Look Into Details: The Benefits of Fine-Grain Streaming Buffer Analysis

Mohammad H. Foroozannejad
Matin Hashemi
Trevor L. Hodges
Soheil Ghiasi

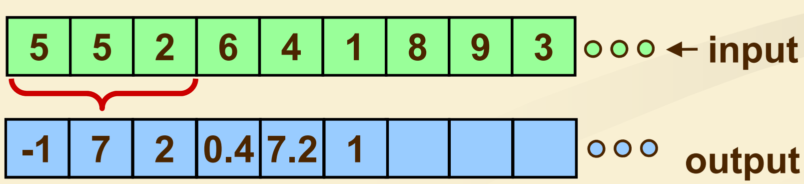
University of California, Davis, CA

Streaming Applications

- Widespread




  - Cell phones , mp3 players, video conference, real-time encryption, graphics, HDTV editing hyperspectral imaging, cellular base stations
- Definition
  - Infinite sequence of data items
  - At any given time, operates on a small window of this sequence
  - Moves forward in data space

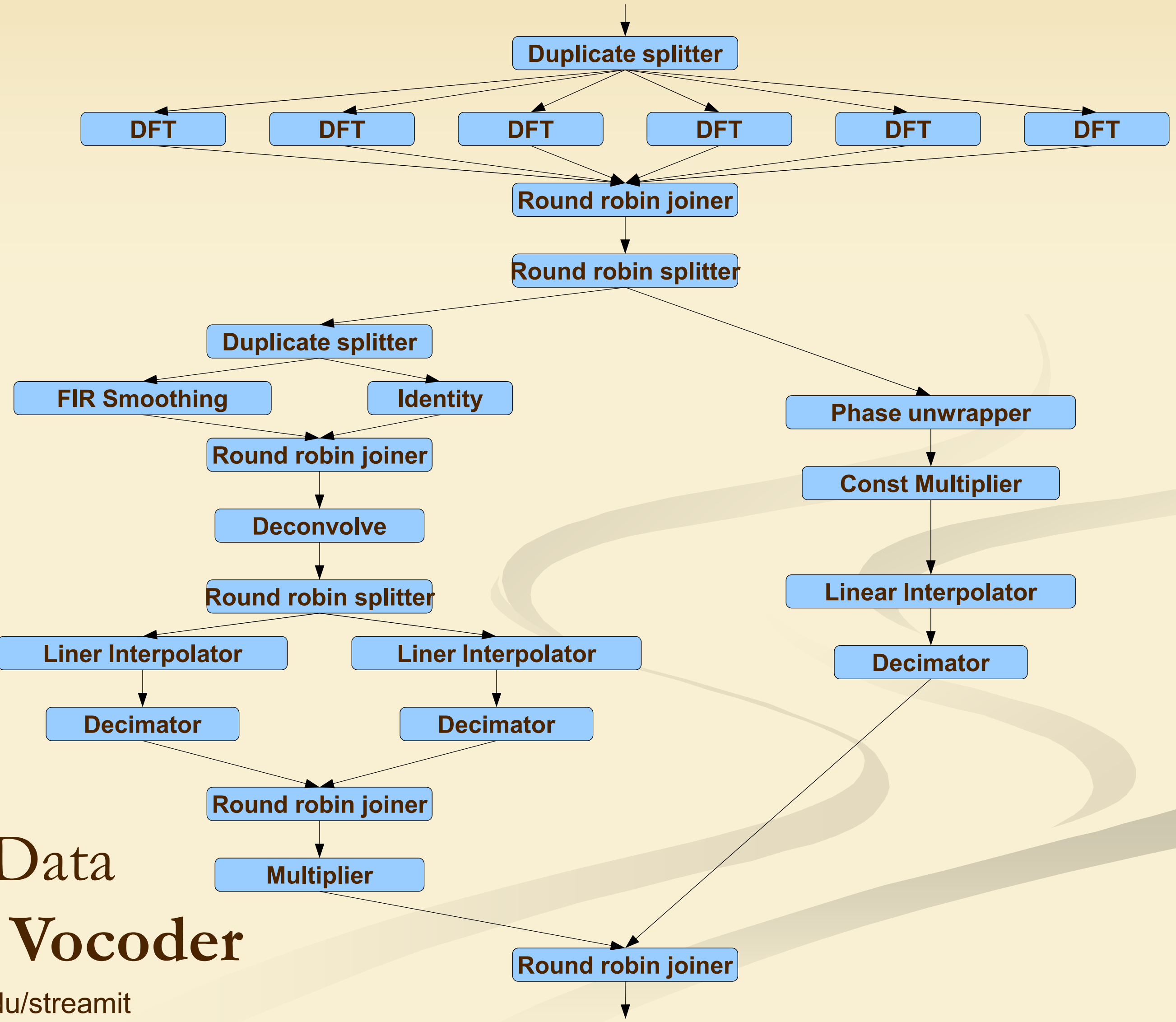



```
//53° around the z axis
const R[3][3]={
      {0.6,-0.8, 0.0},
      {0.8, 0.6, 0.0},
      {0.0, 0.0, 1.0}}
Rotation3D {
 for (i=0; i<3; i++)
  for (j=0; j<3; j++)
   B[i] += R[i][j] * A[j]
 }
```

Application Model

- Data Flow Graph
  - Vertices or Actors
    - functions, computations
  - Edges
    - data dependency, communication between actors
  - Execution Model
    - any actor can perform its computation whenever all necessary input data are available on incoming edges.

Application Model



- An example Data Flow Graph: **Vocoder**

http://www.cag.csail.mit.edu/streamit

Application Model

V= { A, B, C, D} E= { A_B, B_C, C_D }

S1: 6A 4B 4C 1D Flat SAS

S2: 2(3A 2(1B 1C)) 1D SAS

S3: 2A 1B 1C 4A 3(1B 1C) 1D non-SAS

sink (A_B) = B src (A_B) = A

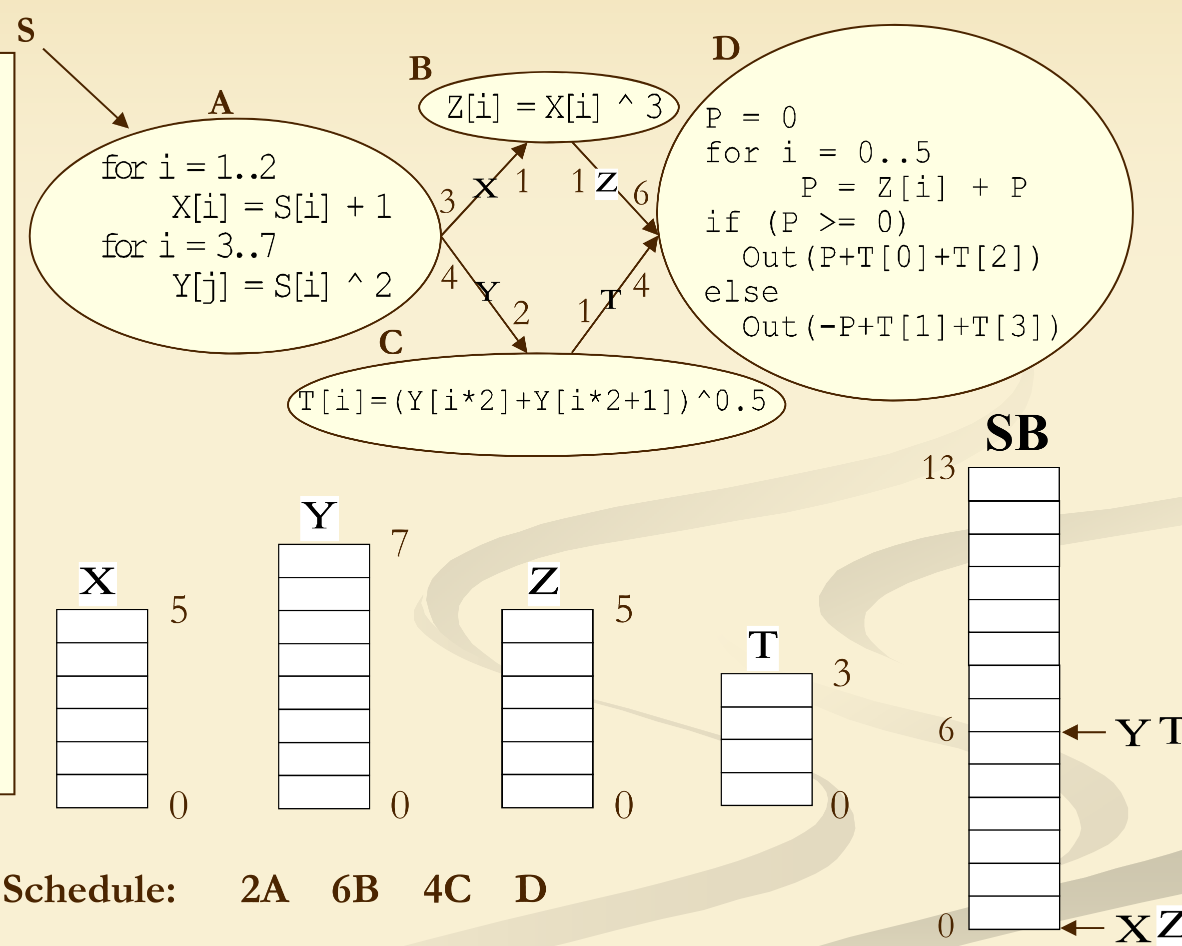
cns (A_B) = 6 prod (A_B) = 4

q = [6, 4, 4, 1] q[A] = 6 q[B] = 4

- **SDF** (Synchronous Data Flow Graph) is one special case
  - Fixed input and output rates on the edges
  - statically schedulable

Software Synthesis from SDF

```
while(1)
   for i = 0..1
      for j = 0..2
        X[i*3+j] = S[j] + 1
      for j = 3..6
        Y[i*4+j] = S[j] ^ 2

   for i = 0..5
      Z[i] = X[i] ^ 3

   for i = 0..3
      T[i]=(Y[i*2]+Y[i*2+1])^0.5

      P = 0
      for i = 0..5
        P = Z[i] + P
      if (P >= 0)
        Out(P+T[0]+T[2])
      else
        Out(-P+T[1]+T[3])
end While
```

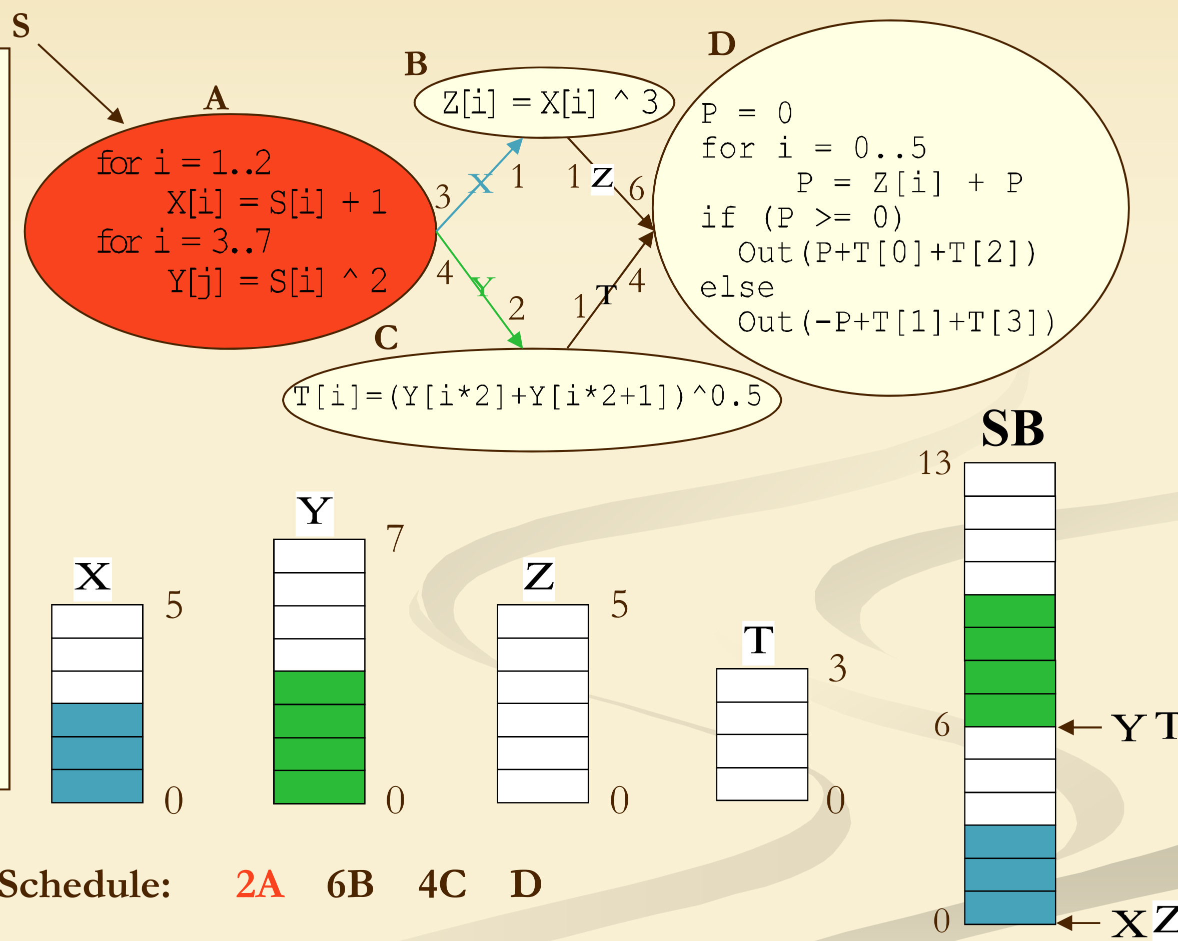


Schedule: **2A 6B 4C D**

Firing Sequence: **A A B B B B B B C C C C D**

Software Synthesis from SDF

```
while(1)
   for i = 0..1
      for j = 0..2
        X[i*3+j] = S[j] + 1
      for j = 3..6
        Y[i*4+j] = S[j] ^ 2
   for i = 0..5
      Z[i] = X[i] ^ 3
   for i = 0..3
      T[i]=(Y[i*2]+Y[i*2+1])^0.5

      P = 0
      for i = 0..5
        P = Z[i] + P
      if (P >= 0)
        Out(P+T[0]+T[2])
      else
        Out(-P+T[1]+T[3])
end While
```

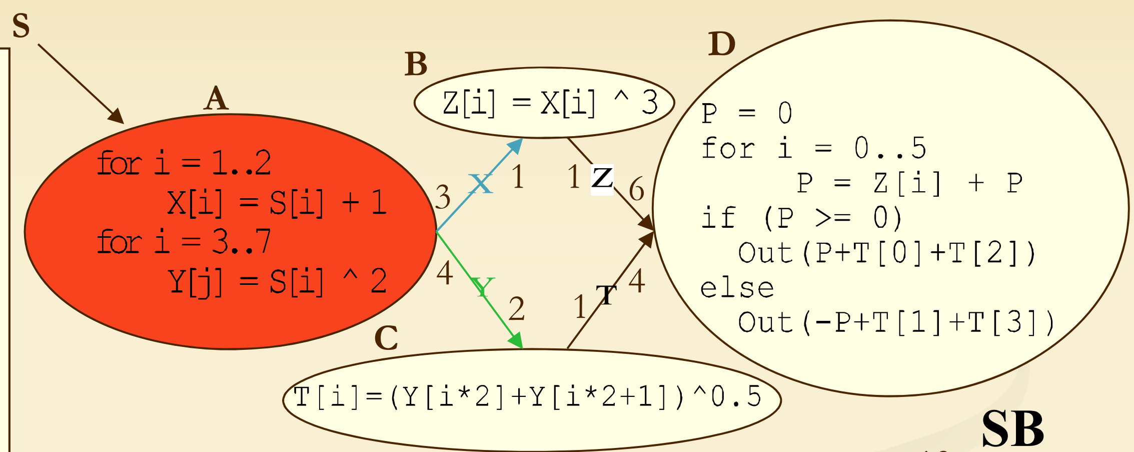


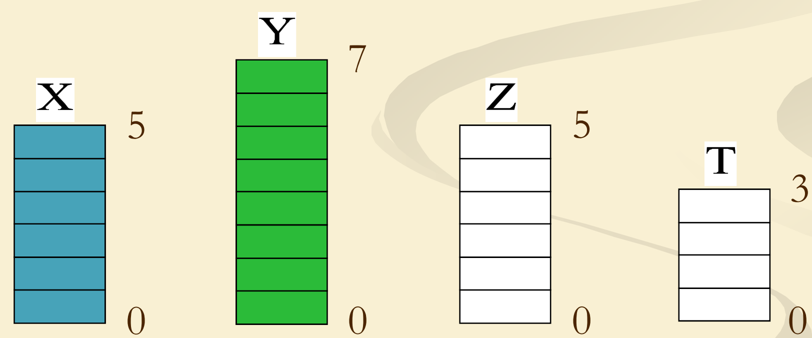
Schedule: 2A 6B 4C D

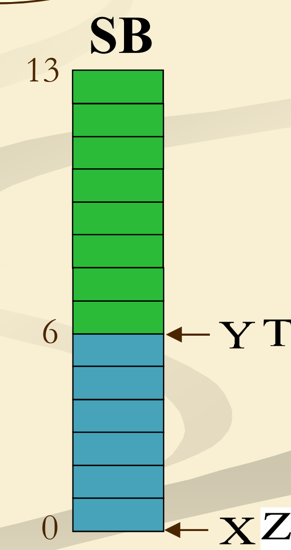
Firing Sequence: A A B B B B B B C C C C D

Software Synthesis from SDF

```
while(1)
   for i = 0..1
      for j = 0..2
        X[i*3+j] = S[j] + 1
      for j = 3..6
        Y[i*4+j] = S[j] ^ 2
   for i = 0..5
      Z[i] = X[i] ^ 3
   for i = 0..3
      T[i]=(Y[i*2]+Y[i*2+1])^0.5
      P = 0
      for i = 0..5
        P = Z[i] + P
      if (P >= 0)
        Out(P+T[0]+T[2])
      else
        Out(-P+T[1]+T[3])
end While
```





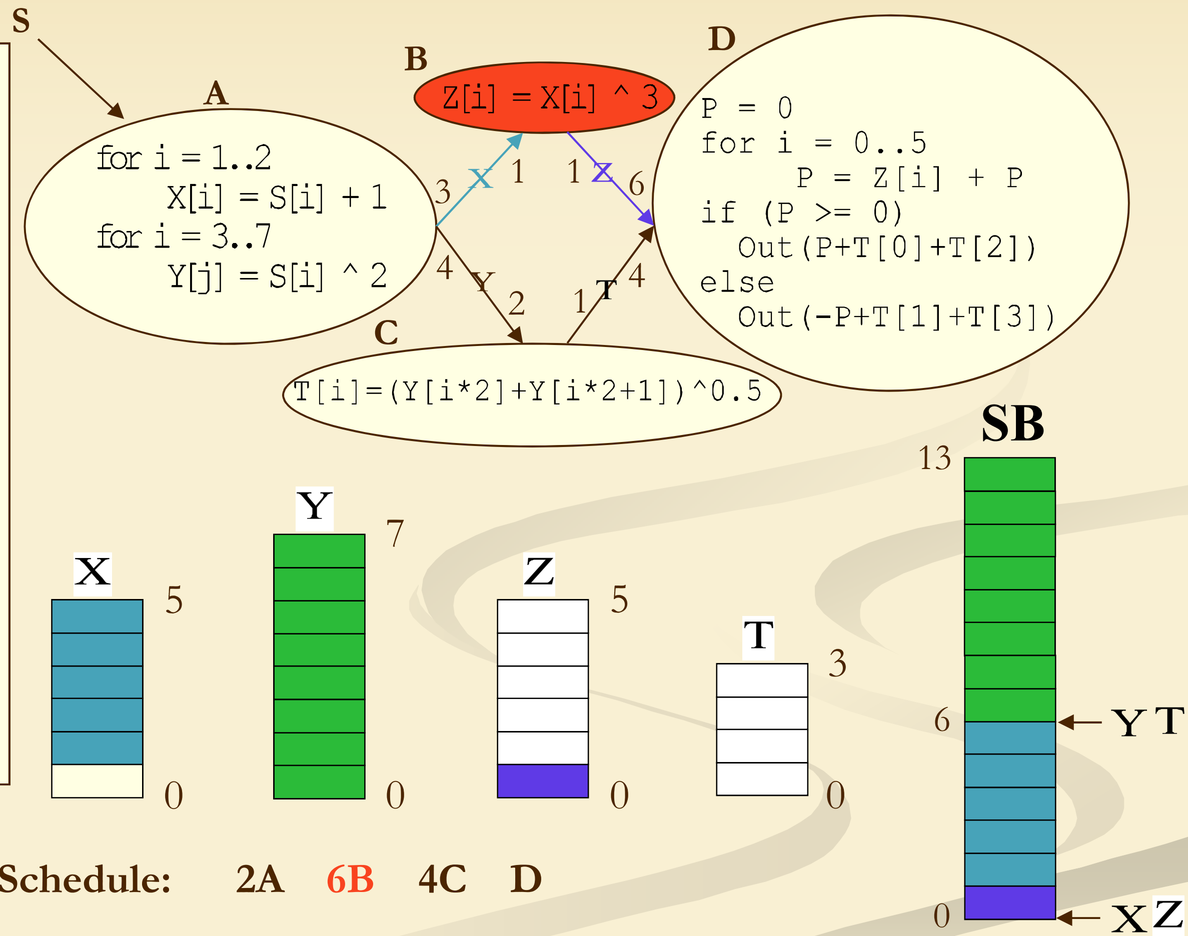


Schedule: 2A 6B 4C D

Firing Sequence: A A B B B B B B C C C C D

Software Synthesis from SDF

```
while(1)
   for i = 0..1
     for j = 0..2
       X[i*3+j] = S[j] + 1
     for j = 3..6
       Y[i*4+j] = S[j] ^ 2
   for i = 0..5
     Z[i] = X[i] ^ 3
   for i = 0..3
     T[i]=(Y[i*2]+Y[i*2+1])^0.5
     P = 0
     for i = 0..5
       P = Z[i] + P
     if (P >= 0)
       Out(P+T[0]+T[2])
     else
       Out(-P+T[1]+T[3])
end While
```

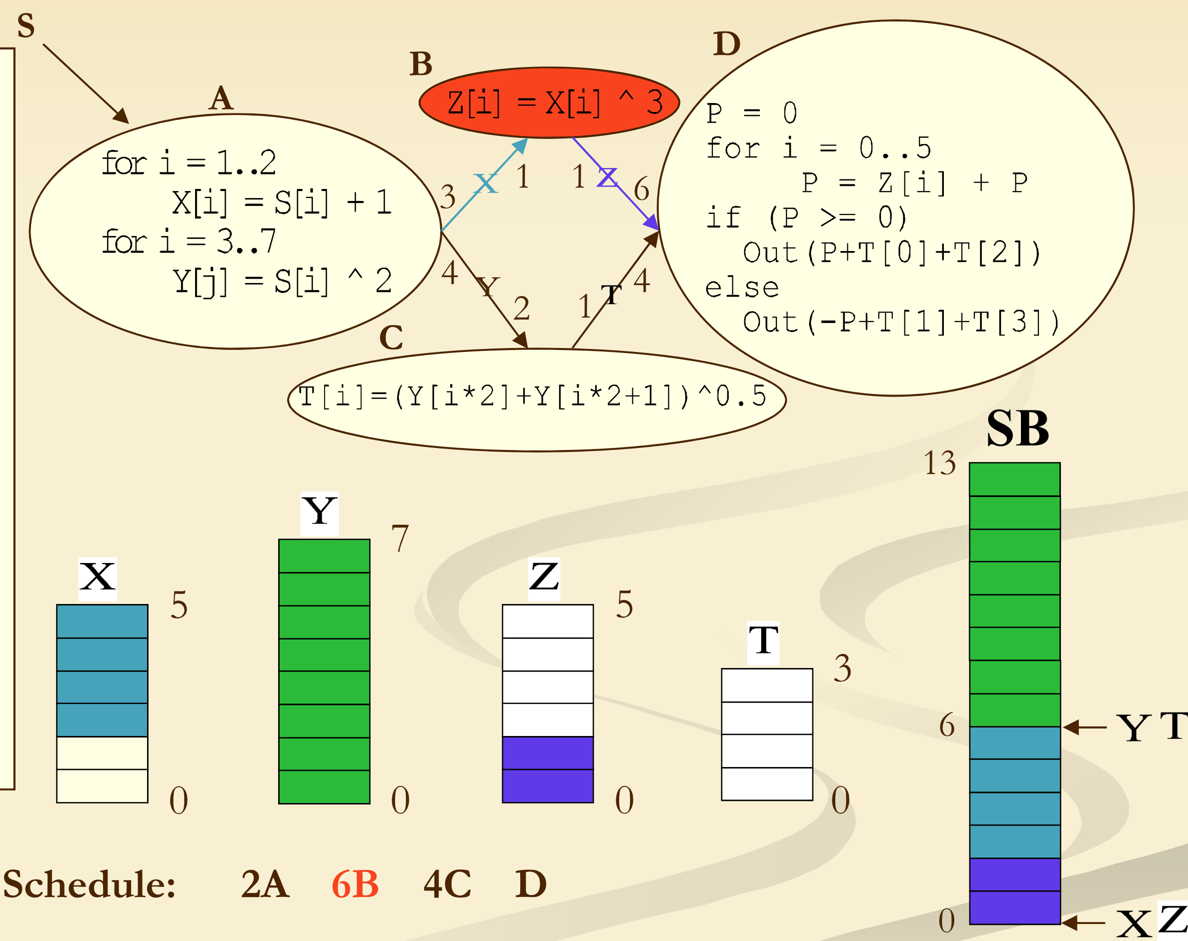


Schedule: 2A 6B 4C D

Firing Sequence: A A B B B B B B C C C C D

Software Synthesis from SDF

```
while(1)
   for i = 0..1
     for j = 0..2
       X[i*3+j] = S[j] + 1
     for j = 3..6
       Y[i*4+j] = S[j] ^ 2

   for i = 0..5
     Z[i] = X[i] ^ 3

   for i = 0..3
     T[i]=(Y[i*2]+Y[i*2+1])^0.5

     P = 0
     for i = 0..5
       P = Z[i] + P
     if (P >= 0)
       Out(P+T[0]+T[2])
     else
       Out(-P+T[1]+T[3])
end While
```

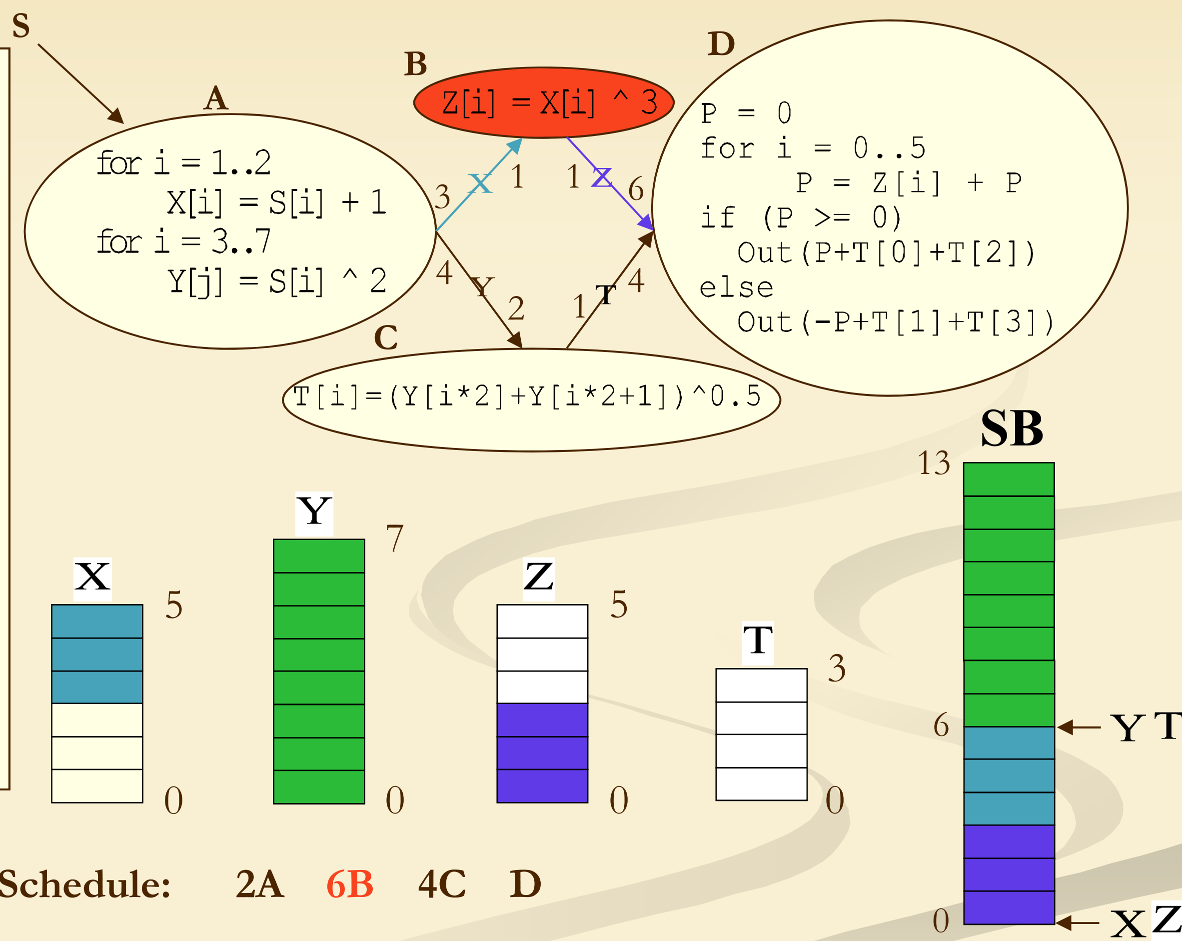


Schedule: 2A 6B 4C D

Firing Sequence: A A B B B B B B C C C C D

Software Synthesis from SDF

```
while(1)
   for i = 0..1
     for j = 0..2
       X[i*3+j] = S[j] + 1
     for j = 3..6
       Y[i*4+j] = S[j] ^ 2
   for i = 0..5
     Z[i] = X[i] ^ 3
   for i = 0..3
     T[i]=(Y[i*2]+Y[i*2+1])^0.5
     P = 0
     for i = 0..5
       P = Z[i] + P
     if (P >= 0)
       Out(P+T[0]+T[2])
     else
       Out(-P+T[1]+T[3])
 end While
```

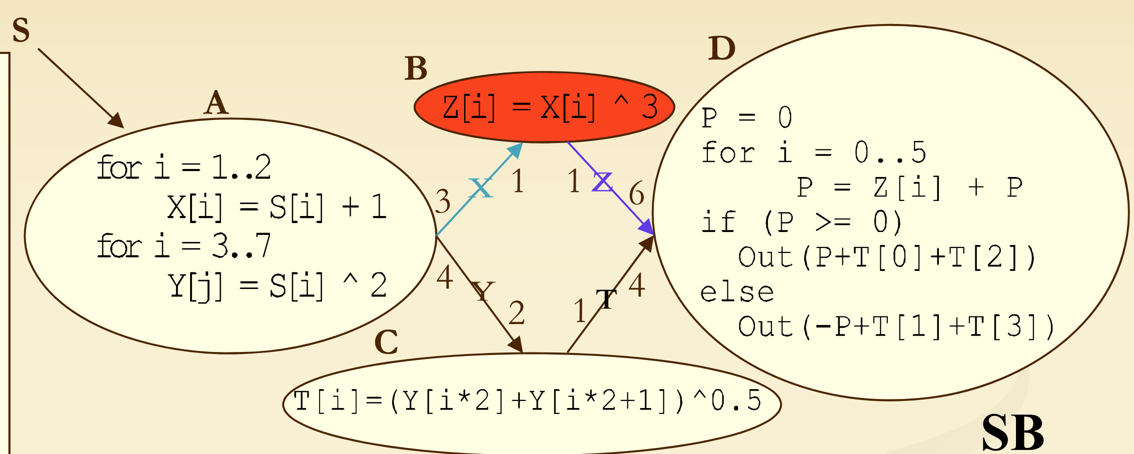


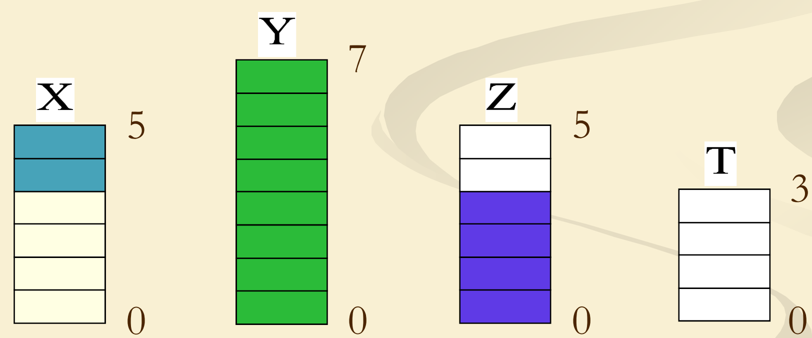
Schedule: 2A 6B 4C D

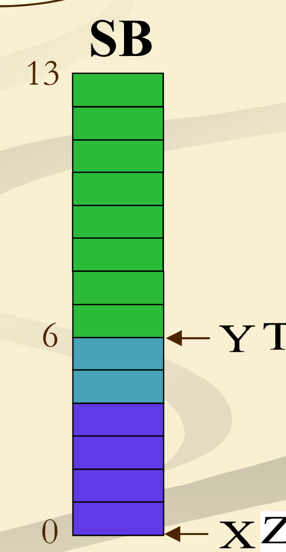
Firing Sequence: A A B B B B B B C C C C D

Software Synthesis from SDF

```
while(1)
   for i = 0..1
      for j = 0..2
        X[i*3+j] = S[j] + 1
      for j = 3..6
        Y[i*4+j] = S[j] ^ 2

   for i = 0..5
      Z[i] = X[i] ^ 3

   for i = 0..3
      T[i]=(Y[i*2]+Y[i*2+1])^0.5

      P = 0
      for i = 0..5
        P = Z[i] + P
      if (P >= 0)
        Out(P+T[0]+T[2])
      else
        Out(-P+T[1]+T[3])
 end While
```





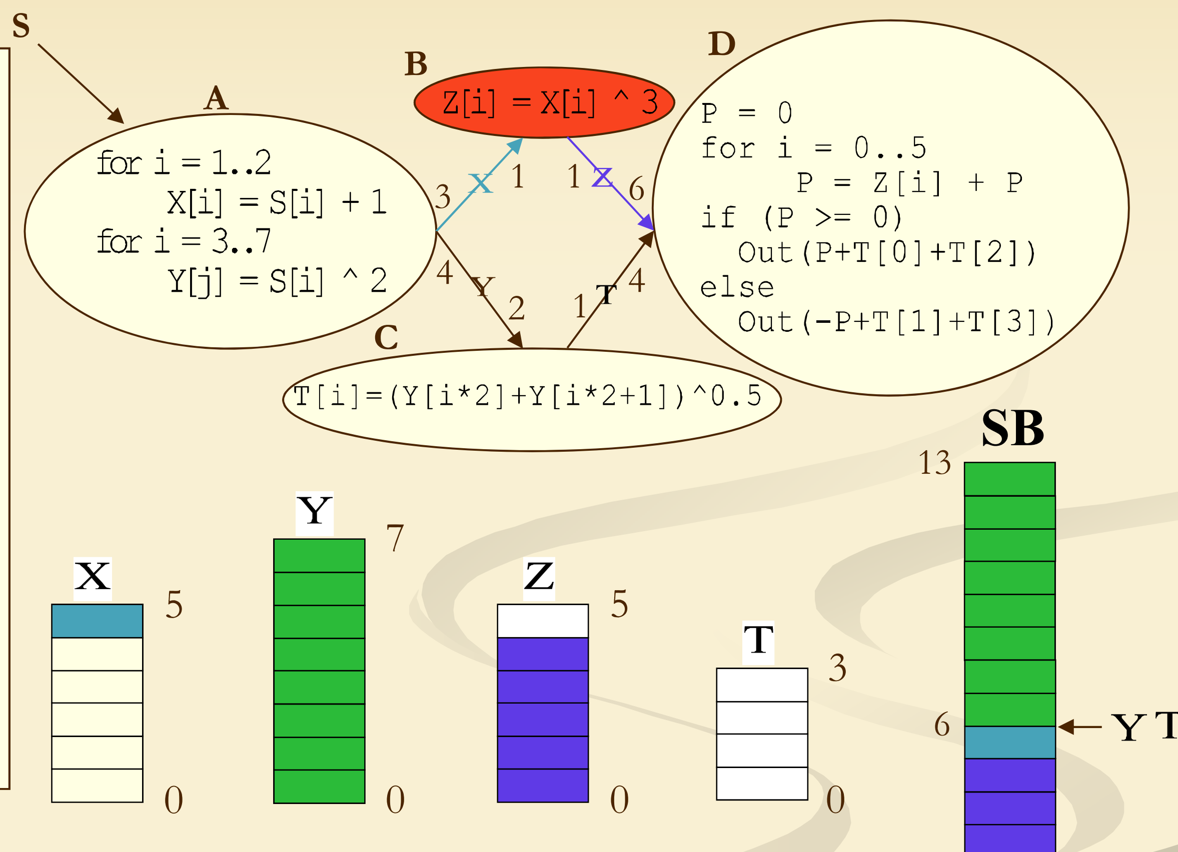


Schedule: **2A** **6B** **4C** **D**

Firing Sequence: **A A B B B B B B C C C C D**

Software Synthesis from SDF

```
while(1)
   for i = 0..1
     for j = 0..2
       X[i*3+j] = S[j] + 1
     for j = 3..6
       Y[i*4+j] = S[j] ^ 2
   for i = 0..5
     Z[i] = X[i] ^ 3
   for i = 0..3
     T[i]=(Y[i*2]+Y[i*2+1])^0.5
     P = 0
     for i = 0..5
       P = Z[i] + P
     if (P >= 0)
       Out(P+T[0]+T[2])
     else
       Out(-P+T[1]+T[3])
 end While
```

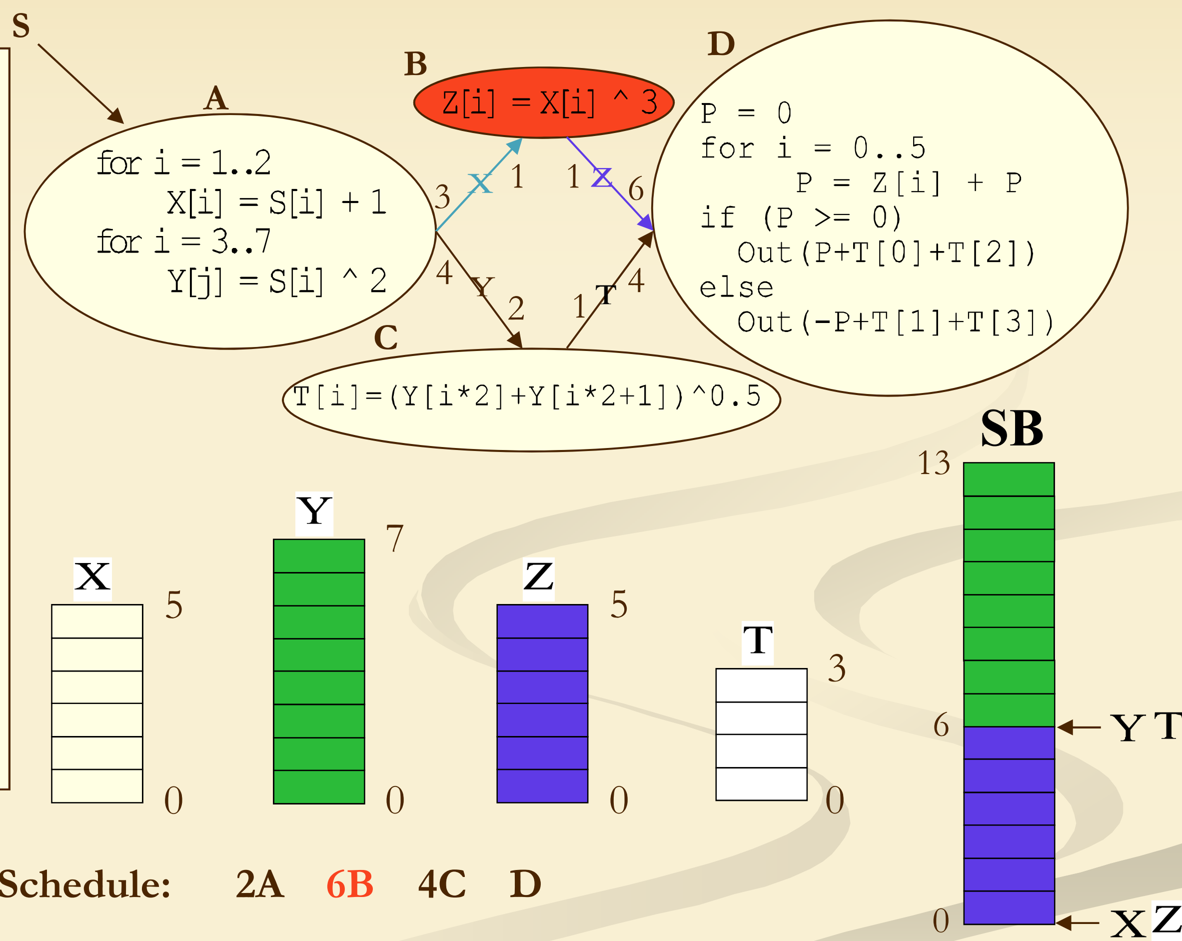


Schedule: 2A 6B 4C D

Firing Sequence: A A B B B B B B C C C C D

Software Synthesis from SDF

```
while(1)
   for i = 0..1
      for j = 0..2
        X[i*3+j] = S[j] + 1
      for j = 3..6
        Y[i*4+j] = S[j] ^ 2

   for i = 0..5
      Z[i] = X[i] ^ 3

   for i = 0..3
      T[i]=(Y[i*2]+Y[i*2+1])^0.5

      P = 0
      for i = 0..5
        P = Z[i] + P
      if (P >= 0)
        Out(P+T[0]+T[2])
      else
        Out(-P+T[1]+T[3])
end While
```

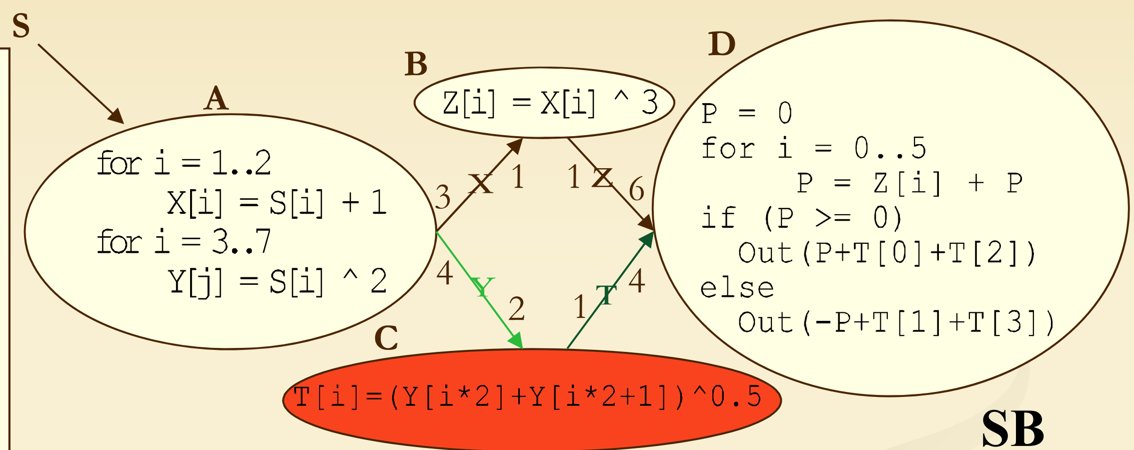


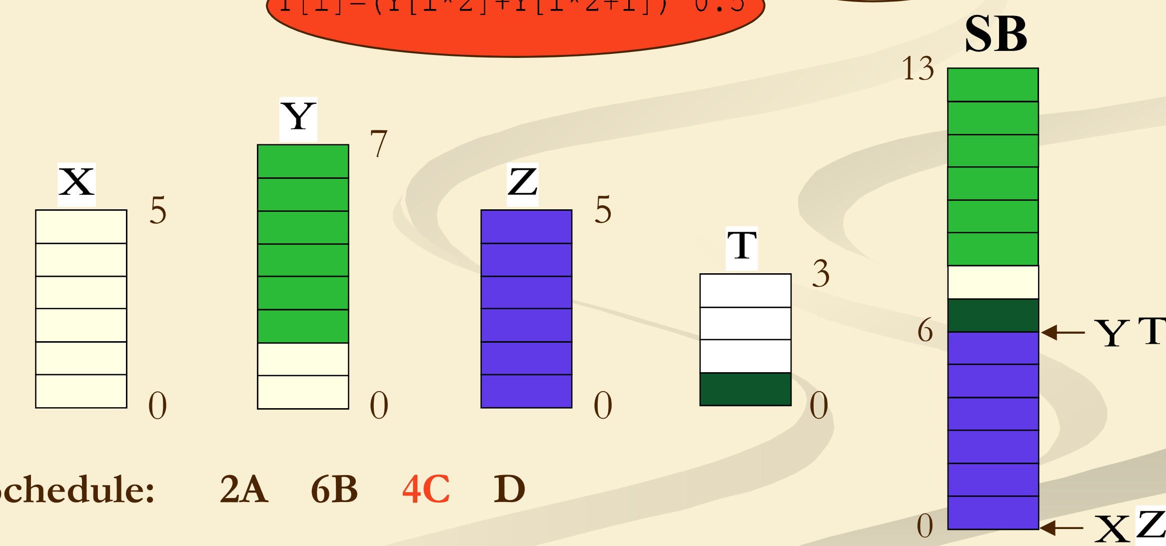
Schedule: **2A** **6B** **4C** **D**

Firing Sequence: **A A B B B B B B C C C C D**

Software Synthesis from SDF

```
while(1)
   for i = 0..1
      for j = 0..2
        X[i*3+j] = S[j] + 1
      for j = 3..6
        Y[i*4+j] = S[j] ^ 2

   for i = 0..5
      Z[i] = X[i] ^ 3

   for i = 0..3
      T[i]=(Y[i*2]+Y[i*2+1])^0.5

      P = 0
      for i = 0..5
        P = Z[i] + P
      if (P >= 0)
        Out(P+T[0]+T[2])
      else
        Out(-P+T[1]+T[3])
end While
```



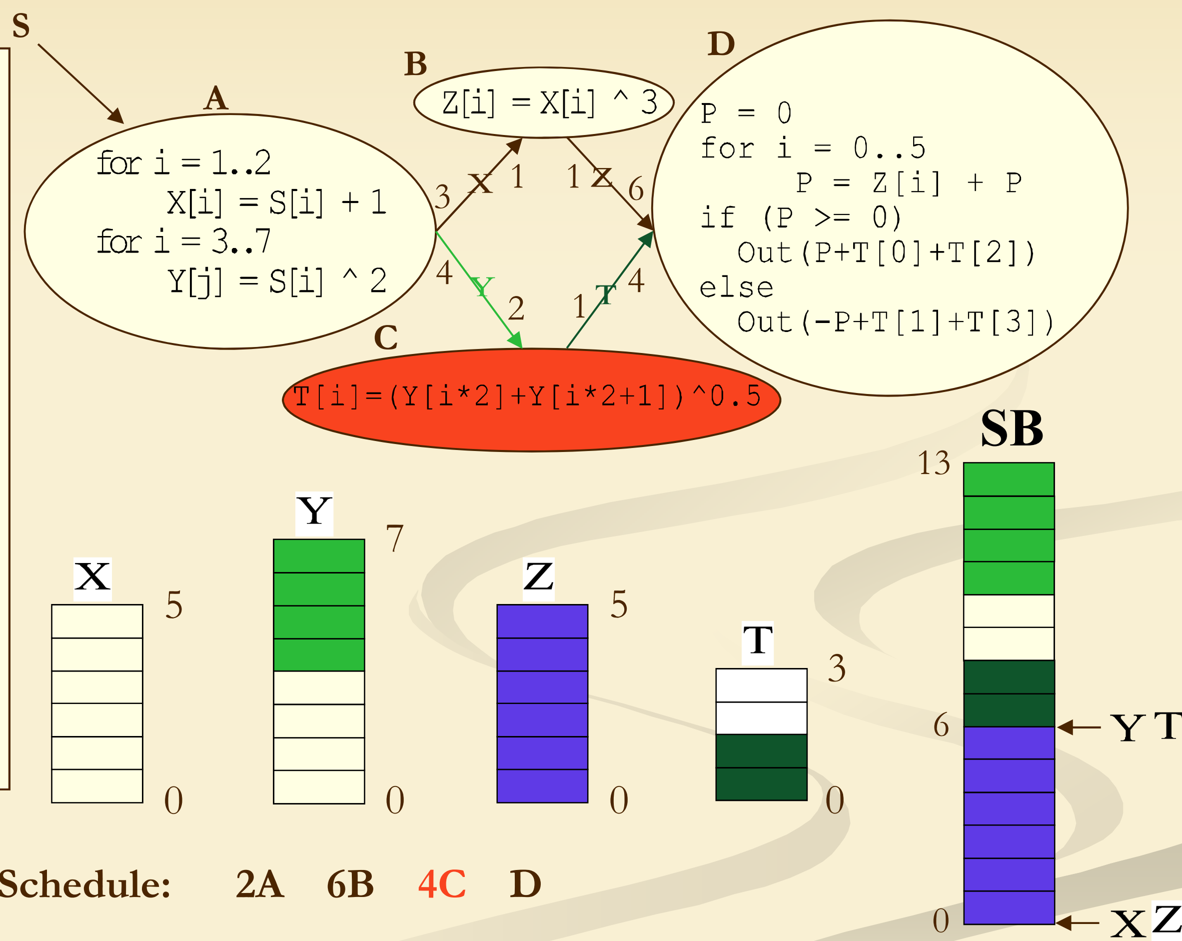


Schedule: **2A** **6B** **4C** **D**

Firing Sequence: **A A B B B B B B C C C C D**

Software Synthesis from SDF

```
while(1)
   for i = 0..1
      for j = 0..2
        X[i*3+j] = S[j] + 1
      for j = 3..6
        Y[i*4+j] = S[j] ^ 2

   for i = 0..5
      Z[i] = X[i] ^ 3

   for i = 0..3
      T[i]=(Y[i*2]+Y[i*2+1])^0.5

      P = 0
      for i = 0..5
        P = Z[i] + P
      if (P >= 0)
        Out(P+T[0]+T[2])
      else
        Out(-P+T[1]+T[3])
 end While
```

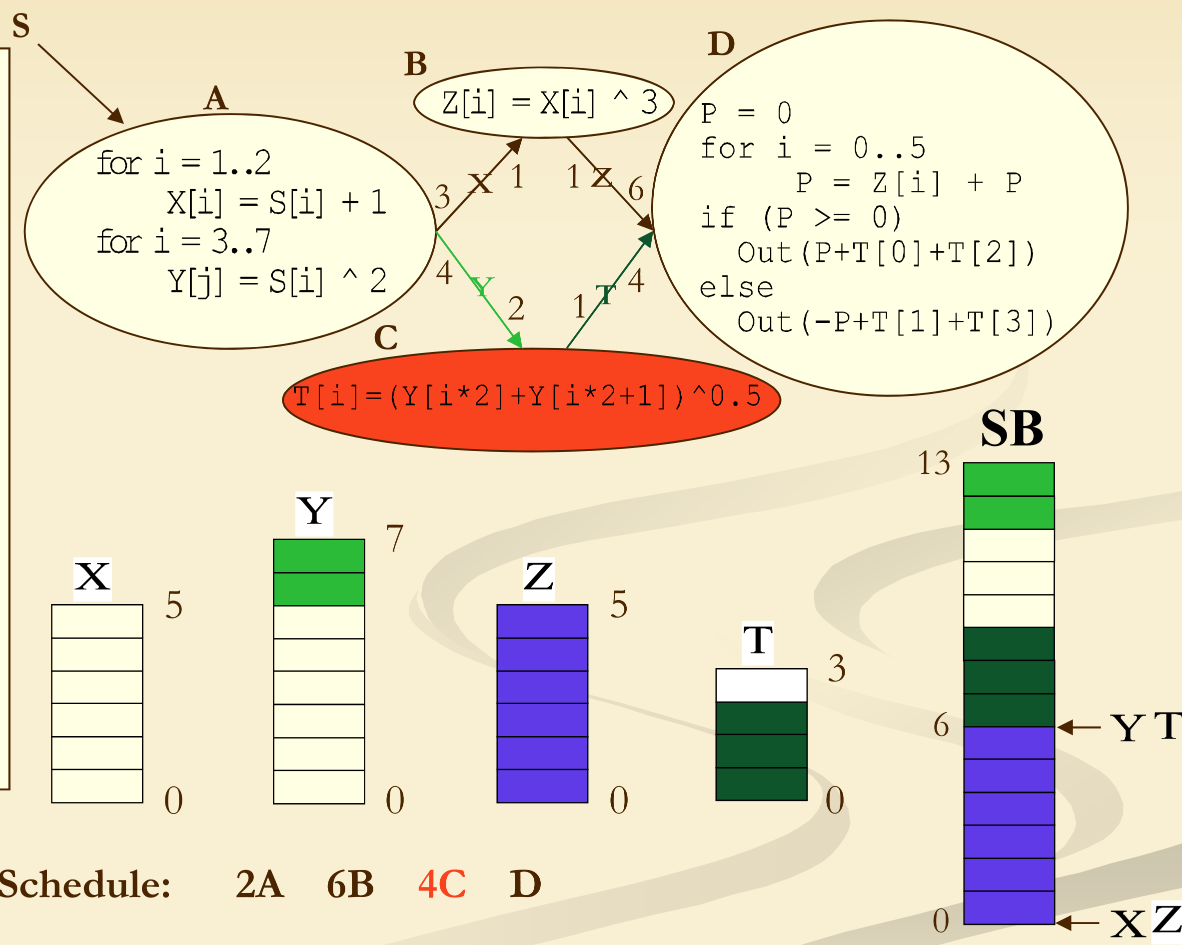


Schedule: 2A 6B 4C D

Firing Sequence: A A B B B B B B C C C C D

Software Synthesis from SDF

```
while(1)
   for i = 0..1
      for j = 0..2
        X[i*3+j] = S[j] + 1
      for j = 3..6
        Y[i*4+j] = S[j] ^ 2
   for i = 0..5
      Z[i] = X[i] ^ 3
   for i = 0..3
      T[i]=(Y[i*2]+Y[i*2+1])^0.5
      P = 0
      for i = 0..5
        P = Z[i] + P
      if (P >= 0)
        Out(P+T[0]+T[2])
      else
        Out(-P+T[1]+T[3])
 end While
```

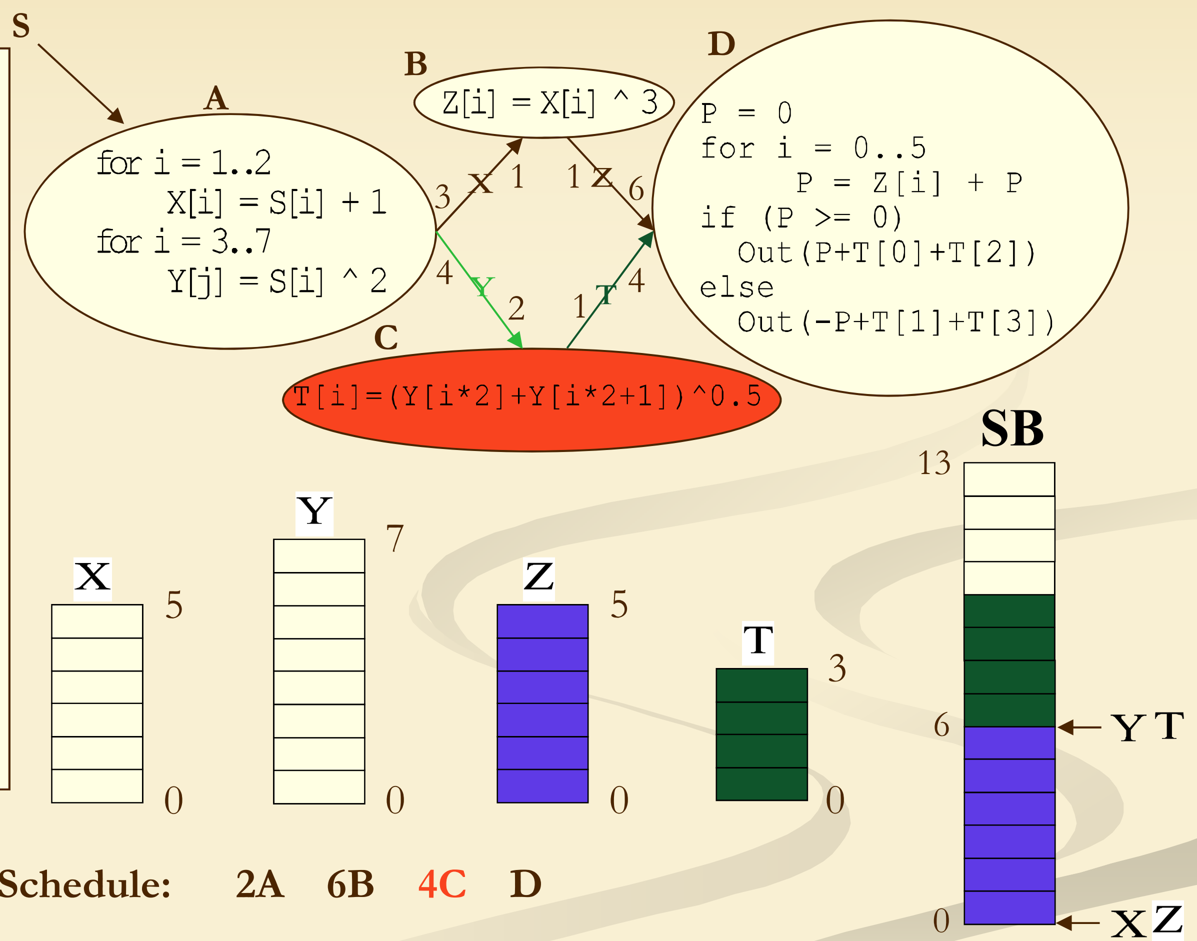


Schedule: 2A 6B 4C D

Firing Sequence: A A B B B B B B C C C C D

Software Synthesis from SDF

```
while(1)
   for i = 0..1
      for j = 0..2
        X[i*3+j] = S[j] + 1
      for j = 3..6
        Y[i*4+j] = S[j] ^ 2

   for i = 0..5
      Z[i] = X[i] ^ 3

   for i = 0..3
      T[i]=(Y[i*2]+Y[i*2+1])^0.5

      P = 0
      for i = 0..5
        P = Z[i] + P
      if (P >= 0)
        Out(P+T[0]+T[2])
      else
        Out(-P+T[1]+T[3])
end While
```

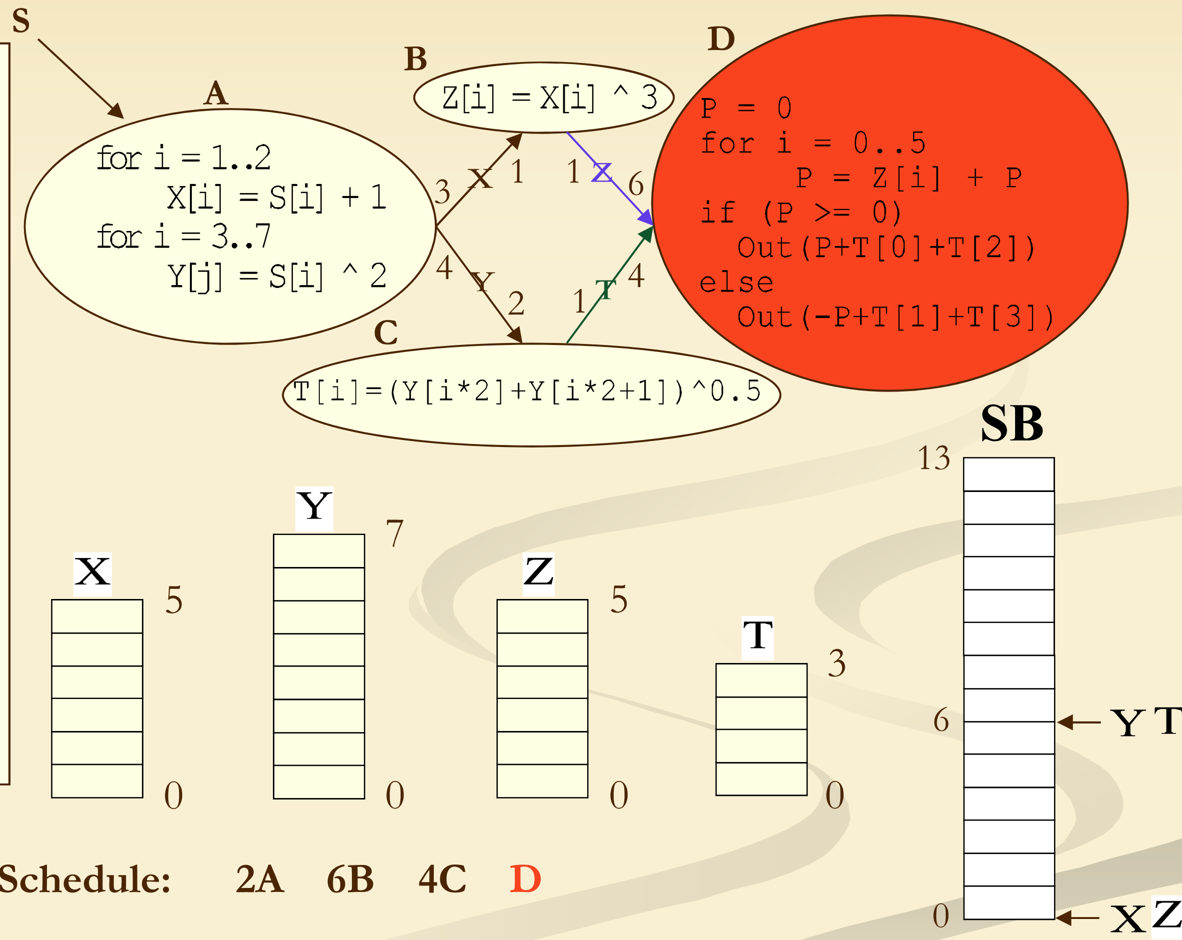


Schedule: 2A 6B 4C D

Firing Sequence: A A B B B B B B C C C C D

Software Synthesis from SDF

```
while(1)
   for i = 0..1
      for j = 0..2
        X[i*3+j] = S[j] + 1
      for j = 3..6
        Y[i*4+j] = S[j] ^ 2

   for i = 0..5
      Z[i] = X[i] ^ 3

   for i = 0..3
      T[i]=(Y[i*2]+Y[i*2+1])^0.5

      P = 0
      for i = 0..5
        P = Z[i] + P
      if (P >= 0)
        Out(P+T[0]+T[2])
      else
        Out(-P+T[1]+T[3])
end While
```



Schedule: **2A** **6B** **4C** **D**

Firing Sequence: **A A B B B B B B C C C C D**

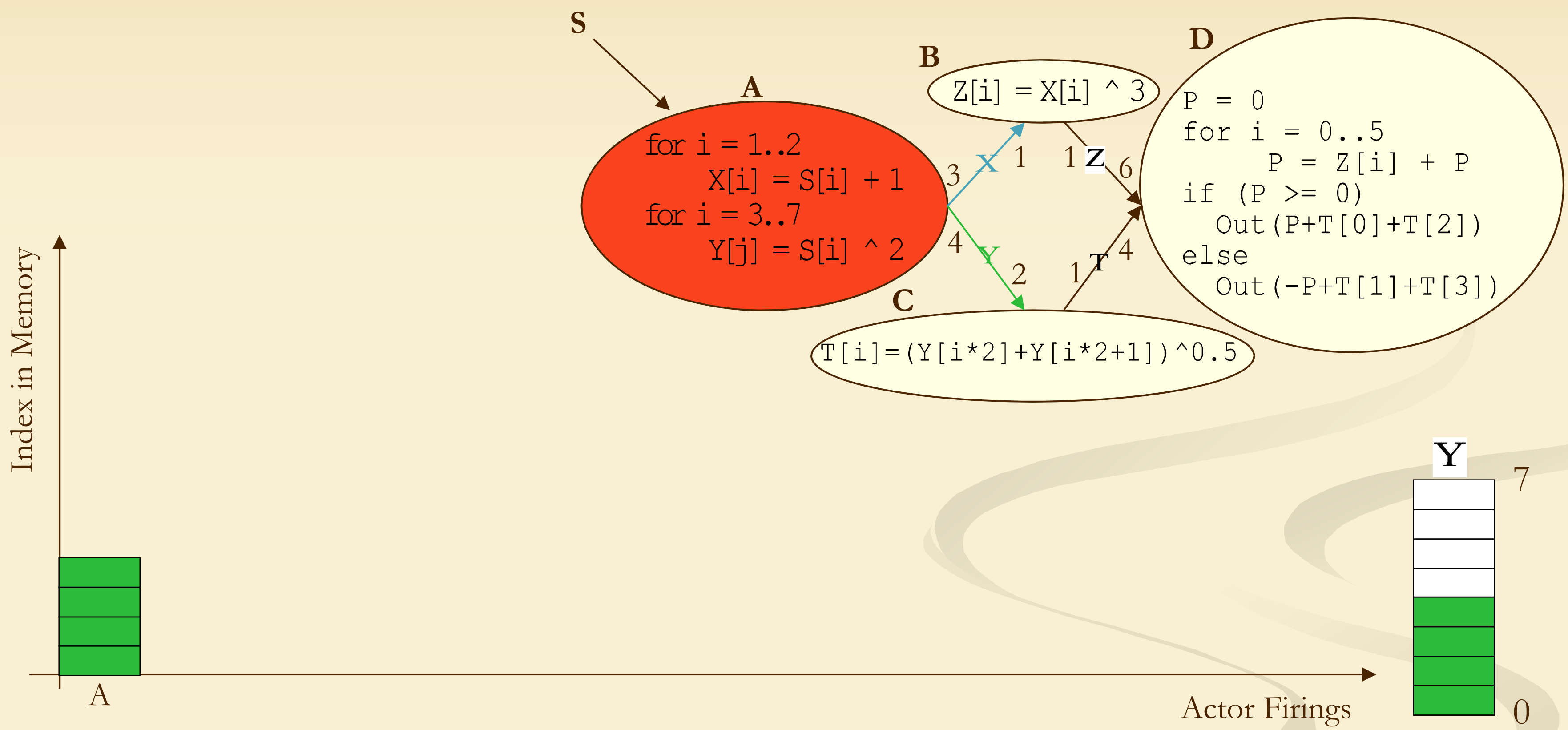
Shared Buffer Implementation

- Idea:
  - Most of the time channel buffers are completely or partially empty.
- Rules:
  - 1. No over-writing or reading another buffer's data.
  - 2. Statically allocated
  - 3. No re-allocation

Visualizing Buffer Analysis

- Tow dimensional plane
  - X-axis: Actor firings in the schedule (time)
  - Y-axis: Buffer location in the memory (space)
  - Filled Area: The range between Head and Tail indices
- Advantage:
  - Memory allocation problem can be viewed as a geometric layout instance
  - A solution is valid when the laid out buffers do not conflict in the time-memory plane.

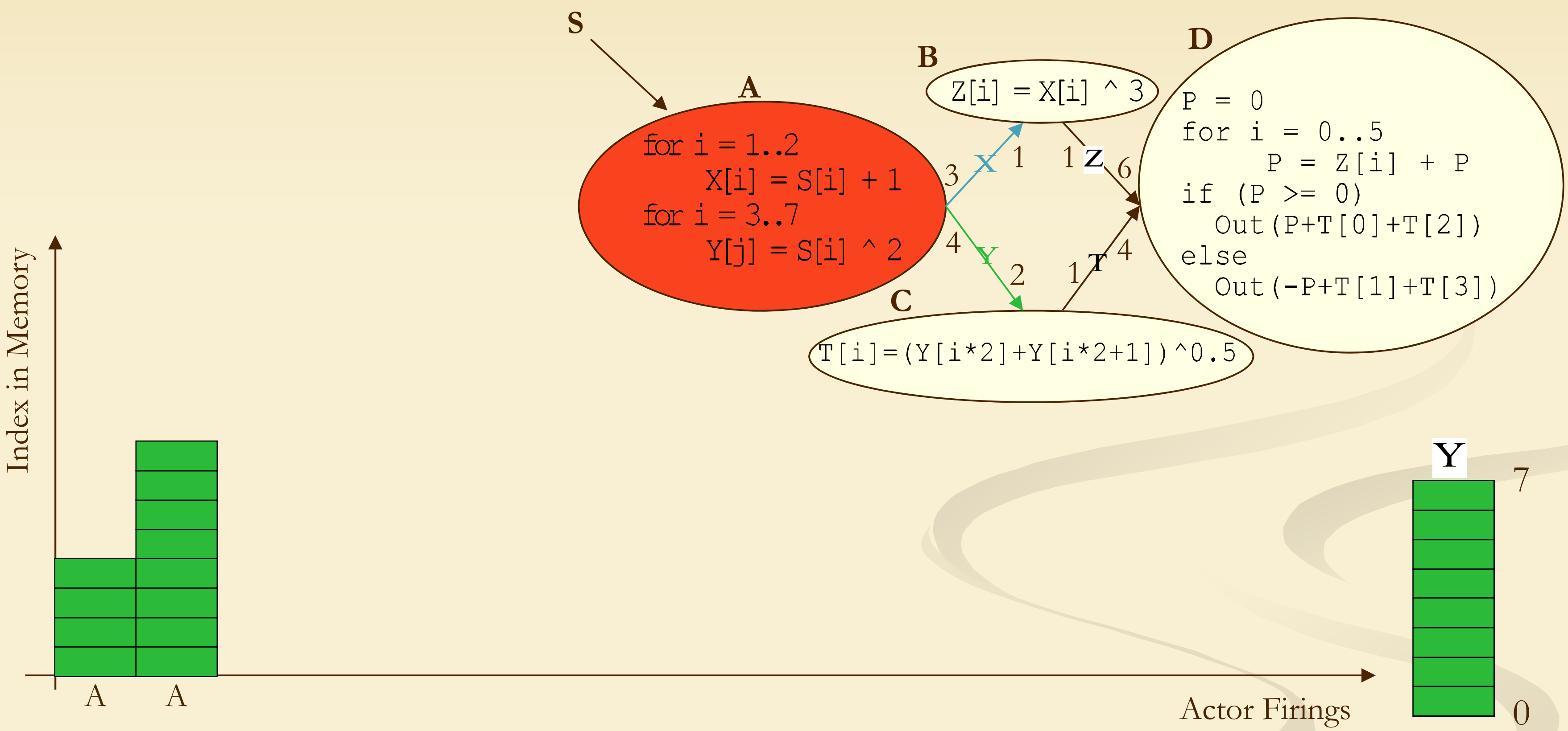
Visualizing Buffer Analysis



Schedule: 2A 6B 4C D

Firing Sequence: A A B B B B B B C C C C D

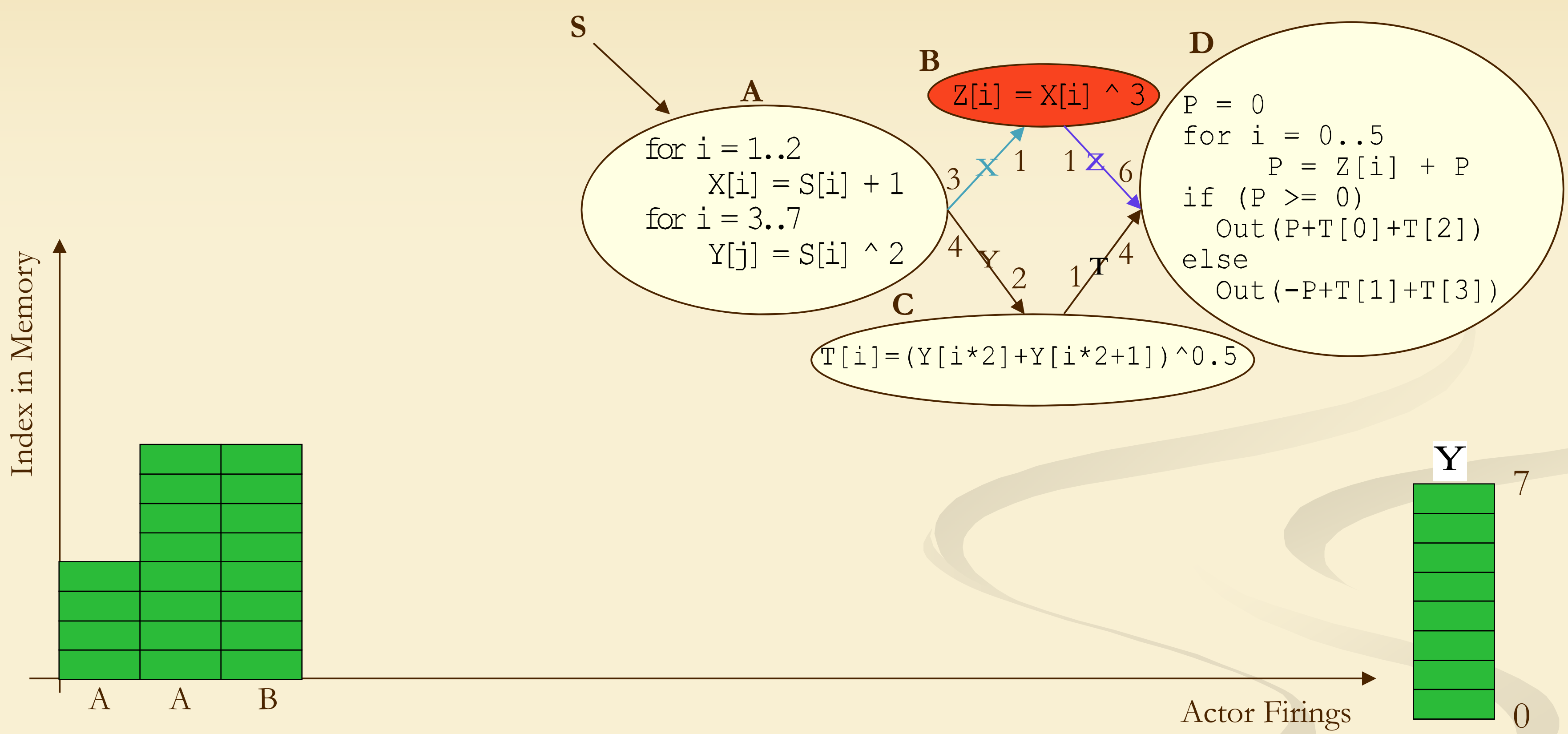
Visualizing Buffer Analysis



Schedule: 2A 6B 4C D

Firing Sequence: A A B B B B B B C C C C D

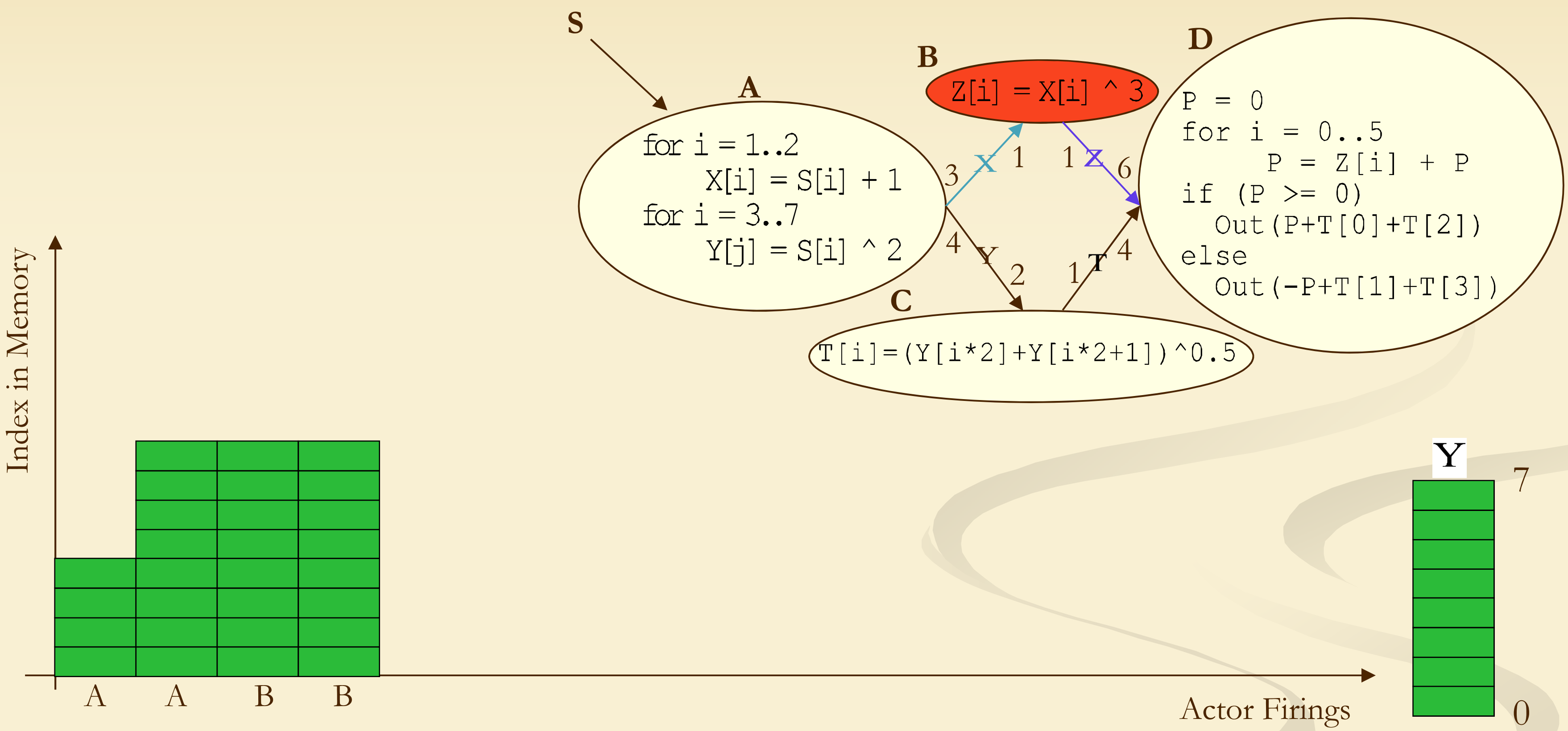
Visualizing Buffer Analysis



Schedule: **2A** **6B** **4C** **D**

Firing Sequence: **A A B B B B B B C C C C D**

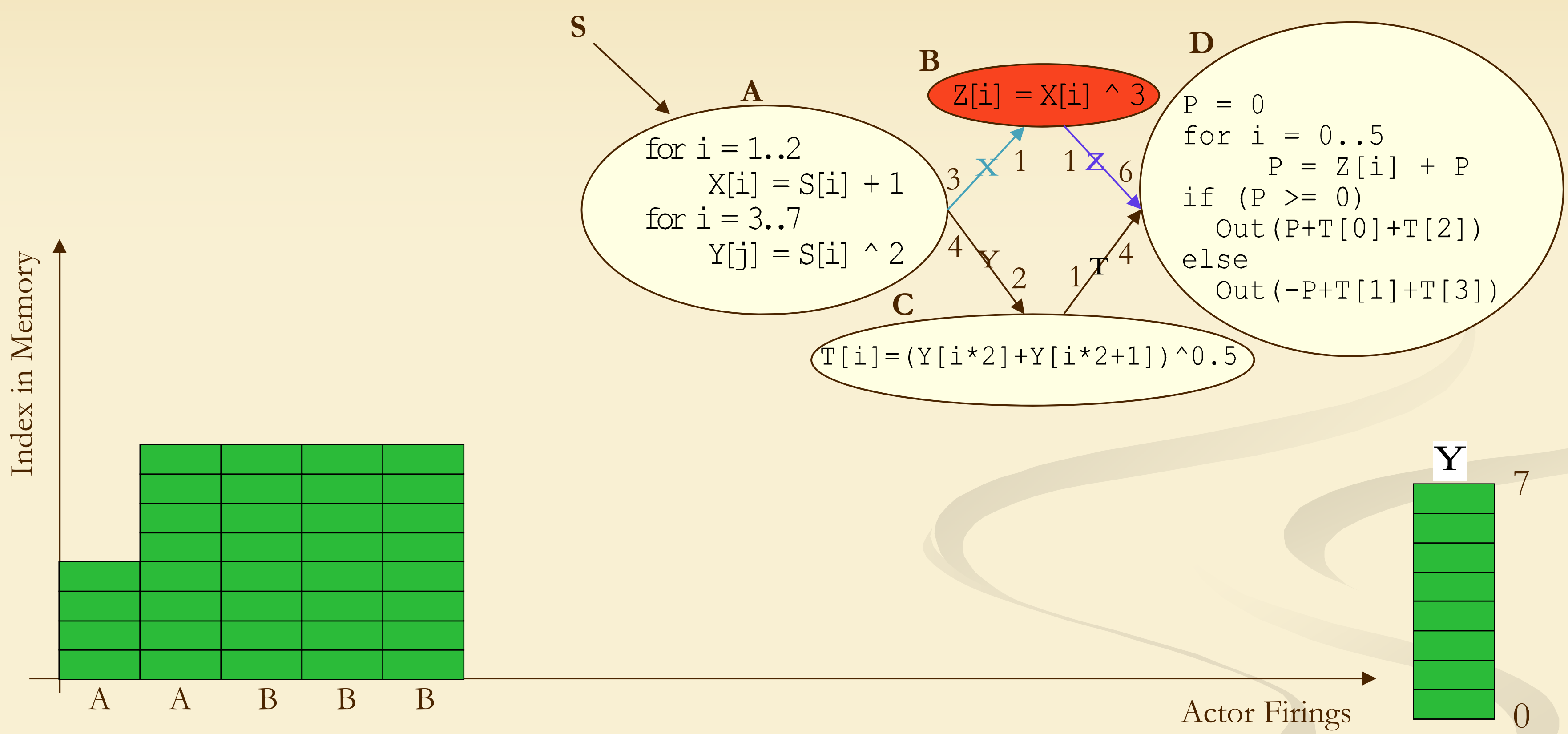
Visualizing Buffer Analysis



Schedule: 2A 6B 4C D

Firing Sequence: A A B B B B B B C C C C D

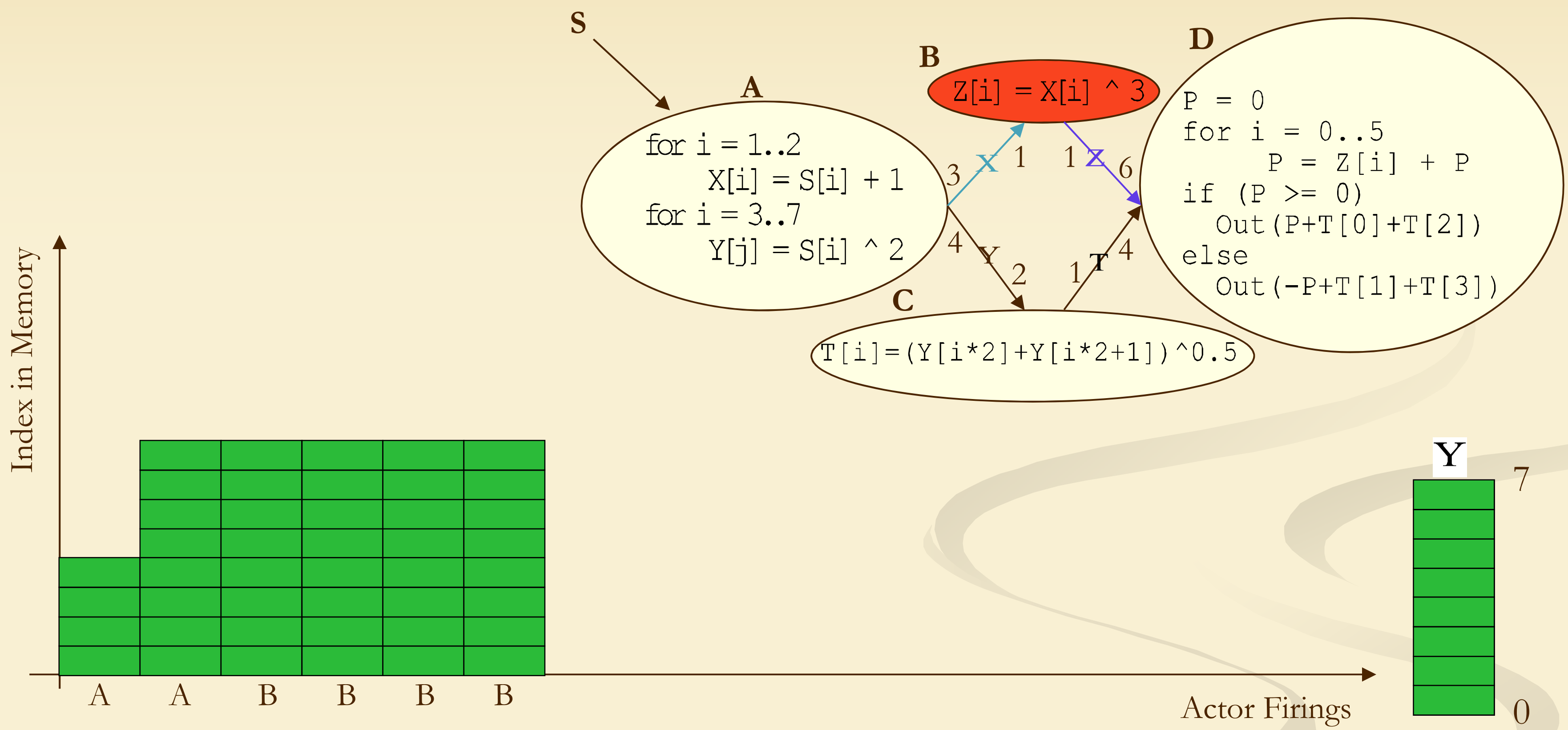
Visualizing Buffer Analysis



Schedule: **2A** **6B** **4C** **D**

Firing Sequence: **A A B B B B B B C C C C D**

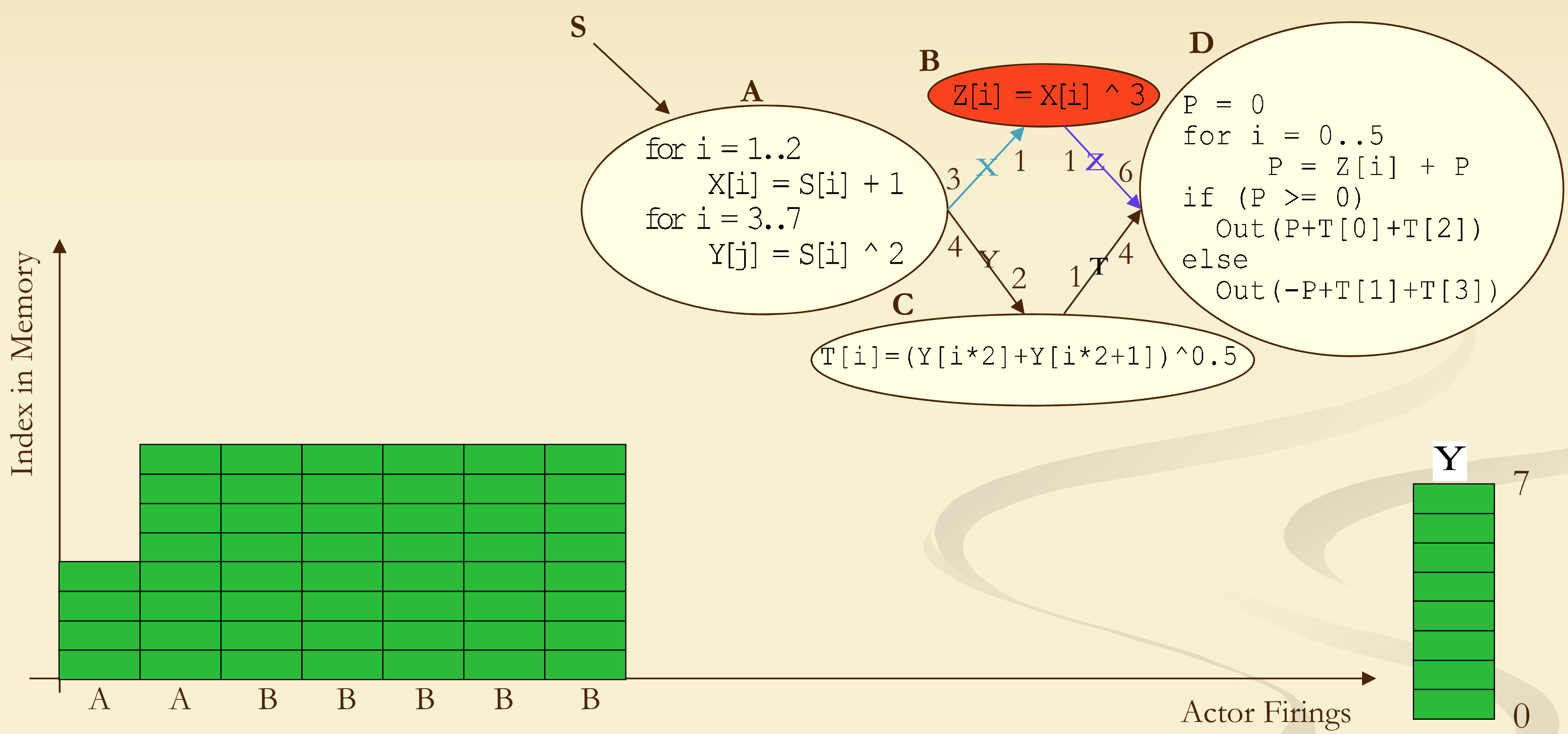
Visualizing Buffer Analysis



Schedule: **2A** **6B** **4C** **D**

Firing Sequence: **A A B B B B B B C C C C D**

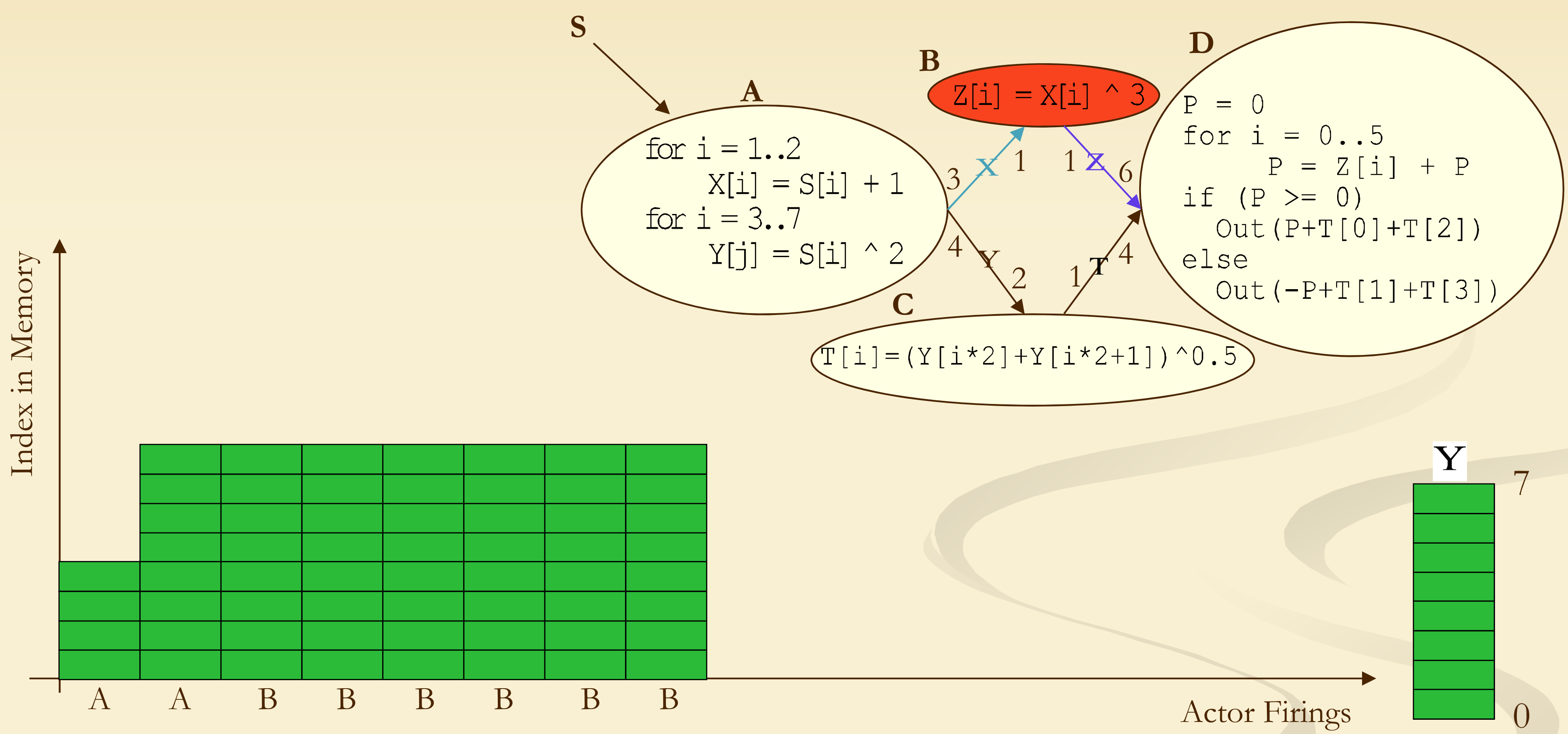
Visualizing Buffer Analysis



Schedule: **2A** **6B** **4C** **D**

Firing Sequence: **A A B B B B B B C C C C D**

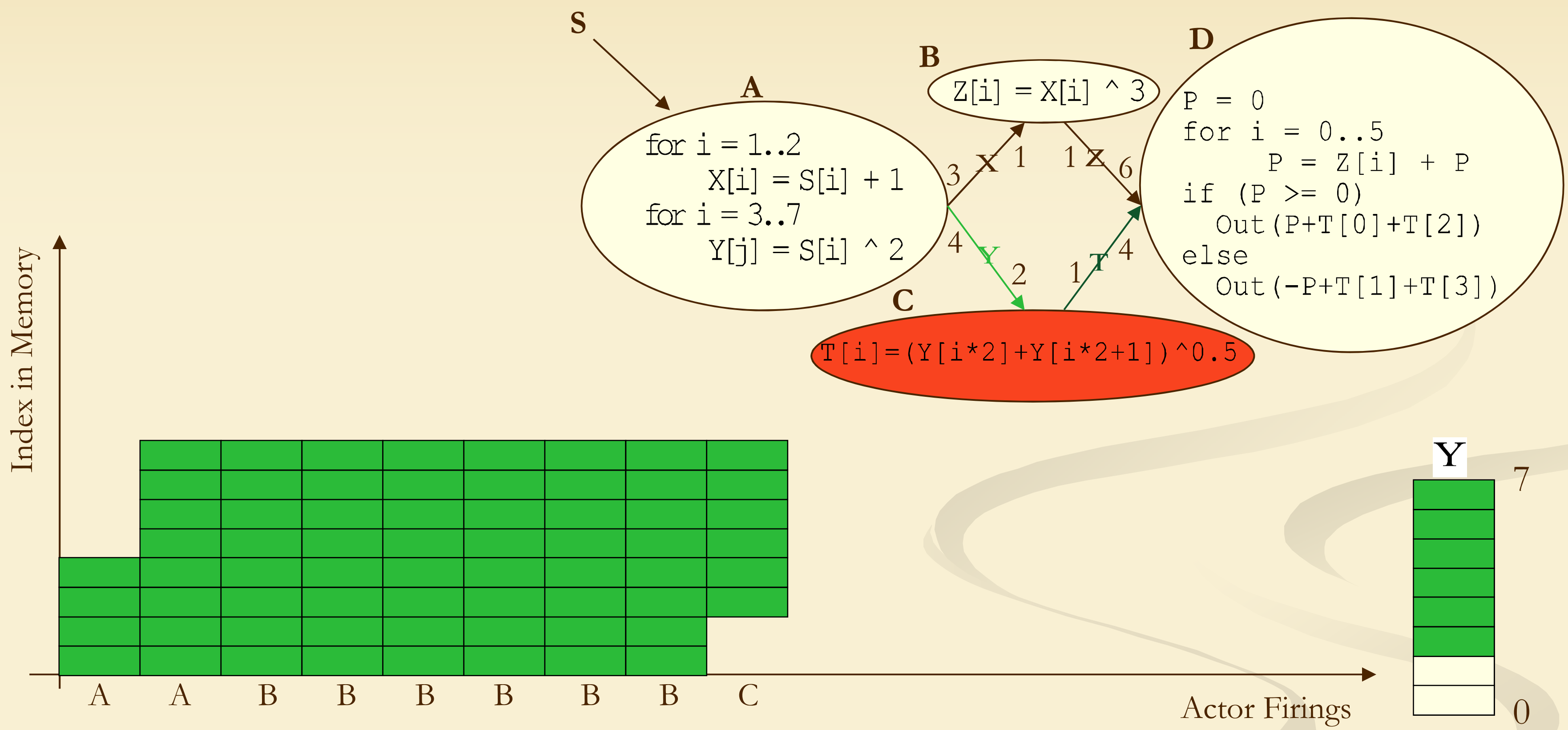
Visualizing Buffer Analysis



Schedule: 2A 6B 4C D

Firing Sequence: A A B B B B B B C C C C D

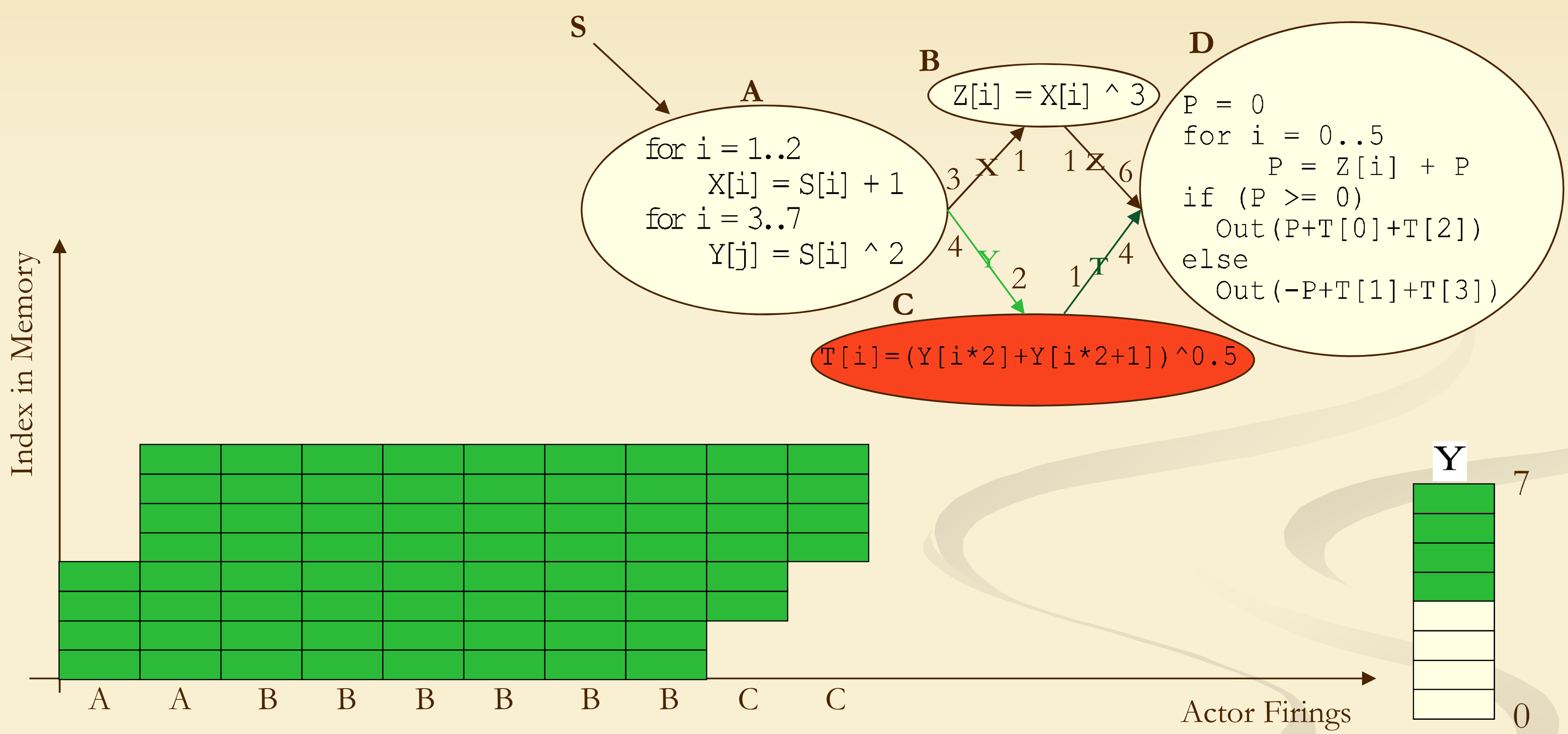
Visualizing Buffer Analysis



Schedule: **2A 6B 4C D**

Firing Sequence: **A A B B B B B B C C C C D**

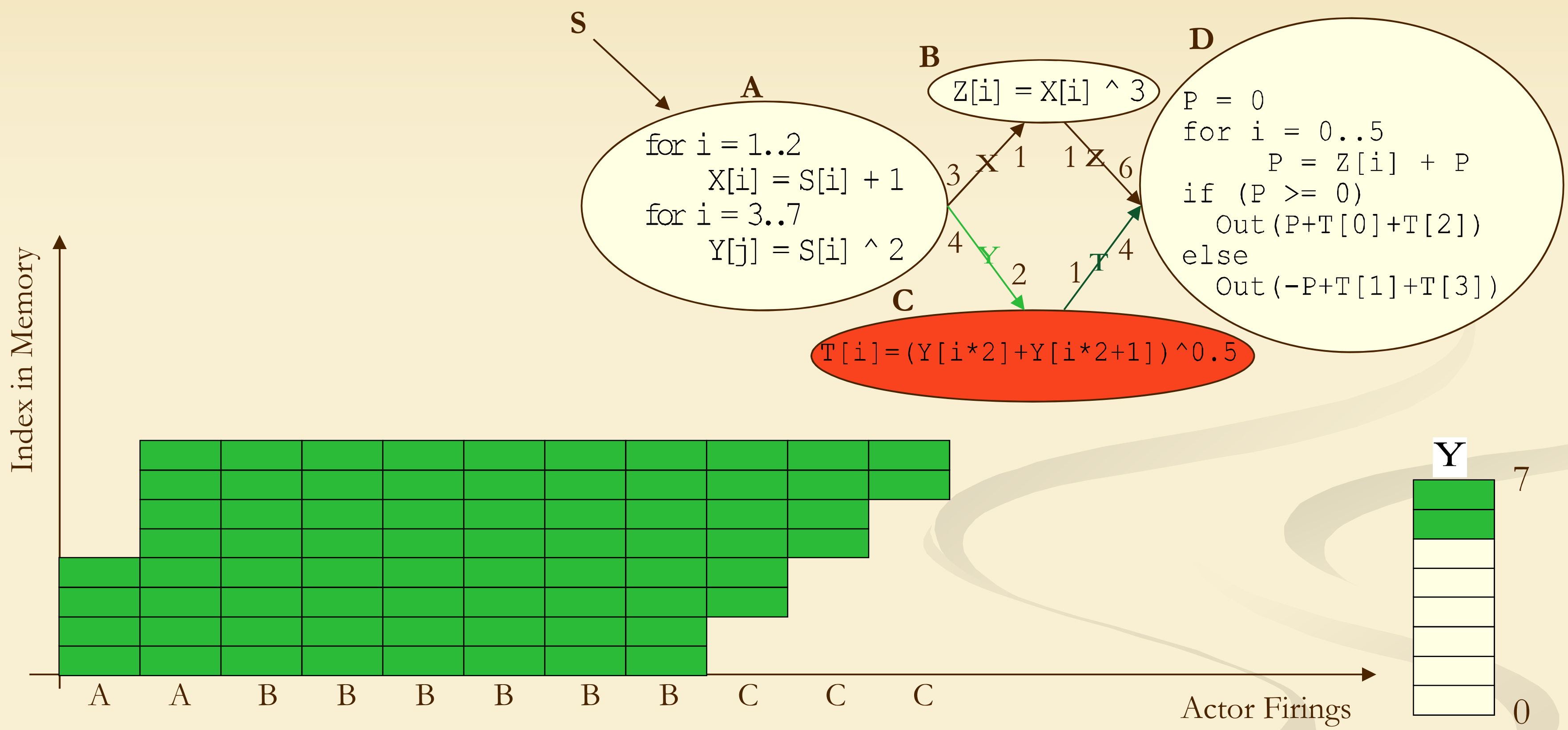
Visualizing Buffer Analysis



Schedule: 2A 6B 4C D

Firing Sequence: A A B B B B B B C C C C D

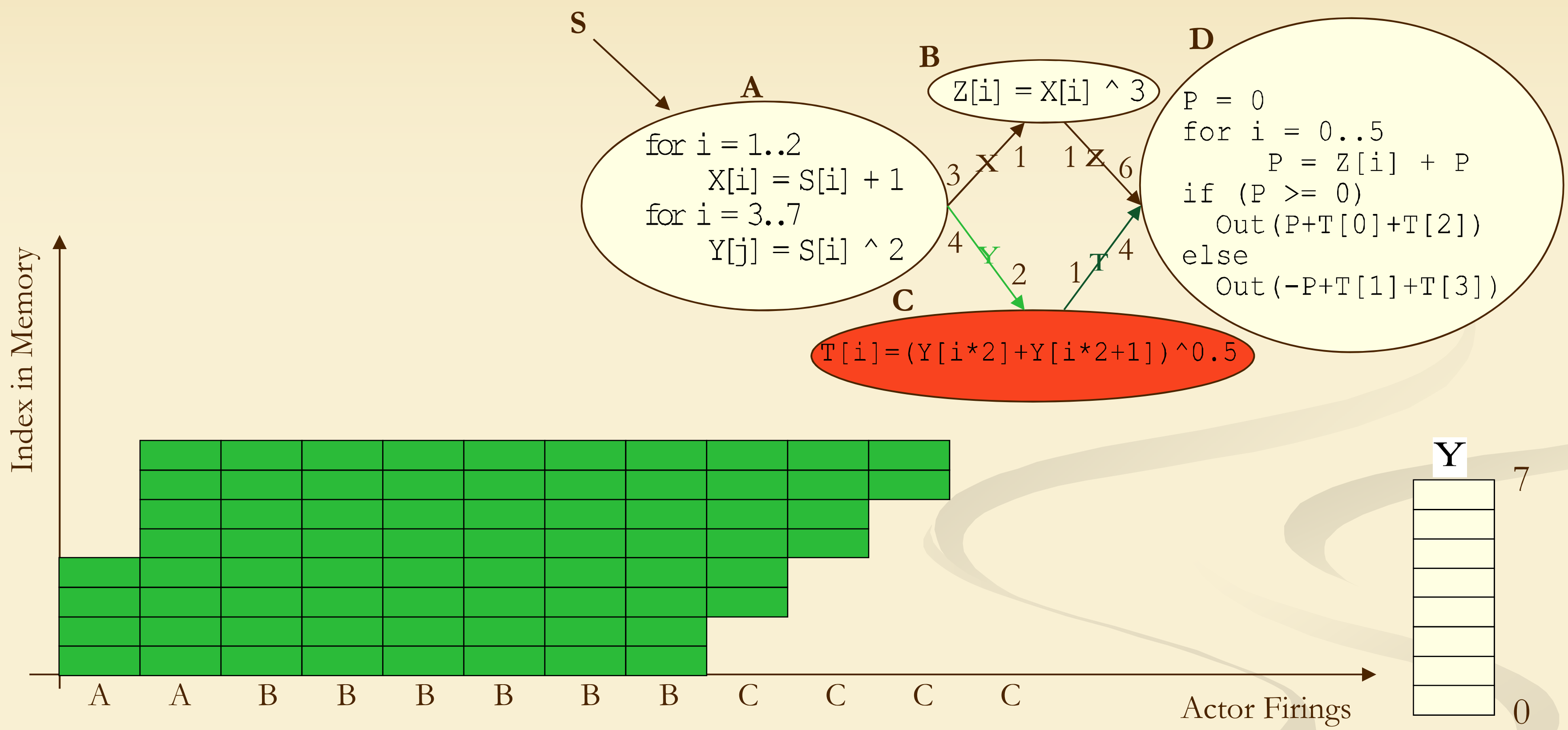
Visualizing Buffer Analysis



Schedule: **2A** **6B** **4C** **D**

Firing Sequence: **A A B B B B B B C C C C D**

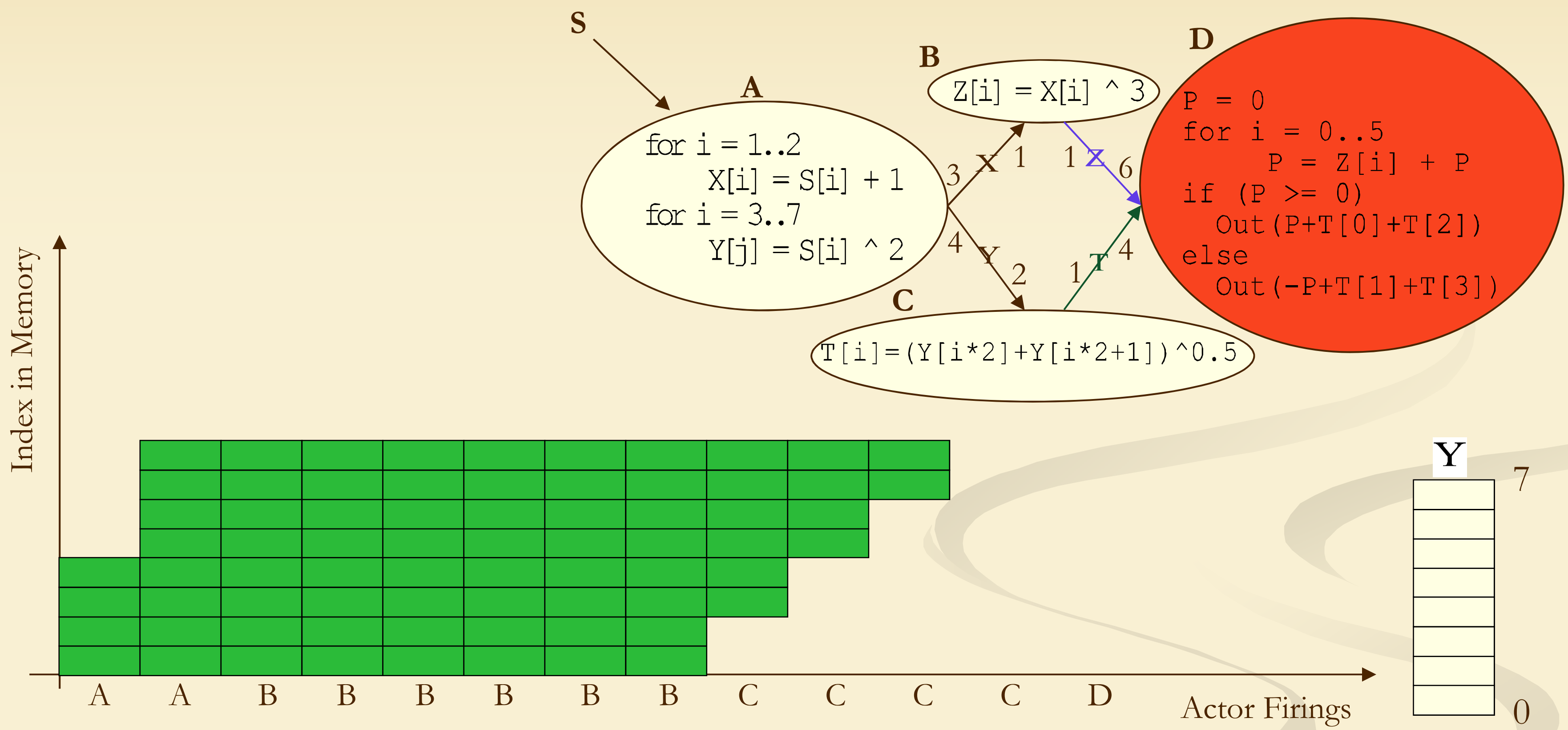
Visualizing Buffer Analysis



Schedule: **2A 6B 4C D**

Firing Sequence: **A A B B B B B B C C C C D**

Visualizing Buffer Analysis



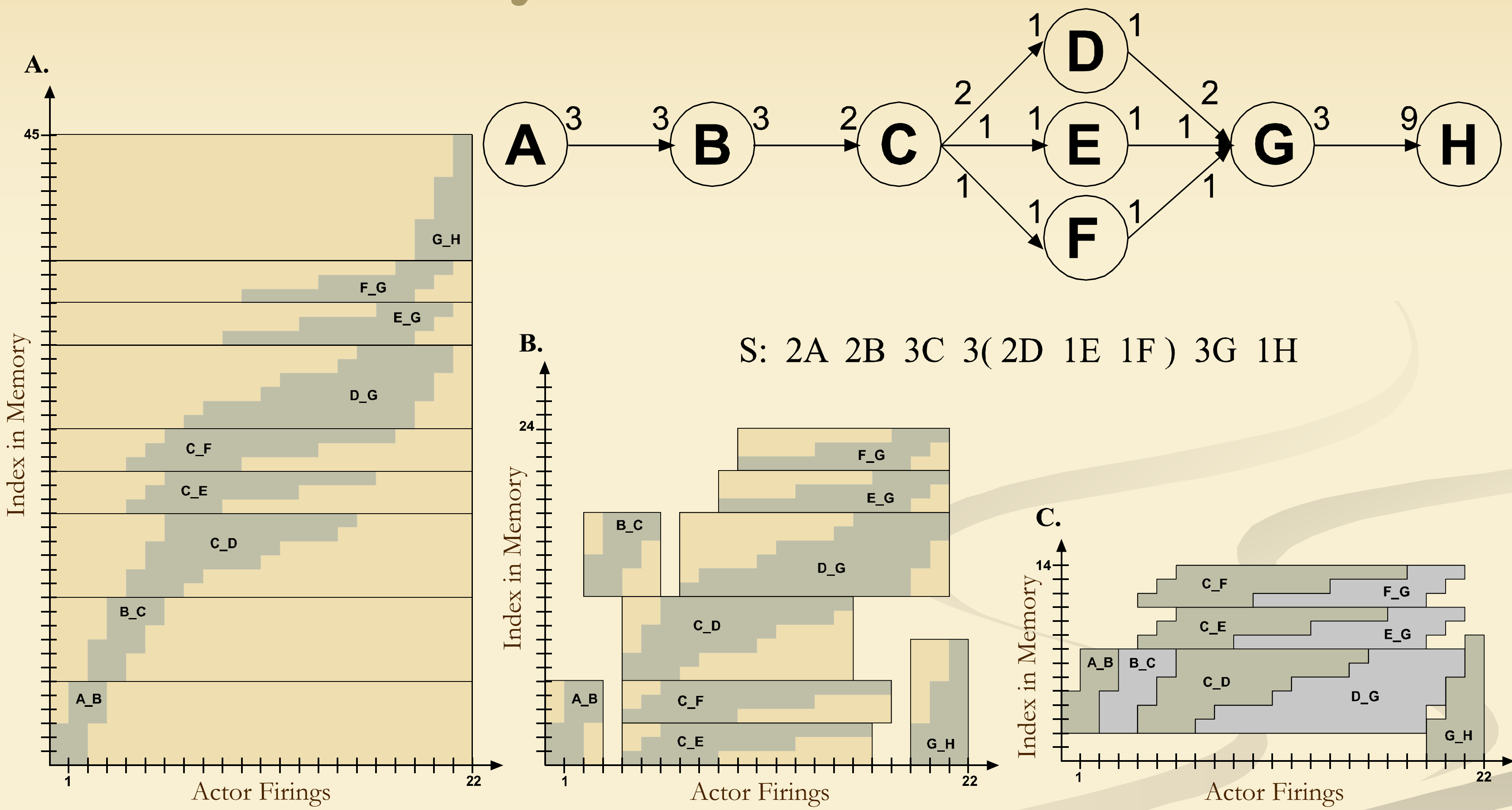
Schedule: **2A 6B 4C D**

Firing Sequence: **A A B B B B B B C C C C D**

Granularity and Buffer Allocation

- The granularity in buffer analysis compromises accuracy in temporal behavior of buffers with analysis complexity:
  - Baseline
  - Coarse-grain
  - Fine-grain

Granularity and Buffer Allocation



Baseline Analysis

Live Range Analysis (Coarse-Grain)

Fine-Grain Analysis

P. K. Murthy and S. S. Bhattacharyya. Shared buffer implementations of signal processing systems using lifetime analysis techniques.

Fine-Grain Buffer Allocation

- Mathematic Formulation:
  - Use of existing tools
  - Choose the best data structure

$$\forall e \in E \quad : \quad B_e = (H_e, L_e)$$

B_e : The Buffer on edge e which we call it buffer e in short

$H_e[t]$: Head index at time $0 \leq t \leq T$ for the buffer on e

$L_e[t]$: Tail index at time $0 \leq t \leq T$ for the buffer on e

$$T = \sum_{v \in q_G} q[v]$$

$$O = \{(o_{e_1}, o_{e_2}, o_{e_3}, \ldots, o_{e_N}) \mid e_1 : e_N \in E \;,\; N = |E|\}$$

Fine-Grain Buffer Allocation

- LEMMA:
  - In SA schedules the head index at the time t is always greater than equal the tail index at the same time: $\forall t \leq T \; : \; H_e[t] \geq L_e[t]$
- Constraints:

$$\forall \, e, b \in E \quad \forall \, 0 \leq t \leq T \; :$$
$$H_e[t] + o_e \leq L_b[t] + o_b \quad OR \quad H_b[t] + o_b \leq L_e[t] + o_e$$

- Objective: Minimize <u>S</u>hared <u>B</u>uffer <u>S</u>ize:

$$SBS = \max_{\forall e \in E} \{ o_e + H_e^{max} \mid H_e^{max} = \max_{0 \leq t \leq T} (H_e[t]) \}$$

ILP Formulation

- The complexity of buffer sharing instance, and ILP runtime grows exponentially.
- Linear constraints cannot be easily used to articulate the "OR" logic:
  - Binary variables For each buffer and each location in the shared memory space
  - Constraints have to be generated for all time steps.

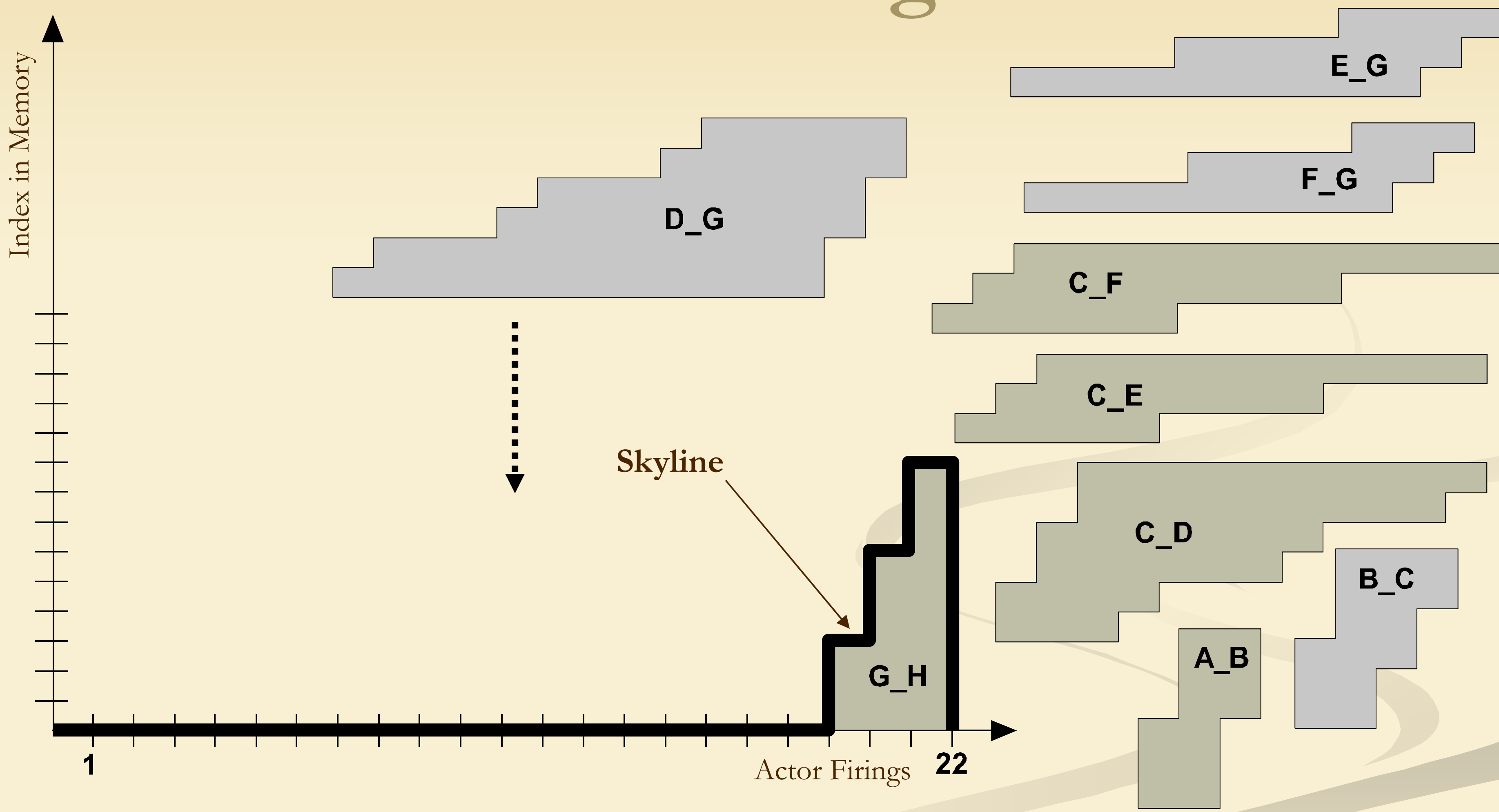
Strip Packing Problem and Buffer-Sharing

- In several industries there is a need for packing a set of 2-dimensional objects on a larger rectangular unit of material by minimizing the waste.
  - Two-Dimensional Bin Packing Problem (2BP):
    - wood or glass industries, warehousing contexts, newspapers paging
  - Two-Dimensional Strip Packing Problem (2SP):
    - paper or cloth industries

Strip Packing Problem and Buffer-Sharing

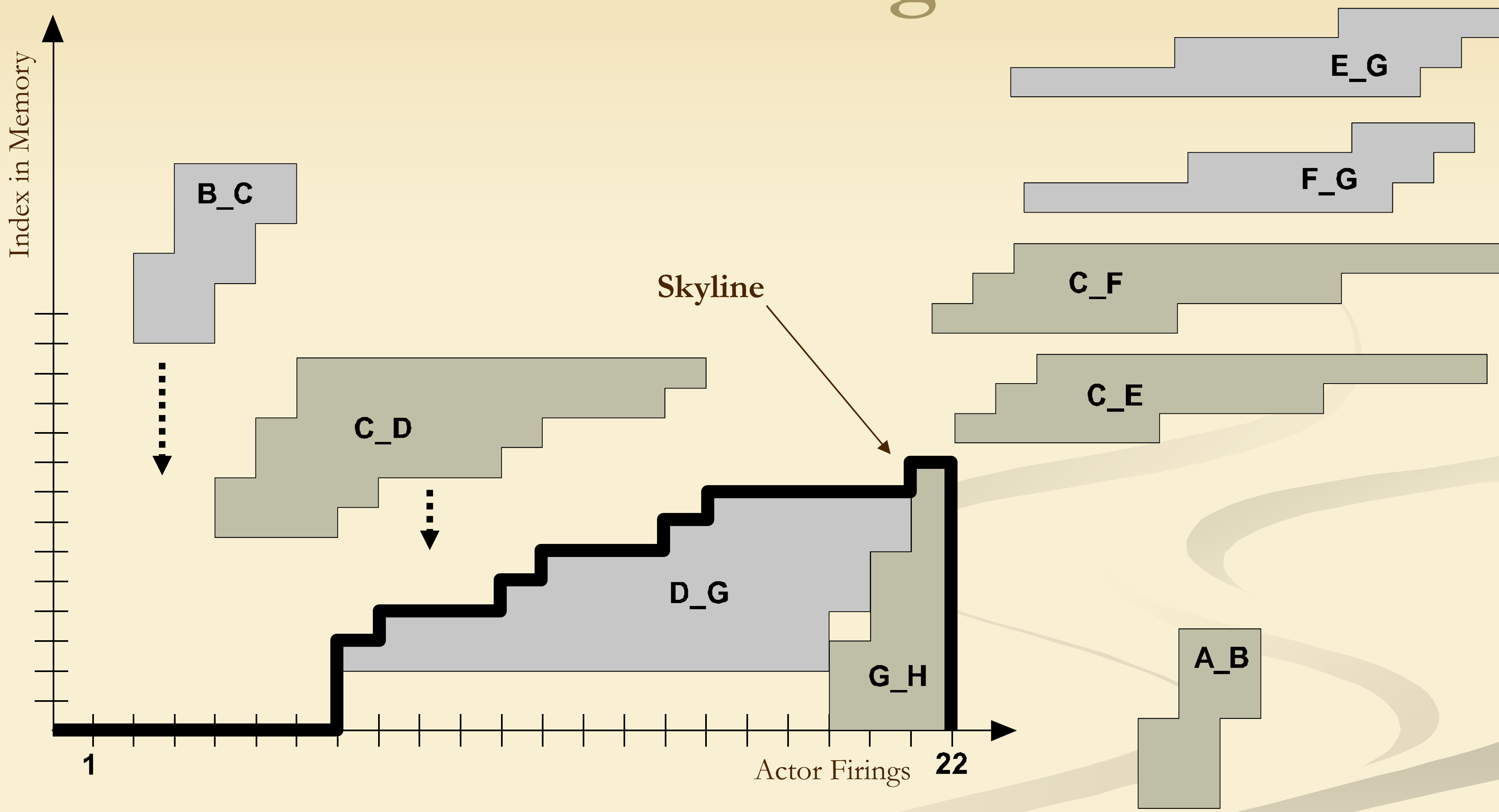
- The relationship between Packing Problems and Buffer Sharing Problem:
  - Objects: Buffer Size in Time which form complex polygons
  - Roll of Material: Shared Buffer Memory
  - Objective: To allocate an index to each buffer in the shared memory with no conflict using minimum space
  - Difference: We cannot move the objects (polygons) in time. We are only allowed to move them vertically. We also have no rotation.

Move-Down Algorithm



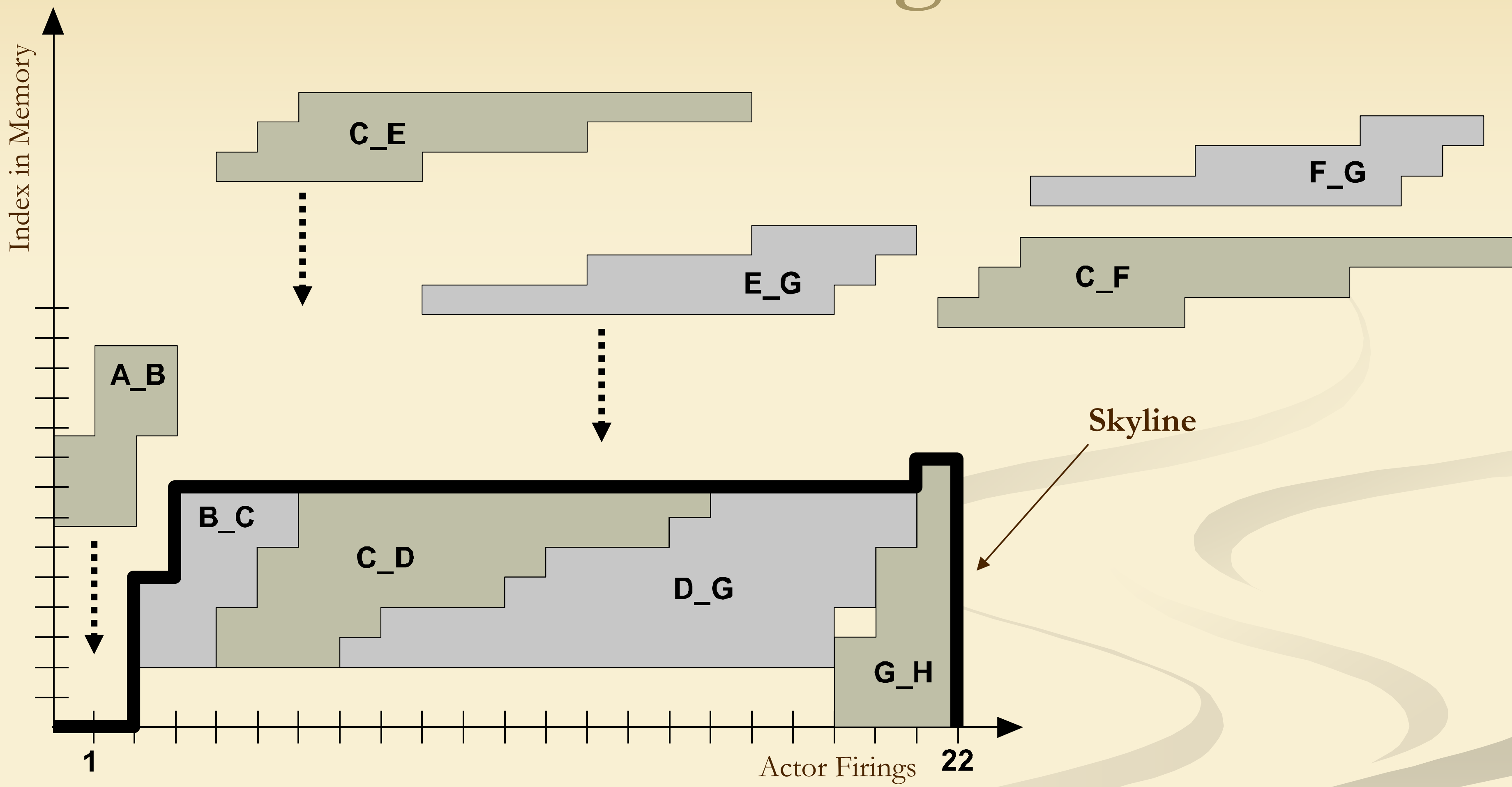
- MDA is moving down the buffers in the following order:
  G_H, D_G, C_D, B_C, A_B, E_G, C_E, F_G, C_F

Move-Down Algorithm



- MDA is moving down the buffers in the following order:
  G_H, D_G, C_D, B_C, A_B, E_G, C_E, F_G, C_F

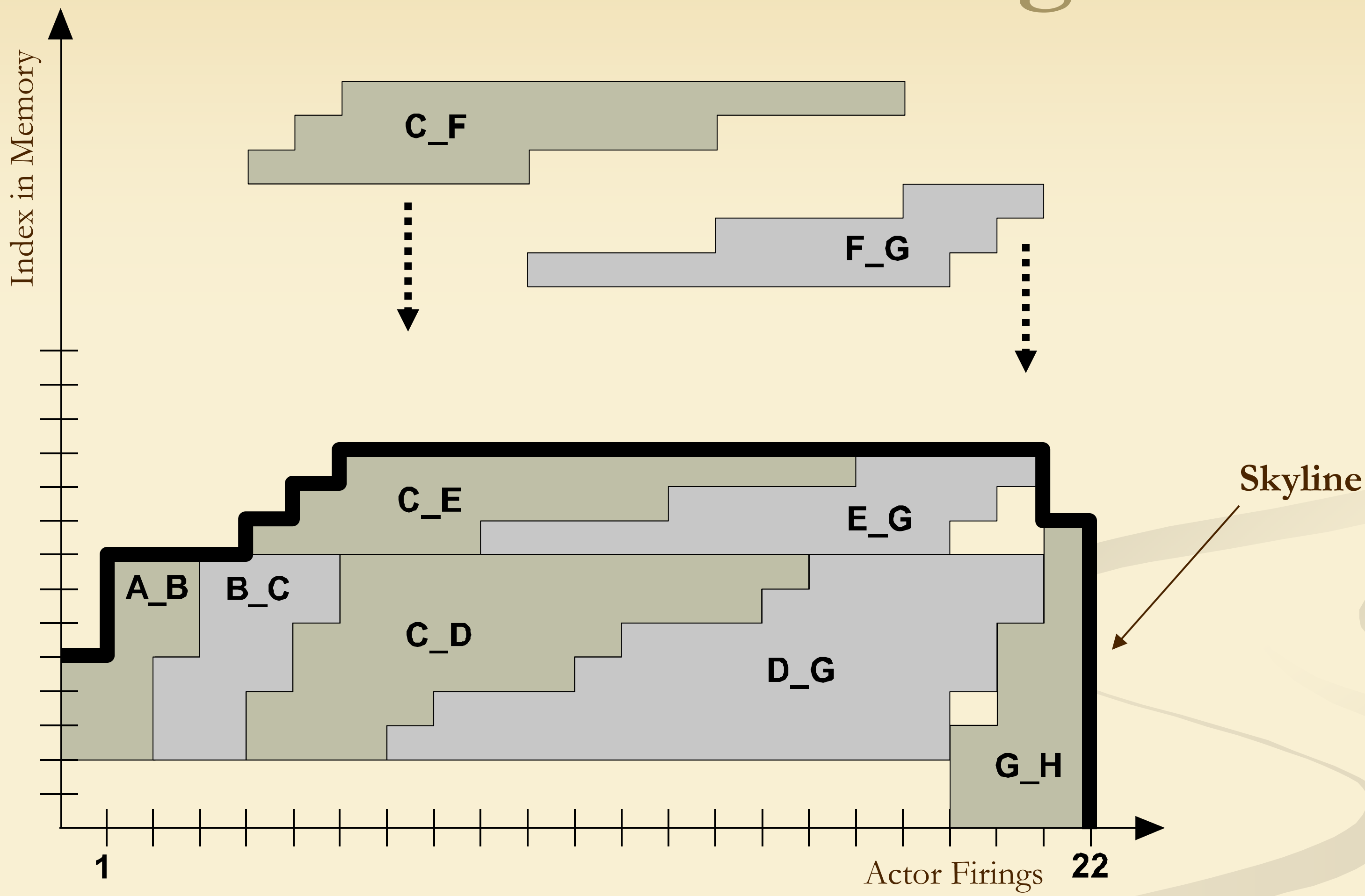
Move-Down Algorithm



- MDA is moving down the buffers in the following order:

  G_H, D_G, C_D, B_C, A_B, E_G, C_E, F_G, C_F

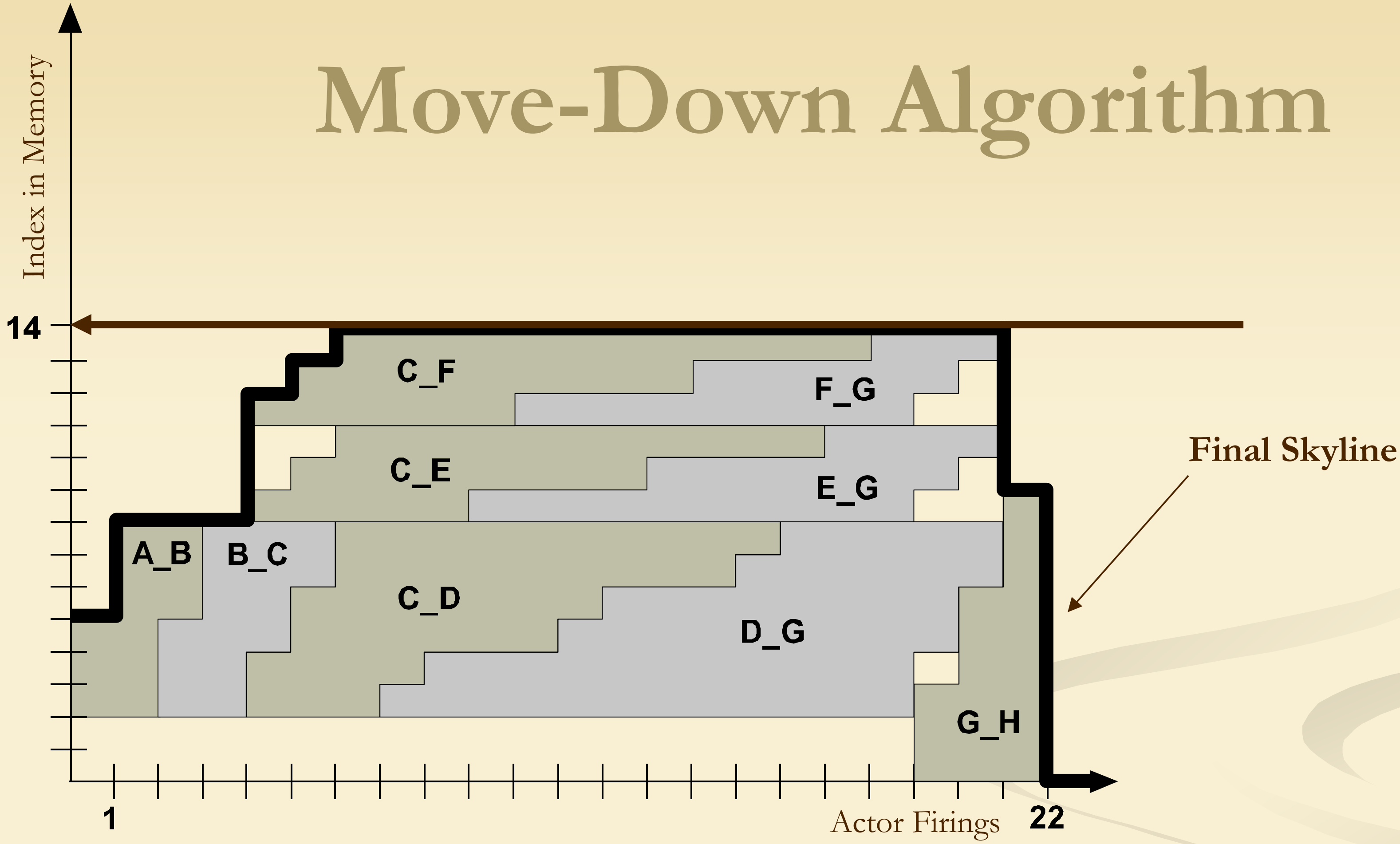
Move-Down Algorithm



- MDA is moving down the buffers in the following order:

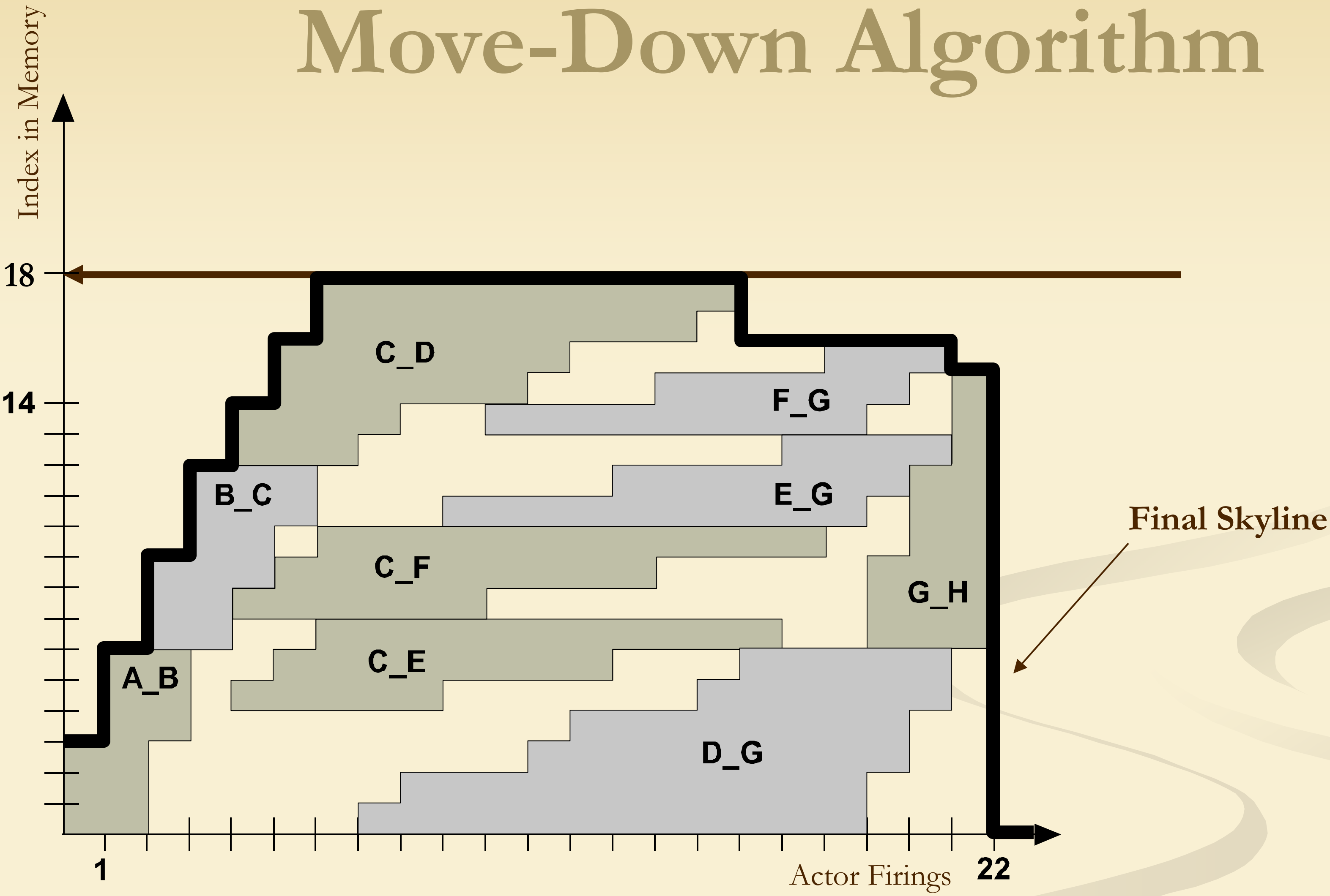
  G_H, D_G, C_D, B_C, A_B, E_G, C_E, F_G, C_F

Move-Down Algorithm



- The final placement of the buffers corresponding to the following order: G_H, D_G, C_D, B_C, A_B, E_G, C_E, F_G, C_F
- The height of the final skyline indicates the shared memory size.

Move-Down Algorithm



- Another sequence which leads to the size 18 (14 is the optimal):
  A_B , D_G , G_H , C_E , C_F , B_C , E_G , F_G , C_D

Evolutionary Optimization using MDA

- Genetic Algorithms in General:
  - **Chromosome**: Provides an abstract representation of solutions in the search space,
  - **Inheritance**: Models the basic operations through which, chromosomes are perturbed to improve the solution quality
    - Crossover
    - Mutation
  - **Fitness Function**: Quantizes the quality of candidate solutions, and determines survival of selected candidates.

Evolutionary Optimization using MDA

- Initialization: Randomly select a set of permutations

$$\text{Sample set} = \{\pi_1, \pi_2, \pi_3, \ldots, \pi_N\}$$

- Fitness function:

$$f(\pi) = \frac{1}{height(\pi)}$$

- Selection:

$$p(\pi_i) = \frac{f(\pi_i)}{\sum_{j=1}^{N} f(\pi_j)}$$

Evolutionary Optimization using MDA

- Crossover:
  - Example: $p = 2 \quad q = 4$

$$\pi_{parent1} = (B_{e1}, \underbrace{\mathbf{B}_{e2}, \mathbf{B}_{e3}, \mathbf{B}_{e4}}, B_{e5}, B_{e6})$$

$$\pi_{parent2} = (\mathbf{B}_{e6}, \mathbf{B}_{e5}, B_{e4}, B_{e3}, B_{e2}, \mathbf{B}_{e1})$$

$$\pi_{child} = (B_{e2}, B_{e3}, B_{e4}, B_{e6}, B_{e5}, B_{e1})$$

- Mutation:
  - Example: $p_{mutation} = 0.4$: the probability of being mutated

$$i = 2 \quad j = 4$$

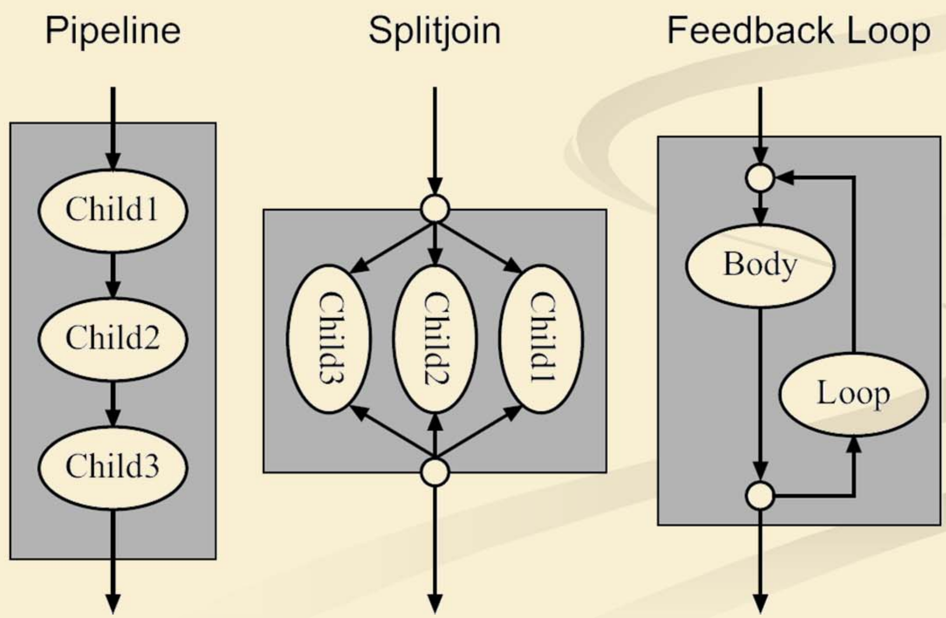
$$\pi_{child} \text{ Before } = (B_{e2}, \mathbf{B}_{e3}, B_{e4}, \mathbf{B}_{e6}, B_{e5}, B_{e1})$$

$$\pi_{child} \text{ After } = (B_{e2}, B_{e6}, B_{e4}, B_{e3}, B_{e5}, B_{e1})$$

- Iteratively, new children are generated and compared to the existing members until the termination point where we can return the best solution found.

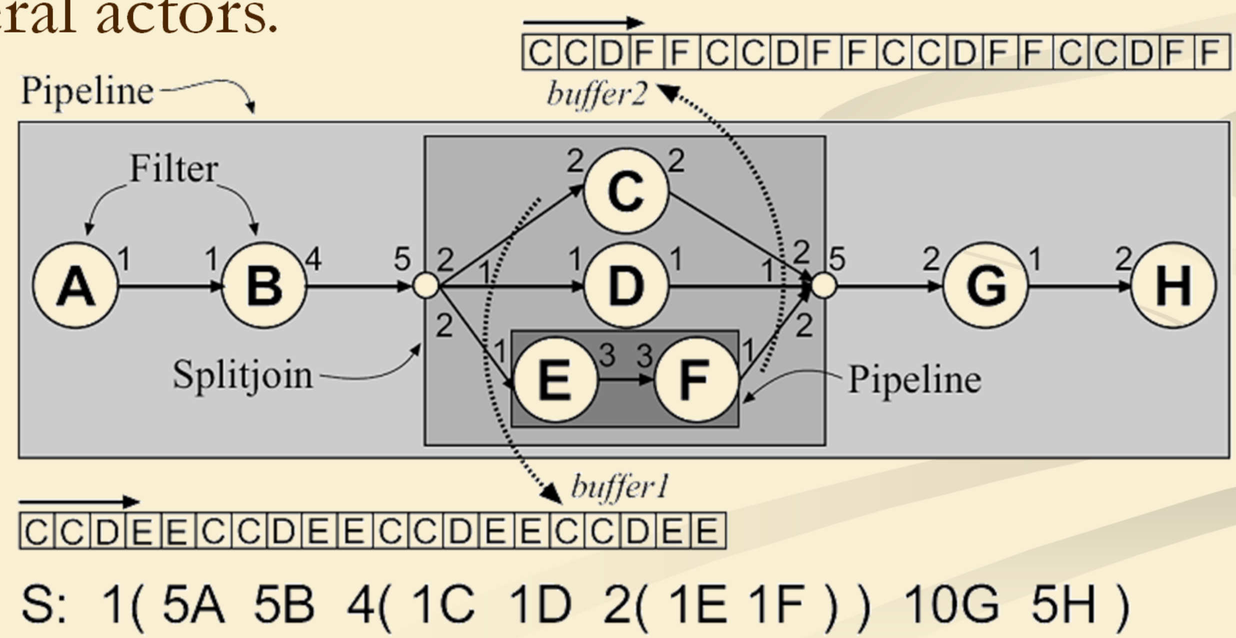
Experimental Evaluation

- We have integrated our algorithm into the MIT StreamIt compiler
- Three composite stream objects in StreamIt
- Filters specify data processing



Experimental Evaluation

- The StreamIt scheduler is designed based on the hierarchical nature of the language.
- In Split-joins, one large buffer is used to implement multiple channels that either split to or join from several actors.

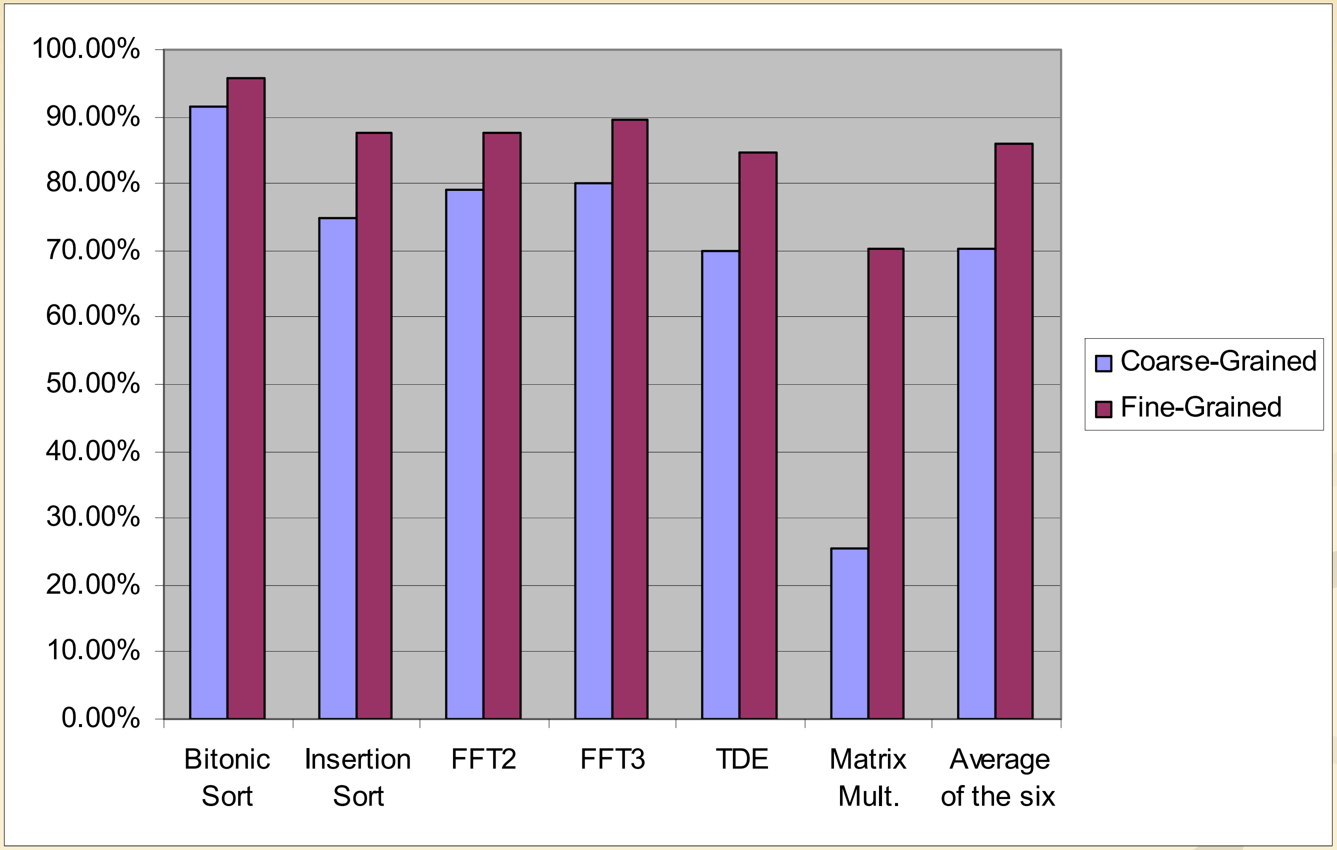


Experimental Evaluation

- Benchmark Applications:
  - Two sorting algorithms: Bitonic Sort, Insertion Sort
  - Two different implementation of the Fast Fourier Transform
  - Time Delay Estimation kernel
  - Matrix Multiplication kernel

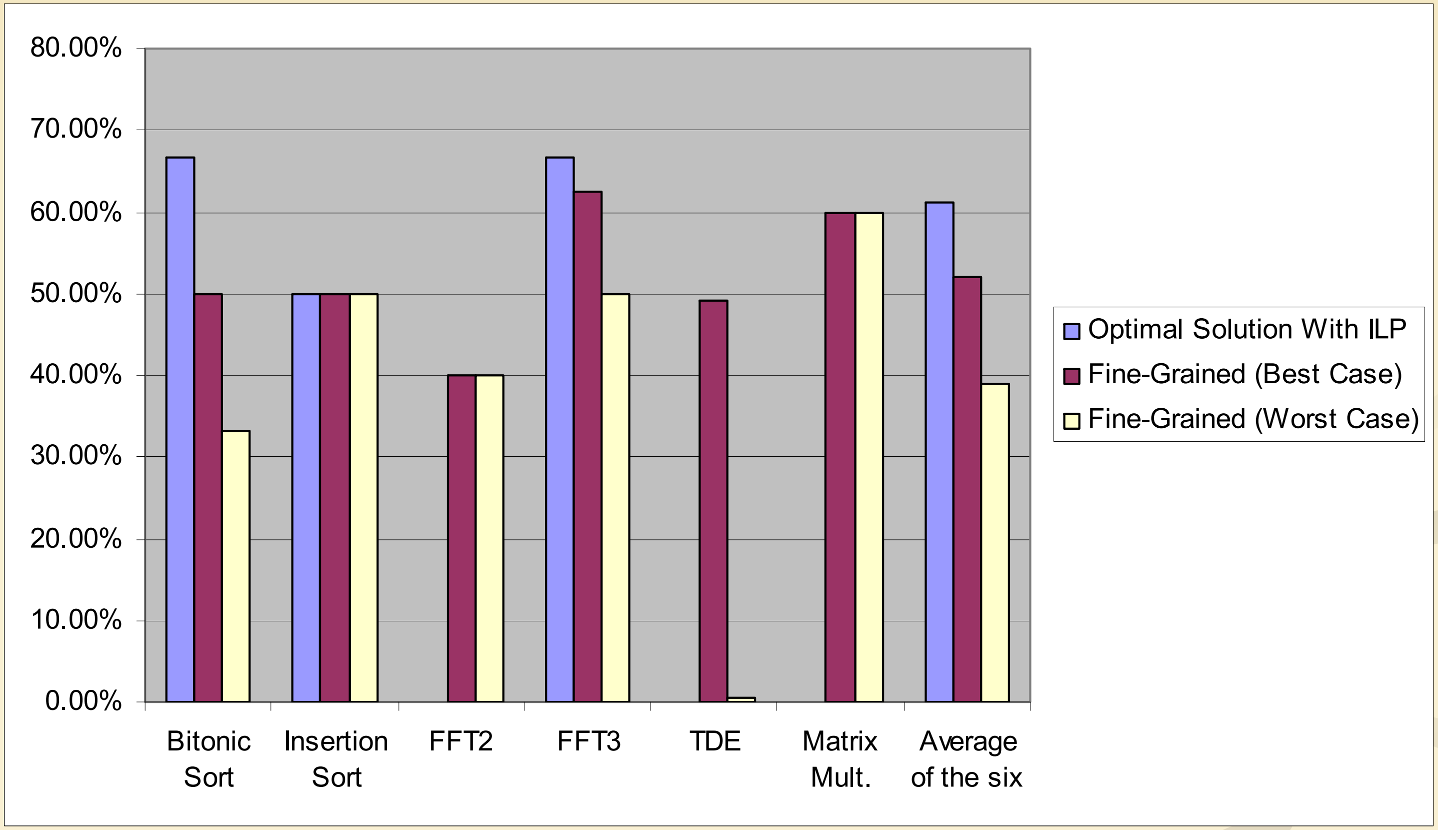
	Number of Buffers	Number of Actores	Number of Time Steps	Base-line	Coarse-Grain	Fine-Grain (Best Case)	Fine-Grain (Worst Case)	Compile Time with GA in Sec.	Optimal Solution by ILP
Bitonic Sort	119	214	340	1152	96	48	64	91	32
Insertion Sort	8	9	263	1024	256	128	128	6	128
FFT2	22	24	446	3072	640	384	384	10	~
FFT3	38	64	175	960	192	72	96	11	64
TDE	48	51	17204	77120	23168	11776	23040	510	~
Matrix Mult.	10	21	2712	5000	4000	2000	2000	13	~

Experimental Evaluation



Improvement of coarse-grain and fine-grain methods compared to the baseline.

Experimental Evaluation



Improvement in all fine-grain cases: GA worst case, GA best case, and ILP, compared to the coarse-grain method

Conclusions

- Visualization of buffers transforms the allocation problem into packing of complex polygons
- Fine-grain analysis vs. conventional coarse-grain live range analysis: dramatic improvements
- The benefits of this approach outweighs the reasonable increase in static analysis latency for a large class of resource-constrained embedded systems.