Out-Of-Order Execution of Synchronous Data-Flow Networks

Daniel Baudisch, Jens Brandt, Klaus Schneider

Embedded Systems Group
Department of Computer Science
University of Kaiserslautern, Germany

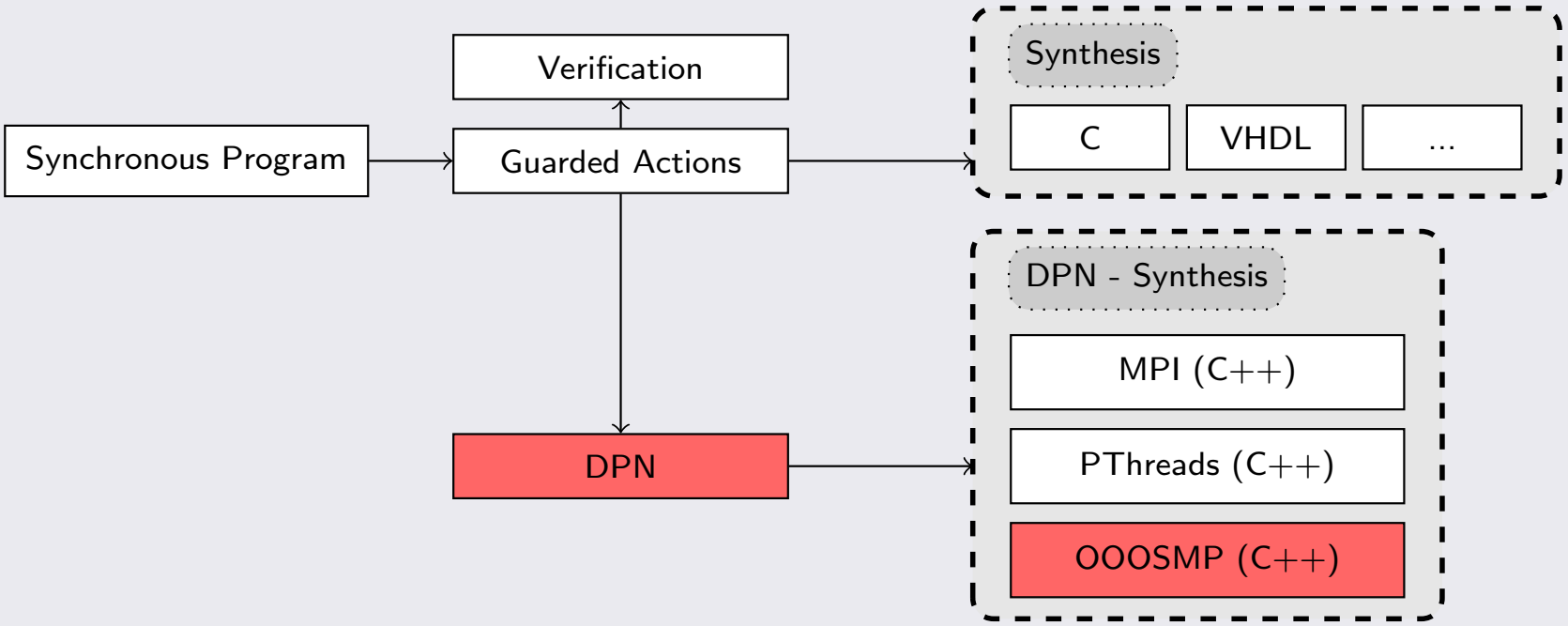
Last Update: May 22, 2013

Outline

Outline

Context

Synthesis Flow

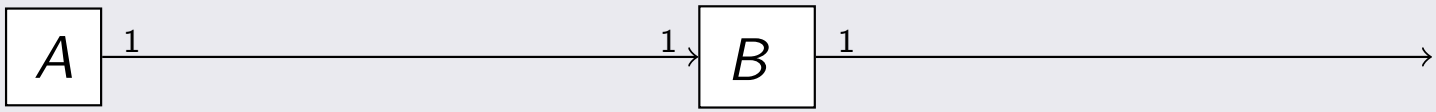


- here: from DPN (given as HSDF) to OOO-Execution in SMP (OOOSMP)

Goal

- Dynamic scheduling (automatic balancing at runtime)
  $\Rightarrow$ Task based execution (OpenMP, Intel TBB, StarSS, ...)
  - decrease overhead of thread-switches (#num nodes $>$ # cores)
  - better balancing (#num nodes $>$ # cores)
  - task $\approx$ execution of node for an iteration (like in SmpSS)

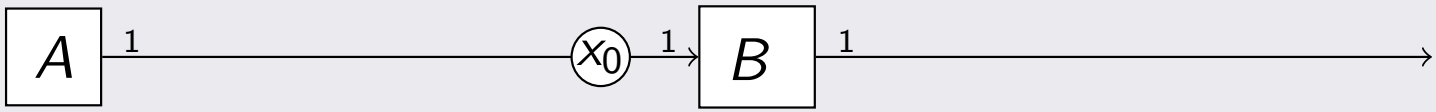
Unbalanced Nodes / Multiple Instances of Nodes



Goal

- Dynamic scheduling (automatic balancing at runtime)
  $\Rightarrow$ Task based execution (OpenMP, Intel TBB, StarSS, ...)
  - decrease overhead of thread-switches (#num nodes $>$ # cores)
  - better balancing (#num nodes $>$ # cores)
  - task $\approx$ execution of node for an iteration (like in SmpSS)

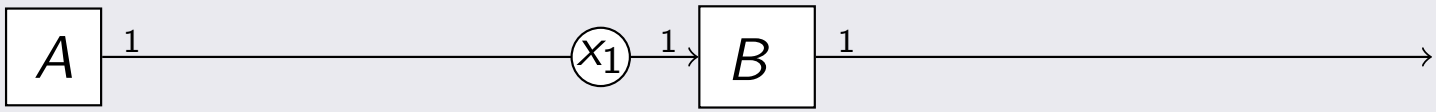
Unbalanced Nodes / Multiple Instances of Nodes



Goal

- Dynamic scheduling (automatic balancing at runtime)
  $\Rightarrow$ Task based execution (OpenMP, Intel TBB, StarSS, ...)
  - decrease overhead of thread-switches (#num nodes $>$ # cores)
  - better balancing (#num nodes $>$ # cores)
  - task $\approx$ execution of node for an iteration (like in SmpSS)

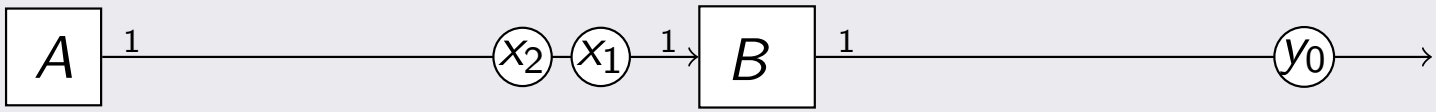
Unbalanced Nodes / Multiple Instances of Nodes



Goal

- Dynamic scheduling (automatic balancing at runtime)
  $\Rightarrow$ Task based execution (OpenMP, Intel TBB, StarSS, ...)
  - decrease overhead of thread-switches (#num nodes $>$ # cores)
  - better balancing (#num nodes $>$ # cores)
  - task $\approx$ execution of node for an iteration (like in SmpSS)

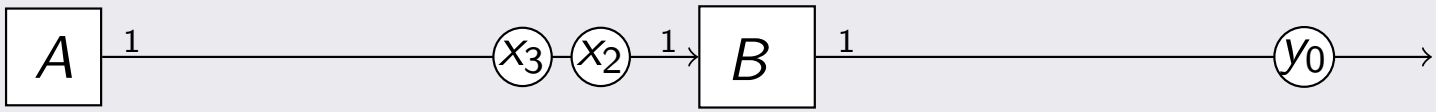
Unbalanced Nodes / Multiple Instances of Nodes



Goal

- Dynamic scheduling (automatic balancing at runtime)
  $\Rightarrow$ Task based execution (OpenMP, Intel TBB, StarSS, ...)
  - decrease overhead of thread-switches (#num nodes $>$ # cores)
  - better balancing (#num nodes $>$ # cores)
  - task $\approx$ execution of node for an iteration (like in SmpSS)

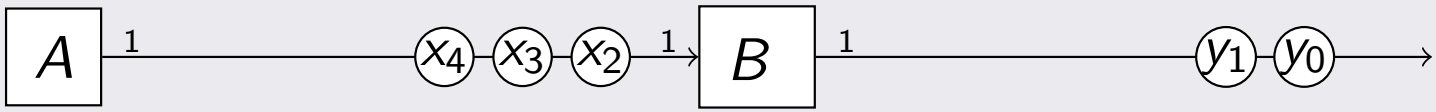
Unbalanced Nodes / Multiple Instances of Nodes



Goal

- Dynamic scheduling (automatic balancing at runtime)
  $\Rightarrow$ Task based execution (OpenMP, Intel TBB, StarSS, ...)
  - decrease overhead of thread-switches (#num nodes $>$ # cores)
  - better balancing (#num nodes $>$ # cores)
  - task $\approx$ execution of node for an iteration (like in SmpSS)

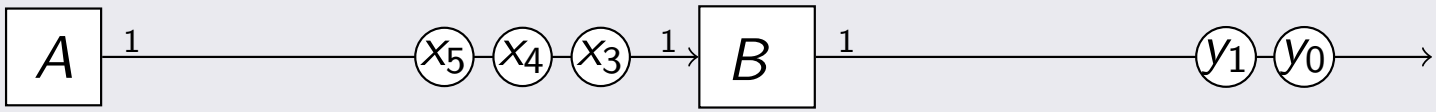
Unbalanced Nodes / Multiple Instances of Nodes



Goal

- Dynamic scheduling (automatic balancing at runtime)
  $\Rightarrow$ Task based execution (OpenMP, Intel TBB, StarSS, ...)
  - decrease overhead of thread-switches (#num nodes $>$ # cores)
  - better balancing (#num nodes $>$ # cores)
  - task $\approx$ execution of node for an iteration (like in SmpSS)

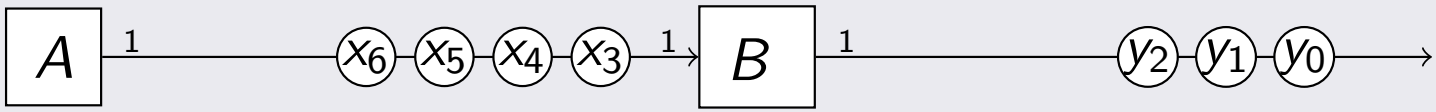
Unbalanced Nodes / Multiple Instances of Nodes



Goal

- Dynamic scheduling (automatic balancing at runtime)
  $\Rightarrow$ Task based execution (OpenMP, Intel TBB, StarSS, ...)
  - decrease overhead of thread-switches (#num nodes $>$ # cores)
  - better balancing (#num nodes $>$ # cores)
  - task $\approx$ execution of node for an iteration (like in SmpSS)

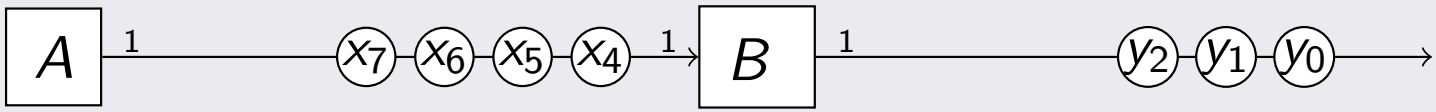
Unbalanced Nodes / Multiple Instances of Nodes



Goal

- Dynamic scheduling (automatic balancing at runtime)
  $\Rightarrow$ Task based execution (OpenMP, Intel TBB, StarSS, ...)
  - decrease overhead of thread-switches (#num nodes $>$ # cores)
  - better balancing (#num nodes $>$ # cores)
  - task $\approx$ execution of node for an iteration (like in SmpSS)

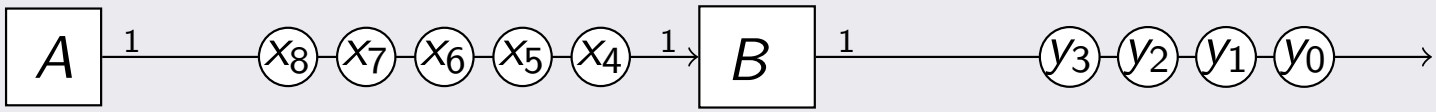
Unbalanced Nodes / Multiple Instances of Nodes



Goal

- Dynamic scheduling (automatic balancing at runtime)
  $\Rightarrow$ Task based execution (OpenMP, Intel TBB, StarSS, ...)
  - decrease overhead of thread-switches (#num nodes $>$ # cores)
  - better balancing (#num nodes $>$ # cores)
  - task $\approx$ execution of node for an iteration (like in SmpSS)

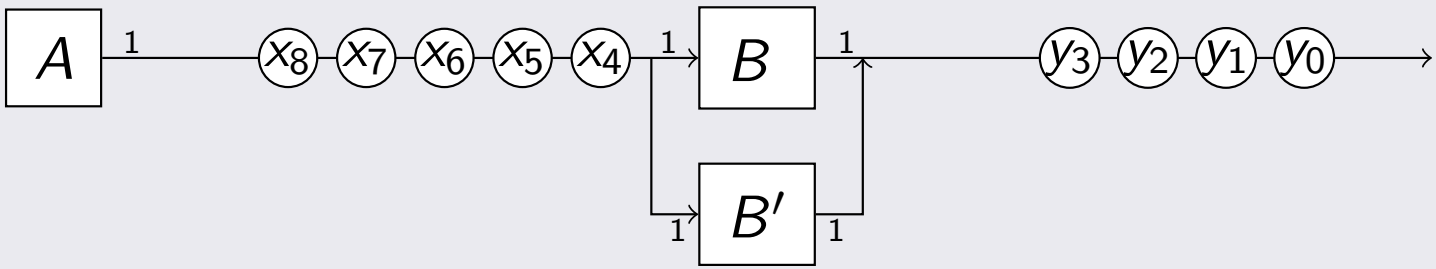
Unbalanced Nodes / Multiple Instances of Nodes



Goal

- Dynamic scheduling (automatic balancing at runtime)
  $\Rightarrow$ Task based execution (OpenMP, Intel TBB, StarSS, ...)
  - decrease overhead of thread-switches (#num nodes $>$ # cores)
  - better balancing (#num nodes $>$ # cores)
  - task $\approx$ execution of node for an iteration (like in SmpSS)
- Execute iterations of single nodes in parallel
  $\Rightarrow$ increase parallelism (#num nodes $<$ # cores)
  $\Rightarrow$ counter unbalanced/varying computational effort

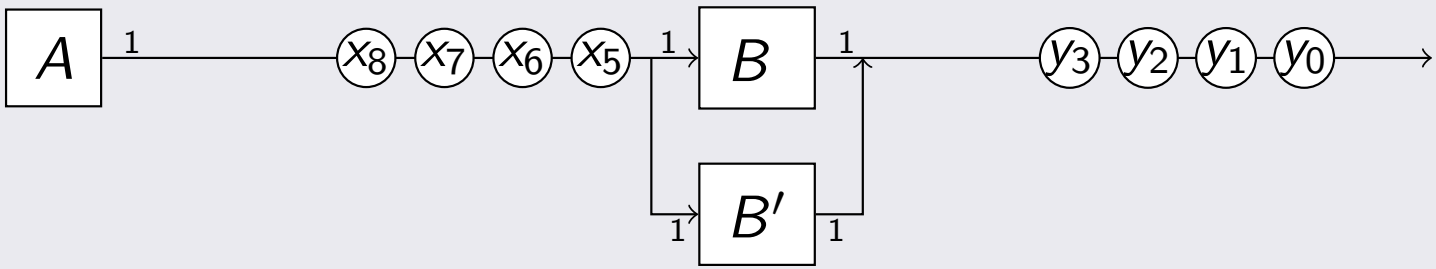
Unbalanced Nodes / Multiple Instances of Nodes



Goal

- Dynamic scheduling (automatic balancing at runtime)
  $\Rightarrow$ Task based execution (OpenMP, Intel TBB, StarSS, ...)
  - decrease overhead of thread-switches (#num nodes $>$ # cores)
  - better balancing (#num nodes $>$ # cores)
  - task $\approx$ execution of node for an iteration (like in SmpSS)
- Execute iterations of single nodes in parallel
  $\Rightarrow$ increase parallelism (#num nodes $<$ # cores)
  $\Rightarrow$ counter unbalanced/varying computational effort

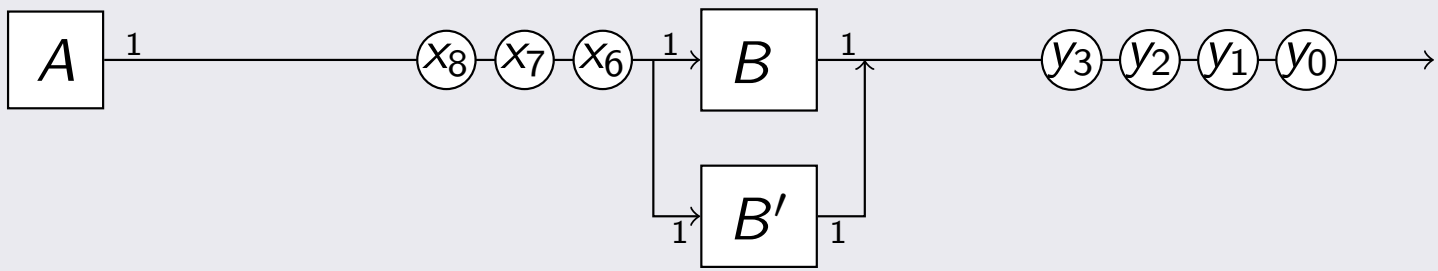
Unbalanced Nodes / Multiple Instances of Nodes



Goal

- Dynamic scheduling (automatic balancing at runtime)
  $\Rightarrow$ Task based execution (OpenMP, Intel TBB, StarSS, ...)
  - decrease overhead of thread-switches (#num nodes $>$ # cores)
  - better balancing (#num nodes $>$ # cores)
  - task $\approx$ execution of node for an iteration (like in SmpSS)
- Execute iterations of single nodes in parallel
  $\Rightarrow$ increase parallelism (#num nodes $<$ # cores)
  $\Rightarrow$ counter unbalanced/varying computational effort

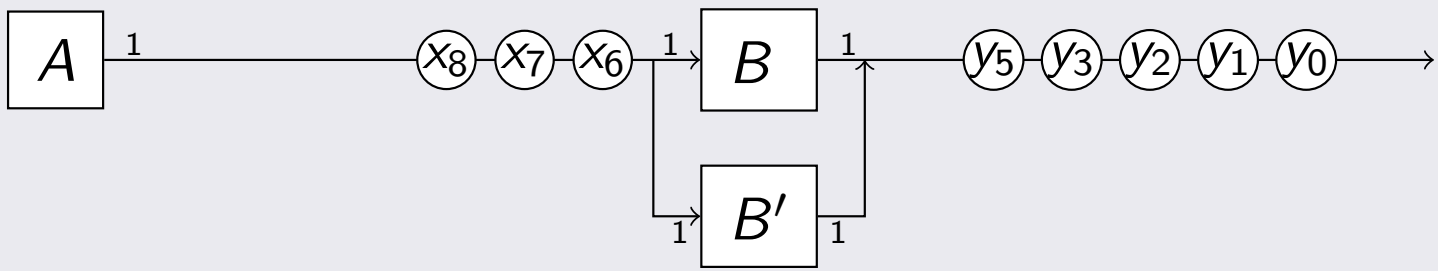
Unbalanced Nodes / Multiple Instances of Nodes



Goal

- Dynamic scheduling (automatic balancing at runtime)
  $\Rightarrow$ Task based execution (OpenMP, Intel TBB, StarSS, ...)
  - decrease overhead of thread-switches (#num nodes $>$ # cores)
  - better balancing (#num nodes $>$ # cores)
  - task $\approx$ execution of node for an iteration (like in SmpSS)
- Execute iterations of single nodes in parallel
  $\Rightarrow$ increase parallelism (#num nodes $<$ # cores)
  $\Rightarrow$ counter unbalanced/varying computational effort

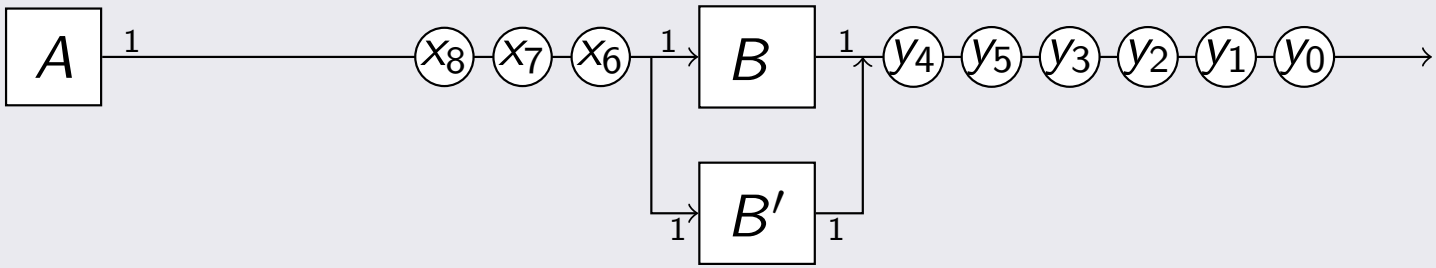
Unbalanced Nodes / Multiple Instances of Nodes



Goal

- Dynamic scheduling (automatic balancing at runtime)
  $\Rightarrow$ Task based execution (OpenMP, Intel TBB, StarSS, ...)
  - decrease overhead of thread-switches (#num nodes $>$ # cores)
  - better balancing (#num nodes $>$ # cores)
  - task $\approx$ execution of node for an iteration (like in SmpSS)
- Execute iterations of single nodes in parallel
  $\Rightarrow$ increase parallelism (#num nodes $<$ # cores)
  $\Rightarrow$ counter unbalanced/varying computational effort

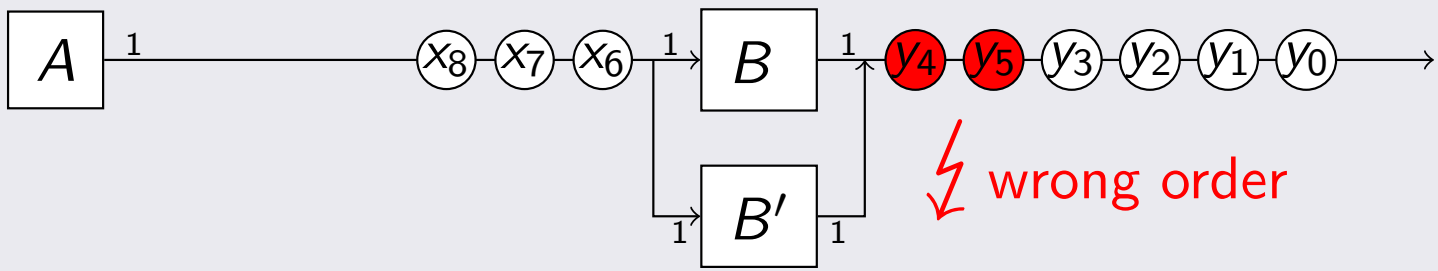
Unbalanced Nodes / Multiple Instances of Nodes



Goal

- Dynamic scheduling (automatic balancing at runtime)
  $\Rightarrow$ Task based execution (OpenMP, Intel TBB, StarSS, ...)
  - decrease overhead of thread-switches (#num nodes $>$ # cores)
  - better balancing (#num nodes $>$ # cores)
  - task $\approx$ execution of node for an iteration (like in SmpSS)
- Execute iterations of single nodes in parallel
  $\Rightarrow$ increase parallelism (#num nodes $<$ # cores)
  $\Rightarrow$ counter unbalanced/varying computational effort
  - in-order reads of buffers does not guarentee in-order writes

Unbalanced Nodes / Multiple Instances of Nodes



Outline

Out-Of-Order in Hardware

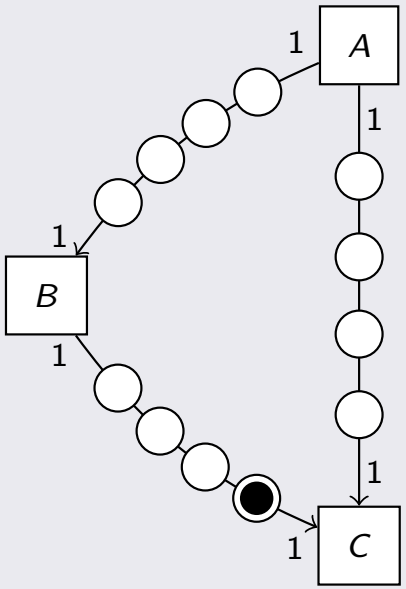
- Tomasulo's Algorithm
- Execute sequence of instructions in data-flow order
- Reservation Station
  - ring buffer
  - each "line" $\approx$ one instruction to execute
    $\Rightarrow$ window of instructions to execute

Out-Of-Order in Software

- Central Buffer Station (CBS)
  - ring buffer
  - each "line" $\approx$ one execution of HSDF
  - each "line" requires each task to be schedules once
    here: several schedules / line
    reservation station: one schedule / line
  - contains counter to keep track of remaining dependencies between tasks
- Worker threads + task queue
  - $\Rightarrow$ task based execution
  - $\Rightarrow$ execute tasks, afterwards release dependencies

Example

SDFG



CBS / Task Queue(TQ) / Worker(WT)

it	AB	AC	BC	A	B	C	h_D
0			$\surd$	0	1	1	3
1				0	1	2	4
2				0	1	2	4
3				0	2	2	4

$TQ = \{(A, 0); (A, 1); (A, 2); (A, 3)\}$

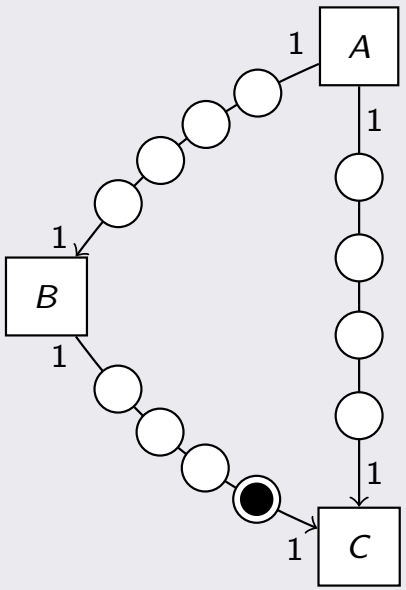
WT_1	WT_2	WT_3	WT_4

What's happening ?

- Initialization

Example

SDFG



CBS / Task Queue(TQ) / Worker(WT)

it	AB	AC	BC	A	B	C	h_D
0			$\surd$	0	1	1	3
1				0	1	2	4
2				0	1	2	4
3				0	2	2	4

$TQ = \{\}$

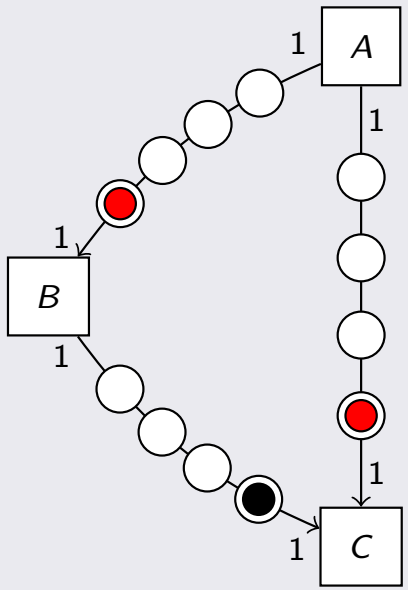
WT_1	WT_2	WT_3	WT_4
$(A, 0)$	$(A, 1)$	$(A, 2)$	$(A, 3)$

What's happening ?

- Workers get tasks from TQ and start processing

Example

SDFG



CBS / Task Queue(TQ) / Worker(WT)

it	AB	AC	BC	A	B	C	h_D
0	$\surd$	$\surd$	$\surd$	0	1	1	3
1				0	1	2	4
2				0	1	2	4
3				0	2	2	4

$TQ = \{\}$

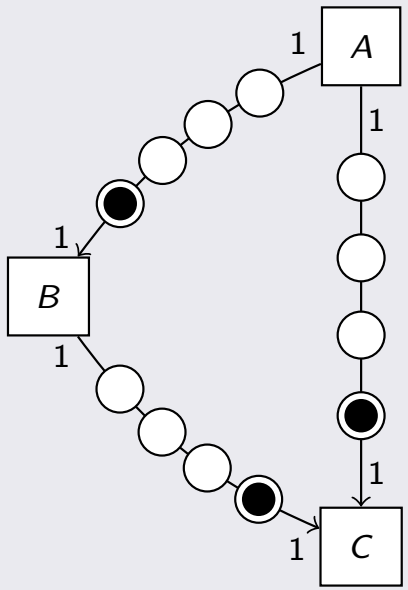
WT_1	WT_2	WT_3	WT_4
$(A, 0)$	$(A, 1)$	$(A, 2)$	$(A, 3)$

What's happening ?

- Worker 1 executed A for iteration 0
- Worker 1 has to remove corresponding dependencies

Example

SDFG



CBS / Task Queue(TQ) / Worker(WT)

it	AB	AC	BC	A	B	C	h_D
0	$\surd$	$\surd$	$\surd$	0	0	1	3
1				0	1	2	4
2				0	1	2	4
3				0	2	2	4

$TQ = \{(B, 0)\}$

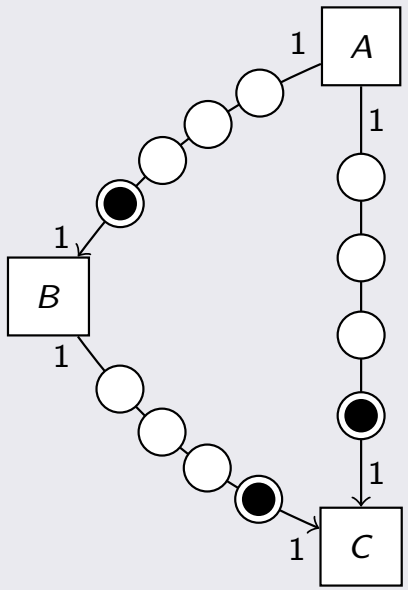
WT_1	WT_2	WT_3	WT_4
$(A, 0)$	$(A, 1)$	$(A, 2)$	$(A, 3)$

What's happening ?

- Worker 1 executed A for iteration 0
- Worker 1 has to remove corresponding dependencies

Example

SDFG



CBS / Task Queue(TQ) / Worker(WT)

it	AB	AC	BC	A	B	C	h_D
0	$\surd$	$\surd$	$\surd$	0	0	0	3
1				0	1	2	4
2				0	1	2	4
3				0	2	2	4

$TQ = \{(B, 0); (C, 0)\}$

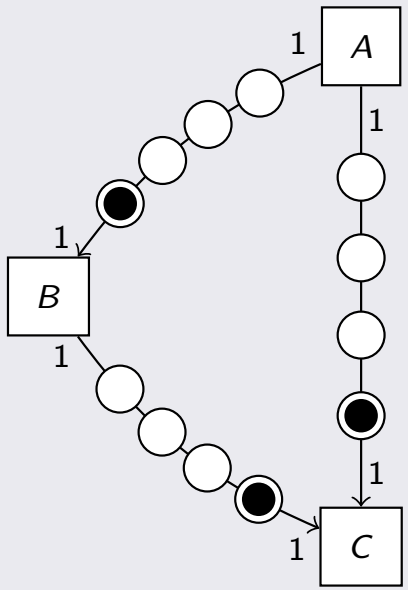
WT_1	WT_2	WT_3	WT_4
$(A, 0)$	$(A, 1)$	$(A, 2)$	$(A, 3)$

What's happening ?

- Worker 1 executed A for iteration 0
- Worker 1 has to remove corresponding dependencies

Example

SDFG



CBS / Task Queue(TQ) / Worker(WT)

it	AB	AC	BC	A	B	C	h_D
0	$\surd$	$\surd$	$\surd$	0	0	0	2
1				0	1	2	4
2				0	1	2	4
3				0	2	2	4

$TQ = \{(B, 0); (C, 0)\}$

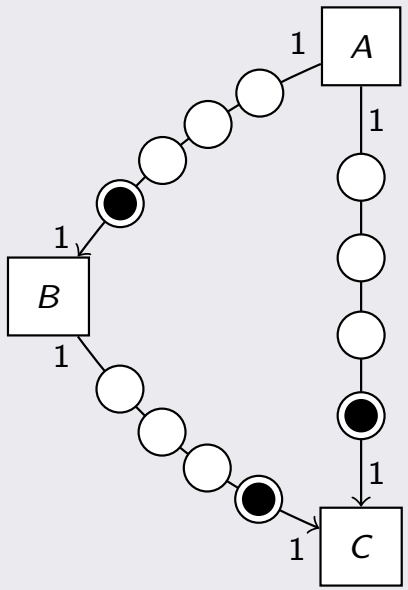
WT_1	WT_2	WT_3	WT_4
$(A, 0)$	$(A, 1)$	$(A, 2)$	$(A, 3)$

What's happening ?

- Worker 1 executed A for iteration 0
- Worker 1 has to remove corresponding dependencies

Example

SDFG



CBS / Task Queue(TQ) / Worker(WT)

it	AB	AC	BC	A	B	C	h_D
0	$\surd$	$\surd$	$\surd$	0	0	0	2
1				0	1	2	4
2				0	1	2	4
3				0	2	2	4

$TQ = \{(B, 0); (C, 0)\}$

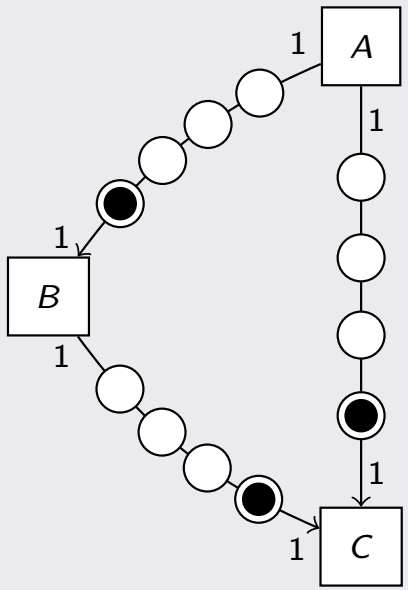
WT_1	WT_2	WT_3	WT_4
	$(A, 1)$	$(A, 2)$	$(A, 3)$

What's happening ?

- Worker 1 is now free to execute the next task

Example

SDFG



CBS / Task Queue(TQ) / Worker(WT)

it	AB	AC	BC	A	B	C	h_D
0	$\surd$	$\surd$	$\surd$	0	0	0	2
1				0	1	2	4
2				0	1	2	4
3				0	2	2	4

$TQ = \{(C, 0)\}$

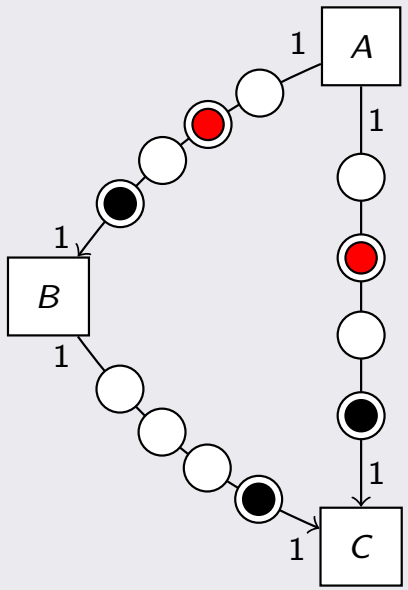
WT_1	WT_2	WT_3	WT_4
$(B, 0)$	$(A, 1)$	$(A, 2)$	$(A, 3)$

What's happening ?

- Worker 1 gets next task from TQ and starts processing

Example

SDFG



CBS / Task Queue(TQ) / Worker(WT)

it	AB	AC	BC	A	B	C	h_D
0	√	√	√	0	0	0	2
1				0	1	2	4
2	√	√		0	1	2	4
3				0	2	2	4

$TQ = \{(C, 0)\}$

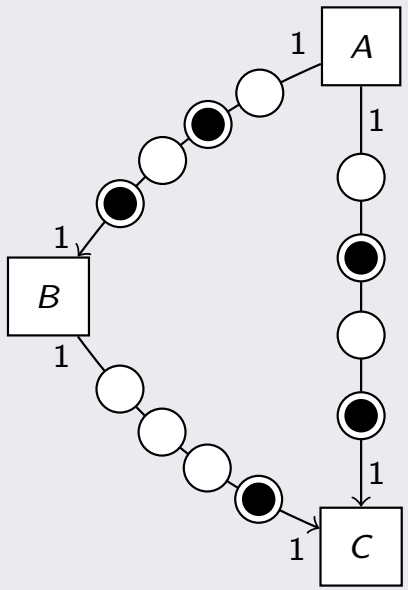
WT_1	WT_2	WT_3	WT_4
$(B, 0)$	$(A, 1)$	$(A, 2)$	$(A, 3)$

What's happening ?

- Worker 3 executed A for iteration 2
- Worker 3 has to remove corresponding dependencies

Example

SDFG



CBS / Task Queue(TQ) / Worker(WT)

it	AB	AC	BC	A	B	C	h_D
0	$\surd$	$\surd$	$\surd$	0	0	0	2
1				0	1	2	4
2	$\surd$	$\surd$		0	0	1	3
3				0	2	2	4

$TQ = \{(C, 0); (B, 2)\}$

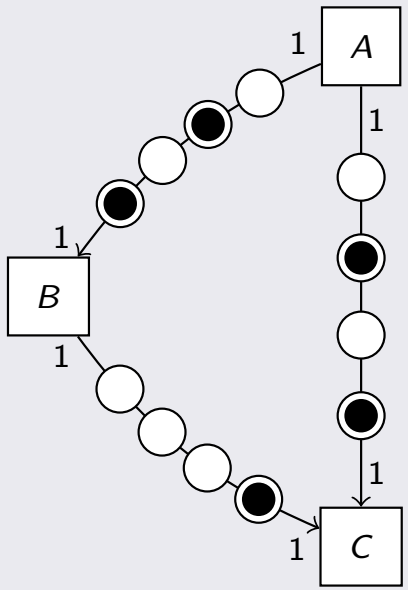
WT_1	WT_2	WT_3	WT_4
$(B, 0)$	$(A, 1)$	$(A, 2)$	$(A, 3)$

What's happening ?

- Worker 3 executed A for iteration 2
- Worker 3 has to remove corresponding dependencies

Example

SDFG



CBS / Task Queue(TQ) / Worker(WT)

it	AB	AC	BC	A	B	C	h_D
0	$\surd$	$\surd$	$\surd$	0	0	0	2
1				0	1	2	4
2	$\surd$	$\surd$		0	0	1	3
3				0	2	2	4

$TQ = \{(C, 0); (B, 2)\}$

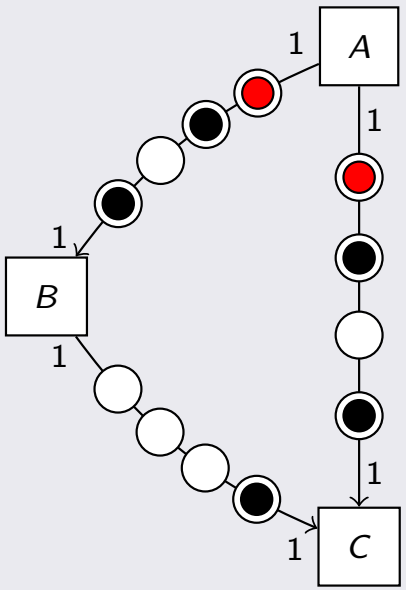
WT_1	WT_2	WT_3	WT_4
$(B, 0)$	$(A, 1)$		$(A, 3)$

What's happening ?

- Worker 3 is now free to execute the next task

Example

SDFG



CBS / Task Queue(TQ) / Worker(WT)

it	AB	AC	BC	A	B	C	h_D
0	√	√	√	0	0	0	2
1				0	1	2	4
2	√	√		0	0	1	3
3	√	√		0	2	2	4

$TQ = \{(C, 0); (B, 2)\}$

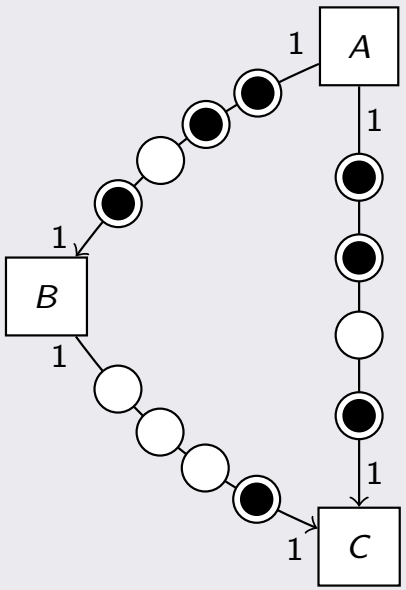
WT_1	WT_2	WT_3	WT_4
$(B, 0)$	$(A, 1)$		$(A, 3)$

What's happening ?

- Worker 4 executed A for iteration 3
- Worker 4 has to remove corresponding dependencies

Example

SDFG



CBS / Task Queue(TQ) / Worker(WT)

it	AB	AC	BC	A	B	C	h_D
0	$\surd$	$\surd$	$\surd$	0	0	0	2
1				0	1	2	4
2	$\surd$	$\surd$		0	0	1	3
3	$\surd$	$\surd$		0	1	1	3

$TQ = \{(C, 0); (B, 2)\}$

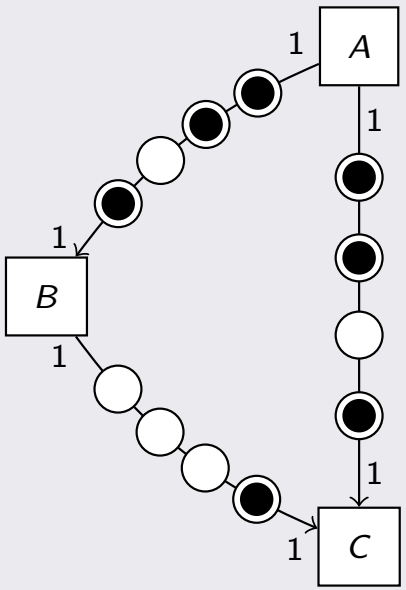
WT_1	WT_2	WT_3	WT_4
$(B, 0)$	$(A, 1)$		$(A, 3)$

What's happening ?

- Worker 4 executed A for iteration 3
- Worker 4 has to remove corresponding dependencies

Example

SDFG



CBS / Task Queue(TQ) / Worker(WT)

it	AB	AC	BC	A	B	C	h_D
0	$\surd$	$\surd$	$\surd$	0	0	0	2
1				0	1	2	4
2	$\surd$	$\surd$		0	0	1	3
3	$\surd$	$\surd$		0	1	1	3

$TQ = \{(C, 0); (B, 2)\}$

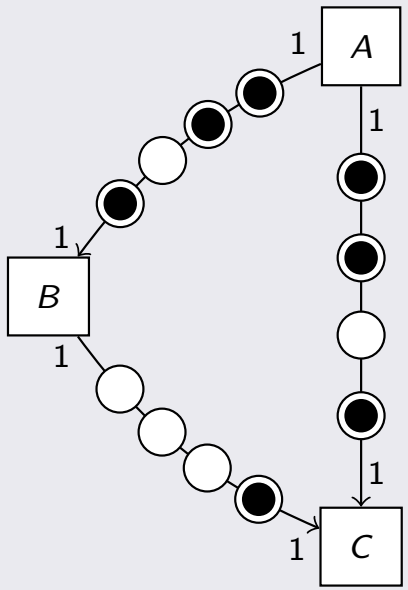
WT_1	WT_2	WT_3	WT_4
$(B, 0)$	$(A, 1)$		

What's happening ?

- Worker 4 is now free to execute the next task

Example

SDFG



CBS / Task Queue(TQ) / Worker(WT)

it	AB	AC	BC	A	B	C	h_D
0	$\surd$	$\surd$	$\surd$	0	0	0	2
1				0	1	2	4
2	$\surd$	$\surd$		0	0	1	3
3	$\surd$	$\surd$		0	1	1	3

$TQ = \{\}$

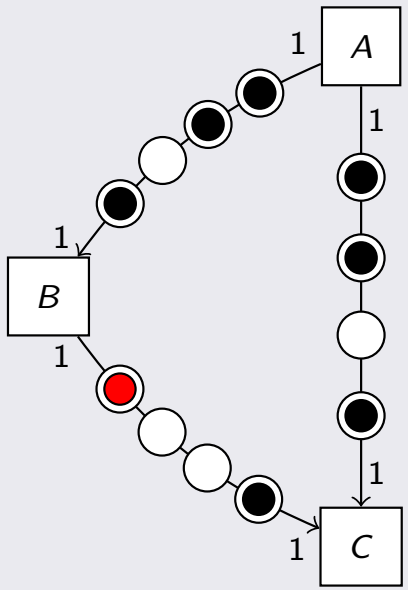
WT_1	WT_2	WT_3	WT_4
$(B, 0)$	$(A, 1)$	$(C, 0)$	$(B, 2)$

What's happening ?

- Worker 3, 4 get tasks from TQ and start processing

Example

SDFG



CBS / Task Queue(TQ) / Worker(WT)

it	AB	AC	BC	A	B	C	h_D
0	$\surd$	$\surd$	$\surd$	0	0	0	2
1				0	1	2	4
2	$\surd$	$\surd$		0	0	1	3
3	$\surd$	$\surd$	$\surd$	0	1	1	3

$TQ = \{\}$

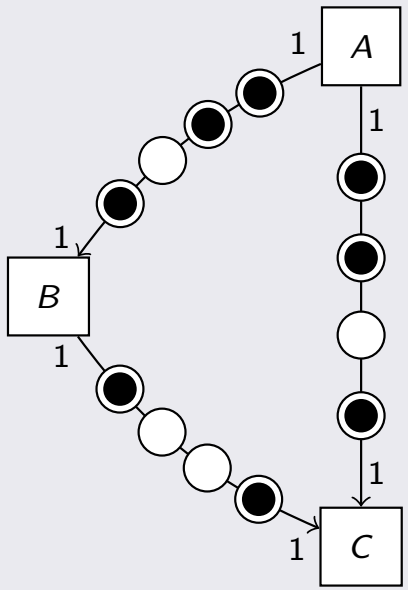
WT_1	WT_2	WT_3	WT_4
$(B, 0)$	$(A, 1)$	$(C, 0)$	$(B, 2)$

What's happening ?

- Worker 4 executed B for iteration 2
- Worker 4 has to remove corresponding dependencies

Example

SDFG



CBS / Task Queue(TQ) / Worker(WT)

it	AB	AC	BC	A	B	C	h_D
0	$\surd$	$\surd$	$\surd$	0	0	0	2
1				0	1	2	4
2	$\surd$	$\surd$		0	0	1	2
3	$\surd$	$\surd$	$\surd$	0	1	0	3

$TQ = \{(C, 3)\}$

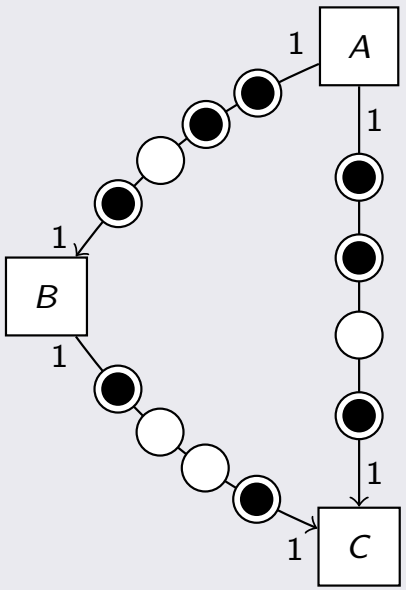
WT_1	WT_2	WT_3	WT_4
$(B, 0)$	$(A, 1)$	$(C, 0)$	$(B, 2)$

What's happening ?

- Worker 4 executed B for iteration 2
- Worker 4 has to remove corresponding dependencies

Example

SDFG



CBS / Task Queue(TQ) / Worker(WT)

it	AB	AC	BC	A	B	C	h_D
0	$\surd$	$\surd$	$\surd$	0	0	0	2
1				0	1	2	4
2	$\surd$	$\surd$		0	0	1	2
3	$\surd$	$\surd$	$\surd$	0	1	0	3

$TQ = \{(C, 3)\}$

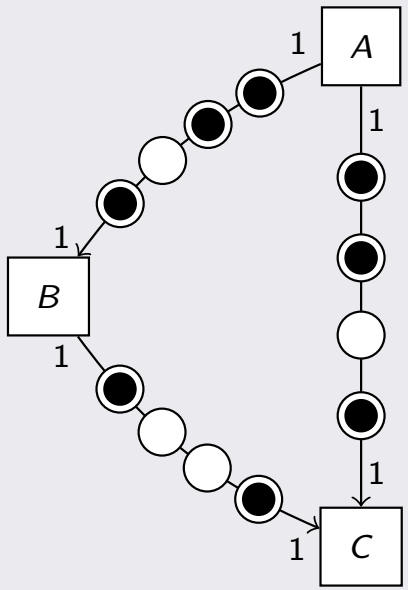
WT_1	WT_2	WT_3	WT_4
$(B, 0)$	$(A, 1)$	$(C, 0)$	

What's happening ?

- Worker 4 is now free to execute the next task

Example

SDFG



CBS / Task Queue(TQ) / Worker(WT)

it	AB	AC	BC	A	B	C	h_D
0	$\surd$	$\surd$	$\surd$	0	0	0	2
1				0	1	2	4
2	$\surd$	$\surd$		0	0	1	2
3	$\surd$	$\surd$	$\surd$	0	1	0	3

$TQ = \{(C, 3)\}$

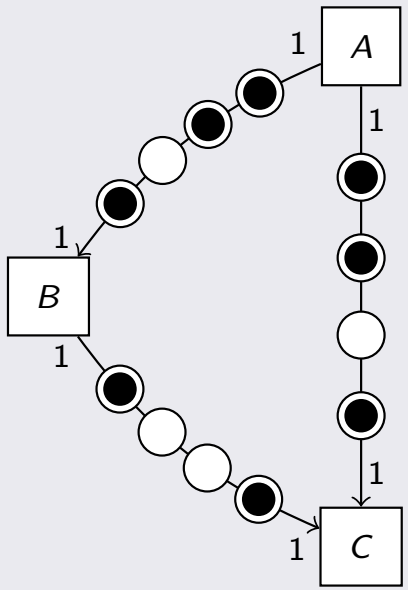
WT_1	WT_2	WT_3	WT_4
$(B, 0)$	$(A, 1)$	$(C, 0)$	

What's happening ?

- Worker 3 executed C for iteration 0
- Worker 3 has to remove corresponding dependencies

Example

SDFG



CBS / Task Queue(TQ) / Worker(WT)

it	AB	AC	BC	A	B	C	h_D
0	$\surd$	$\surd$	$\surd$	0	0	0	1
1				0	1	2	4
2	$\surd$	$\surd$		0	0	1	2
3	$\surd$	$\surd$	$\surd$	0	1	0	3

$TQ = \{(C, 3)\}$

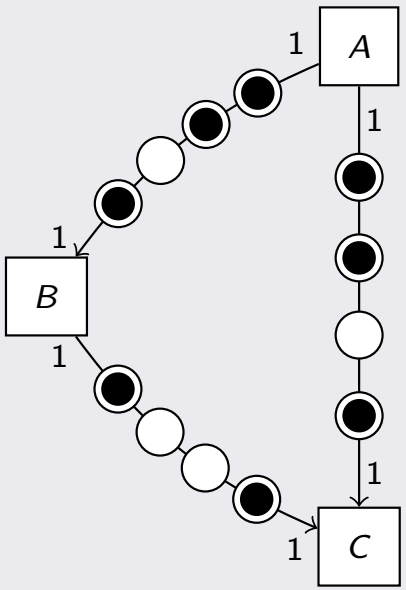
WT_1	WT_2	WT_3	WT_4
$(B, 0)$	$(A, 1)$	$(C, 0)$	

What's happening ?

- Worker 3 executed C for iteration 0
- Worker 3 has to remove corresponding dependencies

Example

SDFG



CBS / Task Queue(TQ) / Worker(WT)

it	AB	AC	BC	A	B	C	h_D
0	$\surd$	$\surd$	$\surd$	0	0	0	1
1				0	1	2	4
2	$\surd$	$\surd$		0	0	1	2
3	$\surd$	$\surd$	$\surd$	0	1	0	3

$TQ = \{(C, 3)\}$

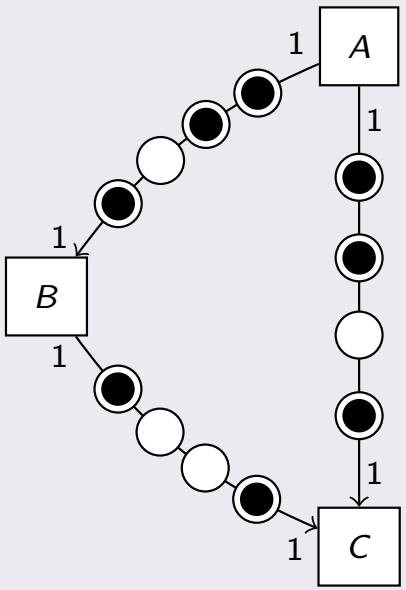
WT_1	WT_2	WT_3	WT_4
$(B, 0)$	$(A, 1)$		

What's happening ?

- Worker 3 is now free to execute the next task

Example

SDFG



CBS / Task Queue(TQ) / Worker(WT)

it	AB	AC	BC	A	B	C	h_D
0	$\surd$	$\surd$	$\surd$	0	0	0	1
1				0	1	2	4
2	$\surd$	$\surd$		0	0	1	2
3	$\surd$	$\surd$	$\surd$	0	1	0	3

$TQ = \{(C, 3)\}$

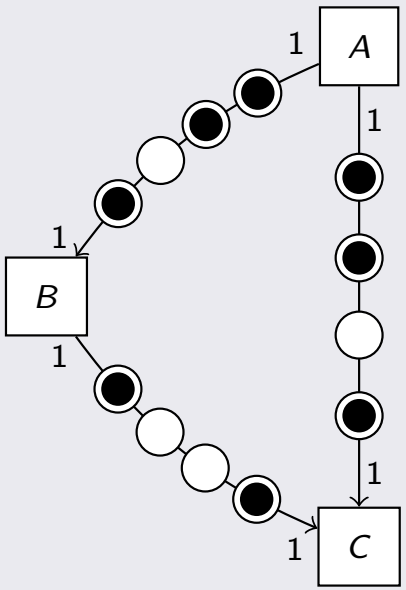
WT_1	WT_2	WT_3	WT_4
$(B, 0)$	$(A, 1)$		

What's happening ?

- Worker 1 executed B for iteration 0
- Worker 1 has to remove corresponding dependencies

Example

SDFG



CBS / Task Queue(TQ) / Worker(WT)

it	AB	AC	BC	A	B	C	h_D
0	$\surd$	$\surd$	$\surd$	0	0	0	0
1				0	1	1	4
2	$\surd$	$\surd$		0	0	1	2
3	$\surd$	$\surd$	$\surd$	0	1	0	3

$TQ = \{(C, 3)\}$

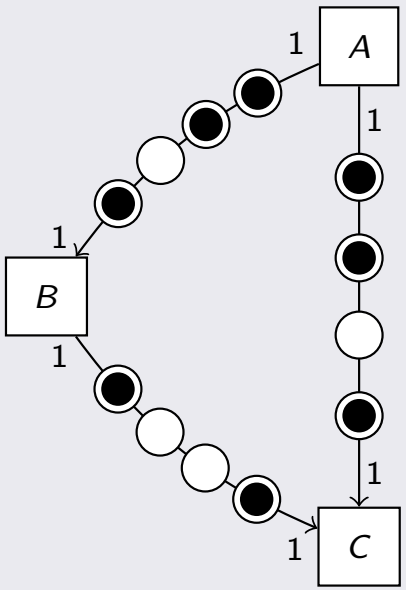
WT_1	WT_2	WT_3	WT_4
$(B, 0)$	$(A, 1)$		

What's happening ?

- Worker 1 executed B for iteration 0
- Worker 1 has to remove corresponding dependencies

Example

SDFG



CBS / Task Queue(TQ) / Worker(WT)

it	AB	AC	BC	A	B	C	h_D
0	$\surd$	$\surd$	$\surd$	0	0	0	0
1				0	1	1	4
2	$\surd$	$\surd$		0	0	1	2
3	$\surd$	$\surd$	$\surd$	0	1	0	3

$TQ = \{(C, 3)\}$

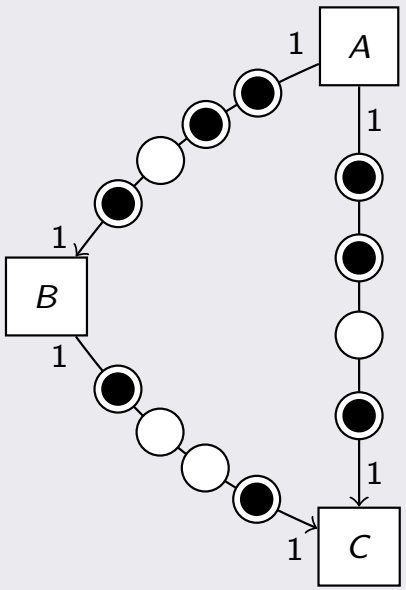
WT_1	WT_2	WT_3	WT_4
	$(A, 1)$		

What's happening ?

- $h_D == 0 \Rightarrow$ head can be removed
- Remove dependencies due to initial tokens

Example

SDFG



CBS / Task Queue(TQ) / Worker(WT)

it	AB	AC	BC	A	B	C	h_D
4				0	2	2	4
1				0	1	1	4
2	$\surd$	$\surd$		0	0	1	2
3	$\surd$	$\surd$	$\surd$	0	1	0	3

$TQ = \{(C, 3), (A, 4)\}$

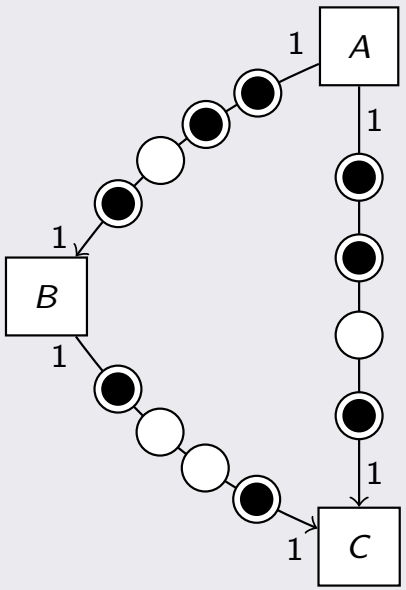
WT_1	WT_2	WT_3	WT_4
	$(A, 1)$		

What's happening ?

- $h_D == 0 \Rightarrow$ head can be removed
- Remove dependencies due to initial tokens

Example

SDFG



CBS / Task Queue(TQ) / Worker(WT)

it	AB	AC	BC	A	B	C	h_D
4				0	2	2	4
1				0	1	1	3
2	$\surd$	$\surd$		0	0	1	2
3	$\surd$	$\surd$	$\surd$	0	0	0	3

$TQ = \{(C, 3), (A, 4), (B, 3)\}$

WT_1	WT_2	WT_3	WT_4
	$(A, 1)$		

What's happening ?

- $h_D == 0 \Rightarrow$ head can be removed
- Remove dependencies due to initial tokens

Outline

Benchmarks - Results

Benchmark systems: 2x Xeon X5450 (= 8x 3.00GHz)
i5-750 (4x 2.66GHz)

Benchmark Name	Speedup on i5-750		Speedup on 2xXeon	
	1n1t⇒io	io⇒ooo	1n1t⇒io	io⇒ooo
MatrixMult (16x16)	14.40	1.33	23.22	1.06
MatrixMult (32x32)	11.48	1.12	28.95	1.05
MatrixMult (48x48)	10.85	1.10	25.83	1.05
LU Decomp. (4x4)	0.75	1.95	0.67	1.37
LU Decomp. (8x8)	3.97	1.23	2.54	1.05
LU Decomp. (16x16)	4.39	1.07	7.17	1.04
LU Decomp. (32x32)	5.26	1.04	5.46	1.05
DFT (4 tasks)	6.20	1.00	7.10	0.97
DFT (8 tasks)	6.64	0.99	9.22	1.01
DFT (16 tasks)	4.87	1.01	5.56	0.99
DFT (32 tasks)	4.80	1.02	5.57	0.99
Landscape Gen. (H=20)	1.84	1.11	2.56	1.12
Landscape Gen. (H=200)	2.09	1.34	6.75	1.14
Landscape Gen. (H=400)	3.20	1.43	11.15	1.12
Landscape Gen. (H=800)	4.84	1.80	17.28	1.28

Outline

Conclusion

Conclusion

- Dynamic task based execution ⇒ better balancing
- Out-Of-Order Execution to improve TLP
  "Rolling-out" SDF ⇒ more tasks / parallelism
- Similar to programs running on OOO-HW: more independence, e. g. #nodes / #cores do not have to match

Future Work

Large arrays / data structures

- memory-requirement
- may introduce copy-actions

The End

Thank you for your attention!

Questions? Suggestions? Ideas?

The End

Backup Slides

Motivation

- model-based development of applications
  in particular: embedded systems
- synchronous languages, e. g. Esterel, Lustre, Quartz
  - can be used for embedded systems
  - hide communication latencies
  - execution of instructions in perfect synchrony
- synthesis more difficult
  (especially for heterogenous/distributed systems)

Guarded Actions (GA)

System (Example)

Interface:

Inputs: i, d, c

Output: o

Locals: $a[N]$, $j0$, j, p

Guarded Actions:

$p \Rightarrow \text{next}(p) = \text{True}$

$c \wedge p \Rightarrow a[j] = i$

$p \Rightarrow \text{next}(j) = (j+1)\%N$

$p \Rightarrow j0 = (j-d)\%N$

$p \Rightarrow o = a[j0]$

Guarded Actions (GA)

System (Example)

Interface:

Inputs: i, d, c

Output: o

Locals: $a[N]$, $j0$, j, p

Guarded Actions:

$p \Rightarrow \text{next}(p) = \text{True}$

$c \wedge p \Rightarrow a[j] = i$

$p \Rightarrow \text{next}(j) = (j+1)\%N$

$p \Rightarrow j0 = (j-d)\%N$

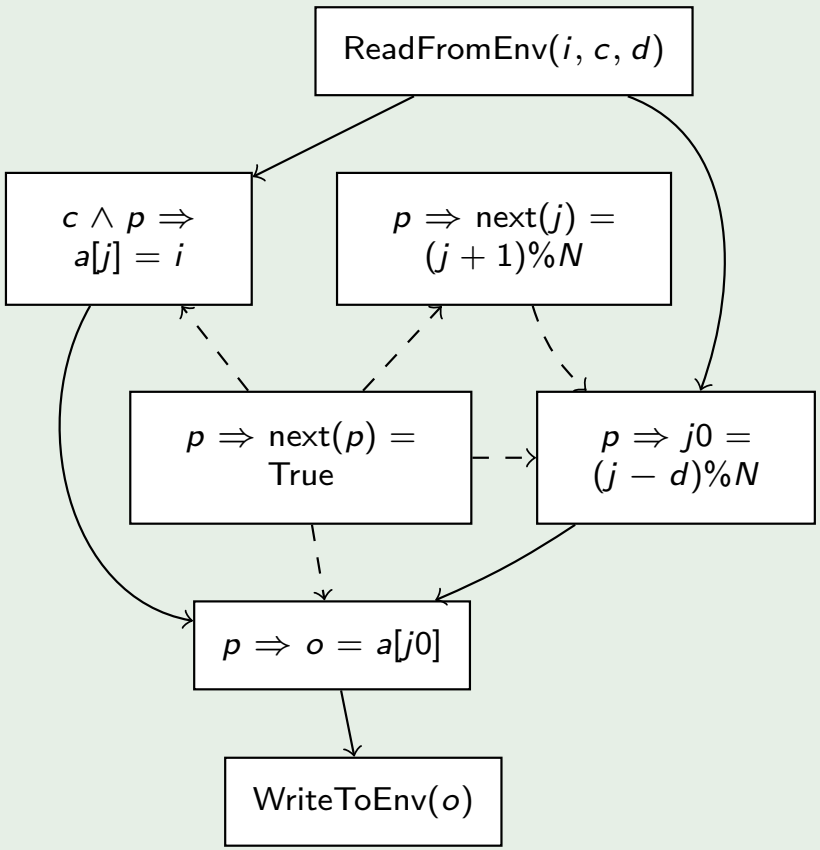
$p \Rightarrow o = a[j0]$

Generic execution

```
while(True)
    read system inputs
    execute GA in data-flow order
        update state of system
        generate outputs of system
    write system outputs
endwhile
```

Guarded Actions (GA)

Dependency Graph

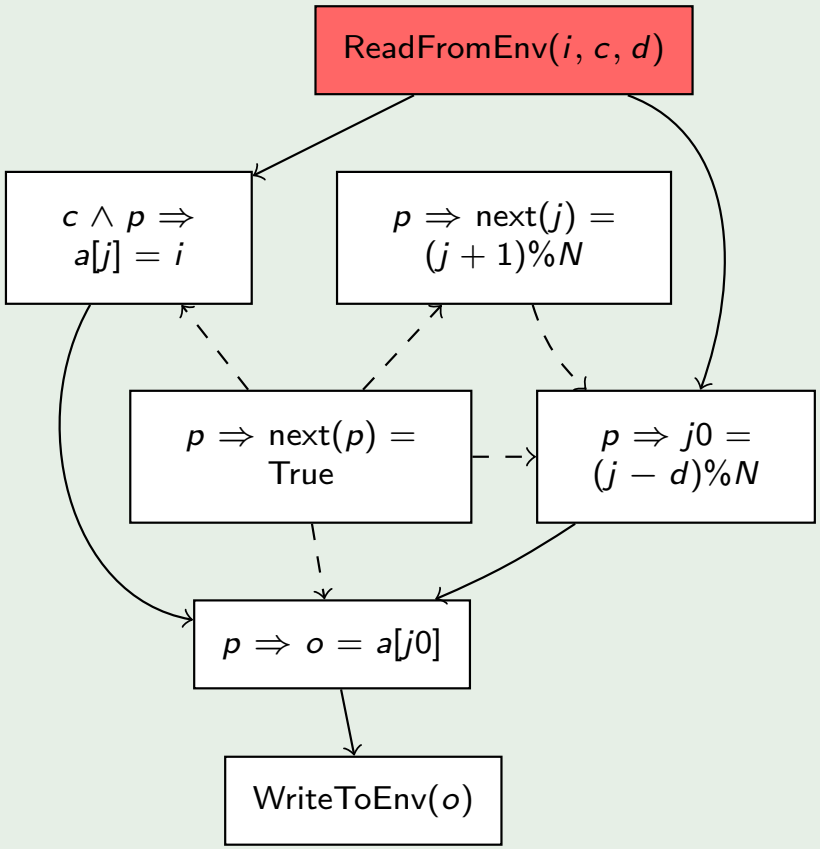


Generic execution

while(True)
    read system inputs
    execute GA in data-flow order
        update state of system
        generate outputs of system
    write system outputs
endwhile

Guarded Actions (GA)

Dependency Graph

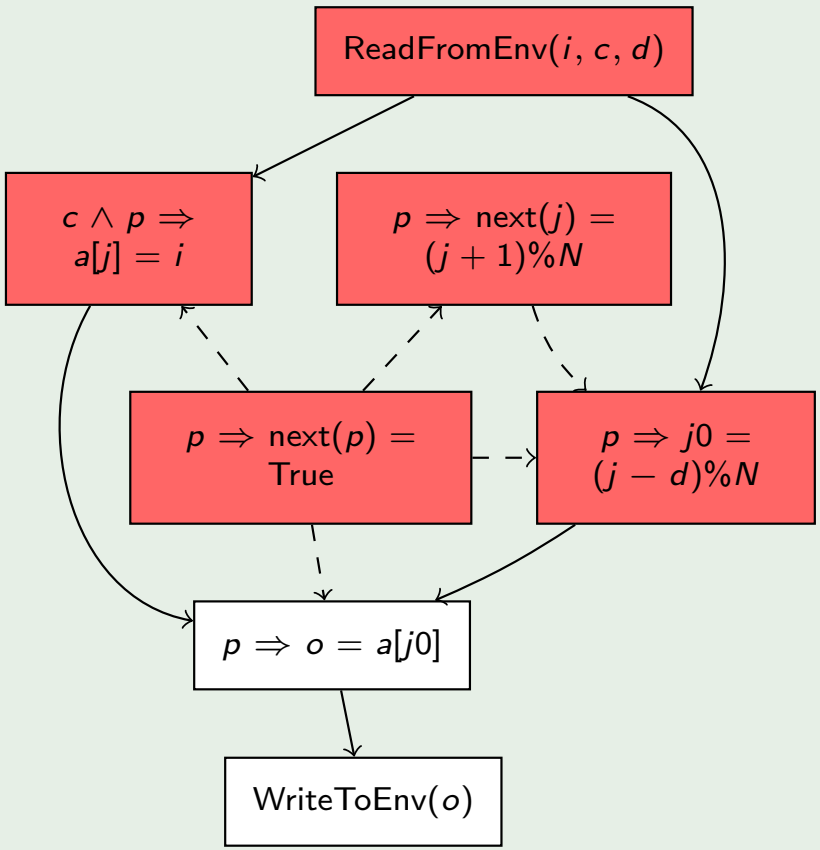


Generic execution

while(True)
    read system inputs
    execute GA in data-flow order
        update state of system
        generate outputs of system
    write system outputs
endwhile

Guarded Actions (GA)

Dependency Graph

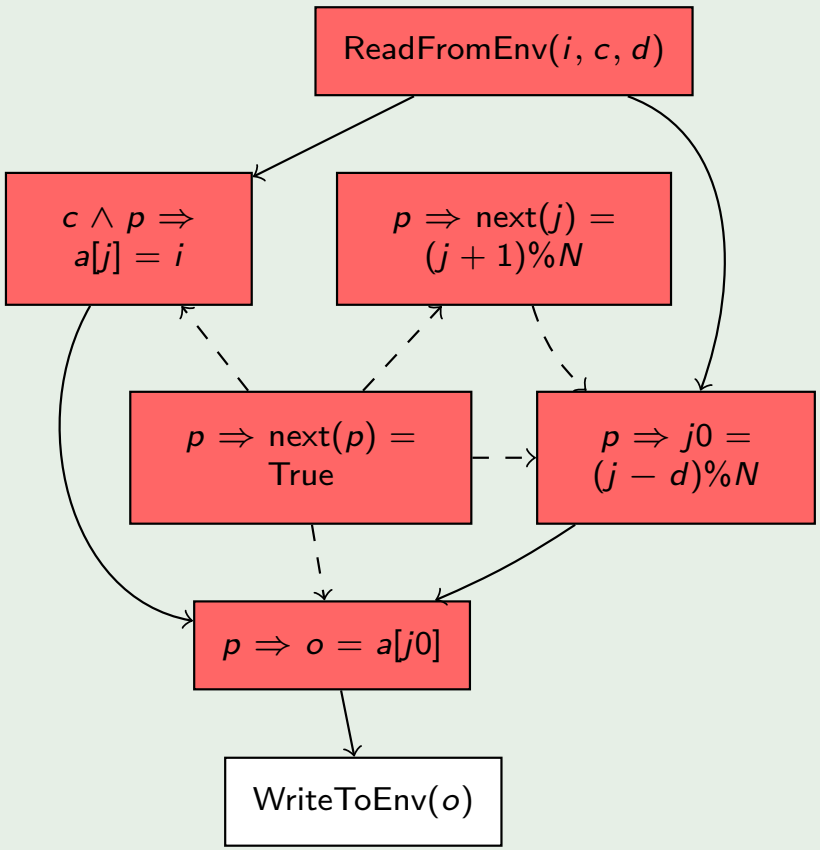


Generic execution

while(True)
    read system inputs
    execute GA in data-flow order
        update state of system
        generate outputs of system
    write system outputs
endwhile

Guarded Actions (GA)

Dependency Graph

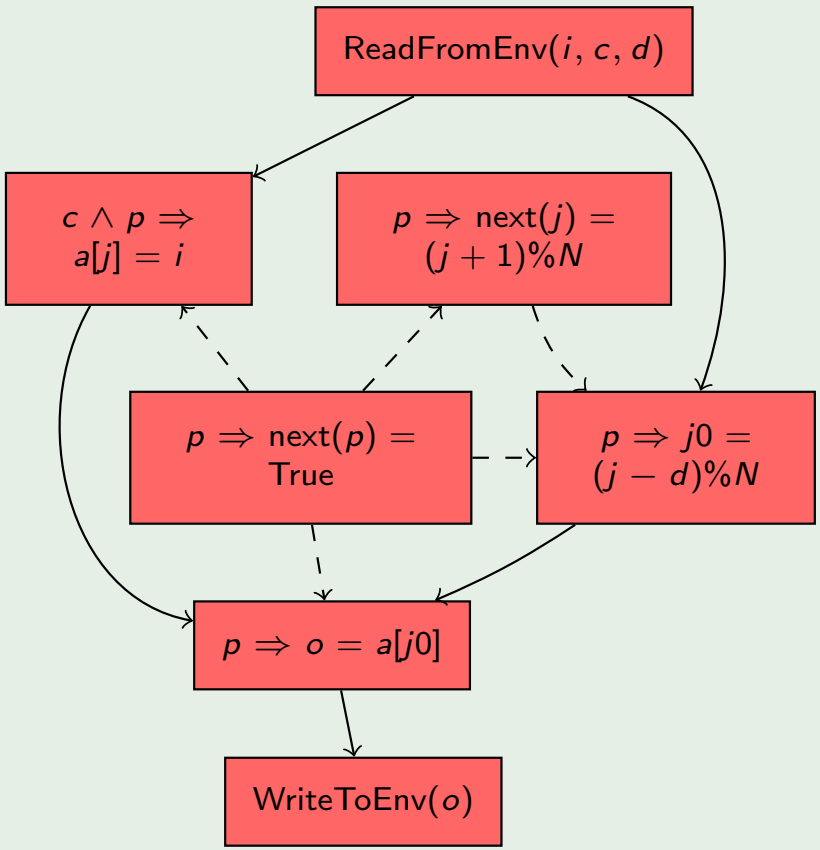


Generic execution

while(True)
    read system inputs
    execute GA in data-flow order
        update state of system
        generate outputs of system
    write system outputs
endwhile

Guarded Actions (GA)

Dependency Graph

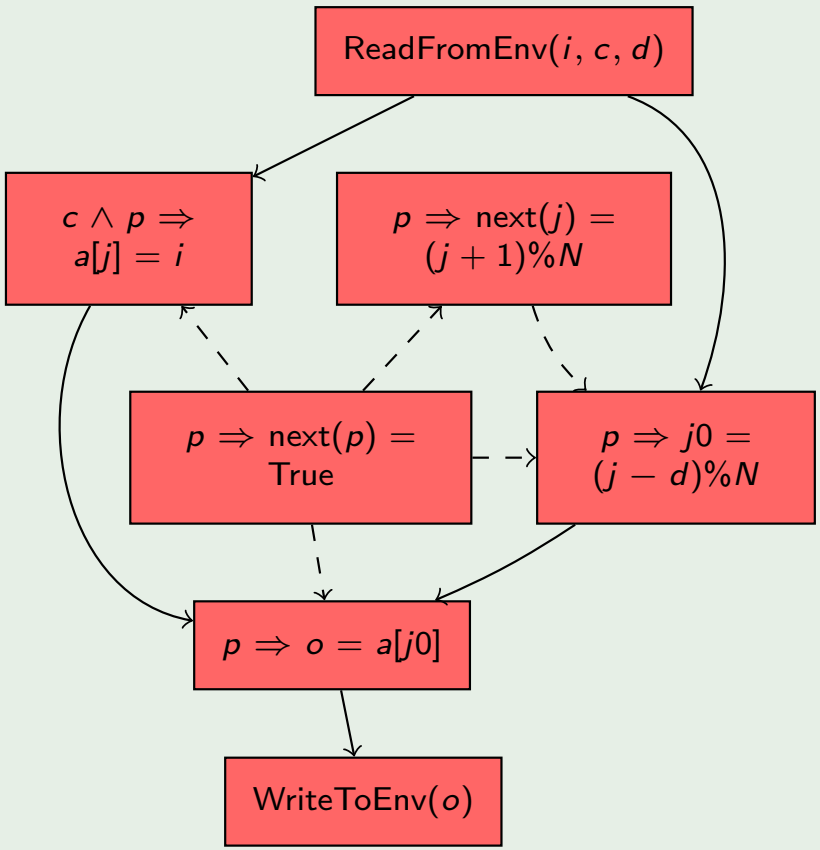


Generic execution

while(True)
    read system inputs
    execute GA in data-flow order
        update state of system
        generate outputs of system
    write system outputs
endwhile

Guarded Actions (GA)

DPN

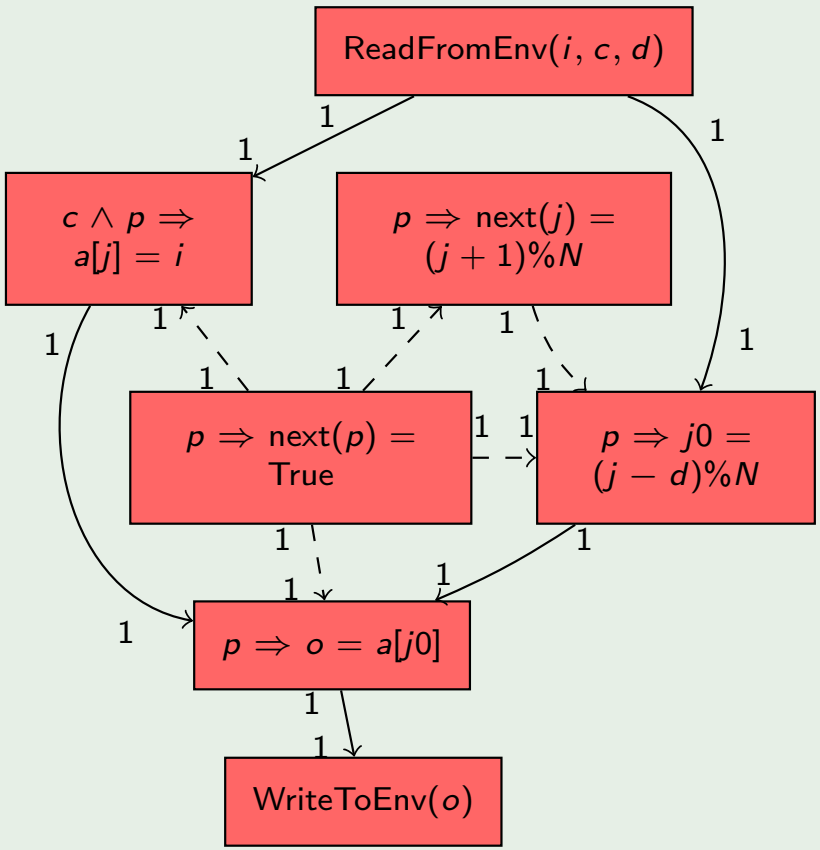


Notes

- DPN is gained by replacing dependencies by FIFO-buffers ⇒ "Theory of Latency Insenstive Design" (McMillan et. al)
- writer must be uniquely determined: group nodes writing to the same variable
- grouping of nodes / partitioning increase computation effort per node

Guarded Actions (GA)

Synchronous DPN

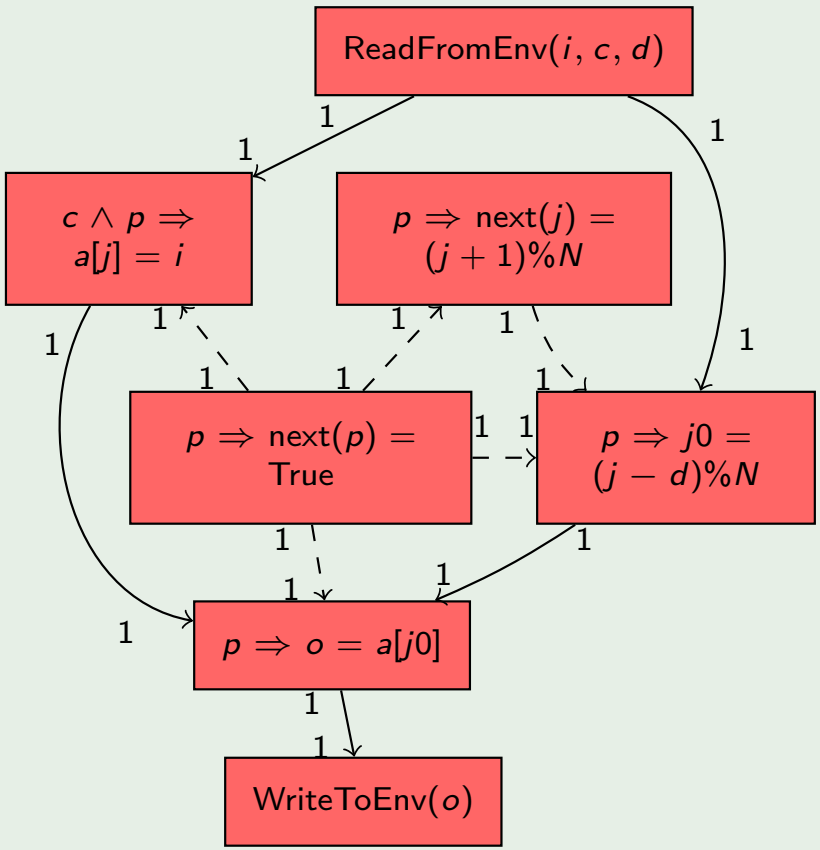


Synchronous DPN

- #num tokens per read/write fixed
- special case in SDPN from sync. lang.: 1 token per read/write per buffer

Guarded Actions (GA)

Synchronous DPN



Synthesis (PThreads)

- create thread for each node
- each thread:

```
while(True)
    read input buffers
    execute GA of node
    write output buffers
endwhile
```