

On Channel Restructuring for Complete FIFO Recovery

Focus: **Dataflow models** as an intermediate representation for **High-Level Synthesis**

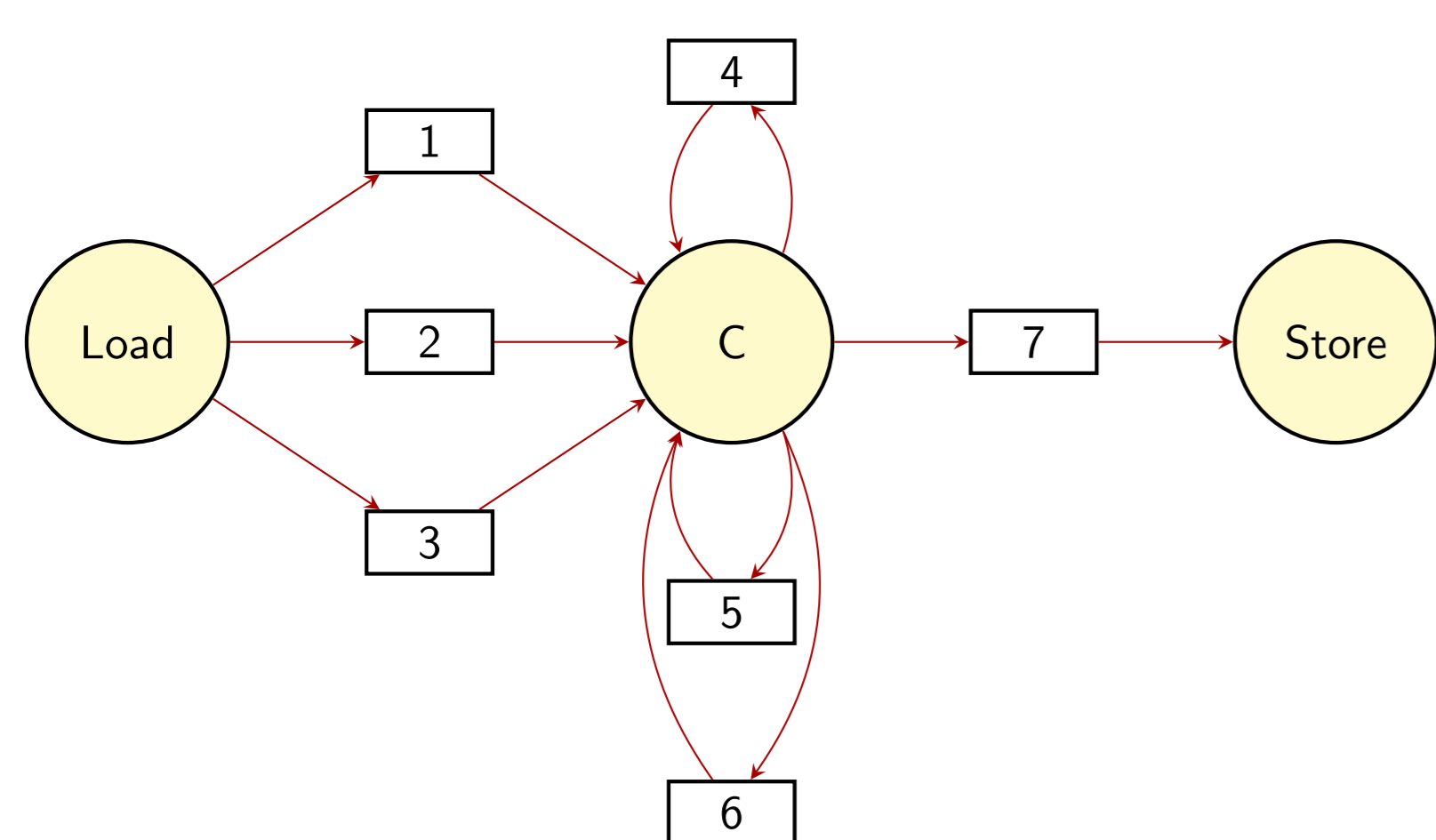
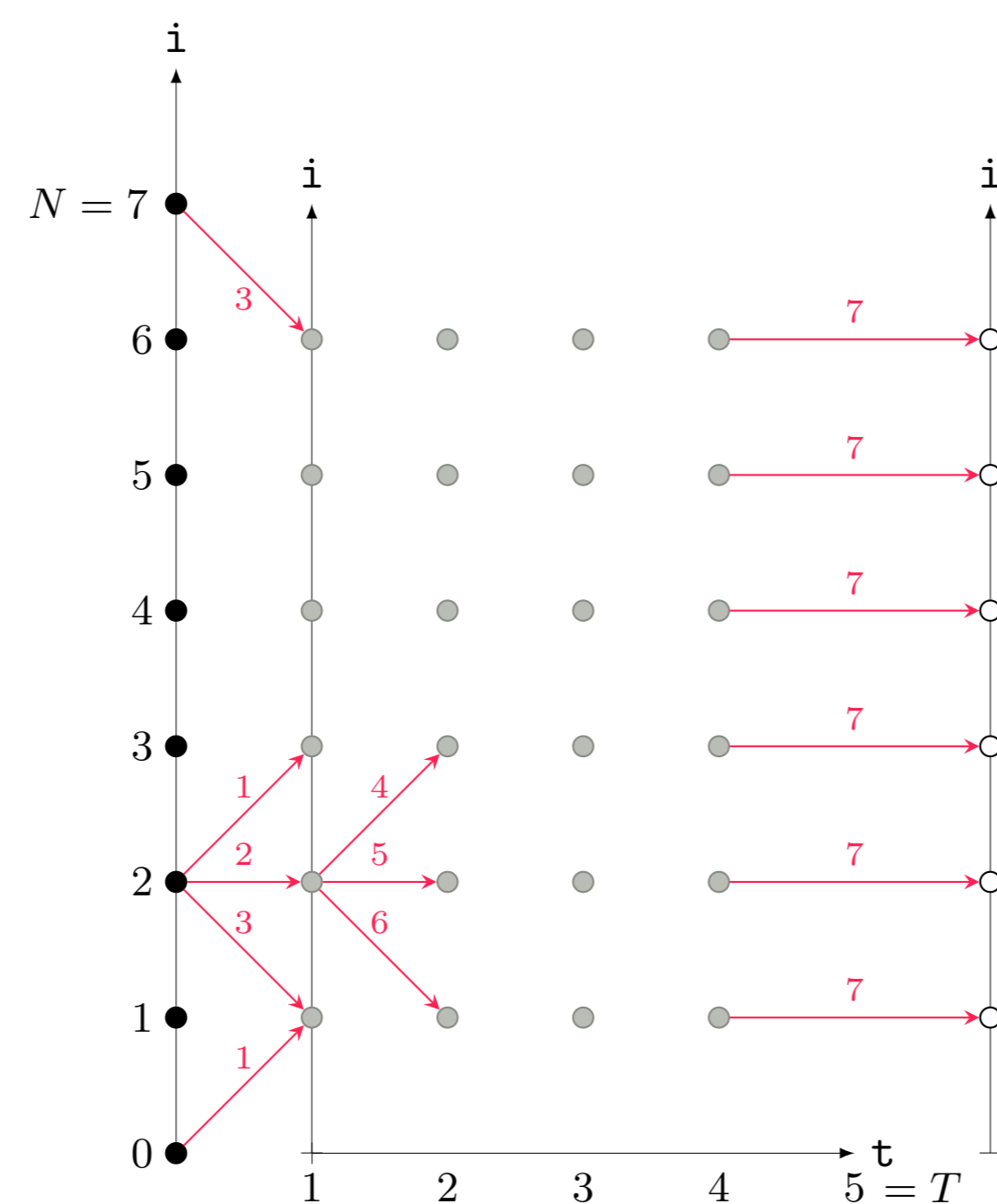
Challenge: Recover FIFO channels **after code restructuring**

Contributions: **HLS algorithm** for channel restructuring + **Dataflow model** to ensure completeness

1) Polyhedral Process Networks

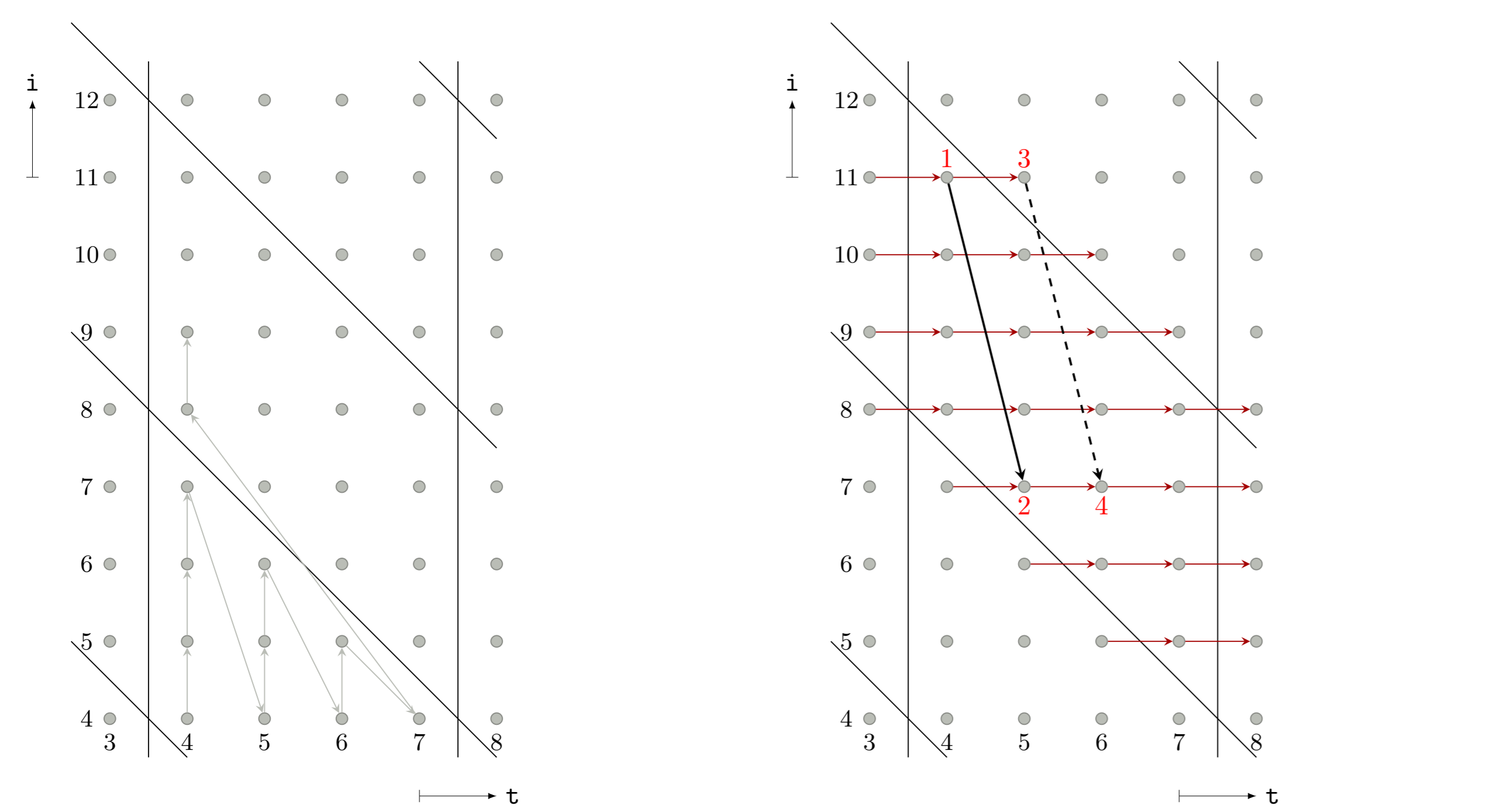
```

for i := 0 to N + 1
• load(a[0, i]);
for t := 1 to T
  for i := 1 to N
    a[t, i] :=
      a[t - 1, i - 1] + a[t - 1, i] + a[t - 1, i + 1];
  for i := 1 to N
    store(a[T, i]);
  
```



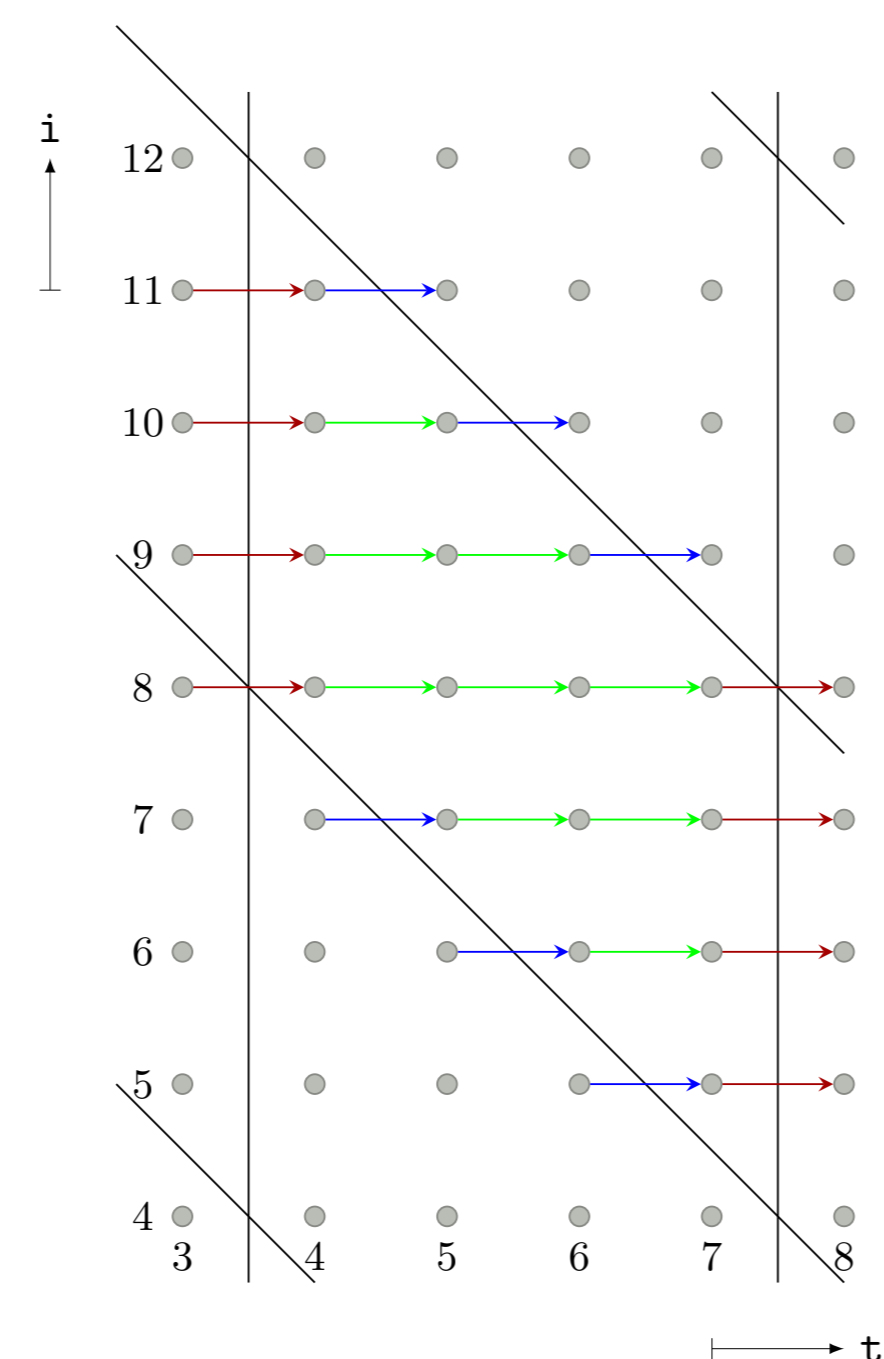
Challenge addressed:
Channel synthesis

3) Our FIFO Recovery Algorithm



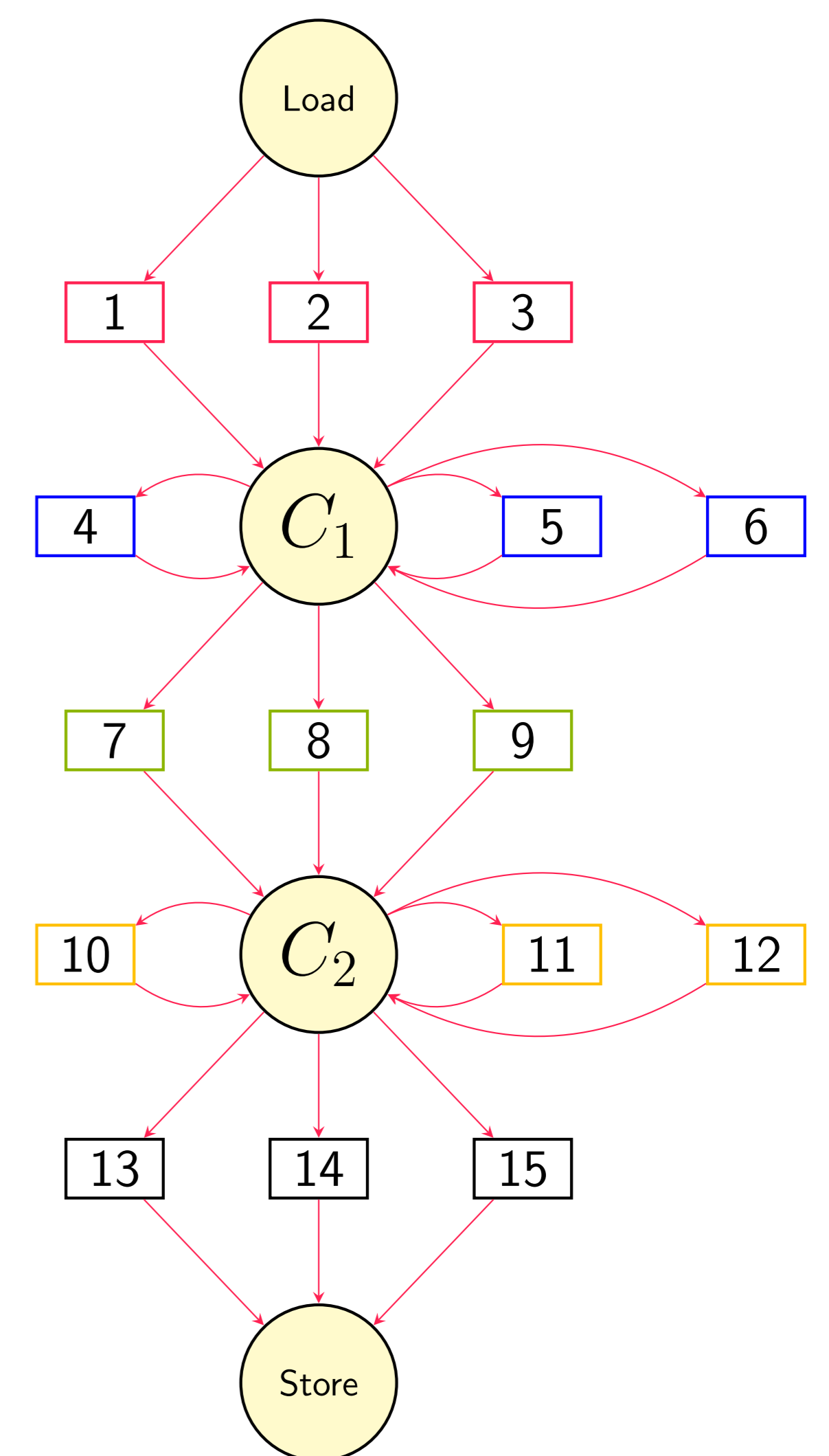
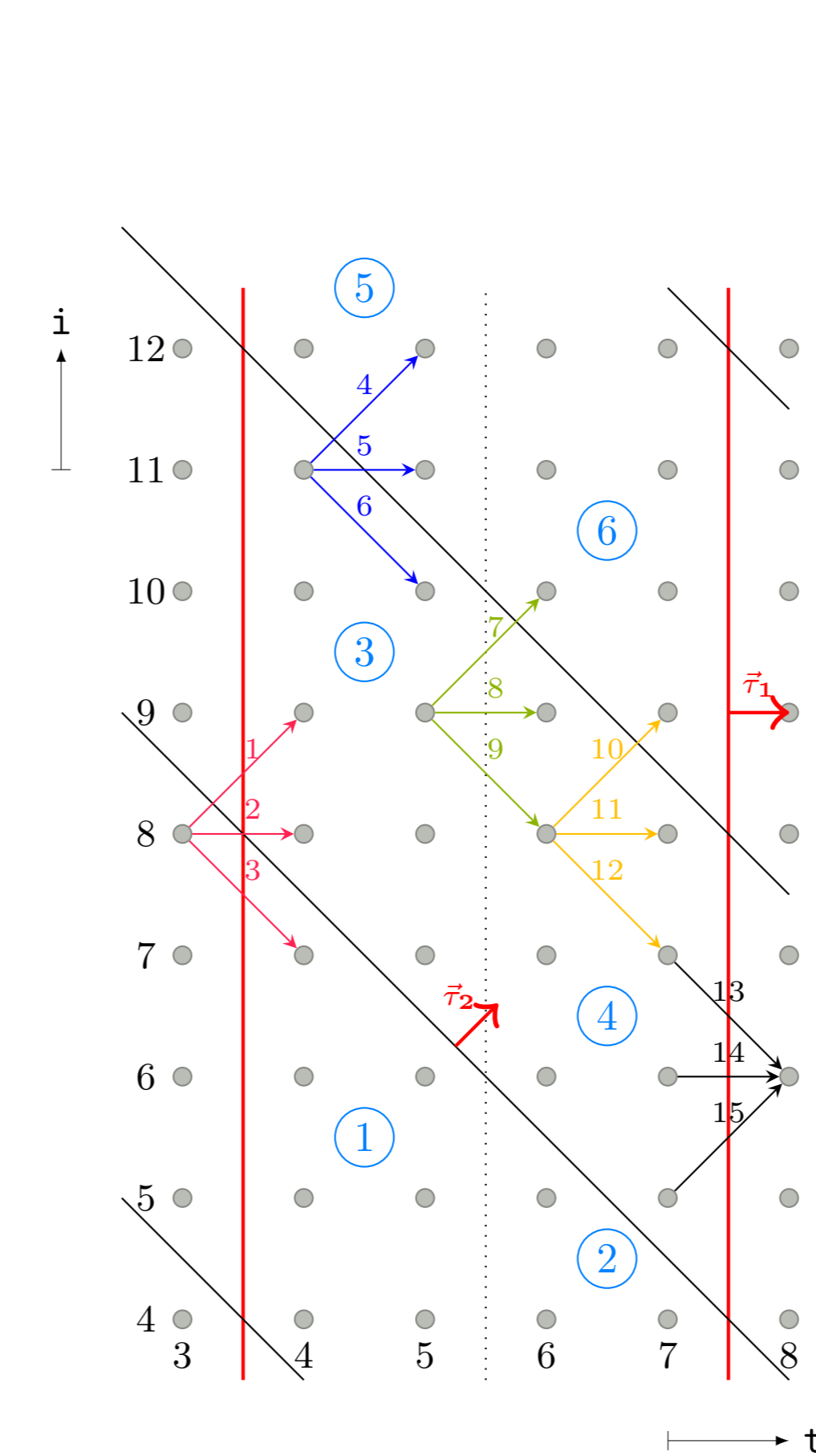
```

1 SPLIT( $\rightarrow_c, \theta_P, \theta_C$ )
2 for k := 1 to n
3   ADD( $\rightarrow_c \cap \{(x, y), \theta_P(x) \ll^k \theta_C(y)\}$ );
4   ADD( $\rightarrow_c \cap \{(x, y), \theta_P(x) \approx^n \theta_C(y)\}$ );
5 FIFOIZE( $(\mathcal{P}, \mathcal{C})$ )
6 for each channel c
7    $\{\rightarrow_c^1, \dots, \rightarrow_c^{n+1}\} := \text{SPLIT}(\rightarrow_c, \theta_{P_c}, \theta_{C_c});$ 
8   if  $\text{fifo}(\rightarrow_c^k, \prec_{\theta_{P_c}}, \prec_{\theta_{C_c}}) \forall k$ 
9     REMOVE( $\rightarrow_c$ );
10    INSERT( $\rightarrow_c^k$ )  $\forall k$ ;
  
```



Theorem: FIFO recovery is complete on our DPN model

2) Data-aware Process Networks



Features:

- PPN partitioning based on **loop tiling**
- Incoming dependences (1,2,3) are **loaded**
- Outgoing dependences (13,14,15) are **stored**
- Internal dependences (4 to 12) are solved through **local channels**

4) Experimental Results

| Kernel | PPN | | | | DPN | | |
|-------------|-------|------|------|-----|------|------|-----|
| | #fifo | #rem | #rec | % | #rem | #rec | % |
| trmm | 2 | 1 | 1 | 100 | 1 | 1 | 100 |
| gemm | 2 | 1 | 1 | 100 | 1 | 1 | 100 |
| syk | 2 | 1 | 1 | 100 | 1 | 1 | 100 |
| symm | 6 | 3 | 3 | 100 | 5 | 1 | 100 |
| gemver | 4 | 2 | 2 | 100 | 3 | 1 | 100 |
| gesummv | 6 | 6 | 0 | 100 | 6 | 0 | 100 |
| syk2k | 2 | 1 | 1 | 100 | 1 | 1 | 100 |
| lu | 3 | 0 | 3 | 100 | 0 | 3 | 100 |
| trisolv | 4 | 3 | 1 | 100 | 3 | 1 | 100 |
| cholesky | 6 | 3 | 3 | 100 | 4 | 2 | 100 |
| doitgen | 3 | 2 | 1 | 100 | 2 | 1 | 100 |
| bicg | 4 | 2 | 2 | 100 | 2 | 2 | 100 |
| mvt | 2 | 0 | 2 | 100 | 0 | 2 | 100 |
| 3mm | 6 | 2 | 2 | 50 | 3 | 3 | 100 |
| 2mm | 4 | 2 | 1 | 50 | 2 | 2 | 100 |
| covariance | 7 | 4 | 2 | 66 | 4 | 3 | 100 |
| correlation | 13 | 9 | 3 | 75 | 9 | 4 | 100 |
| fdtd-2d | 12 | 0 | 6 | 50 | 5 | 7 | 100 |
| jacobi-2d | 10 | 0 | 2 | 20 | 2 | 8 | 100 |
| seidel-2d | 9 | 0 | 3 | 33 | 2 | 7 | 100 |
| jacobi-1d | 6 | 1 | 5 | 100 | 2 | 4 | 100 |
| heat-3d | 20 | 0 | 0 | 0 | 2 | 18 | 100 |

Experimental setup:

- We have run our algorithm on the kernels of **PolyBench/C v3.2**
- We have checked the completeness of our algorithm on PPN with DPN partitioning (**DPN**)
- We have studied the behavior of our algorithm on general PPN, without DPN partitioning (**PPN with tiling**).