Simultaneous Multithreading Processor

Paper presented:
Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor

James Lue

Some slides are modified from http://hassan.shojania.com/pdf/SMT_presentation.pdf

Processors for parallelism

- Super scalar and multithreaded processor: Architecture for increasing parallelism
- Super scalar: Instruction level parallelism
- Multithreaded processor: Thread level parallelism

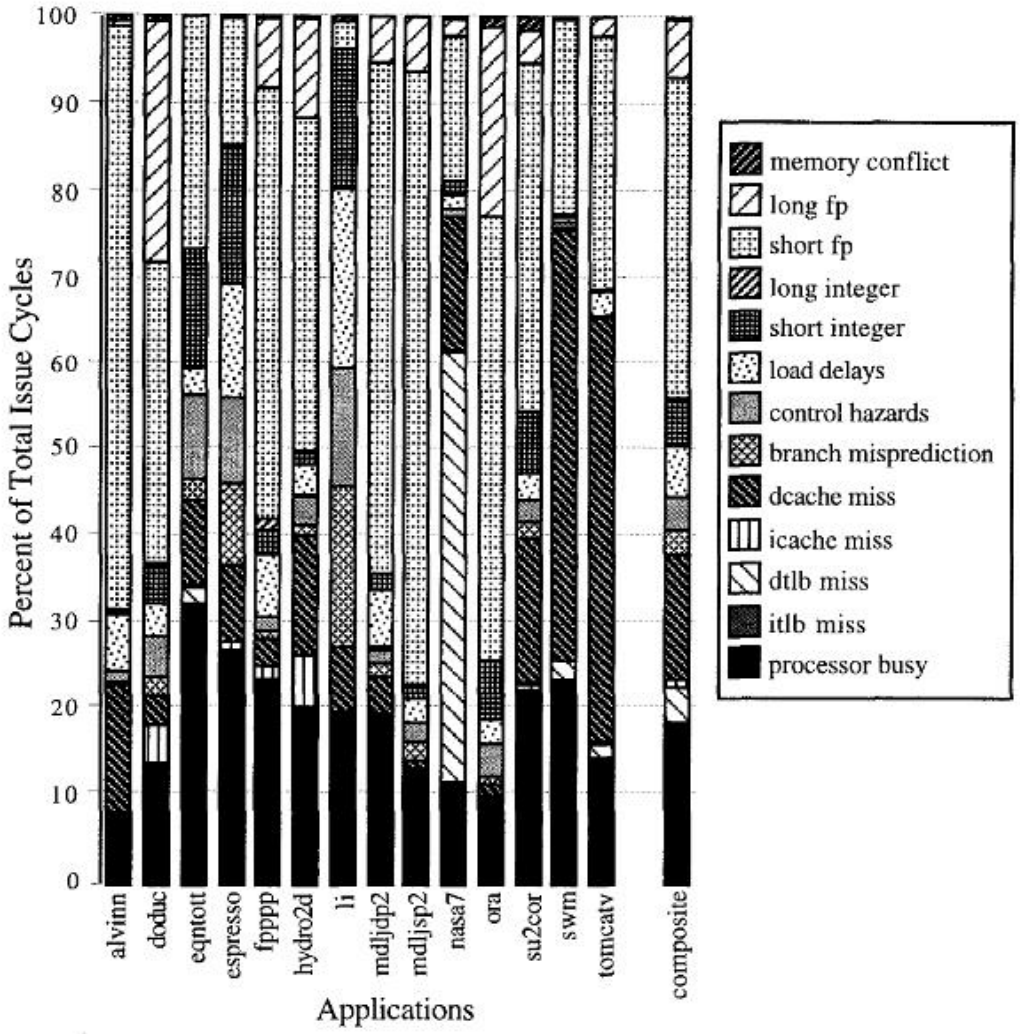
Limits on super scalar

Which parts to improve?

Simulation Result:
1.IPC<1.5
2.No general dominant cause for all applications

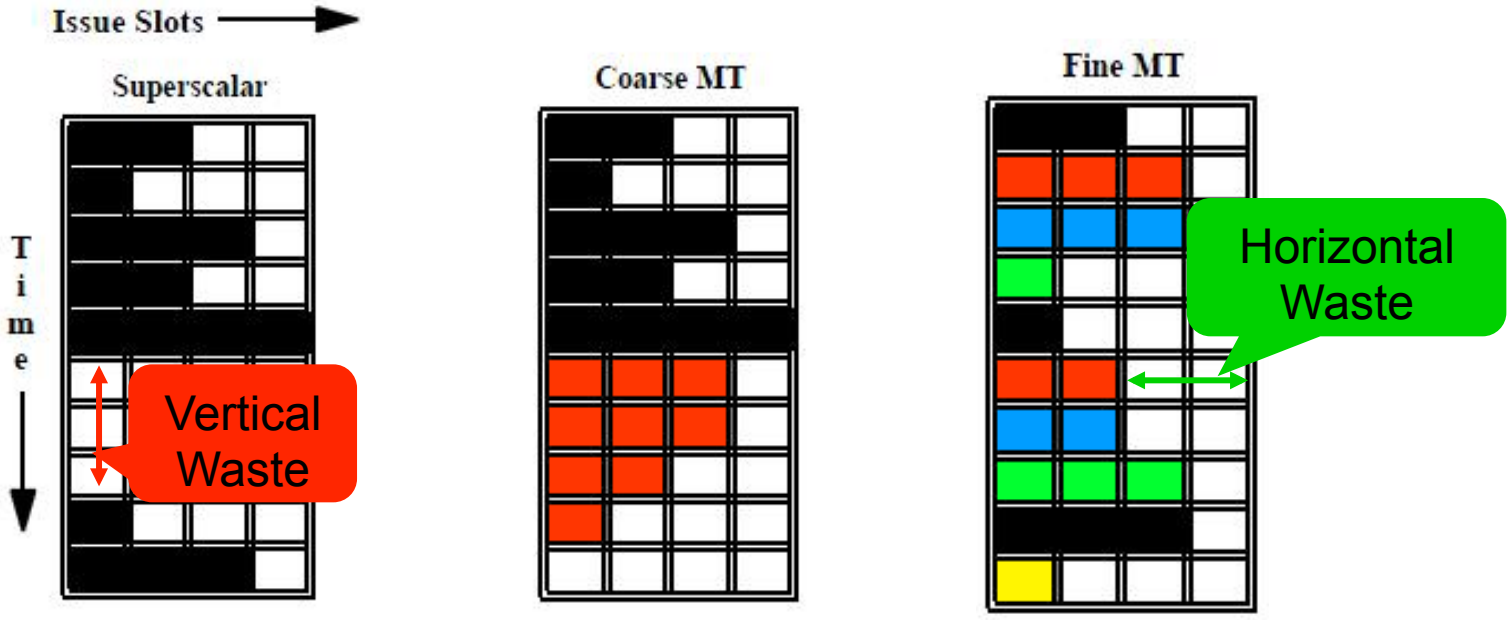
Simulation for unused issue cycles in pure super scalar.

Processor busy: utilized issue slots, other are wasted issue slots.



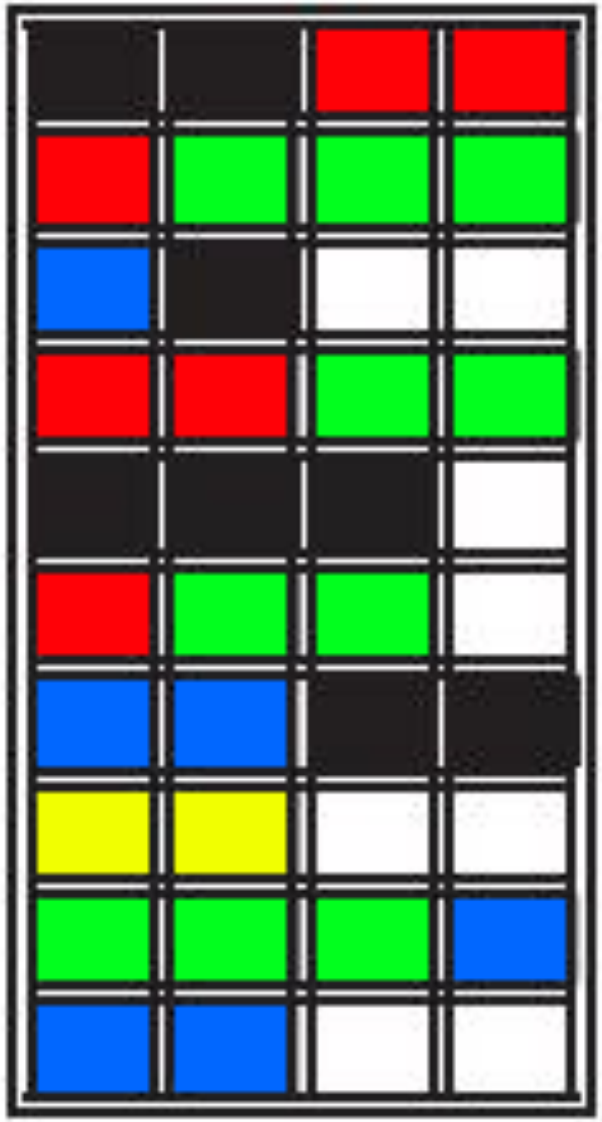
Problem with super scalar and multithreaded processor

Color: Thread



Source: Computer Architecture A Quantitative Approach 4.ed

SMT



- Problem with the experiments results: The model is too **idealized** (huge number of register file ports, complex scheduling,…)

Need more **Realistic** Model !!

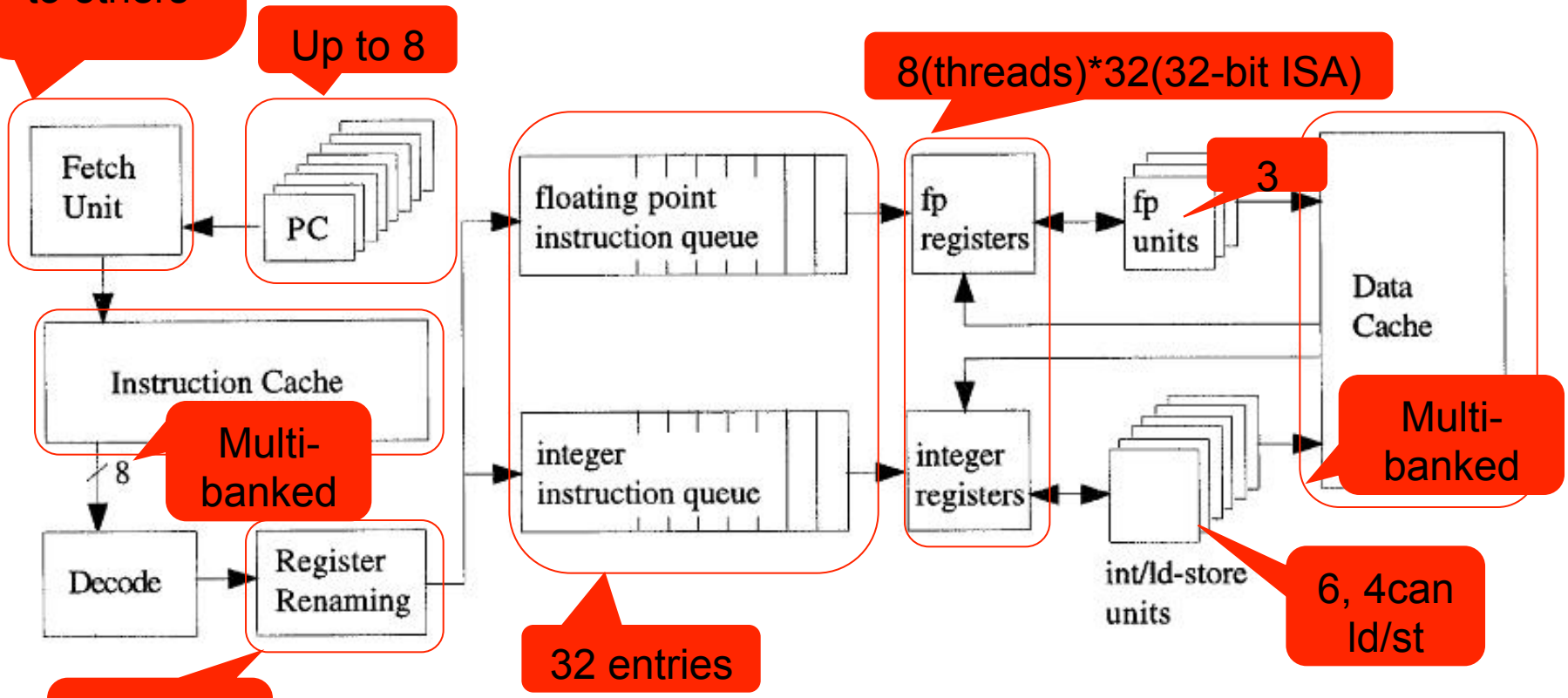
Goal

- Minimize architectural impact on superscalar to build SMT
- Minimal performance impact on single thread applications
- High throughputs (IPC) with multiple threads
- Be able to issue the threads that use processor most efficiently in each cycle.

Base Architecture

Derive from OOO super scalar

Choose the thread that is without I cache miss and conflict to others



Per-thread return stack, instruction retirement, IQ flush, trap mechanism, ID (in IQ,BTB)

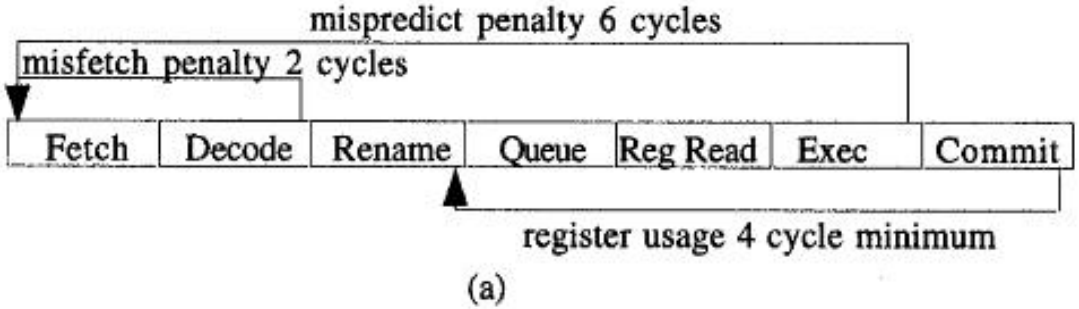
- **Larger register file has some effects on the pipeline:**

  1.Latency for accessing large register file is high => break it into 2 cycles for register access

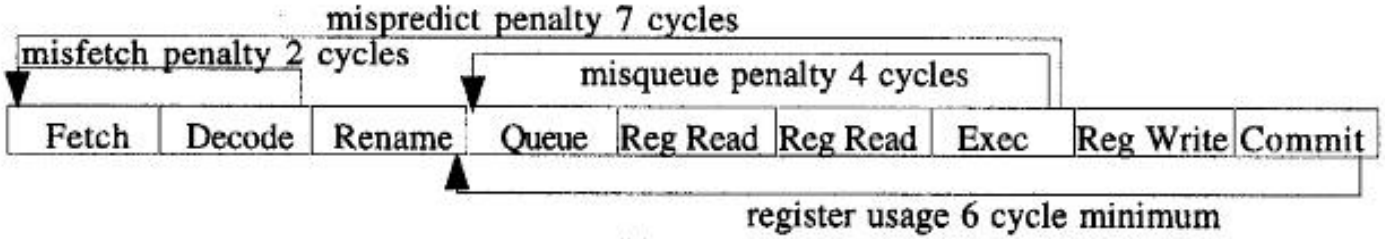
  2. So, 2 pipeline stages (RR,WB) need to access register file => 2 more pipeline stages added in total

  3. Effects of more pipeline stages:

  Increase branch misprediction penalty, one more bypass level for WB, instructions stays in pipeline for longer time ⇨ more instructions in the pipeline ⇨ registers may be not enough for more instructions



(a)

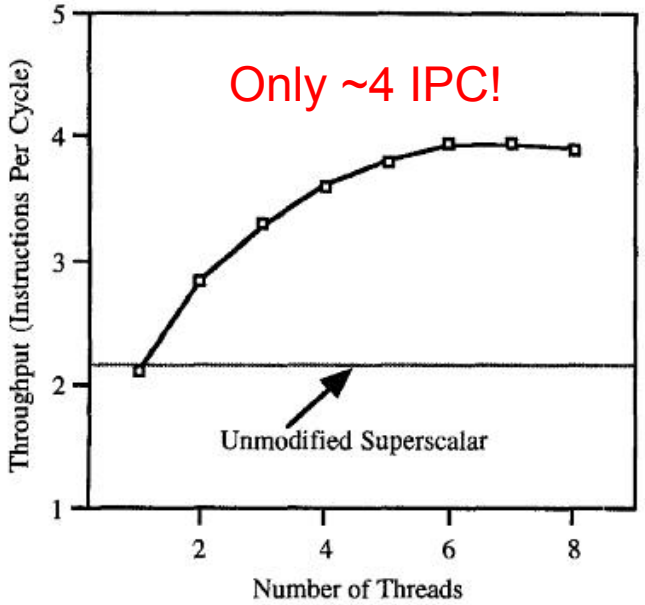


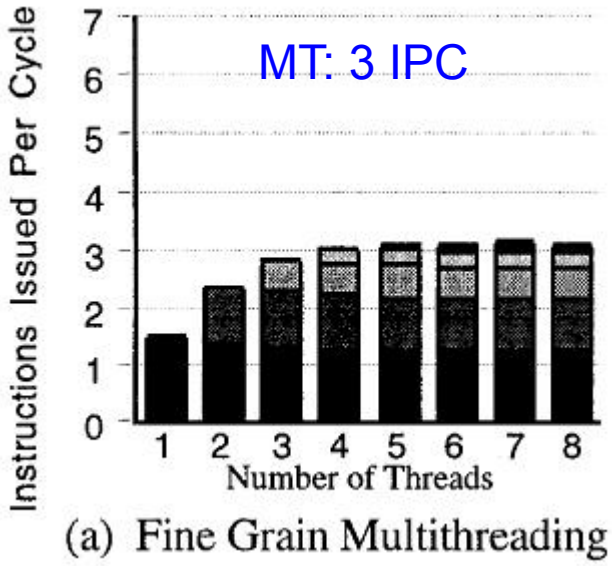
(b)

Increasing pipeline stages doesn’ t increase inter-instructions latency between instructions except “load”

Optimistic issue:
Load instruction needs 2 cycles but they still assume one cycle to load (do not wait 2 cycles for load):
When load suffer cache miss or bank conflict => squash this load, re-issue later

Performance of the base architecture



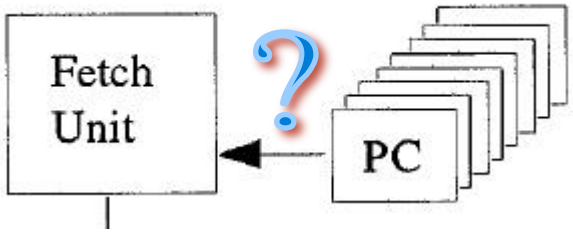


(a) Fine Grain Multithreading

Improve base architecture

Improve fetching

1. Partition the Fetch Unit :



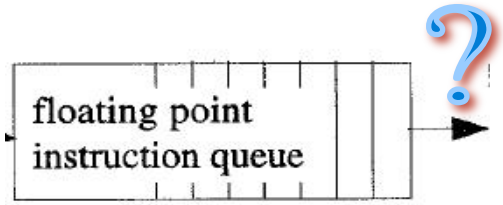
   fetch instructions from multiple threads.

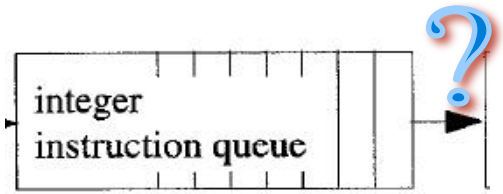
2. Fetch policy: fetch useful instructions

3. Unblocking the Fetch Unit: prevent conditions that make fetch unit stop

Improve issuing

1. Issuing policy

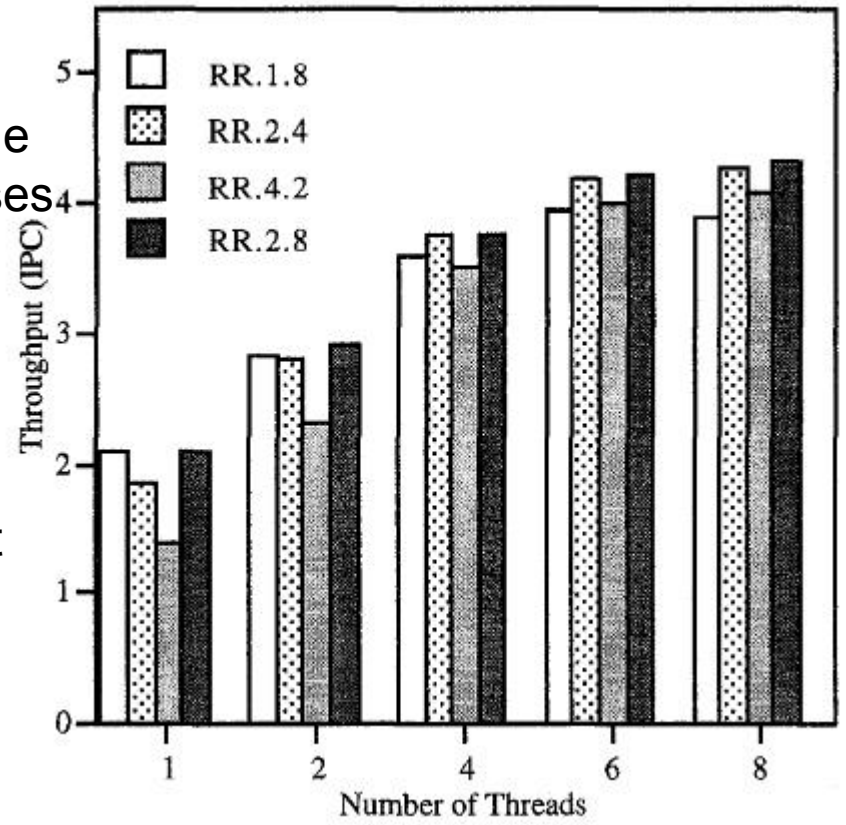




Partition the Fetch Unit

Alg. # threads to fetch in a cycle. Max # instr. per thread in a cycle

RR.1.8: -baseline (MT)
RR.2.4: -2 cache output buses, each 4 instr. wide
-additional circuit for multiple output buses
RR.4.2: -more expensive changes than RR.2.4
-suffer from thread shortage (available threads are less than required to fill fetch bandwidth)
RR.2.8: -Best
-fetch as many instr. As possible for first thread then second thread fills up to 8
-high throughput-> cause IQ full



Fetch Policy:

Factors to consider:

1.Don’ t fetch threads that follow wrong paths (misprediction)

2.Don’ t fetch the threads that stay in IQ for very long time

Policy:

BRCOUNT:

favor branches with fewest unresolved branches (1st factor)

MISSCOUNT:

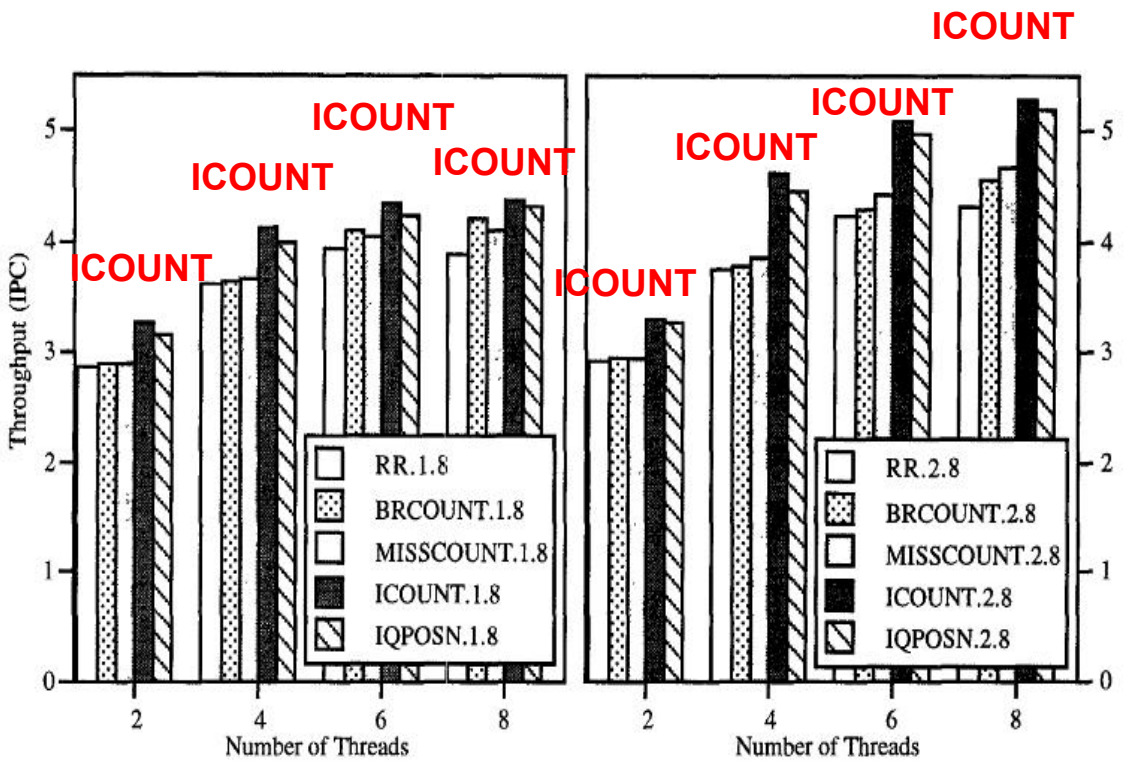
Favor threads with fewest outstanding D cache miss (2nd factor)

ICOUNT:

Favor threads with fewest instruction in decode, rename, IQ (2nd factor)

IQPOSN:

Don’ t favor threads that have old instructions in IQ (2nd factor)



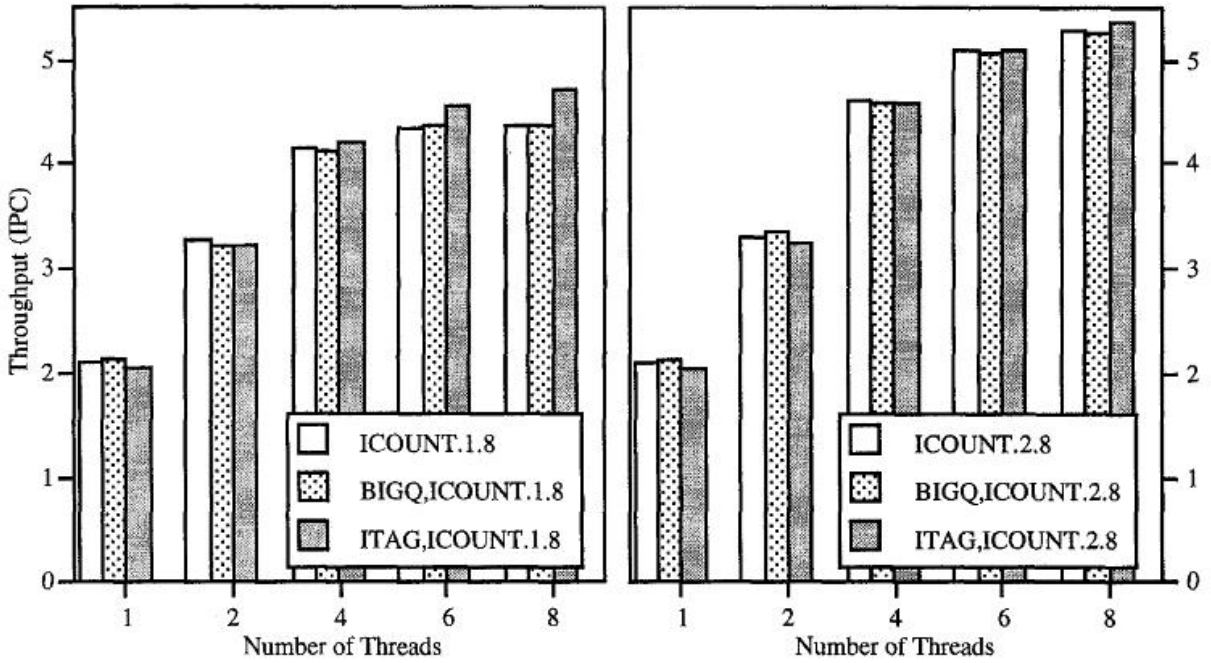
Unblocking Fetching Unit

Conditions that cause the fetch unit to be blocked:

1. IQ full
2. I cache miss

Scheme to solve the conditions:

1. BIGQ: restriction on IQ: searching time
   Thus, double the IQ size but only search first 32 instructions in the IQ. Additional space in IQ is for buffering instruction when IQ overflow.
2. ITAG: look up I cache tag one cycle early, if it' s miss, choose other thread and load the missing instruction into cache(fetch after the instruction is in the cache). Need more pipeline stages.



Which Instruction to issue?

Issue priority: aim to avoid wasting issue slots

Causes for issue slot waste:

1. Wrong path instruction
2. Optimistically (issue load-dependant instructions a cycle before have D cache hit information, if miss or bank conflict, squash opt issued instruction, waste slot)

OLDEST_FIRST: deepest instructions in IQ first (default)

OPT_LAST: issue Optimistic instructions as late as possible

SPCE_LAST: issue speculative instructions as late as possible

BRANCH_FIRST: issue branch instructions as soon as possible
(to identify mispredicted branch quickly)

Issue Method	Number of Threads					Useless Instructions	
	1	2	4	6	8	wrong-path	optimistic
OLDEST	2.10	3.30	4.62	5.09	5.29	4%	3%
OPT_LAST	2.07	3.30	4.59	5.09	5.29	4%	2%
SPEC_LAST	2.10	3.31	4.59	5.09	5.29	4%	3%
BRANCH_FIRST	2.07	3.29	4.58	5.08	5.28	4%	6%

Need multiple passes to search IQ

OLDEST

No much difference

Bottlenecks

- **Issue BW:**
- Infinite FUs, throughput increases 0.5%
- **IQ size:**
- Reduced by ICOUNT; increases1% with 64-entry IQ
- **Fetch BW:**
  Fetch up to 16 instr. ⇨ 5.7 IPC (increase 8%)
- Fetch up tp 16 instr., 64-entry IQ, 140 regs. ⇨ 6.1 IPC

Bottleneck!

- **Branch prediction:**
- Doubling BHT,BTB size, increases 2%

  Speculative execution:

  Wrong path rare, mechanisms to prevent speculative execution reduce throughput
- **Memory/cache:**
- No bank conflict (infinite BW cache) increases 3%
- **Register file:**
- Infinite excess register increase 2%
- ICOUNT reduces clog/strain on Reg. file

Summary

No heavily change to conventional superscalar

No great impact on single-thread performance

Achieve 5.4 IPC, 2.5x compare to 8 threads super scalar

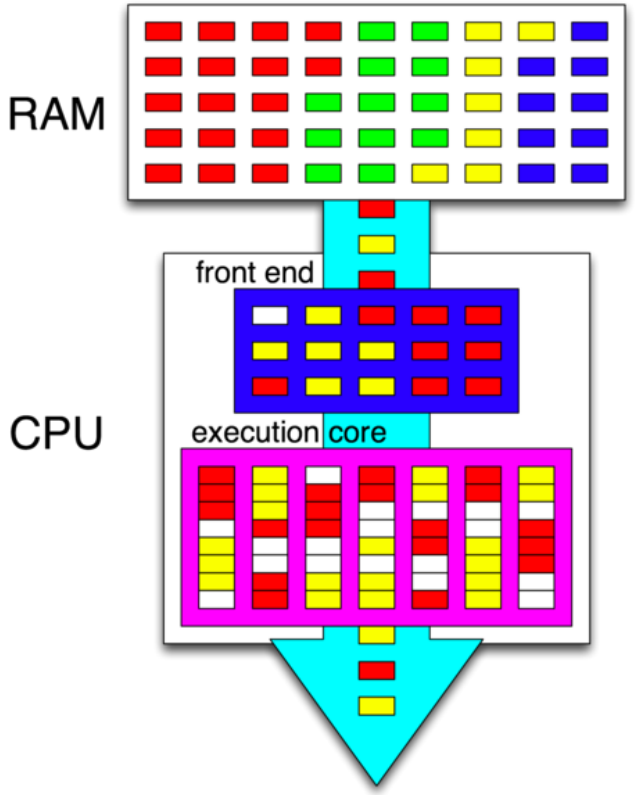
These improvements are from:

-1.Partition fetch bandwidth

-2.Be able to select the threads that uses resources most efficiently

SMT implemented processors

- Hyper-Threading in Intel
- Implemented in Atom, Intel Core i3/i7, Itanium, Pentium 4 and Xeon
- 2 threads per core
- Need OS support and optimization



http://en.wikipedia.org/wiki/Hyper-threading

SMT implemented processors- Hyper Threading Architecture

Figure 2: Processors without Hyper-Threading Tech

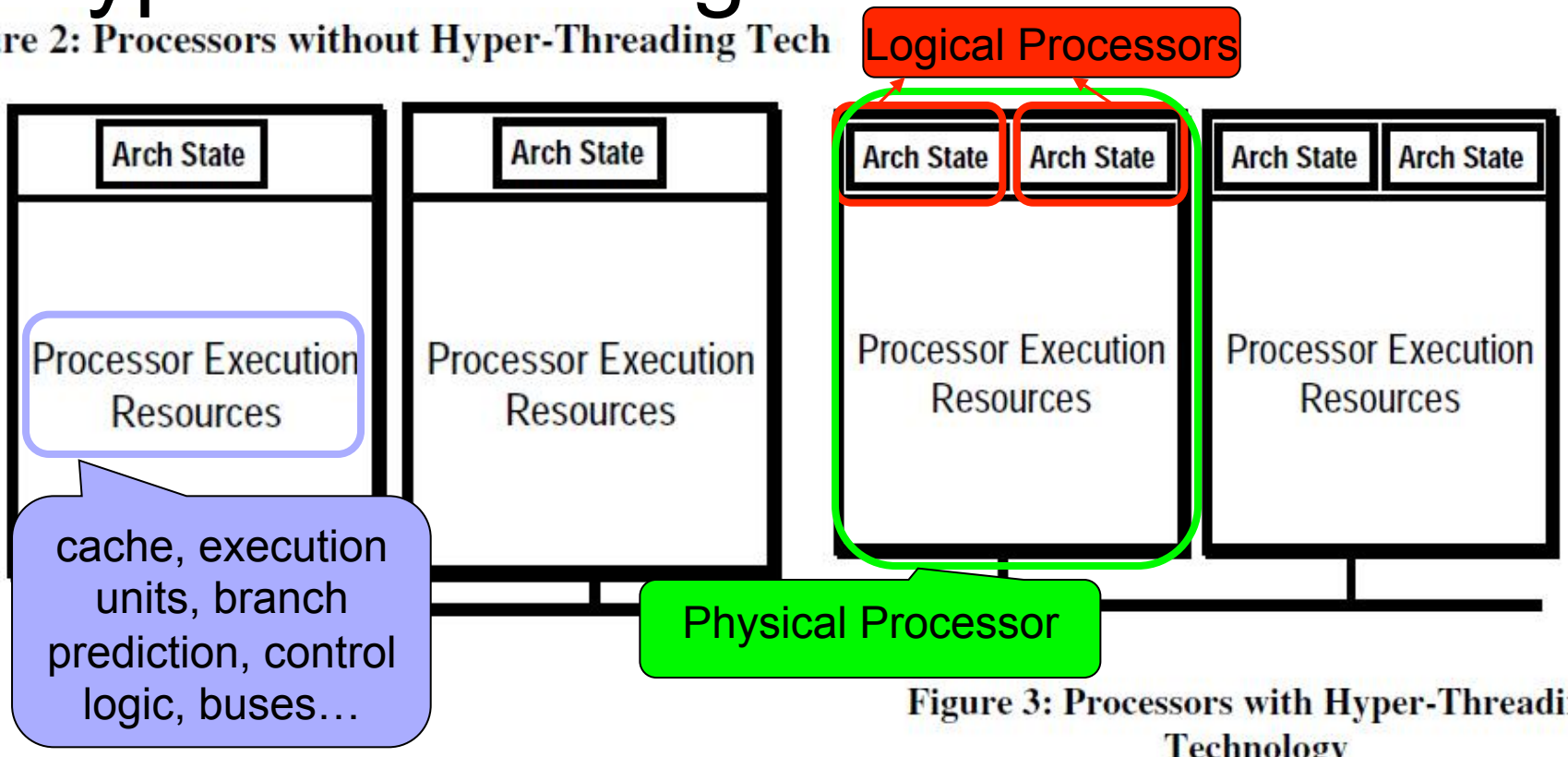
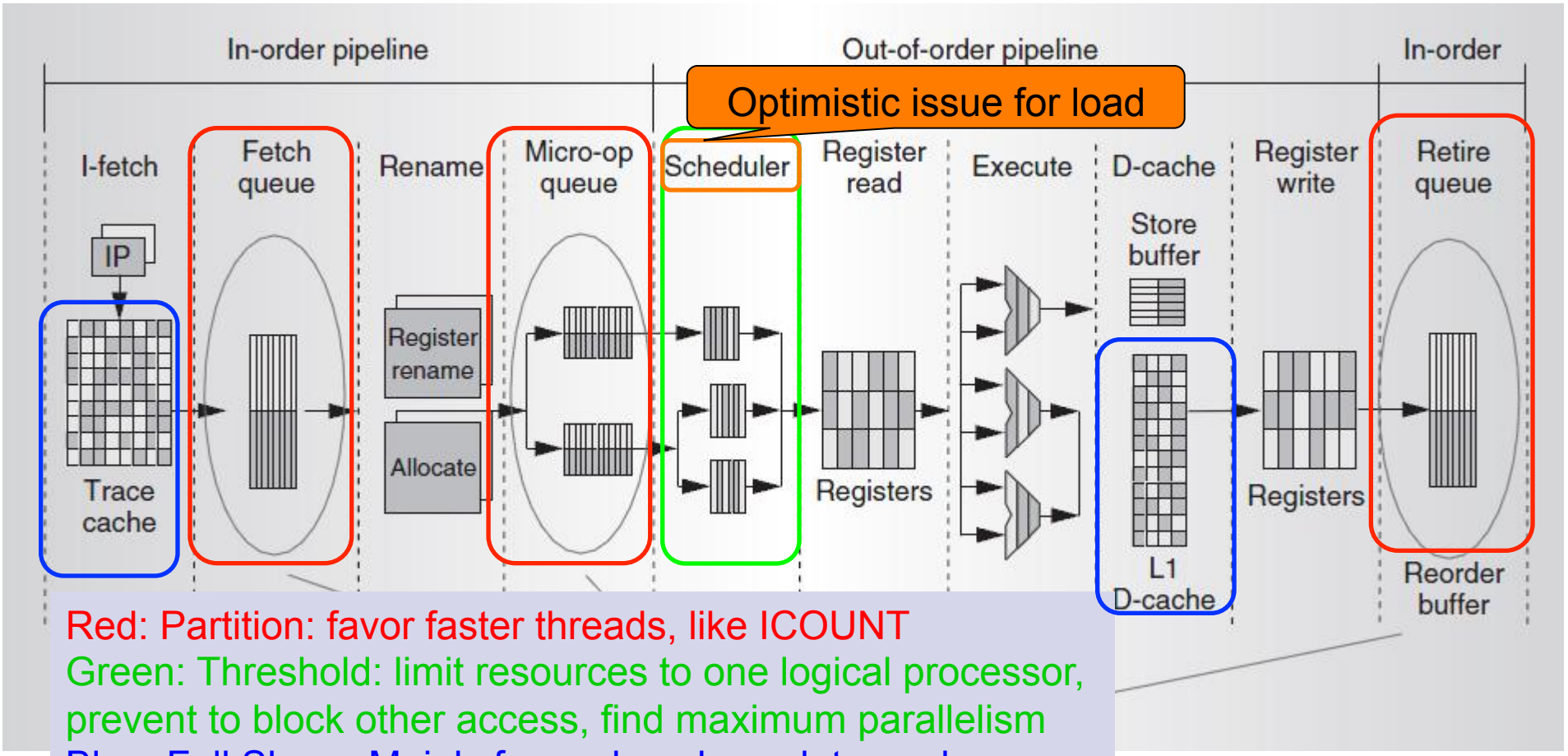


Figure 3: Processors with Hyper-Threading Technology

Source: Intel Technology Journal. Volume 6 Issue 1. February 14, 2002

SMT implemented processors

- Netburst microarchitecture(Pentium 4 and Pentium D)



Source: HYPERTHREADING TECHNOLOGY IN THE NETBURST MICROARCHITECTURE

Thank you