

Data-flow prescheduling for large instruction windows in out-of-order processors

Pierre Michaud, André Seznec

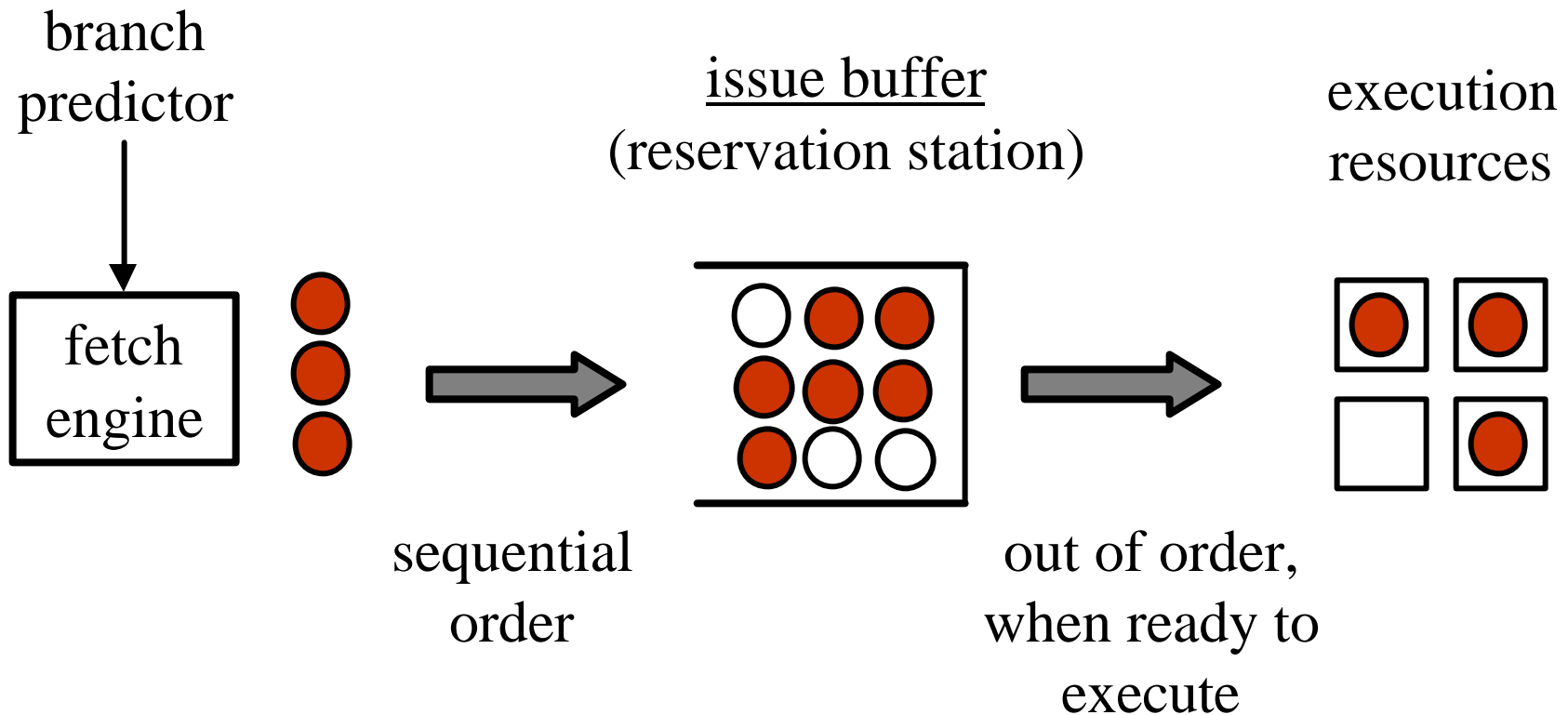
IRISA / INRIA

January 2001

Introduction

- Context: dynamic instruction scheduling in out-of-order superscalar processors
- Problem tackled: how to increase the processor instruction window without impacting the clock cycle

Dynamic scheduling in OoO processors



What issue buffer size ?

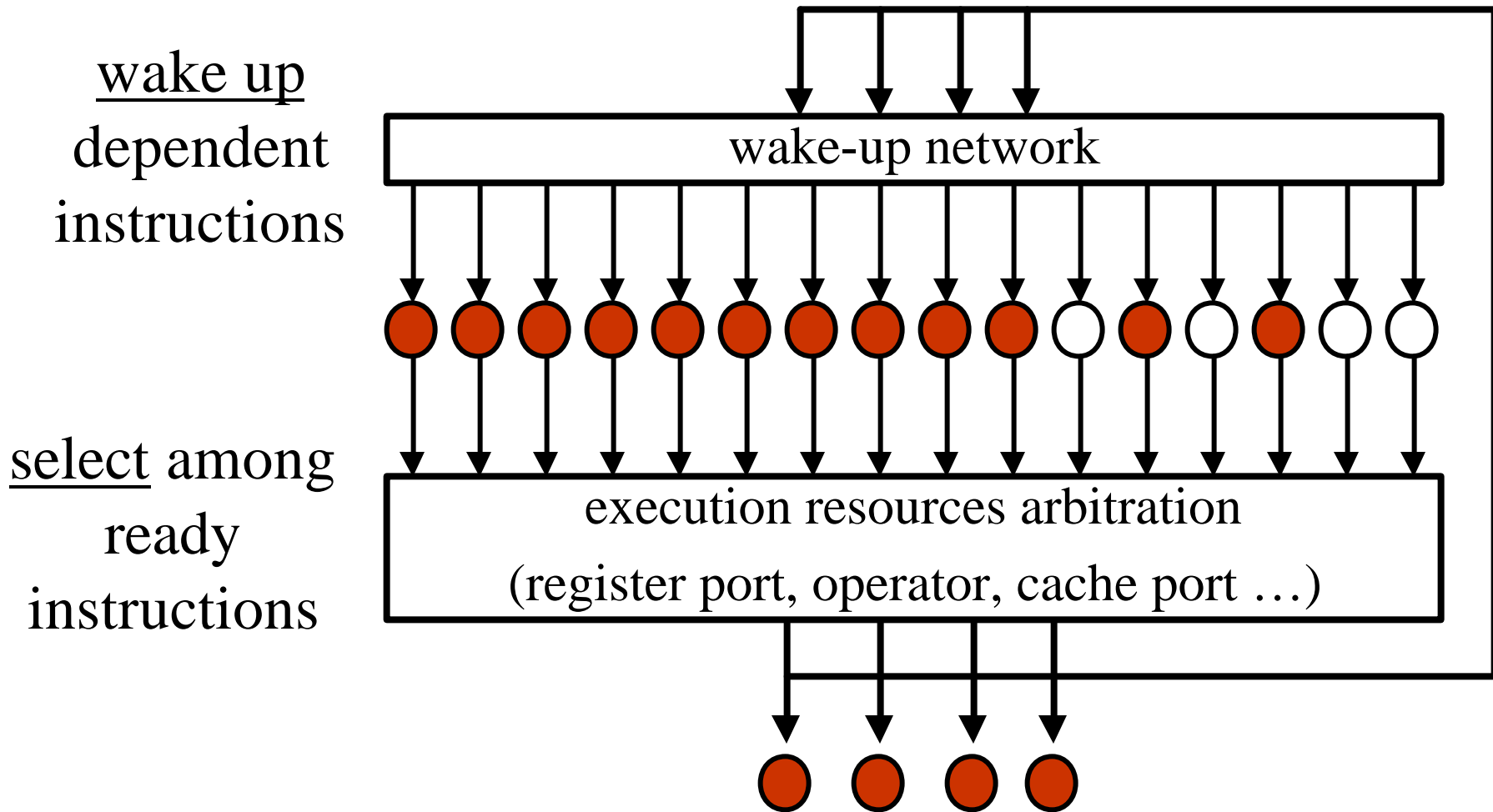
- Experiment
 - let's double the IB as long as issue rate increase > 5 %

load latency LL (cycles)

		1	2	4
issue width IW	4	16	32	64
	8	64	128	256

$$LL \times IW^2$$

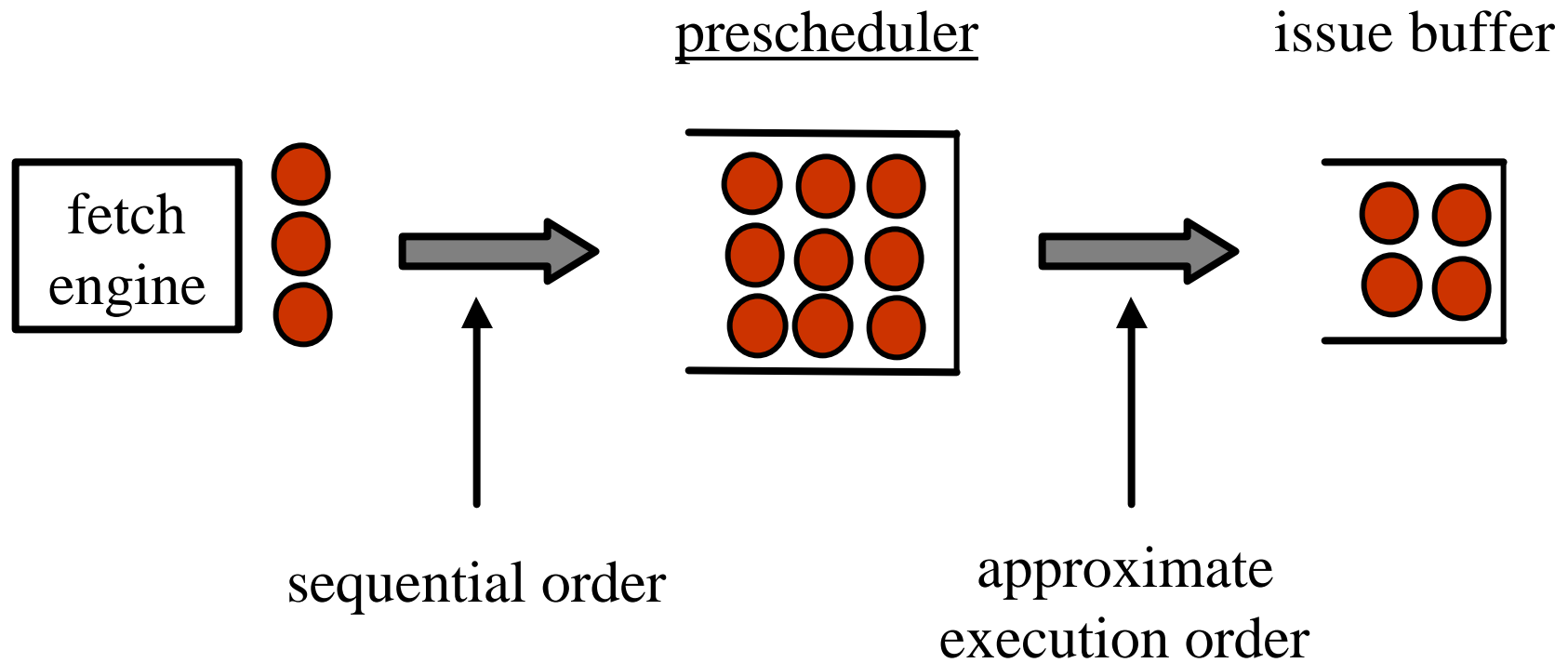
IB size is limited by cycle constraint



Known strategies

- Window partitioning
 - instruction-window quota per execution resource
 - hierarchical wake-up network
 - total window $\propto IW$
- Static prescheduling
 - reordering of instructions by the compiler in order to lessen the impact of data dependencies
 - ISA with many registers

Dynamic prescheduling



Rationale

- Prescheduling looks only at data dependencies
 - scheduling is much simpler without resource constraints
- When preceded by a prescheduler, a small issue buffer is able to saturate execution resources
 - the issue buffer “sees” an instruction flow with sparse data dependencies

Let's be careful ...

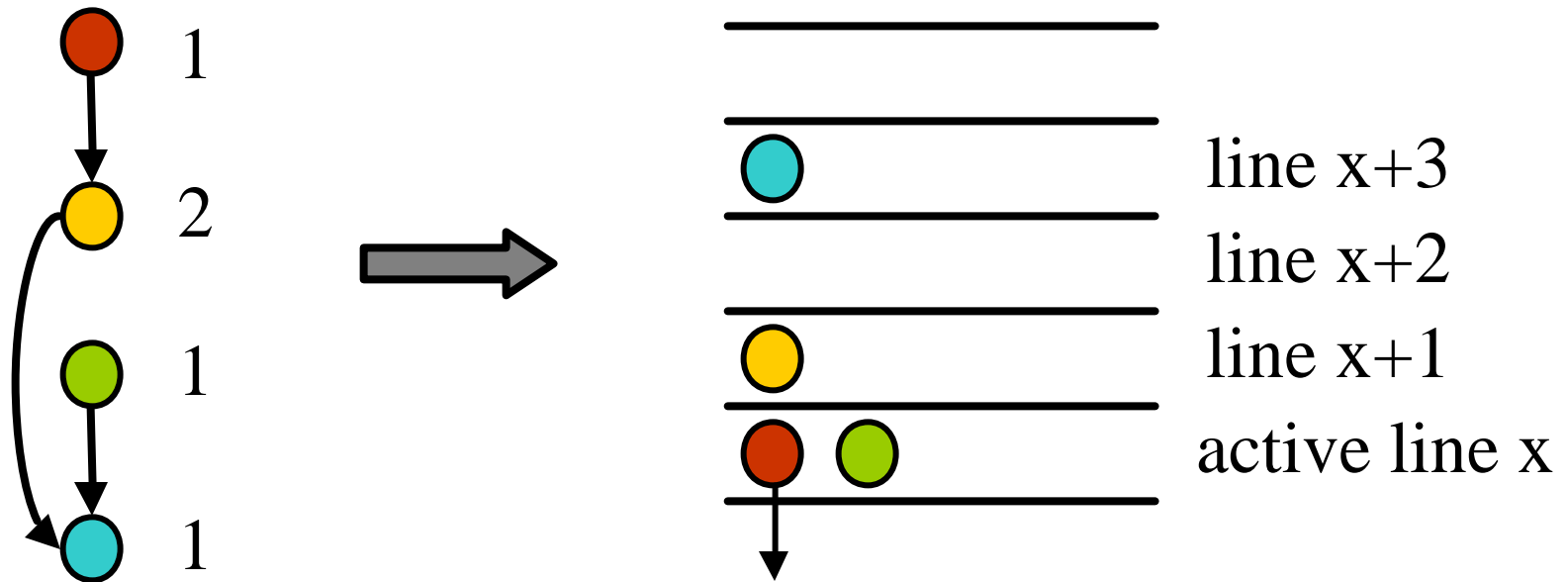
- The predicted schedule needs not be exact, however ...
- Wrong path
 - we don't want execution resources to be saturated with wrong-path instructions
 - there should be a mechanism to prevent the prescheduler from reordering instructions “too much”
- Deadlocks
 - if instruction B is dependent on instruction A, B must enter the issue buffer after A



Method proposed

- For each instruction, compute its depth in the data-flow graph: call it the schedule line
- Store the instruction in the preschedule buffer, using the schedule line number as an identifier
 - each line is associated with a line counter
 - the line counter value can be used as a second identifier
- The issue buffer asks for instructions in line 0, then line 1, line 2, and so on ...
 - the line feeding the issue buffer is the active line
 - once the active line is consumed, the active line number is incremented

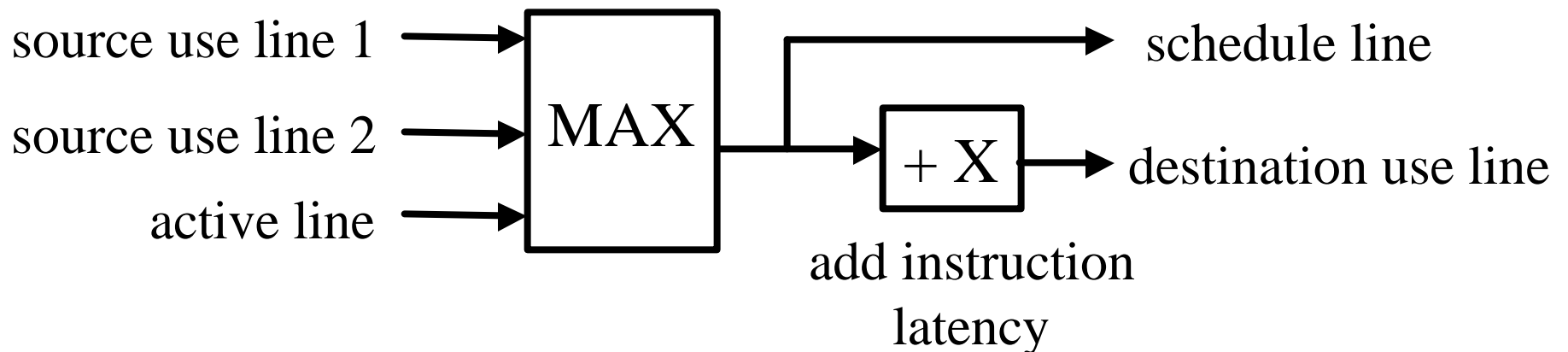
Example



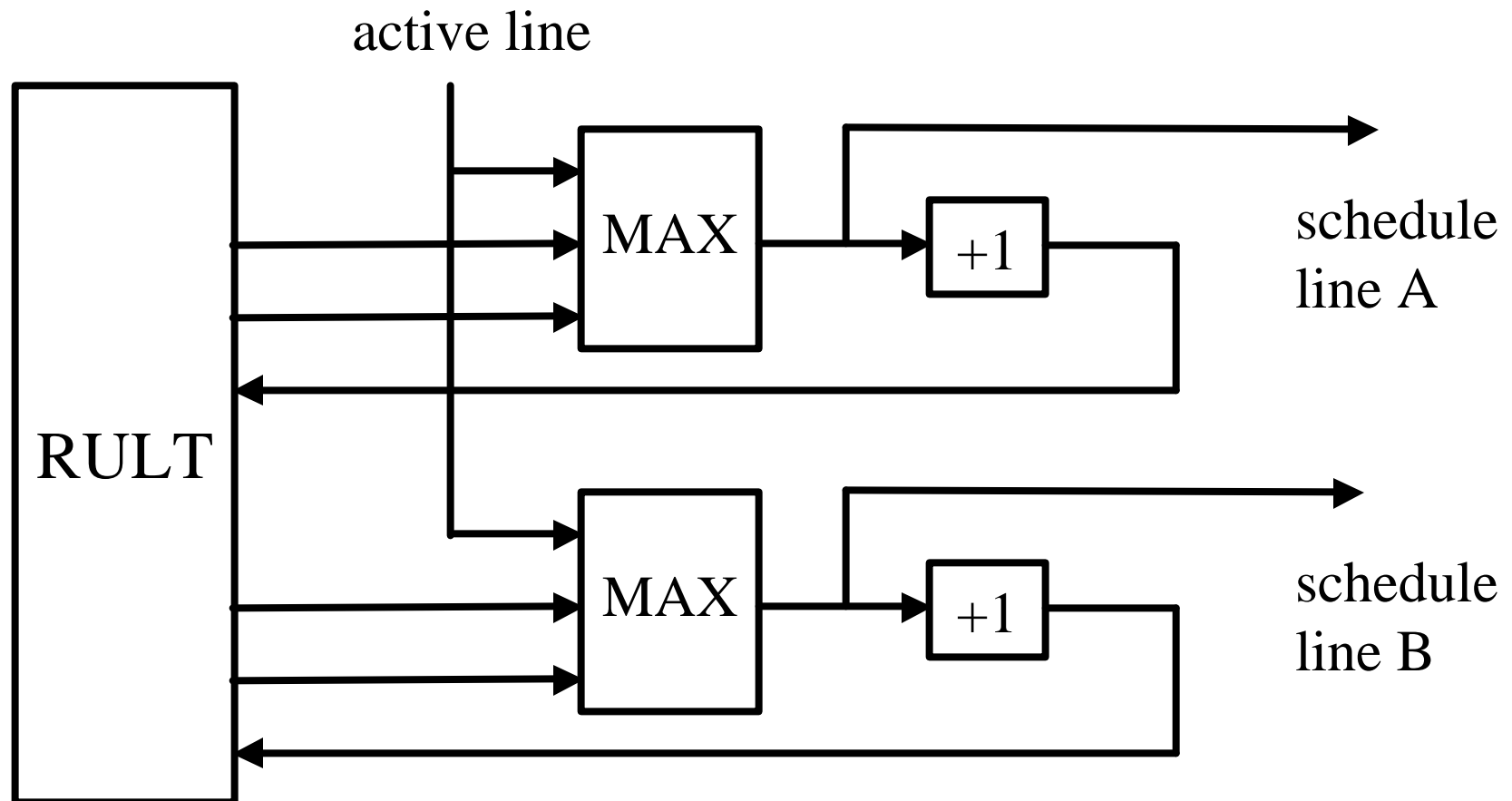
to the issue buffer

Computing the schedule line is simpler than executing the instruction

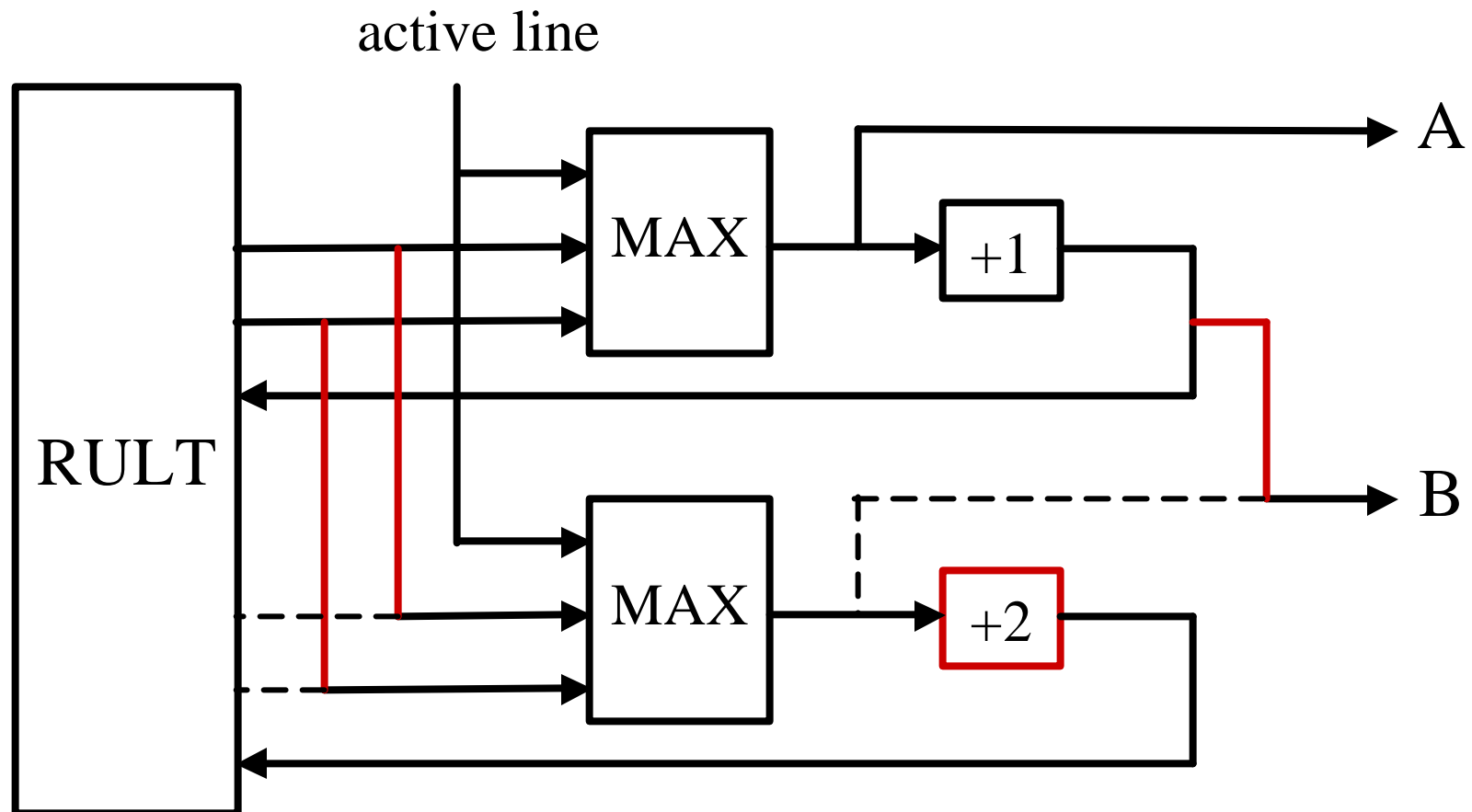
- The preschedule operation is the same for all instructions
- Operands are small line numbers (10 -12 bits)
 - line numbers are stored in a Register Use Line Table (RULT) having one entry per logical register



Case 1: instructions A and B independent

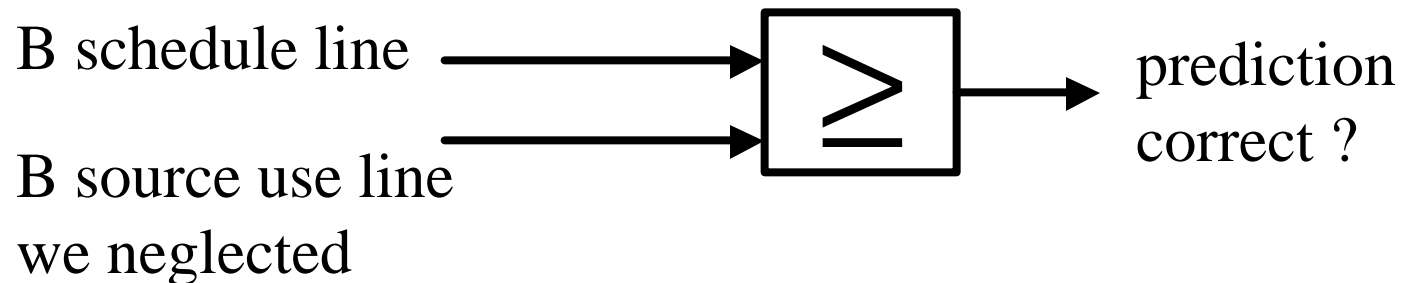


Case 2: B monadic, depends on A



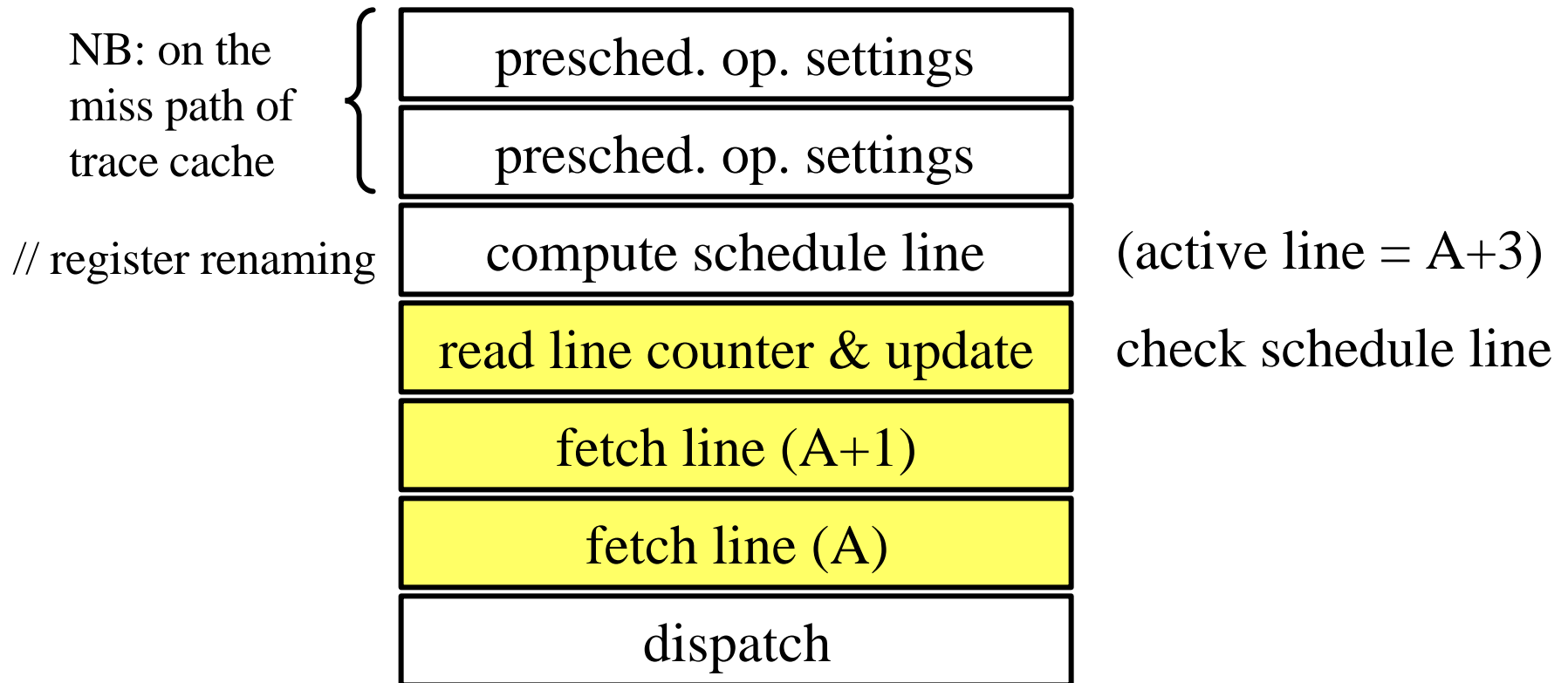
Case 3: B dyadic, depends on A

- Predict which of the two operands can be neglected
 - 2-bit counter per dyadic instruction
 - 100 - 150 instructions / misprediction
- After that, processing of B similar to case 1 or 2
- Verify the prediction once B source use lines are known
 - if wrong, break the instruction group



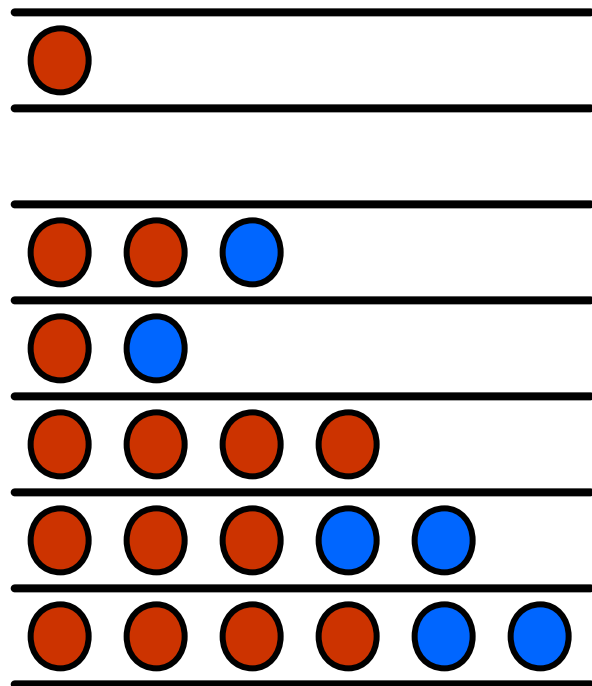
Example pipeline



⋮



⋮

Any mispredicted branch ?



-  right path
-  wrong path



Sometimes it is better to stop filling the preschedule buffer and wait ...

Defining a line size



- We should not allow the number of instructions in the same line to become much larger than the issue width IW
 - wrong-path instructions are not a problem as long as execution resources are not saturated
- Define a line size L
 - when line counter exceeds L , prescheduling is suspended
 - prescheduling resumes when active line $>$ overflowed line
- From experimentation:

$$L \approx 1.5 \times IW$$

3 - 5 % performance
loss from wrong path

Issue rate

- Experiment
 - line $L = 1.5 \times IW$
 - issue buffer $IB = 2 \times IW$
 - issue rate given as fraction of issue width

		load latency		
		1	2	4
IW/ IB/ L	4/8/6	94 %	92 %	86 %
	8/16/12	91 %	88 %	82 %

loss comes from
unregularity of
parallelism

Performance speedup

- Experiment (IBS benchmarks)
 - 2-level predictor: 100-300 instructions per branch misprediction
 - 10+3 cycles branch misprediction penalty
 - D-cache: 2-8 % miss rate / 10-cycle miss penalty

		D-cache hit latency		
		1	2	4
issue width / issue buffer	4 / 8	1.05	1.11	1.26
	8 / 16	1.13	1.21	1.33

Conclusion

- A method for scalable instruction windows
- Interest of the method increases with
 - branch prediction accuracy
 - issue width
 - load latency

Future work and open questions

- D-cache misses degrade the predicted schedule
 - predict cache misses ? enlarge the D-cache ?
- Clustered microarchitectures ?
- Other prescheduling methods ?