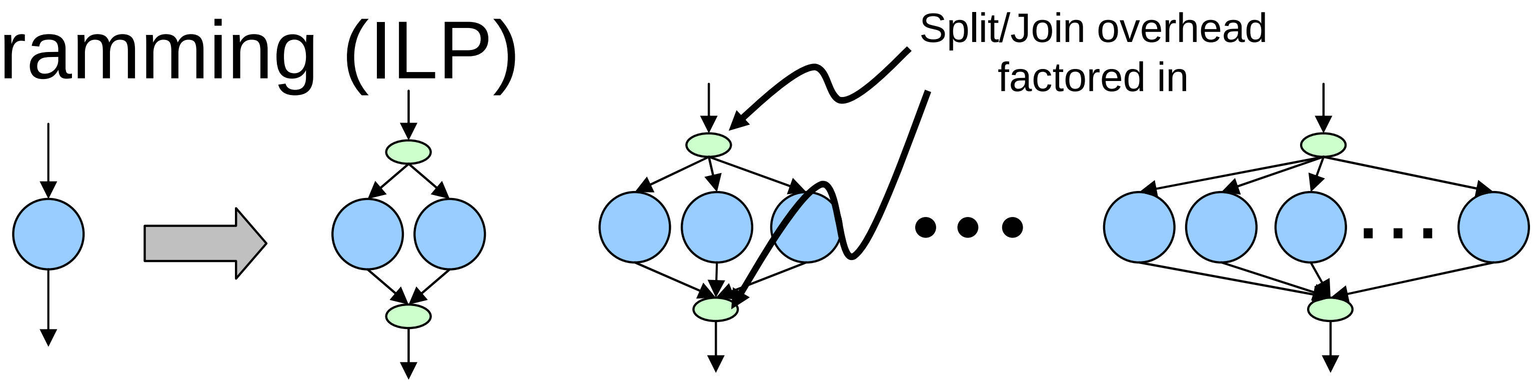
Filter Fission Choices

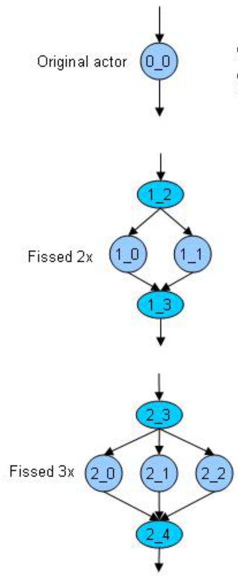




Integrated Fission + PE Assign

- Exact solution based on Integer Linear Programming (ILP)





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

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

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

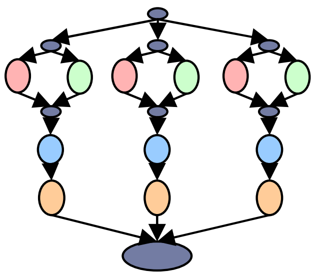
$$\sum_{i=1}^{P}\sum_{j=0}^{4} a_{1,2,j,i} - \mathbf{b}_{1,2} \le M\mathbf{b}_{1,2}$$

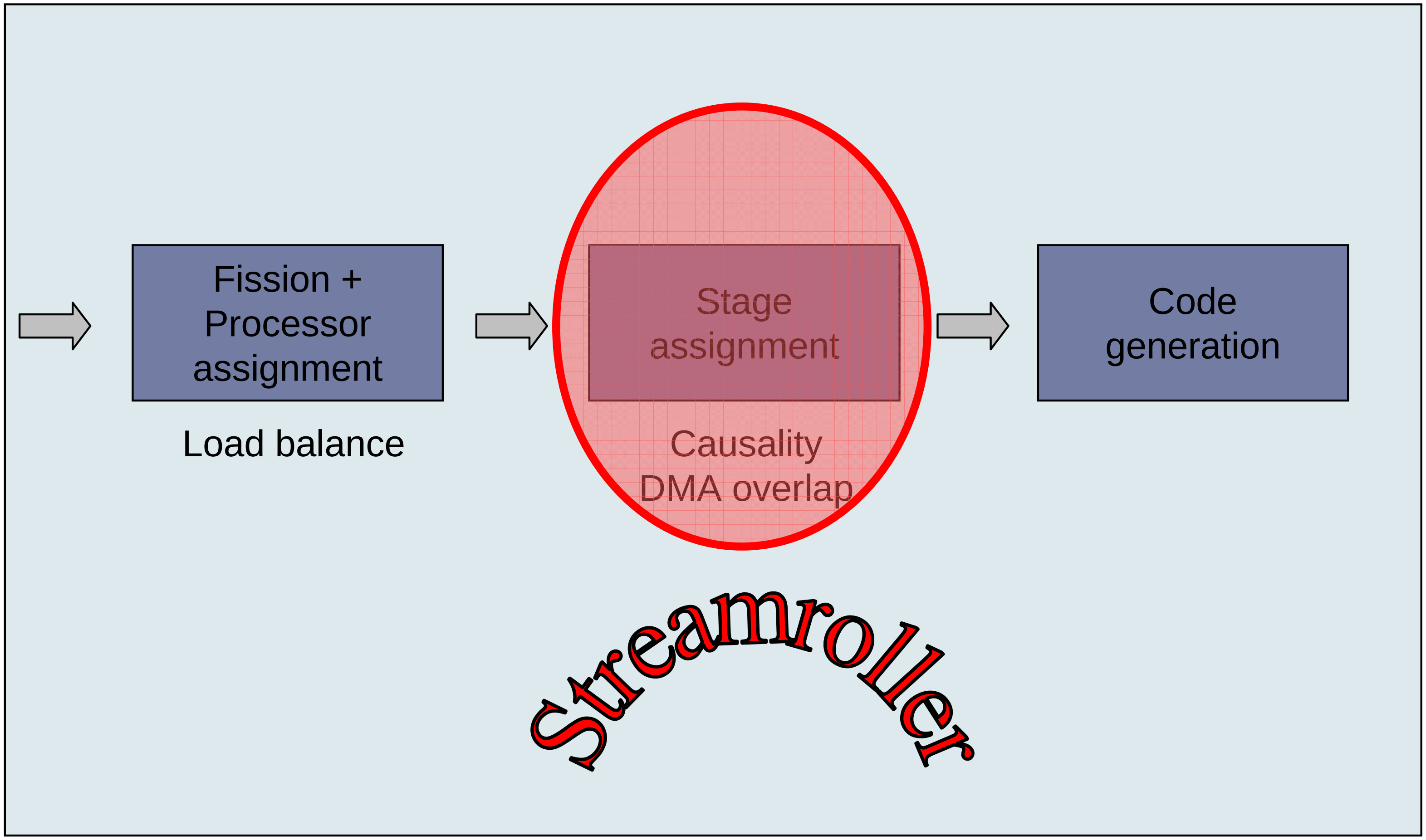
$$\sum_{i=1}^{P}\sum_{j=0}^{4} a_{1,2,j,i} - \mathbf{b}_{1,2} - 3 \ge -M + M\mathbf{b}_{1,2}$$

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

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

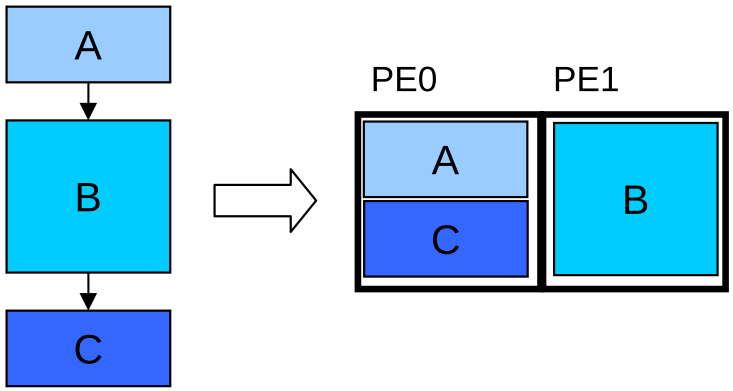
Static Stream Compilation – Step 2

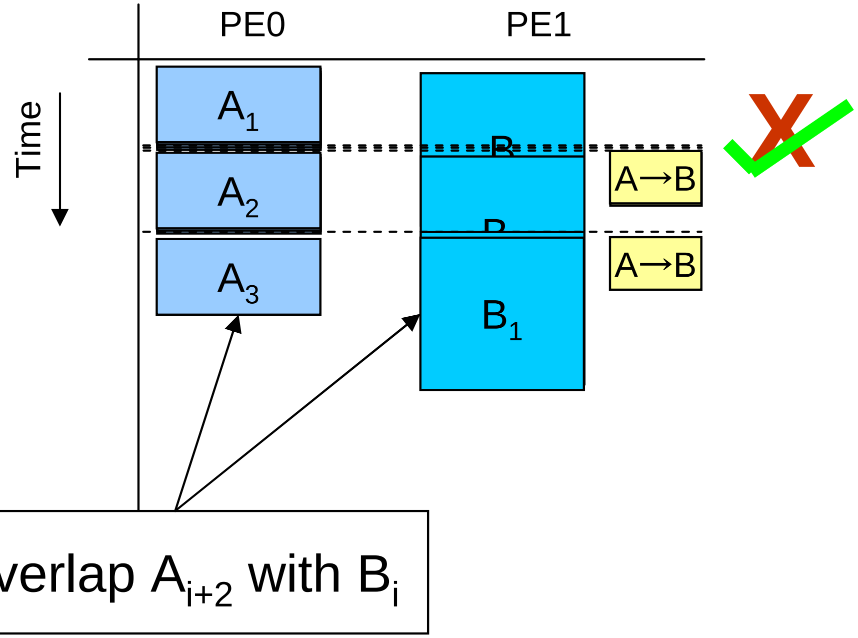




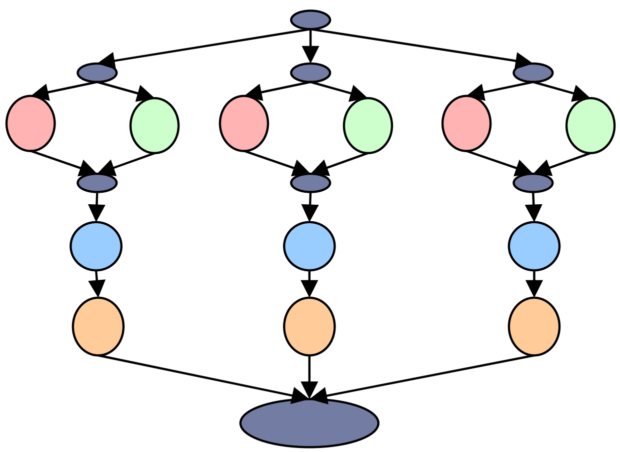
Forming the Software Pipeline

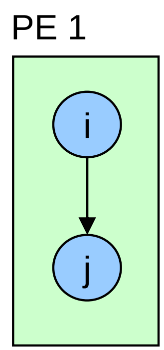
- To achieve speedup
  - All chunks should execute concurrently
  - Communication should be overlapped
- Processor assignment alone is insufficient information





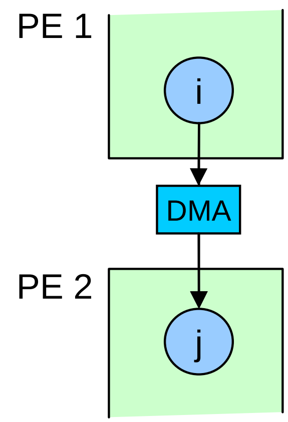
Stage Assignment





$S_j \geq S_i$

Preserve causality
(producer-consumer dependence)



S_i

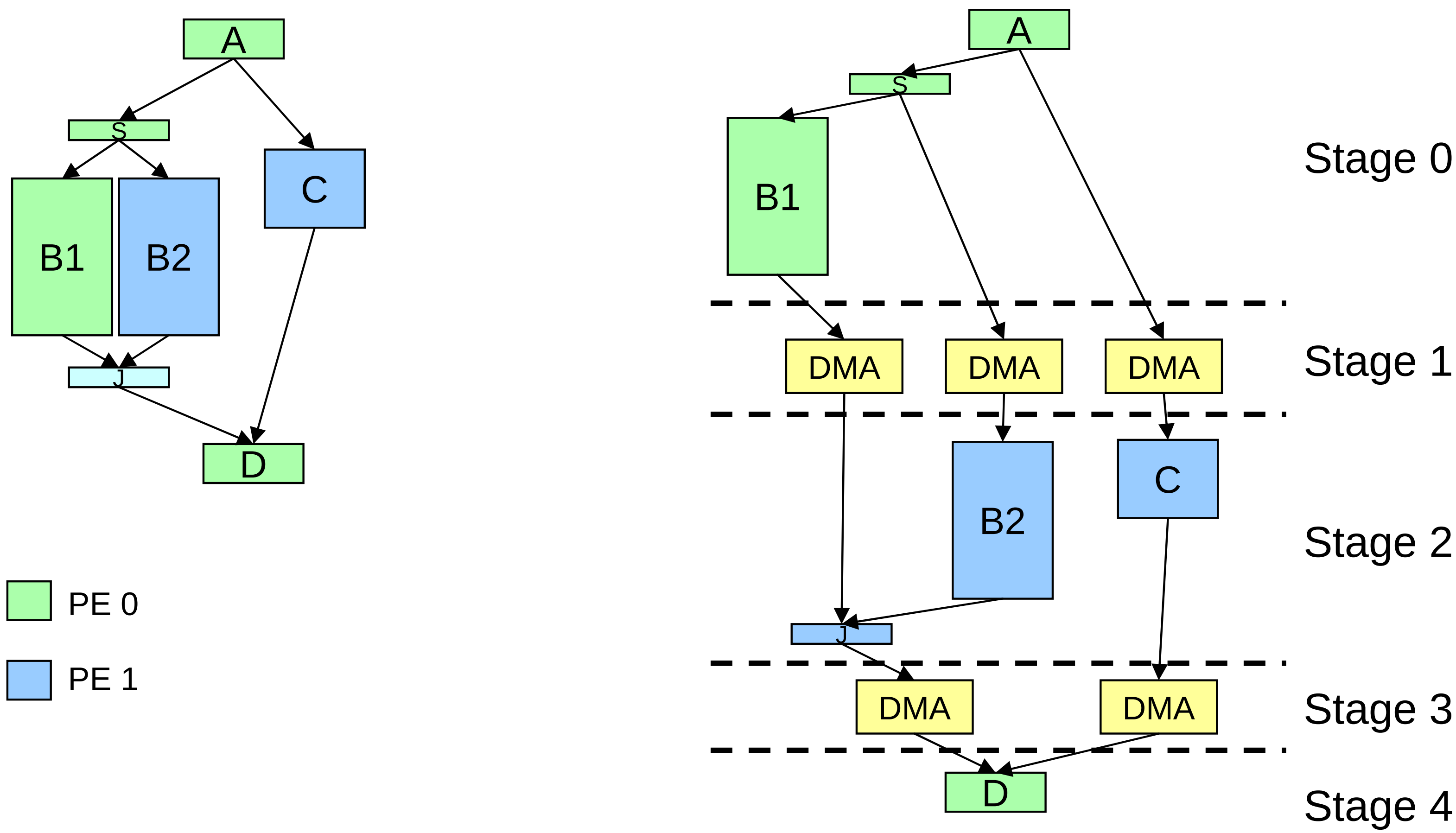
$S_{DMA} > S_i$

$S_j = S_{DMA}+1$

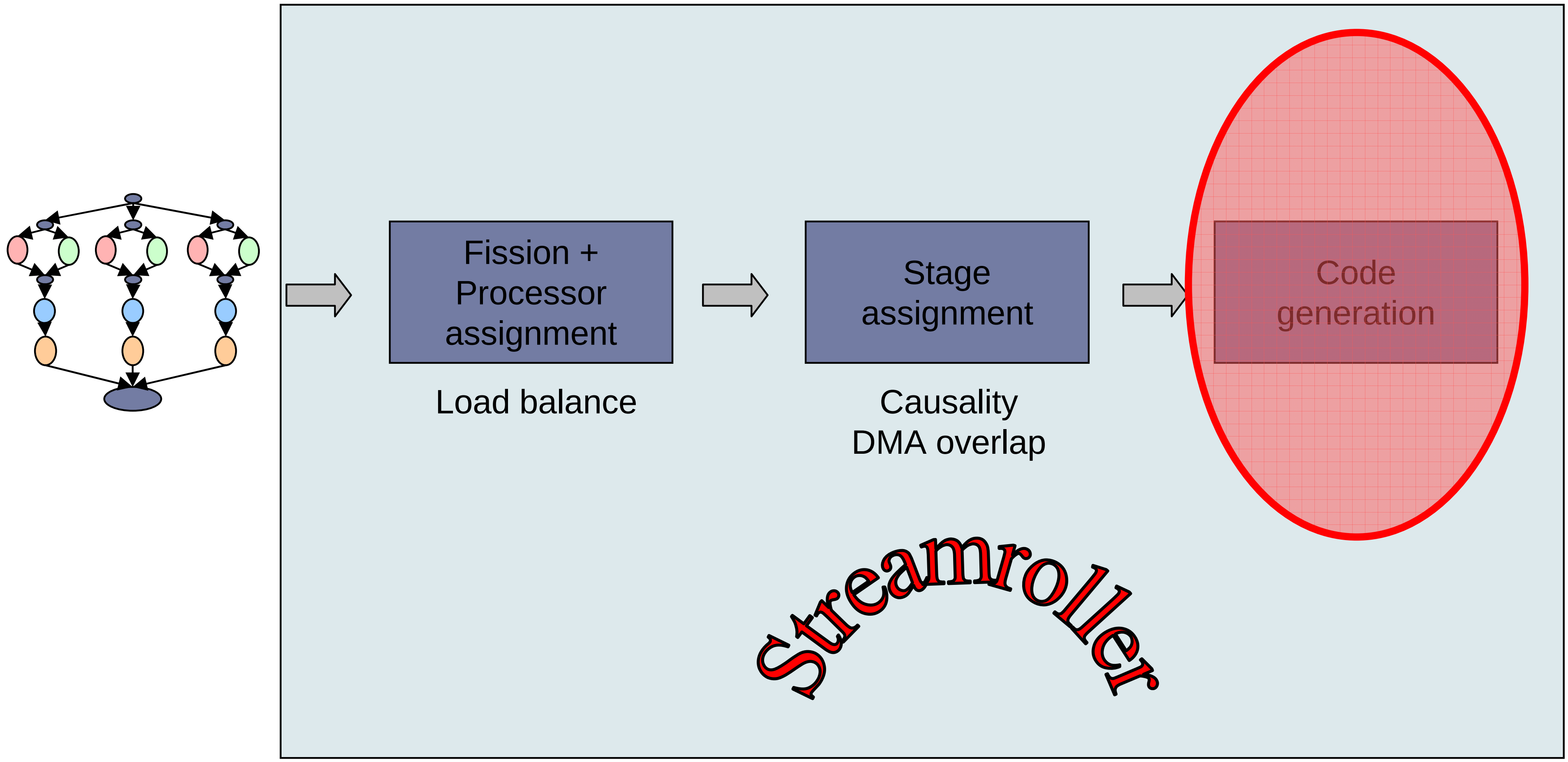
Communication-computation overlap

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

Stage Assignment Example



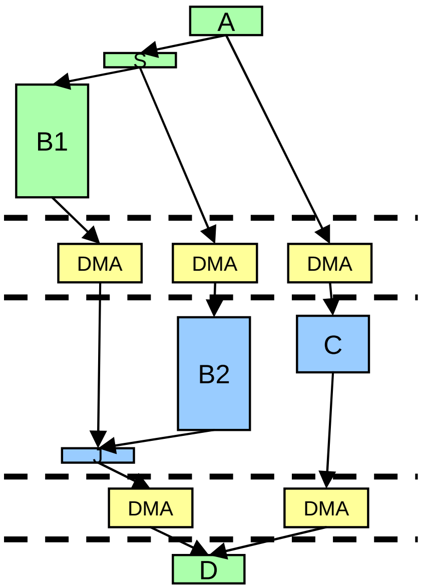
Static Stream Compilation – Step 3



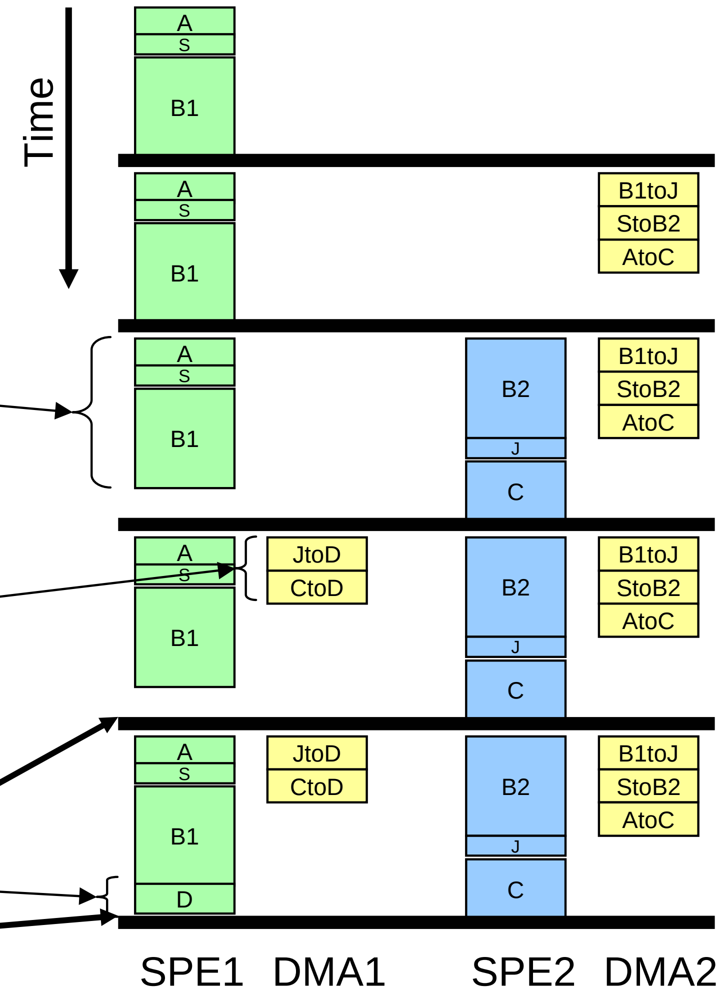
Code Generation for Cell

- 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

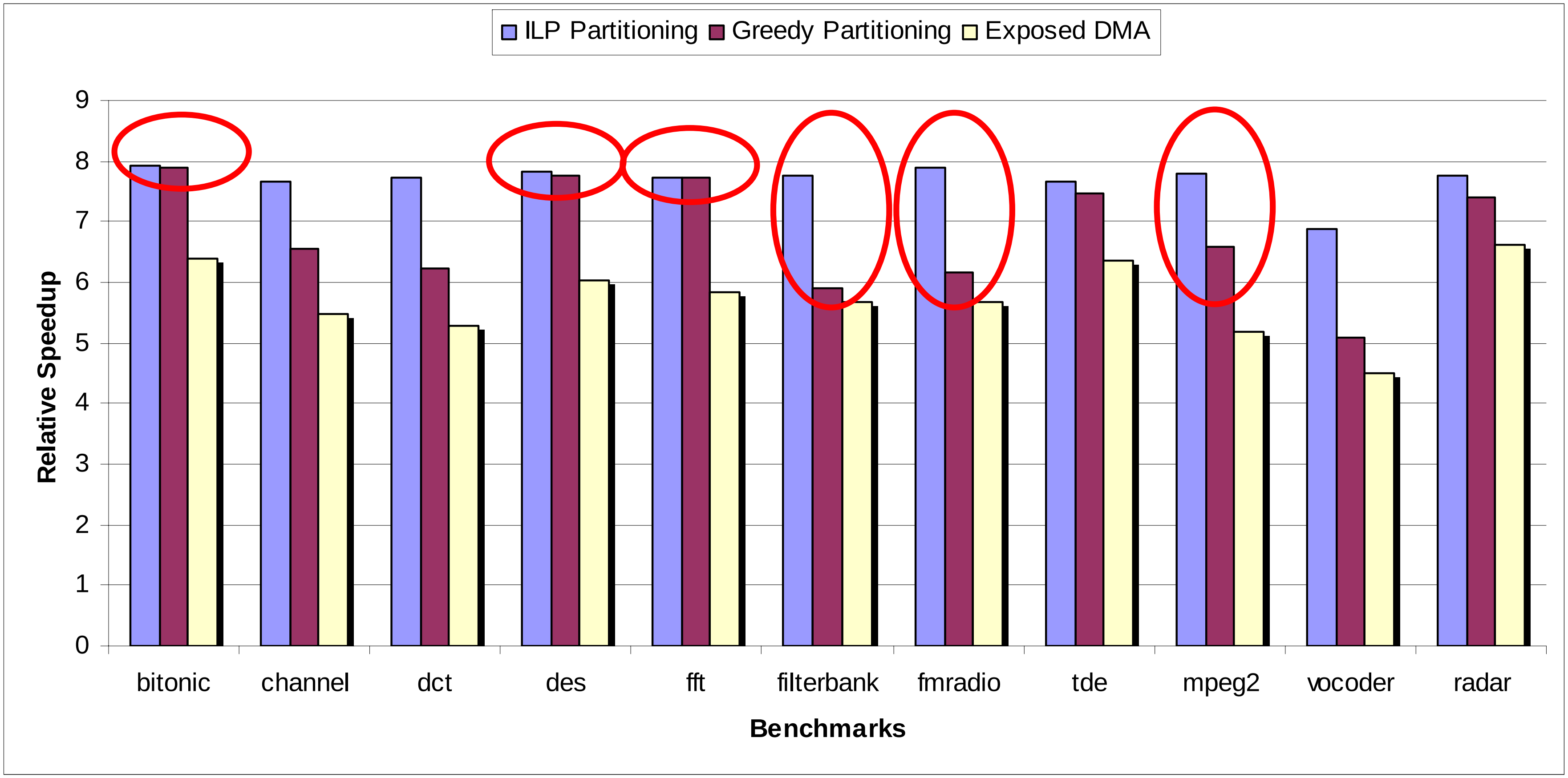
Complete Example




```
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 (8 core Cell) (MIT method, ASPLOS'06)

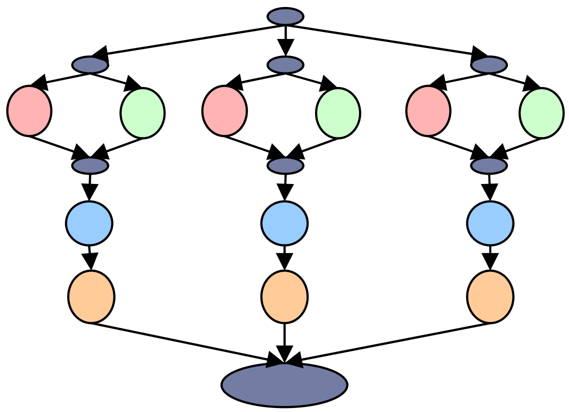


- Solver time < 30 seconds for 16 processors

Summary of Static Approach

- Advantages:
  - Optimal load balance
  - Allocate local memory
  - Overlap DMAs with computation
  - No runtime overhead
- But, lacks ability to change
  - Filter behavior
    - Dynamic stream rates
    - Data-dependent control flow
  - Execution environment
    - Stationary vs. moving
    - Noise
  - Resource availability
    - Multiple applications concurrently executing
- Worst case conditions

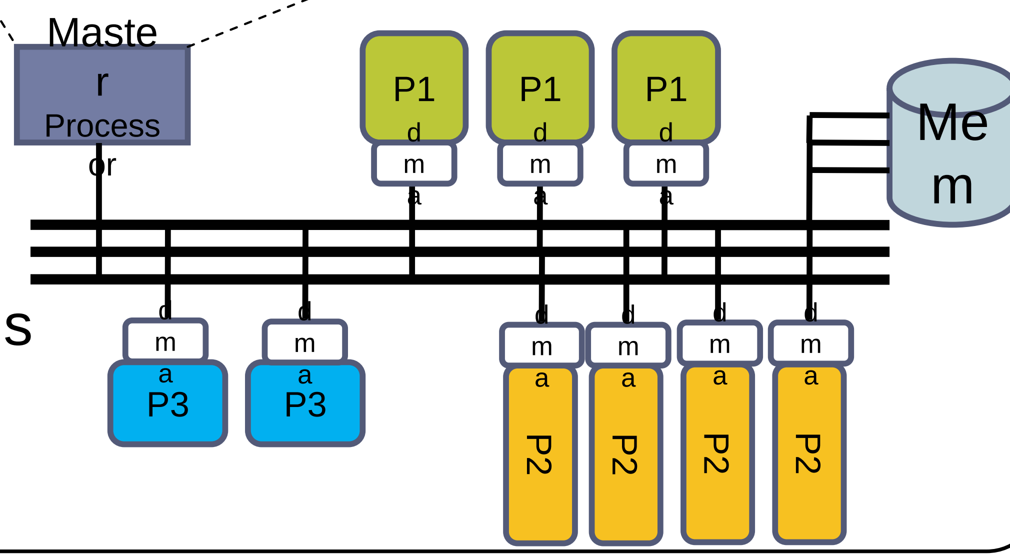
Dynamic Approach



- Scoreboard
- Resource usage
- Memory map

- Commands
- Execution history

Dynamic stream scheduler

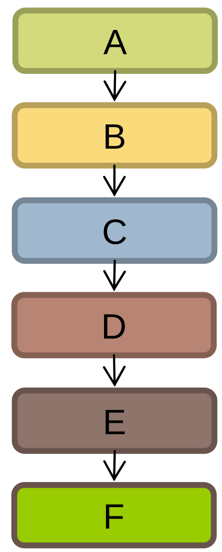


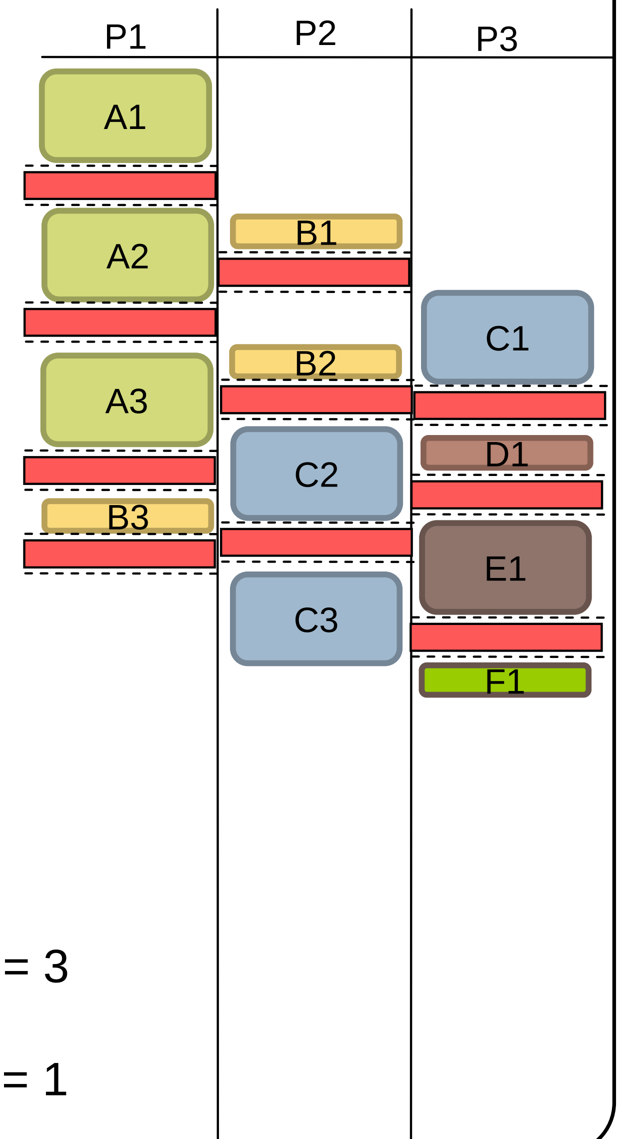
- Similar to superscalar scheduler
- Global memory serves as central repository for all stream data
- Master processor issues commands
- Focus on mapping/scheduling

Dynamic Example

- Use a heuristic functions to select the next filter to run on a free processor

- Each filter after completion notifies the main processor





$W_A = W_C = W_E = 3$

$W_B = W_D = W_F = 1$

- The main processor

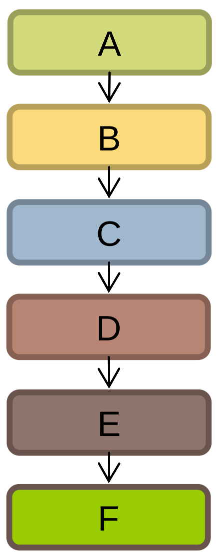
Tradeoffs in Dynamic Approach

- Execute filters when inputs are available
- Advantages:
  - Responsive to resource availability and filter variability
  - Lightweight algorithm
- Disadvantages:
  - Exposes DMA latency
  - Simple management of local buffers required
  - Scalability

Can We Have Our Cake and Eat It Too?

- Cake
  - Distributed static schedule for typical scenario
  - Relocatable filters/DMA commands
- Eat
  - Greedy folding at run-time
  - Space folding – Intra-stage filter migration between cores
  - Time folding – Extend/contract stage length
- Maintain same pipeline flow but with different workers

Adaptive Approach Idea



$W_A = W_C = W_E = 3$

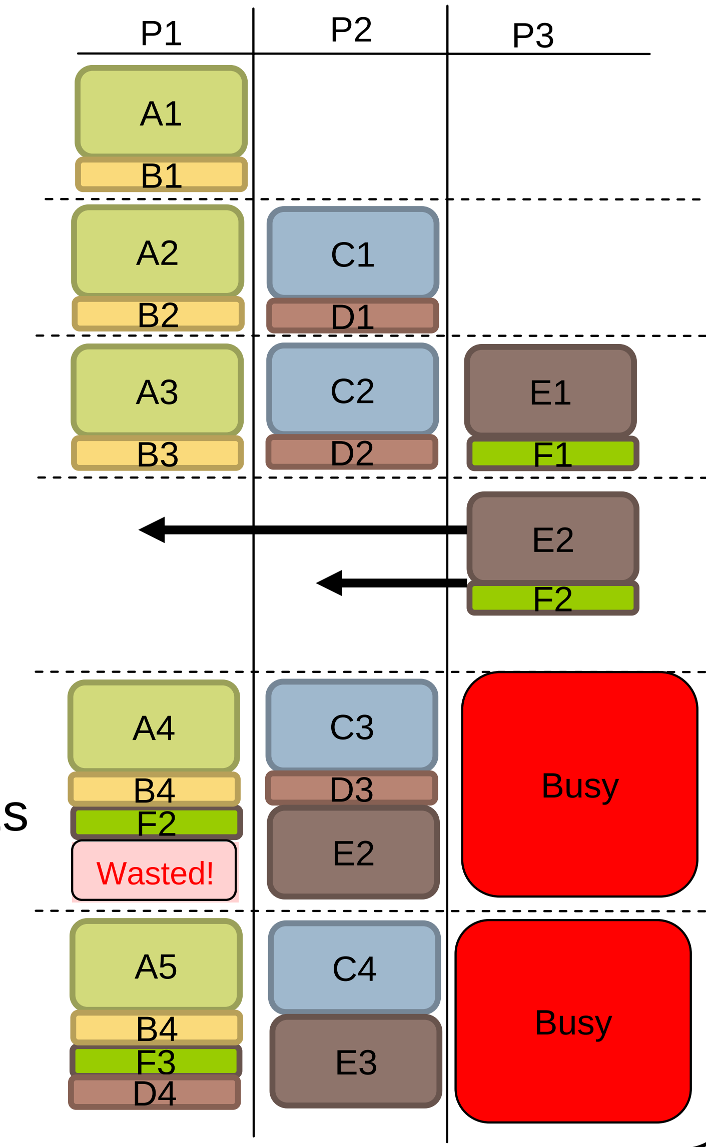
$W_B = W_D = W_F = 1$

Common case schedule

P3 unavailable → reschedule

Resume execution with reduced PEs

Further refinement



Unsolved Issues and Final Thoughts

- Memory management
  - – Folding memory spaces
  - – Spill to global memory
- DMA transfers
  - – Run-time configurable source/target
- Adaptive streaming
  - – Static baseline schedule for performance efficiency
  - – Dynamic adjustment for dealing with run-time events
  - – Will this work, does folding loose too much