Kilo-instructions Processors

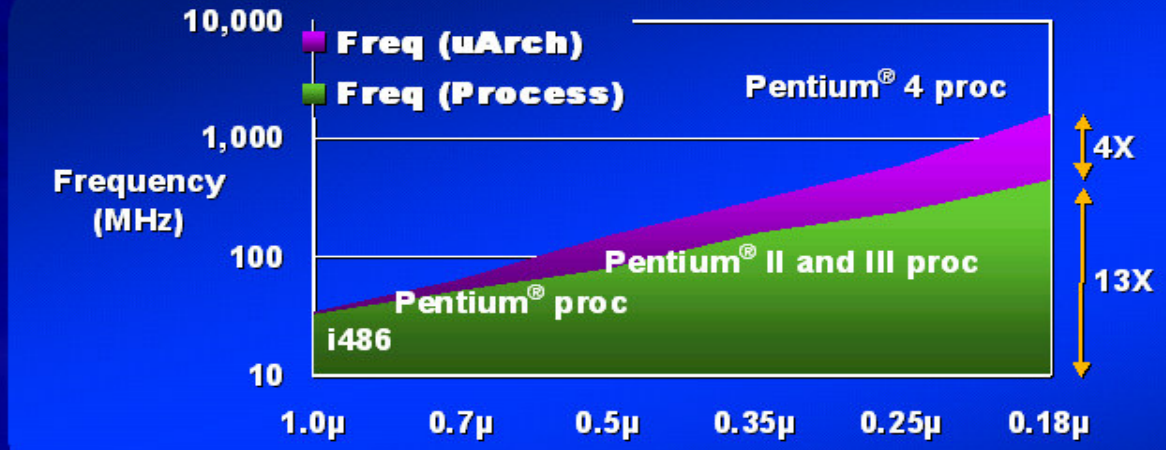
Speaker: Mateo Valero, UPC

Team: Adrián Cristal, Pepe Martínez, Josep Llosa, Daniel Ortega and Mateo Valero

IBM Seminar on Compilers and Architecture

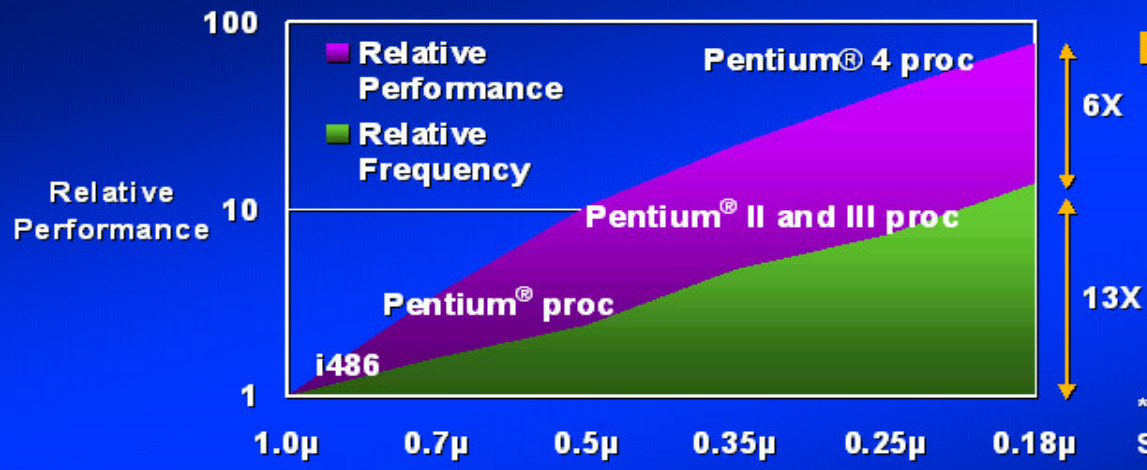
Haifa November 11th. 2003

Frequency and Performance Advances



Frequency Increased 50X

- 13X due to process technology
- Additional 4X due to microarchitecture



Performance Increased >75X

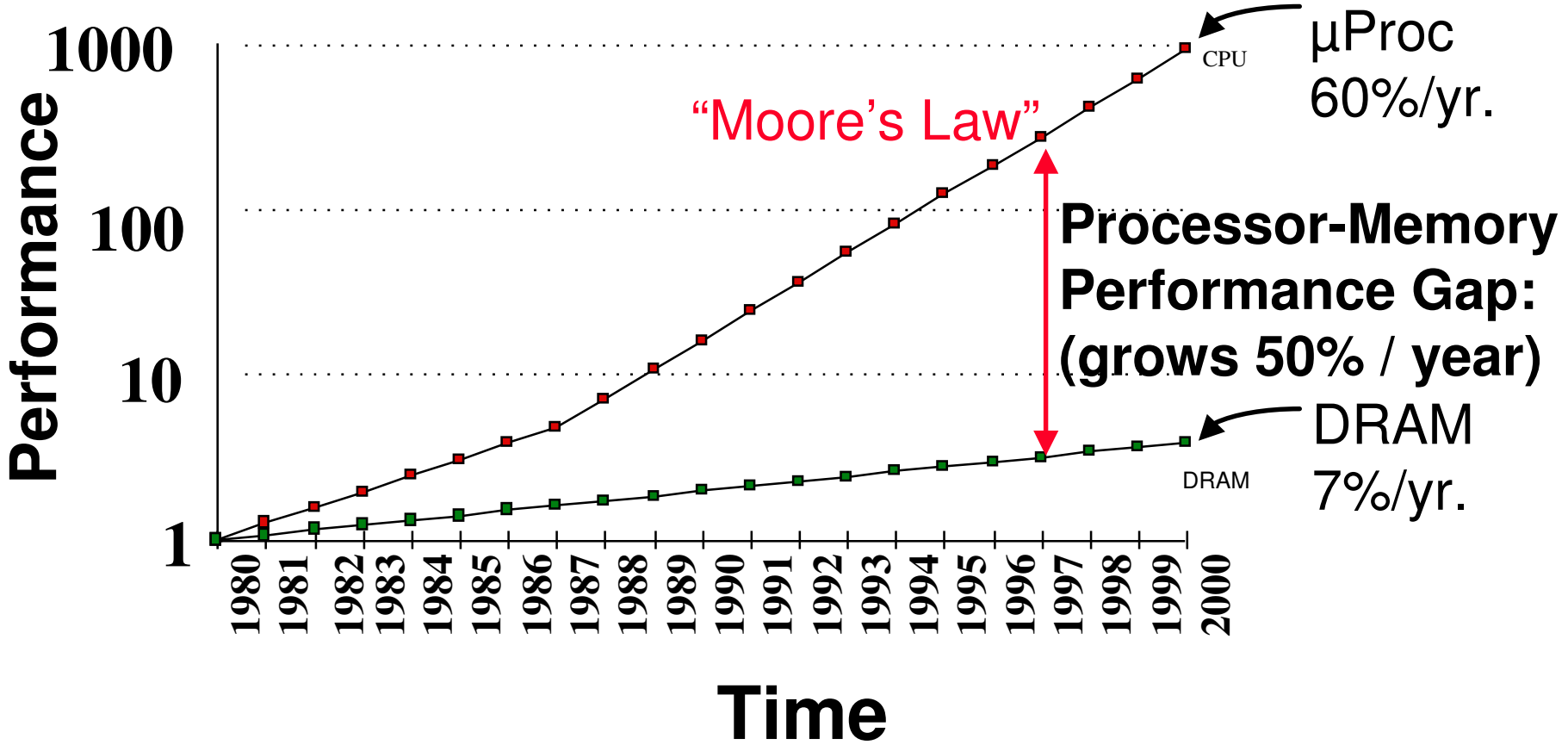
- 13X due to frequency
- Additional >6X due to microarchitecture and design

*Note: Performance measured using SpecINT and SpecFP

❒Technology works against ILP: Faster clock rates => Lower ILP

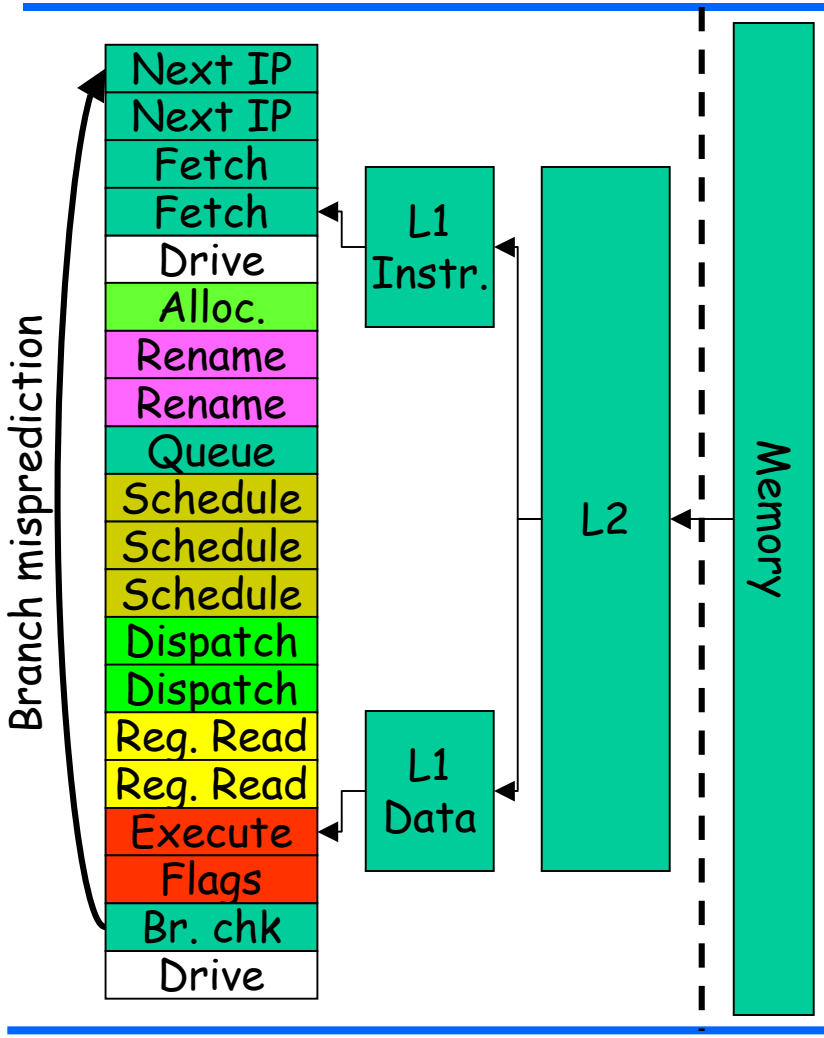
UPC

Processor-DRAM Gap (latency)



D.A. Patterson " New directions in Computer Architecture" Berkeley, June 1998

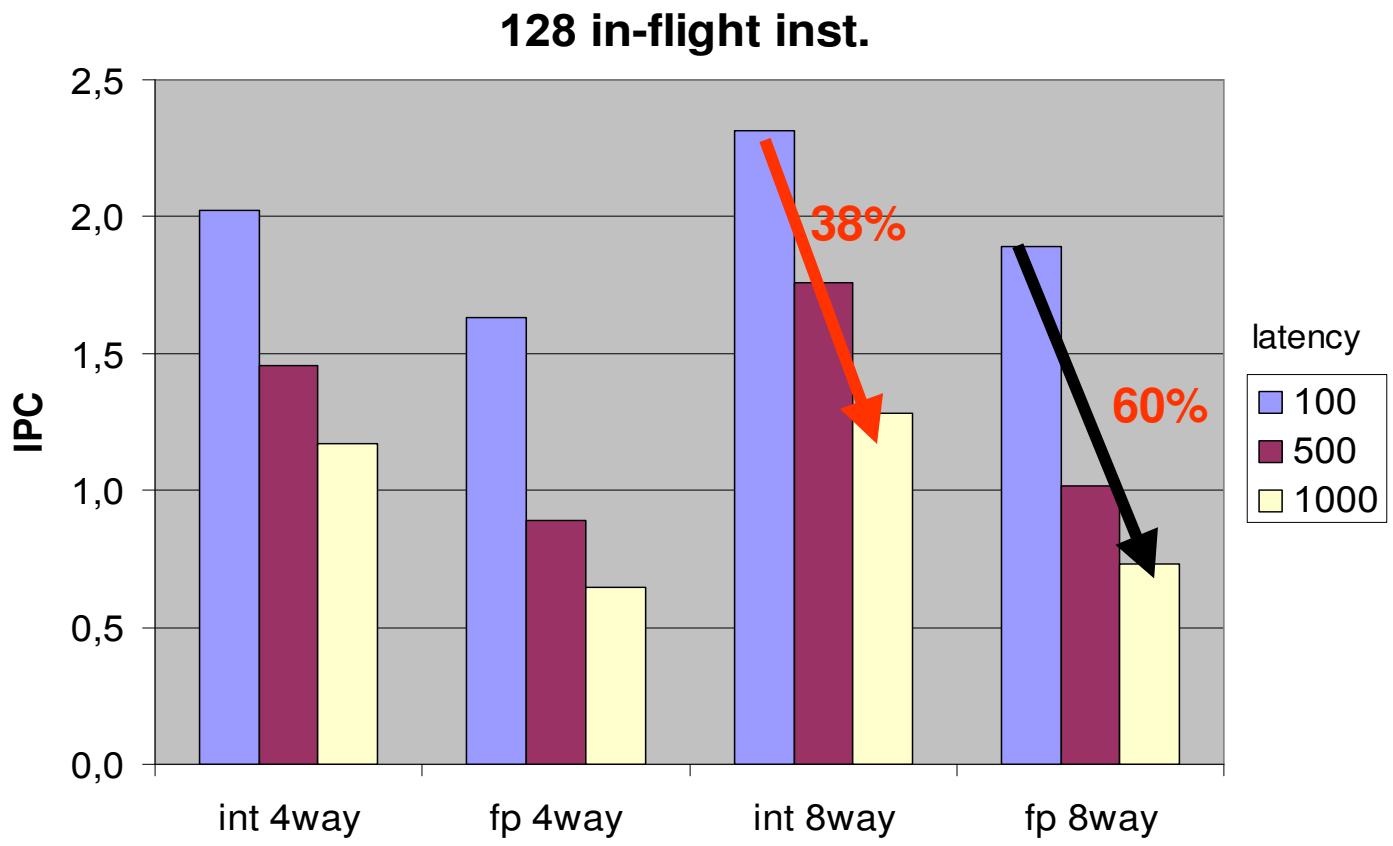
The situation has changed



- 1990's architecture
  - Low memory latencies
  - Single cycle caches
  - Short pipelines
  - Full bypass networks
- 2010+ architectures
  - Long memory latencies
    - 500 to 1000 cycles
    - ISCA-2003: 50 to 160
  - Multi-cycle caches
    - 3 to 10 cycles
  - Long pipelines
    - 20-50 stages
  - Clustered bypass networks
    - Multi-cycle bypas

UPC

Memory Latency and IPC



Memory latency has enormous impact on IPC

Reducing Memory Latency

- Caches
- Prefetching:
  - Hardware, Software and combined
- Helper Threads

- Kilo-instructions Processors

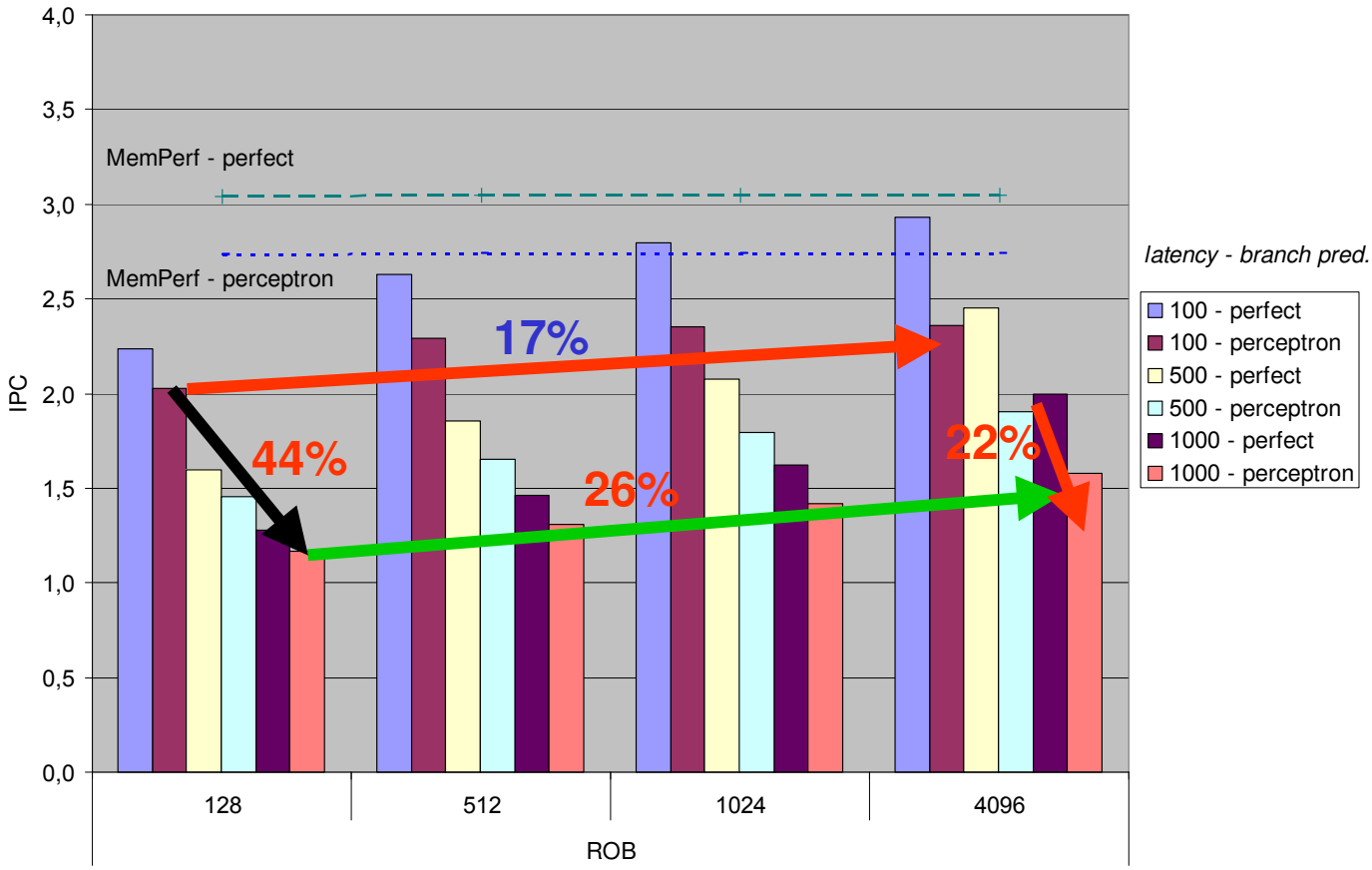
Outline

- Motivation
- Kilo-instruction Processors
  - Resource Utilization
  - Checkpointing
  - Out-of-order Commit Processors
  - Ephemeral Registers
  - Instruction Queues
- Performance Evaluation
- Related Work
- Conclusion

UPC

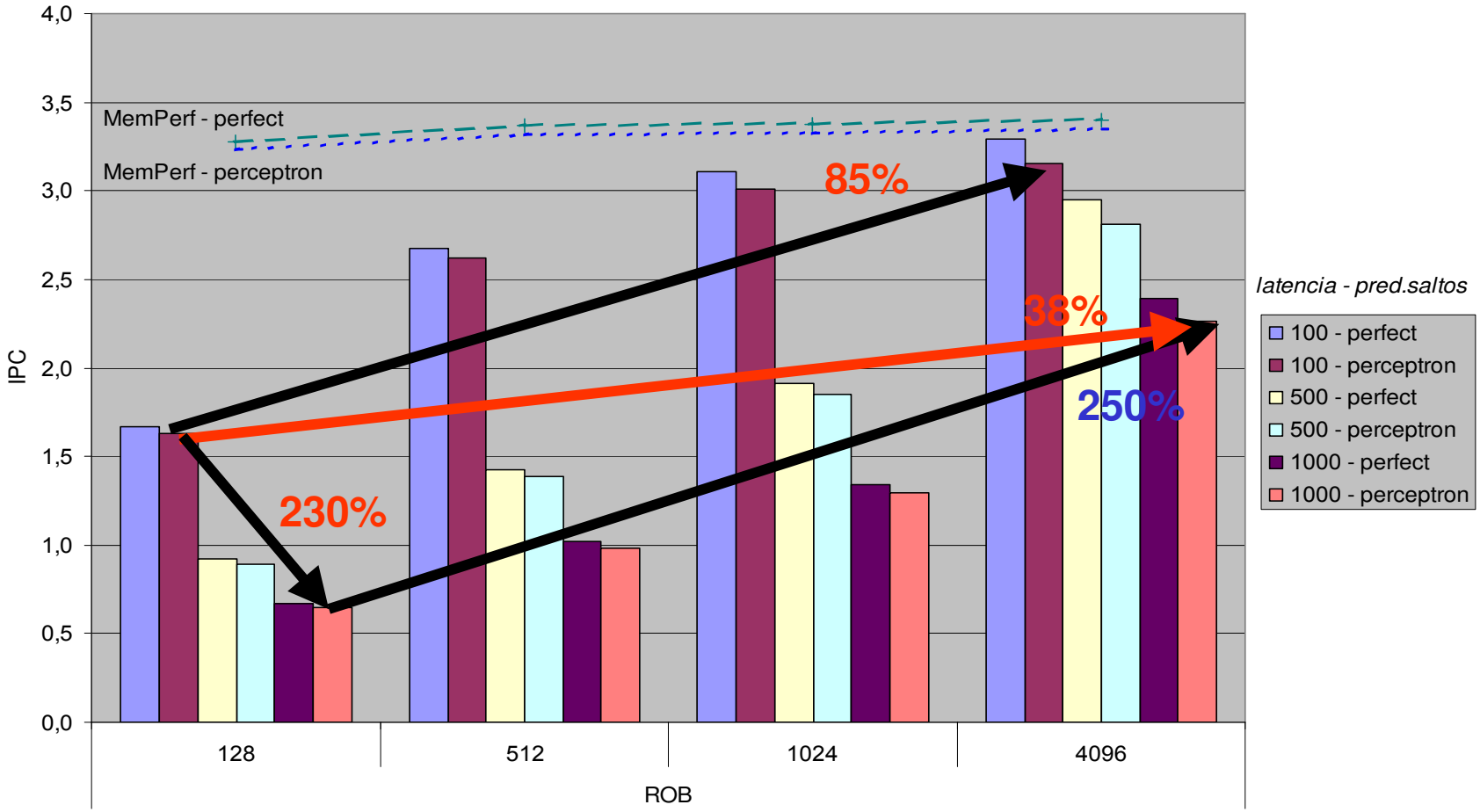
Kilo-instructions processors

- Motivation
  - 8 Specint
  - 4 Way SS
  - 32 KB L1
  - 256 KB L2
  - 16 K entries Gshare BP
  - 500 cycles memory latency



Integer - Near perfect branch prediction is necessary

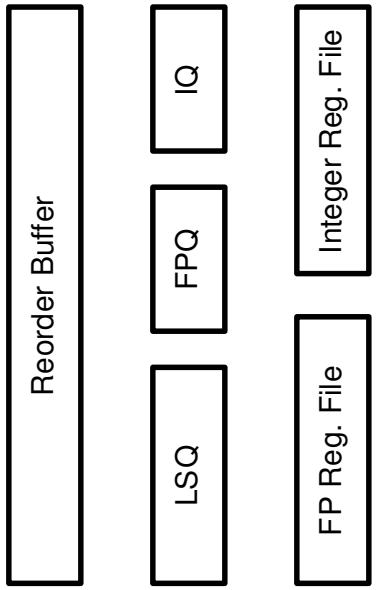
Kilo-instructions processors: specfp2000



Floating Point - Large ROB tolerates latencies better

Scalability

- Thousands of In-flight Instructions and In-Order Commit make designs impractical:
  - ROB : Needs to maintain a copy of every in-flight instruction
  - IQs : Instructions depending on long latency instructions remain in these queues for a long time
  - LSQs : Instructions remain in the queue until commit
  - Registers : A new physical register for each instruction producing a new value
- We would like to get the IPC of thousands of instructions in-flight with the resources required by few of them



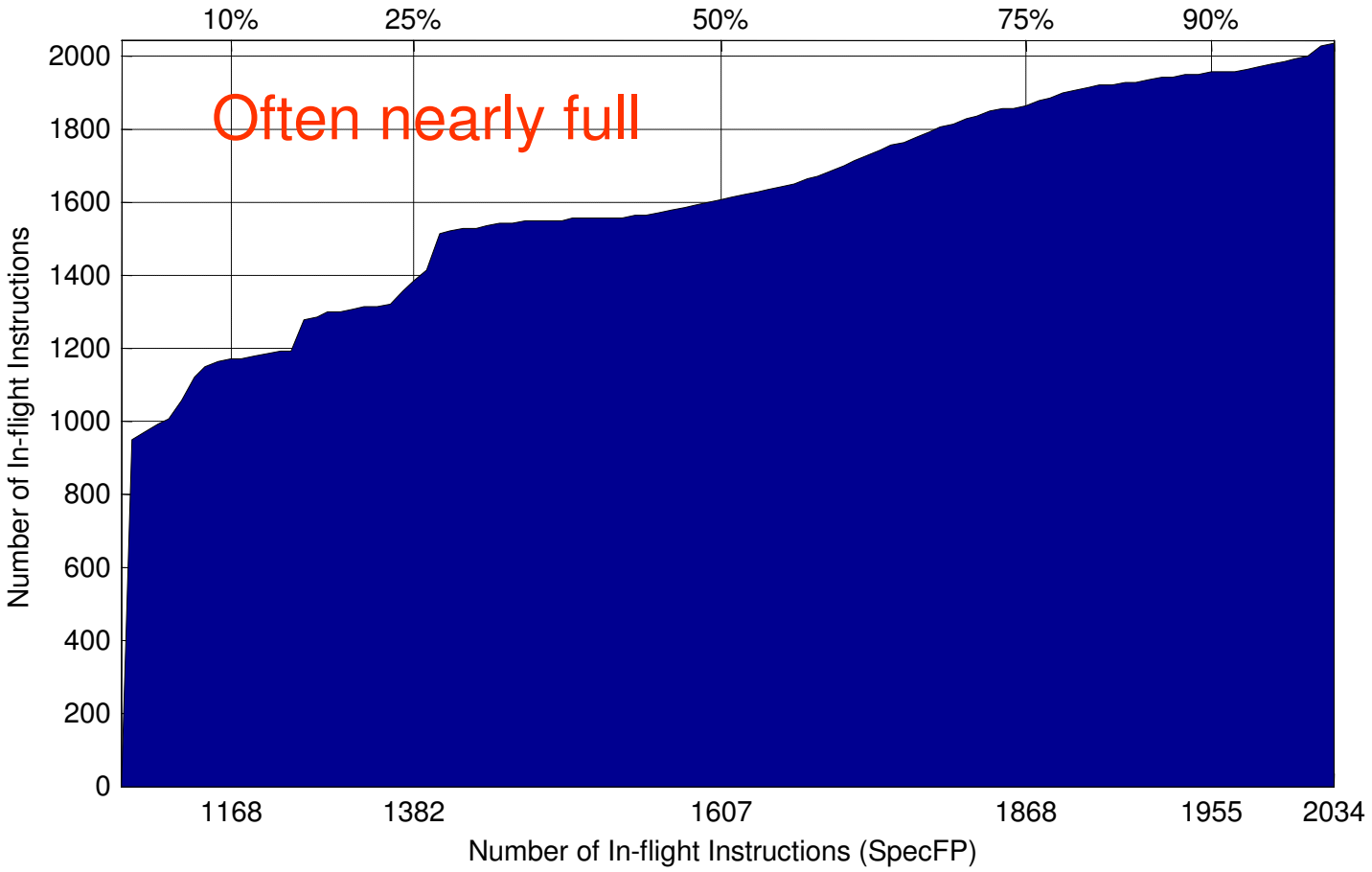
UPC

Utilization of the Resources

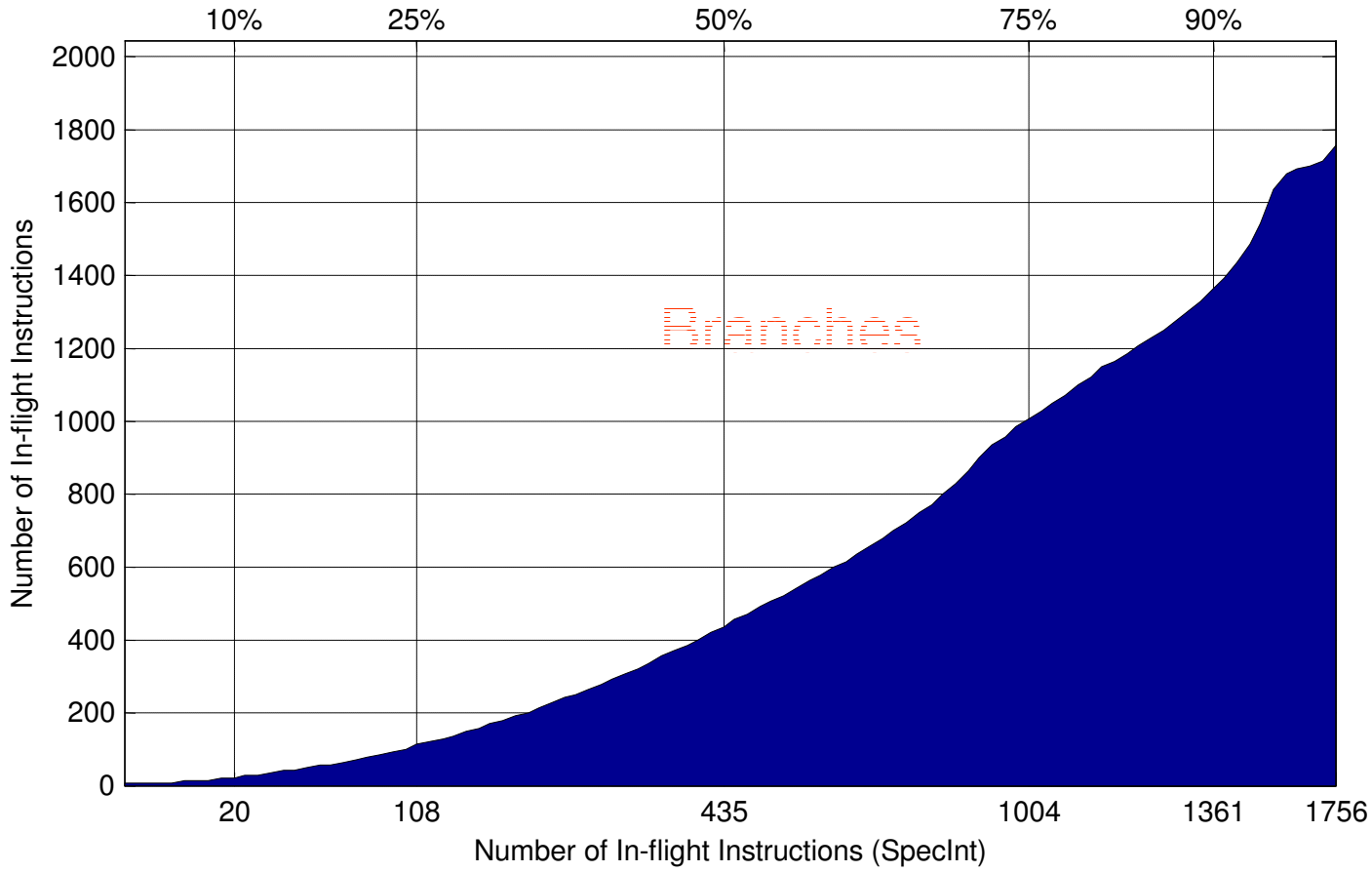
- Motivation
- Kilo-instruction Processors
  - Resource Utilization
  - Checkpointing
  - Out-of-order Commit Processors
  - Ephemeral Registers
  - Instruction Queues
- Performance Evaluation
- Related Work
- Conclusion

UPC

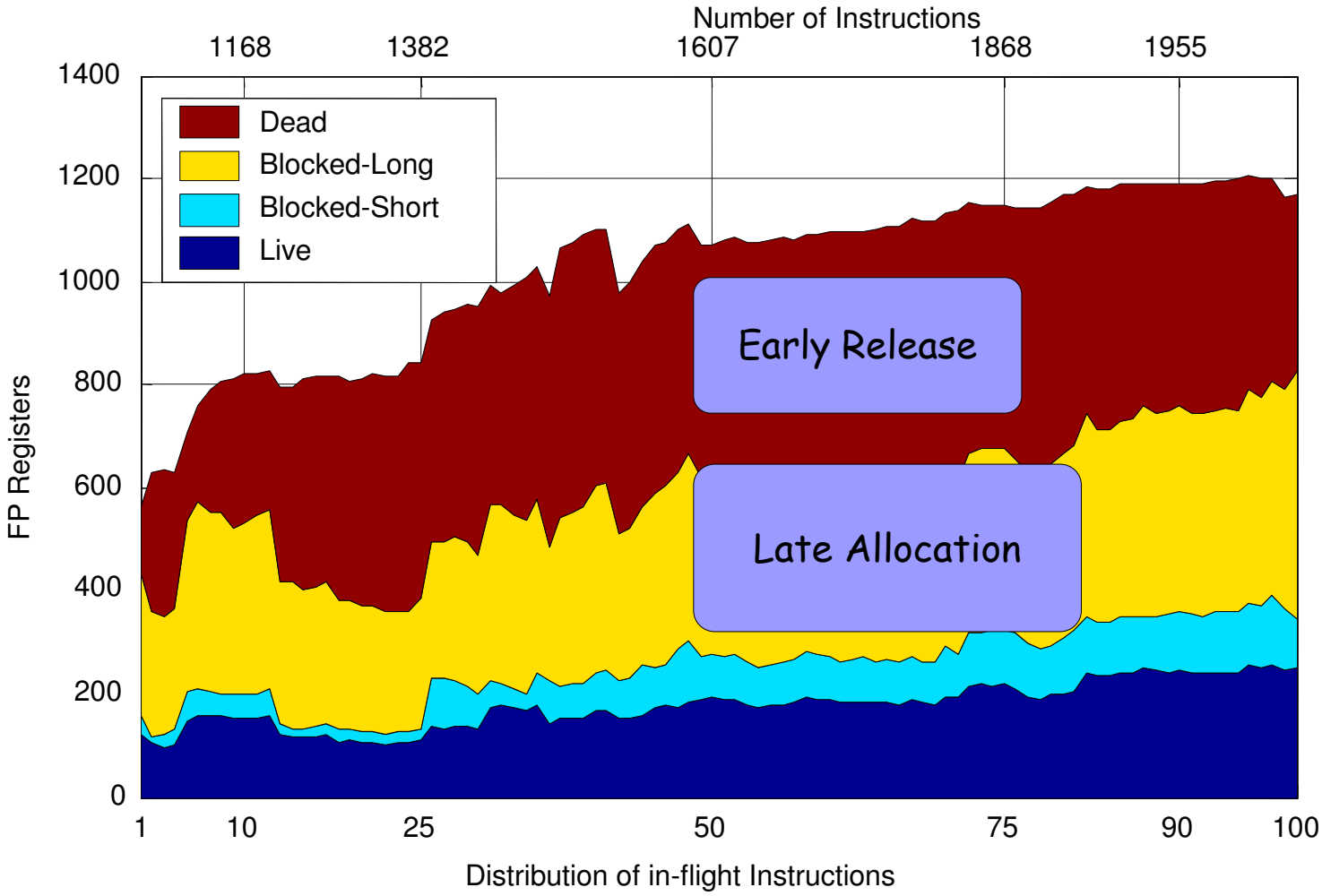
Instructions in-flight (FP, ROB=2048)



Instructions in-flight (Int, ROB=2048)

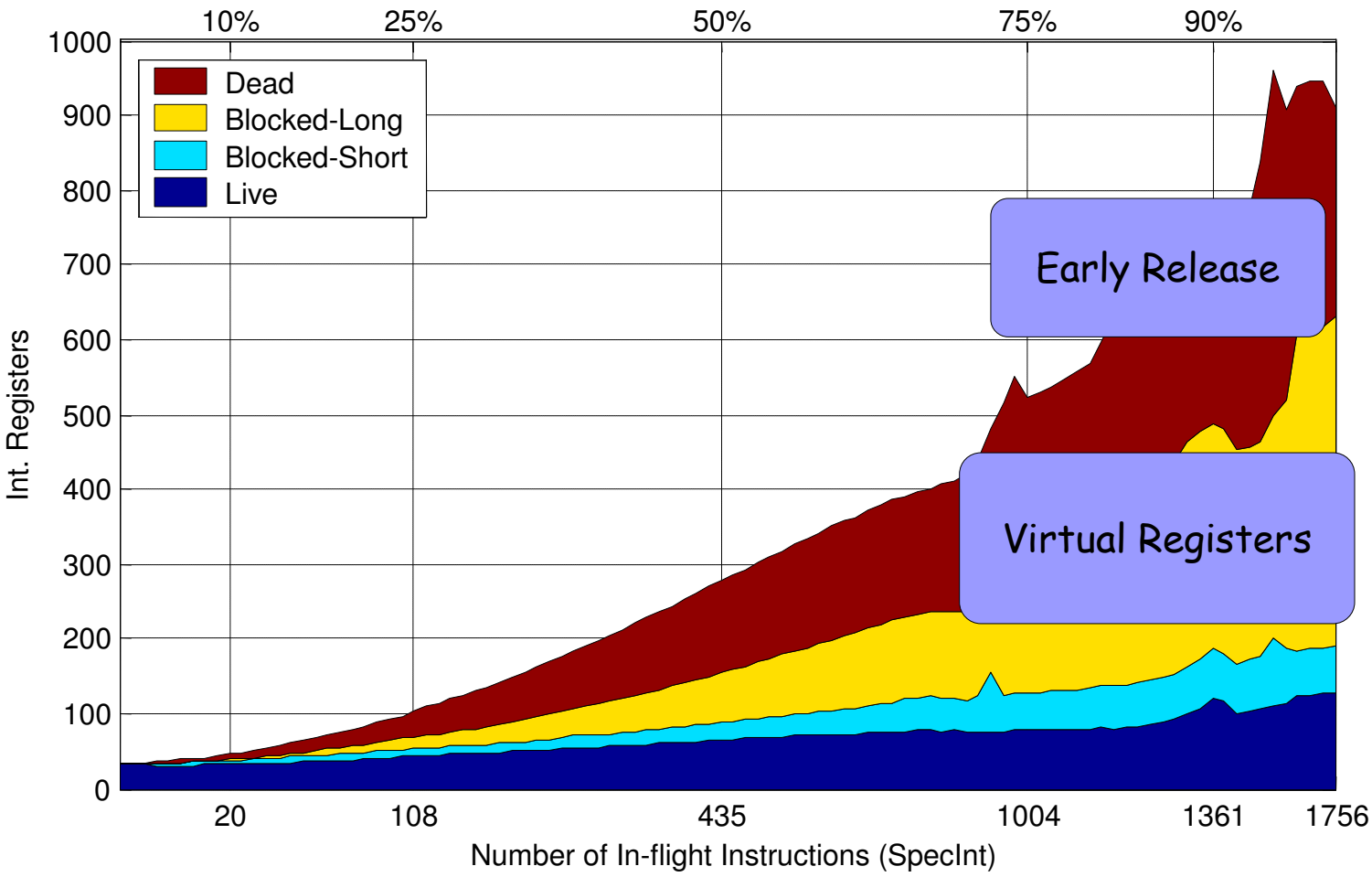


State of Registers (FP, ROB=2048)

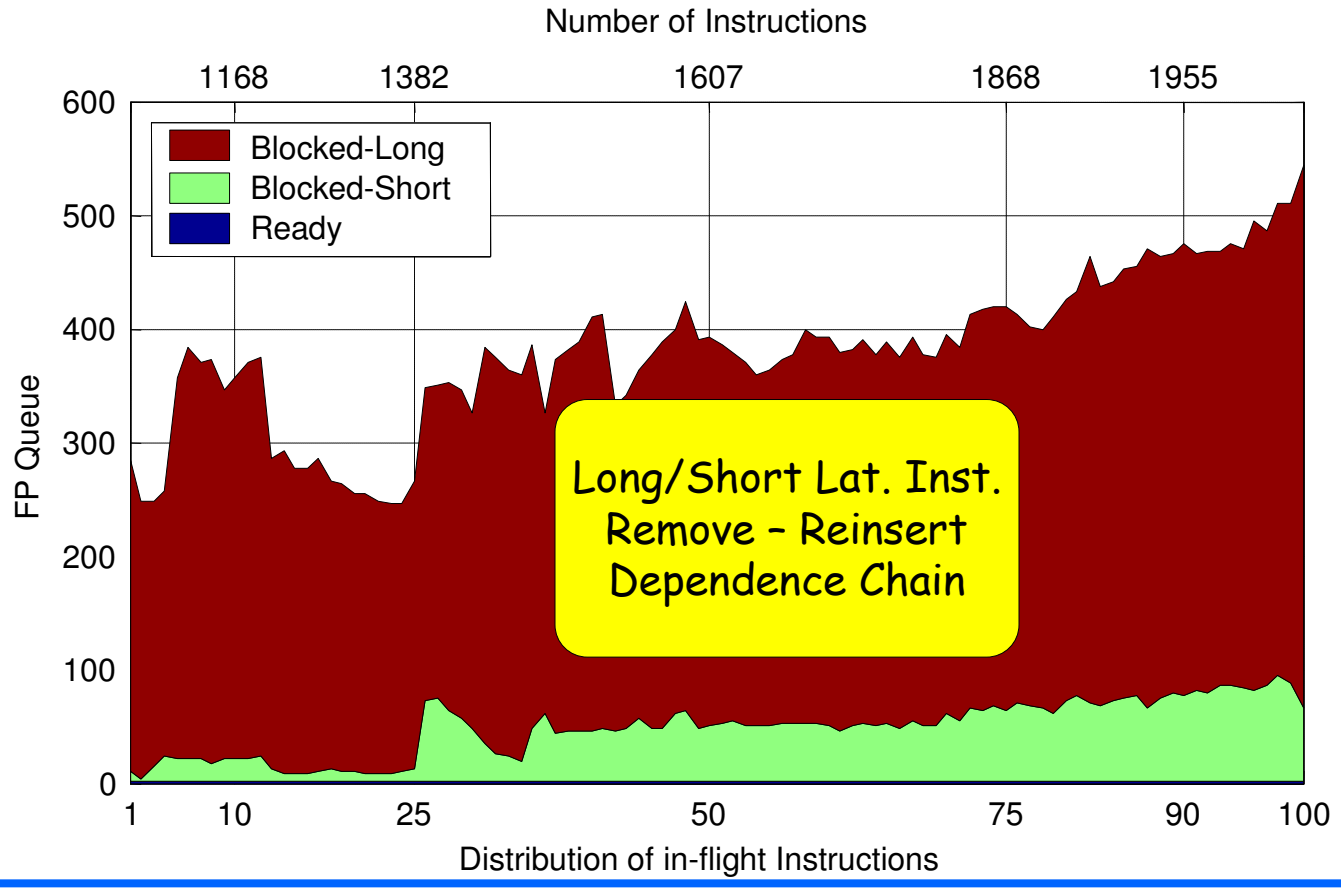


Example: 2034 instructions DO NOT require 1200 registers

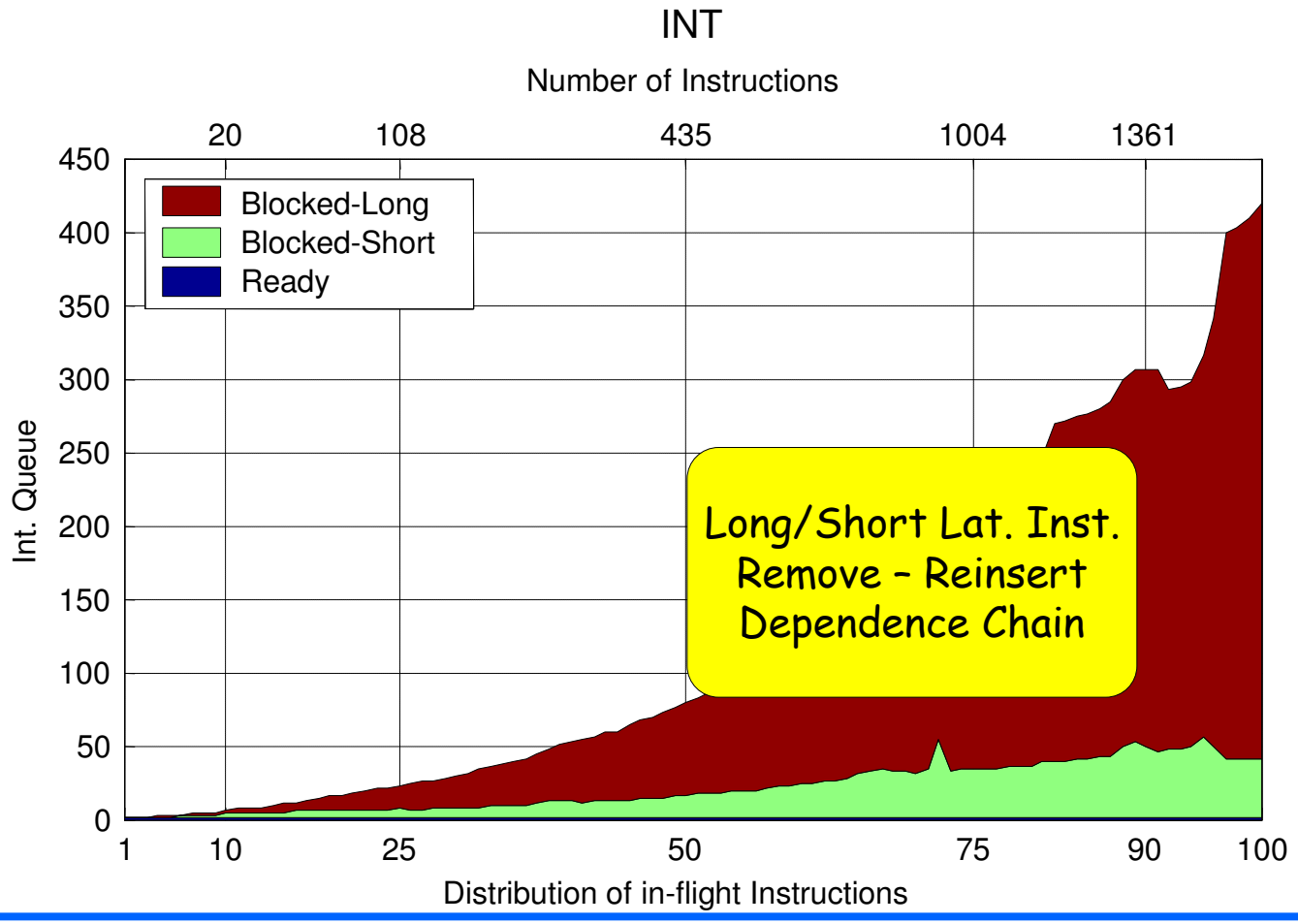
State of Registers (Int, ROB=2048)



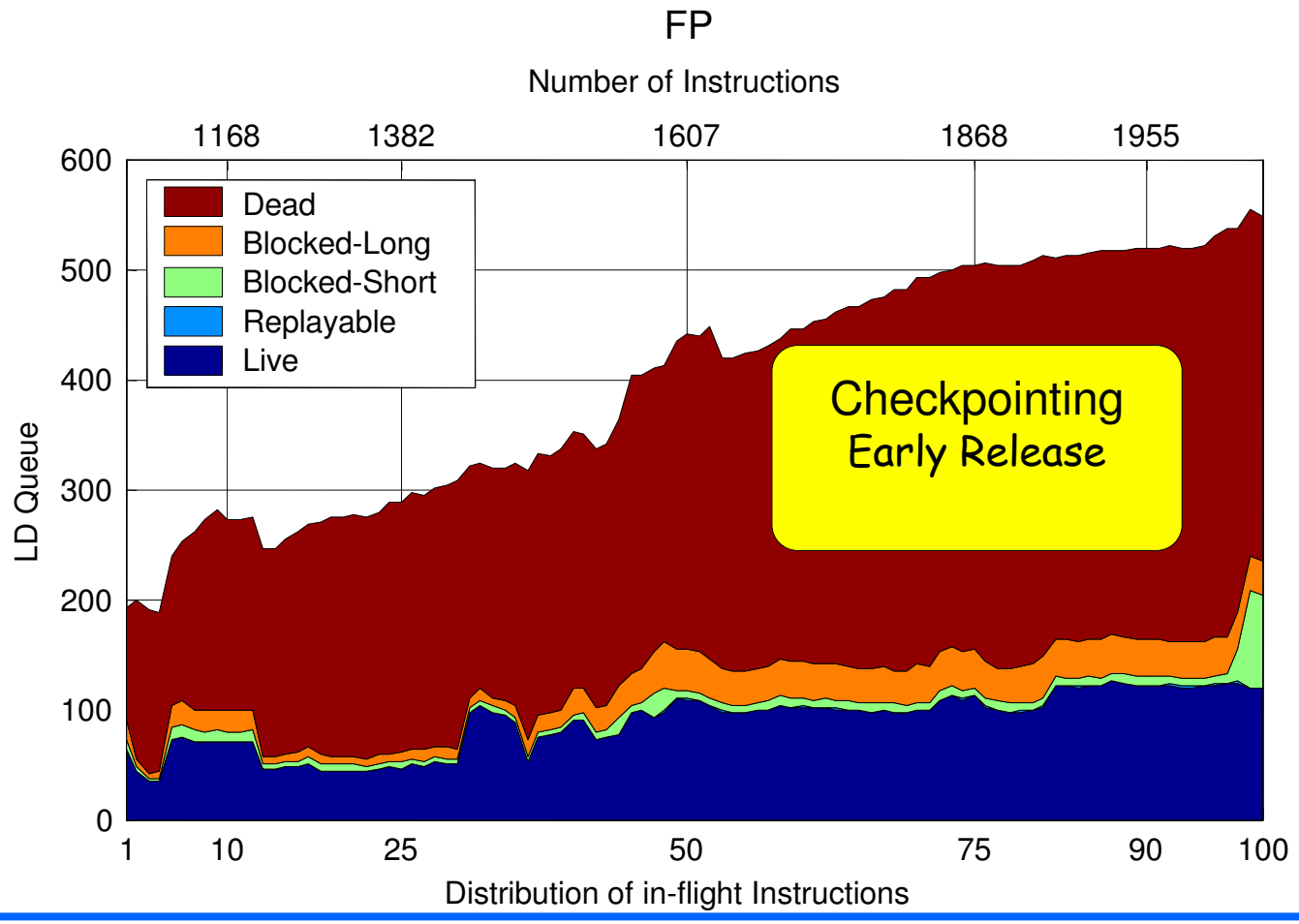
State of FP Queues (specFP, ROB=2048)



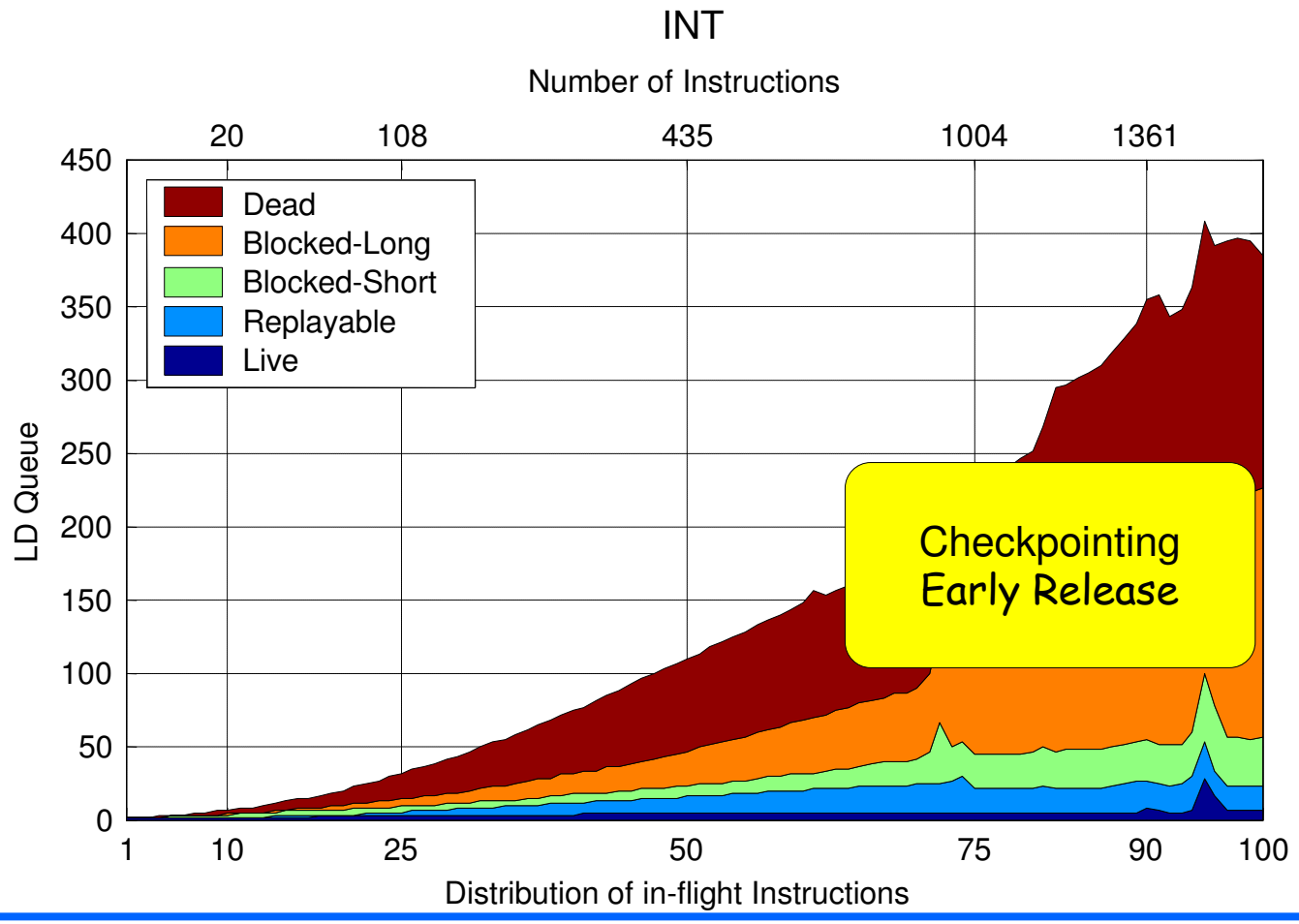
State of Int Queues (specInt, ROB=2048)



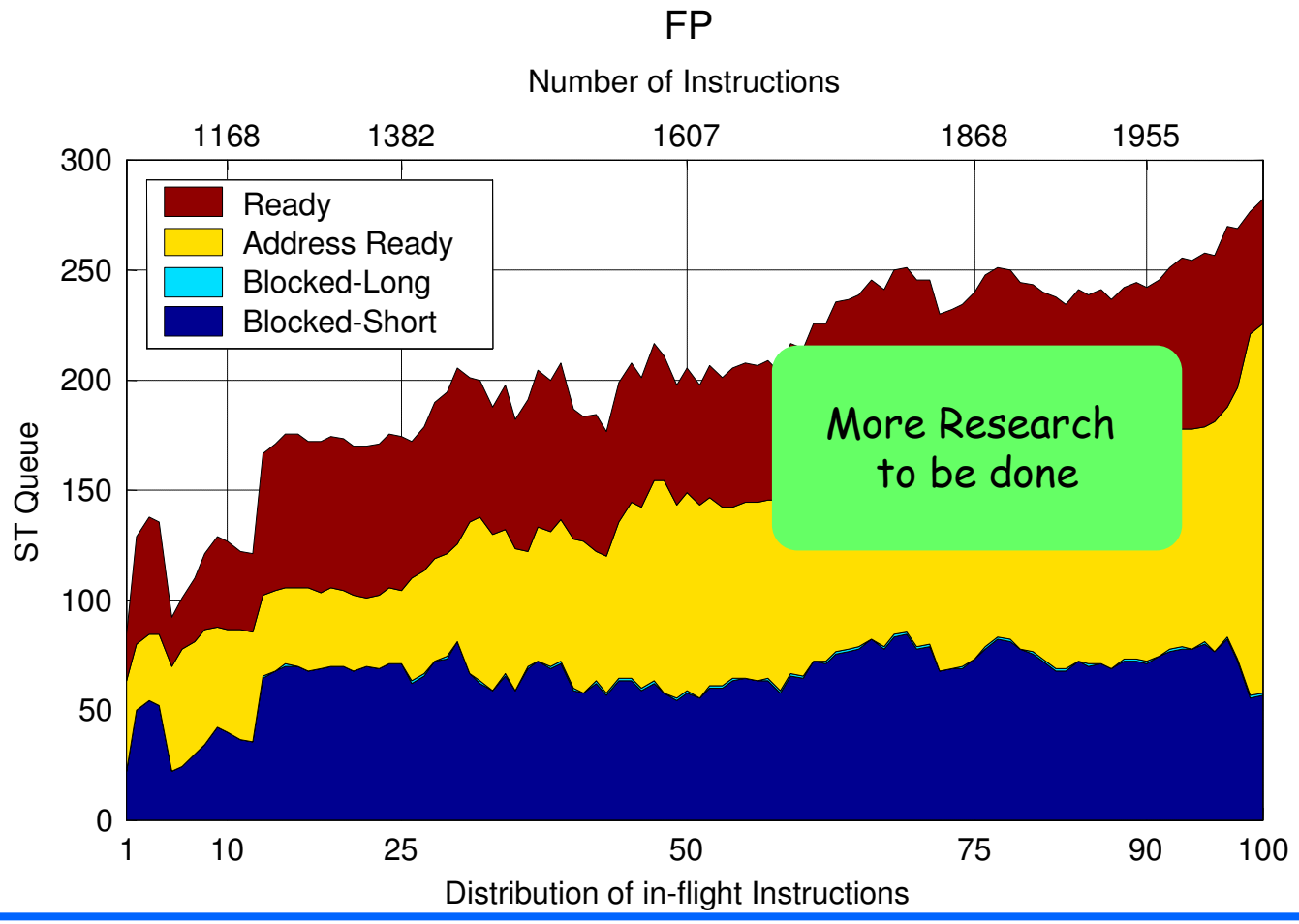
State of LD Queues (specFP, ROB=2048)



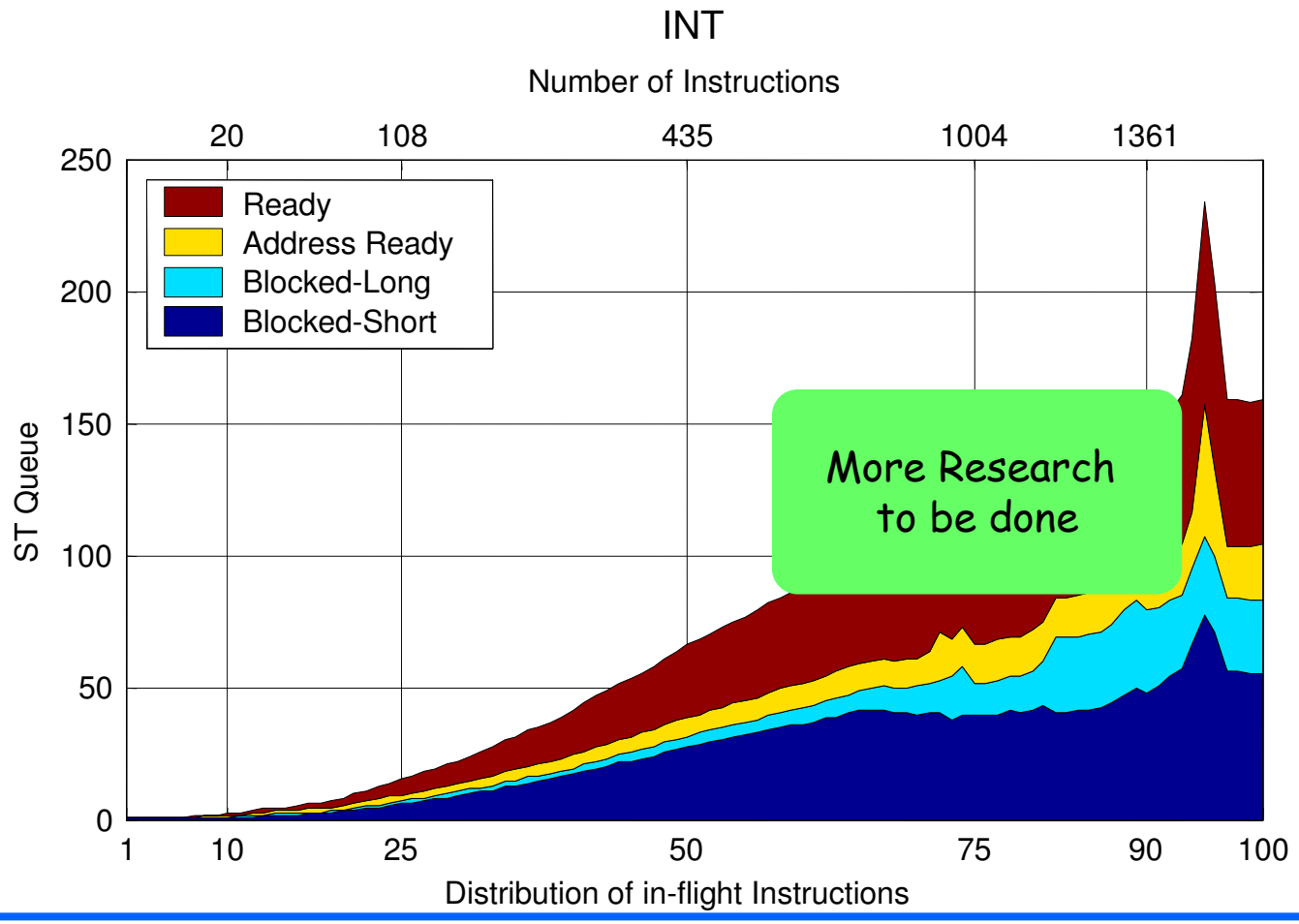
State of LD Queues (specInt, ROB=2048)



State of ST Queues (specFP, ROB=2048)



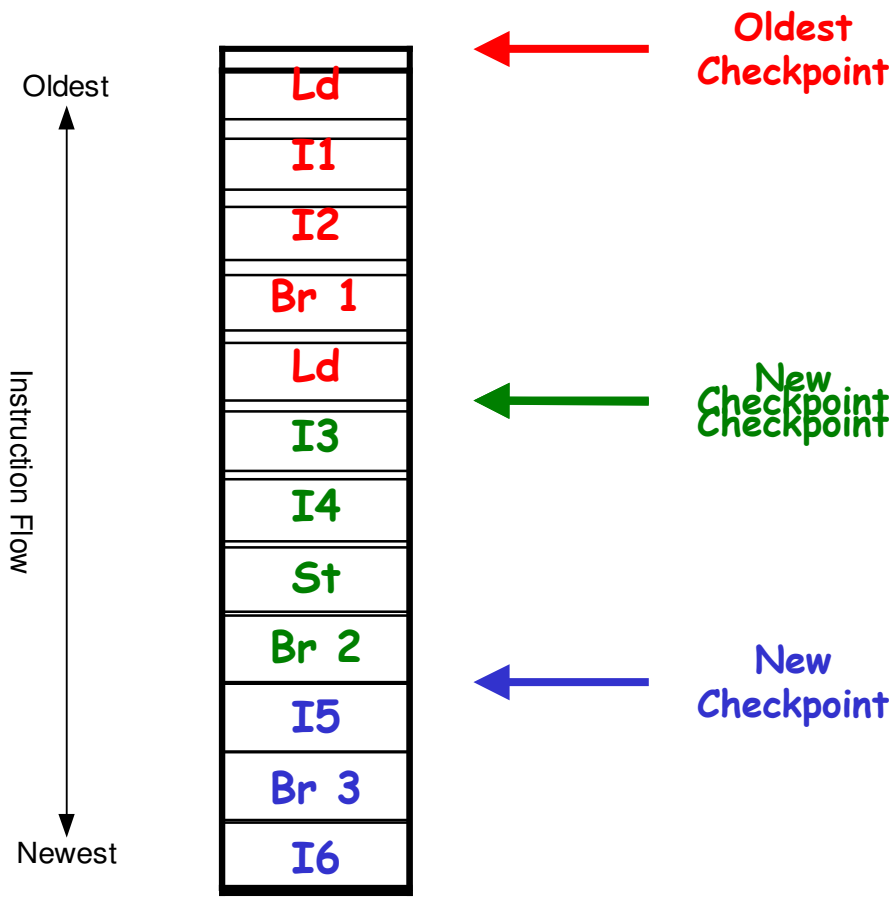
State of ST Queues (specInt, ROB=2048)



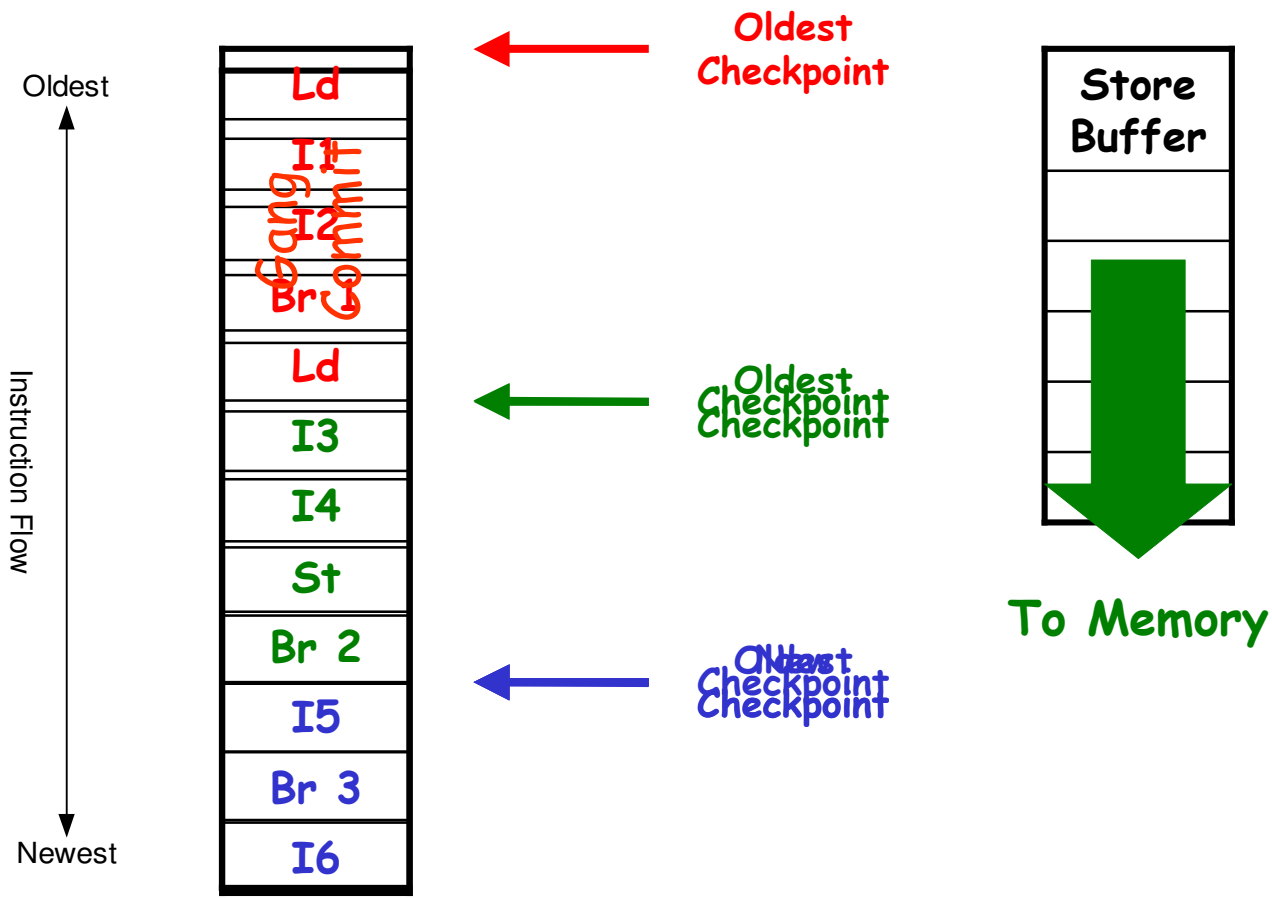
Out-of-order Commit Processors

- Motivation
- Kilo-instruction Processors
  - Resource Utilization
  - Checkpointing
  - Out-of-order Commit Processors
  - Ephemeral Registers
  - Instruction Queues
- Performance Evaluation
- Related Work
- Conclusion

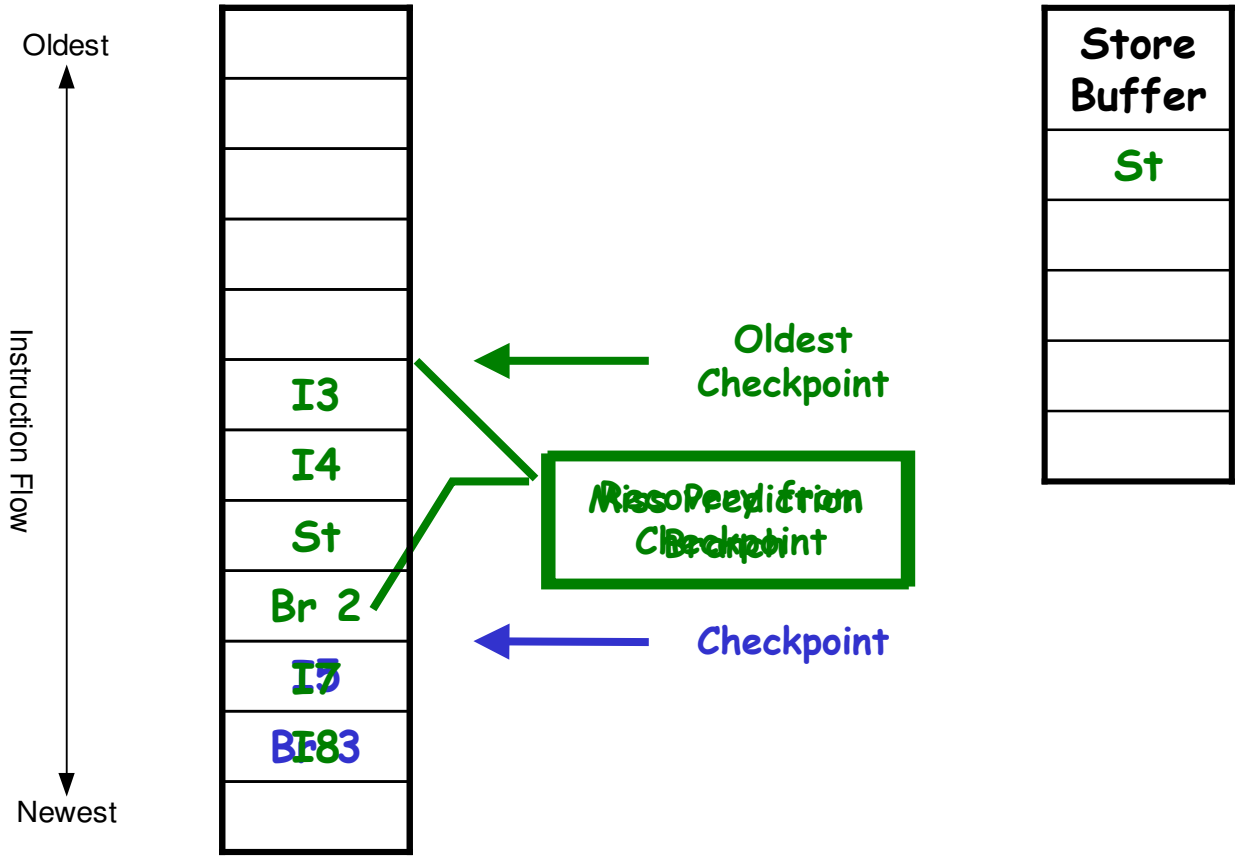
Checkpointing Instructions



Checkpointing Instructions



Checkpointing Instructions



Checkpoint Information

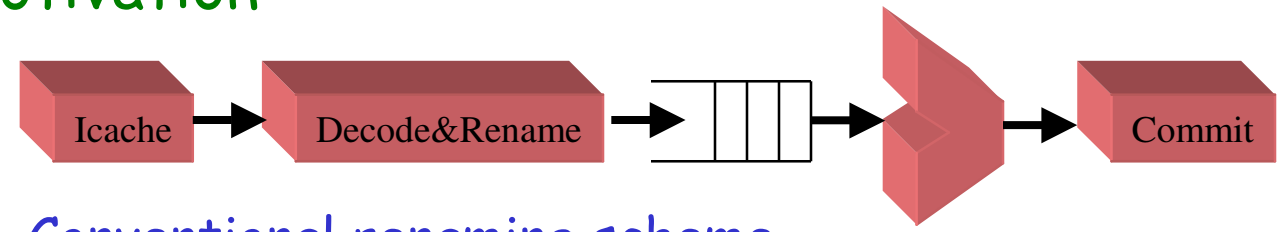
- PC of the ckeckpointed Instruction
- Instruction Counter: Count the number of instructions currently alive
- Map Table: to recover the register file
- Pointer to the Store Buffer
- Many policies to decide the instructions to be ckeckpointed: delinquent load and/or hard branches

Ephemeral registers

- Motivation
- Kilo-instruction Processors
  - Resource Utilization
  - Checkpointing
  - Out-of-order Commit Processors
  - Ephemeral Registers
  - Instruction Queues
- Performance Evaluation
- Related Work
- Conclusion

Virtual-Physical Registers

- Motivation



  - Conventional renaming scheme

  - Virtual-Physical Registers

  - Early Release

  - Ephemeral Registers

Early Release + Virtual Registers I

- Early Release of registers
  - Needs a mechanism to recover from exceptions
    - Checkpointing
  - Needs a pending counter for each register
    - When an instruction is decoded, each pending counter associated with the source registers is incremented and when the instructions is issued, the pending counter is decremented.
    - The instructions in a wrong path, are "nullified" and issued in order to maintain the pending counter.
  - Highly coupled with the renaming logic
    - CAM maps table scheme
    - A register can be freed if it is not referenced in any map table, and if his pending counter is zero.

Early Release + Virtual Registers II

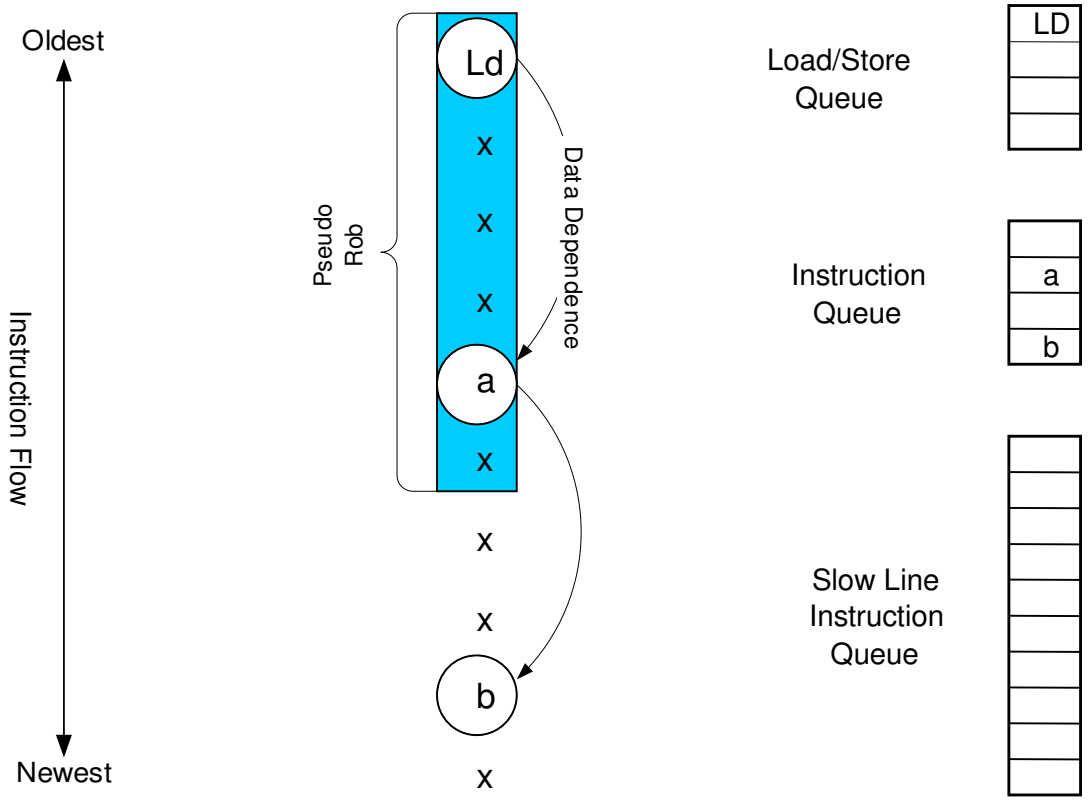
- Virtual Registers
  - Decouple renaming from physical register allocation.
  - Needs a new map table from Virtual Register to Physical Register.
    - The CAM Maps Tables, map from Logical Registers to Virtual Registers. Each entry is added with an entry for the physical register.

CAM Maps Tables	Virtual Registers															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
# Logical register																
# Physical register																
Map Table 1																
…																
Map Table N																
Pending Counter																

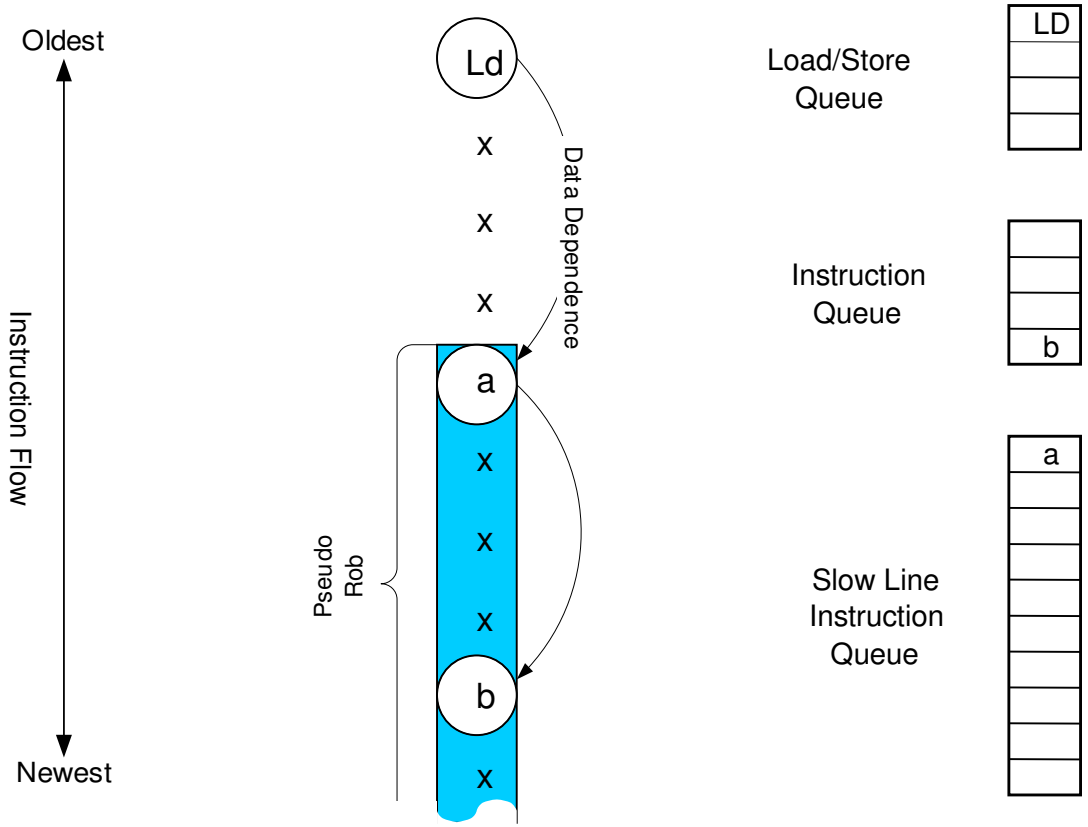
Instruction Queues

- Motivation
- Kilo-instruction Processors
  - Resource Utilization
  - Checkpointing
  - Out-of-order Commit Processors
  - Ephemeral Registers
  - Instruction Queues
- Performance Evaluation
- Related Work
- Conclusion

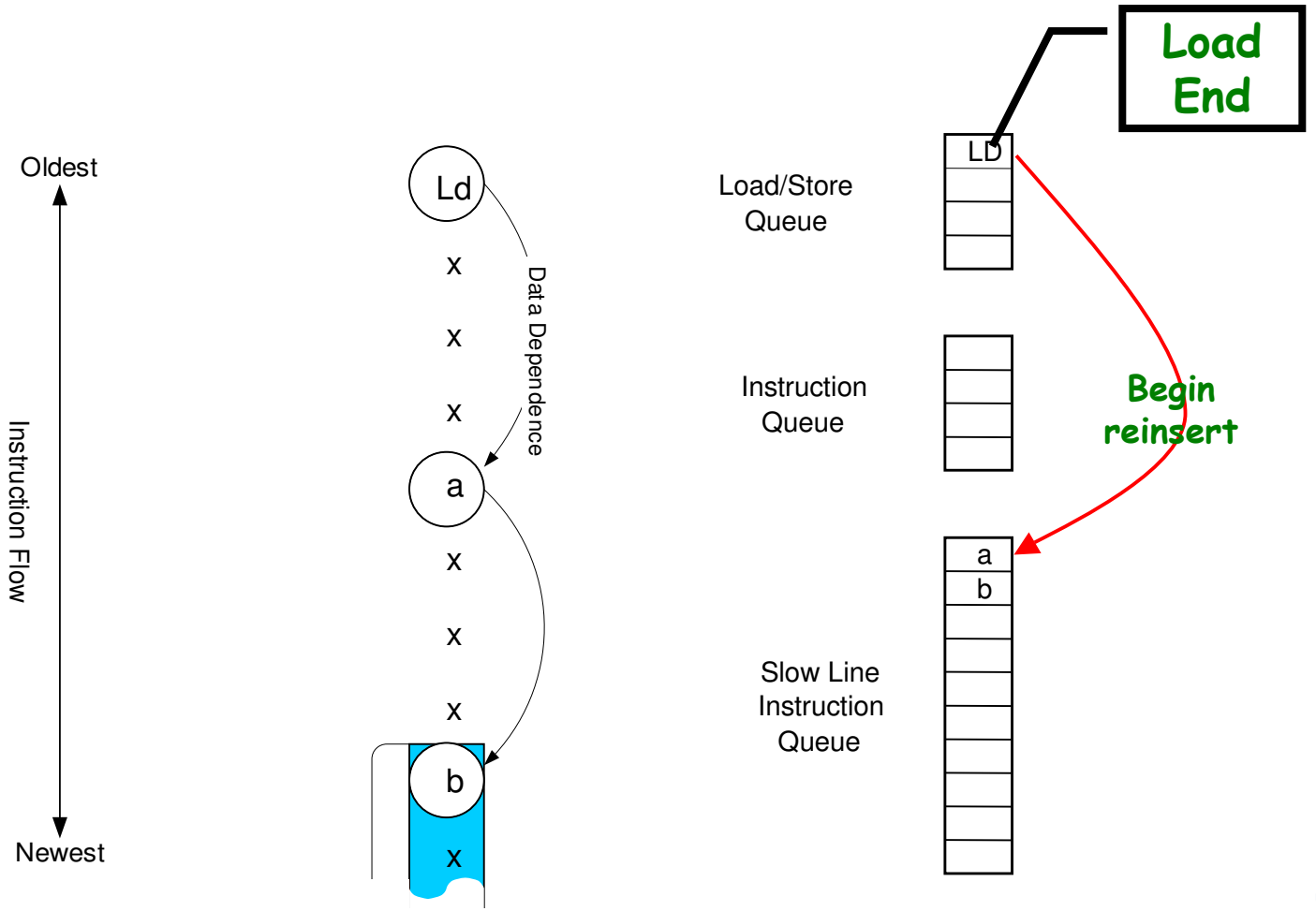
Instruction Queues



Instruction Queues



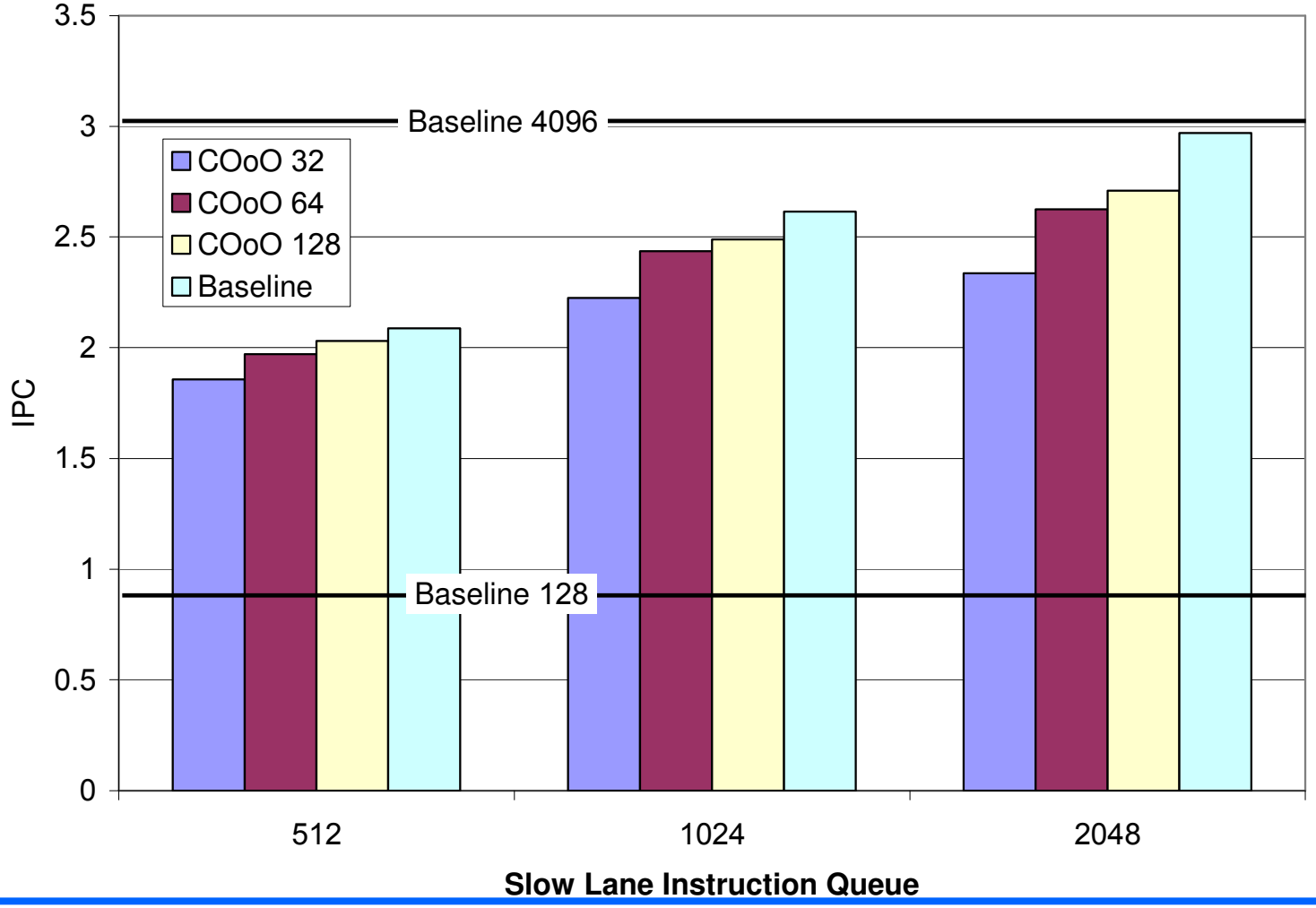
Instruction Queues



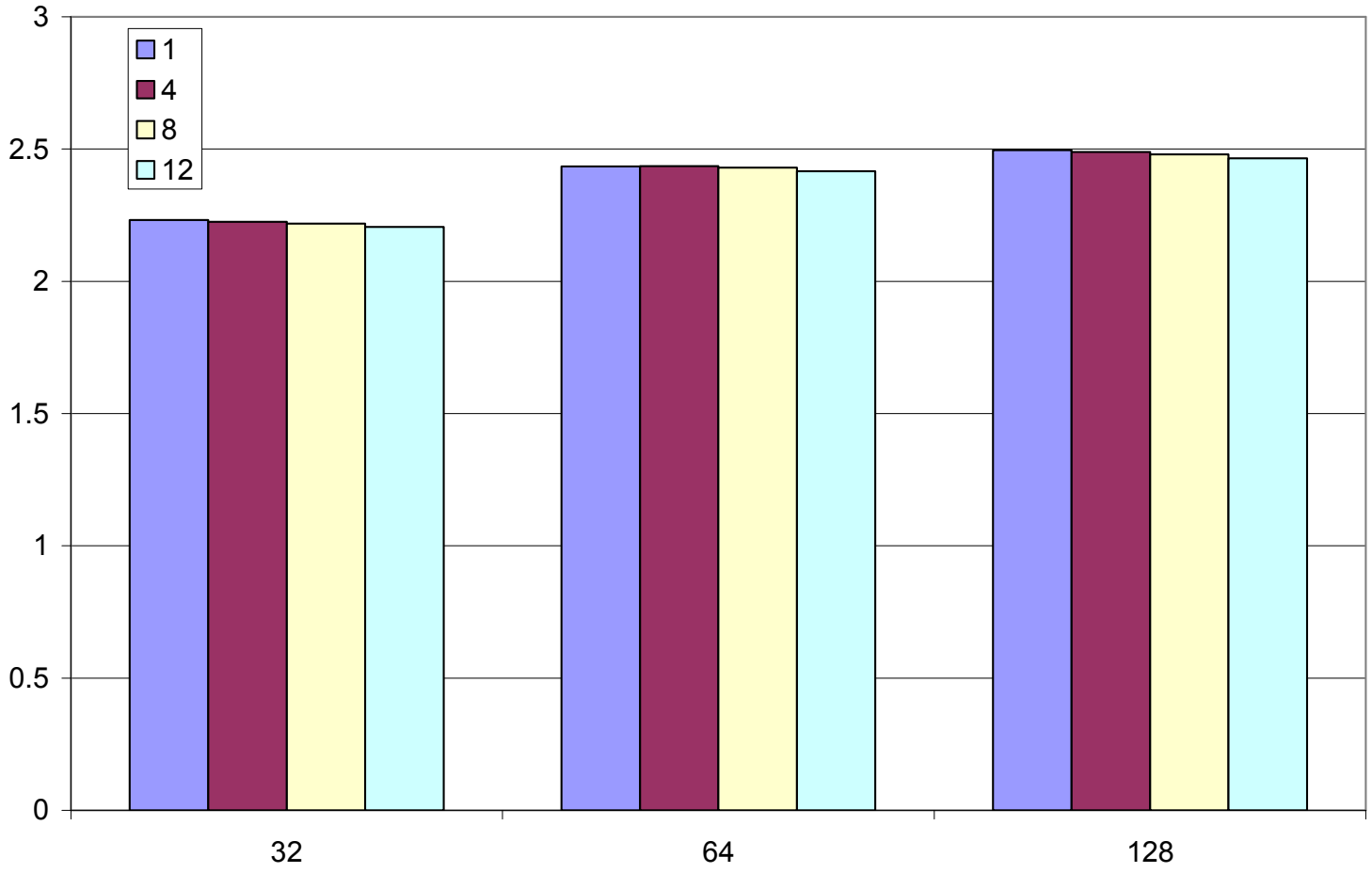
Performance Evaluation

- Processor Configuration (Baseline):
  - Fetch/Commit width — 4
  - Branch Predictor — 16K entries Gshare
  - Instruction L1 — 32Kb, 4-way, 32 bytes line, 2 cycle
  - Data L1 — 32Kb, 4-way, 32 bytes line, 2 cycle
  - L2 size — 512Kb, 4-way, 64 bytes line, 10 cycle
  - Memory Latency — 1000 cycles
  - Physical Registers — 4096 entries
  - Load/Store Queue — 4096 entries
  - Reorder Buffer — 4096 entries
  - Integer General Units — 4 (lat/rep 1/1)
  - Integer Mult/Div Units — 2 (lat/rep 3/1 and 20/20)
  - FP Functional Units — 4 (lat/rep 2/1)
  - FP Mult/Div/Sqrt Units — 2 (lat/rep 4/1, 12/12, 24/24)

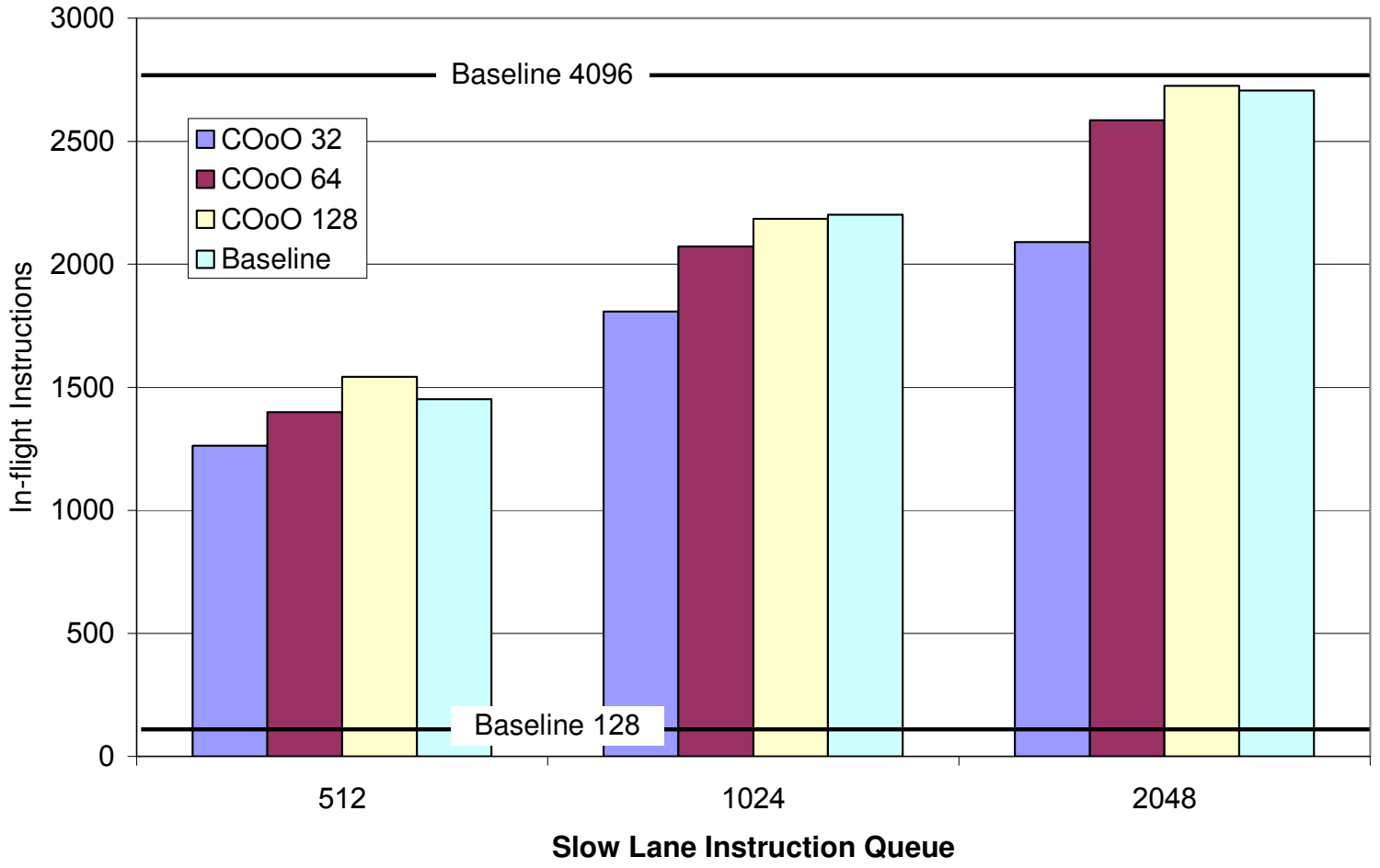
Slow-Lane Instruction Queue



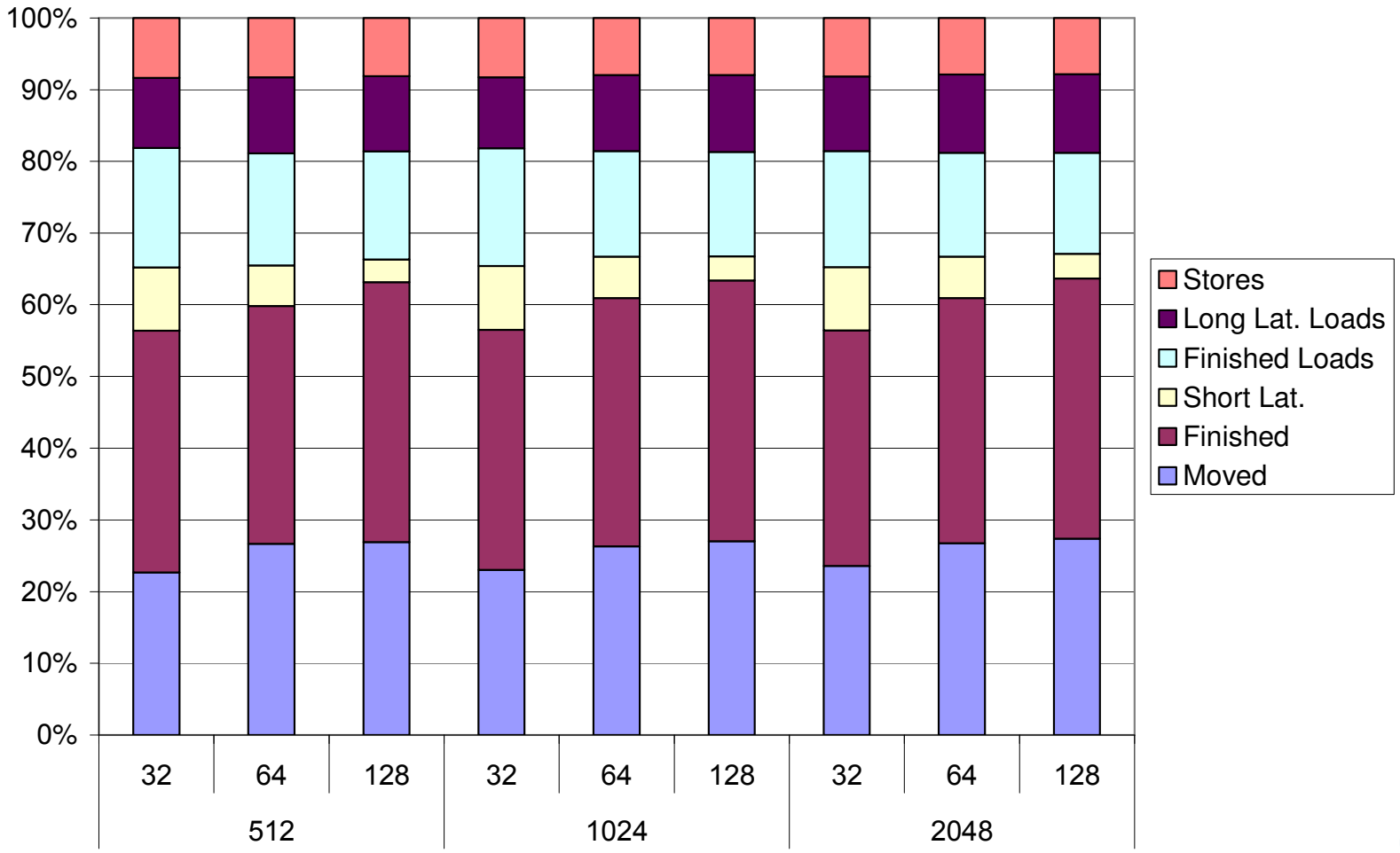
SLIQ to IQ cycles delay and Performance Degradation



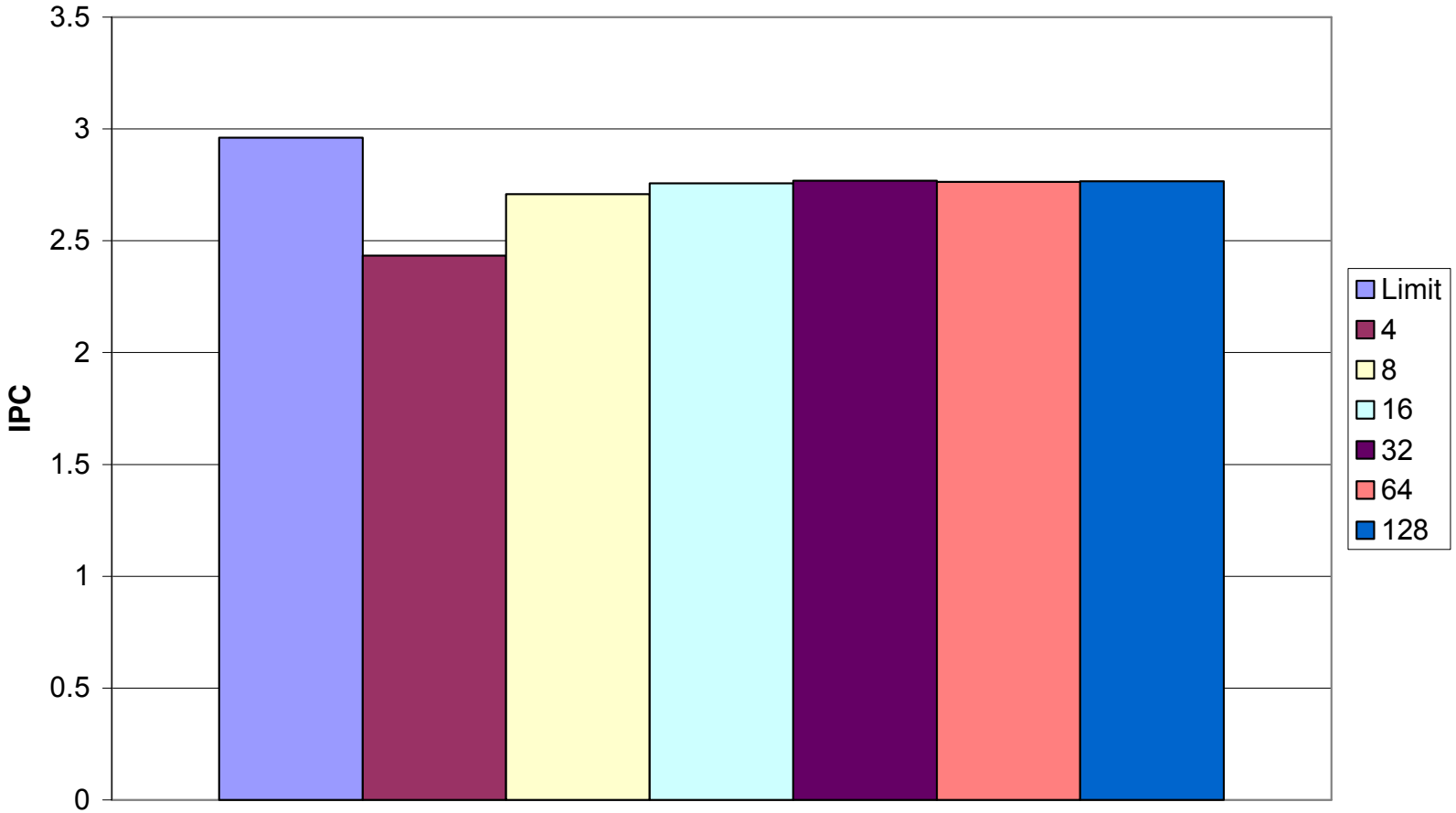
Length of the Virtual ROB



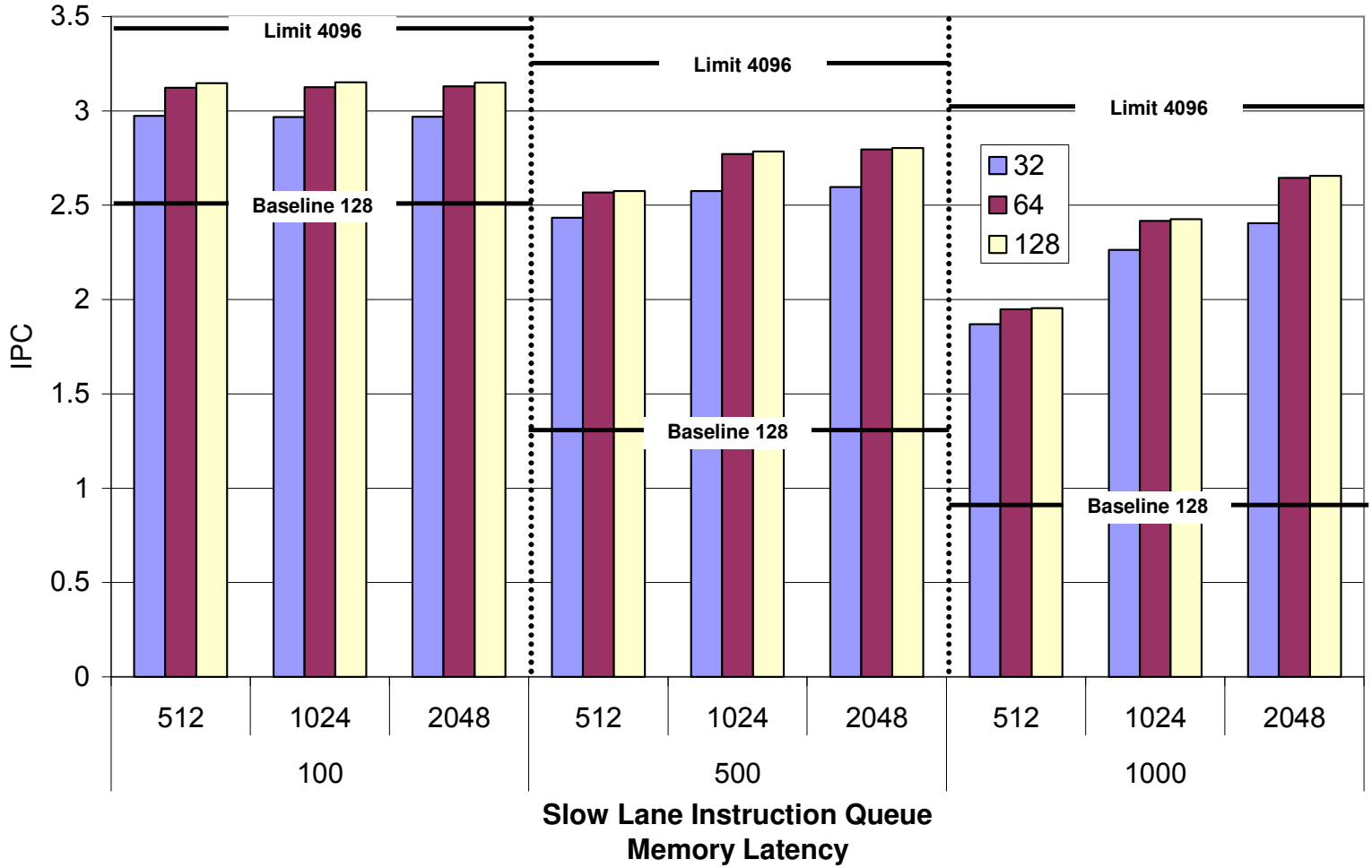
Status of the removed instructions



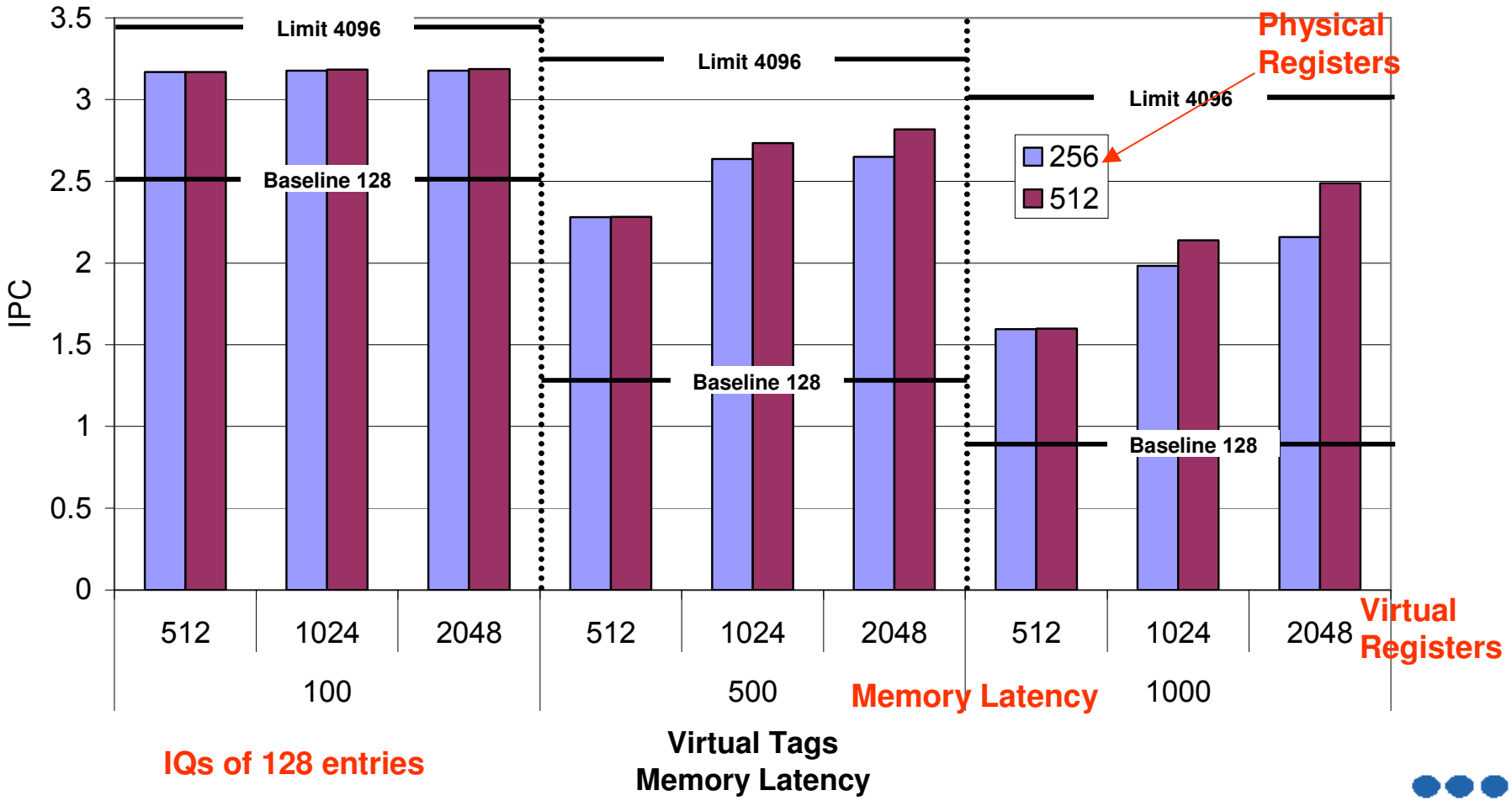
Number of Checkpointers and Performance



Memory Latency, SLIQ size and Performance



Putting All Together



Memory Latency

- Jouppi and P. Ranganathan. " The relative importance of memory latency, bandwidth and branch prediction,,," Whorkshop on Mixing Logic and DRAM: Chips that compute and remember" during ISCA-24, 1997

- S. Srinivasan and A. Lebeck " Load latency tolerance in dynamically scheduled processors" Micro-31, 1998

- K. Skadron, P. Ahuja, M. Martonosi and D. Clark "Branch prediction,instruction window size and cache size: Performance tradeoffs and simulation techniques" IEEE-TC, pp. 1260-1281, 1999.

UPC

Large Reorder Buffers

- G. Sohi, S. Breach and T. N. Vijaykumar "Multiscalar processors" ISCA-22, 1995.
- E. Rotenberg, Q. Jacobson, Y. Sazeides and J. Smith "Trace processors" ISCA-24, 1997
- H. Akkari and M. Driscoll "A dynamic multithreaded processor" Micro-31, 1998
- R. Balasubramonian, S. Dwarkadas, and D. Albonesi."Dynamically allocating processor resources between nearby and distant ilp" ISCA, June 2001.
  - Save some resourses allocated for a eager execution
- O. Mutlu, J. Stark, C. Wilkerson, and Y.N. Patt "Runahead execution: An alternative to very large instruction windows for out-of-order processors" HPCA-9, February 2003.
  - All resources are allocated for a eager execution
- P. Ranganathan, V. Pai and S. Adve "Using speculative retirement and large instruction windows to narrow the performance gap between memory consistency models" SPAA, 1997
- J. M. Tendler, S. Dodson, S. Fields, H. Lee, and B. Sinharoy "Power4 System Microarchitecture" IBM Journal of Research and Development, pp. 5-25, January 2002.

UPC

Checkpointing

- W.M. Hwu and Y. N. Patt."Checkpoint repair for out-of-order execution machines" ISCA-14, 1987.
  - Checkpointing as a recovery mechanism

- Early Release of Resources:

  - A. Cristal, M. Valero, and J. LLosa. "Large virtual ROBs by processor checkpointing" *Technical Report UPC-DAC-2002-39*, July 2002.

    - Multiple Checkpointers
    - Out-of.order Commit
    - Early release of registers and loads

  - J.F. Martínez, J. Renau, M.C. Huang, M. Prvulovic, and J. Torrellas. Cherry: checkpointed early resource recycling in out-of-order microprocessors. MICRO-35, 2002.
    - One checkpoint
    - Early release of resources

  - H. Akkari, R. Rajwar and S. T. Srinivasan "Checkpointing Processing and Recovery: Towards Scalable Large Instruction Window Processors" Micro-36, 2003
    - Similar to A. Cristal et al. "Large Virtual ROB....."
    - Load/store queue

UPC

Instruction Queues

- S. Palacharla, N.P. Jouppi, and J.E. Smith "Complexity-effective superscalar processors" ISCA-24, 1997.
  - Divide the Instruction queues in a set of FIFO queues
- A.R. Lebeck, J. Koppanalil, T. Li, J. Patwardhan, and E. Rotenberg "A large, fast instruction window for tolerating cache misses" ISCA-29, 2002.
  - Remove-Reinsert Mechanism
  - Keep the load dependence of all instructions
- E. Brekelbaum, J. Rupley, C.Wilkerson, and B. Black "Hierarchical scheduling windows" ISCA-35, 2002.
  - Two clusters, a slow/big one, and a faster/small one for critical instructions

- A. Cristal, D. Ortega, J. Llosa and M. Valero "Out-of-Order Commit Processors" *Technical Report UPC-DAC-2003-44*, July 2003. HPCA-10, Madrid, Feb. 2004
  - Remove-Reinsert Mechanism
  - Simple reinsert mechanism

Register File

- M. Moudgill and K. Pingali and S. Vassiliadis "Register renaming and dynamic speculation: an alternative approach", In *Proceedings of the 26th annual international symposium on Microarchitecture,* pages 202-213, 1993.
  - Early Release of Registers
- T. Monreal, A. González, M. Valero, J. González, V. Viñals "Delaying Physical Register Allocation through Virtual-Physical Registers" In *Proceedings of the 33th annual international symposium on Microarchitecture,* 1999.
  - Virtual Registers, Late allocation of registers

- A. Cristal, J. Martínez, M. Valero and J. Llosa. "Ephemeral Registers". *Technical Report CSL-TR-2003-1035* , University, 2003.
  - **Ckeckpoint** + Early Release + Late allocation of registers

Load-Store Queues

- A. Cristal, M. Valero, and J. LLosa. "Large virtual ROBs by processor checkpointing" *Technical Report UPC-DAC-2002-39*, July 2002.
  - Early release of registers and loads
- J.F. Martínez, J. Renau, M.C. Huang, M. Prvulovic, and J. Torrellas "Cherry: checkpointed early resource recycling in out-of-order microprocessors" MICRO-35, 2002.
  - Early release of loads
- H. Akkari, R. Rajwar and S. T. Srinivasan "Checkpointing Processing and Recovery: Towards Scalable Large Instruction Window Processors" Micro-36, 2003
  - Load/store queue

Conclusion

- Affordable "Kilo-instructions Processors"
- Checkpointing to resource-conscious architectures
  - Out-of- order commit
  - Ephemeral registers
  - Two-level instruction queues
  - Early release of loads
  - Load/store queues management
- New ideas to watch for
  - Better branch predictors
  - Predication and Multi-path execution
  - Control independence instructions
  - Reuse of large blocks of instructions

UPC

Thank you very much ☺

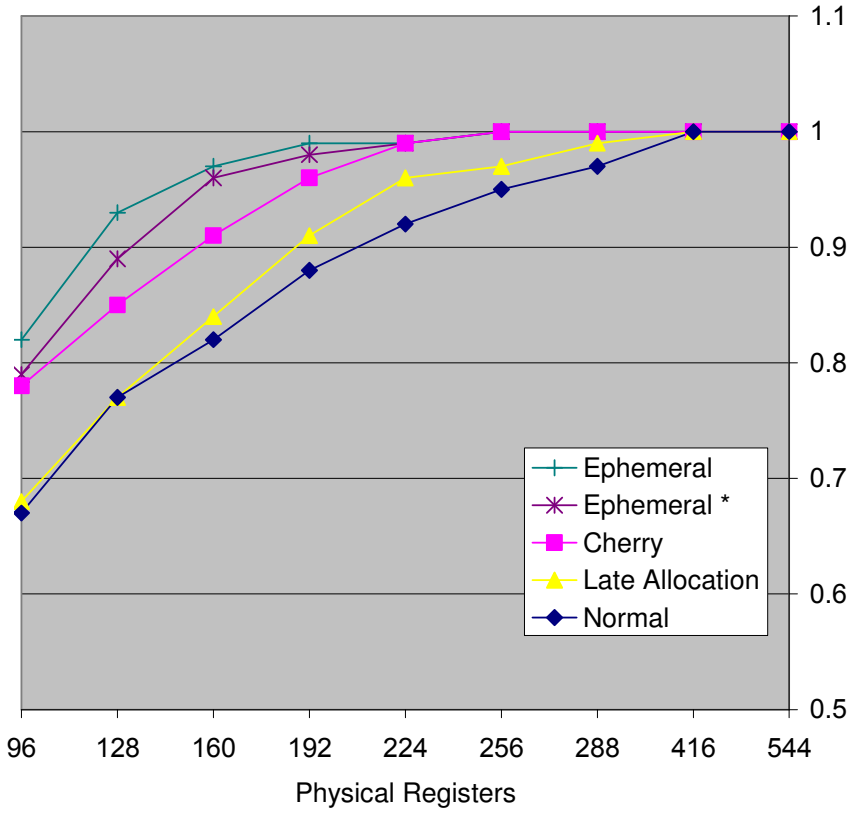
Acknowledgments

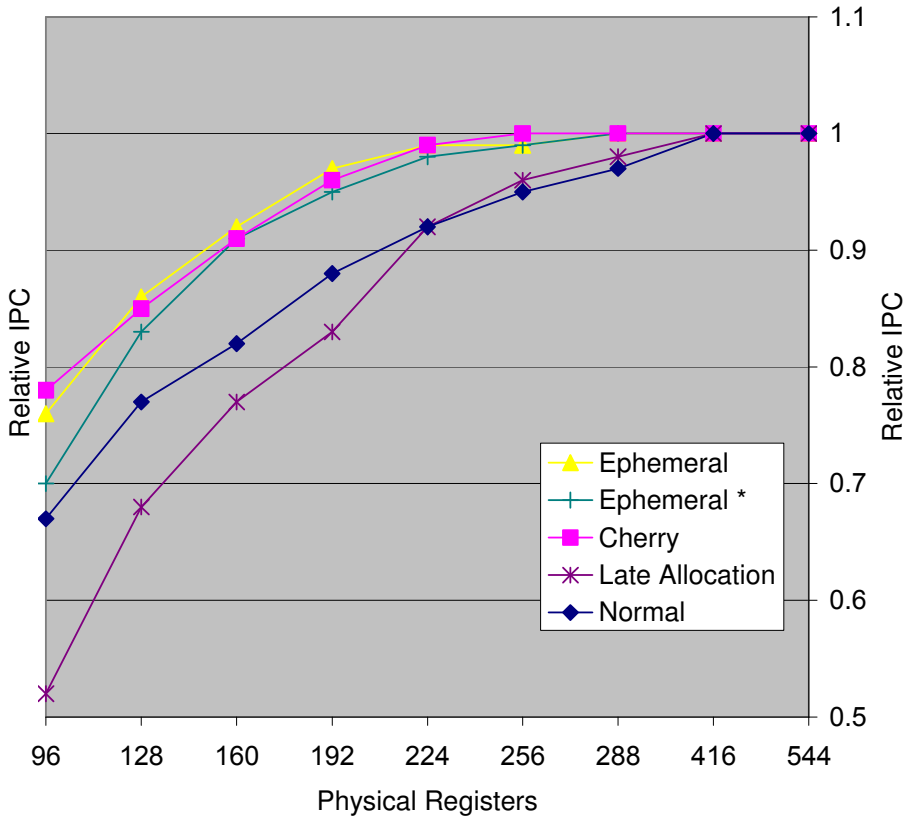
- Adrián Cristal
- José Martínez
- Josep Llosa
- Daniel Ortega

- Jim Smith
- Yale Patt
- Guri Sohi
- Mark Hill

- Intel and Ronny Ronen

Ephemeral Registers


1.1
1
0.9
0.8
0.7
0.6
0.5
96
128
160
192
224
256
288
416
544
Physical Registers
Ephemeral
Ephemeral *
Cherry
Late Allocation
Normal


Relative IPC
1.1
1
0.9
0.8
0.7
0.6
0.5
96
128
160
192
224
256
288
416
544
Physical Registers
Ephemeral
Ephemeral *
Cherry
Late Allocation
Normal