Carnegie Mellon University

Improving Performance of Irregular Applications on Spatial-Dataflow Processors

Nathan Serafin

November 11, 2024

Outline

Spatial dataflow review

Spatial dataflow is efficient + irregular apps are important

Spatial dataflow not efficient with irregular apps

Make your irregular app look regular

Exploit app properties to increase clock frequency

Eliminate clock and fully fuse

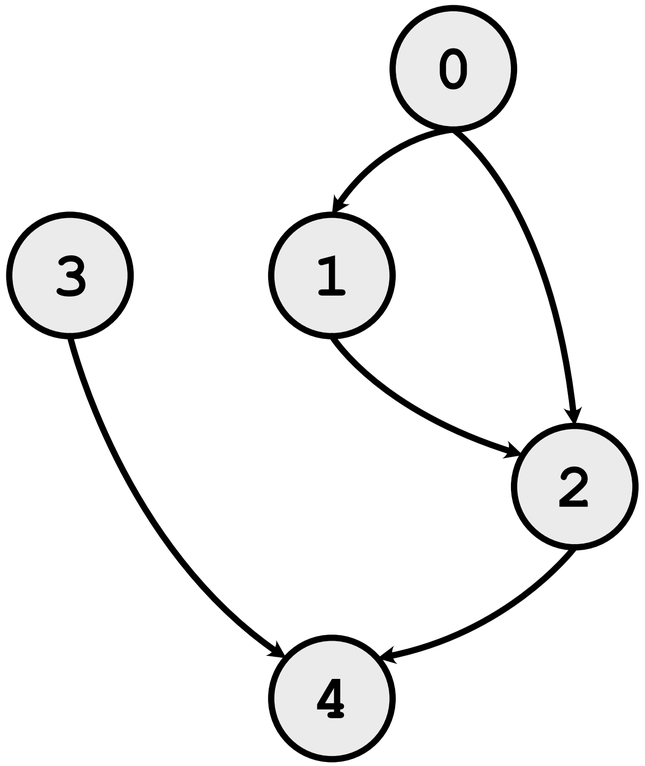
Spatial dataflow review

Execution driven by data dependencies

```
0: add x0, x1, x2
1: add x3, x4, x0
2: add x5, x0, x3
3: add x6, x7, x8
4: add x9, x6, x5
```

Execution driven by data dependencies

```
0: add x0, x1, x2
1: add x3, x4, x0
2: add x5, x0, x3
3: add x6, x7, x8
4: add x9, x6, x5
```

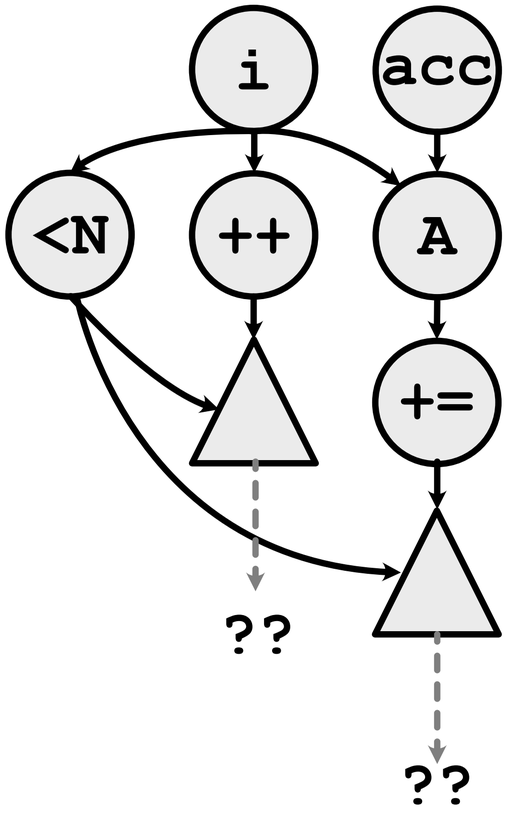


What to do about loops?

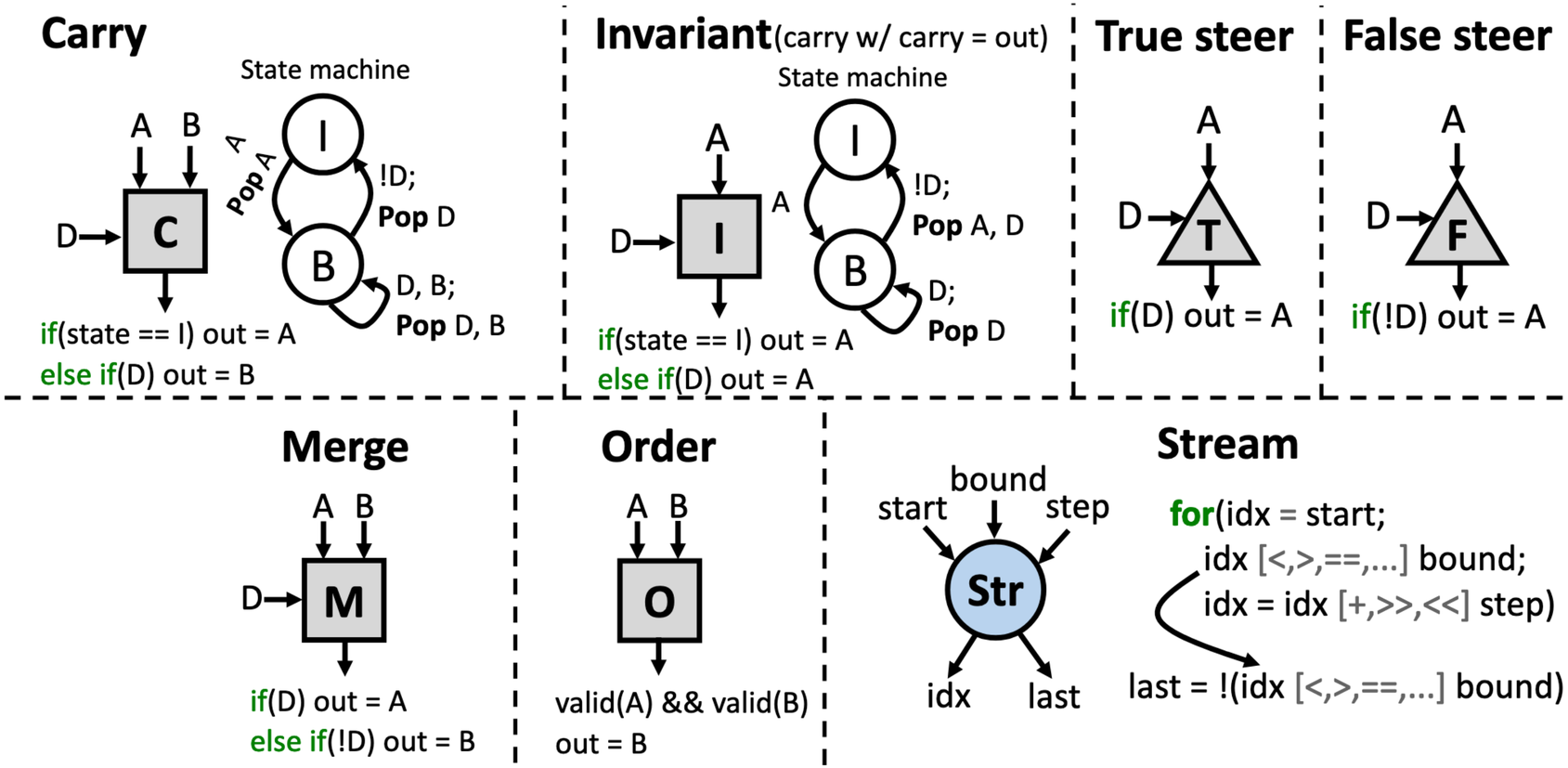
```
acc = 0
for i = 0; i < N; i++
     acc += A[i]
```

What to do about loops?

```
acc = 0
for i = 0; i < N; i++
      acc += A[i]
```

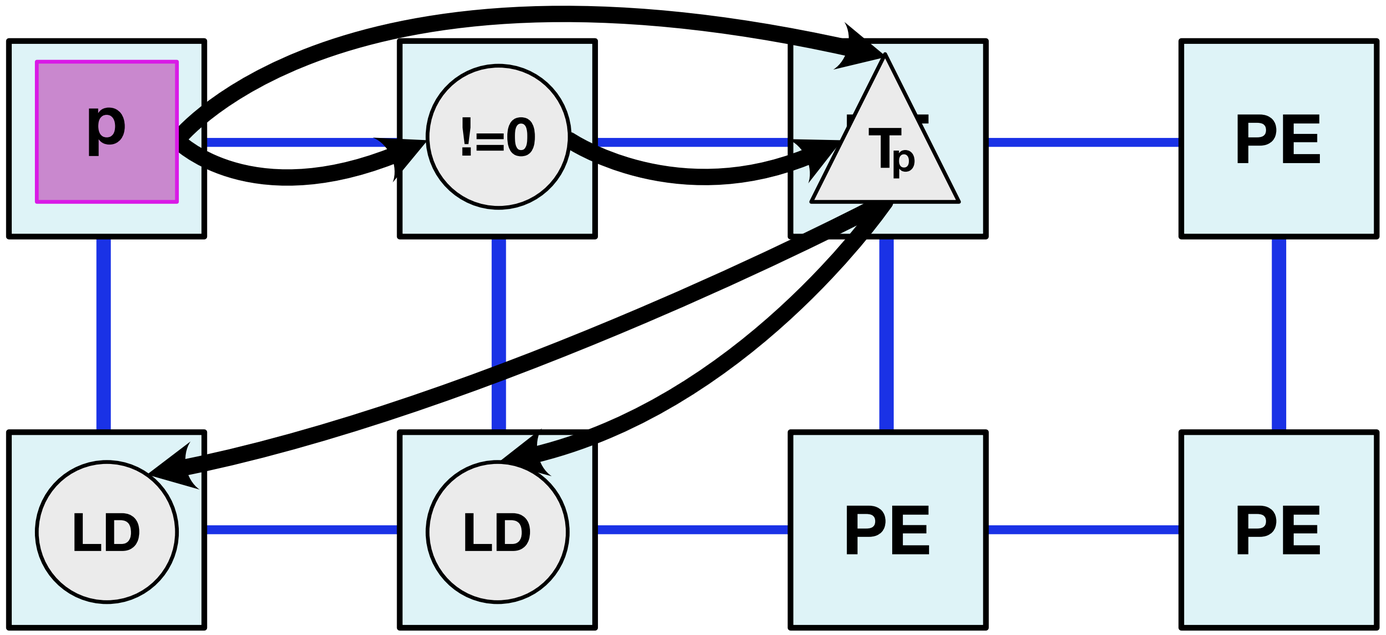


Add control-flow operators



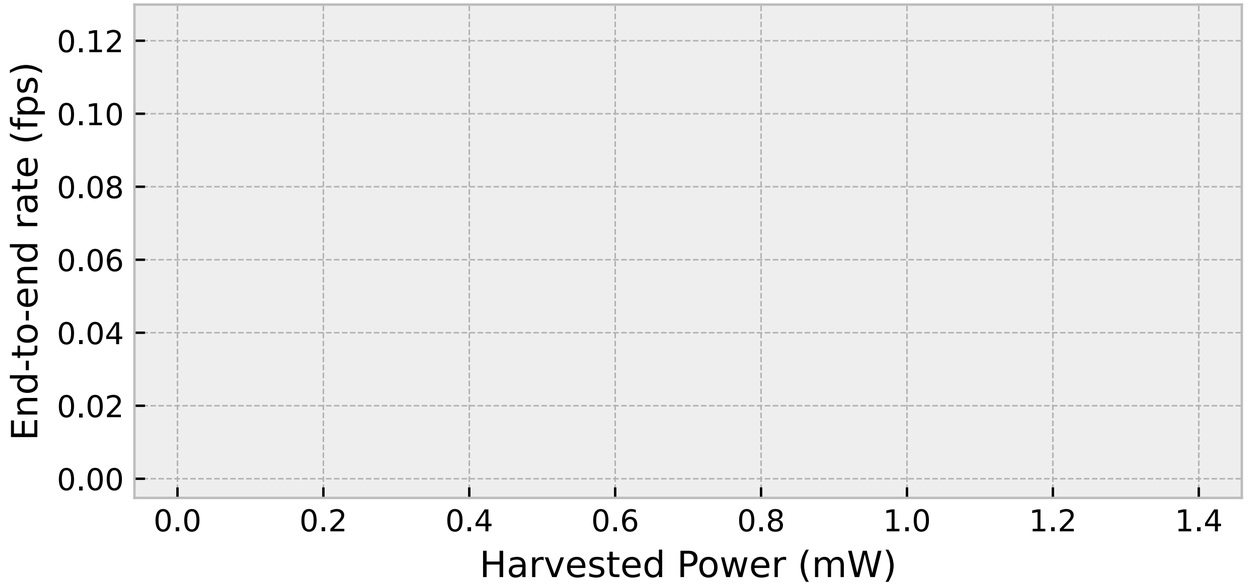
[RiptTide, Gobieski '22]

Map full programs onto compute fabric

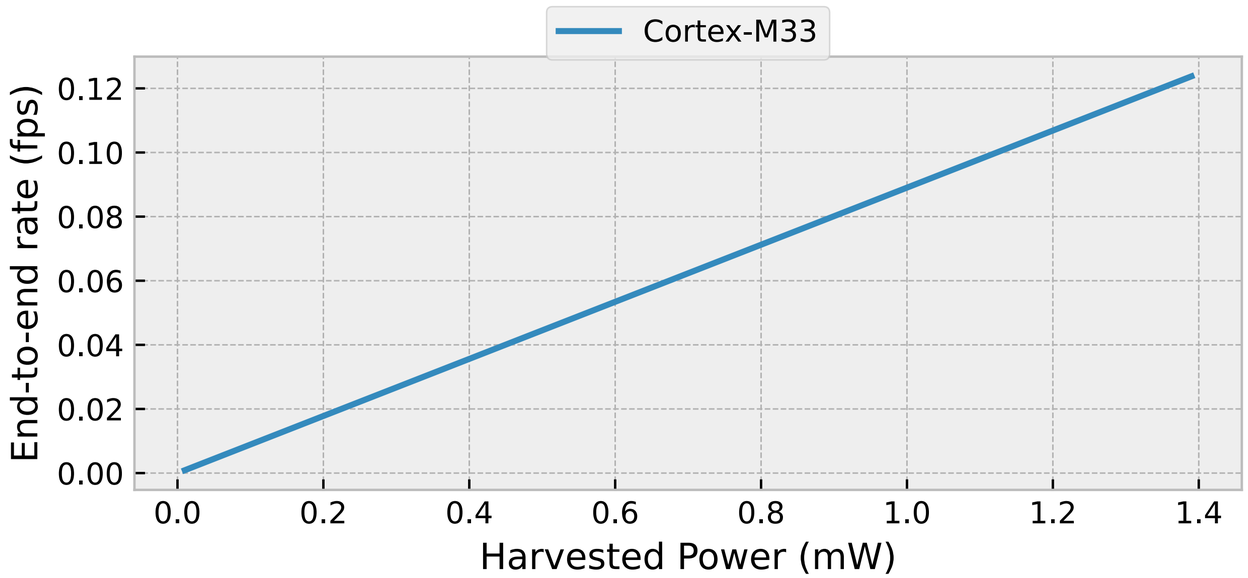


Spatial dataflow is fast and efficient

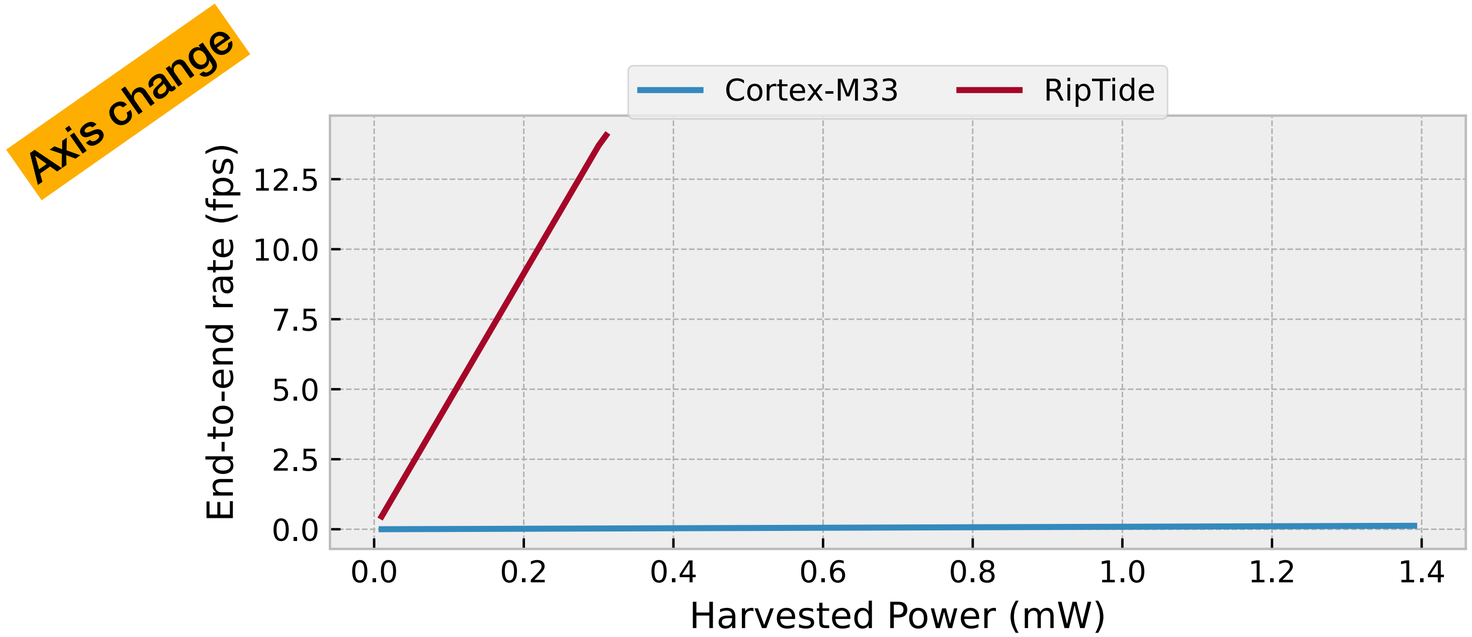
Case study: Sparse DNN



Commodity processors are unsuitable



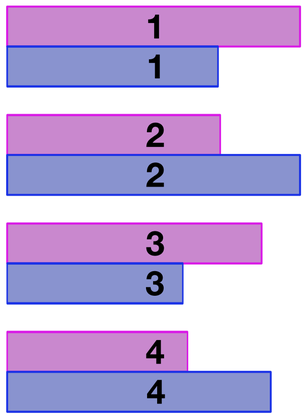
State-of-the-art better at low power



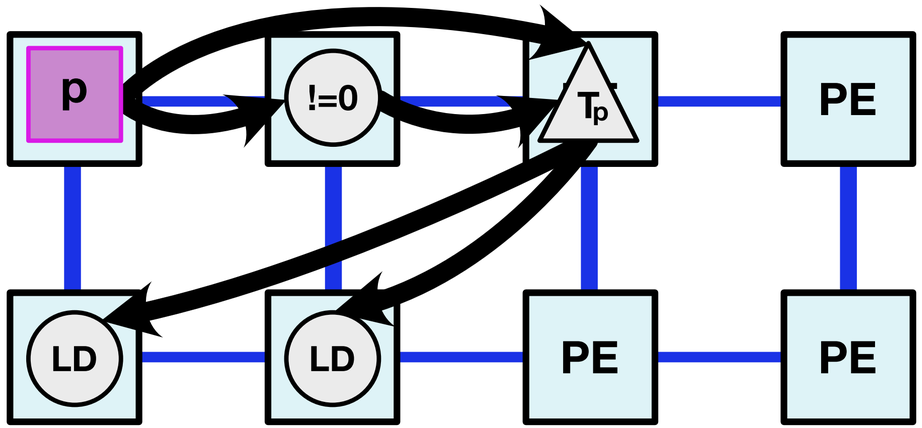
Irregular apps are important

Irregular apps are important

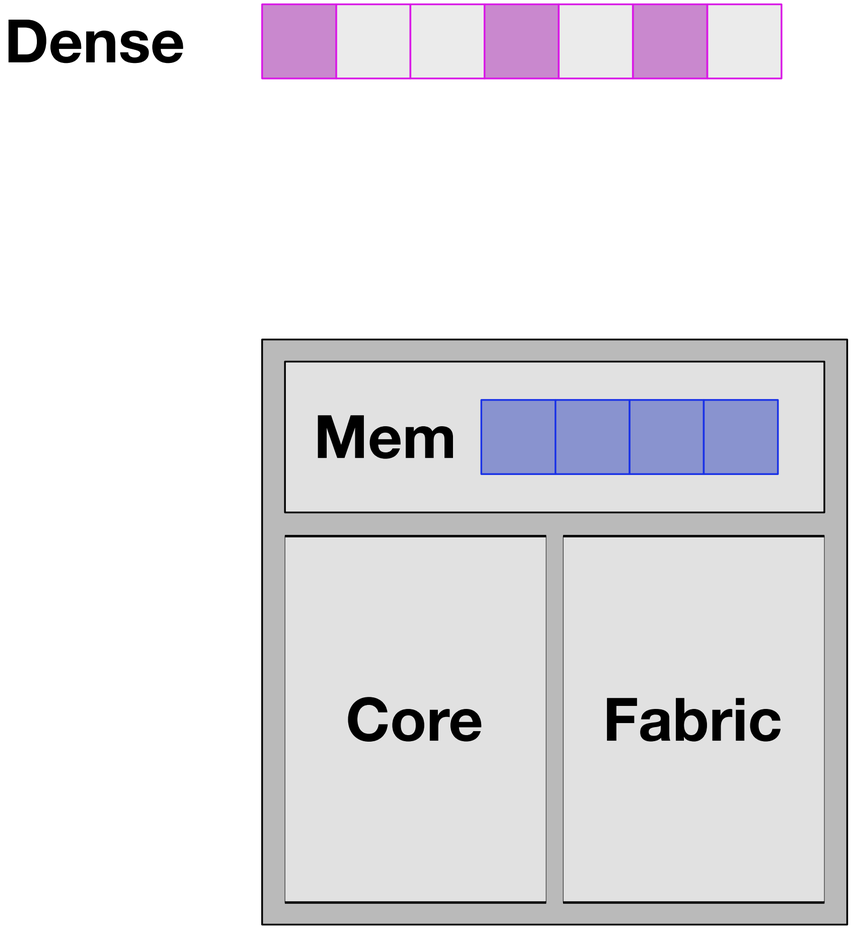
Sparse linear algebra



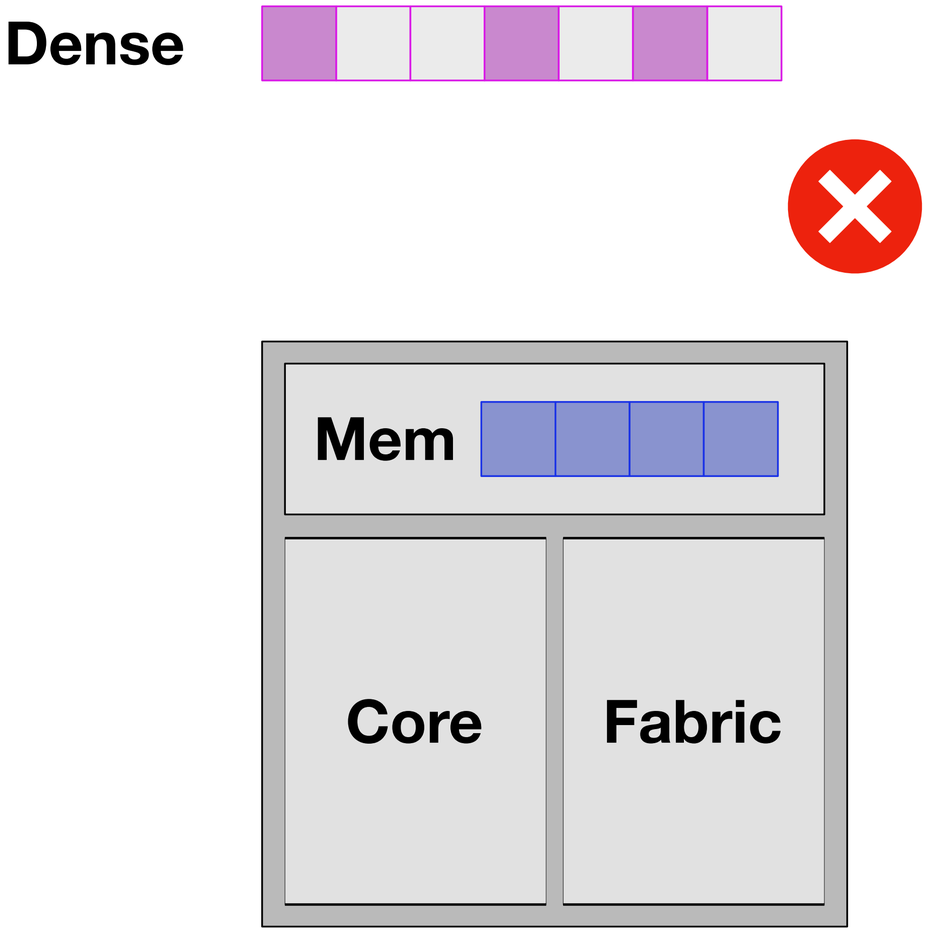
Computer-aided design



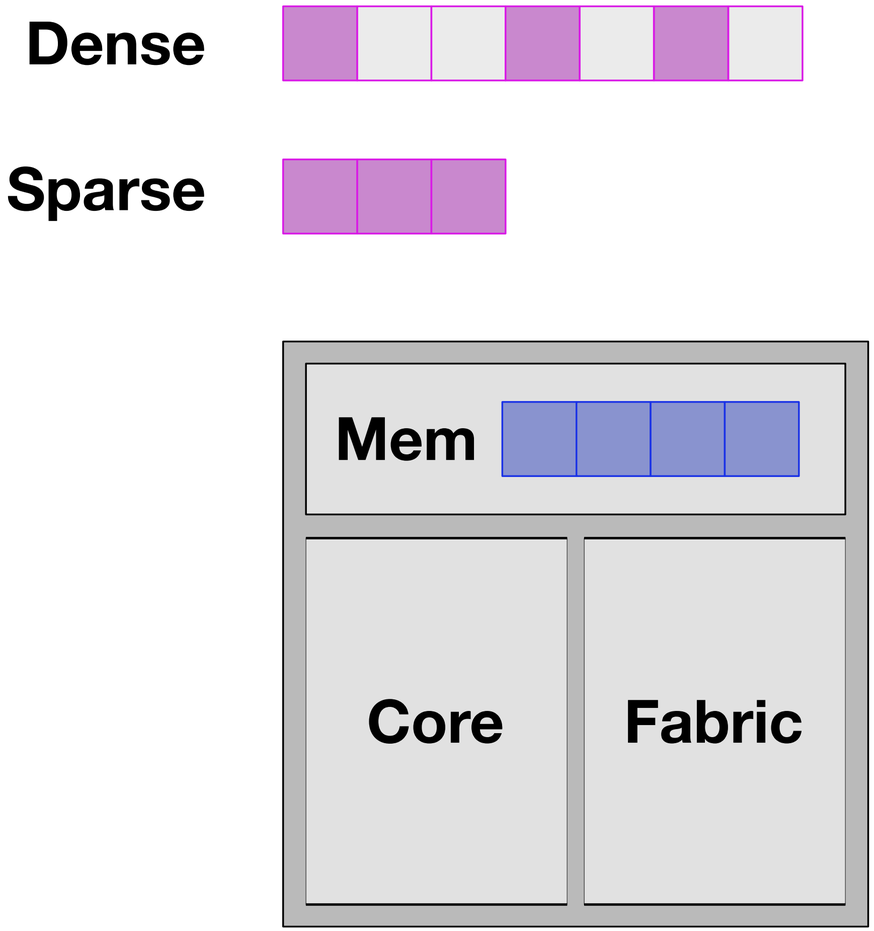
Sparsity lets tiny devices run large apps



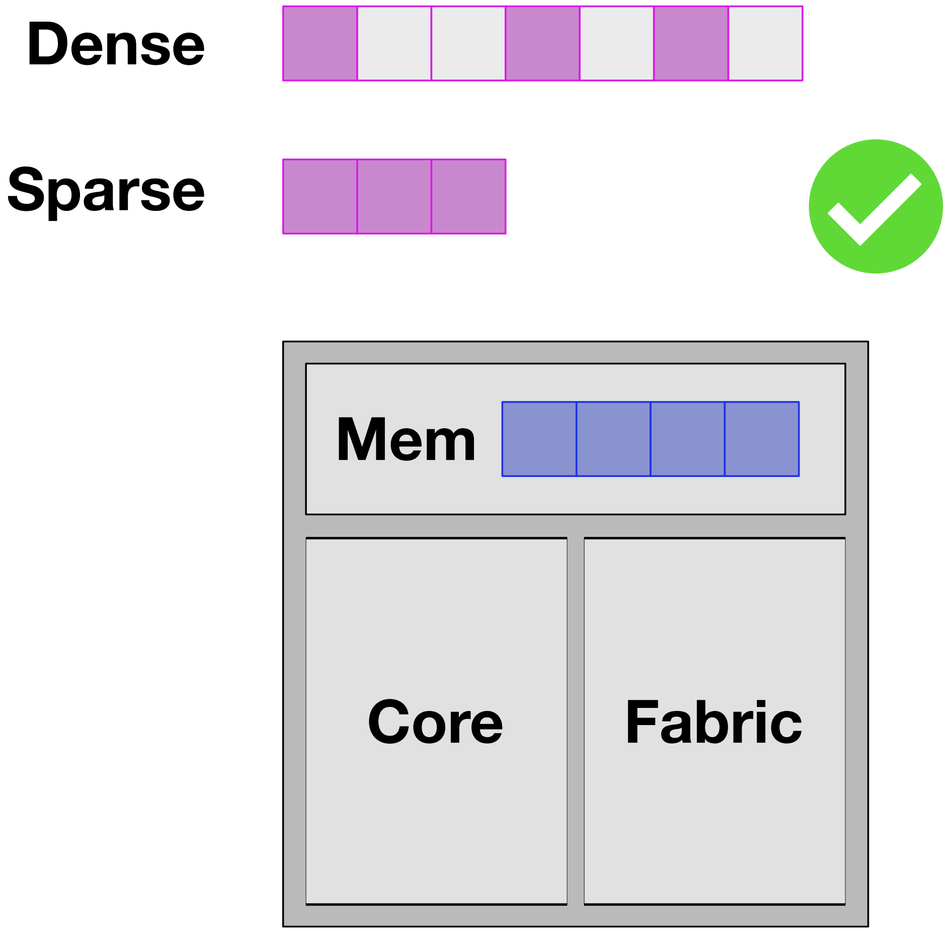
Sparsity lets tiny devices run large apps



Sparsity lets tiny devices run large apps

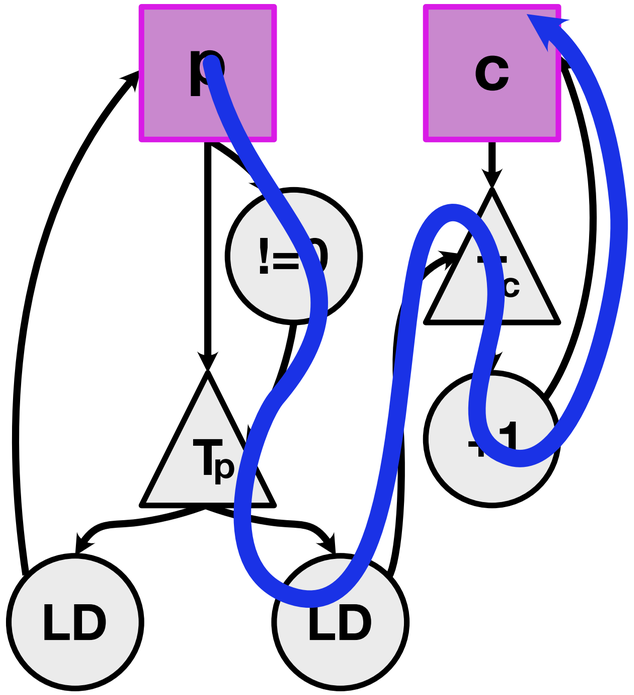


Sparsity lets tiny devices run large apps

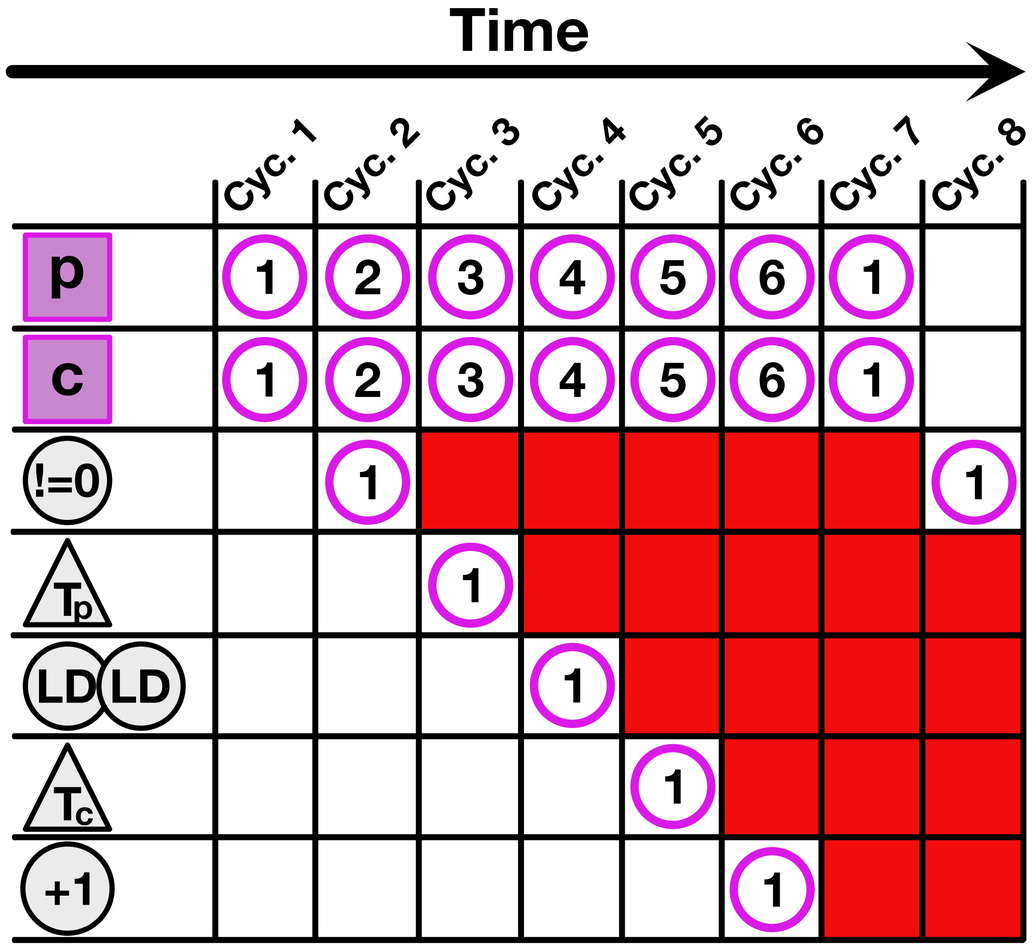


Irregular applications are challenging

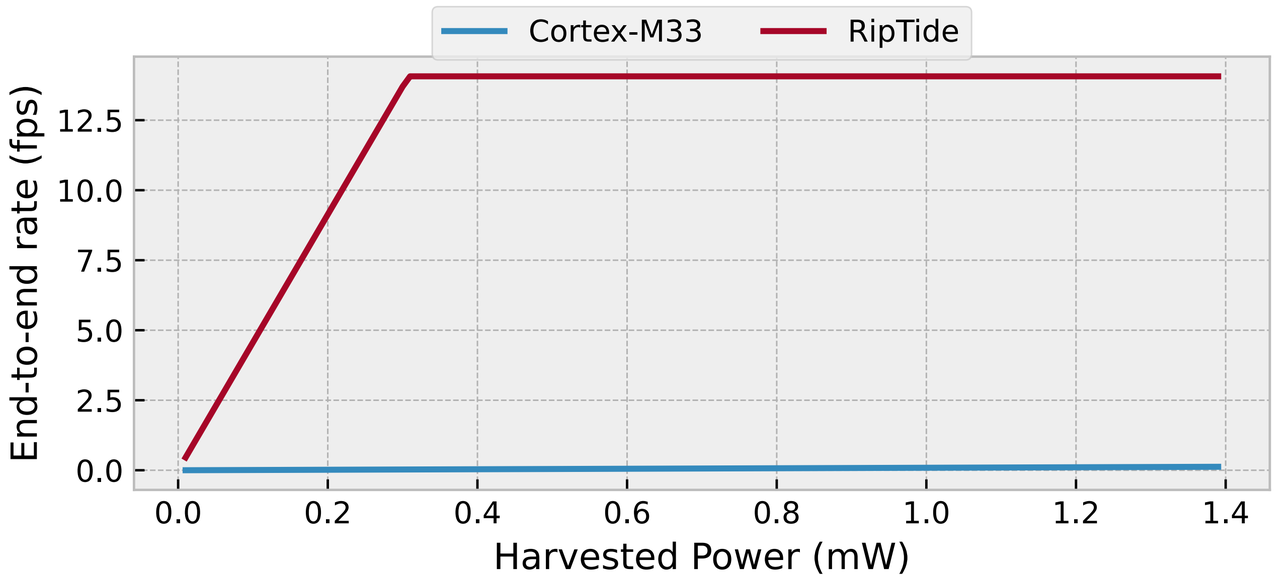
Irregular apps have long IIs



Irregular apps pipeline poorly



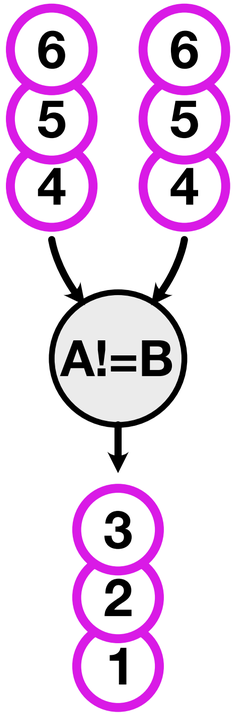
Spatial dataflow + long II = inefficiency



Why do existing processors struggle with irregular apps?

Conservative execution loses parallelism

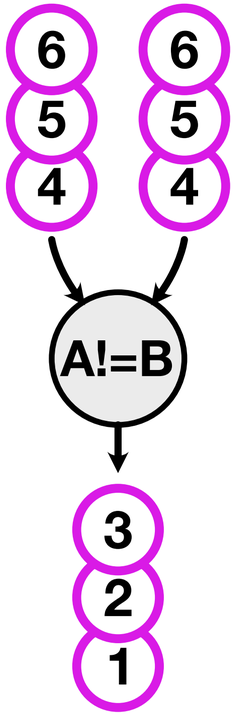
Ordered dataflow saves energy



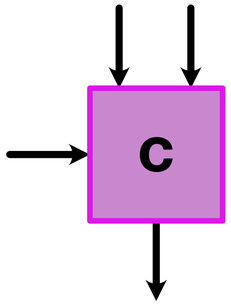
Token

Conservative execution loses parallelism

Ordered dataflow saves energy



Carry gate sequentializes to preserve ordering

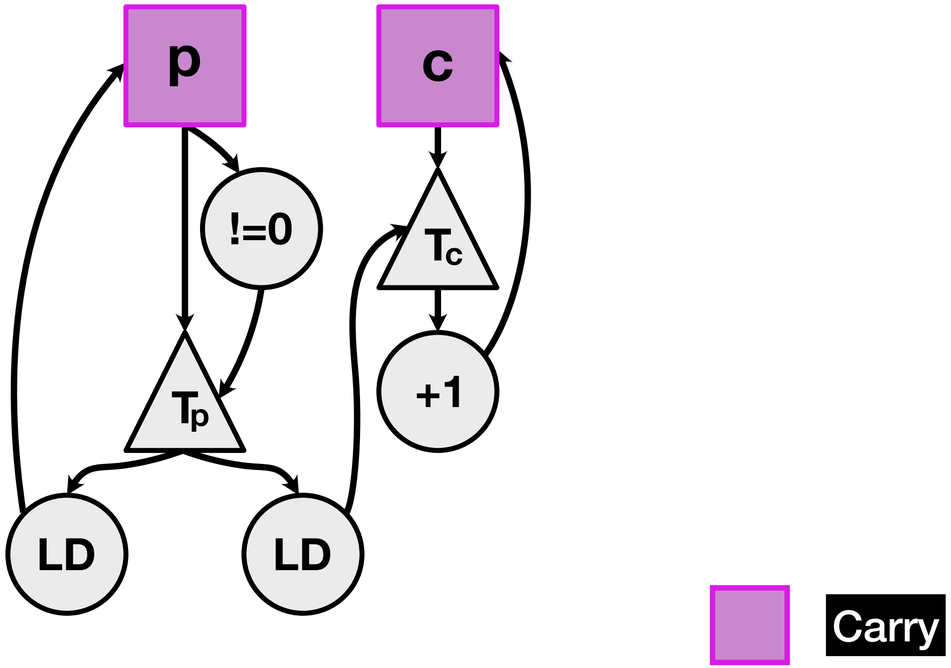


Carry gates manage loops

```
for i = 0..N:
  p = map[i], c = 0
  while p != NULL:
    if p.val: c++
    p = p->next
  ...
```

Carry gates manage loops

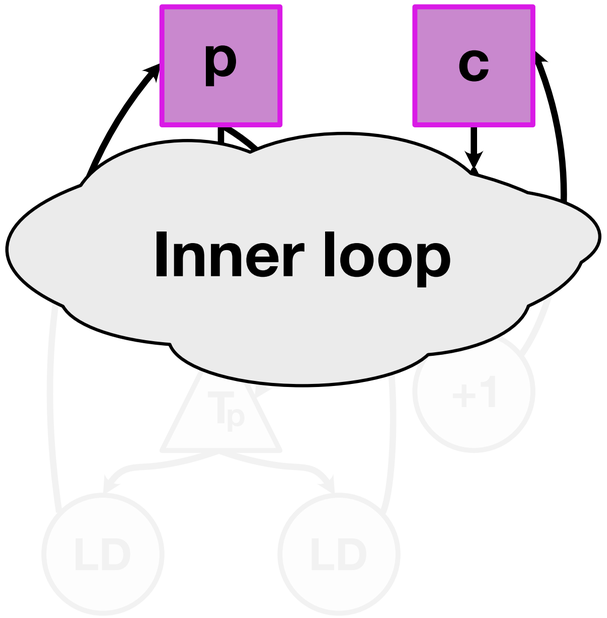
```
for i = 0..N:
  p = map[i], c = 0
  while p != NULL:
    if p.val: c++
    p = p->next
  ...
```



Carry gates manage loops

```
for i = 0..N:
  p = map[i], c = 0
  while NULL:
    ...
```

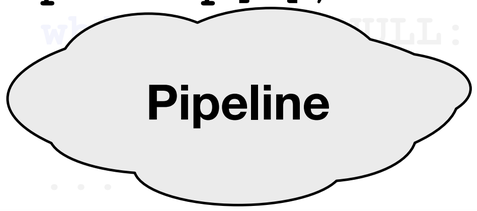
Inner loop

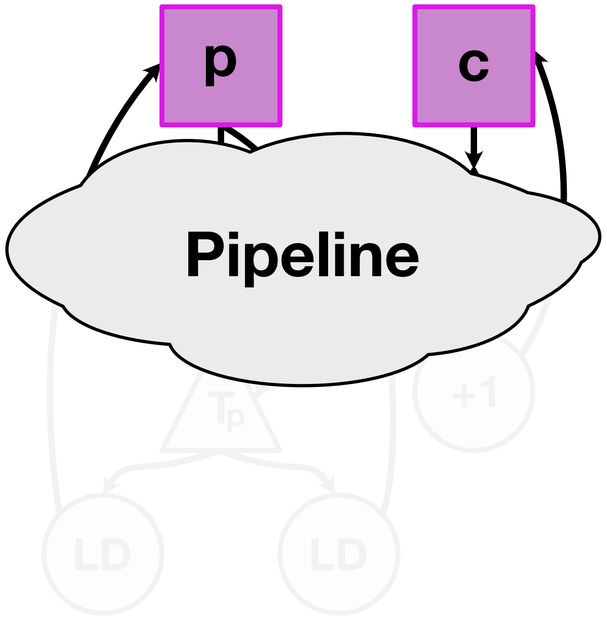


Carry

Carry gates manage loops

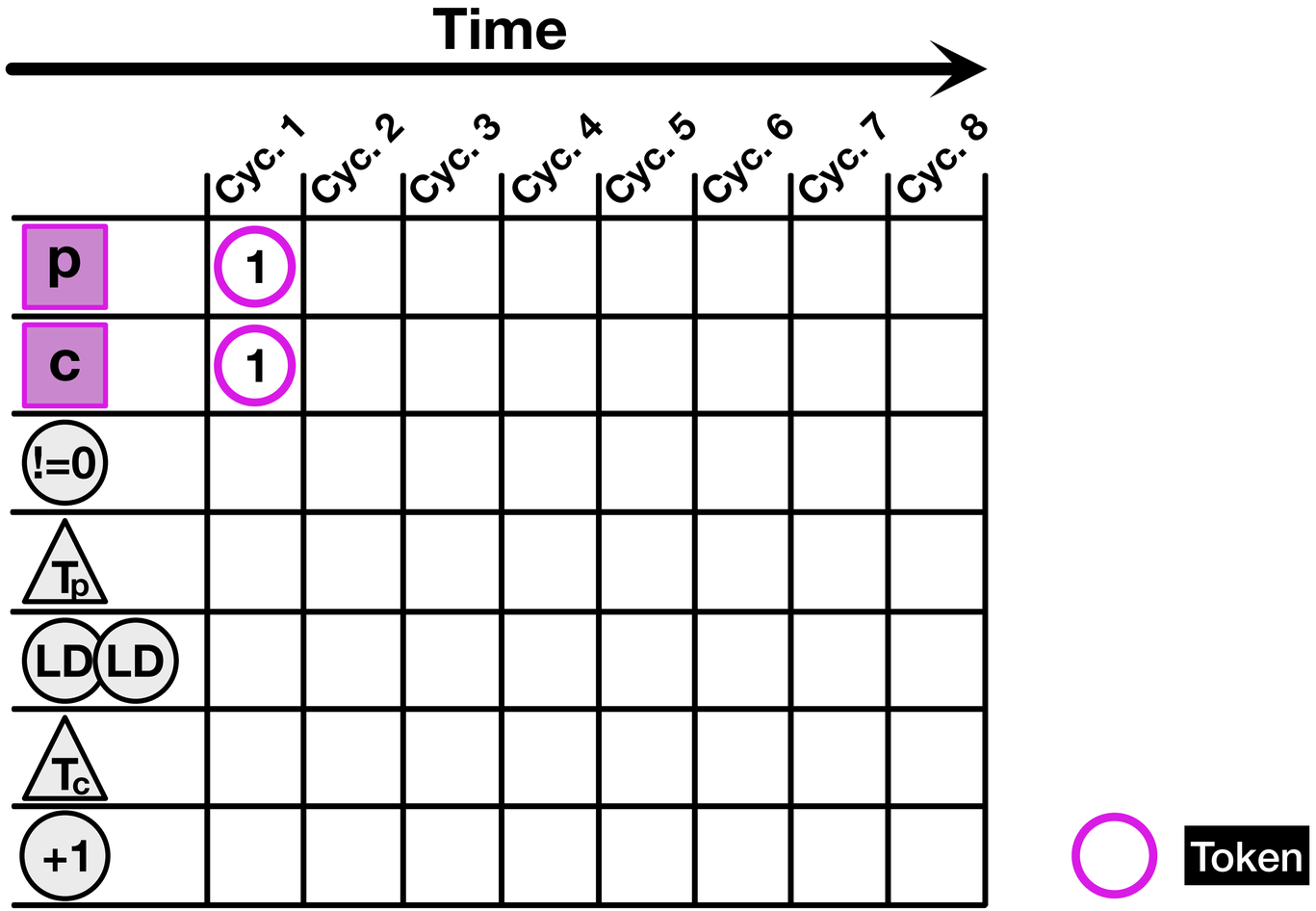
```
for i = 0..N:
  p = map[i], c = 0
```



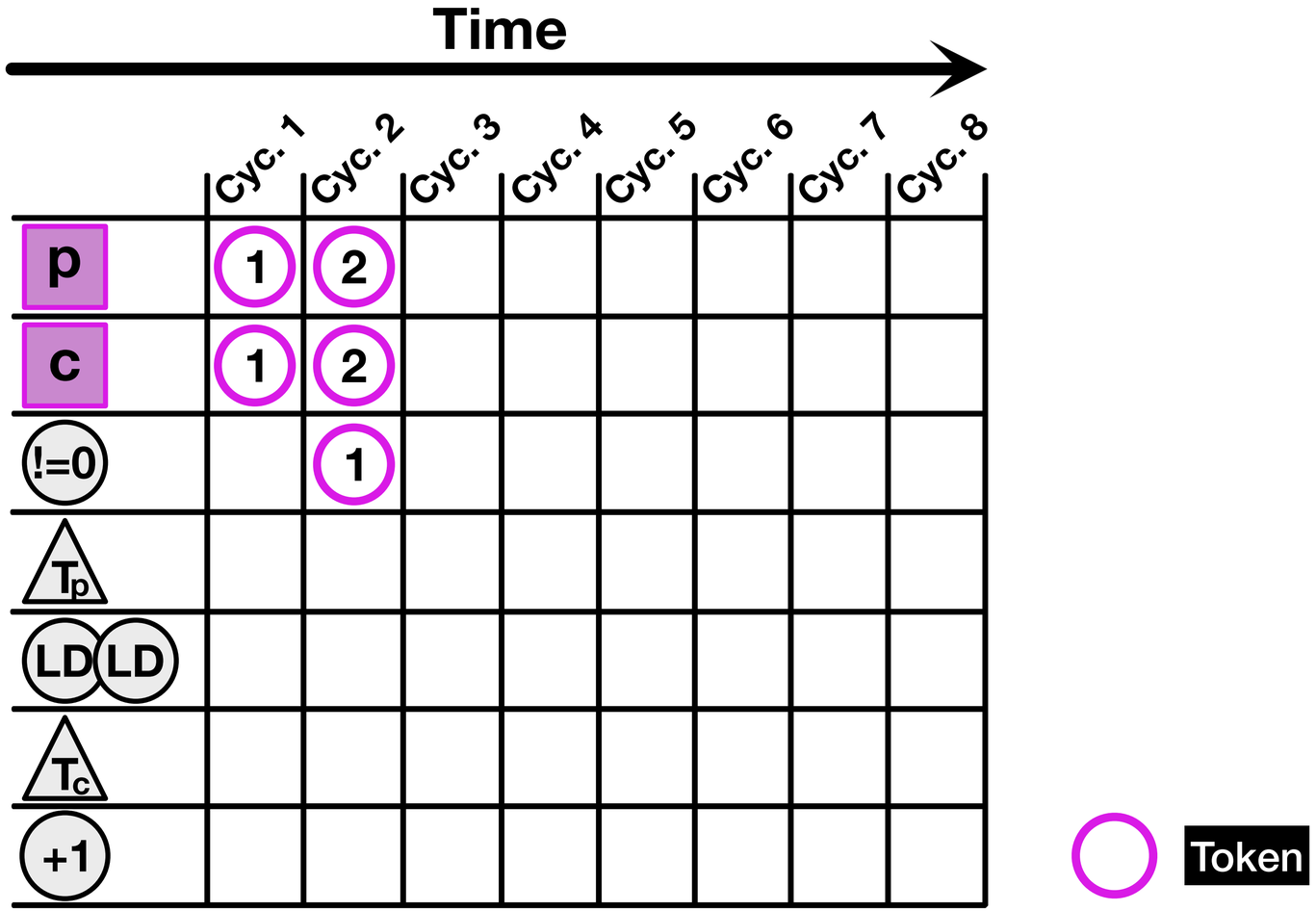


Carry

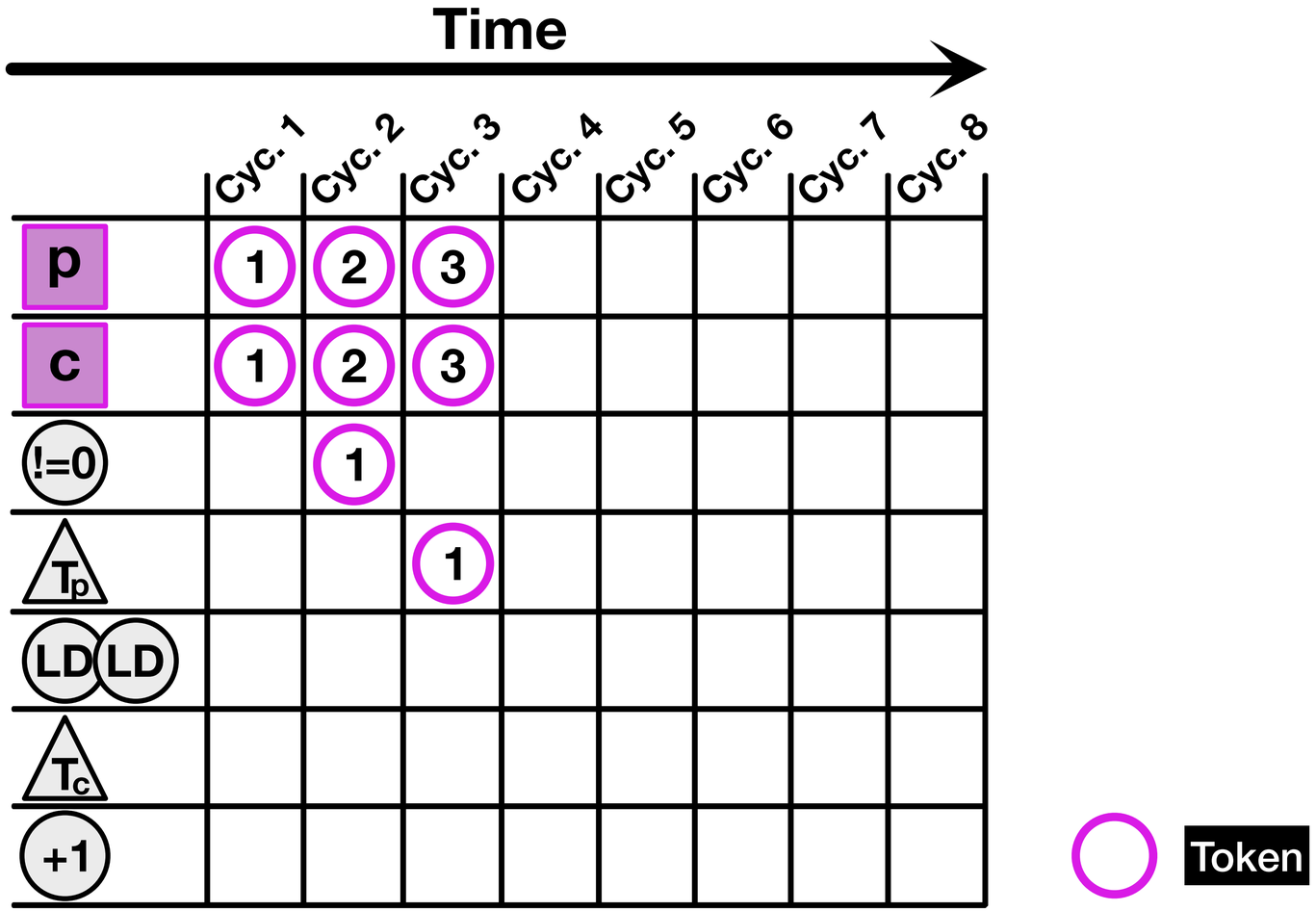
Carry sequentialization kills utilization



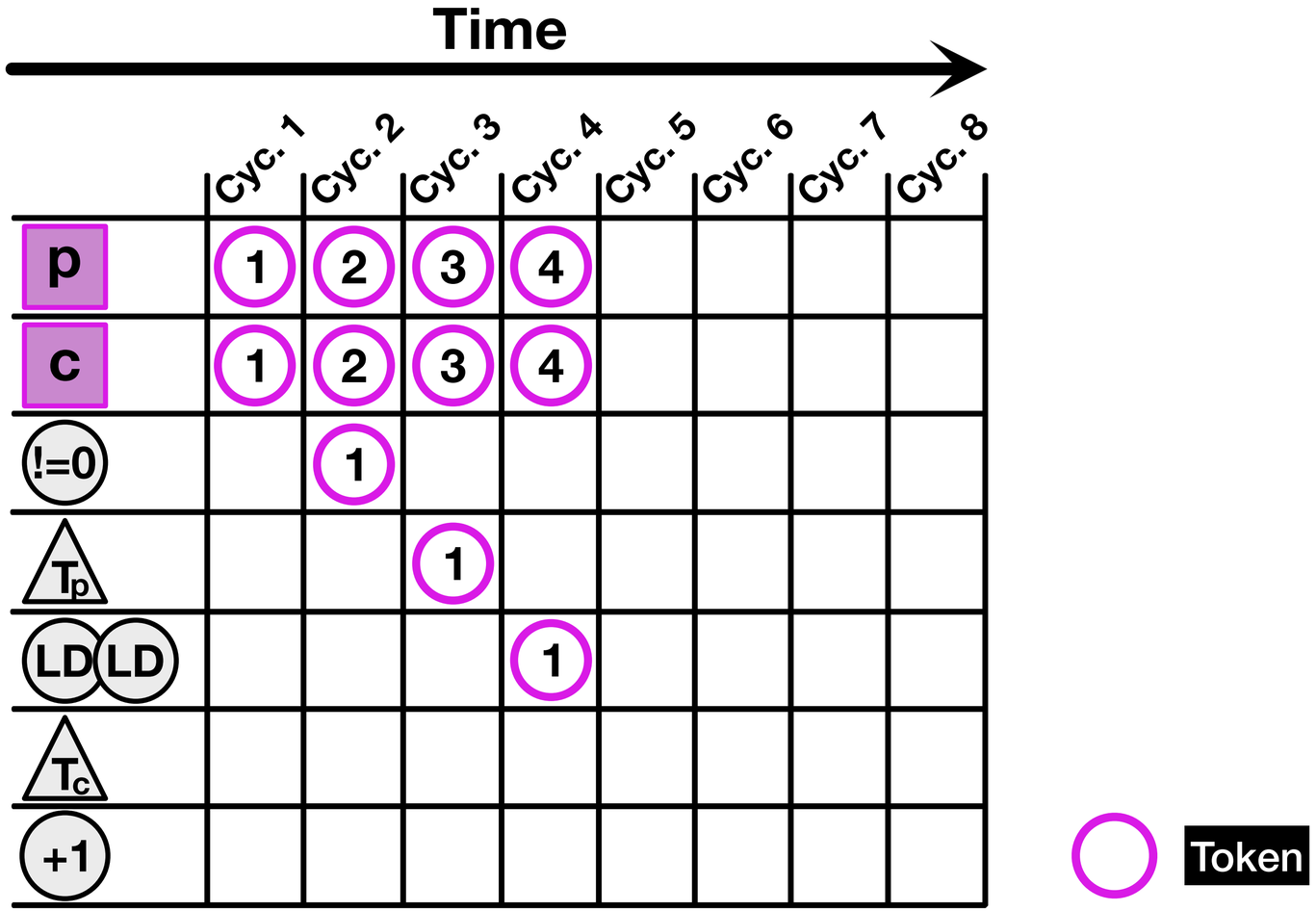
Carry sequentialization kills utilization



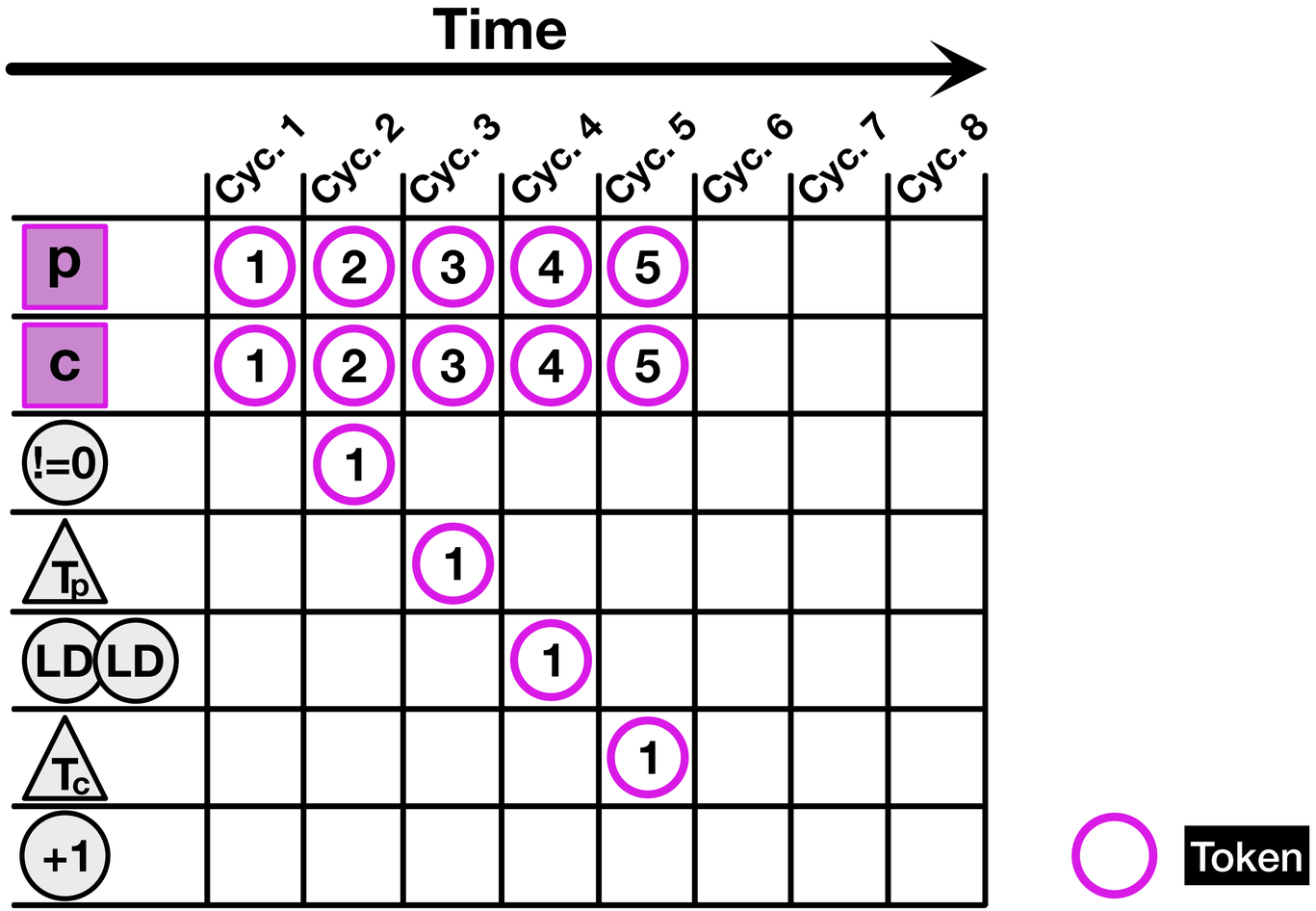
Carry sequentialization kills utilization



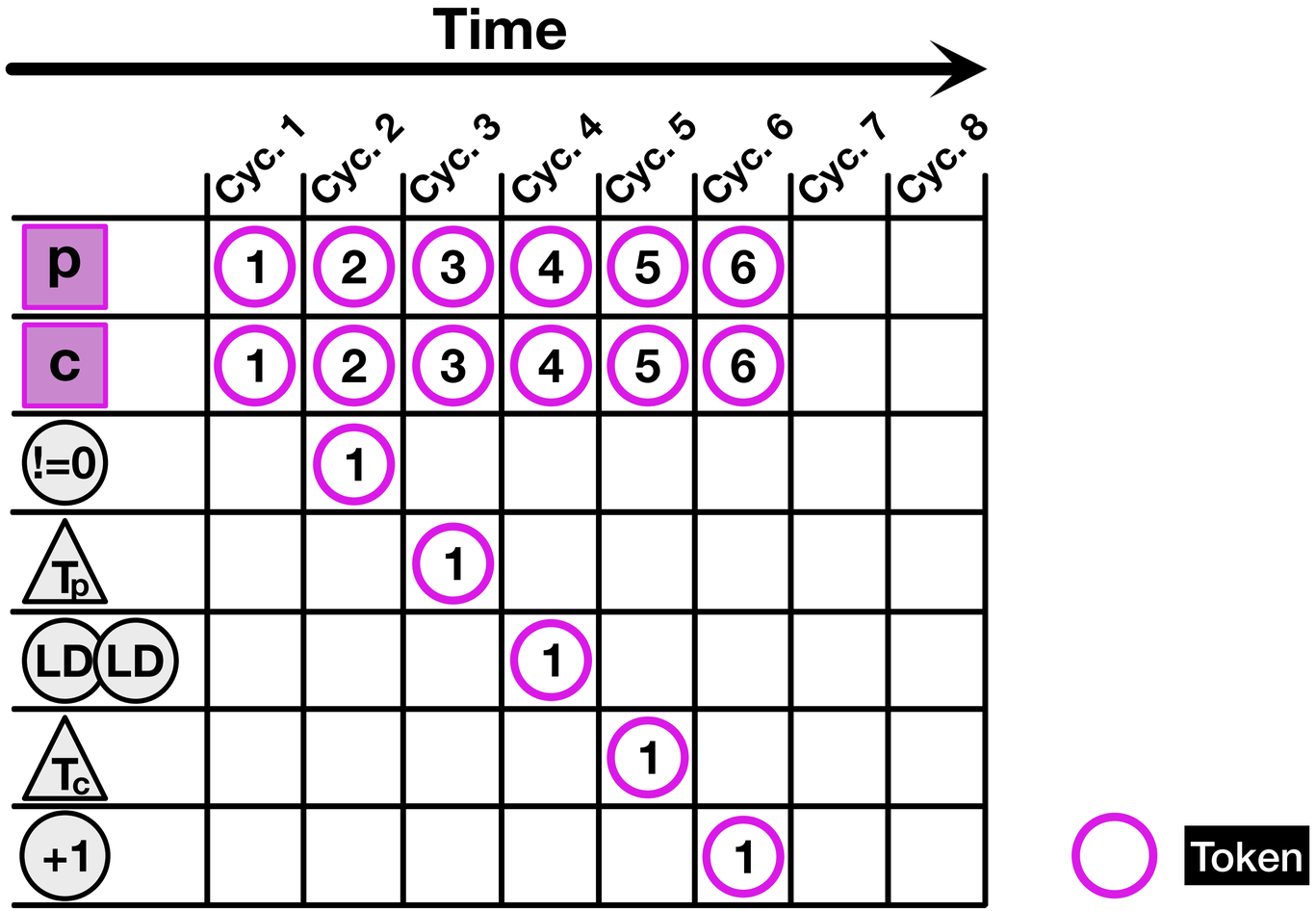
Carry sequentialization kills utilization



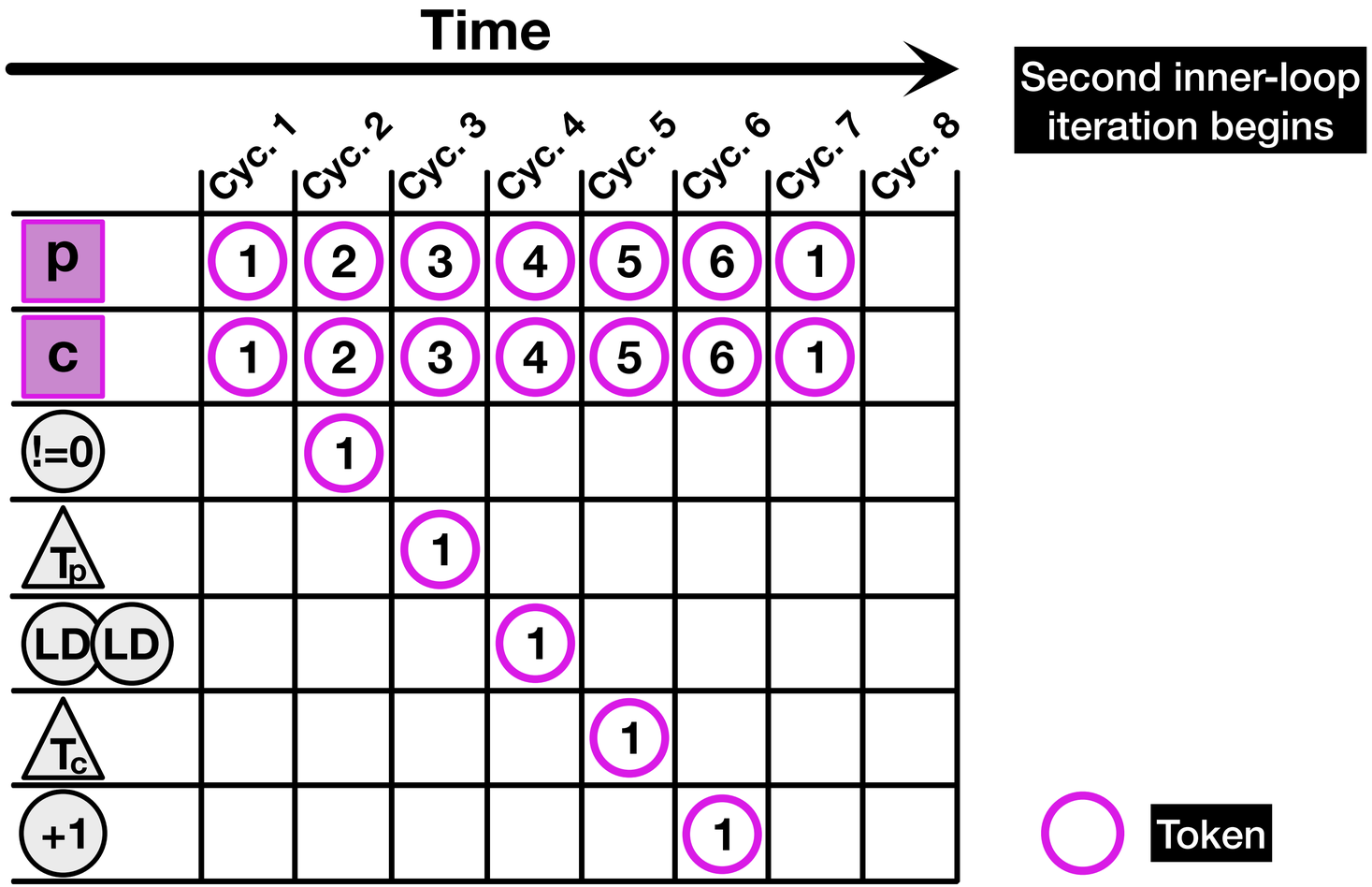
Carry sequentialization kills utilization



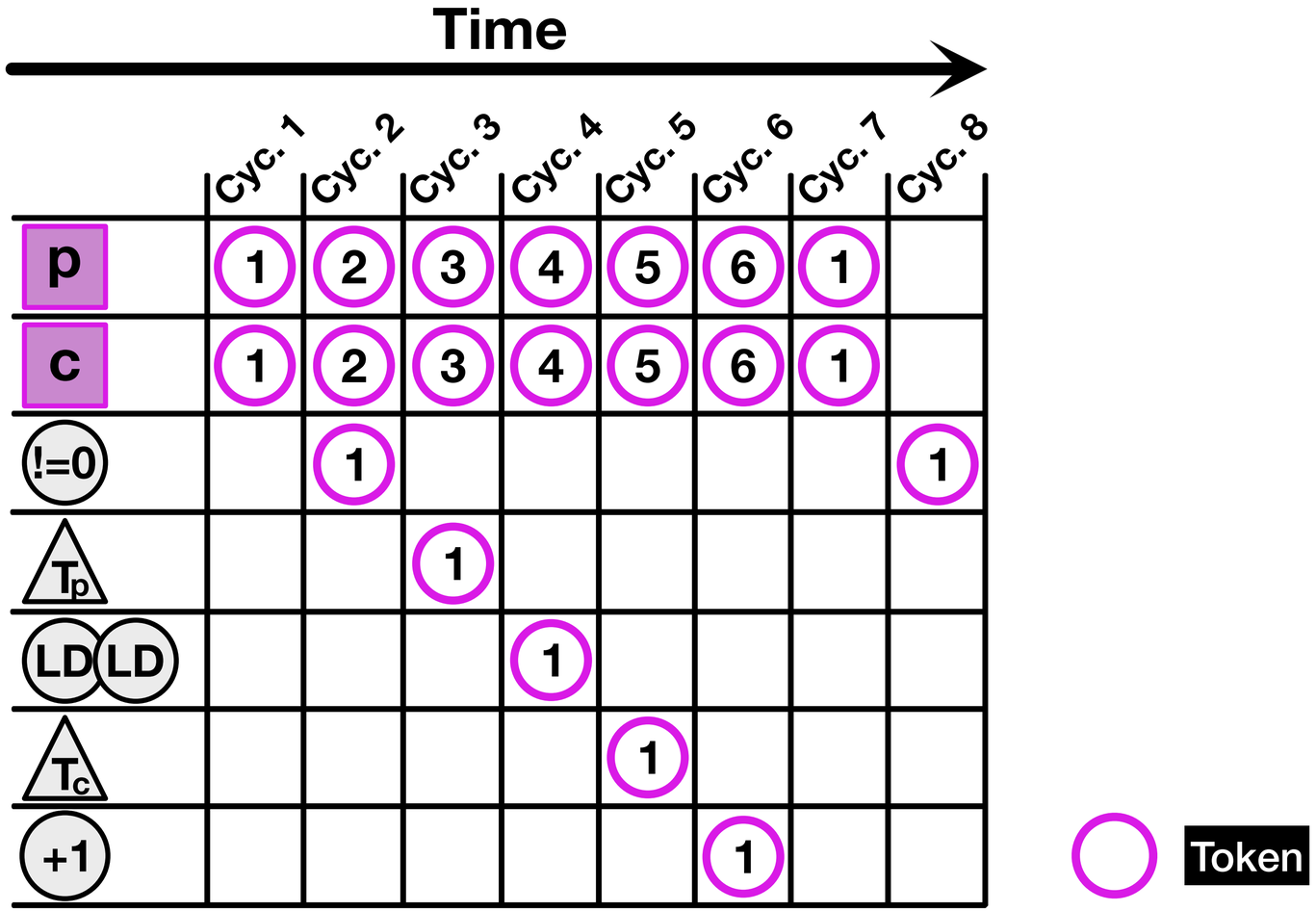
Carry sequentialization kills utilization



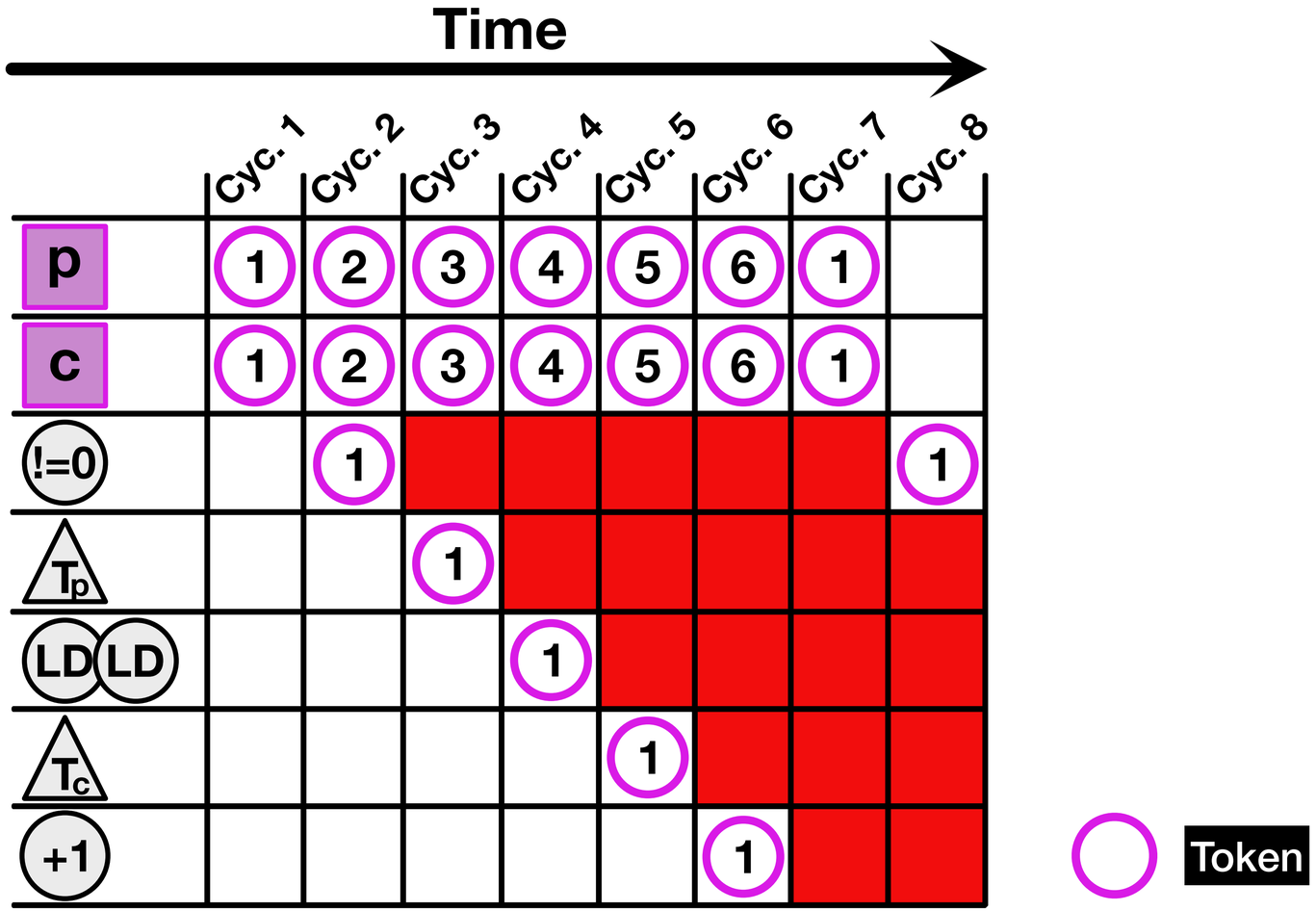
Carry sequentialization kills utilization



Carry sequentialization kills utilization



Carry sequentialization kills utilization



Outline

Spatial dataflow review

Spatial dataflow is efficient + irregular apps are important

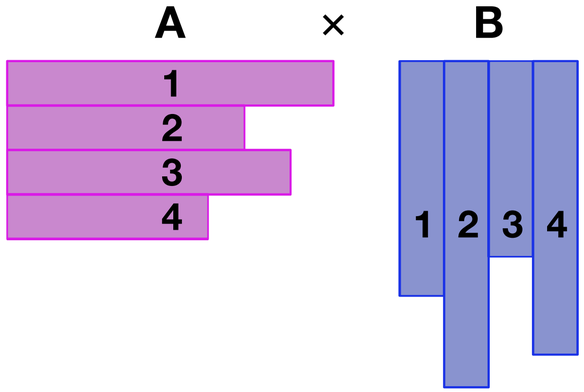
Spatial dataflow not efficient with irregular apps

Make your irregular app look regular

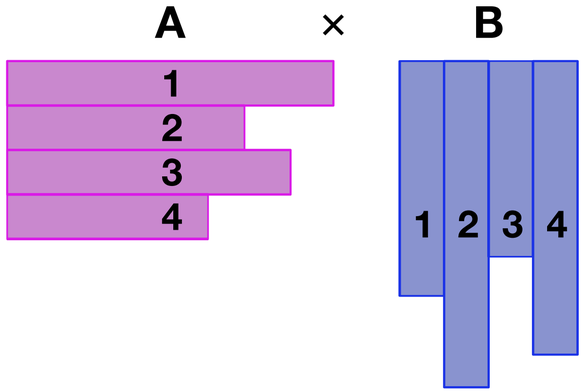
Exploit app properties to increase clock frequency

Eliminate clock and fully fuse

Sparse linalg has abundant parallelism

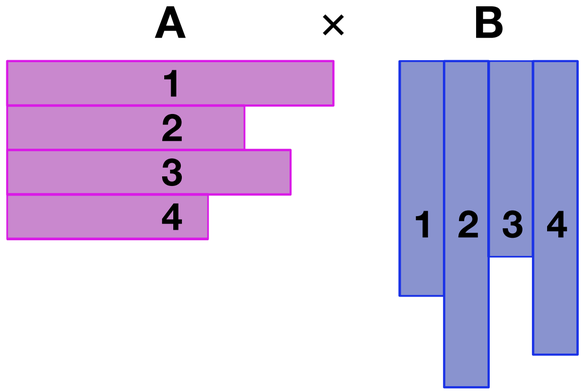


Sparse linalg has abundant parallelism




```
parallel for row = 0..R:
  acc = 0
  ...
  while iA < ARE && iB < BCE
    if a_col == b_col:...
    if a_col <= b_col:...
    if a_col >= b_col:...
    acc = new_acc
    iA = new_iA
    iB = new_iB
```

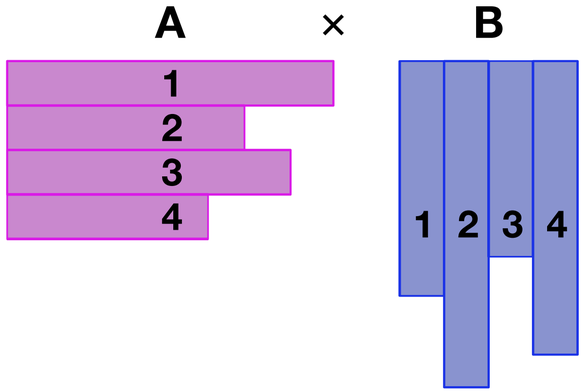
Sparse linalg has abundant parallelism




```
parallel for row = 0..R:
  acc = 0
  ...
  while iA < ARE && iB < BCE
    if a_col == b_col:...
    if a_col <= b_col:...
    if a_col >= b_col:...
    acc = new_acc
    iA = new_iA
    iB = new_iB
```

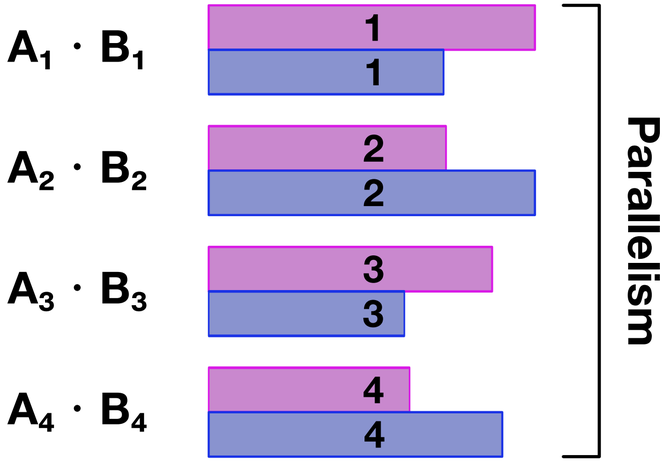
Sparse dot

Sparse linalg has abundant parallelism

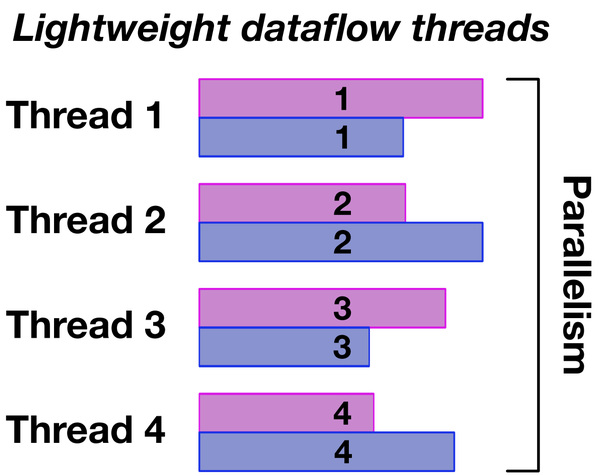



```
parallel for row = 0..R:
  acc = 0
  ...
  while iA < ARE && iB < BCE
    if a_col == b_col:...
    if a_col <= b_col:...
    if a_col >= b_col:...
    acc = new_acc
    iA = new_iA
    iB = new_iB
```

Sparse dot

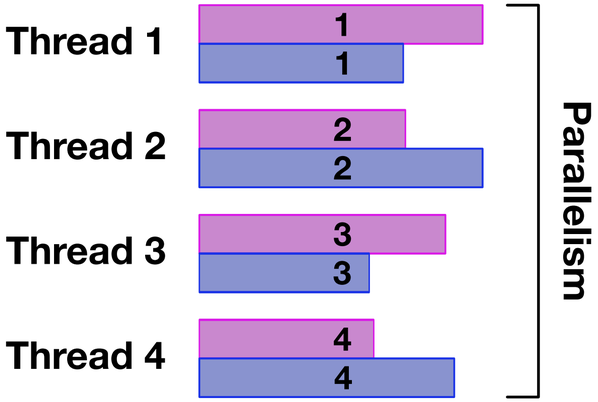


Pipestitch overview



Pipestitch overview

Lightweight dataflow threads

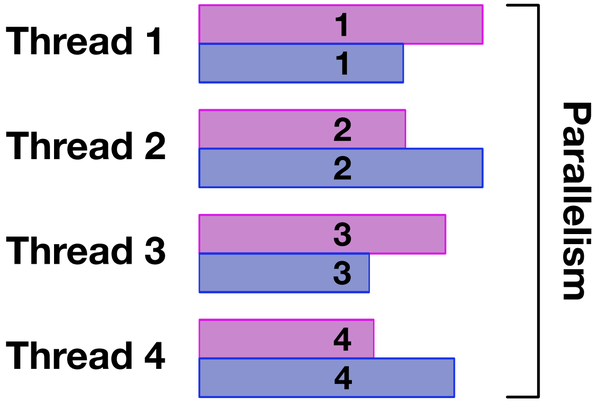


Expressed with parallel for

```
parallel for row = 0..R:
  acc = 0
  ...
  while iA < ARE && iB < BCE
    if a_col == b_col...
    if a_col <= b_col...
    if a_col >= b_col...
    acc = new_acc
    iA = new_iA
    iB = new_iB
```

Pipestitch overview

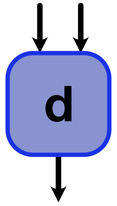
Lightweight dataflow threads



Expressed with parallel for

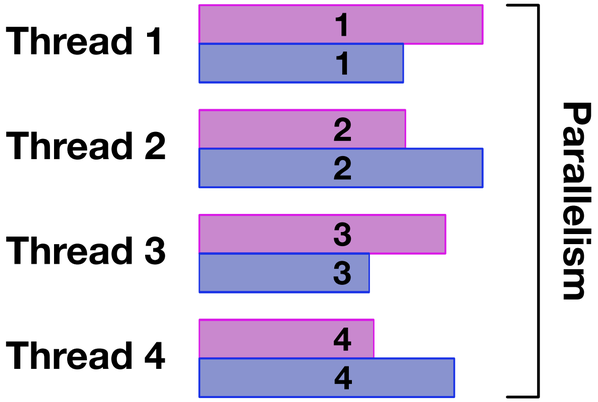
```
parallel for row = 0..R:
  acc = 0
  ...
  while iA < ARE && iB < BCE
    if a_col == b_col...
    if a_col <= b_col...
    if a_col >= b_col...
    acc = new_acc
    iA = new_iA
    iB = new_iB
```

Threads launched by new dispatch gate



Pipestitch overview

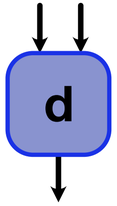
Lightweight dataflow threads



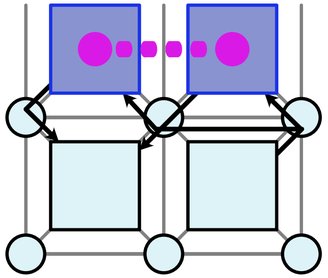
Expressed with parallel for

```
parallel for row = 0..R:
  acc = 0
  ...
  while iA < ARE && iB < BCE
    if a_col == b_col...
    if a_col <= b_col...
    if a_col >= b_col...
    acc = new_acc
    iA = new_iA
    iB = new_iB
```

Threads launched by new dispatch gate



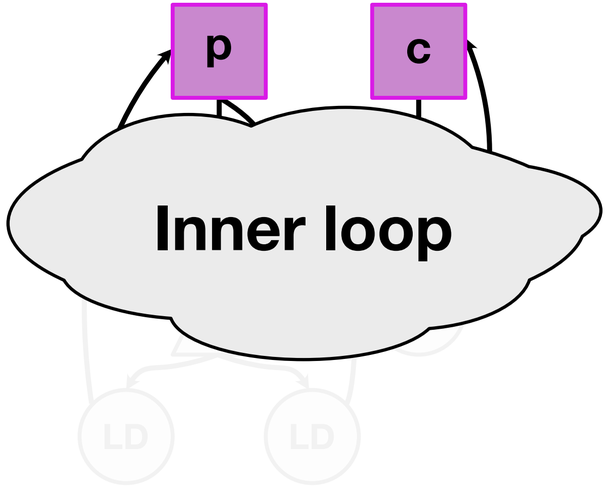
Sync network orders threads



Dispatch launches threads

Dispatch replaces carry when threading

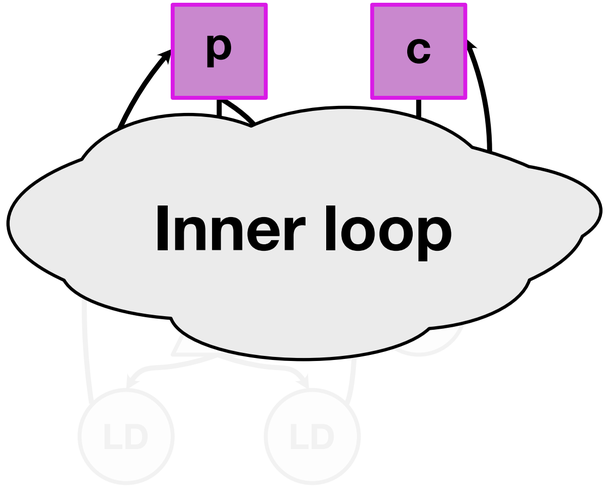
```
for i = 0..N:
  p = map[i], c = 0
  while p != NULL:
    if p.val: c++
    p = p->next
  ...
```



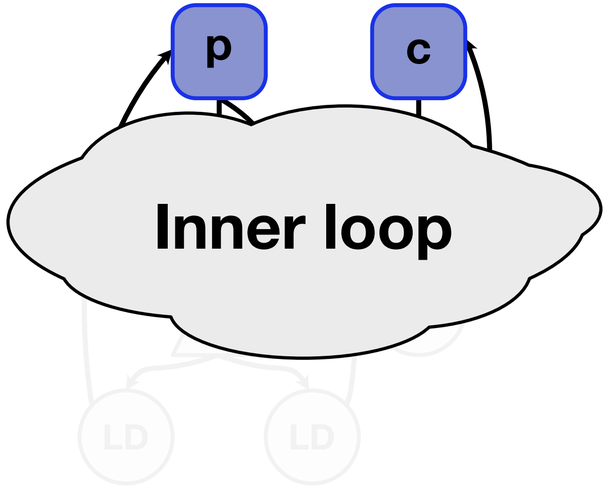
Carry

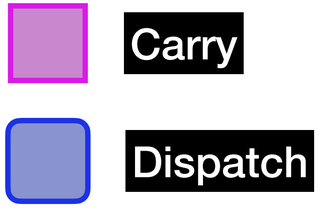
Dispatch replaces carry when threading

```
for i = 0..N:
  p = map[i], c = 0
  while p != NULL:
    if p.val: c++
    p = p->next
  ...
```

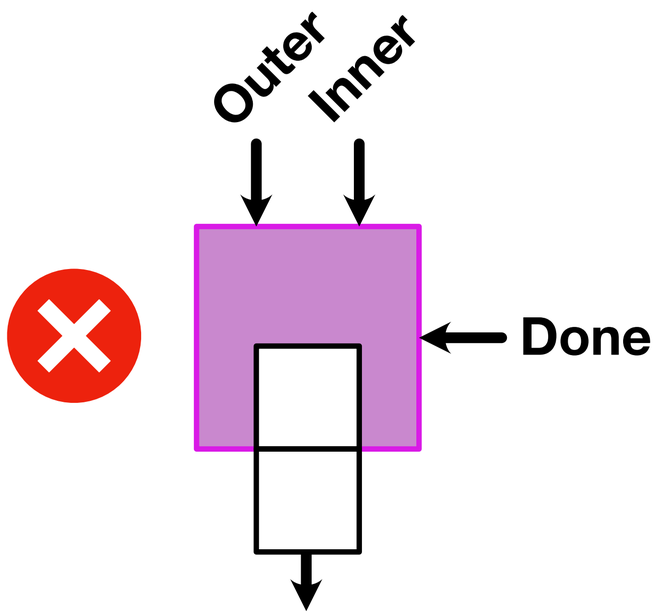



```
parallel for i = 0..N:
  p = map[i], c = 0
  while p != NULL:
    if p.val: c++
    p = p->next
  ...
```

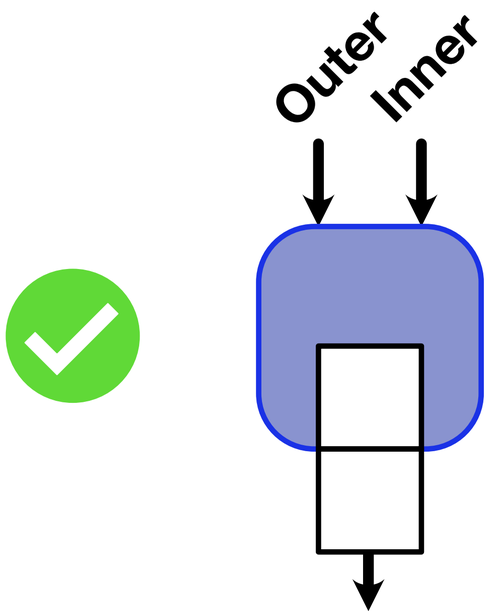




Dispatch unblocks threads



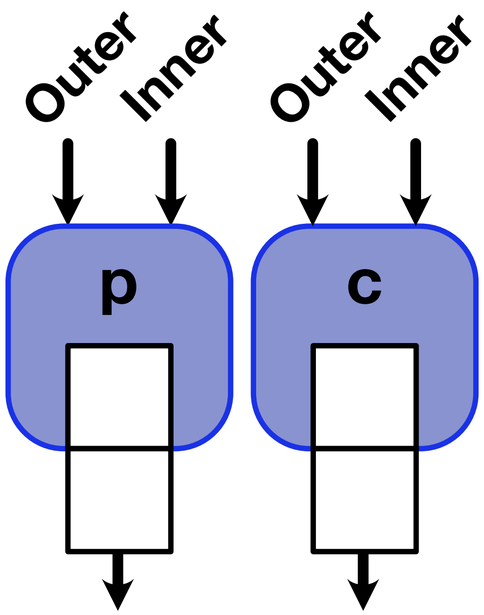
Blocking



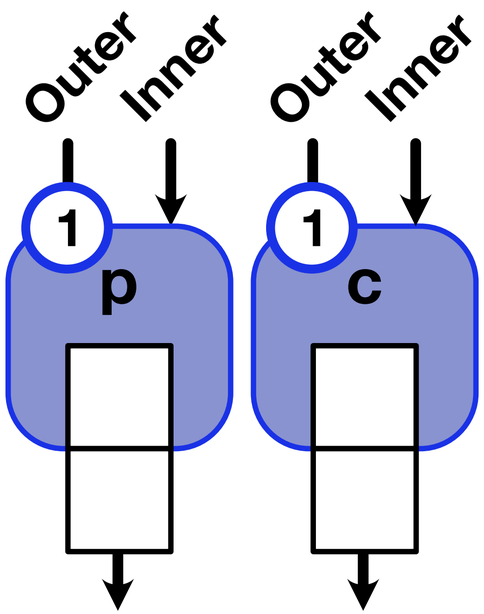
Non-blocking

Threading can violate ordering

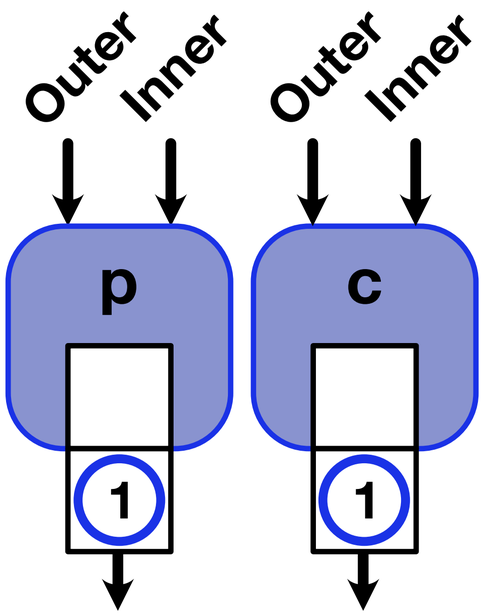
Ordering violated without synchronization



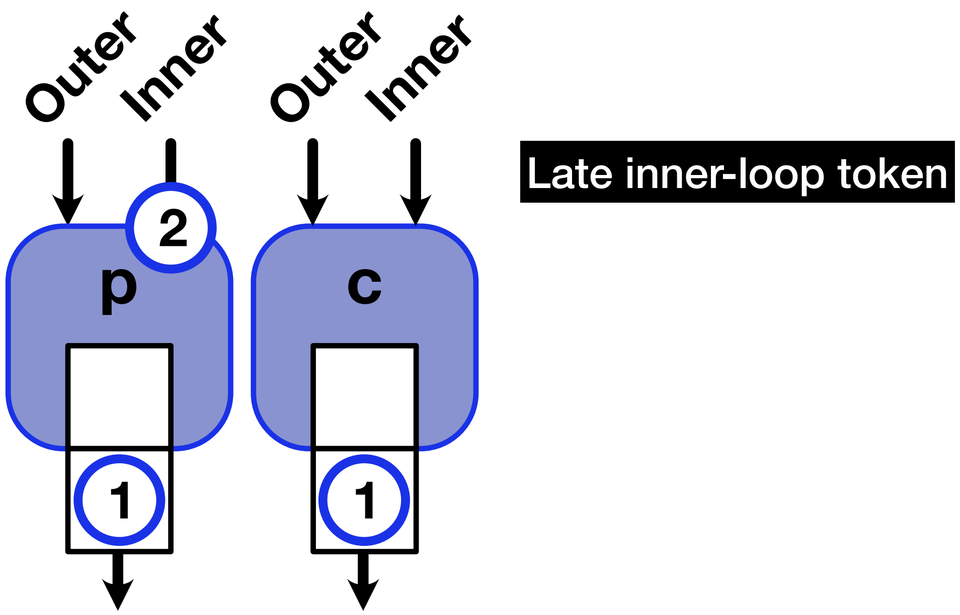
Ordering violated without synchronization



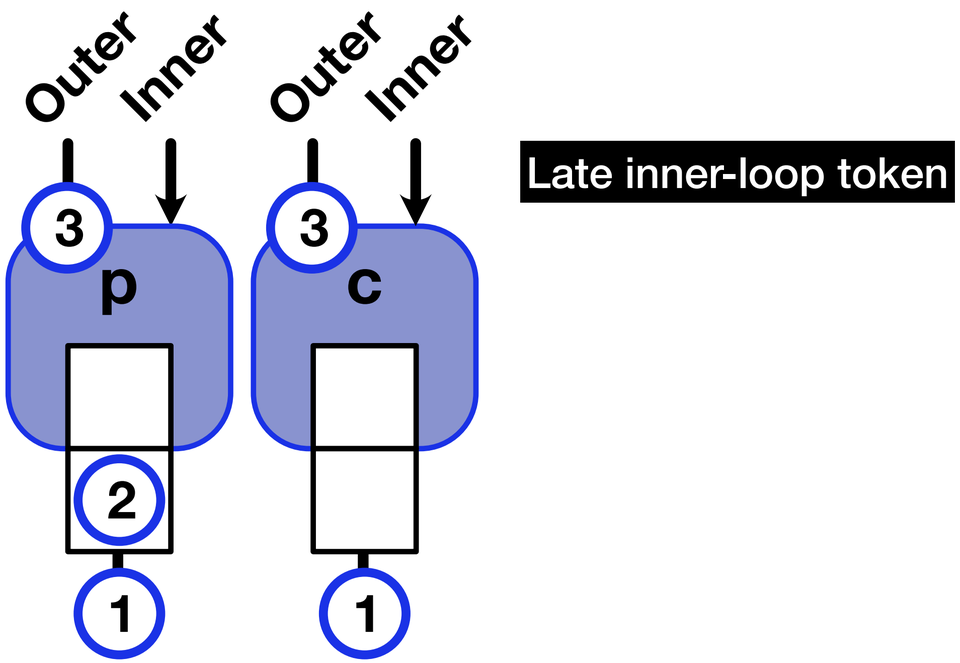
Ordering violated without synchronization



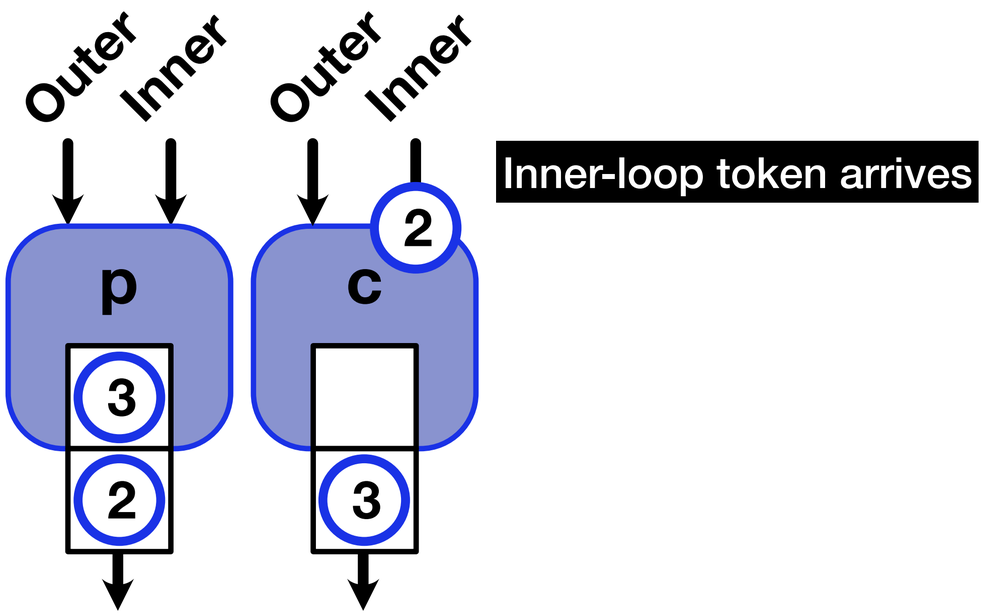
Ordering violated without synchronization



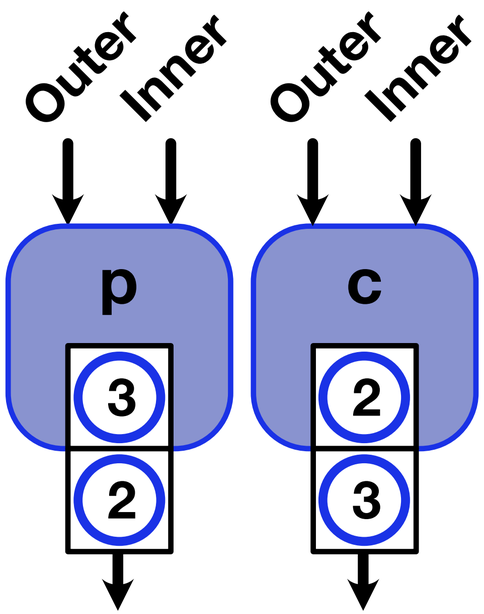
Ordering violated without synchronization



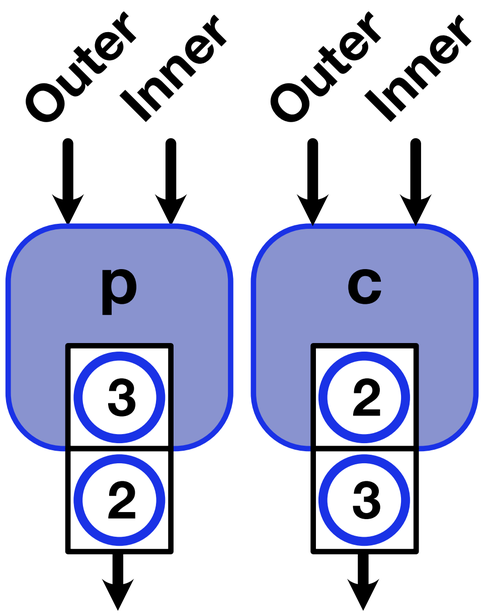
Ordering violated without synchronization



Ordering violated without synchronization



Ordering violated without synchronization

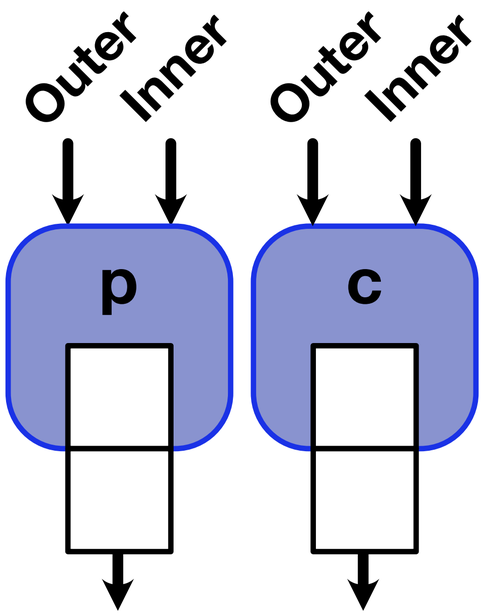


Mismatched tokens

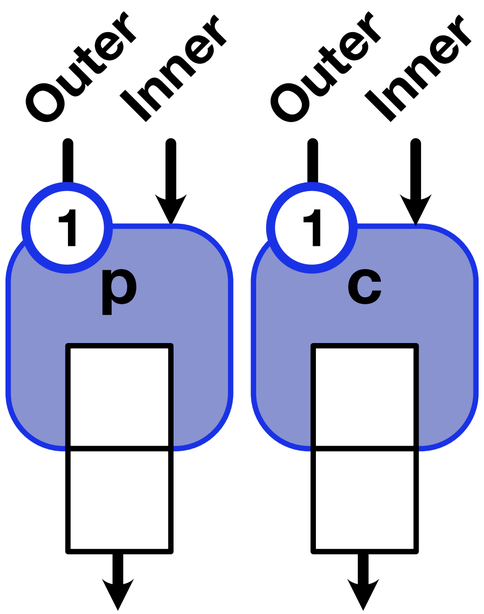
Dispatch

Synchronization restores order

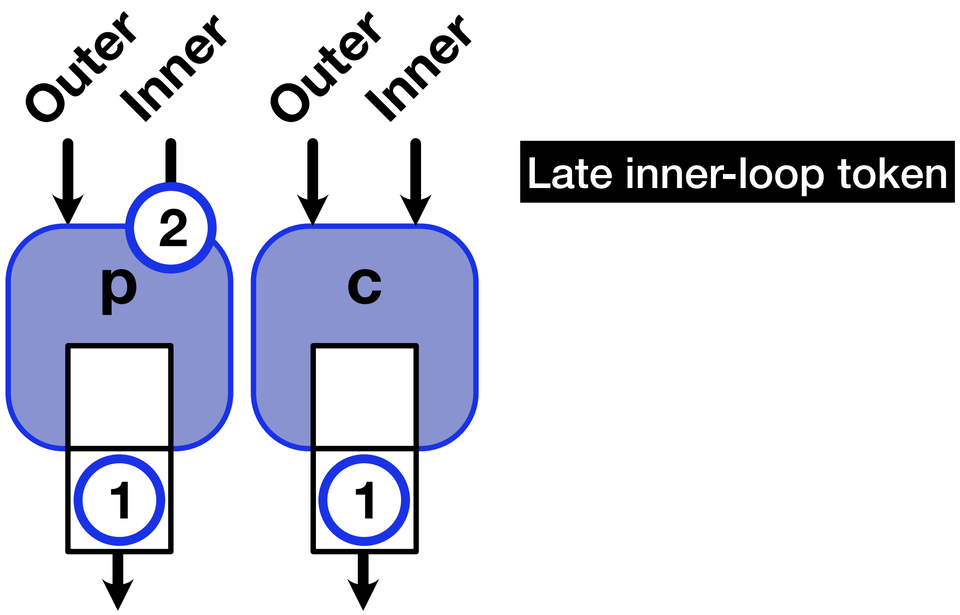
Synchronization fixes ordering violations



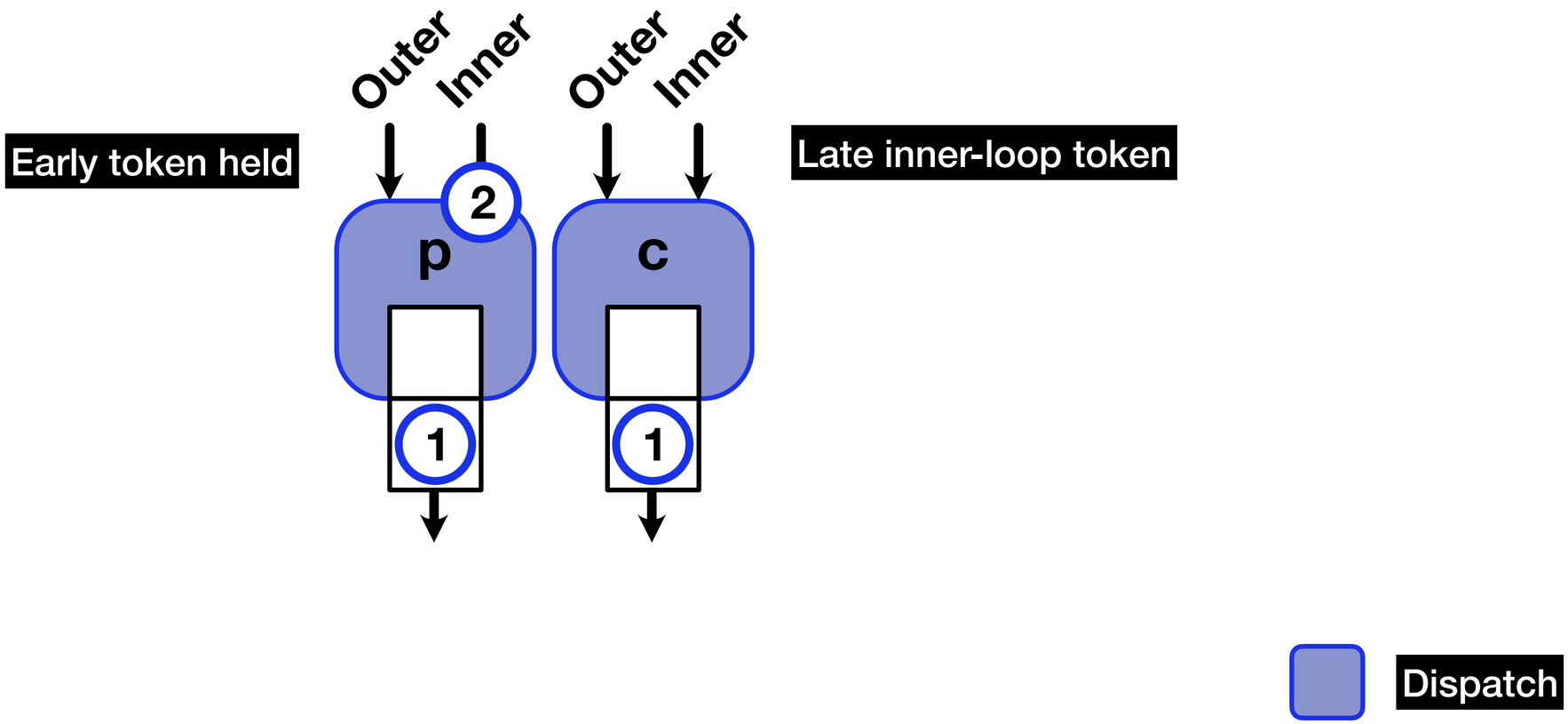
Synchronization fixes ordering violations



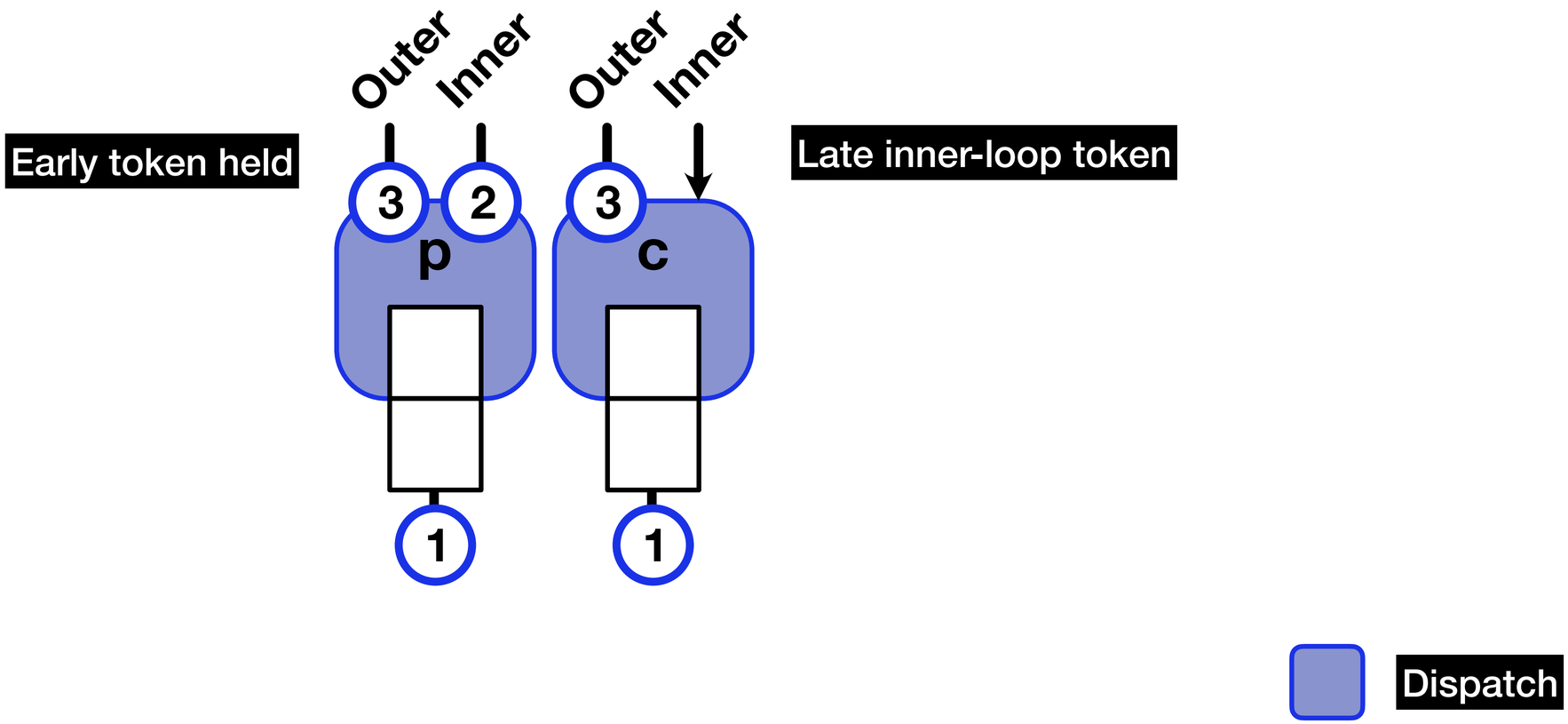
Synchronization fixes ordering violations



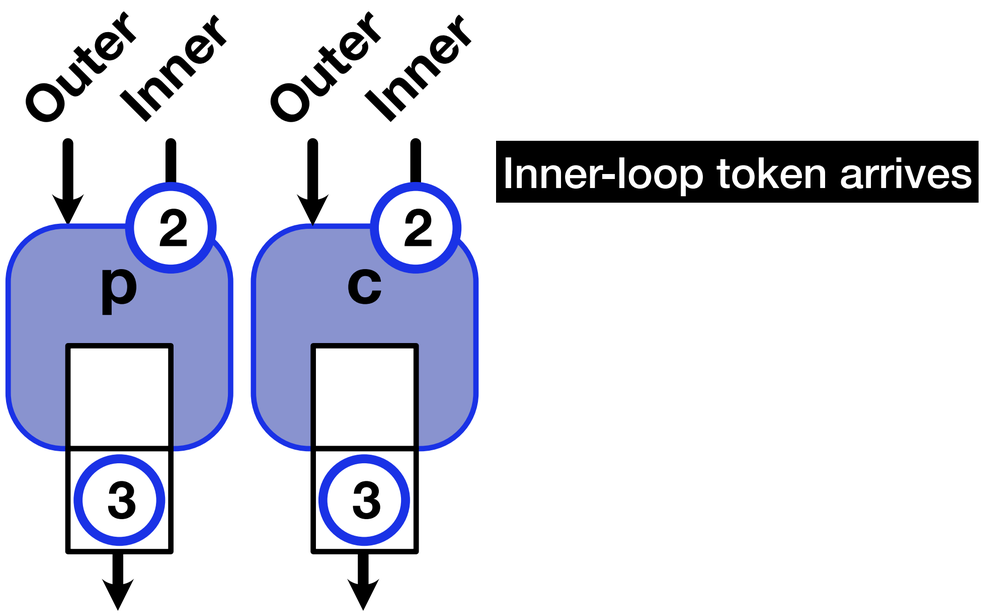
Synchronization fixes ordering violations



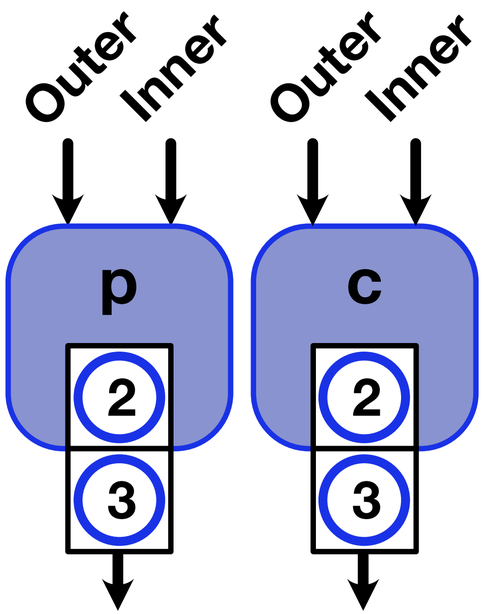
Synchronization fixes ordering violations



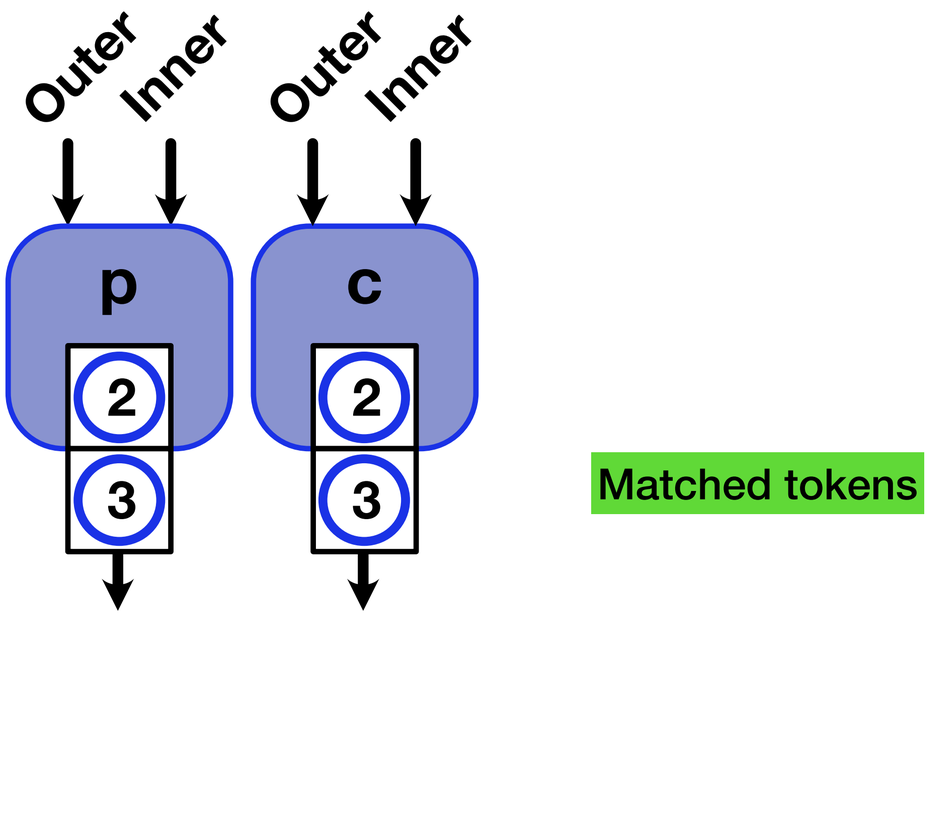
Synchronization fixes ordering violations



Synchronization fixes ordering violations



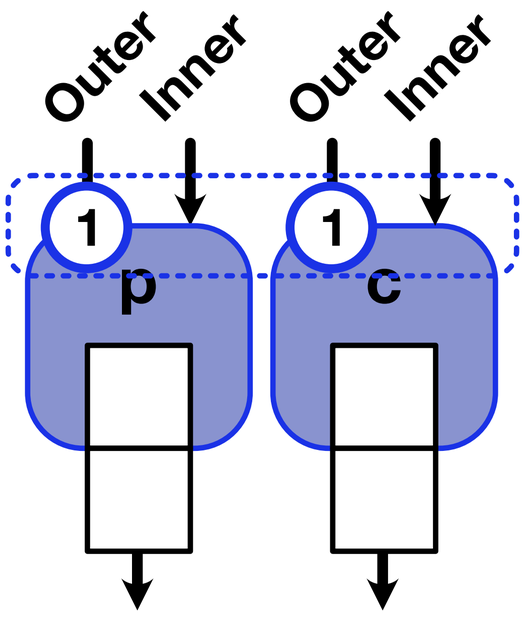
Synchronization fixes ordering violations



Defining ordering requirements

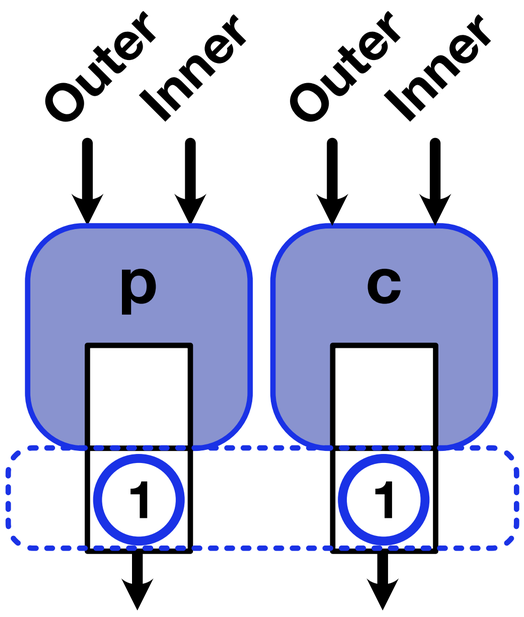
Synchronization accepts token groups

Complete set of
outer-loop tokens



Synchronization accepts token groups

Complete set of
outer-loop tokens

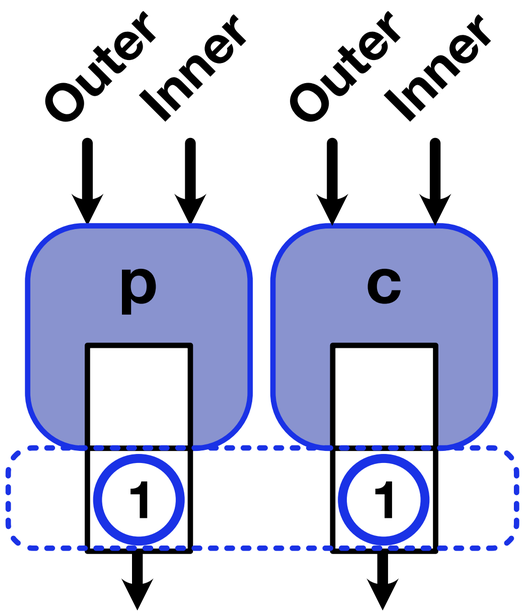


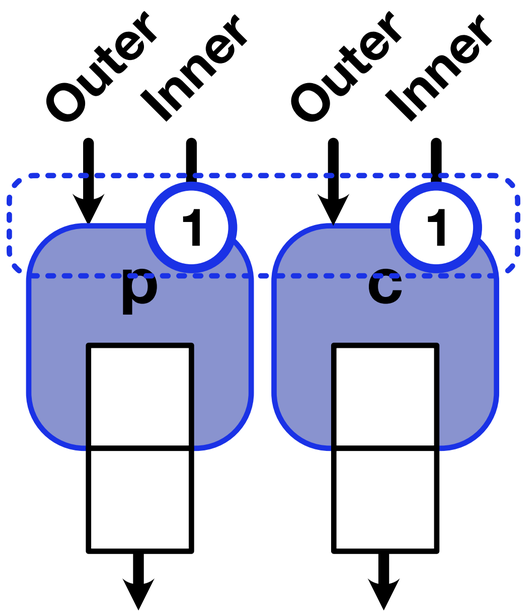
Synchronization accepts token groups

Complete set of outer-loop tokens

or

Complete set of inner-loop tokens



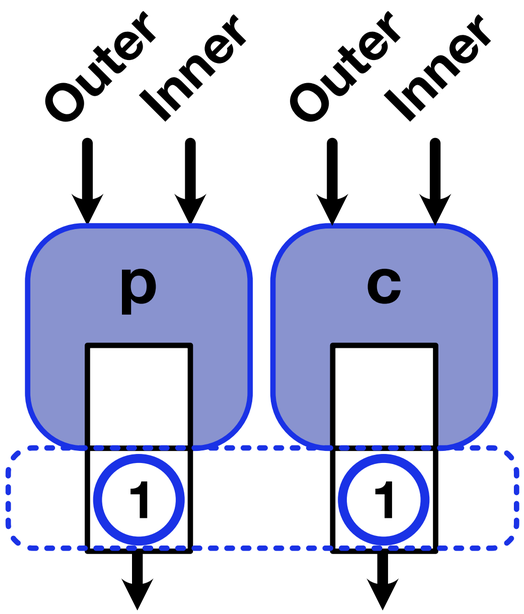


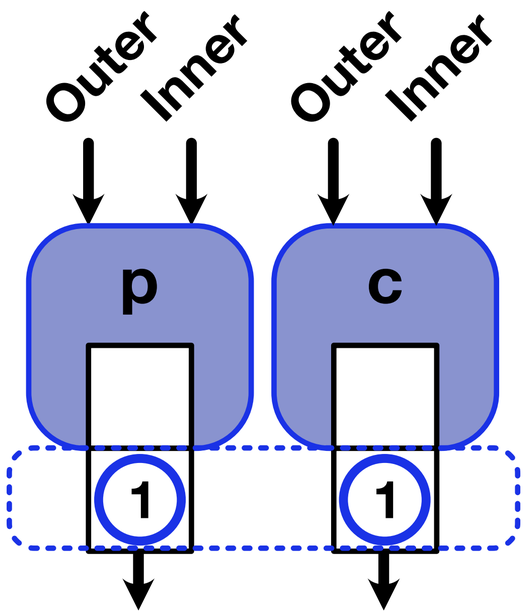
Synchronization accepts token groups

Complete set of outer-loop tokens

or

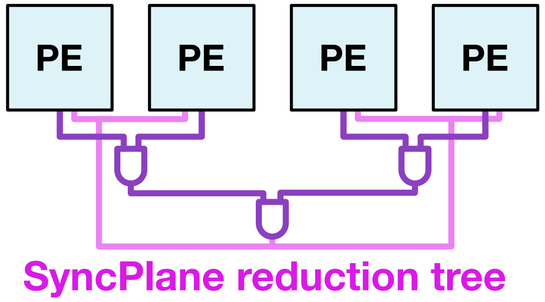
Complete set of inner-loop tokens



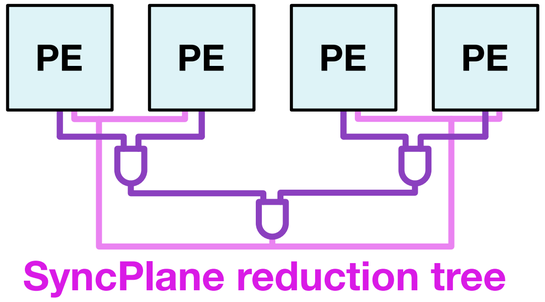


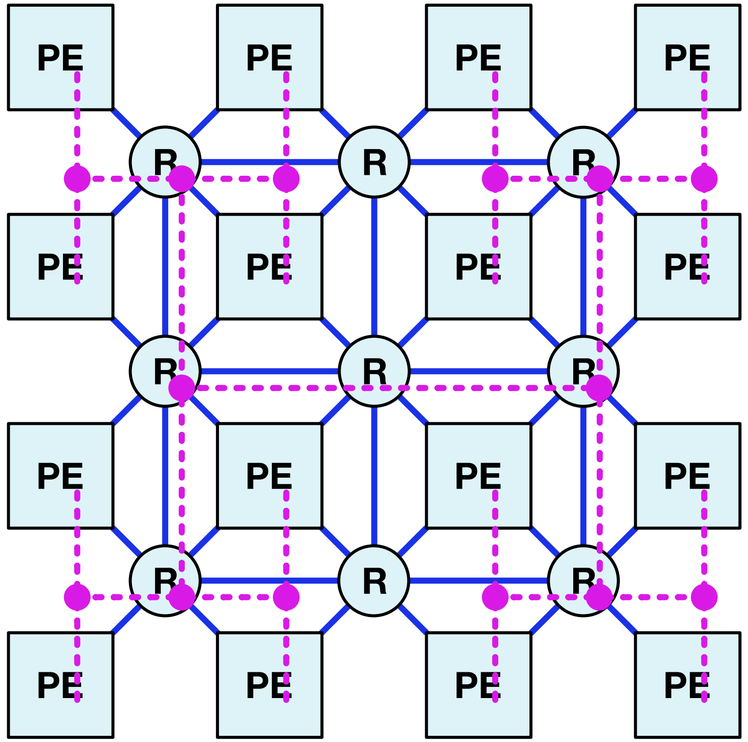
SyncPlane enables ordering

SyncPlane implements synchronization



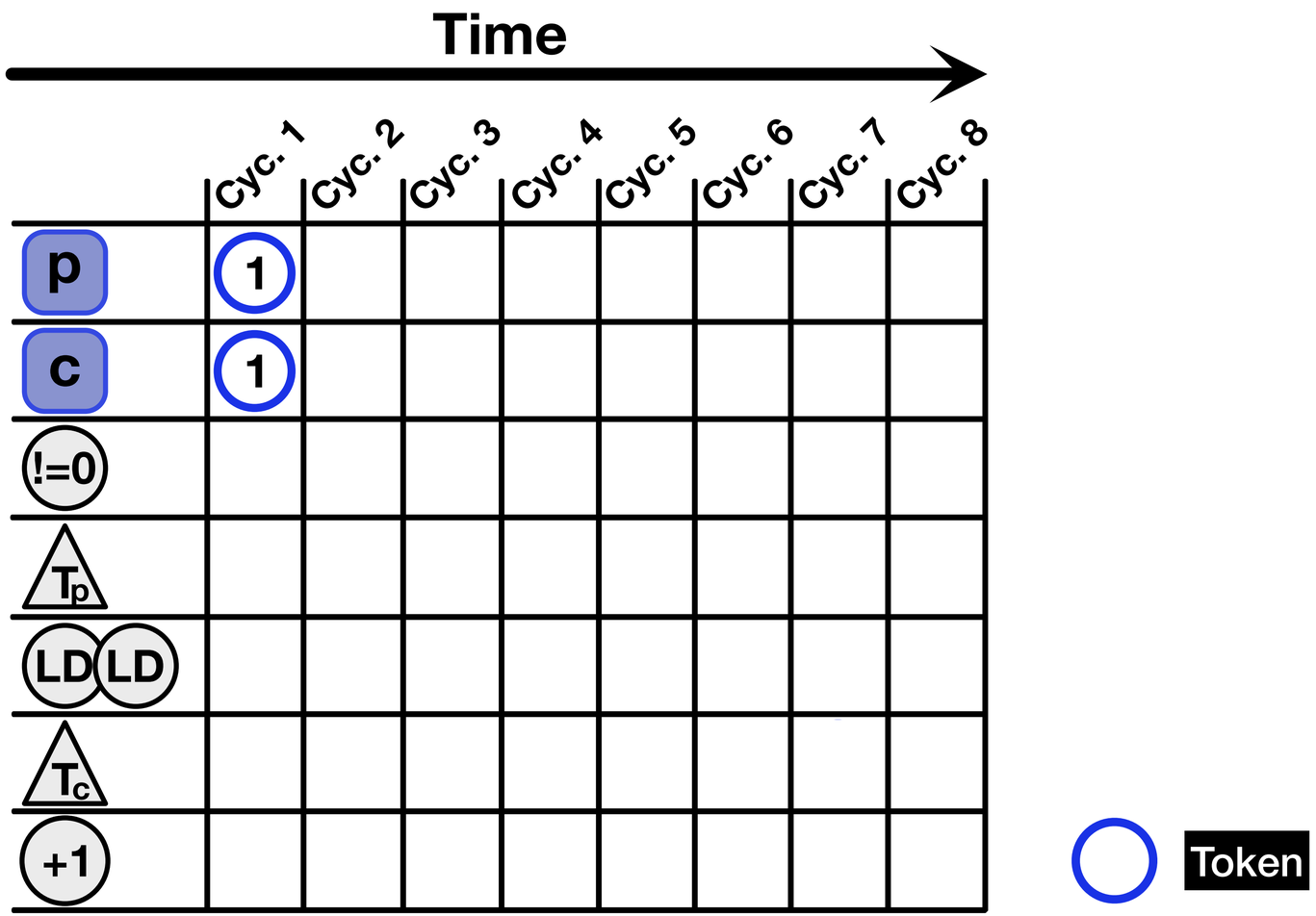
SyncPlane implements synchronization



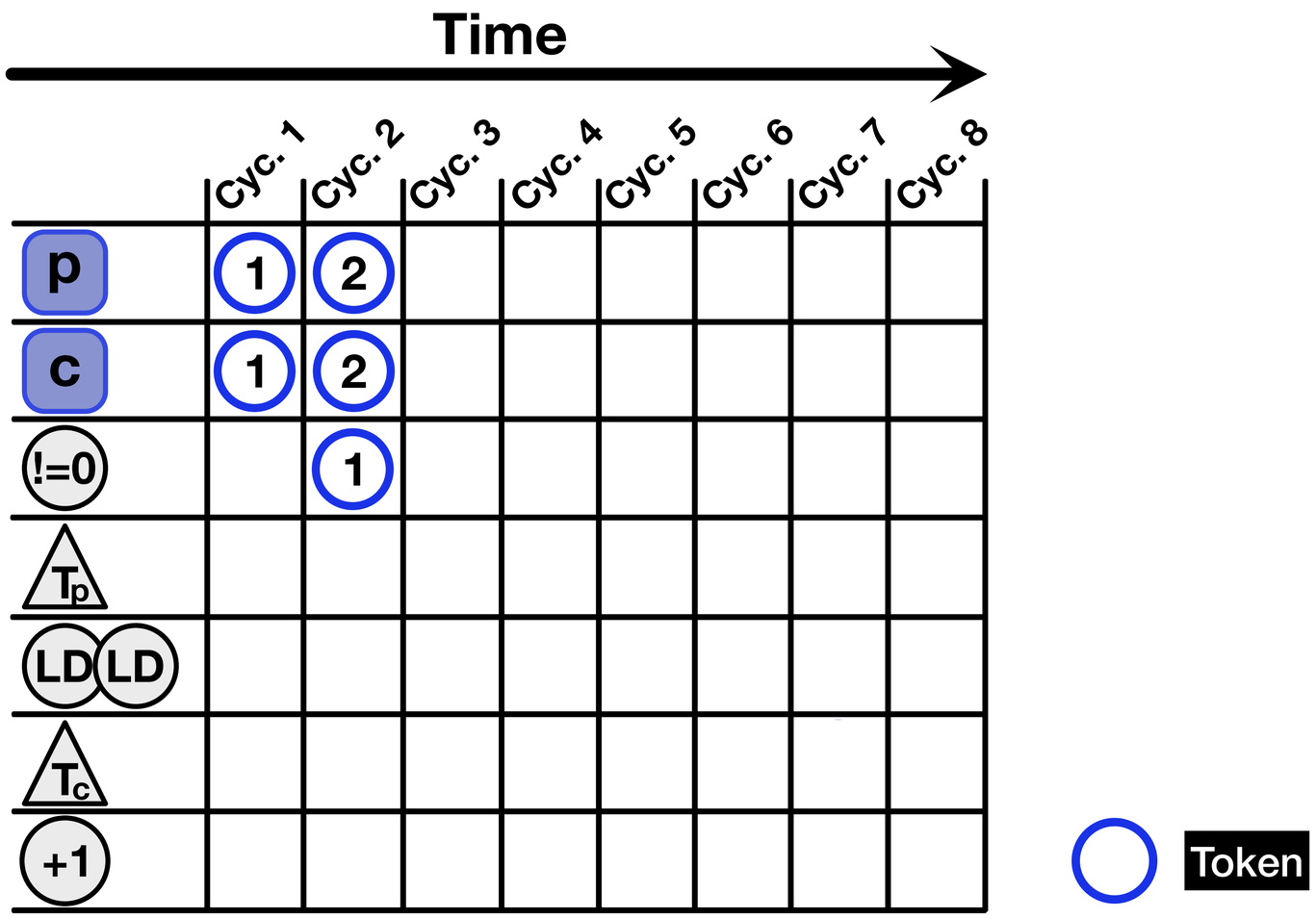


Pipestitch pipelines

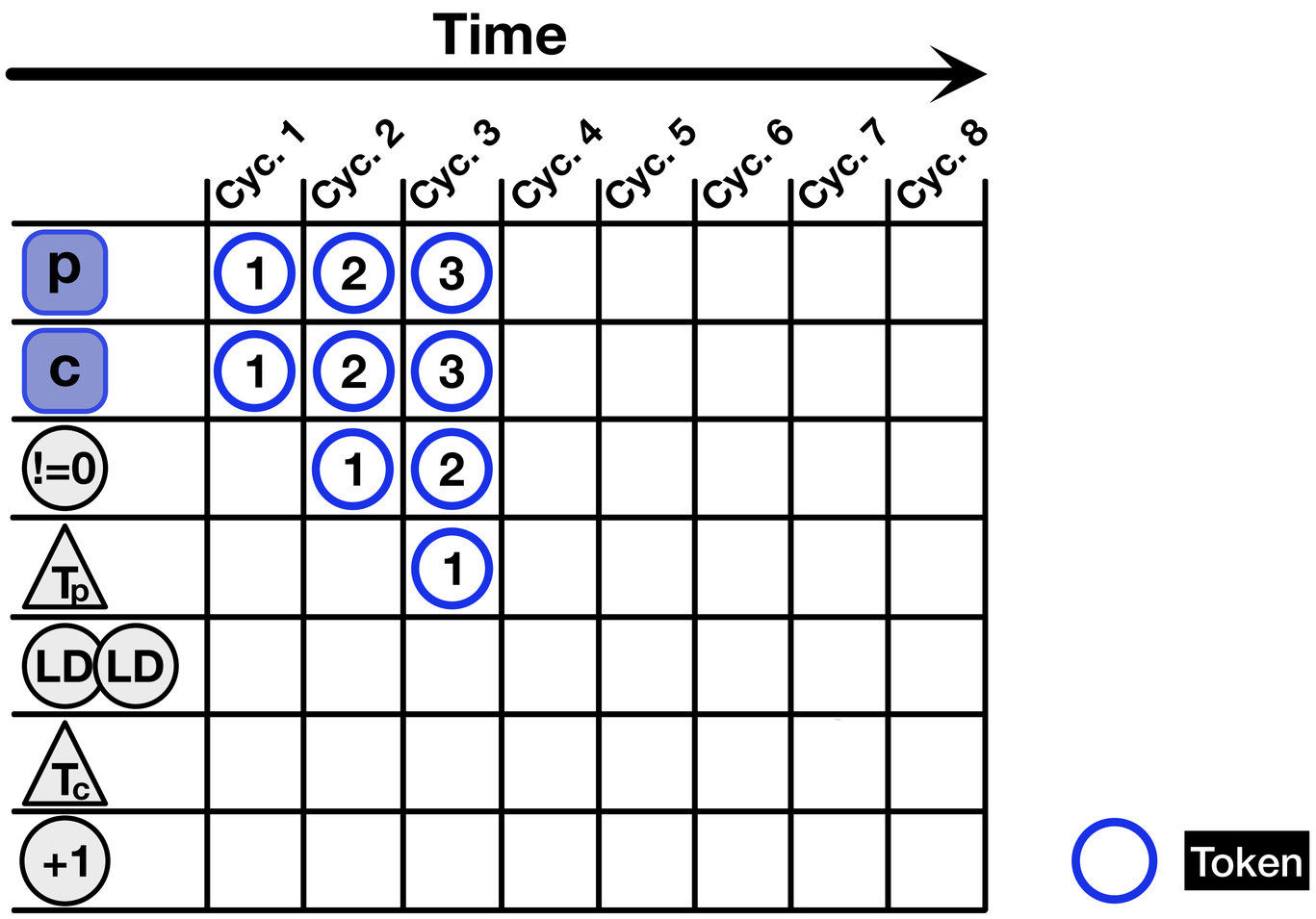
Pipestitch enables perfect pipelining



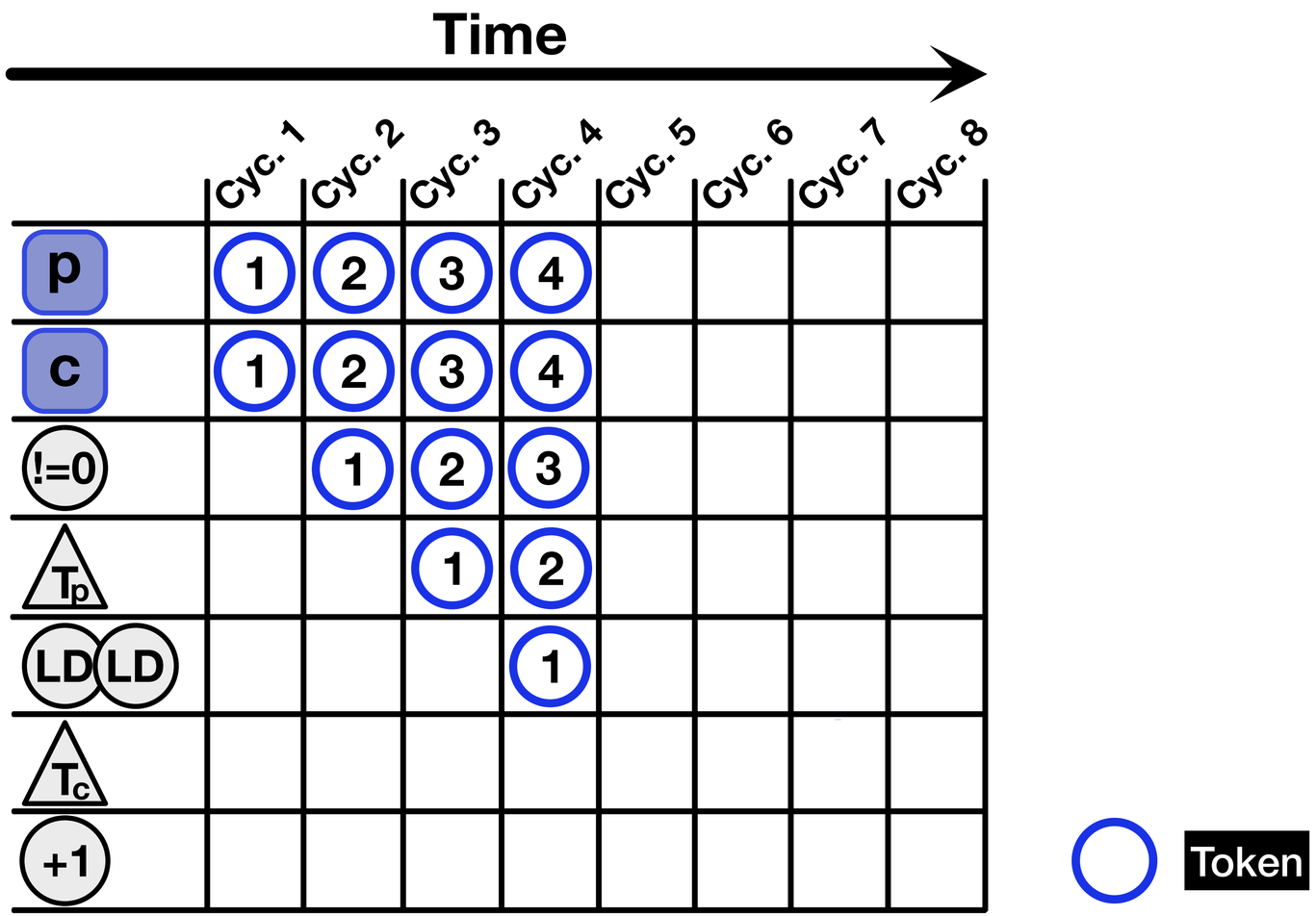
Pipestitch enables perfect pipelining



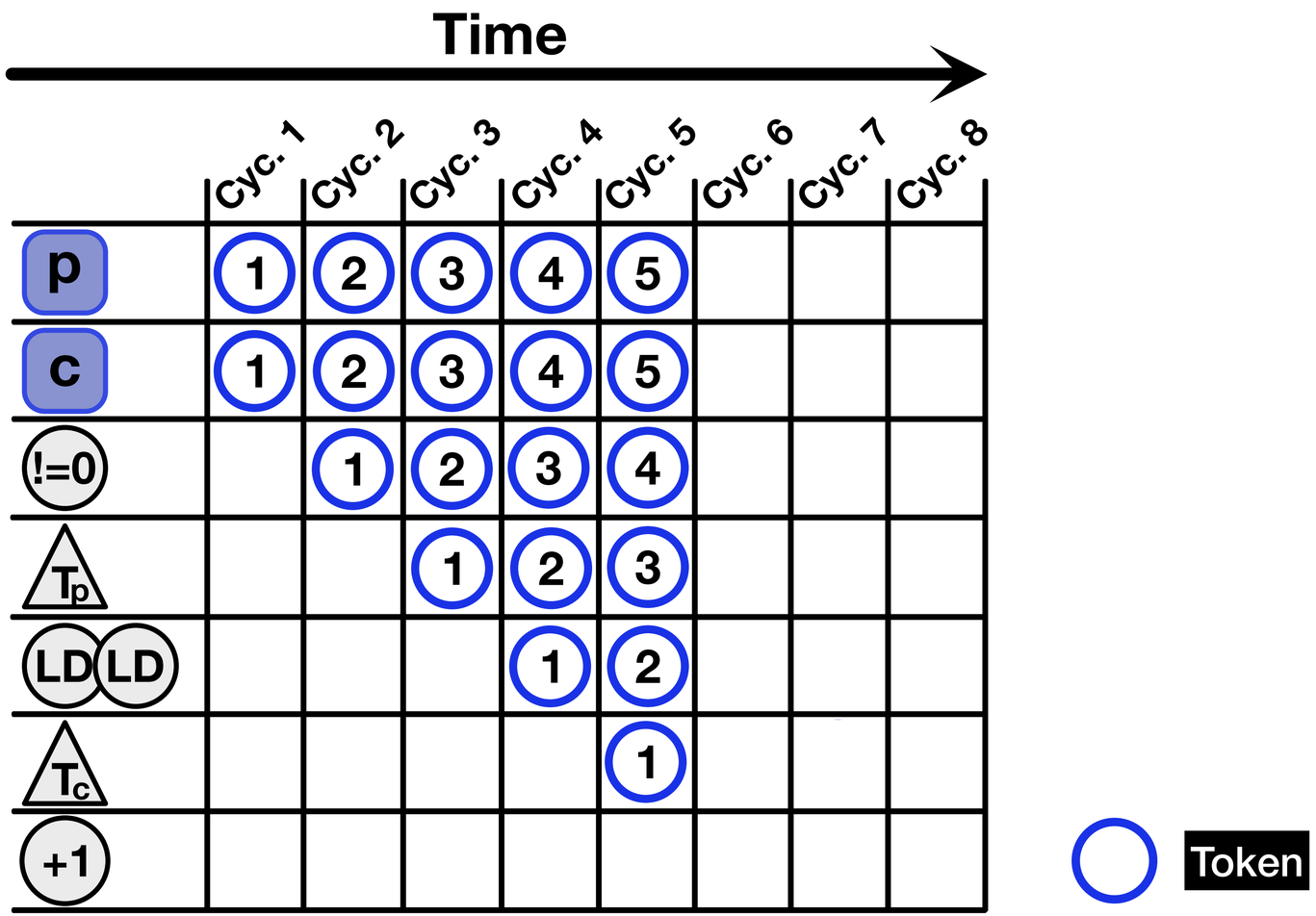
Pipestitch enables perfect pipelining



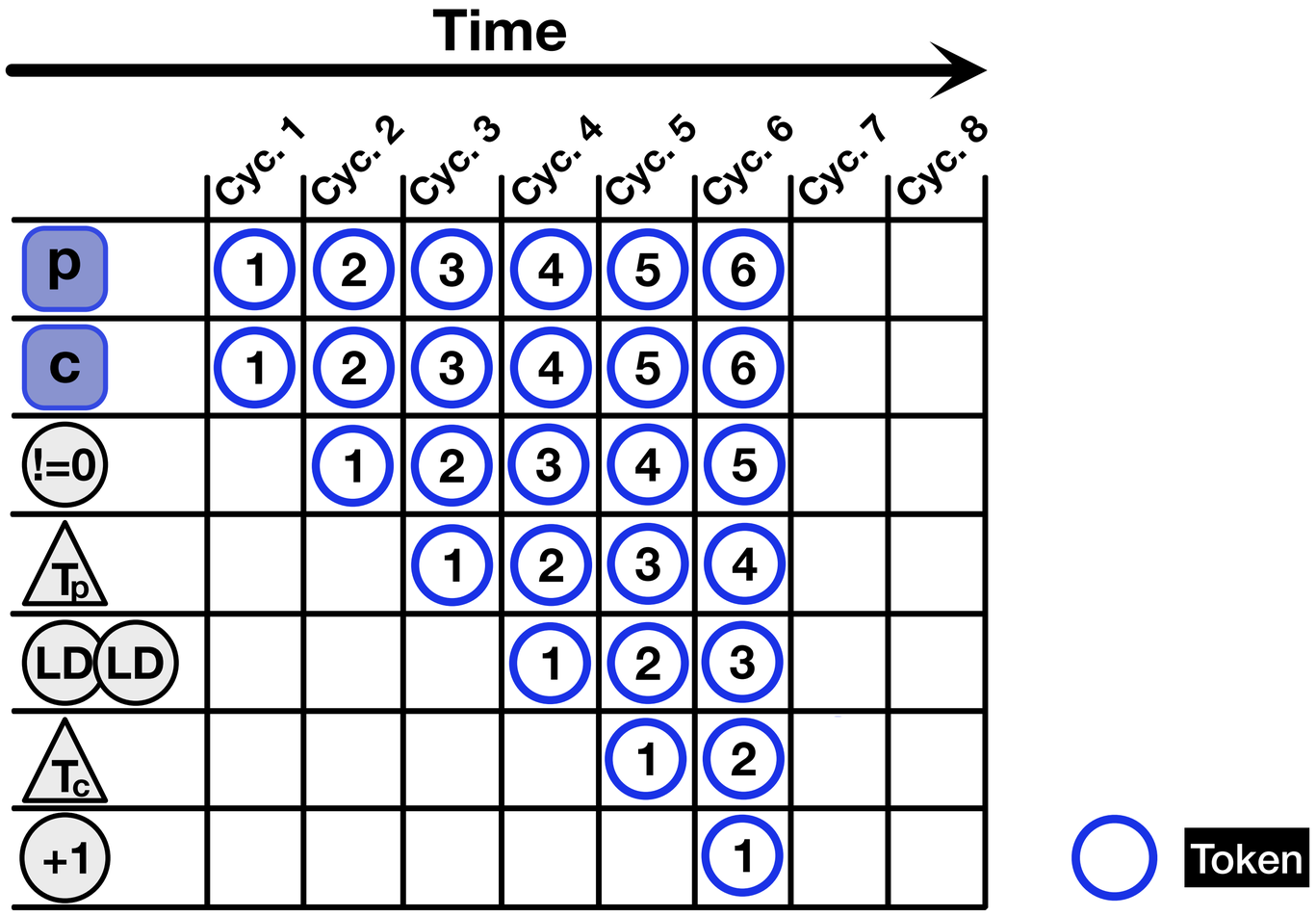
Pipestitch enables perfect pipelining



Pipestitch enables perfect pipelining



Pipestitch enables perfect pipelining



Pipestitch enables perfect pipelining

Time →

	Cyc. 1	Cyc. 2	Cyc. 3	Cyc. 4	Cyc. 5	Cyc. 6	Cyc. 7	Cyc. 8
p	1	2	3	4	5	6	1	
c	1	2	3	4	5	6	1	
!=0		1	2	3	4	5	6	
T_p			1	2	3	4	5	
LD LD				1	2	3	4	
T_c					1	2	3	
+1						1	2	

Token

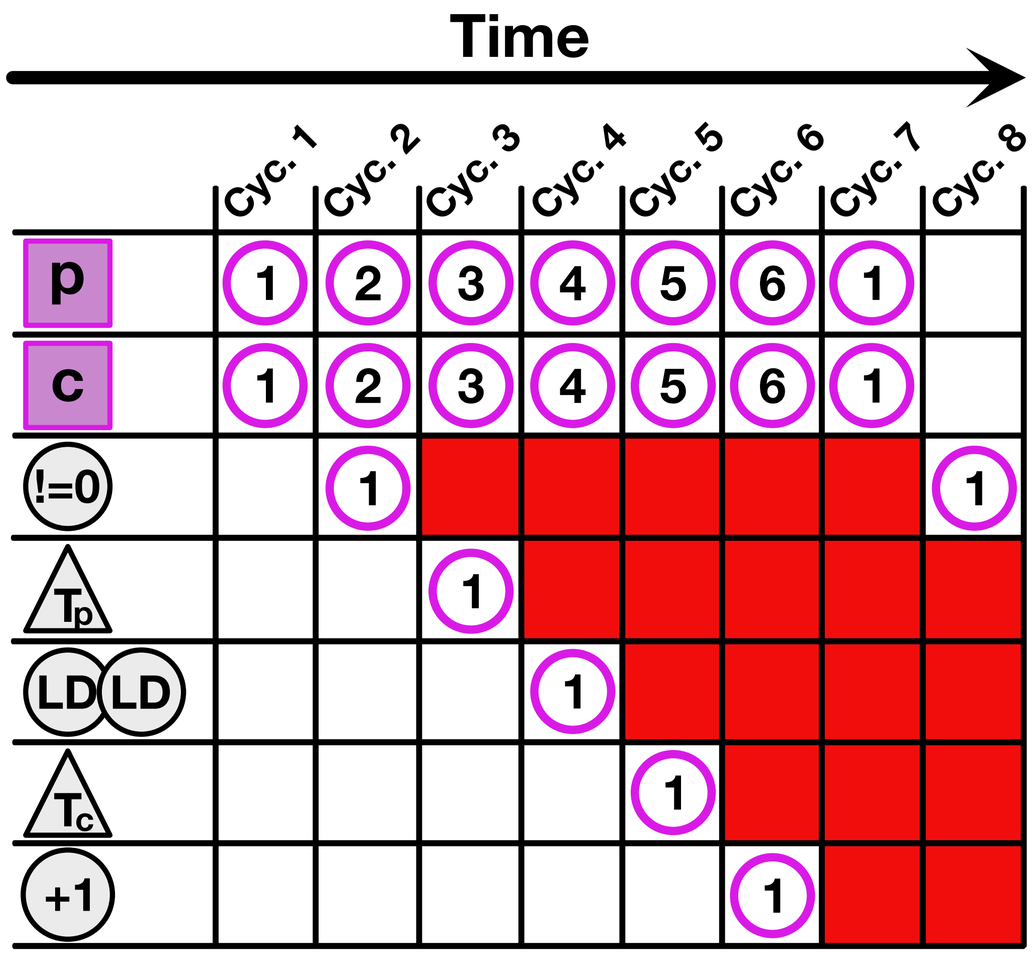
Pipestitch enables perfect pipelining

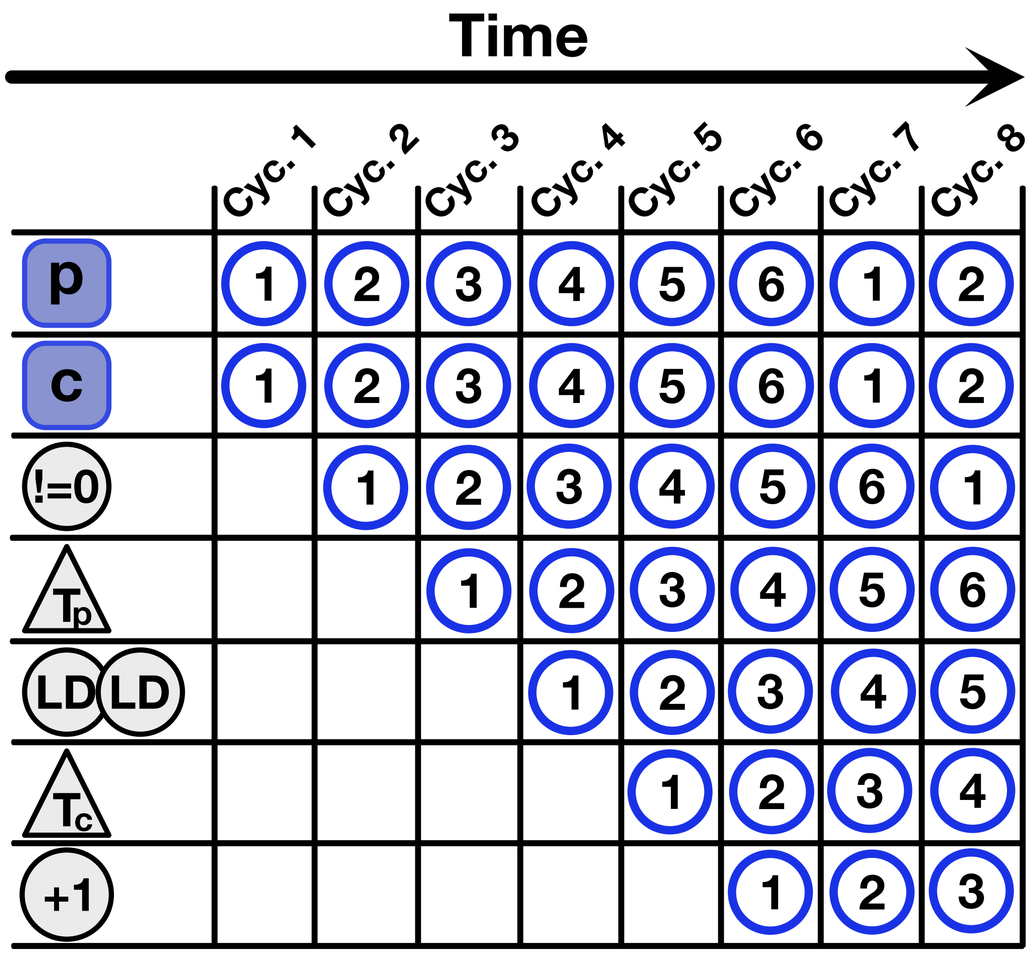
Time →

	Cyc. 1	Cyc. 2	Cyc. 3	Cyc. 4	Cyc. 5	Cyc. 6	Cyc. 7	Cyc. 8
p	1	2	3	4	5	6	1	2
c	1	2	3	4	5	6	1	2
!=0		1	2	3	4	5	6	1
T_p			1	2	3	4	5	6
LD LD				1	2	3	4	5
T_c					1	2	3	4
+1						1	2	3

Token

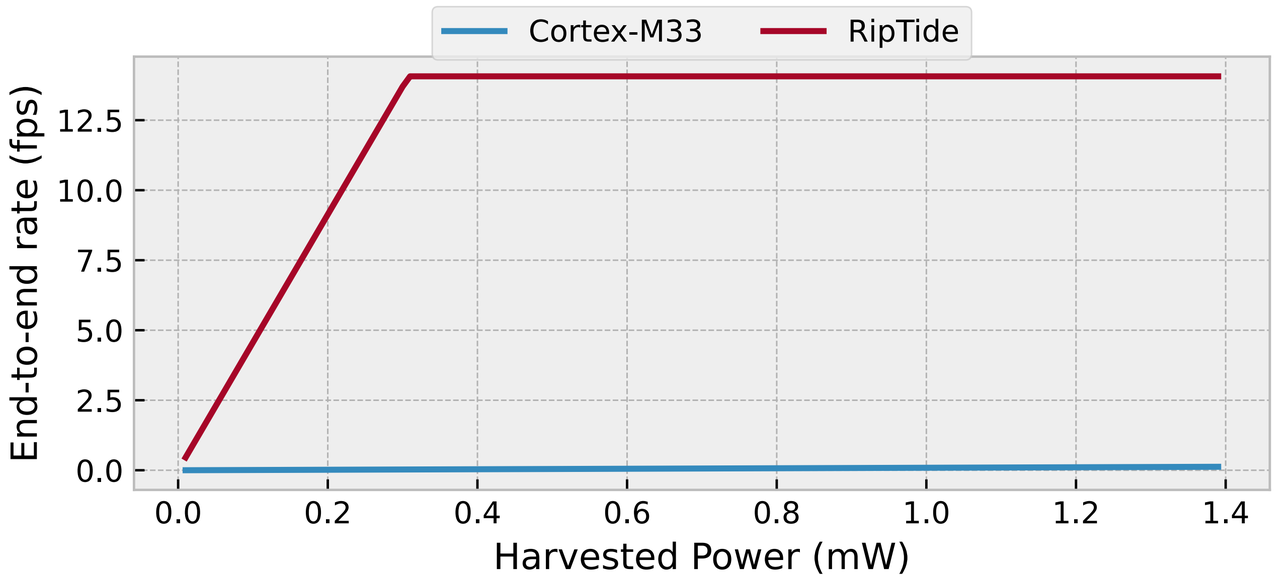
Pipestitch fixes utilization problem





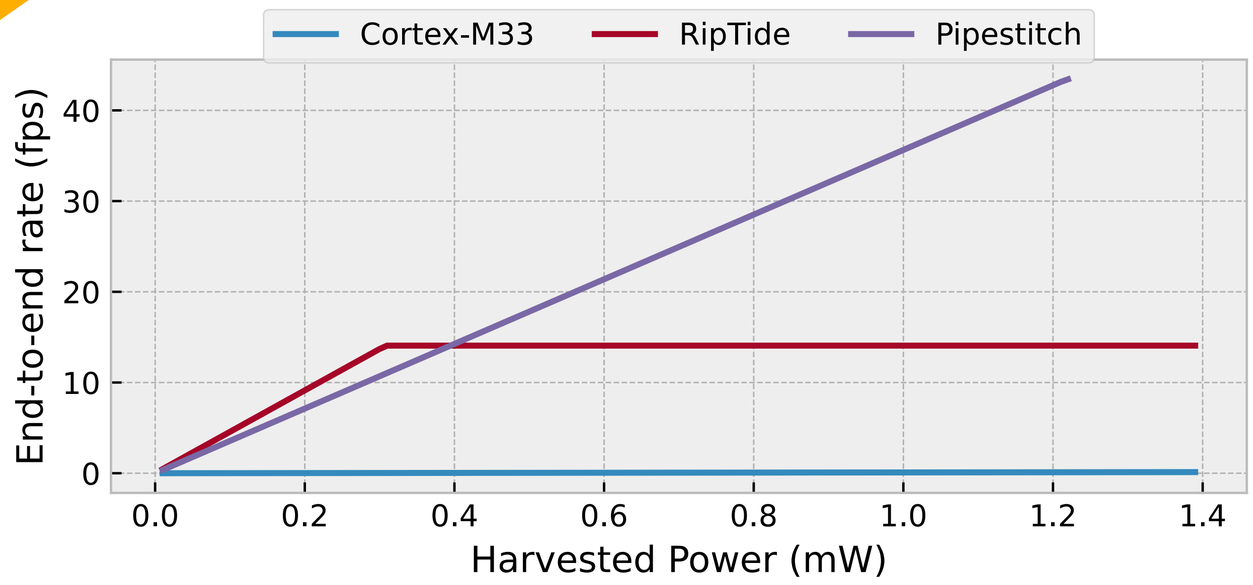
Pipestitch in the DNN study

RipTide performance plateaus early

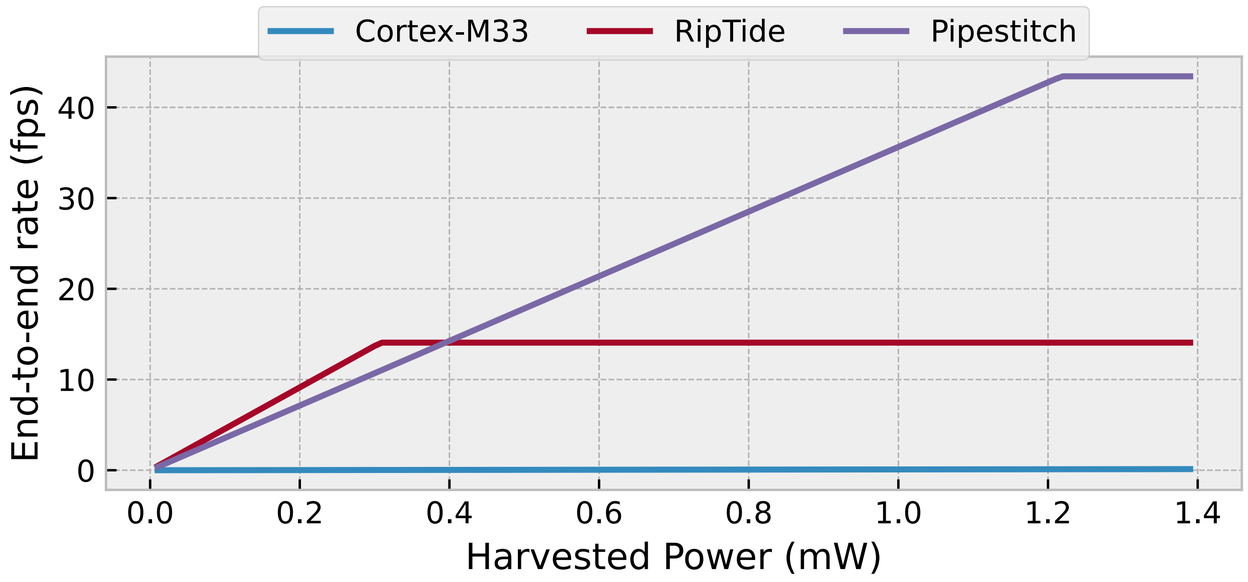


Pipestitch supports tighter deadlines

Axis change



Pipestitch utilizes greater harvested power



Outline

Spatial dataflow review

Spatial dataflow is efficient + irregular apps are important

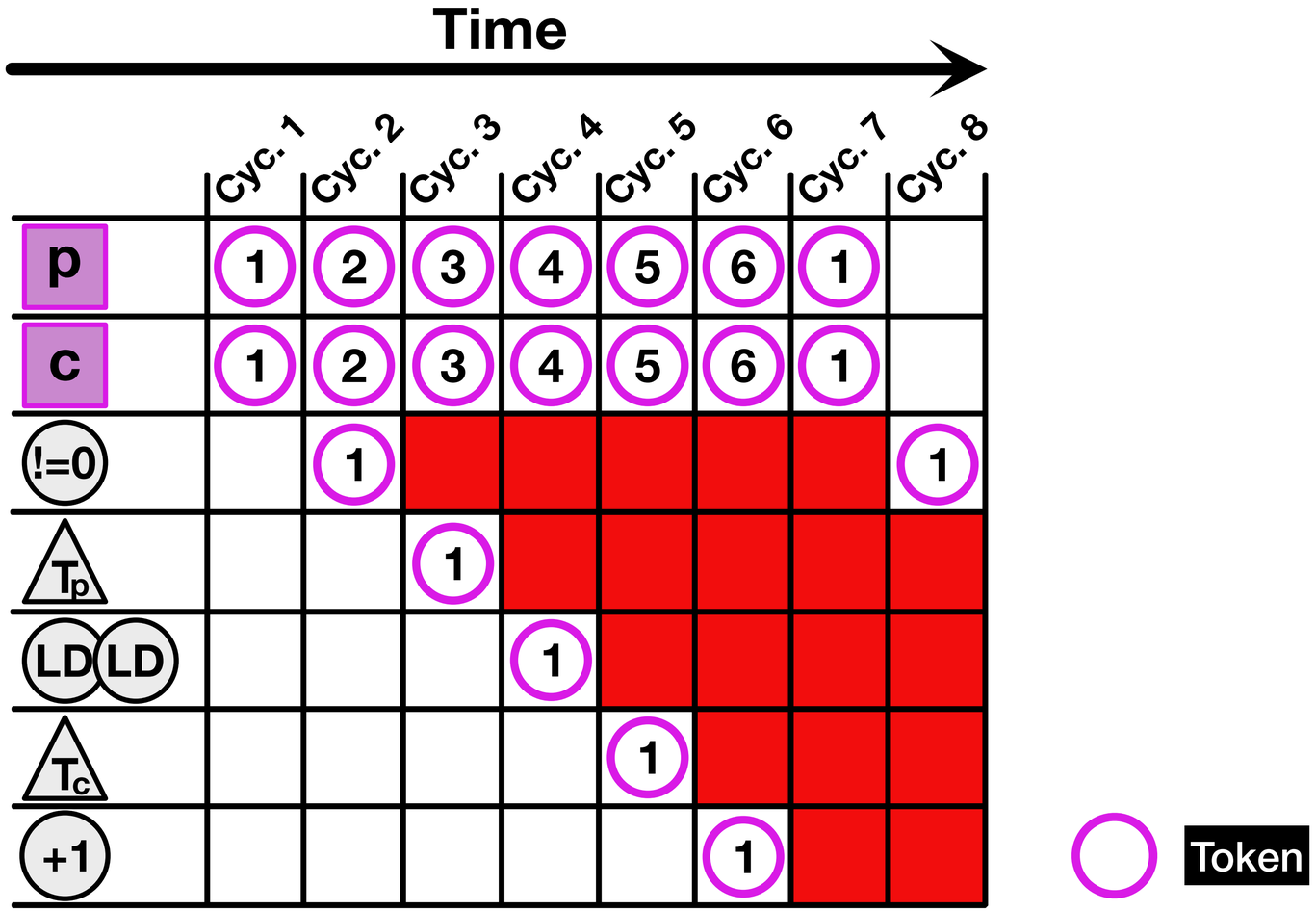
Spatial dataflow not efficient with irregular apps

Make your irregular app look regular

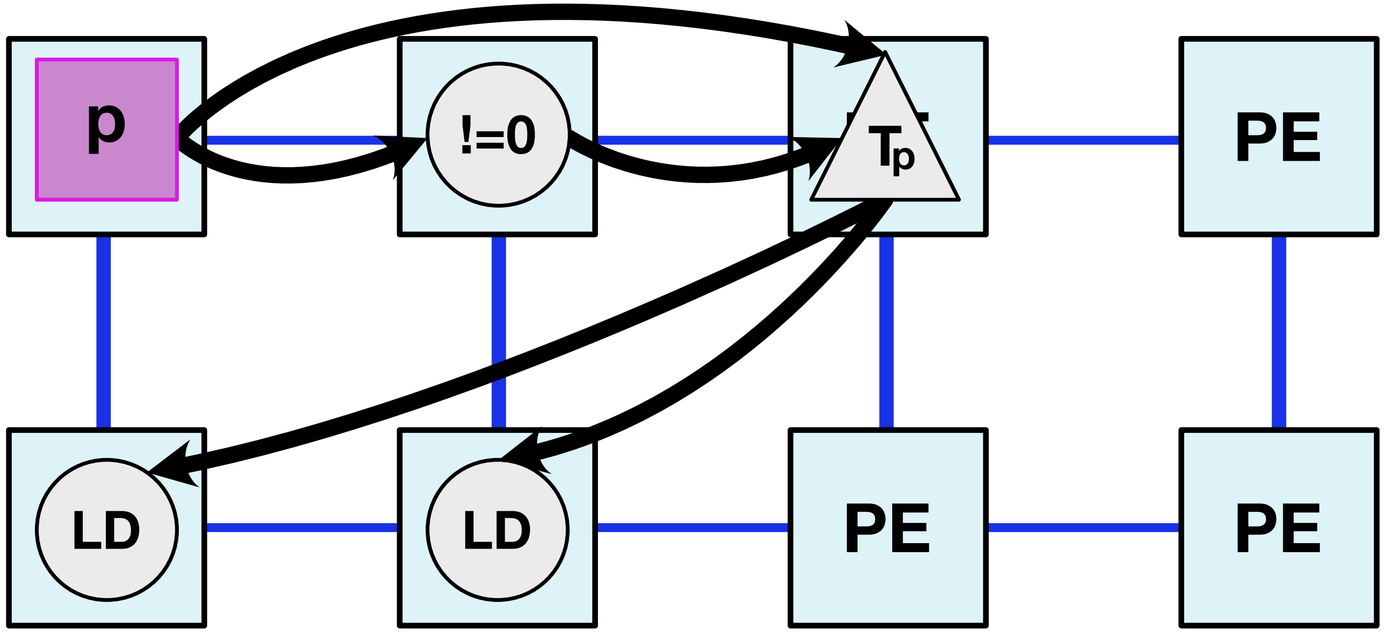
Exploit app properties to increase clock frequency

Eliminate clock and fully fuse

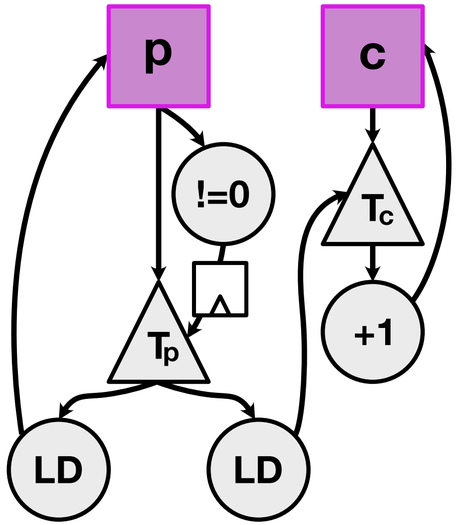
Some programs can't be pipelined



Long paths in NoC limit frequency



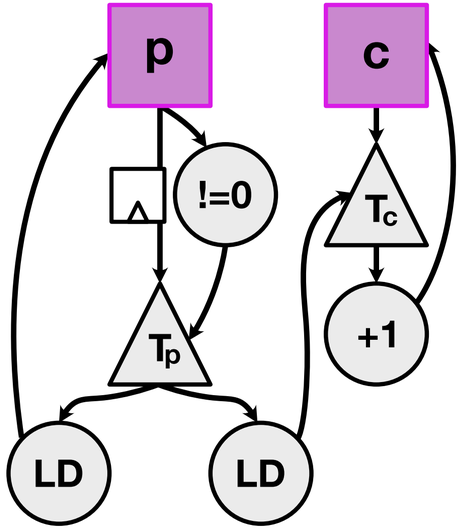
Some buffers add cycles

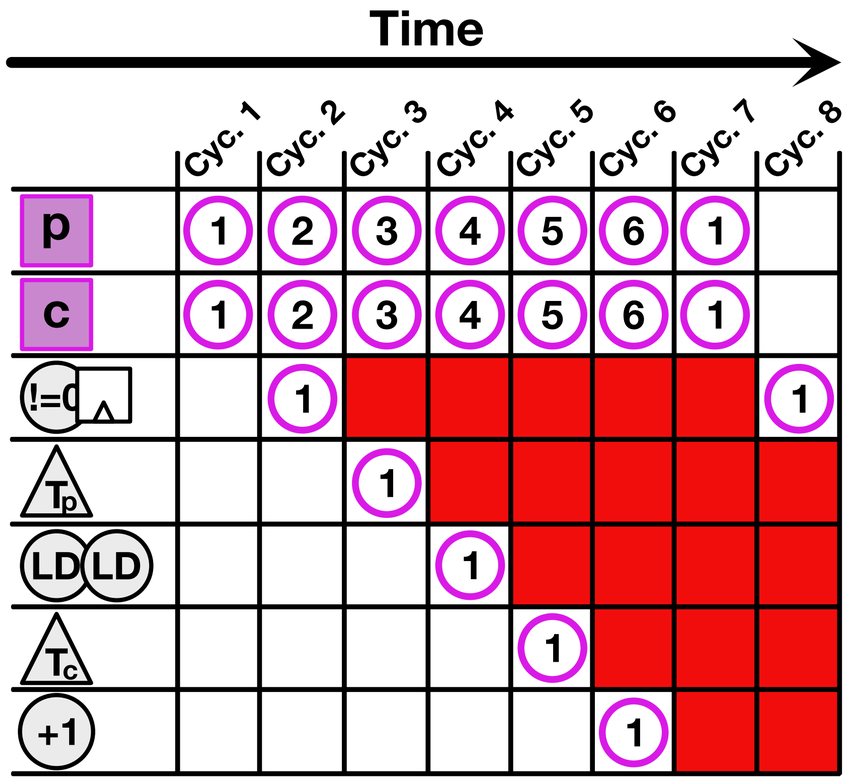


Time

	Cyc. 1	Cyc. 2	Cyc. 3	Cyc. 4	Cyc. 5	Cyc. 6	Cyc. 7	Cyc. 8
p	1	2	3	4	5	6	7	1
c	1	2	3	4	5	6	7	1
!=0		1						
			1					
Tp				1				
LD LD					1			
Tc						1		
+1							1	

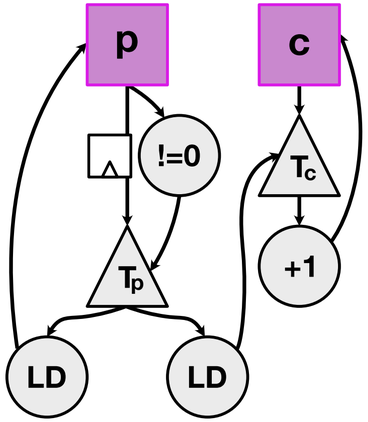
Some buffers do not





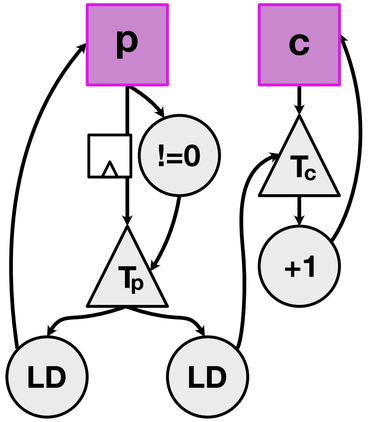
Flopjack overview

Program analysis of bufferability

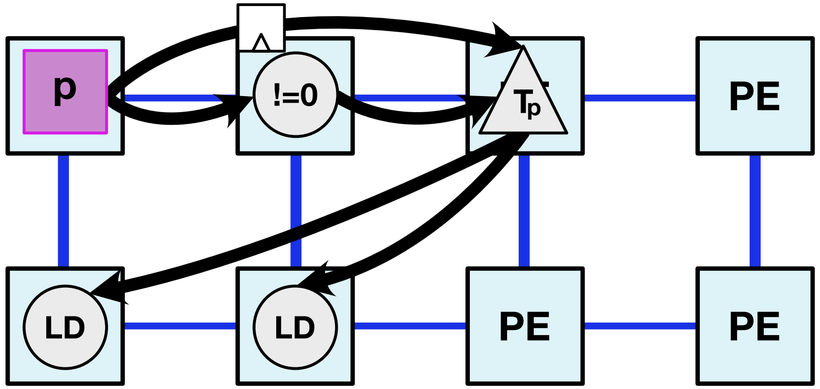


Flopjack overview

Program analysis of bufferability

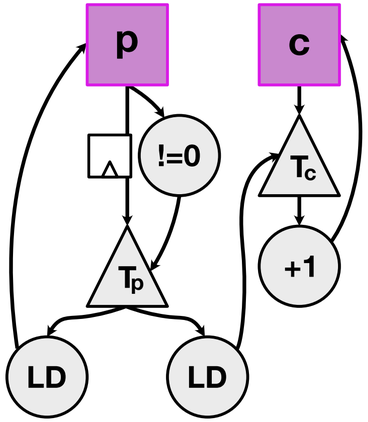


Buffer-aware compiler

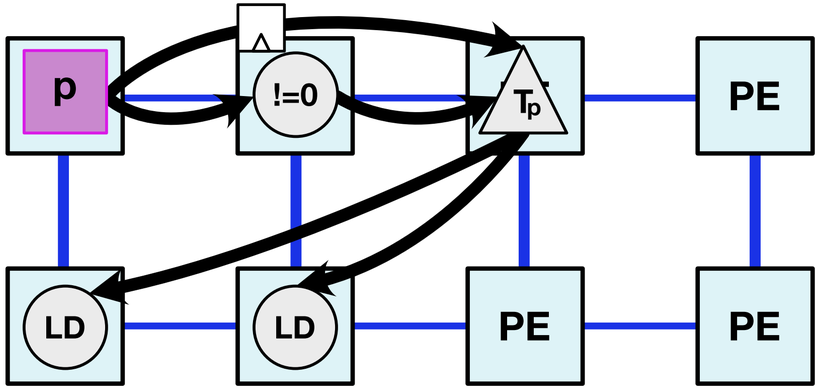


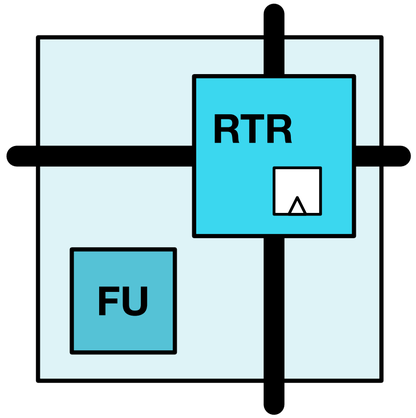
Flopjack overview

Program analysis of bufferability



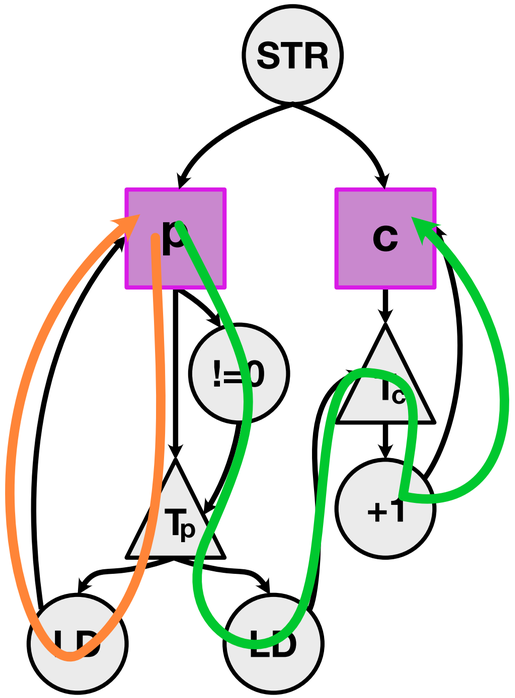
Buffer-aware compiler



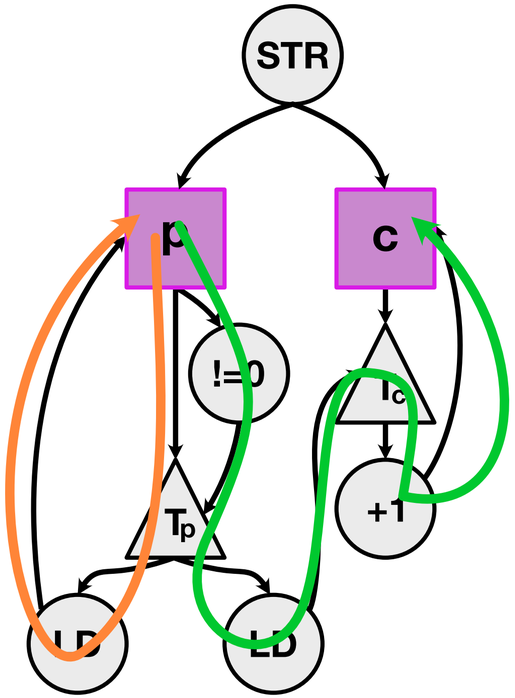


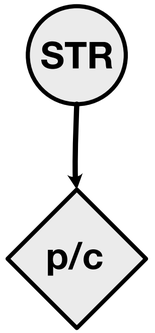
Flexible network buffering

Program structure provides budgets



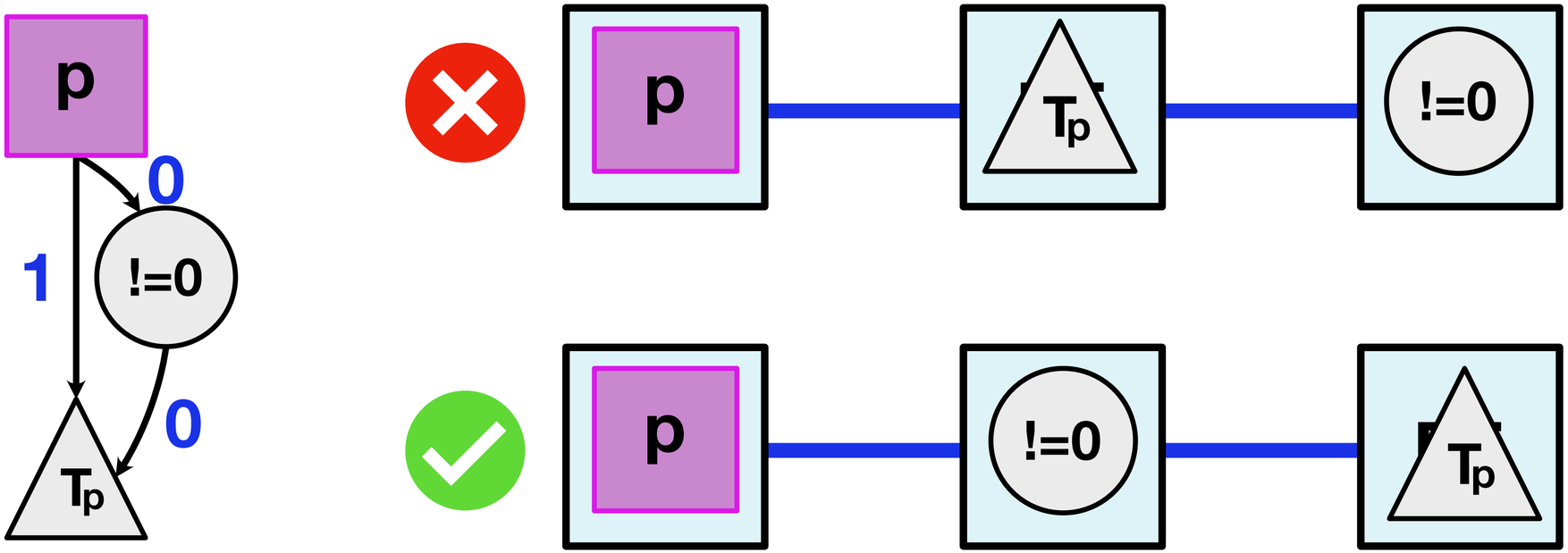
Program structure provides budgets



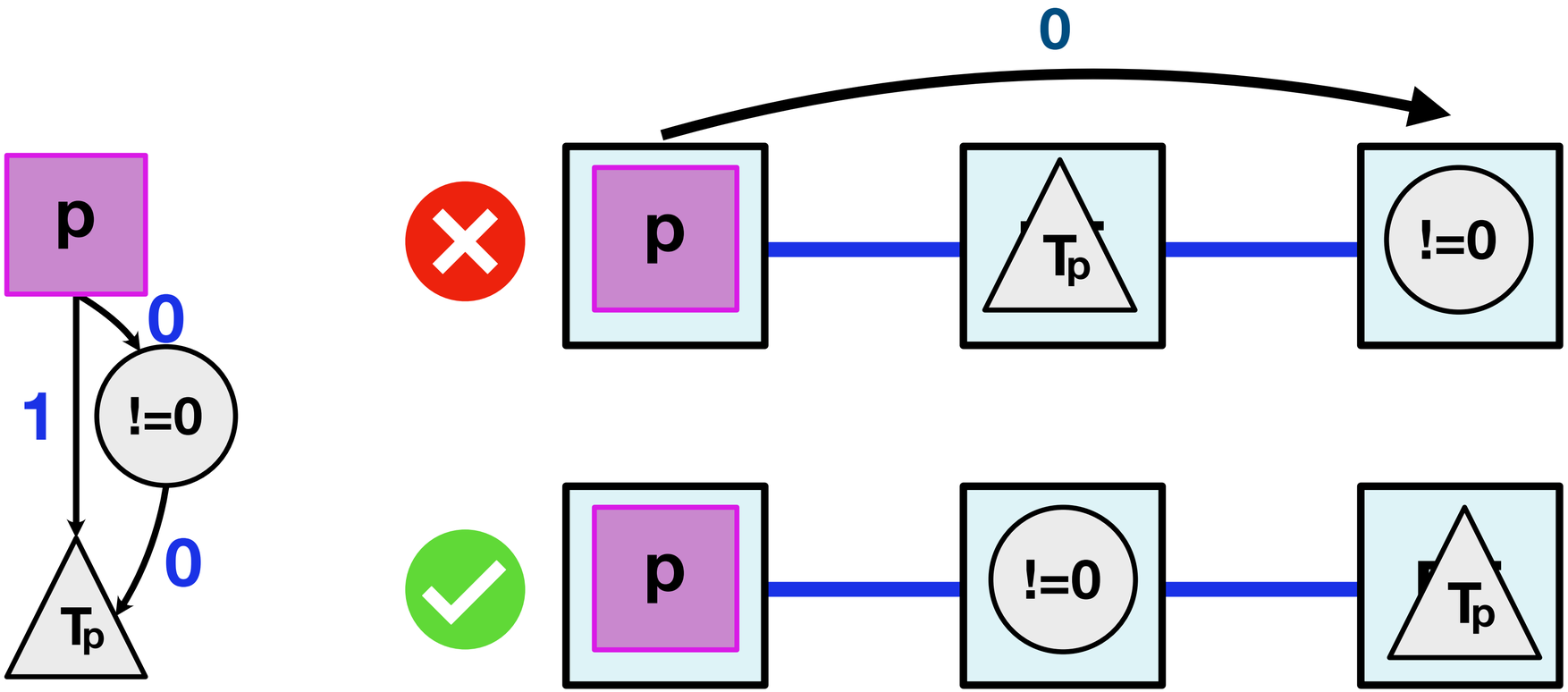


delay = IIp/c * trip_count

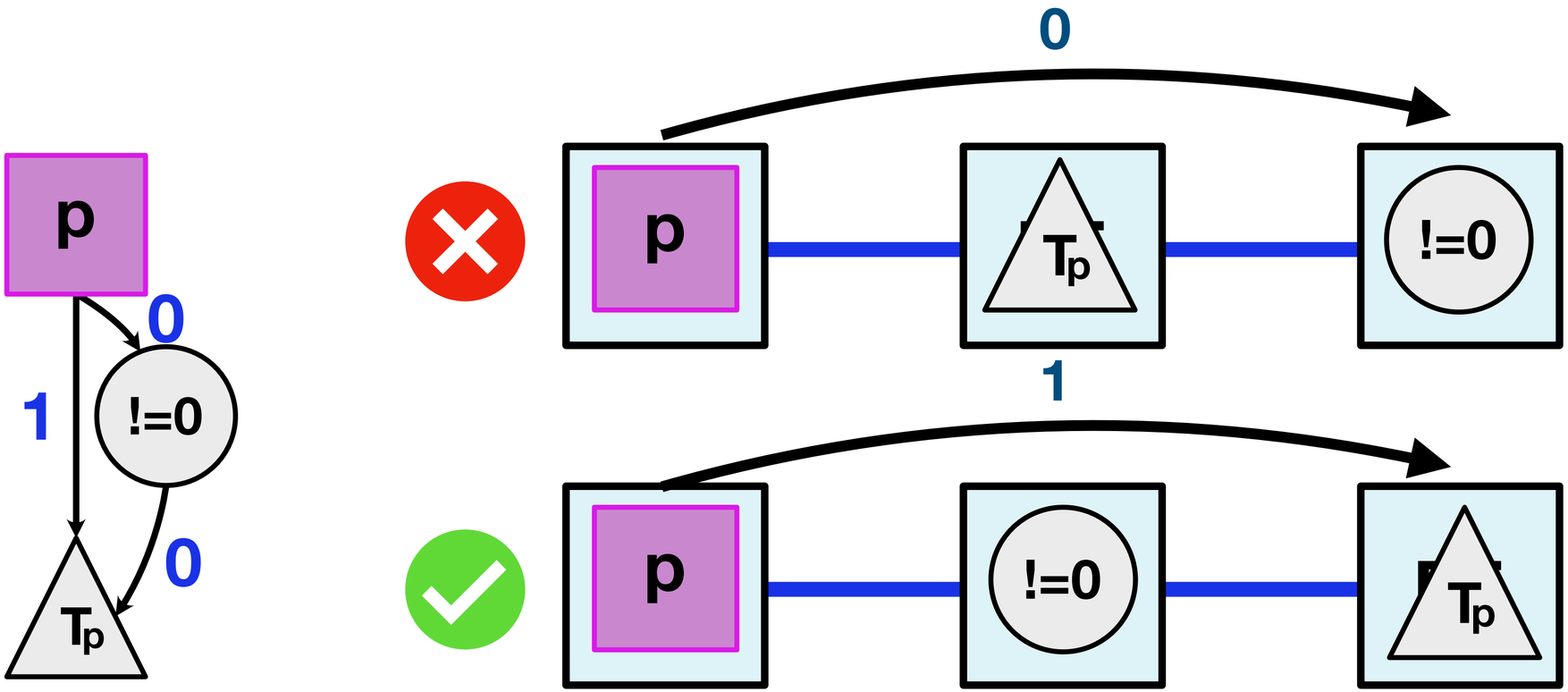
Budgets inform mapping



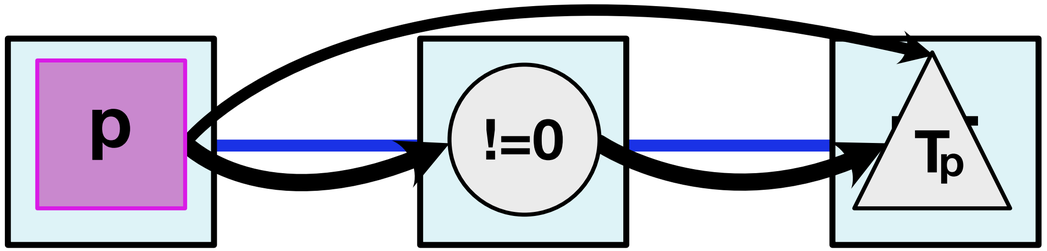
Budgets inform mapping



Budgets inform mapping



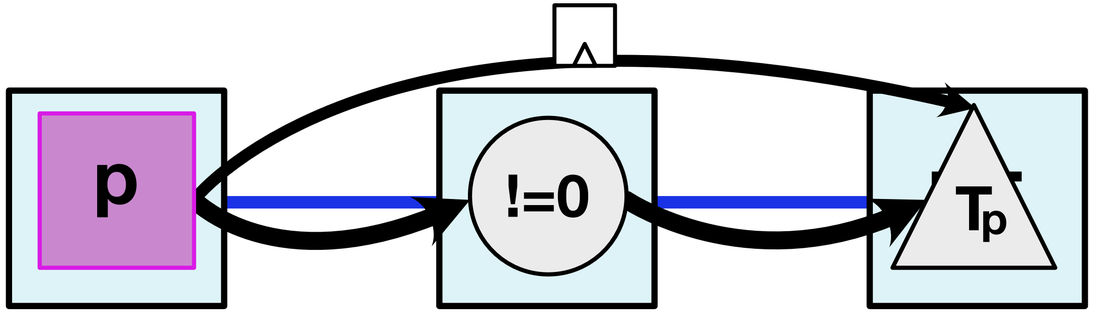
Weight edges by budget



$$w_e = \frac{1}{b_e + 1}$$

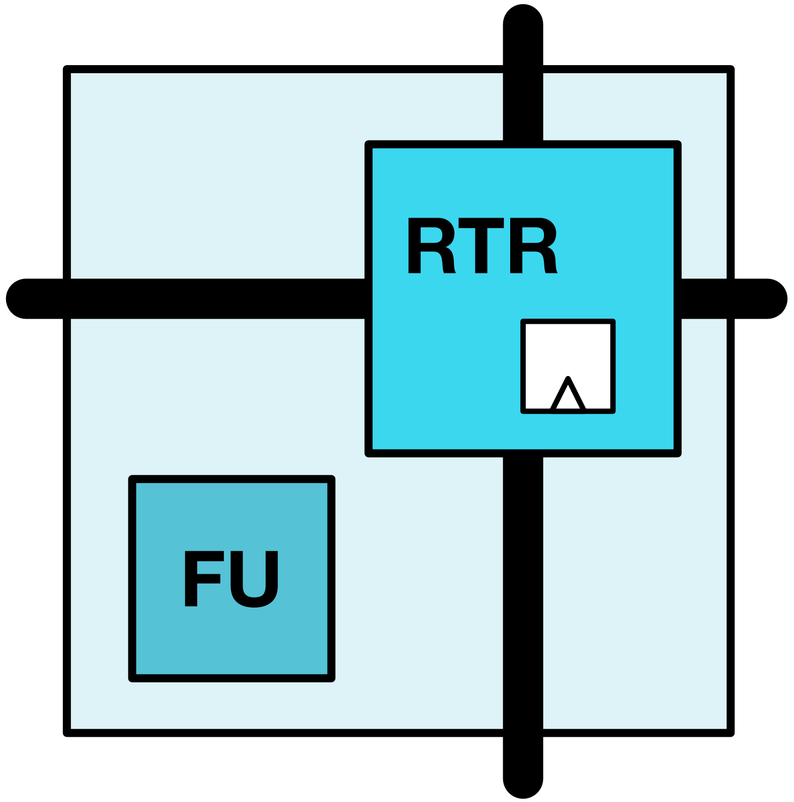
$$C = \sum_e D_e + \max(D_e w_e) L$$

Route based on bufferability



$$D_{e,eff} = \left\lceil \frac{D_e}{b_e + 1} \right\rceil$$

Flex buffers included in routers



Outline

Spatial dataflow review

Spatial dataflow is efficient + irregular apps are important

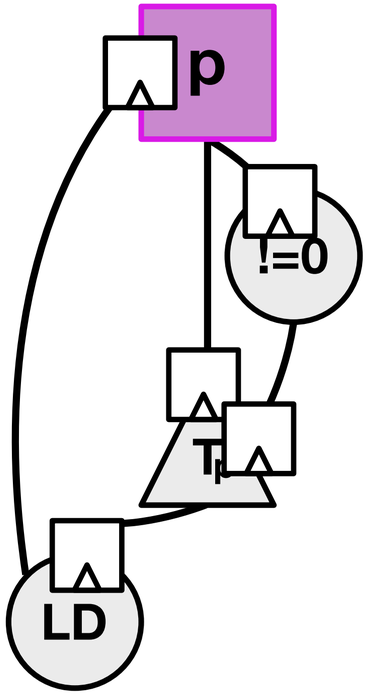
Spatial dataflow not efficient with irregular apps

Make your irregular app look regular

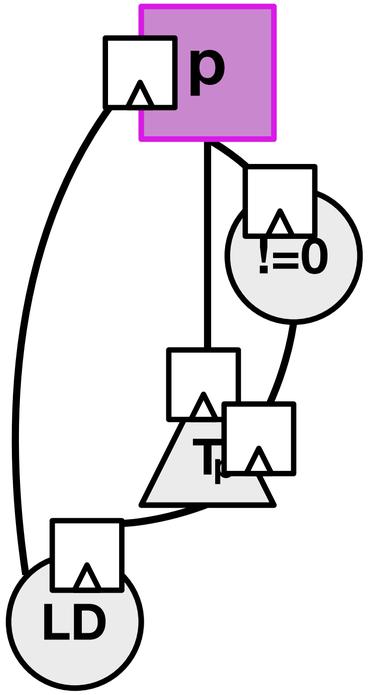
Exploit app properties to increase clock frequency

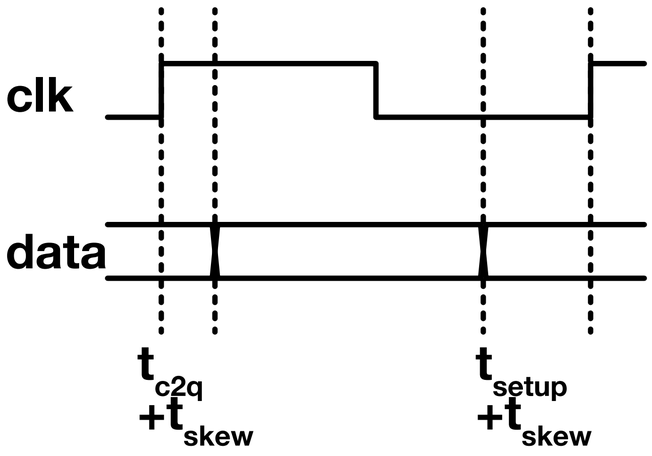
Eliminate clock and fully fuse

Each PE flops its inputs

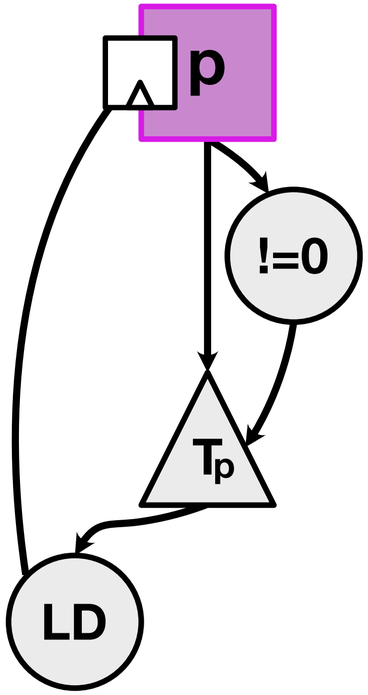


Sequencing overhead reduces performance

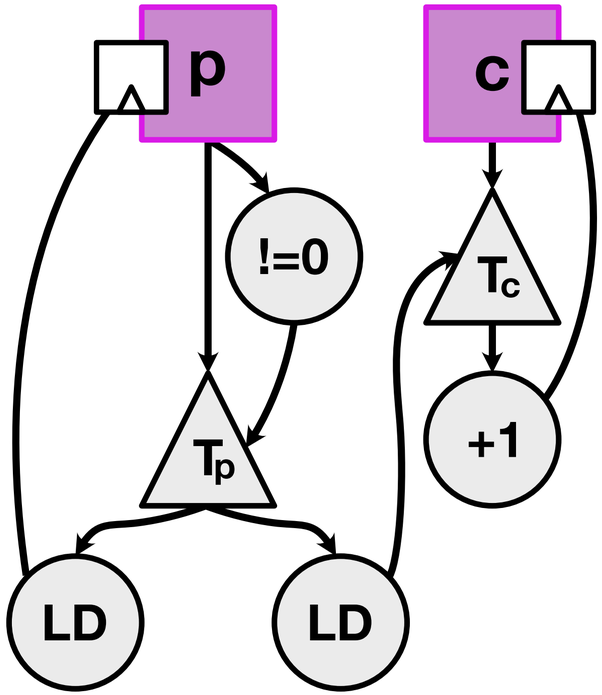




Fuse loops → minimize sequencing overhead

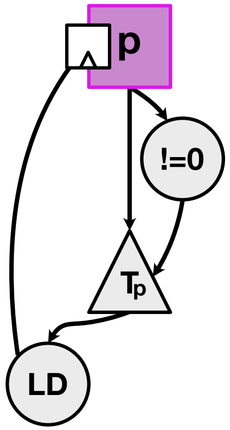


Different cycles at different frequencies



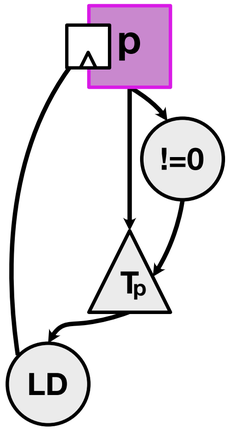
Asynine overview

Fully fuse cycles

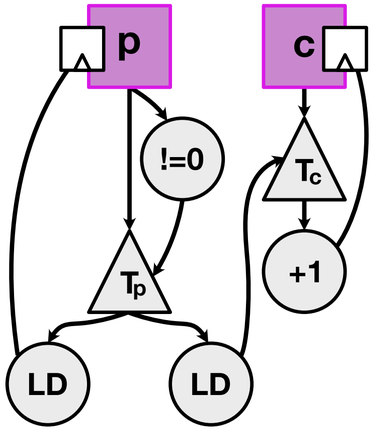


Asynine overview

Fully fuse cycles

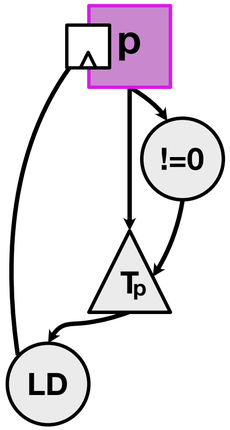


Cycles communicate asynchronously

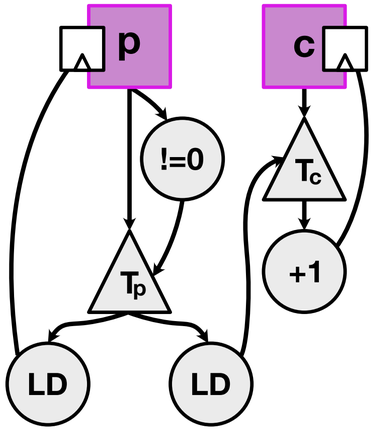


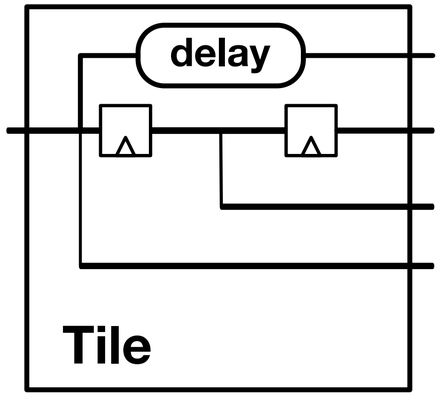
Asynine overview

Fully fuse cycles



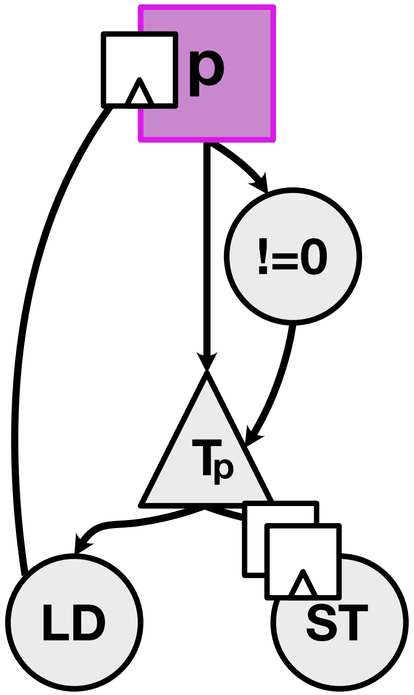
Cycles communicate asynchronously



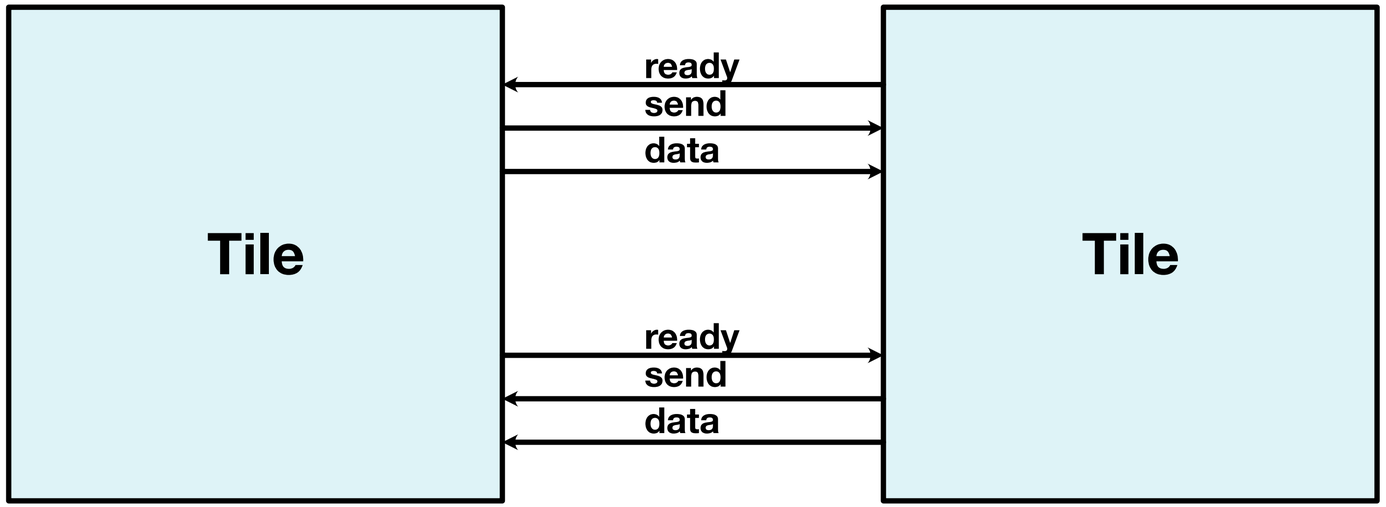


Digital design flow for async CGRAs

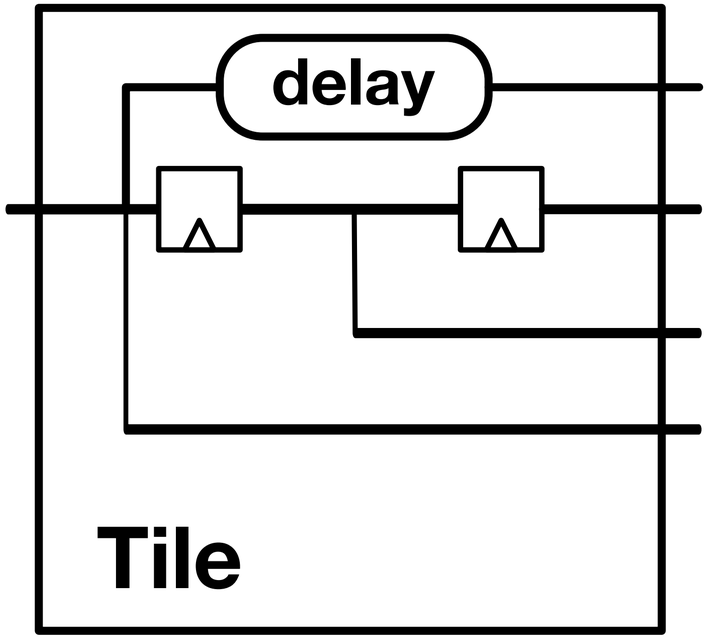
Fully fuse cycles; pipeline other paths



Leverage existing handshaking protocol



Individually constrain tiles and add matched delays



Conclusion

Spatial dataflow review

Spatial dataflow is efficient + irregular apps are important

Spatial dataflow not efficient with irregular apps

Make your irregular app look regular

Exploit app properties to increase clock frequency

Eliminate clock and fully fuse