Overcoming the Memory Wall: Kilo-Instruction , Runahead, and SMT Processors

Prof. Mateo Valero,

UPC, Barcelona

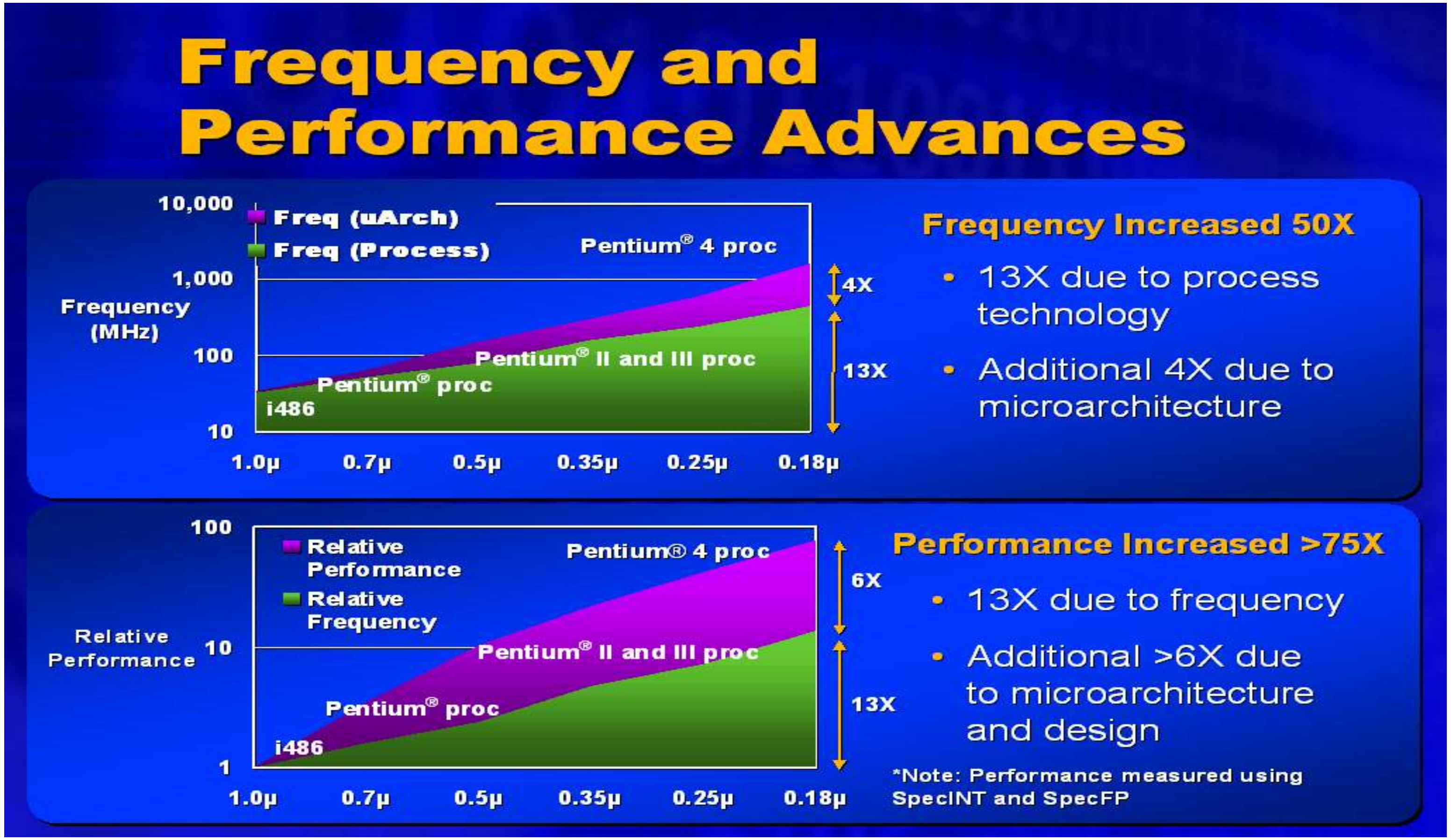
BSC Director

Onassis Foundation

Lectures on Computer Science

Heraklion, Crete, July 21-25, 2008

Motivation



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

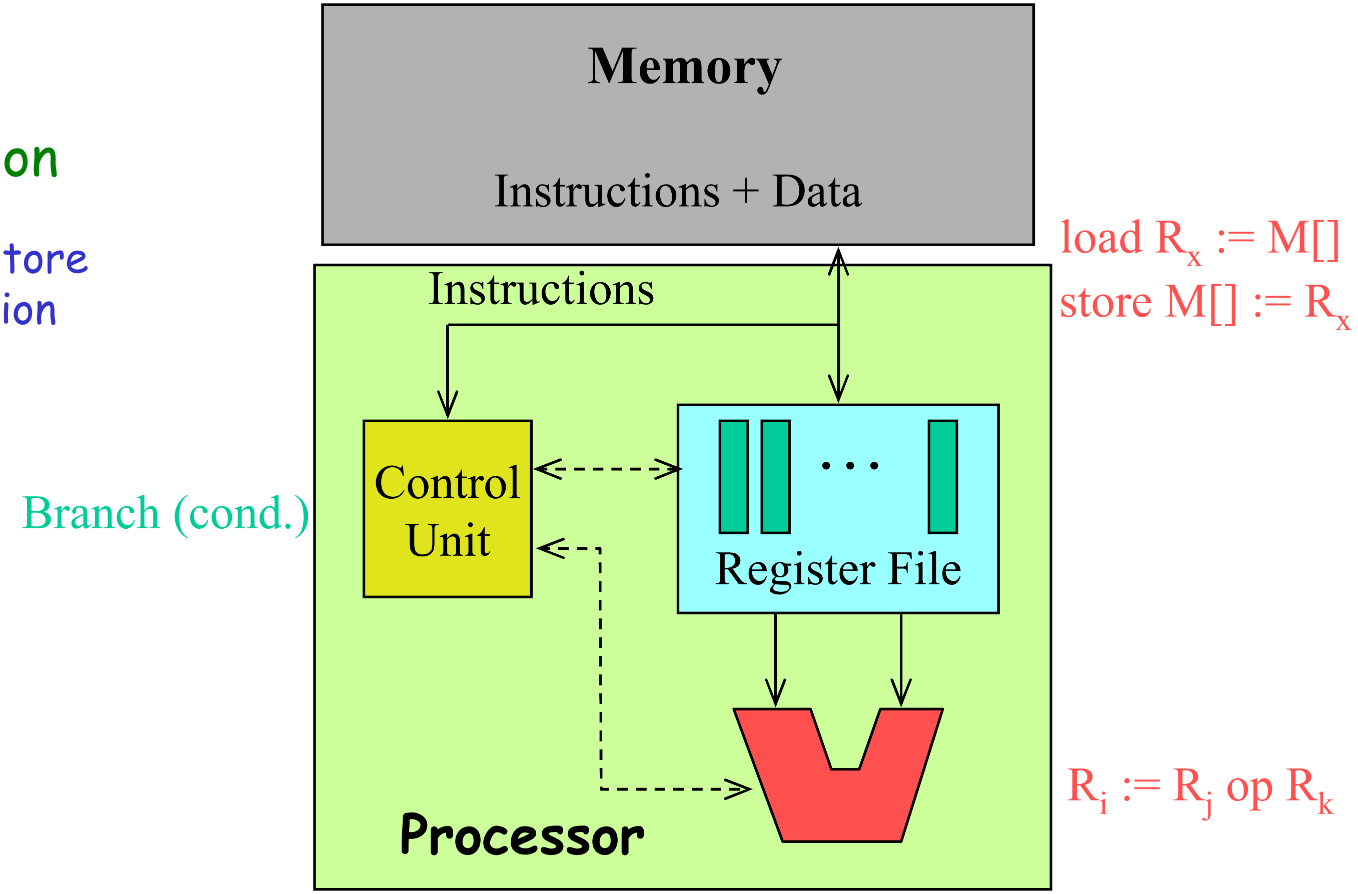
*Note: Performance measured using SpecINT and SpecFP

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

Justin Rattner, Intel-MRL, Keynote lecture, Micro-32

Processor Organization: Basic Concepts

- Instruction types:
  - Load/Store
  - Operation
  - Control



load R_x := M[]

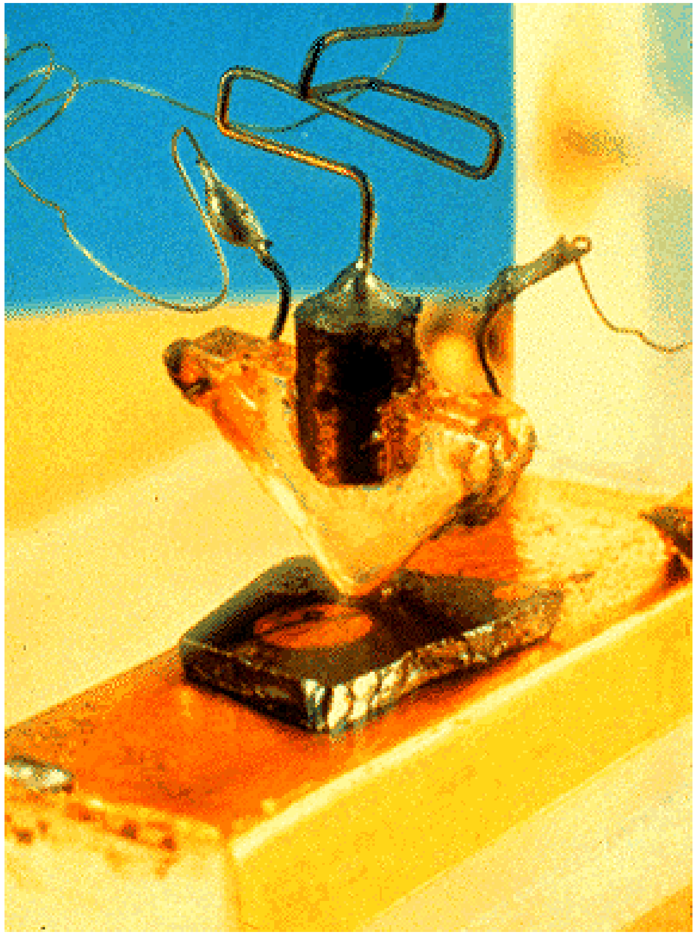
store M[] := R_x

Branch (cond.)

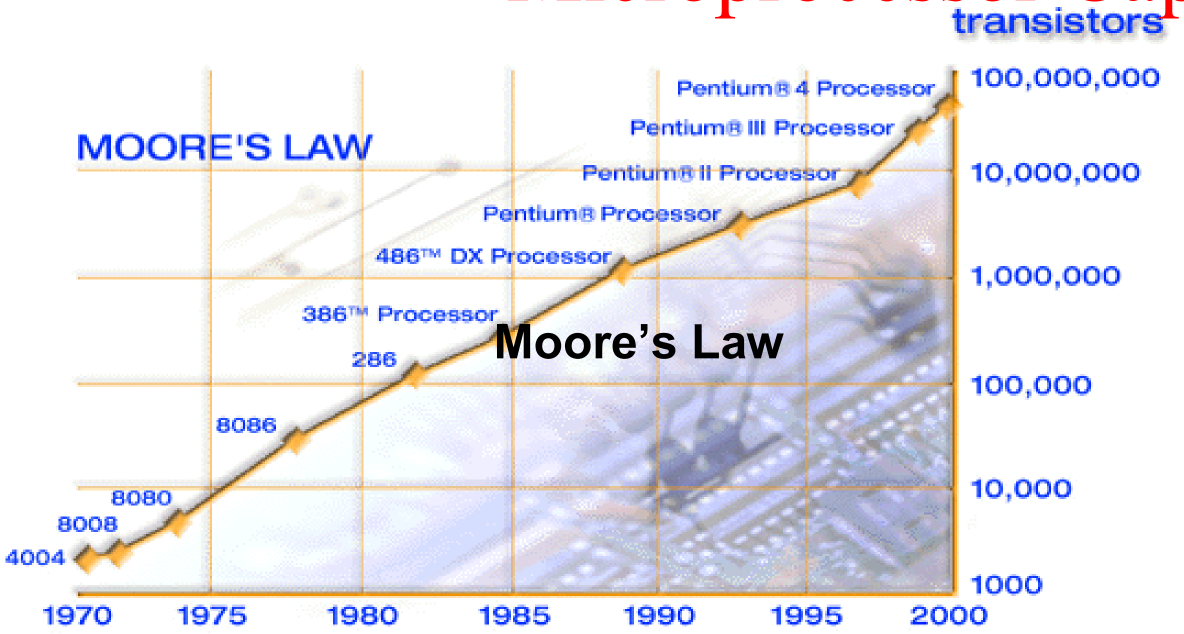
$R_i := R_j$ op R_k

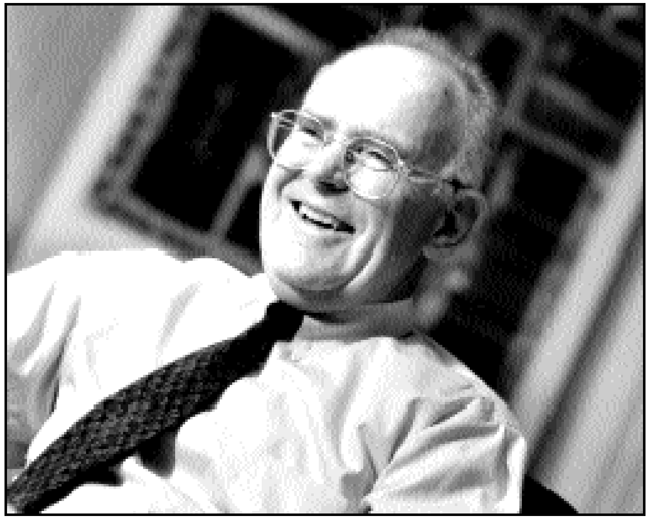
Technological Achievements

- Transistor (Bell Labs, 1947)
  - DEC PDP-1 (1957)
  - IBM 7090 (1960)
- Integrated circuit (1958)
  - IBM System 360 (1965)
  - DEC PDP-8 (1965)
- Microprocessor (1971)
  - Intel 4004



Technology Trends: Microprocessor Capacity





2X transistors/Chip Every 1.5 years

Called "Moore's Law"

Gordon Moore (co-founder of Intel) predicted in 1965 that the transistor density of semiconductor chips would double roughly every 18 months.

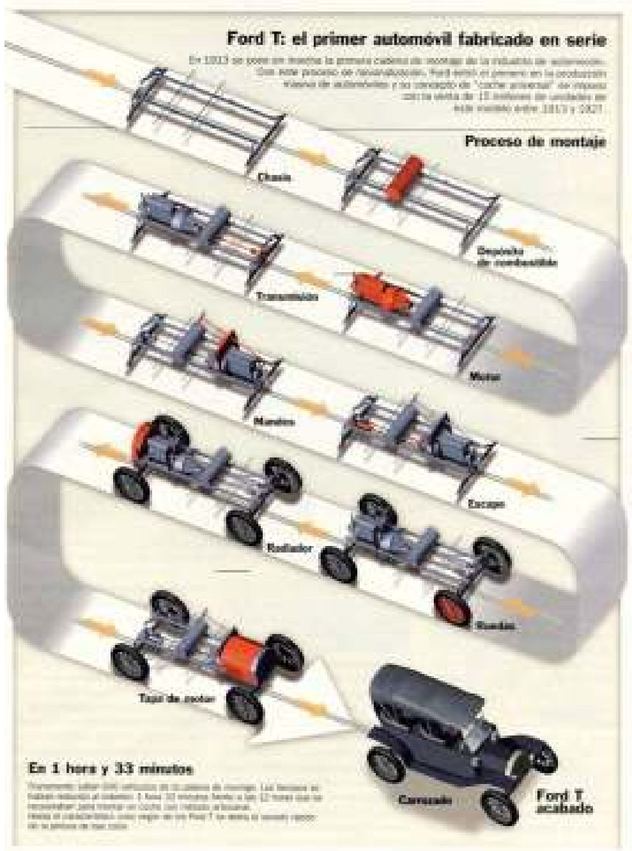
Microprocessors have become smaller, denser, and more powerful. Not just processors, bandwidth, storage, etc

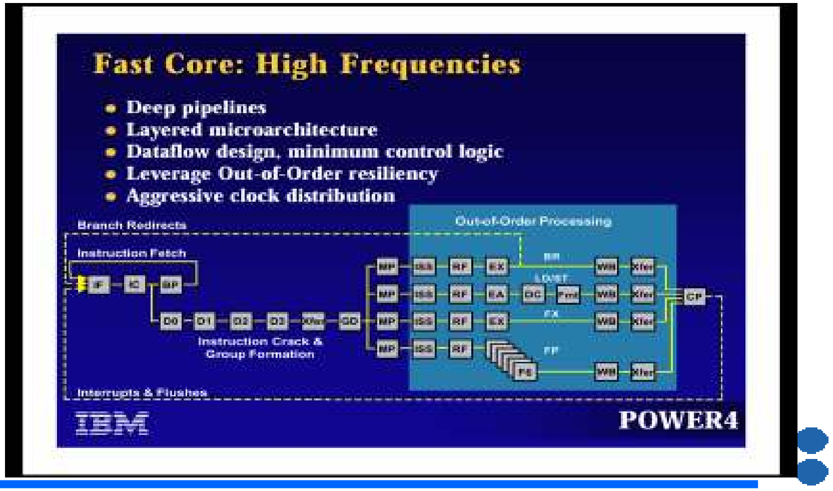
Technology Outlook

High Volume Manufacturing	2004	2006	2008	2010	2012	2014	2016	2018
Technology Node (nm)	90	65	45	32	22	16	11	8
Integration Capacity (BT)	2	4	8	16	32	64	128	256
Delay = CV/I scaling	0.7	~0.7	>0.7	Delay scaling will slow down				
Energy/Logic Op scaling	>0.35	>0.5	>0.5	Energy scaling will slow down				
Bulk Planar CMOS	High Probability						Low Probability	
Alternate, 3G etc	Low Probability						High Probability	
Variability	Medium			High			Very High	
ILD (K)	~3	<3	Reduce slowly towards 2-2.5					
RC Delay	1	1	1	1	1	1	1	1
Metal Layers	6-7	7-8	8-9	0.5 to 1 layer per generation				

Shekhar Borkar, Micro37, P

Pipeline (H. Ford)





Fast Core: High Frequencies

- Deep pipelines
- Layered microarchitecture
- Dataflow design, minimum control logic
- Leverage Out-of-Order resiliency
- Aggressive clock distribution

Program Dependences

- ❐ Data dependences

a R_i := …

b := R_i op R_j

c R_i := ...

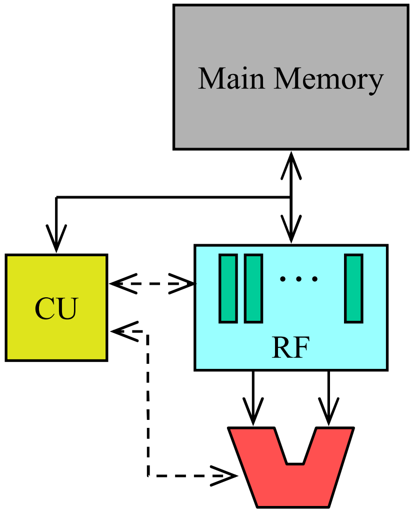
Data dependences

a and b (RAW)

Name dependences

b and c (WAR)

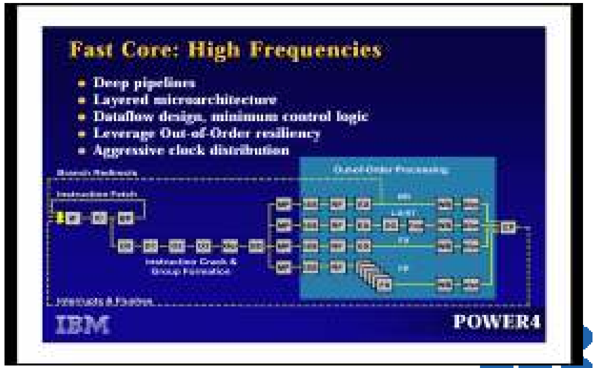
a and c (WAW)



- ❐ Control dependencies

.
.
.
a
.
.
.
b branch (cond.) a
↓
b+1

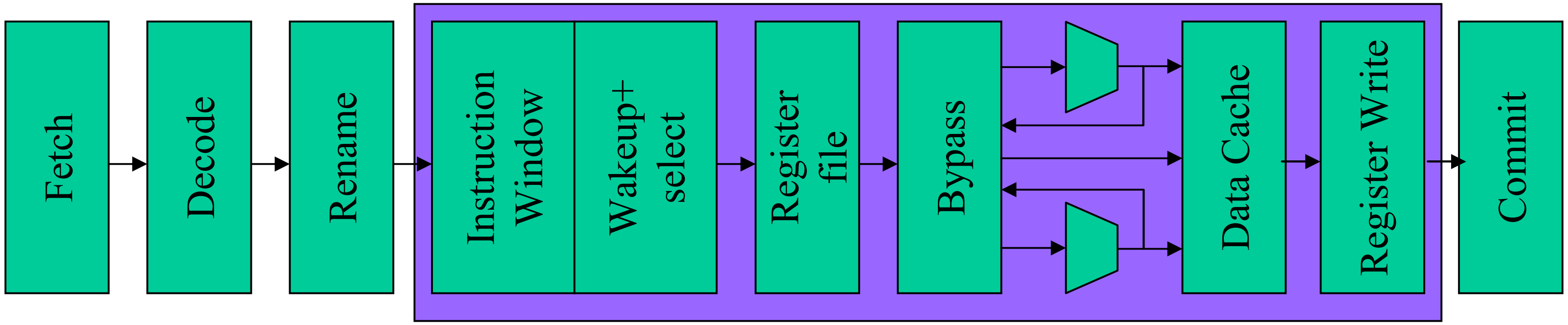
F	D	R	E	W					
	F	D	R	E	W				
		F	D	R	E	W			
			F	D	R	E	W		
				F	D	R	E	W	
					F	D	R	E	W



Fast Core: High Frequencies

- Deep pipelines
- Layered microarchitecture
- Dataflow design, minimum control logic
- Leverage Out-of-Order resiliency
- Aggressive clock distribution

Superscalar Processor



Fetch of multiple instructions every cycle.

Rename of registers to eliminate added dependencies.

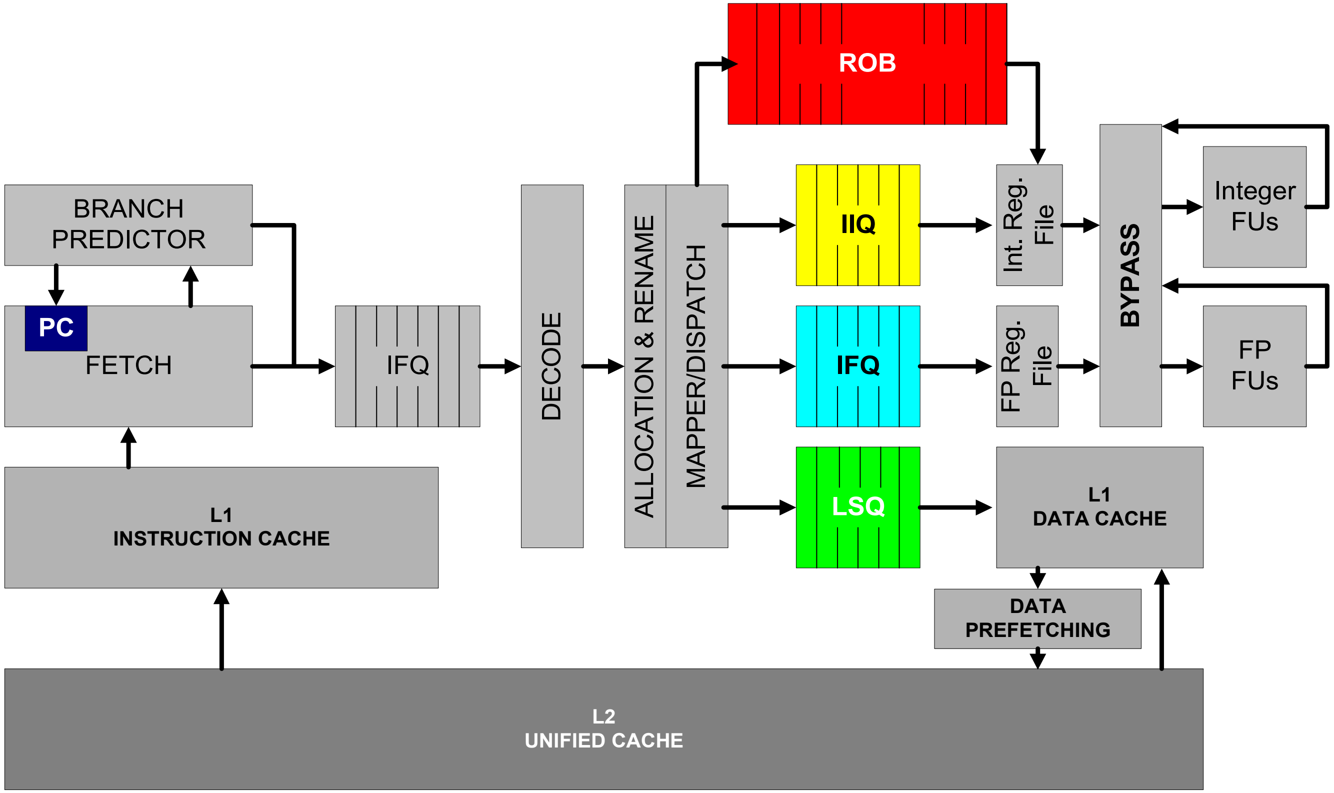
Instructions wait for source operands and for functional units.

Out- of -order execution, but in order graduation.

Predict branches and speculative execution

J.E. Smith and S.Vajapeyam.¨Trace Processors…¨ IEEE Computer.Sept. 1997. pp68-74.

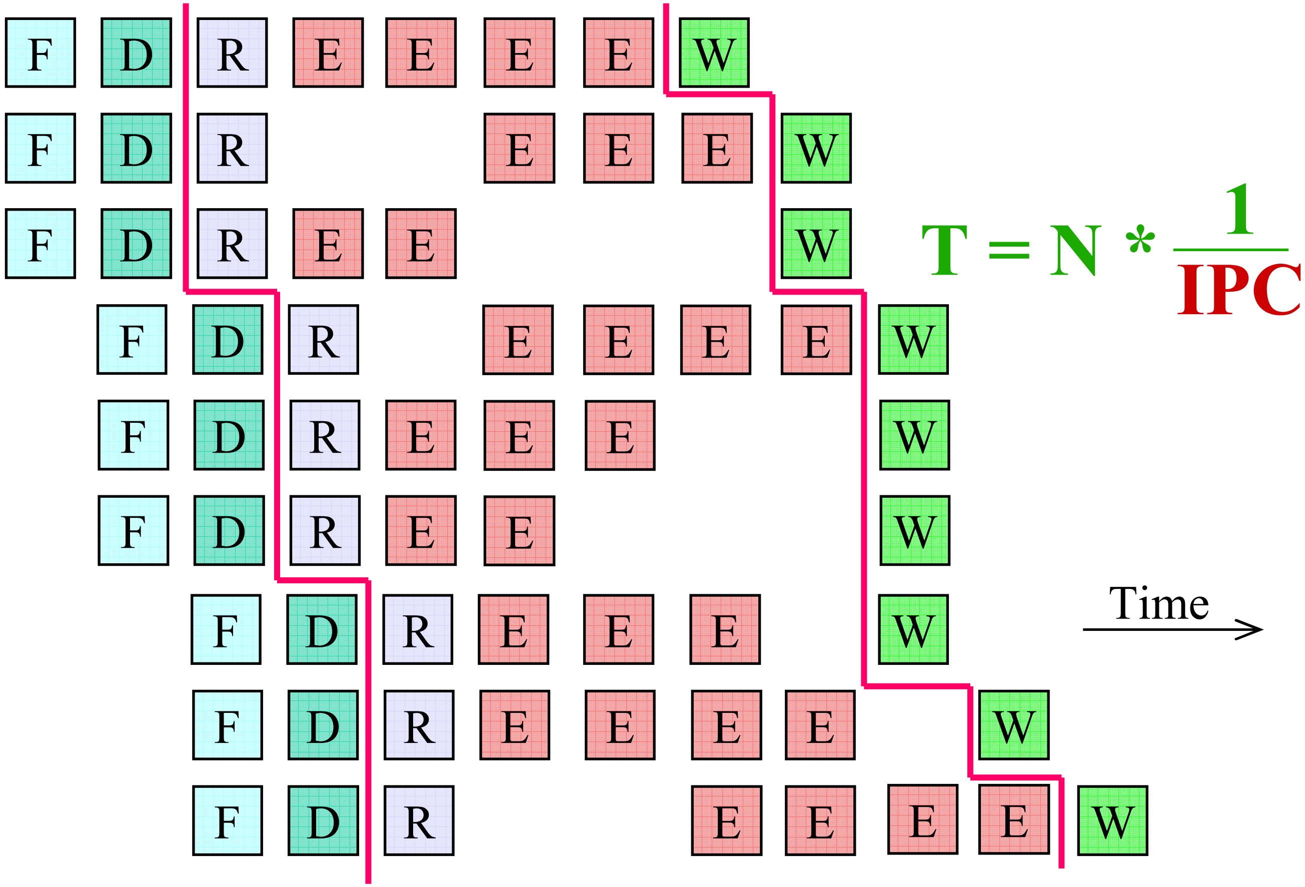
Modern Superscalar Processors



Marco A. Ramírez Salinas, PhD Thesis, Barcelona July 9th, 2007

Superscalar Processors

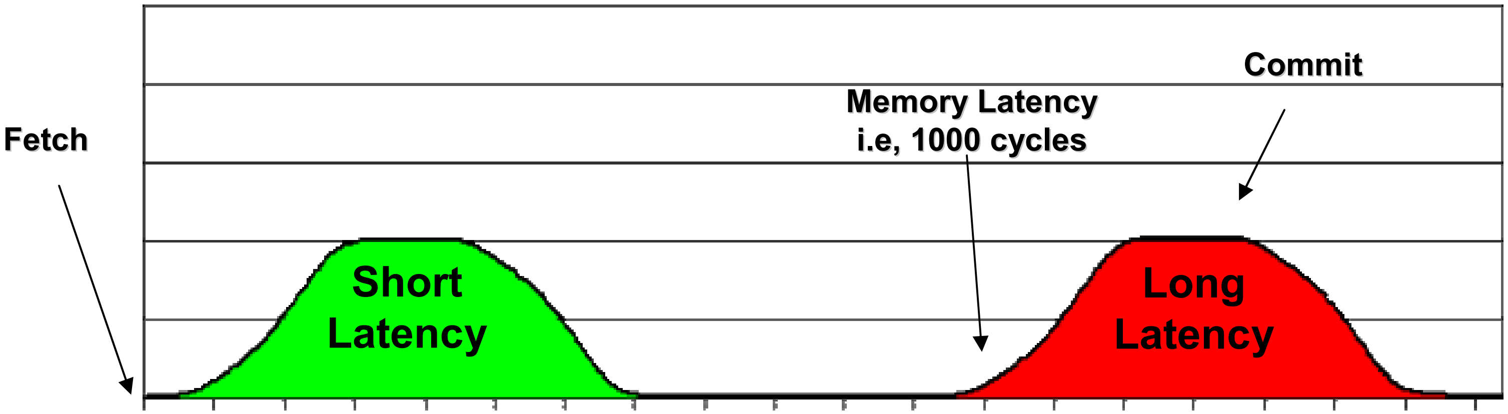
- Out of order (IPC <= 3)



F	D	R	E	E	E	E	W				
F	D	R			E	E	E	W			
F	D	R	E	E				W			
	F	D	R		E	E	E	E	W		
	F	D	R	E	E	E			W		
	F	D	R	E	E				W		
		F	D	R	E	E	E		W		
		F	D	R	E	E	E	E		W	
		F	D	R			E	E	E	E	W

$$T = N * \frac{1}{IPC} * t_c$$

Assignment, use and release of Resources



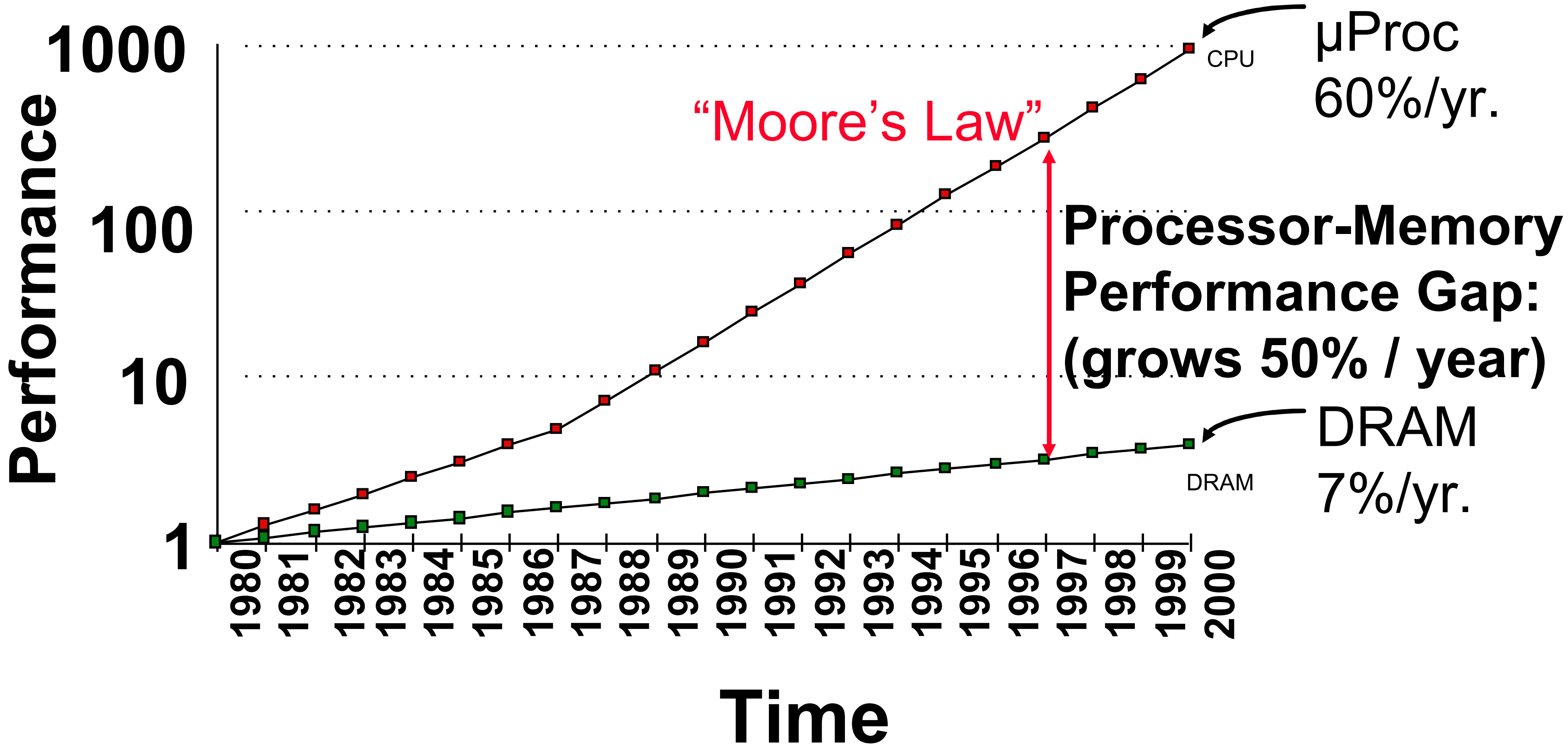
Decode, Renaming		
Resource	*Assignment*	ROB IQ LSQ registers
	Release	IQ(issued)

Commit		
Resource	*Assignment*	---
	Release	ROB IQ LSQ registers

T. Karkhanis and J.Smith, "A day in the life of a data cache miss" Workshop Memory Performance Issues. ISCA-2002

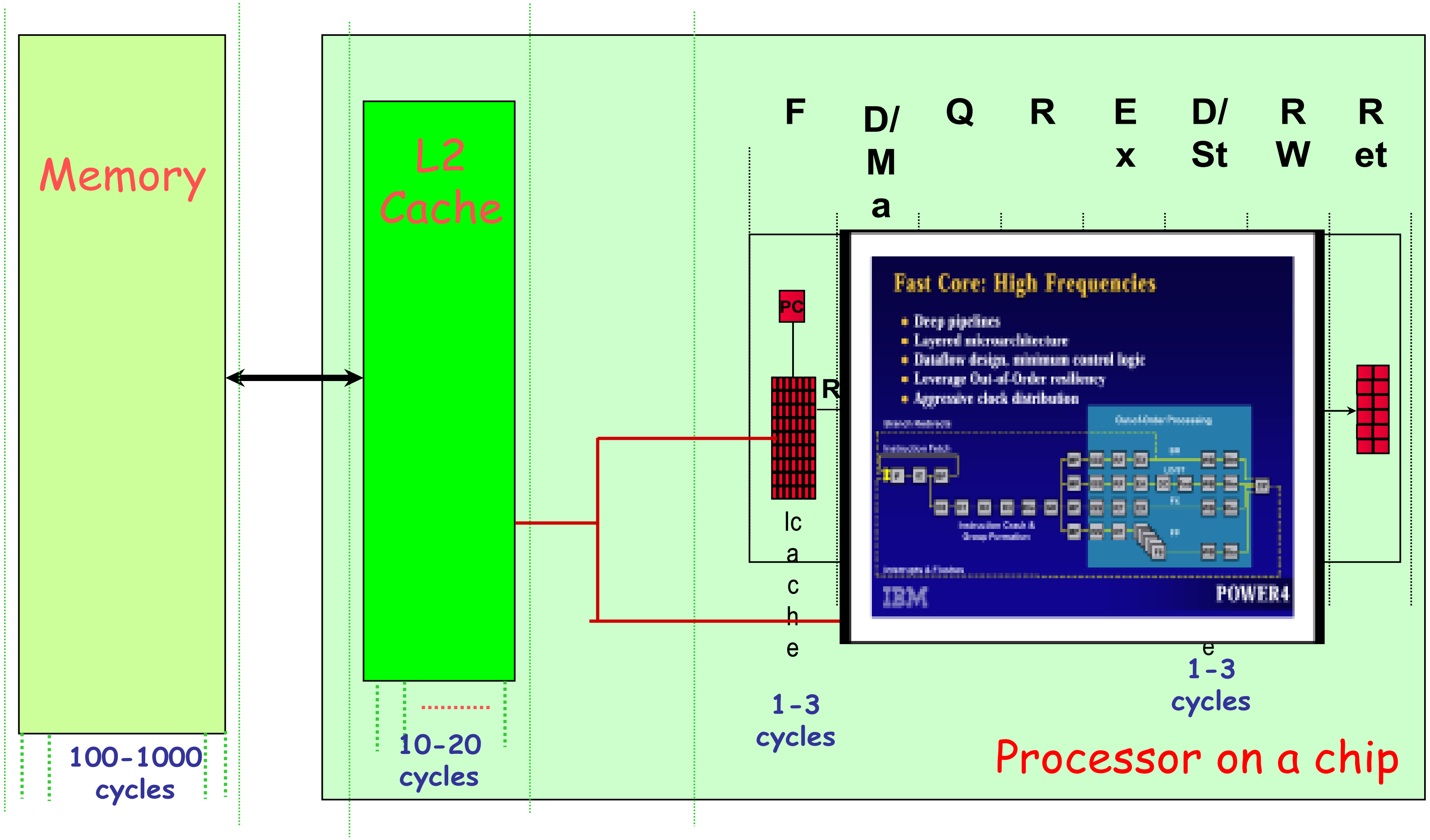
M.Valero NSF Workshop on Computer Architecture. ISCA Conference. San Diego, June 2003

Processor-DRAM Gap (latency)



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

Latencies and Pipelines

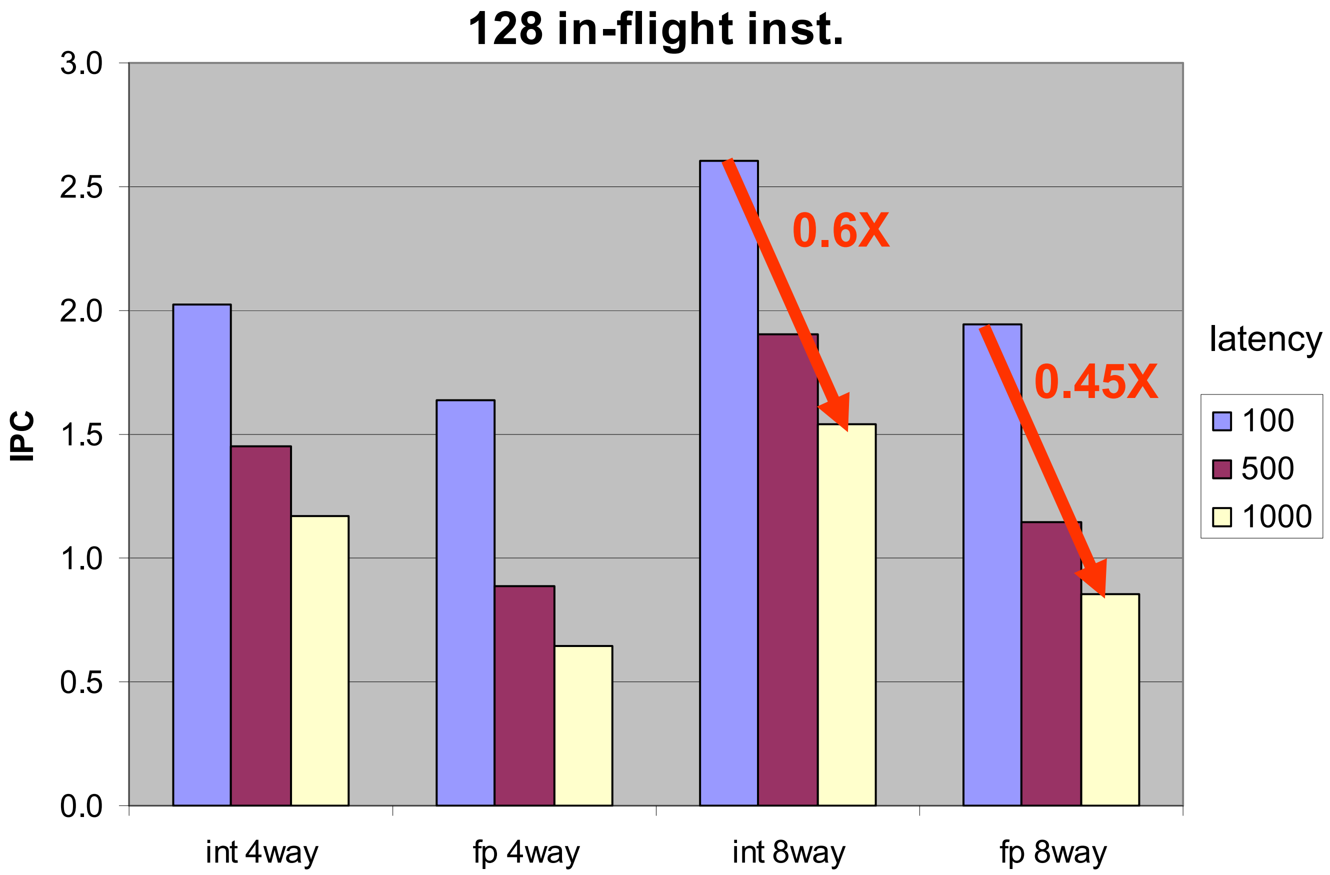


Fast Core: High Frequencies

- Deep pipelines
- Layered microarchitecture
- Dataflow design, minimum control logic
- Leverage Out-of-Order resiliency
- Aggressive clock distribution

Processor on a chip

Memory Wall Problem



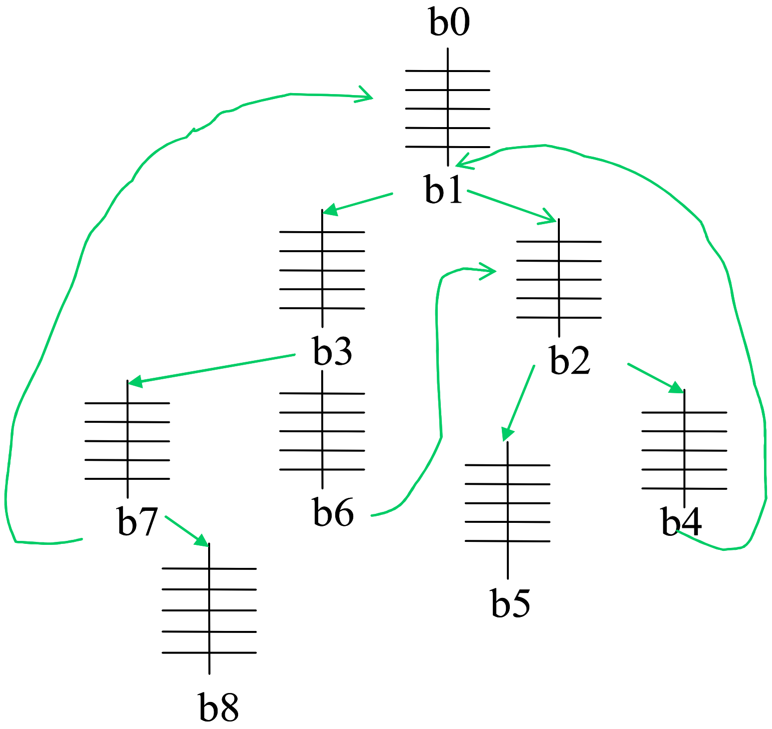
Memory latency has enormous impact on IPC

M. Valero. NSF Workshop on Computer Architecture. ISCA Conference. San Diego, June 2003

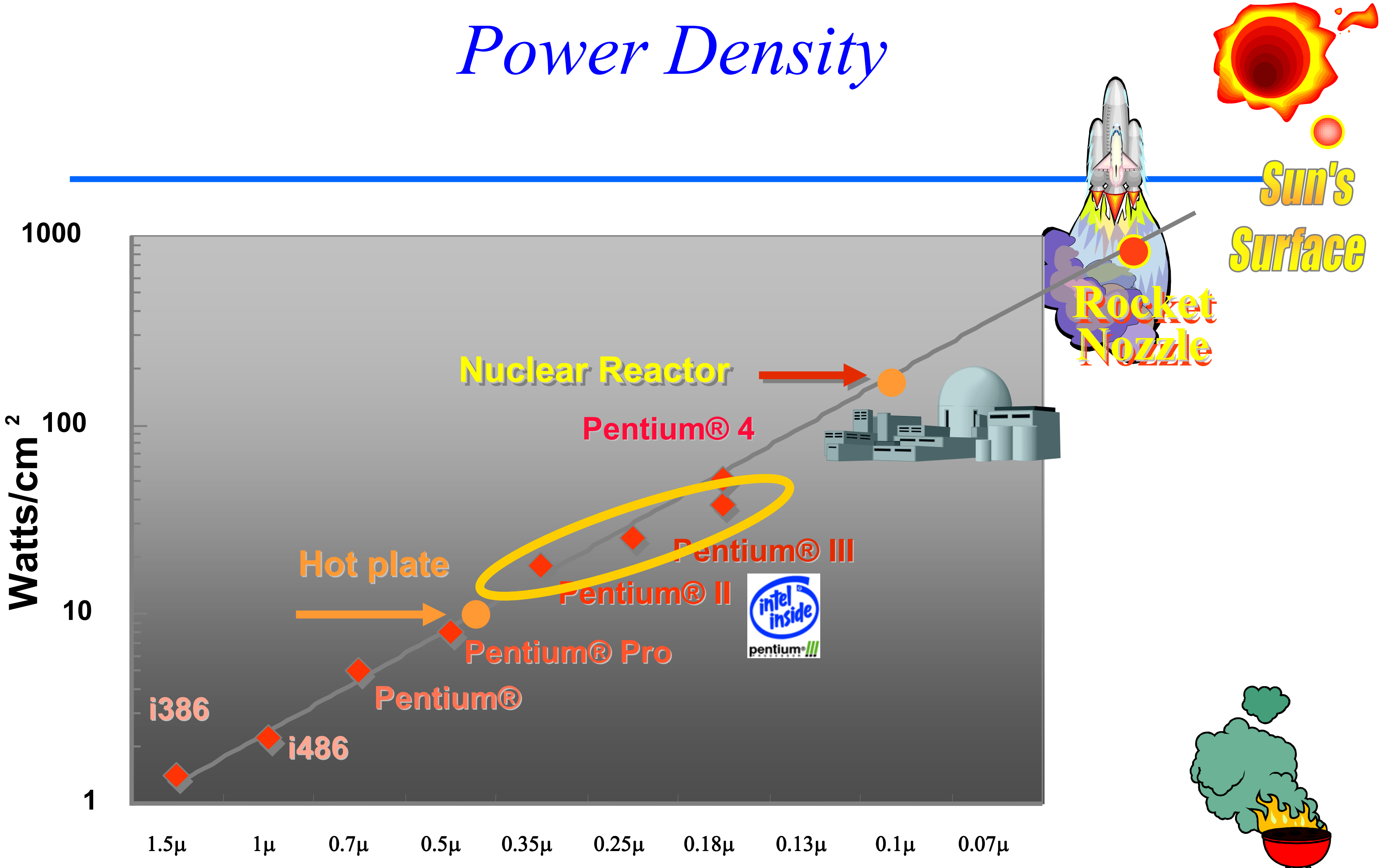
Branch Instructions

Every 5-6 instructions

Limits to high-speed



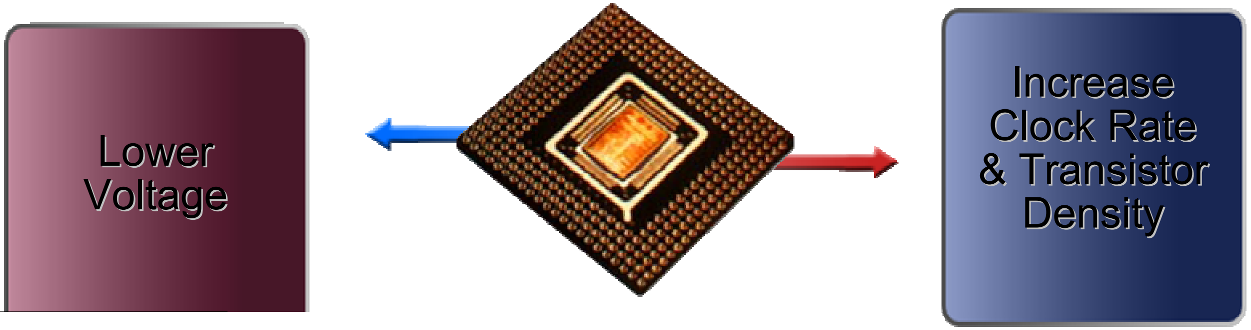
Power Density



* "New Microarchitecture Challenges in the Coming Generations of CMOS Process Technologies" – Fred Pollack, Intel Corp. Micro32 conference keynote - 1999.

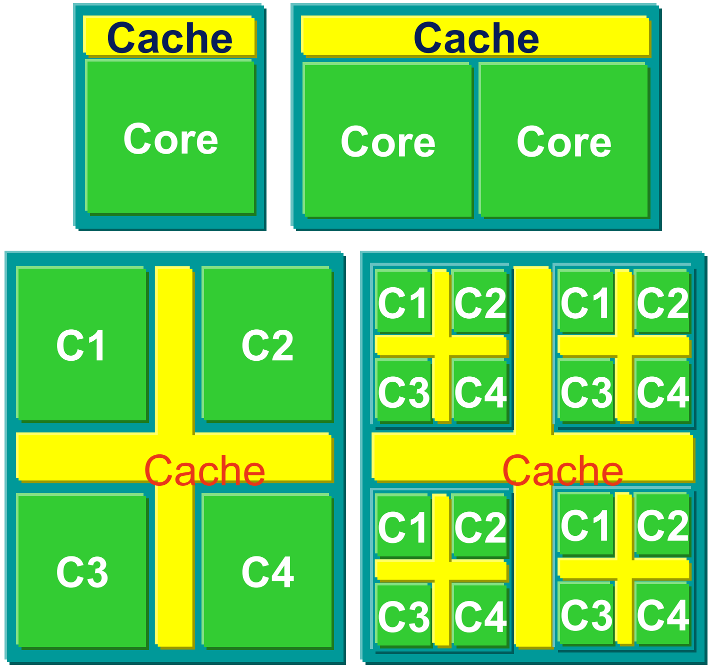
Increasing CPU performance: a delicate balancing act

Increasing the number of gates into a tight knot and decreasing the cycle time of the processor



Lower Voltage

Increase Clock Rate & Transistor Density



We have seen increasing number of gates on a chip and increasing clock speed.

Heat becoming an unmanageable problem, Intel Processors > 100 Watts

We will not see the dramatic increases in clock speeds in the future.

However, the number of gates on a chip will to increase.

Reducing Memory Latency

- Technology
- Caches
- Prefetching
    - Hardware, Software and combined
- Assisted/SSMT Threads
- Runahead Processors
- ………

- "Kilo-instruction" Processor

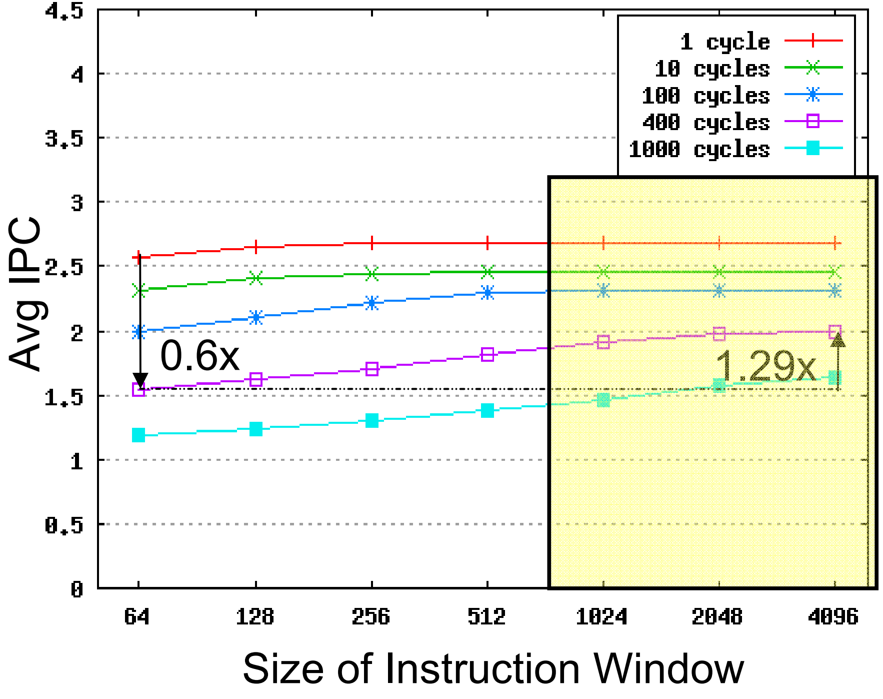
Motivation: The Memory Wall and KIPs

- Since the 80s processor frequencies have accelerated about 40% every year.
- However, main memory access time has decreased much slower.

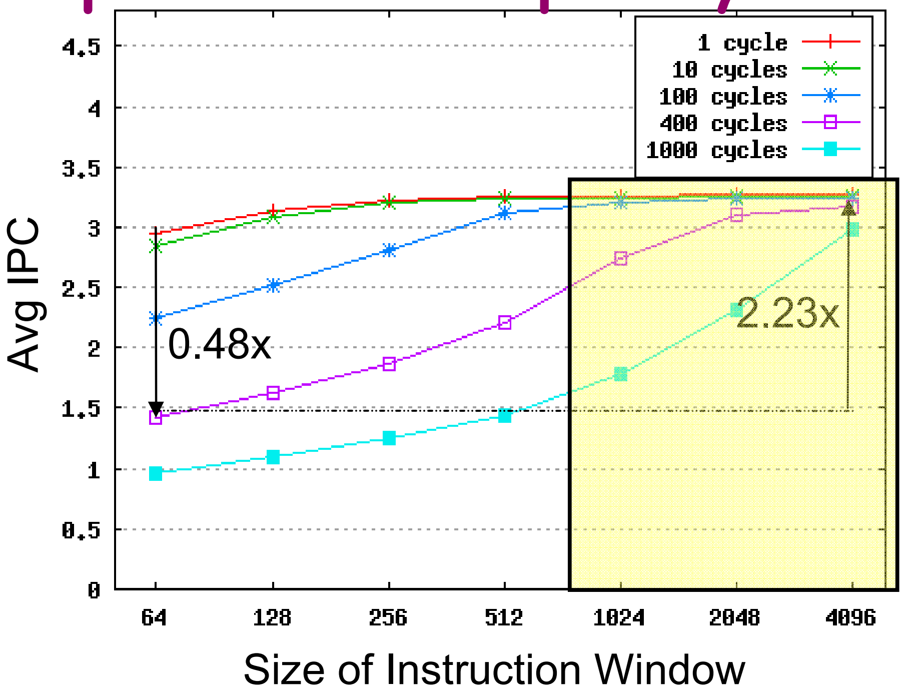
KILO-Instruction Processors (KIPs)

Cristal et al. (1st Proposal to Intel, 2001) + many later works

KILO-Instruction Processors can overcome the Memory Wall for many codes, but how do we design them to be efficient in power and complexity?



SPECINT2000



SPECFP2000

“Kilo-instruction” Processors

- Our goals
  - Better tolerate increasing memory latency
  - Further improve ILP, even for such longer memory latency
  - Allow additional optimizations enabled by the new architecture (See below)

- Our proposal: “Kilo-instruction” Processors
  - Out-Of-Order processors with thousands of instructions in-flight (Very Large Instruction Windows)
  - Intelligent use of resources (Resource requirements growing much slower than window size)

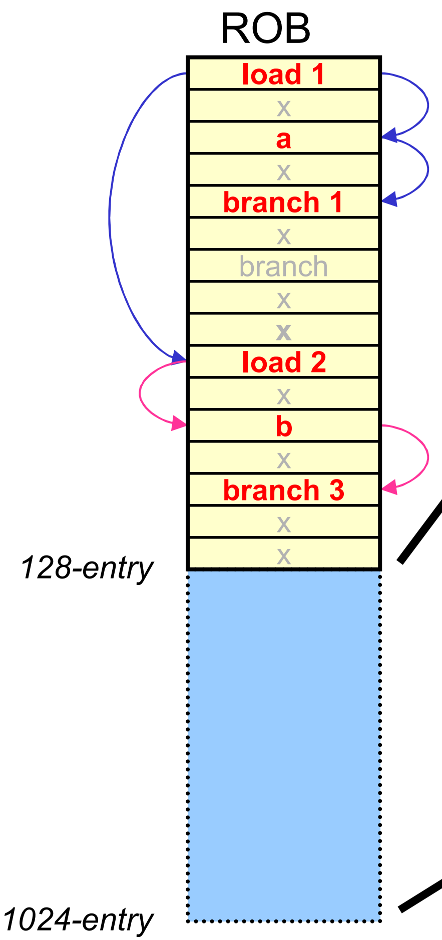
“Kilo-instruction Processsor”

- ❒ It is not…..
  - ❒ A “heavy” processor ☺
  - ❒ Cyber-205 like processor
  - ❒ Vector Processor
  - ❒ Blue-Gene like
  - ❒ Multiscalar,Trace Processor
  - ❒ Raw, Imagine, Levo,TRIPS
- ❒ It is …….
  - ❒ An Affordable O-O-O Superscalar Processor having “Thousands of In-flight Instructions”

Outline

- Motivation
- Increasing the number of in-flight instructions
- Kilo-instruction Processor Ingredients:
  - Multi-Checkpointing the ROB
    - Out-of-Order Commit
  - Early Release of Resources
    - Ephemeral Registers
    - Load Queues
  - Locality Exploitation
    - Instruction Queues
    - LSQ
- Affordable KILO-Instruction Processors
- Cross-pollination with other techniques:
  - "Kilo-processor" and multiprocessor systems
  - "kilo-vector" processors and "kilo-valpred" processors
  - "Kilo-SMT" processor
  - Further Improvements:
    - Branch prediction
    - Control Independence
    - Reuse
    - Predicated and multipath execution
- Conclusion

ROB Activity



ROB

load 1

x

a

x

branch 1

x

branch

x

x

load 2

x

b

x

branch 3

x

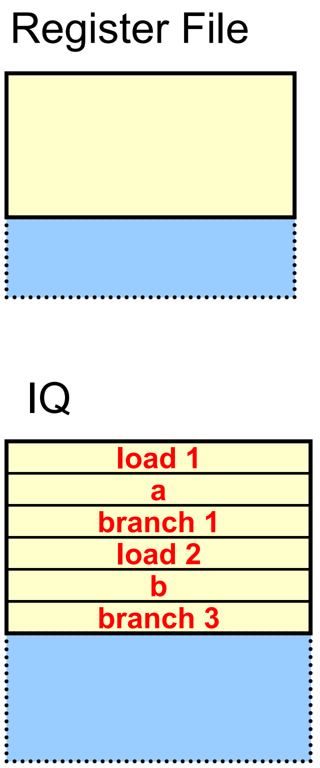
x

128-entry

1024-entry

ROB full:		
	Lat=100	Lat=1000
int	62%	72%
fp	76%	90%

ROB full:		
	Lat=100	Lat=1000
int	30%	48%
fp	50%	75%



Register File

IQ

load 1

a

branch 1

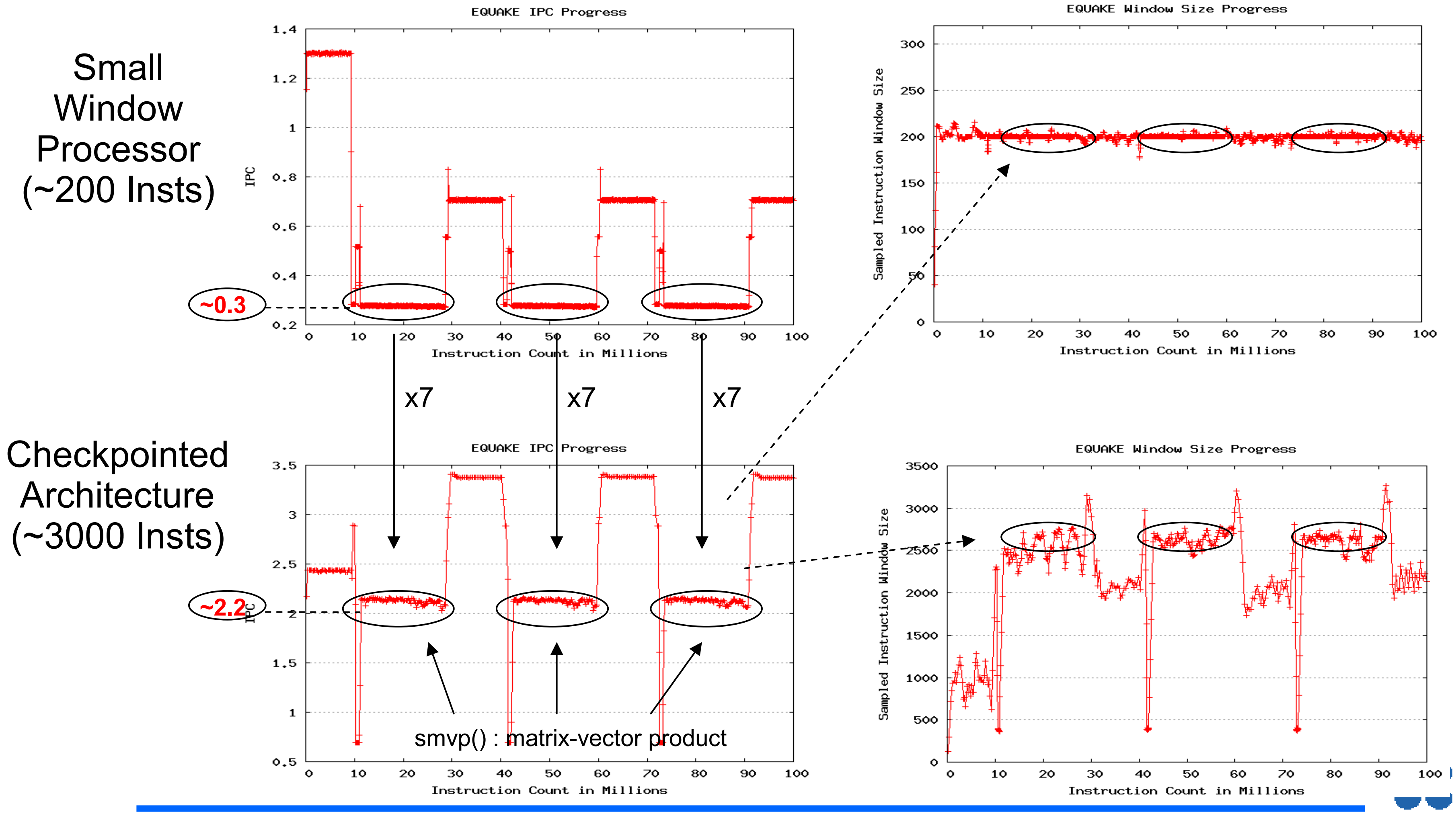
load 2

b

branch 3

M. Valero. NSF Workshop on Computer Architecture. ISCA Conference. San Diego, June 2003

Effect of Window Size on Performance (EQUAKE)



Small Window Processor (~200 Insts)

Checkpointed Architecture (~3000 Insts)

~0.3

~2.2

x7

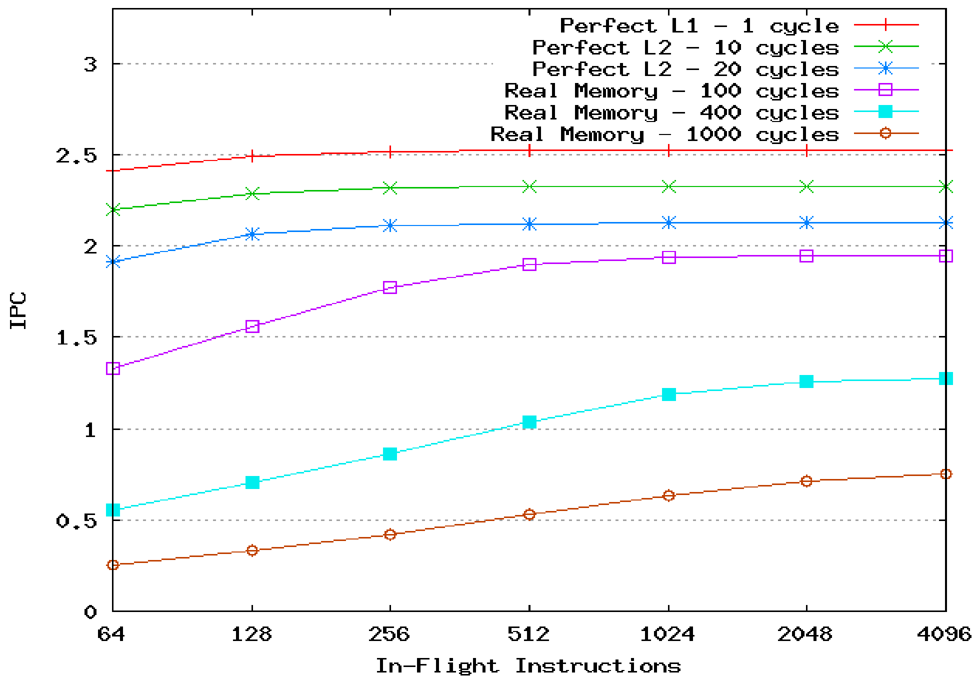
x7

x7

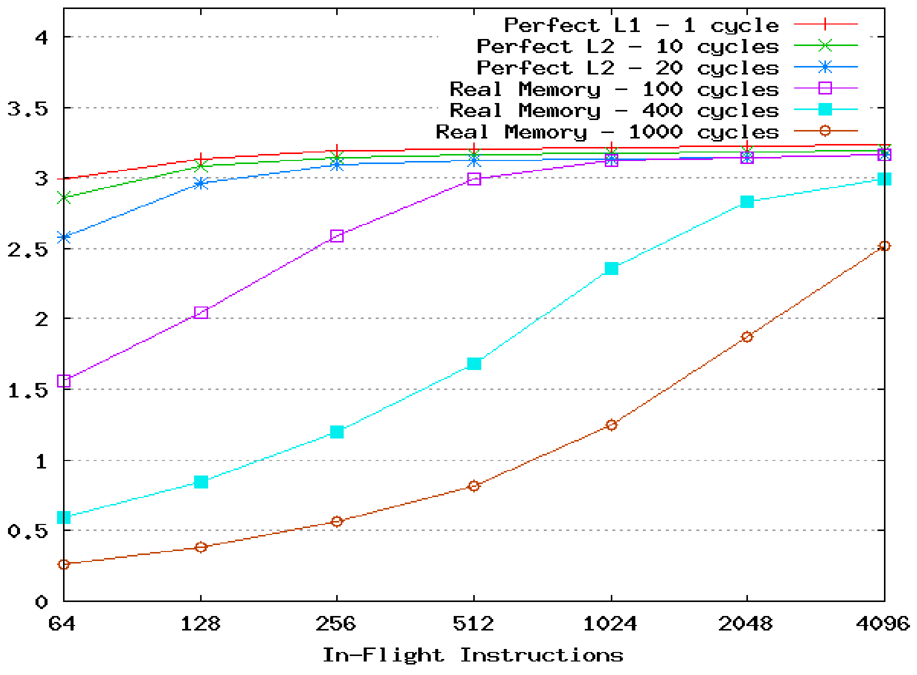
smvp() : matrix-vector product

Impact of Memory Wall

The memory wall has a big impact on the performance on current hardware. Large-window processors can overcome this problem for many codes, but technology constraints limit the scalability of current designs



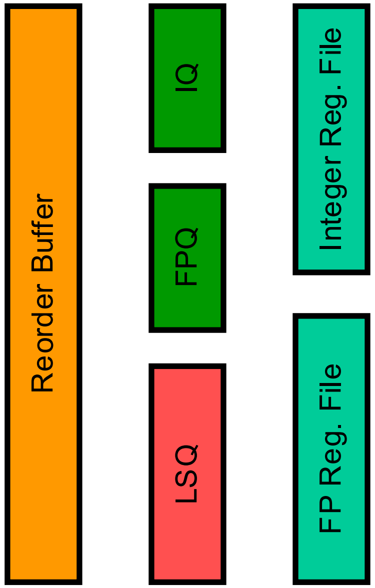
SPECINT2000



SPECFP2000

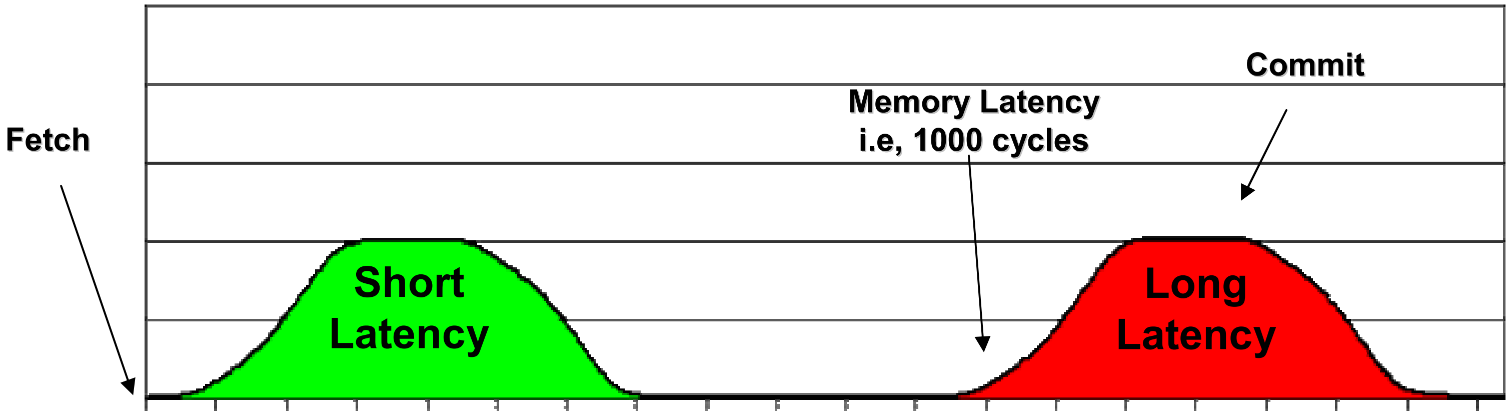
Scalability

- 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 without drastically increasing resource requirements



M. Valero. NSF Workshop on Computer Architecture. ISCA Conference. San Diego, June 2003

Early Release of Resources



Decode, Renaming		
Resource	*Assignment*	ROB IQ LSQ registers
	Release	IQ(issued)

Commit		
Resource	*Assignment*	---
	Release	ROB IQ LSQ registers

T. Karkhanis and J.Smith, “A day in the life of a data cache miss” Workshop Memory Performance Issues. ISCA-2002

M.Valero NSF Workshop on Computer Architecture. ISCA Conference, San Diego, June 2003

Outline

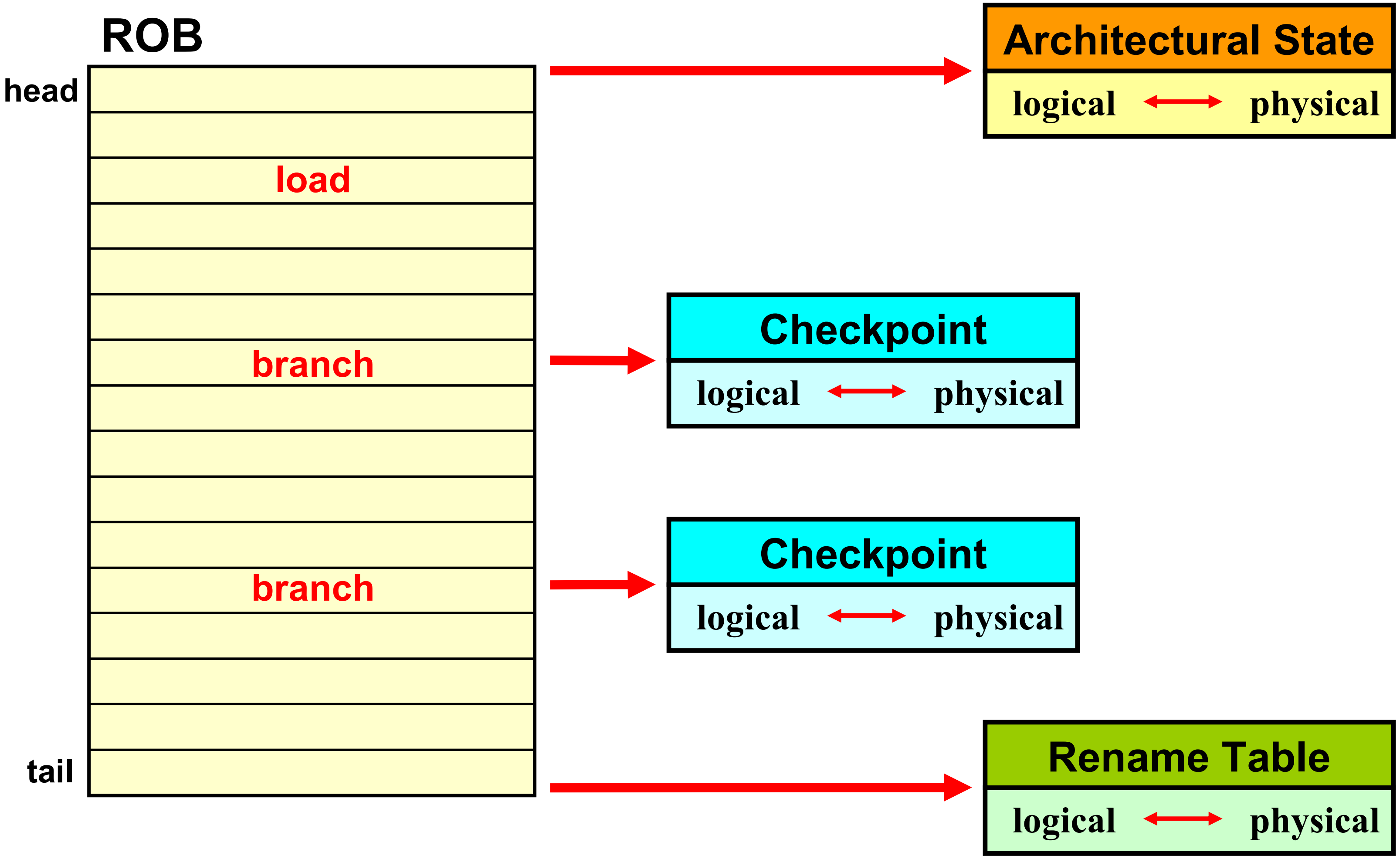
- Motivation
- Increasing the number of in-flight instructions
- Kilo-instruction Processor Ingredients:
  - Multi-Checkpointing the ROB
    - Out-of-Order Commit
  - Early Release of Resources
    - Ephemeral Registers
    - Load Queues
  - Locality Exploitation
    - Instruction Queues
    - LSQ
- Affordable KILO-Instruction Processors
- Cross-pollination with other techniques:
  - "Kilo-processor" and multiprocessor systems
  - "kilo-vector" processors and "kilo-valpred" processors
  - "Kilo-SMT" processor
  - Further Improvements:
    - Branch prediction
    - Control Independence
    - Reuse
    - Predicated and multipath execution
- Conclusion

Checkpointing the ROB

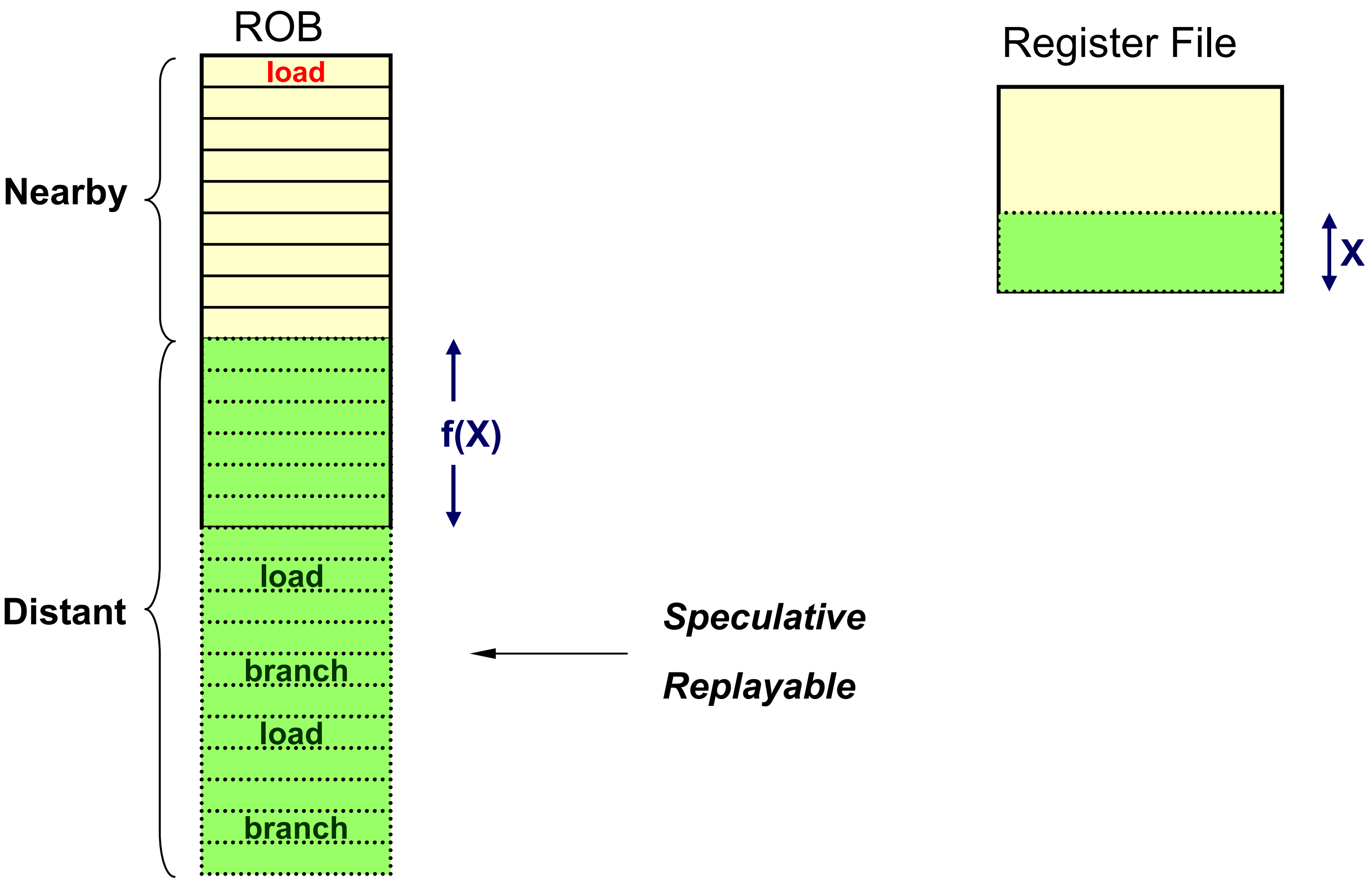
- Checkpointing to support precise exceptions:
  - Quite well established and used technique
    - J.E. Smith and A.R. Pleszkun, ISCA 1985
    - W.M.Hwu and Y.N.Patt, ISCA 1987
- Checkpointing to early release resources:
  - Quite recent concept
    - Cherry: J. Martínez et al., MICRO, Nov. 2002
    - Large VROB: A. Cristal et al. TR-UPC-DAC, July 2002

M. Valero. NSF Workshop on Computer Architecture. ISCA Conference. San Diego, June 2003

Checkpointing Example: MIPS Like



Nearby & Distant Parallelism

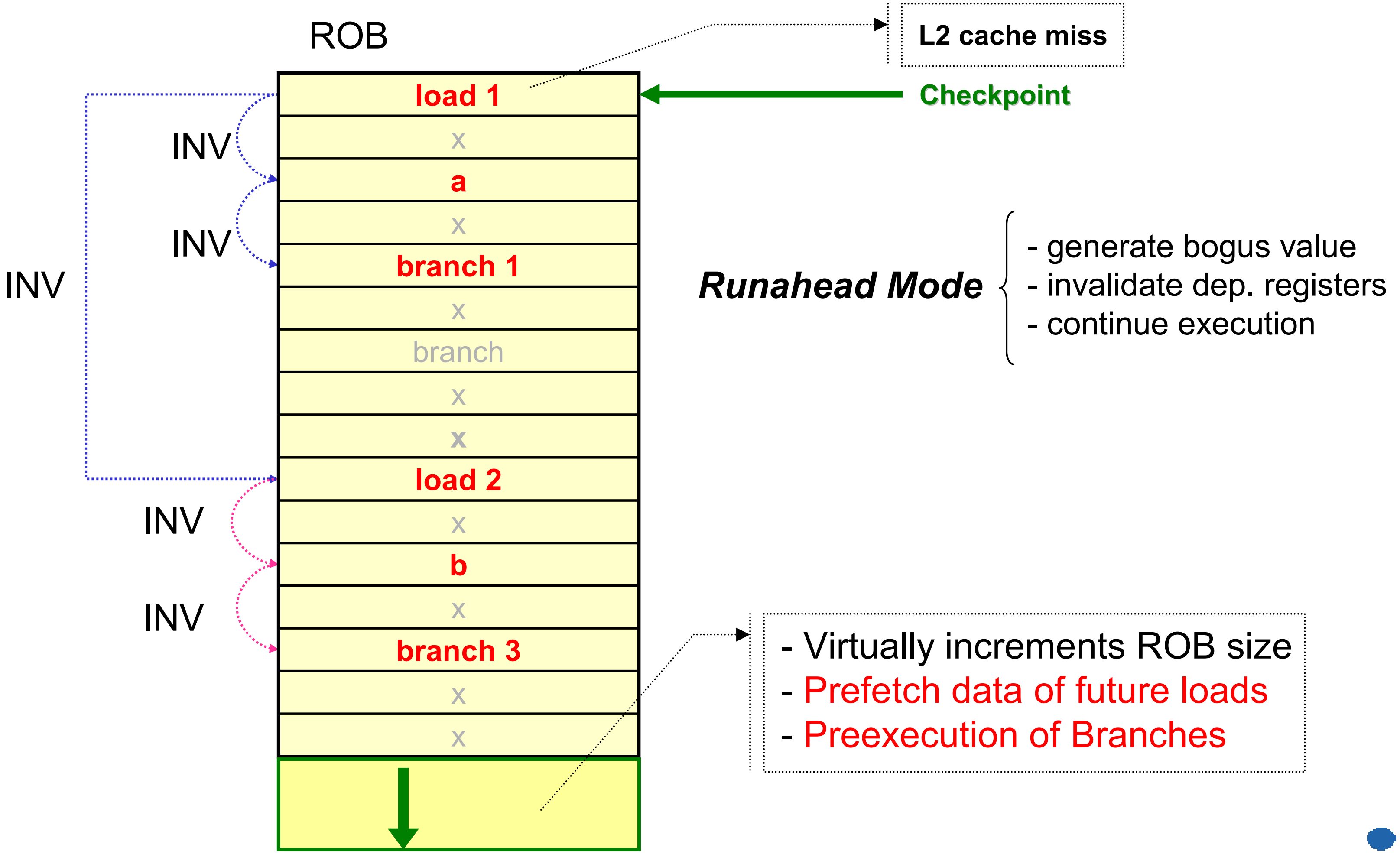


Speculative

Replayable

Balasubramonian et al.: "Dynamically Allocating Processor Resources…", ISCA'01

Runahead Execution



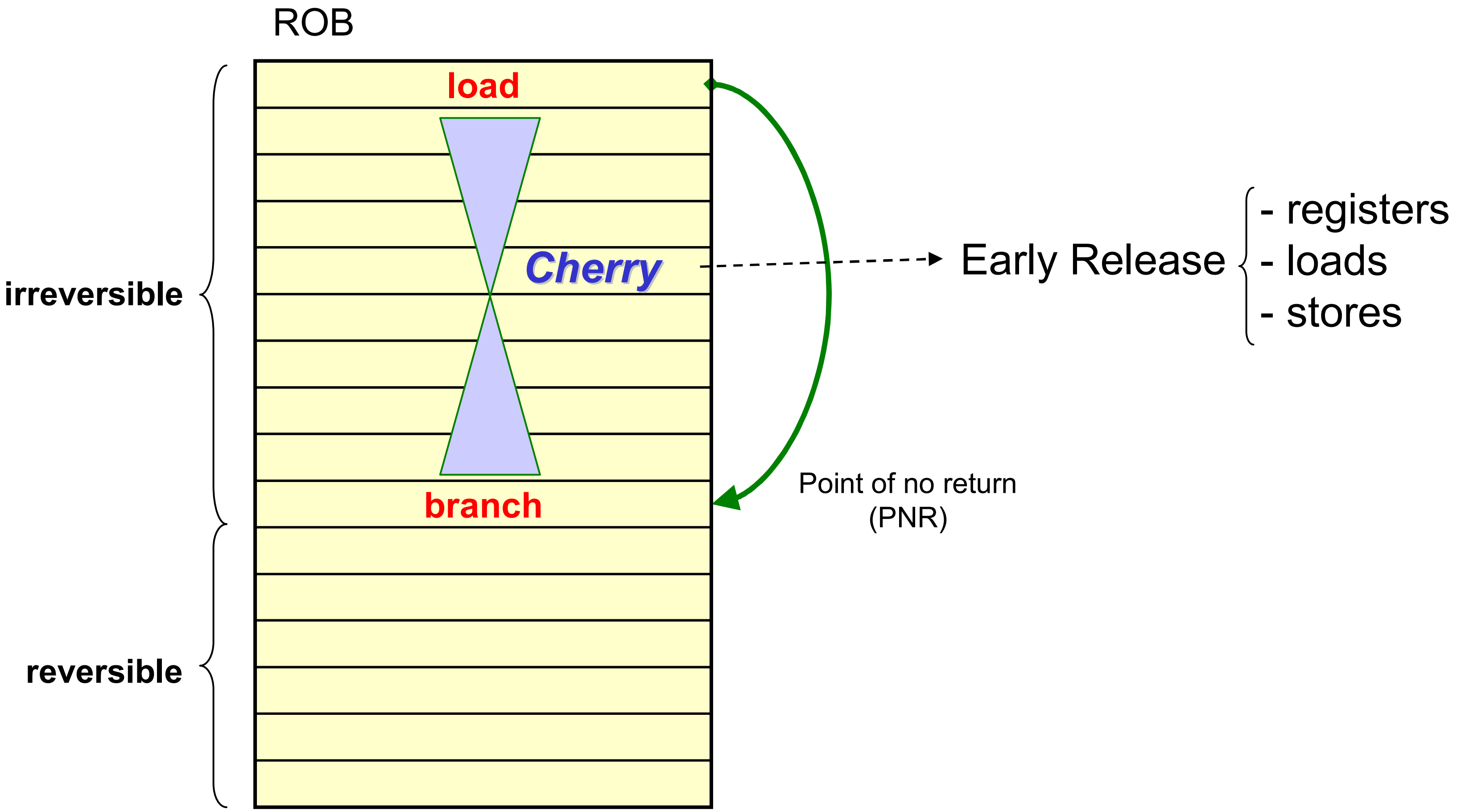
Runahead Mode

- generate bogus value
- invalidate dep. registers
- continue execution

- Virtually increments ROB size
- Prefetch data of future loads
- Preexecution of Branches

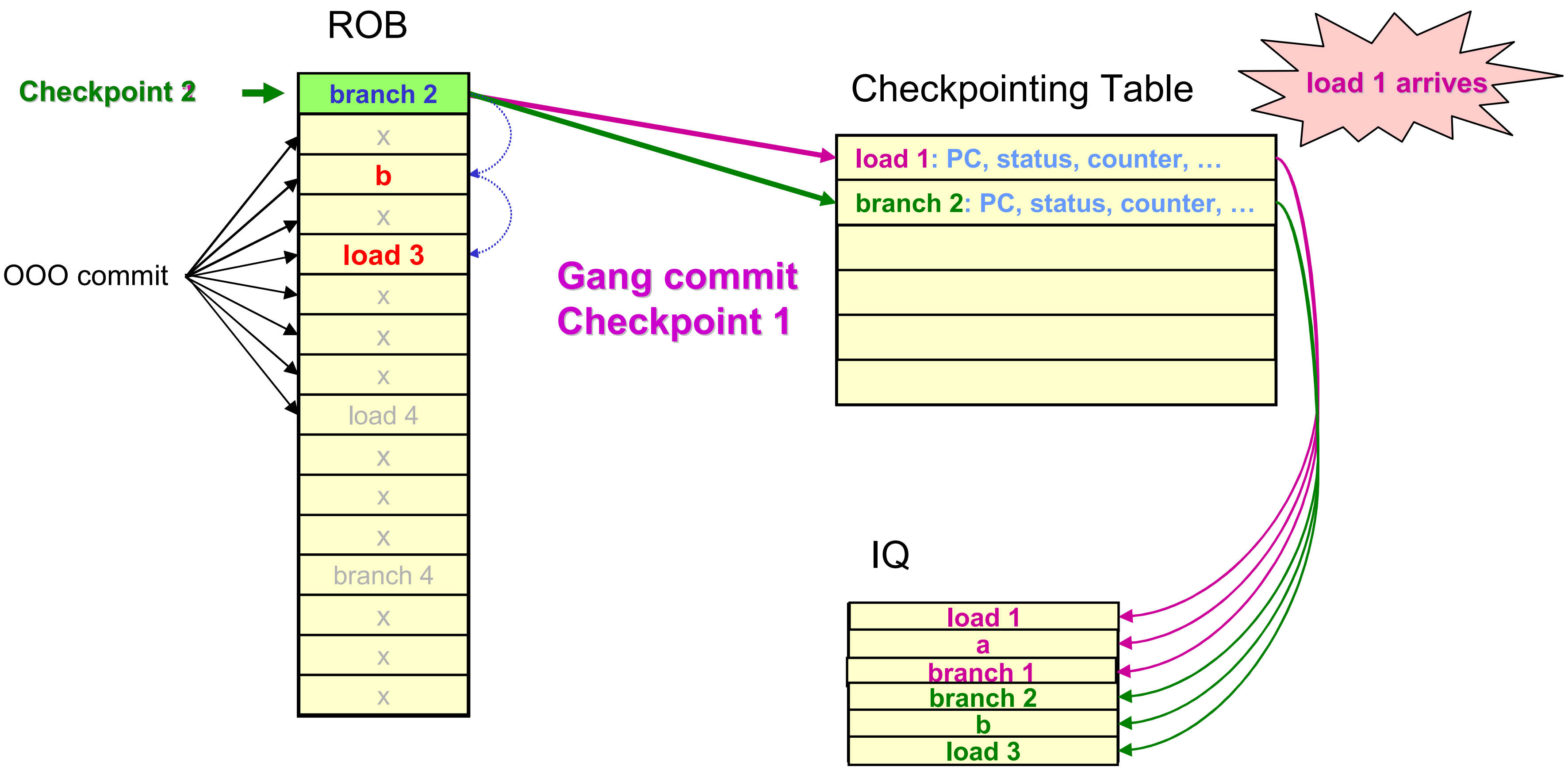
Mutlu et al.: "Runahead Execution: An Alternative…", HPCA'03

Cherry



Martínez et al.: “Cherry: Checkpointed Early Resource Recycling…”, MICRO’02

Multi-Checkpoint



ROB

Checkpoint 2

branch 2

x

b

x

load 3

x

x

x

load 4

x

x

x

branch 4

x

x

x

OOO commit

Gang commit
Checkpoint 1

Checkpointing Table

load 1: PC, status, counter, …

branch 2: PC, status, counter, …

load 1 arrives

IQ

load 1

a

branch 1

branch 2

b

load 3

Cristal et al.: "Large Virtual ROBs by Processor Checkpointing", TR UPC-DAC, July 2002
Research Proposal to Intel (January 2002) and presentation to Intel-MRL Feb. 2002

Checkpointing Description

- How many in-flight checkpoints should be supported?
- What kind of instructions should be checkpointed?
- How often should a checkpoint be taken?
- How much information should be kept?

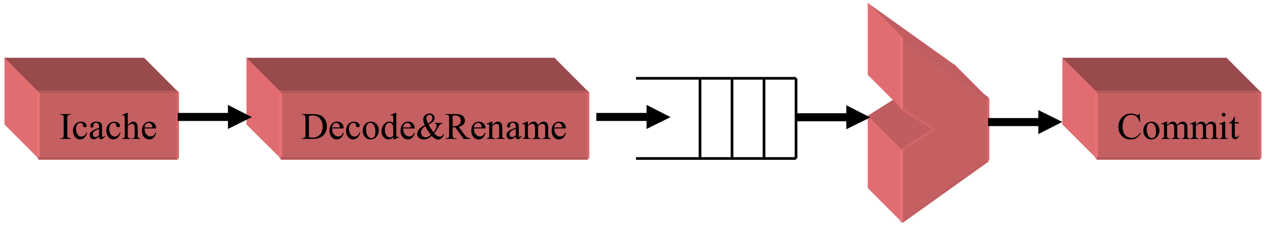
Outline

- Motivation
- Increasing the number of in-flight instructions
- Kilo-instruction Processor Ingredients:
  - Multi-Checkpointing the ROB
    - Out-of-Order Commit
  - Early Release of Resources
    - Ephemeral Registers
    - Load Queues
  - Locality Exploitation
    - Instruction Queues
    - LSQ
- Affordable KILO-Instruction Processors
- Cross-pollination with other techniques:
  - "Kilo-processor" and multiprocessor systems
  - "kilo-vector" processor and "kilo-valpred" processors
  - "Kilo-SMT" processor
  - Further Improvements:
    - Branch prediction
    - Control Independence
    - Reuse
    - Predicated and multipath execution
- Conclusion

Registers

- Register File is a critical component of a modern superscalar processor
  - Large number of entries to support out-of-order execution and memory latency
  - Large number of ports to increase issue width
- Power and access time are key issues for register file design
- It is always beneficial, to reduce the number of physical registers

Physical Registers



- Conventional renaming scheme

🟥 Register Unused | 🟩 Register Used | 🟥 Register Unused

- Virtual-Physical Registers

🟩 Register Used | 🟥 Register Unused

- Early Release

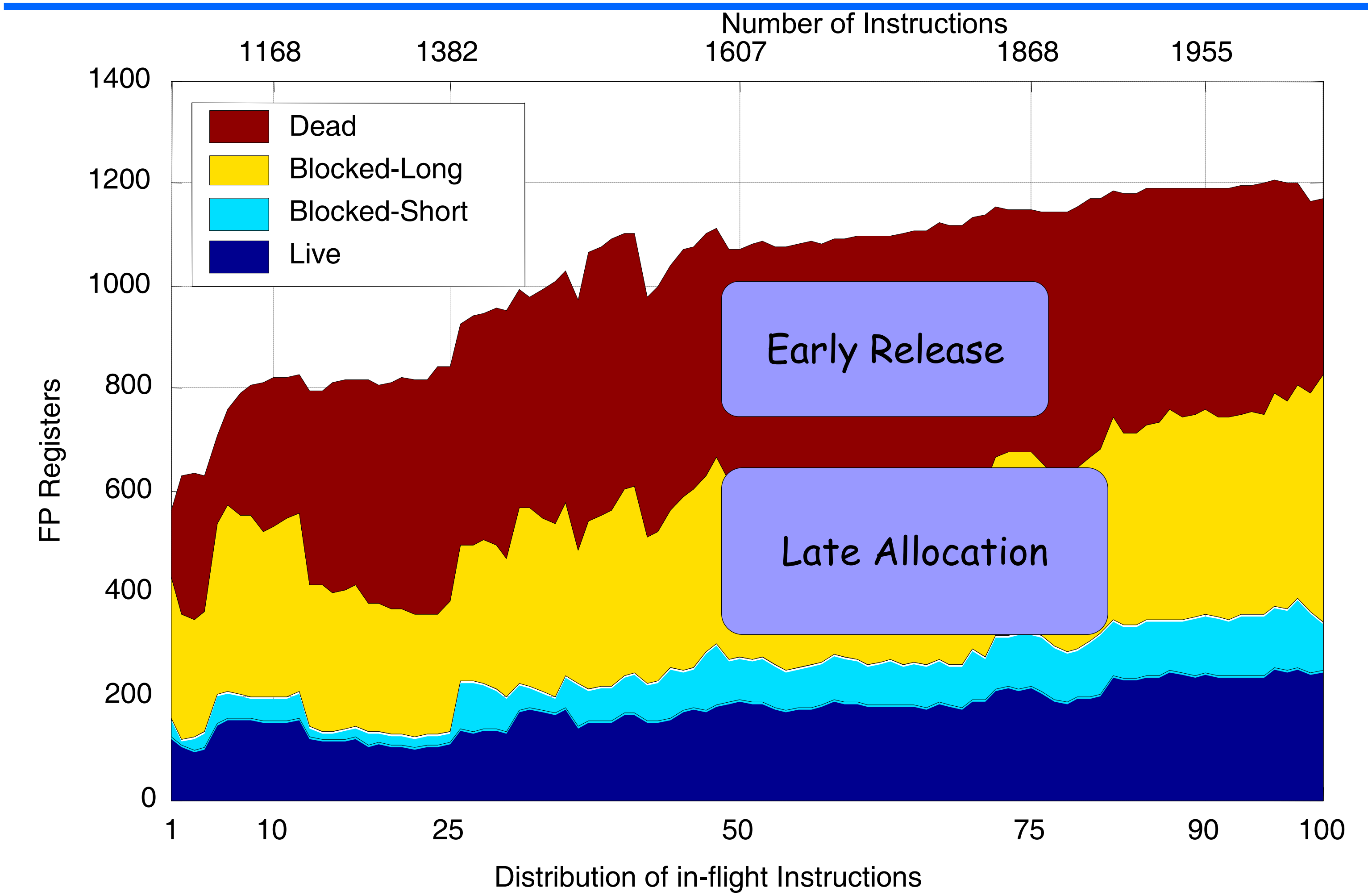
🟥 Register Unused | 🟩 Register Used

- Ephemeral Registers: checkpoint + virtual-physical

🟩 Register Used

T. Monreal et al.: "Delaying physical register allocation through virtual-physical registers", MICRO'99
M. Moudgill et al, "Register renaming and dynamic speculation: an alternative approach", MICRO93
T. Monreal et al., "Late allocation and early release of physical registers", IEEE-TC (to appear)
J. Martínez et al, "Ephemeral Registers", *Technical Report CSL-TR-2003-1035* , 2003
A. Cristal et al, "Ephemeral Registers with Multicheckpointing" Technical report UPC-DAC-2003-51, Oct 2003

State of Registers (FP, ROB=2048)



A. Cristal, et al, “ A case for resource-concious out-of-order processors”, IEEE TCCA CA Letters, Vol. 2, Oct. 2003

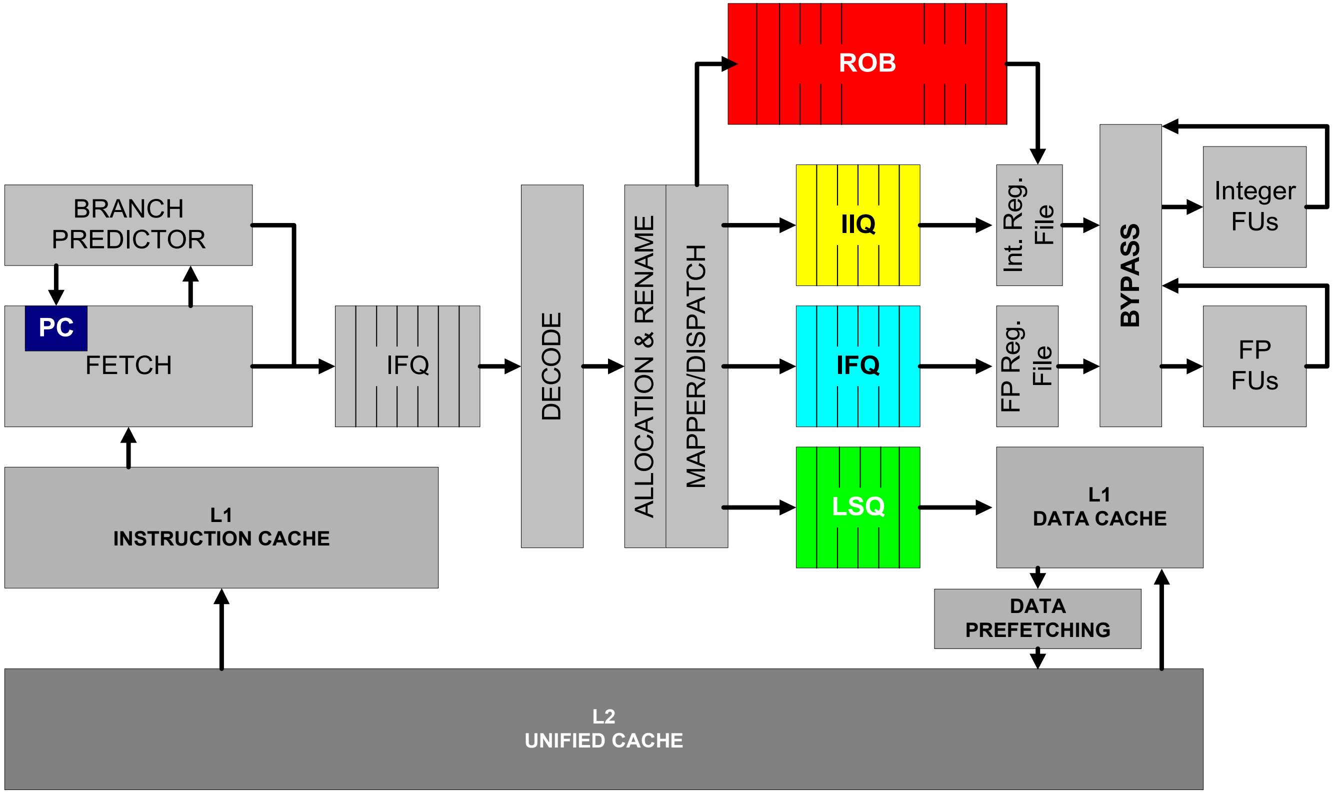
Outline

- Motivation
- Increasing the number of in-flight instructions
- Kilo-instruction Processor Ingredients:
  - Multi-Checkpointing the ROB
    - Out-of-Order Commit
  - Early Release of Resources
    - Ephemeral Registers
    - Load Queues
  - Locality Exploitation
    - Instruction´s Queues
    - LSQ
- Affordable KILO-Instruction Processors
- Cross-pollination with other techniques:
  - "Kilo-processor" and multiprocessor systems
  - "kilo-vector" processor and "kilo-valpred" processors
  - "Kilo-SMT" processor
  - Further Improvements:
    - Branch prediction
    - Control Independence
    - Reuse
    - Predicated and multipath **execution**
- Conclusion

Issue Queues

- Increasing the number of IQ entries increase the power, area and access time
- Wake-up and selection logic need to be done efficiently
- “Kilo-instruction” processors may have many “in-flight” instructions
- We need new organization for the IQs in order to have affordable “kilo-instruction processors”

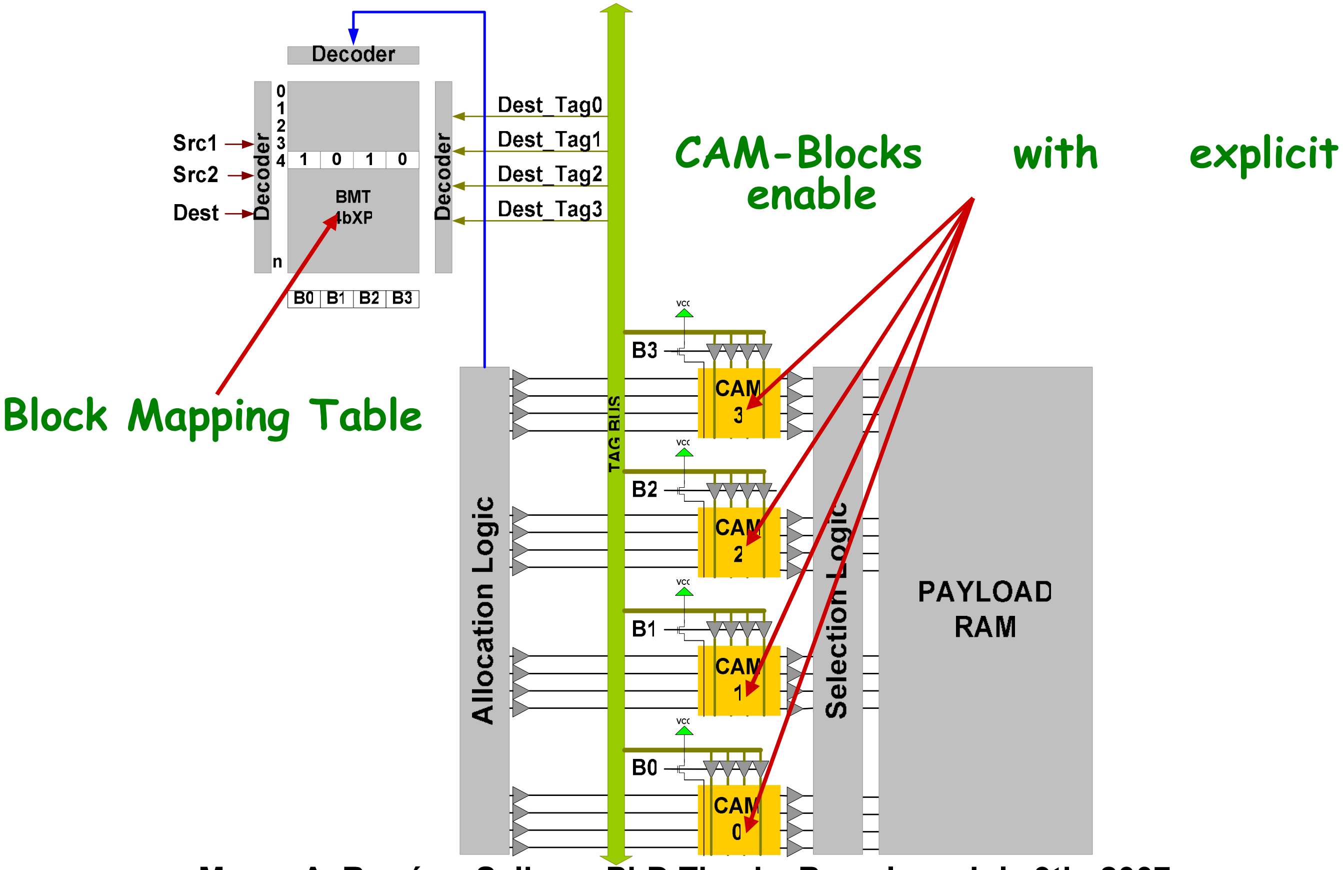
Instruction Queues



Marco A. Ramírez Salinas, PhD Thesis, Barcelona July 9th, 2007

A new low power wakeup mechanism

Each in-flight instruction has an only identifier namely the Destination Register Tag

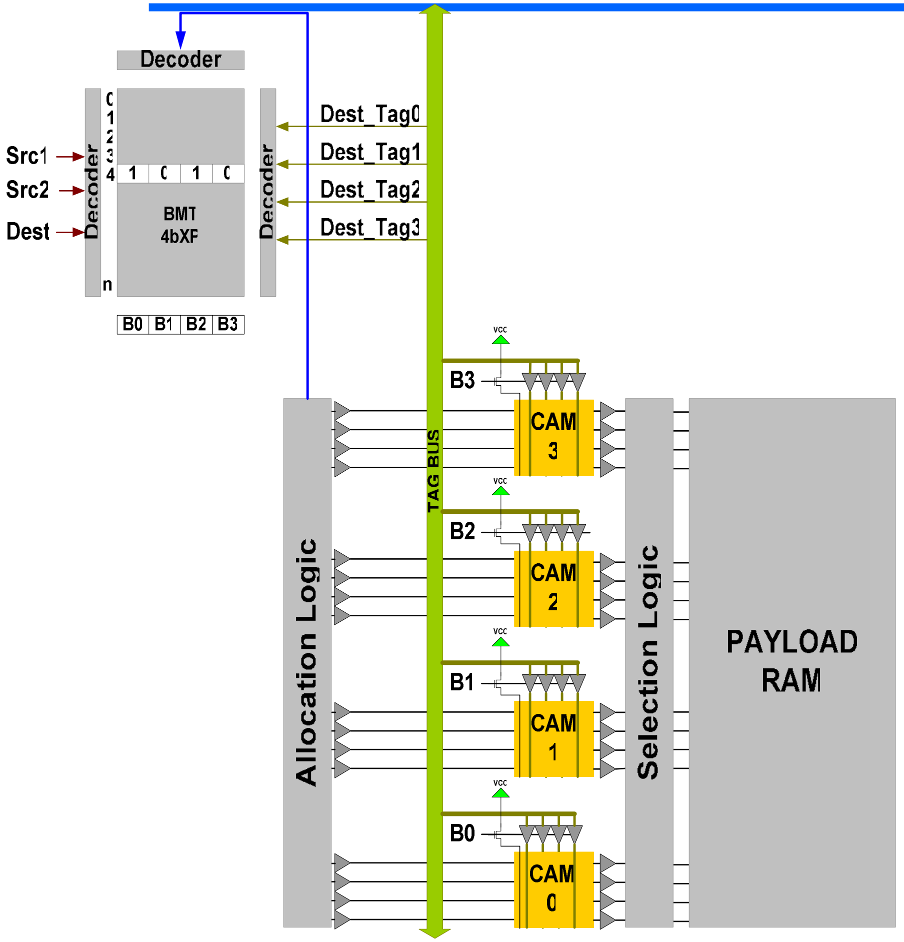


CAM-Blocks with explicit enable

Block Mapping Table

Marco A. Ramírez Salinas, PhD Thesis, Barcelona July 9th, 2007

A new low power wakeup mechanism



Contributions:

Block Mapping Table Mechanism

Allow fast blocks enabling in wakeup phase

Power Reduction

1.5 comparison per committed instruction

Save: ~70% of power for 32-entries queues

Hardware

Minimal hardware requirements

Publications:

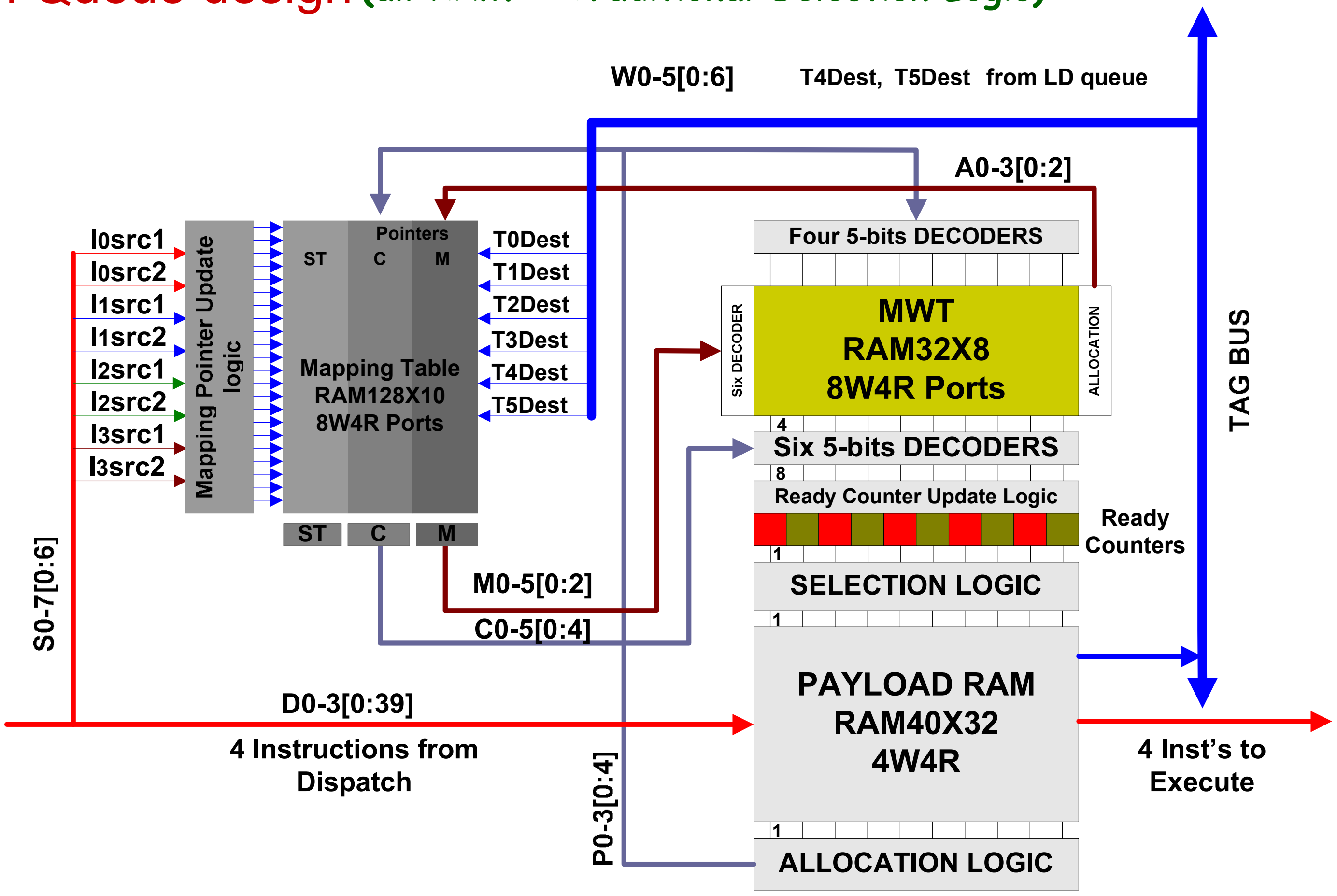
[1] Marco A. Ramírez, A. Cristal, Luis Villa, Alex V. Veidenbaum and Mateo Valero *"A Low-Power Instruction-Queue Wakeup Mechanism "* XIV Jornadas de Paralelismo '03 Leganés-Madrid Spain.

[2] Marco A. Ramírez, A. Cristal, Luis Villa, Alex V. Veidenbaum and Mateo Valero *"A Simple Low-Energy Instruction Wakeup Mechanism"* International Symposium on High Performance Computer '03 Tokyo Japan.

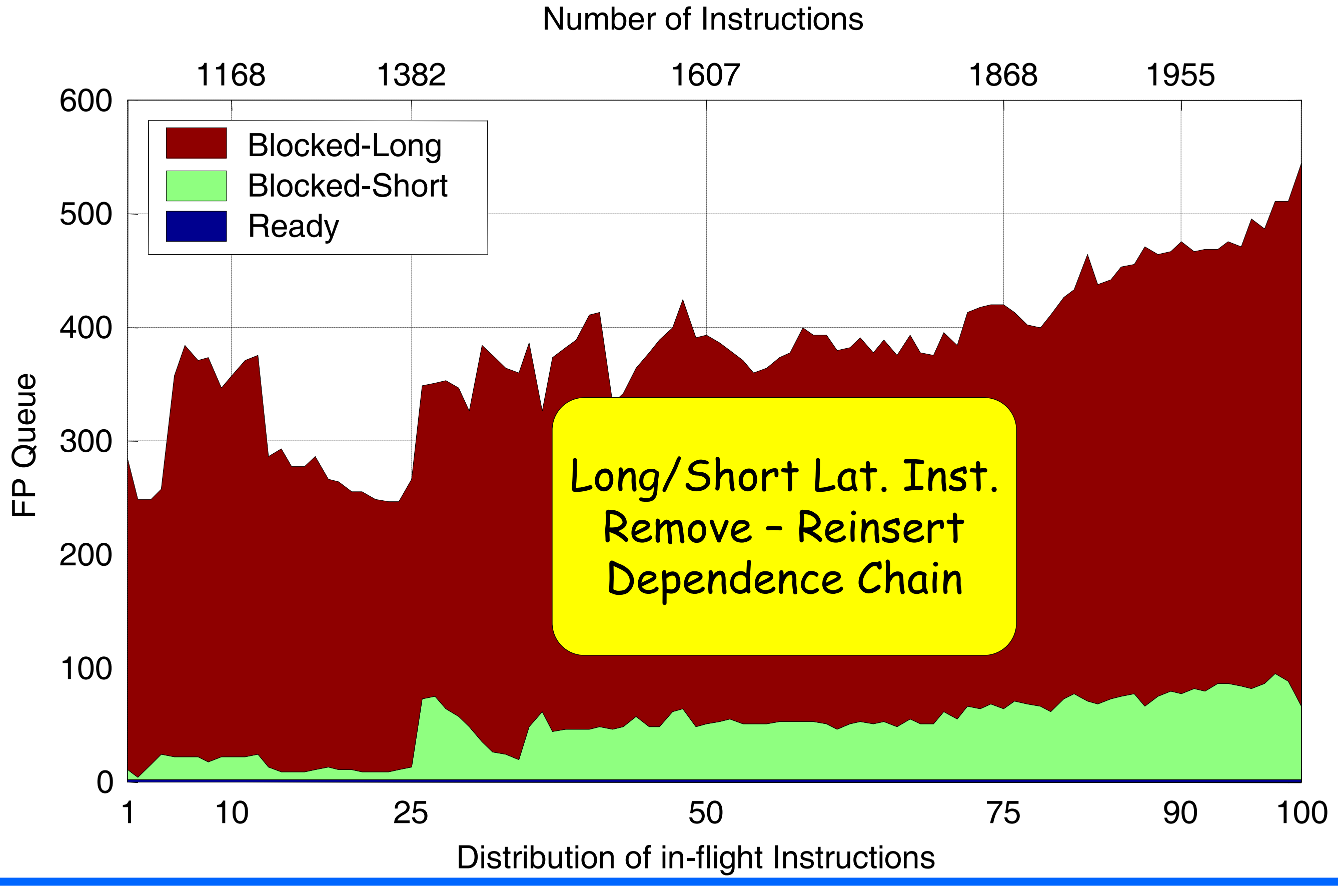
[3] M. A. Ramirez, A. Cristal, M. Valero, A. V. Veidenbaum and L. Villa, *A partitioned instruction queue to reduce instruction wakeup energy,* International Journal of High Performance Computing and Networking '04.

A New Direct Instruction Wakeup Queue design

A RAM Queue design (all RAM + Traditional Selection Logic)



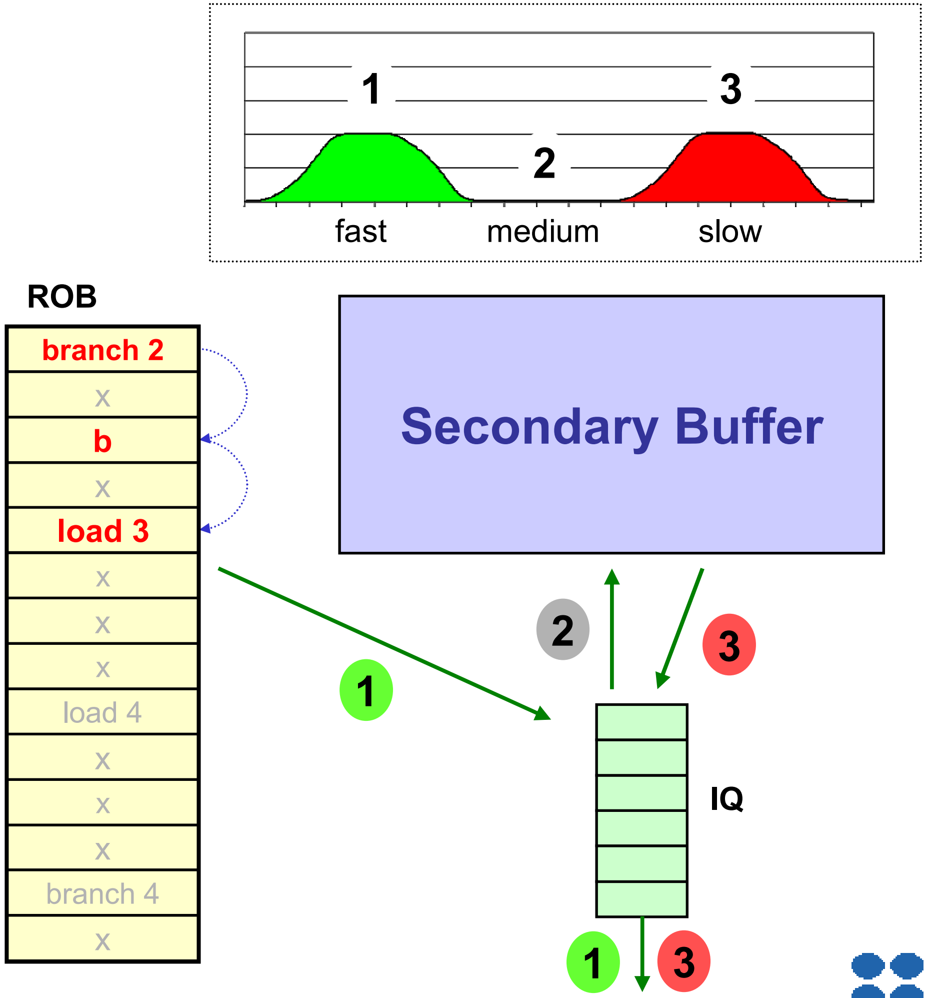
State of FP IQs (specFP, ROB=2048)



Long/Short Lat. Inst.
Remove - Reinsert
Dependence Chain

Execution Time of Instructions

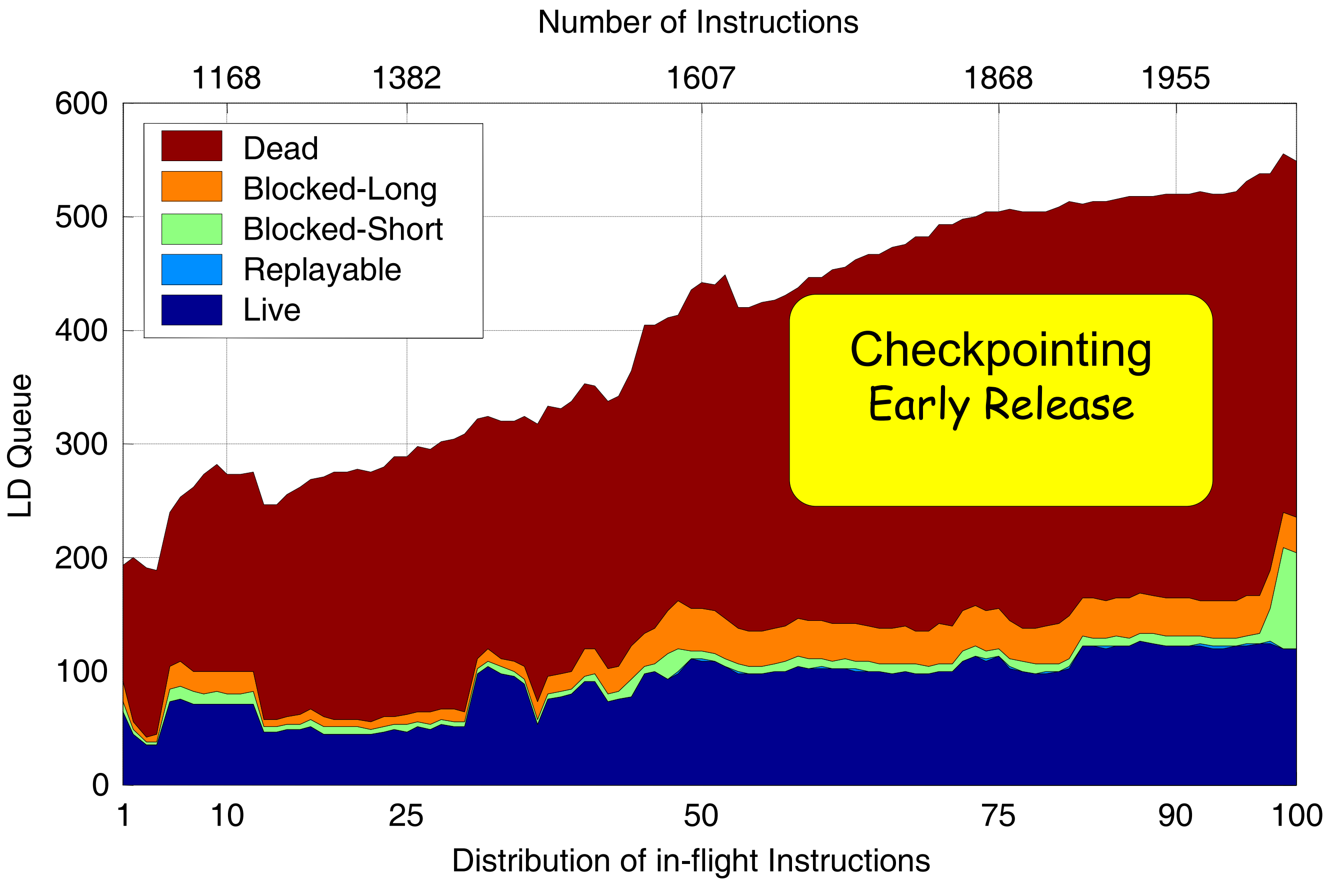
- Lebeck et al., "A large, fast instruction window for tolerating cache misses", ISCA-29, 2002.
- Brekelbaum et al., "Hierarchical scheduling windows", ISCA-35, 2002.
- Cristal et al., "Out-of-Order Commit Processors", TR *UPC-DAC-2003-44*, July 2003 & HPCA-10, Feb. 2004



Load/Store Queues

- Efficient and affordable memory disambiguation is mandatory for kilo-instruction processors
  - We need to guarantee that loads and stores arrive to the memory in the correct order
- Increasing the number of in-flight instructions, can make the load/store queues a true bottleneck both in latency and power

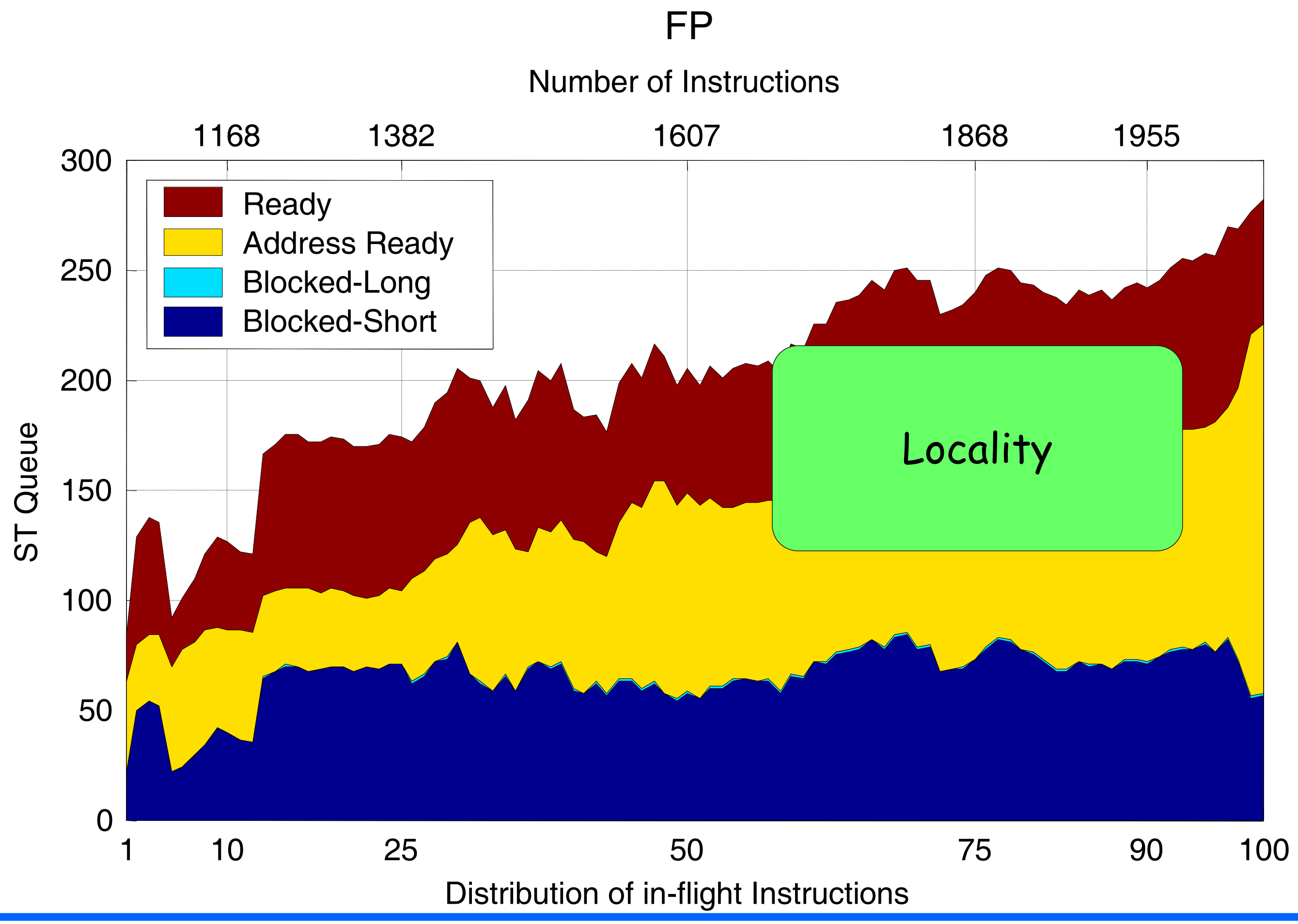
State of LD Queues (specFP, ROB=2048)



Checkpointing
Early Release

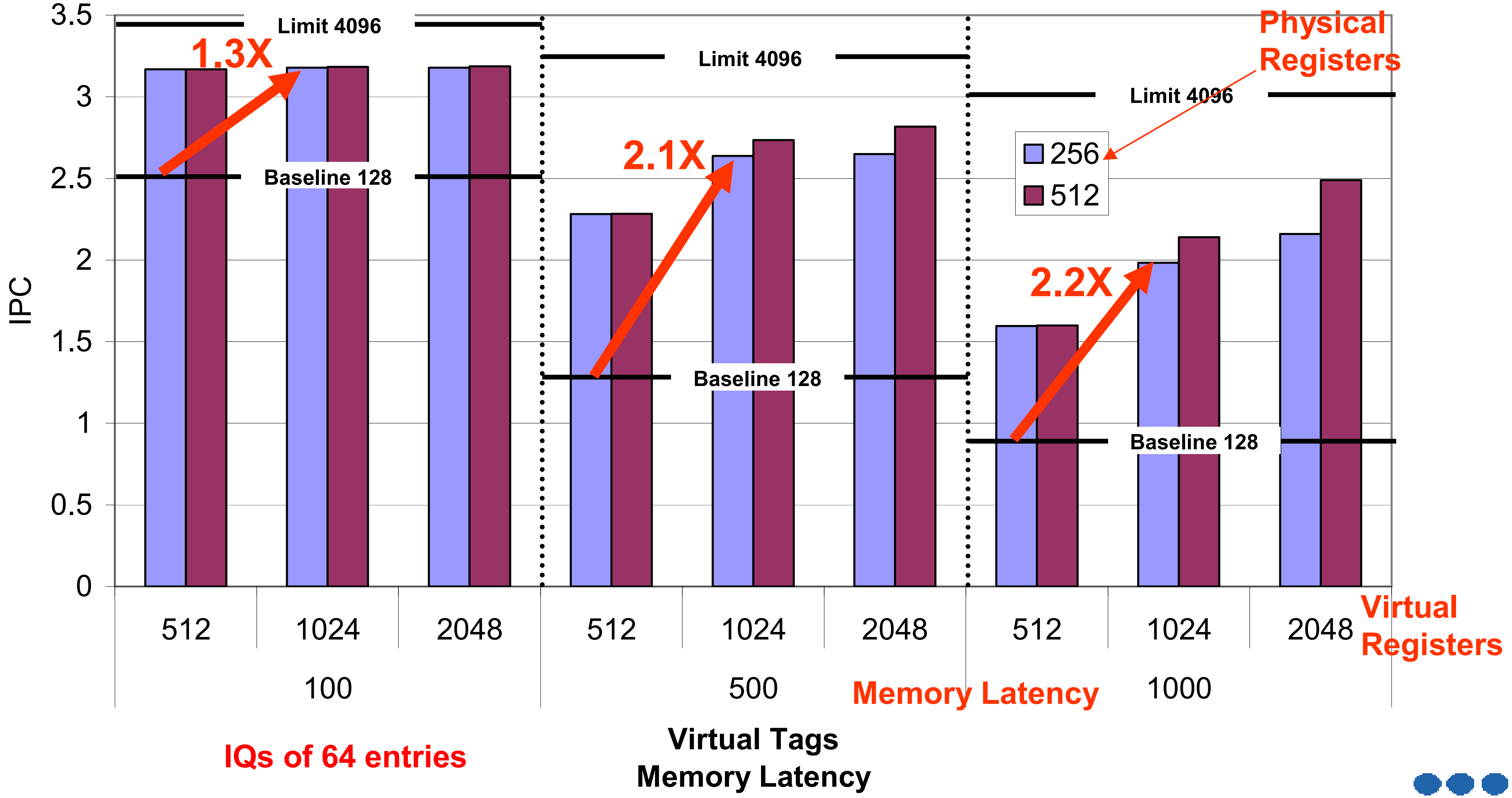
Cristal et al., “A case for resource-conscious out-of-order processors”, IEEE TCCA CA Letters, Vol. 2, 2003.
Cristal et al.: “Large Virtual ROBs by Processor Checkpointing”, TR UPC-DAC, July 2002
J.F. Martínez et al., “Cherry: checkpointer early resource recycling in out-of-order microprocessors”, MICRO-35, 2002.

State of ST Queues (specFP, ROB=2048)



Locality

Putting It All Together



IQs of 64 entries

A. Cristal et al. "Kilo-instruction Processors". Invited paper. ISHPC-V.Tokyo, LNCS-2858. October 20-22th, 2003

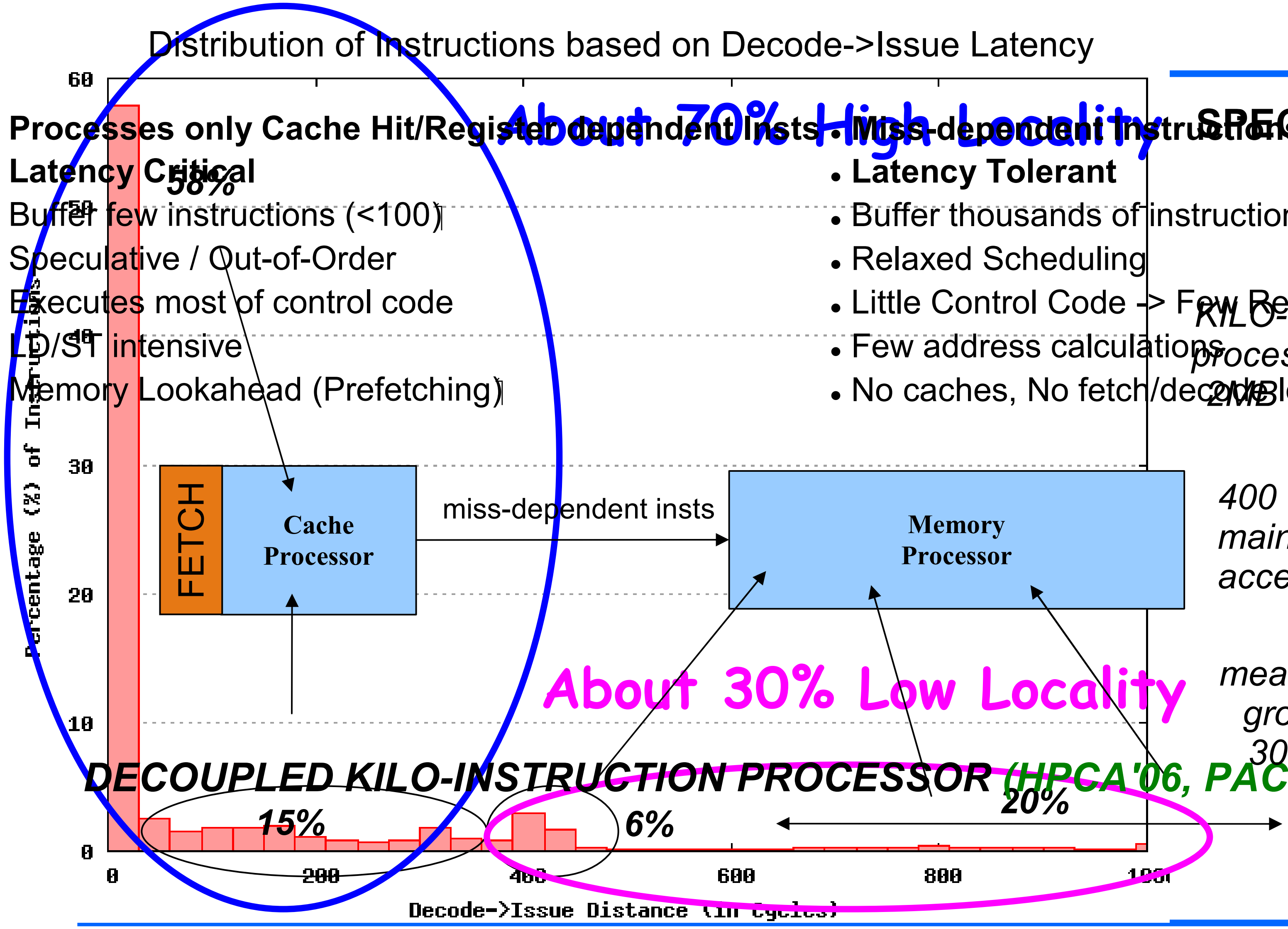
Outline

- ❒ Motivation
- ❒ Increasing the number of in-flight instructions
- ❒ Kilo-instruction Processor Ingredients:
  - ❒ Multi-Checkpointing the ROB
    - Out-of-Order Commit
  - ❒ Early Release of Resources
    - Ephemeral Registers
    - Load Queues
  - ❒ Locality Exploitation
    - Instruction´s Queues
    - LSQ
- ❒ Affordable Kilo-Instruction Processors:
- ❒ Cross-pollination with other techniques:
  - ❒ "Kilo-processor" and multiprocessor systems
  - ❒ "kilo-vector" processors and "kilo-valpred" processors
  - ❒ "Kilo-SMT" processor
  - ❒ Further Improvements:
    - Branch prediction
    - Control Independence
    - Reuse
    - Predicated and multipath execution
- ❒ Conclusion

A different view: D-KIP

- **Processes only Cache Hit/Register dependent Insts**
- **Latency Critical**
- Buffer few instructions (<100)
- Speculative / Out-of-Order
- Executes most of control code
- LD/ST intensive
- Memory Lookahead (Prefetching)

- **Miss-dependent Instructions**
- **Latency Tolerant**
- Buffer thousands of instructions
- Relaxed Scheduling
- Little Control Code -> Few Recoveries
- Few address calculations
- No caches, No fetch/decode logic



SPEC FP 2000

KILO-instruction processor model

2MB L2 Cache

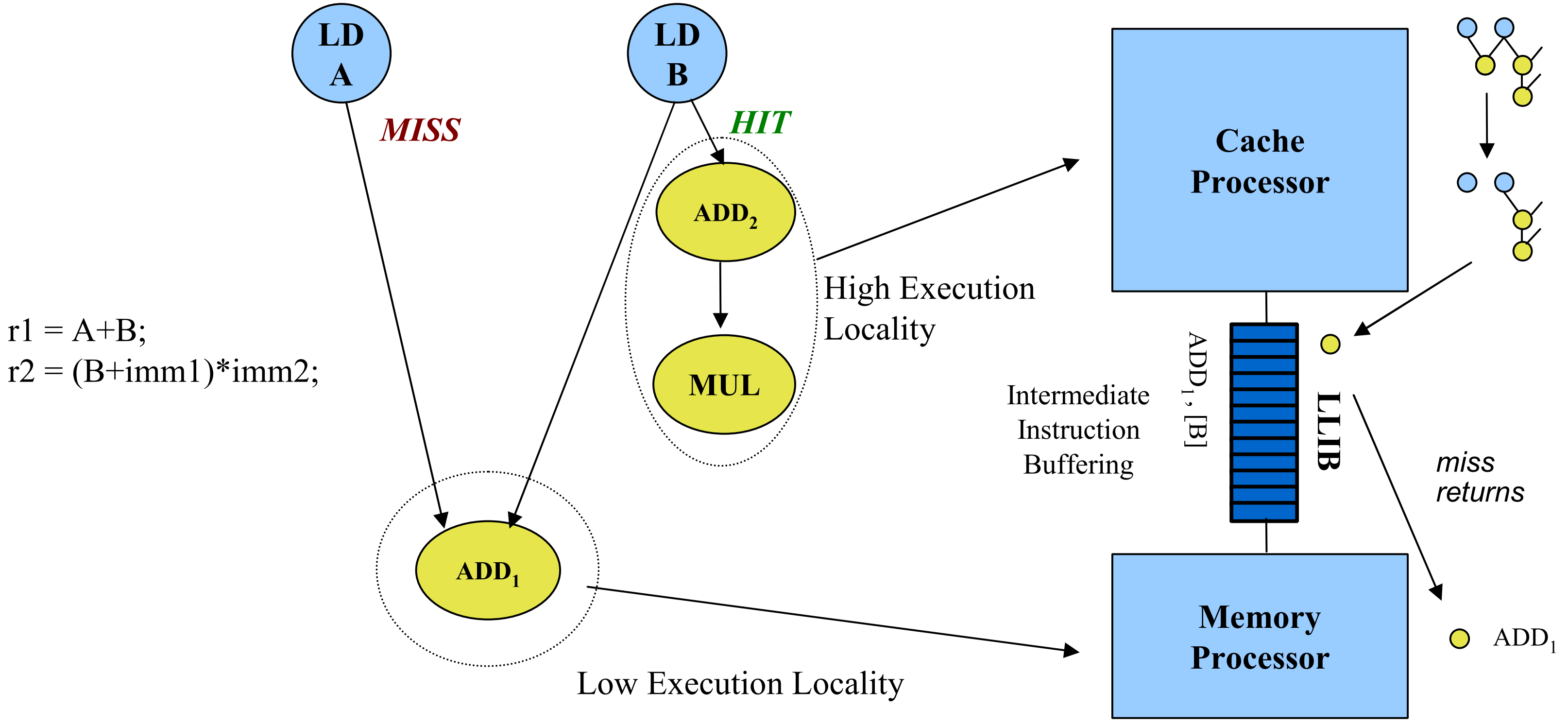
400 cycles main memory access latency

measured in groups of 30 cycles

DECOUPLED KILO-INSTRUCTION PROCESSOR (HPCA'06, PACT'07)

Decoupled Kilo-Instruction Processor (D-KIP)

- Code with ***short decode-to-issue*** distance (*high locality code*) is executed by a small "Cache Processor"
- Miss-dependent code with ***large decode-to-issue*** distance (*low locality code*) migrates to a simpler "Memory Processor" through an in-order instruction buffer

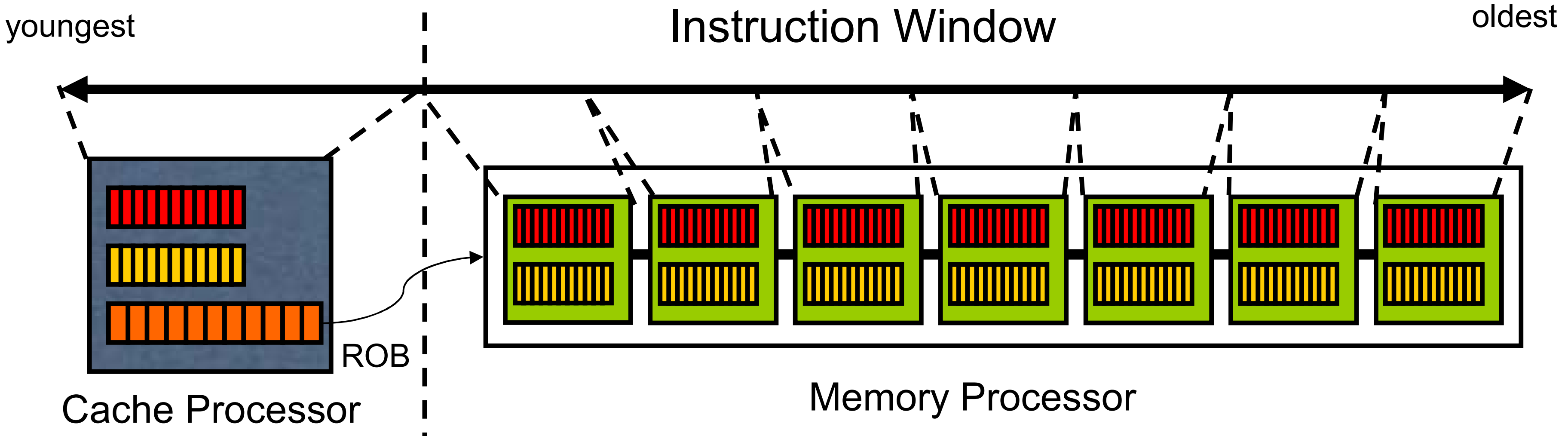


Pericas et al, "A Decoupled KILO-Instruction Processor", HPCA-12 (2006)

The Flexible Heterogeneous MultiCore (I)

The D-KIP architecture forces in-order processing of all low-locality code. Codes covered by a disparity of miss latencies among parallel statements suffer an unnecessary penality

New Idea: **Cache Processor** + ***DataFlow* Memory Processor**



youngest

Instruction Window

oldest

ROB

Cache Processor

Memory Processor

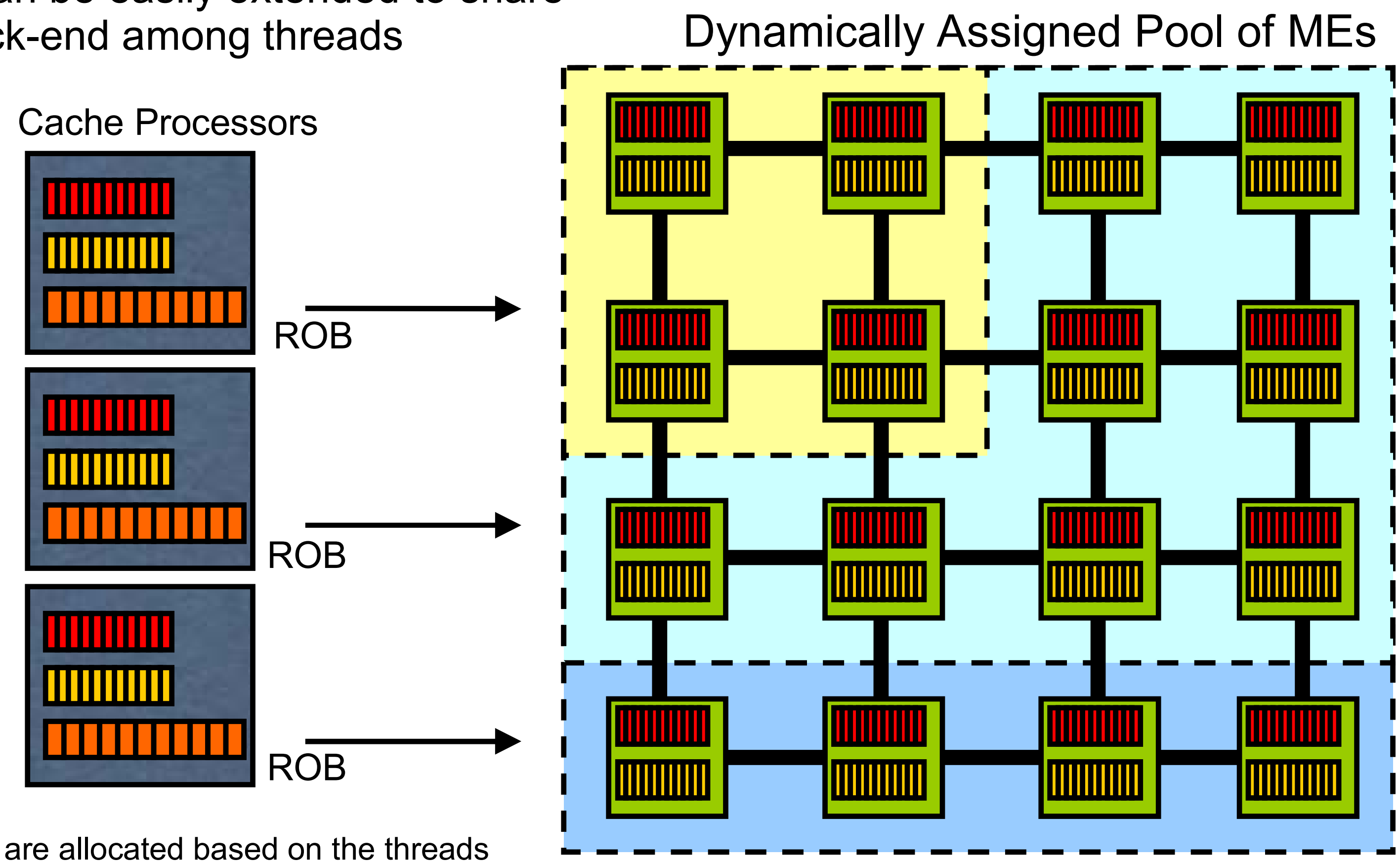
- small
- out-of-order

- multiple engines where each one processes a sequential subset of the low-locality instruction window
- every engine performs multiple in-order scans emulating OoO
- use downsized engines (2wide, in-order) since execution of low-locality code is tolerant to additional latencies
- commit/recovery like checkpointed architecture (one chkpt per engine)
- there is no back-communication from MP to CP except for recovery

Pericas et al, "A Flexible Heterogeneous MultiCore Architecture", PACT-16 (2007)

The Flexible Heterogeneous MultiCore (II)

FMC can be easily extended to share the back-end among threads



Engines are allocated based on the threads needs, incrementing throughput and fairness

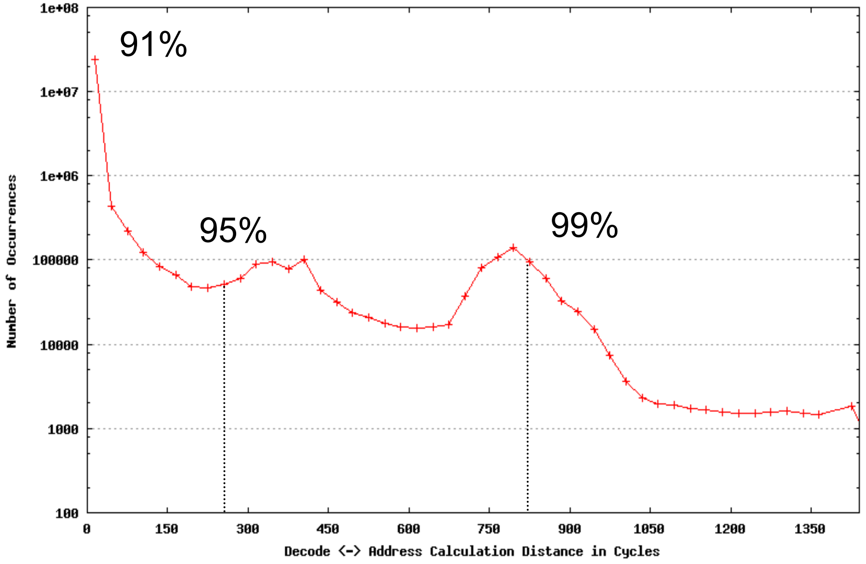
Pericas et al, "A Flexible Heterogeneous MultiCore Architecture", PACT-16 (2007)

Designing a Load/Store Queue for FMC

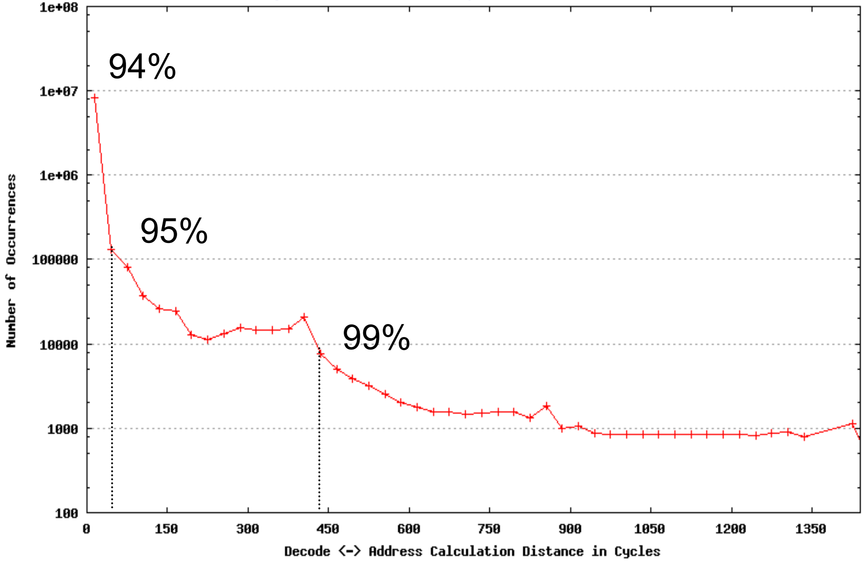
- ❐ The original proposal of FMC used a centralized LSQ in the Cache Processor.
- ❐ In this version, every load access the LSQ/Cache hierarchy from the Memory Processor pays a round trip penalty
- ❐ Using the ideas of Execution Locality we propose a new LSQ design for the FMC

Address Calculation Distribution based on Decode-to-Issue Latency

Most addresses calculations have high locality!



Load Address Calculation Distance



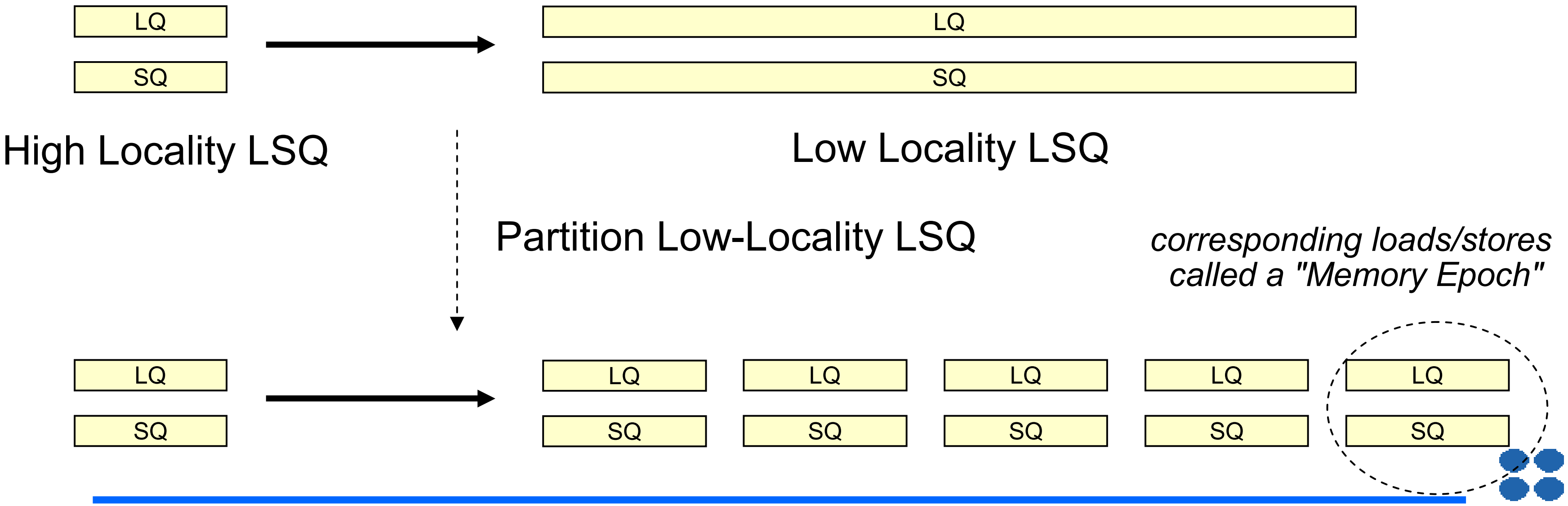
Store Address Calculation Distance

Pericas et al., "A two-level Load/Store Queue based on Execution Locality", ISCA-35 (2008)

Epoch-based Load/Store Queue

- The Epoch-based Load/Store Queue is based on two main principles:
  - Execution Locality (Classification of Loads/Stores into high and low locality)
  - Local and Global Disambiguation
- High/Low Locality Distribution
  - Same as in D-KIP / FMC

Migrate Loads/Stores with known address or loads/stores whose address calculation is miss-dependent (sooner or later all are migrated)

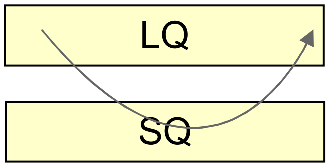


corresponding loads/stores called a "Memory Epoch"

Pericas et al., "A two-level Load/Store Queue based on Execution Locality", ISCA-35 (2008)

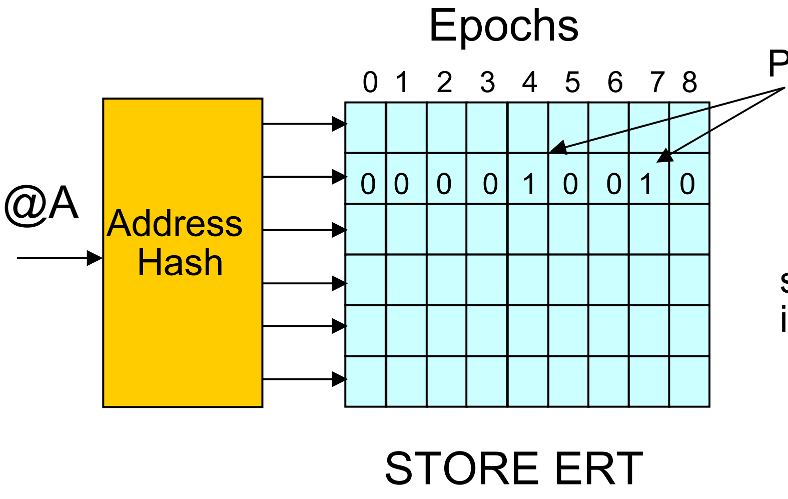
Epoch based Load/Store Queue: Disambiguation

- First Local Disambiguation: Traditional Method
  - High-Locality LSQ Loads Search locally for forwarding Stores
  - Epoch LSQs loads search locally for forwarding Stores
  - Stores also search locally for store-load violation



search store queue

- Global Disambiguation: Consult ERT (Epoch Resolution Table)
  - Only if Local Disambiguation does not return a definitive answer



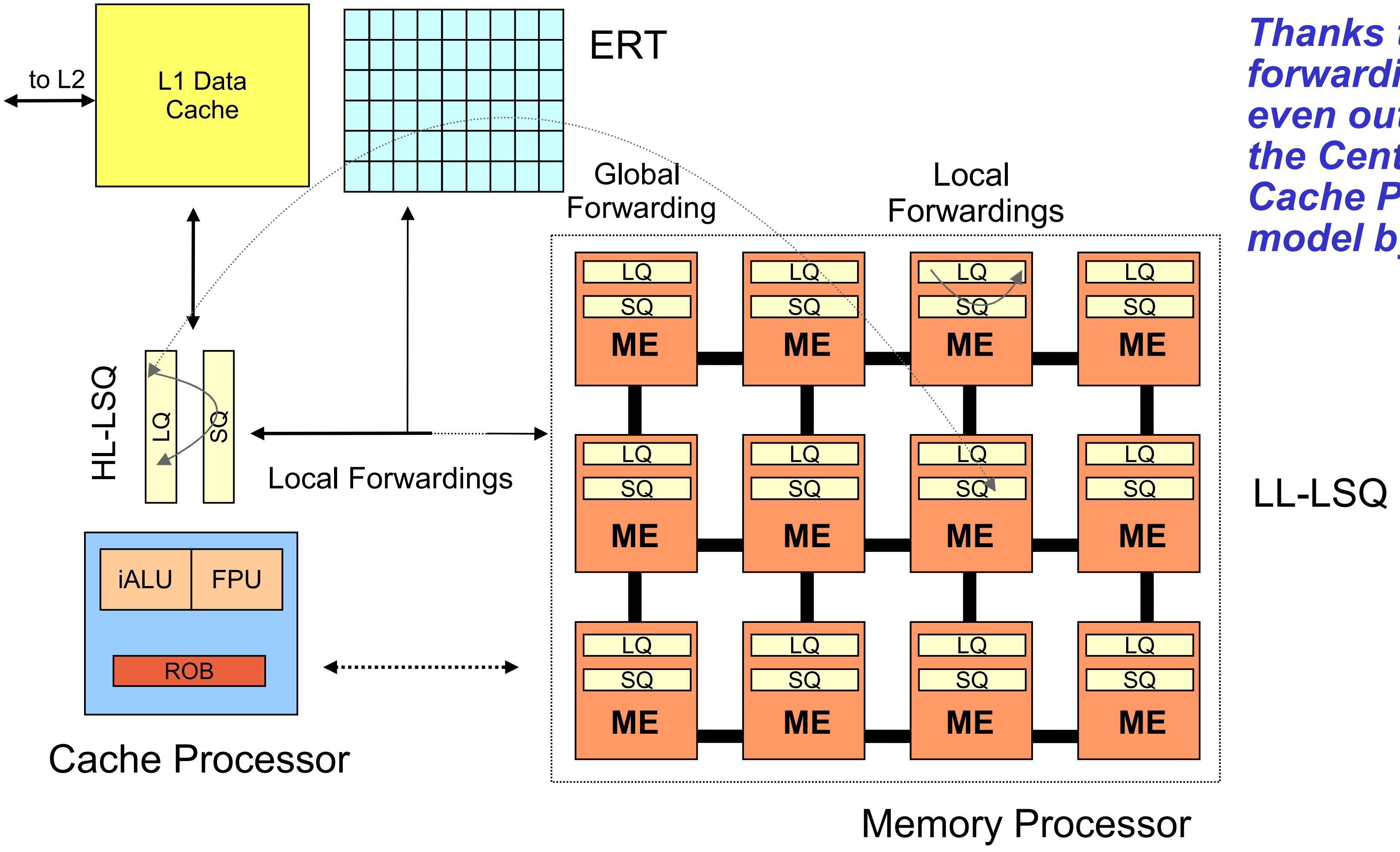
Possibly Matching Stores
in Epochs 4 and 7

```
search (epoch 7);
if(!match)
    search (epoch 4);
```

Epoch Resolution Table:
- Tracks LL loads/stores
- Very simple:
  - no counters
  - column clean when engine is commited

Pericas et al., "A two-level Load/Store Queue based on Execution Locality", ISCA-35 (2008)

ELSQ: Global Picture



Thanks to local forwarding, ELSQ even outperforms the Central LSQ in Cache Processor model by 1-2%

Pericas et al., "A two-level Load/Store Queue based on Execution Locality", ISCA-35 (2008)

ELSQ: LSQ Network Utilization

- The network of LSQs has many interesting properties that allow reduced-power operation:
  - Around 50% of all cycles the Memory Processor is empty. Only static power is consumed. Low-power techniques (e.g., sleep transistors) can be applied.
  - Due to high locality, LD/ST activity in the MP is very reduced. ~80% of all searches happen in the Cache Processor.
  - As ~98% of stores get their address in the CP, we can simplify the architecture by forcing all stores to compute their address in the CP. This completely eliminates the Load Queue from the MP and has less than a 2% IPC penality.

Pericas et al., "A two-level Load/Store Queue based on Execution Locality", ISCA-35 (2008)

Evaluation of Architecture: Parameters

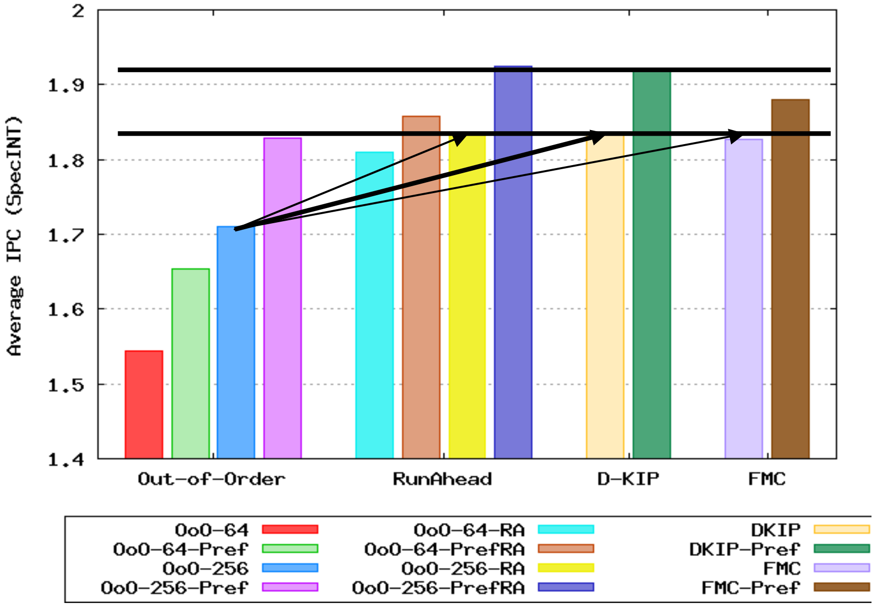
- Several Architectures and Configurations are evaluated:
  - Out-of-Order Processors with ROBs of 64 and 256 instructions (Issue Queue and Register File sized to avoid stalls)
  - D-KIP model is implemented with:
    - 4-way OoO 64-ROB Cache Processor (40-entries IQ, 96 registers)
    - 2K FIFO Instruction Buffer
    - 4-way In-Order Memory Processor
  - FMC Parameters:
    - 4-way OoO 64-ROB Cache Processor (40-entries IQ, 96 registers)
    - 16 Memory Engines
    - Memory Engines are 2-way & In-Order. Up to 128 long-latency instructions, 64 loads and 32 stores per engine
- RunAhead is implemented on the OoO model with unlimited fully-associative runahead cache
- Stream prefetcher holding up to 64KB of prefetched data (up to 16 streams)
- Memory Latency: 400 cycles

Performance: SPECINT2000

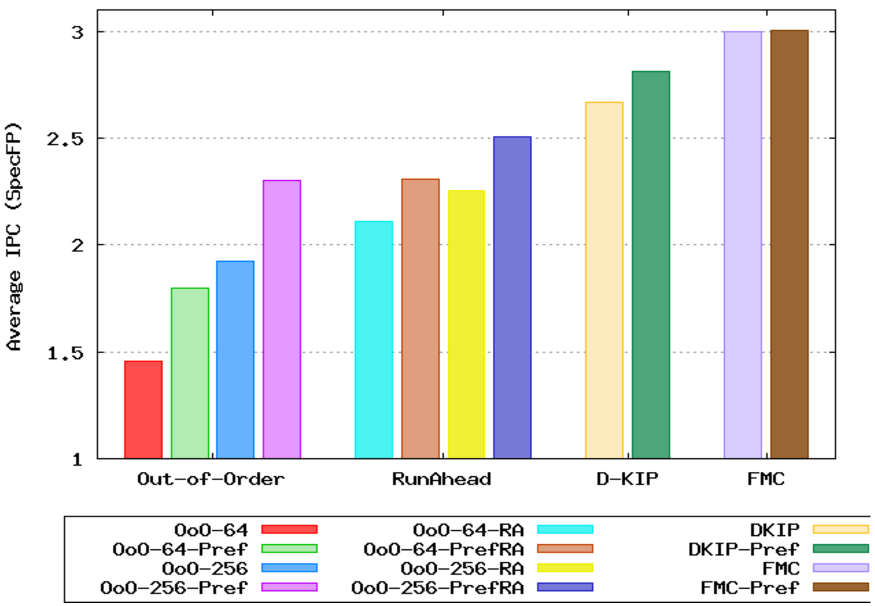
- D-KIP and FMC perform similar to OoO-256-RA and OoO-256-Prefetch
- OoO-64-RA performs only slightly worse than OoO-256-RA and OoO-256-Pref: RA very resource-efficient for integer programs

- D-KIP, FMC and OoO-64-RA outperform OoO-256 by 7% despite much smaller associative structures

- D-KIP-Pref and OoO-256-PrefRA have similar performance and slightly better than FMC-Pref



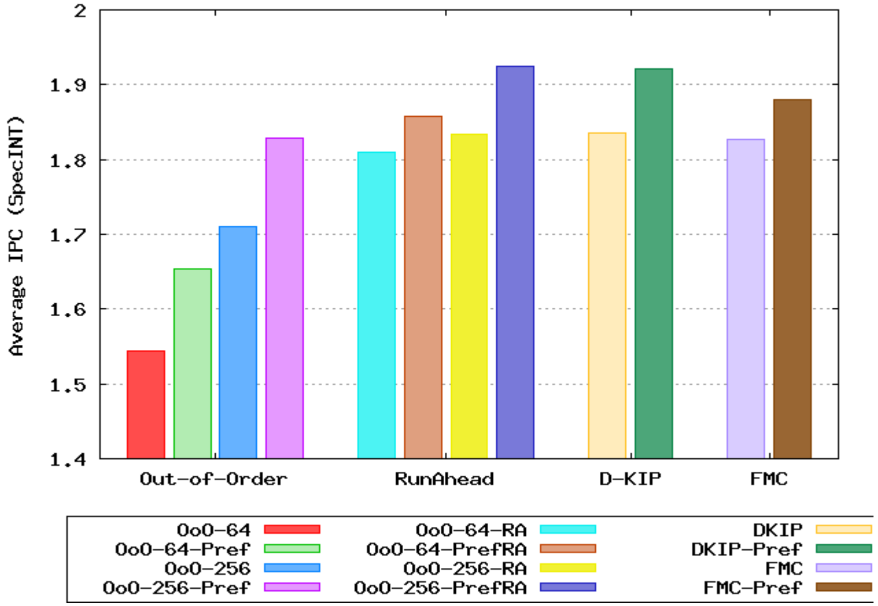
SPECINT2000



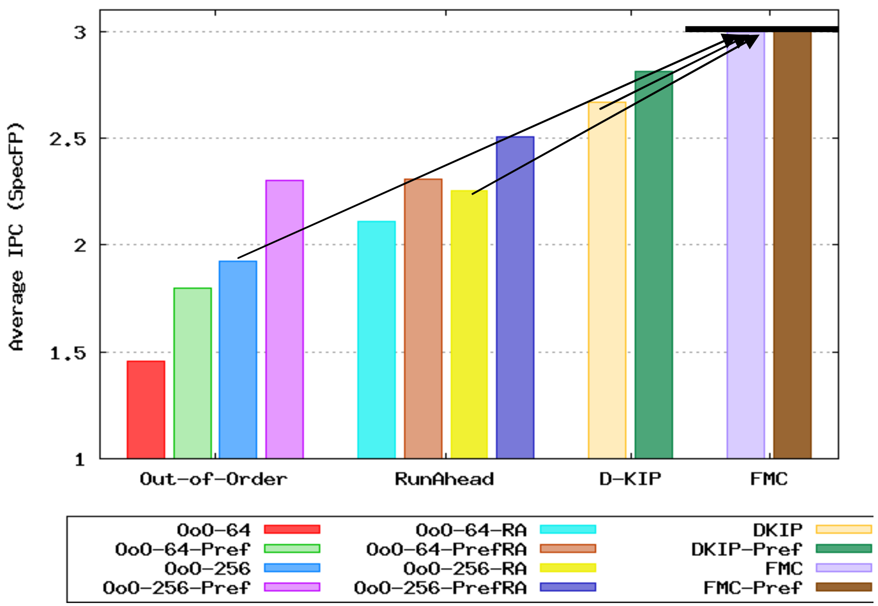
SPECFP2000

Performance: SPECFP2000

- FMC outperforms D-KIP by 12%, OoO-256-RA by 33% and OoO-256 by 56%
- FMC lookahead is so far and accurate that a prefetcher is *no longer necessary*



SPECINT2000



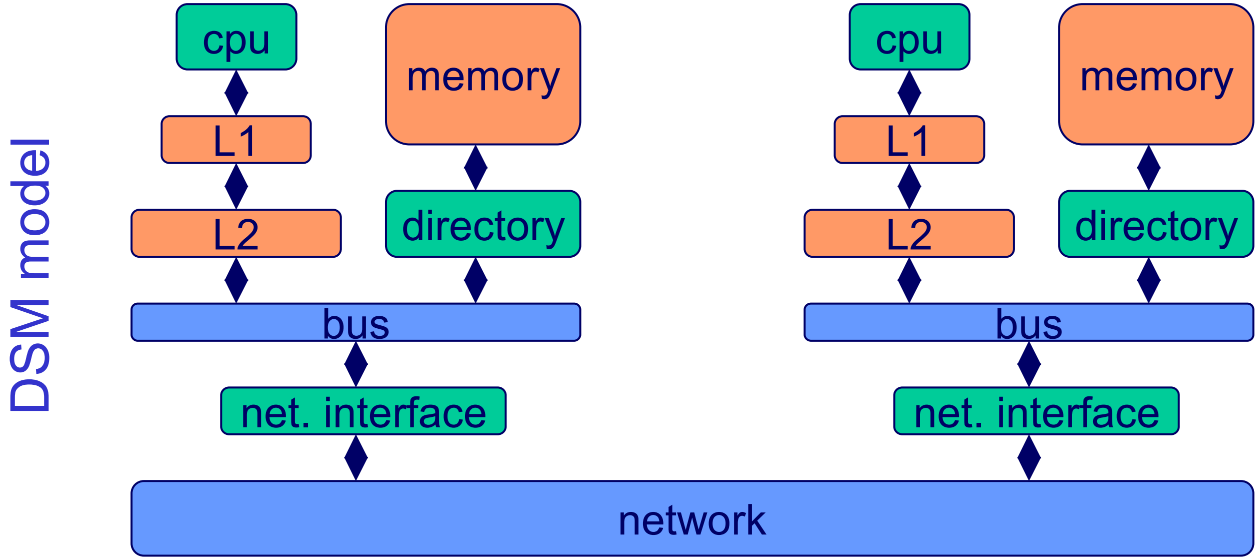
SPECFP2000

Outline

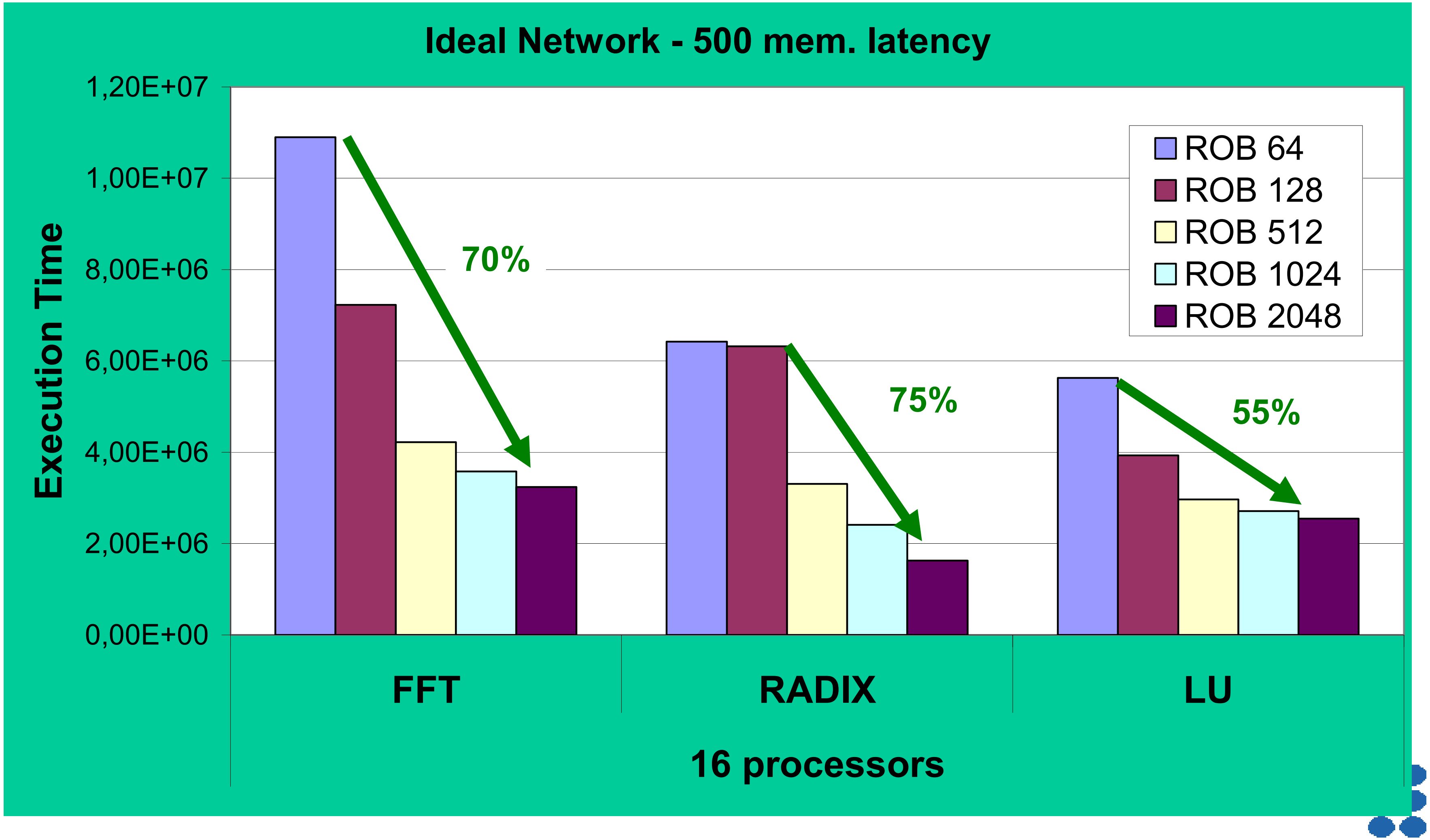
- Motivation
- Increasing the number of in-flight instructions
- Kilo-instruction Processor Ingredients:
  - Multi-Checkpointing the ROB
    - Out-of-Order Commit
  - Early Release of Resources
    - Ephemeral Registers
    - Load Queues
  - Locality Exploitation
    - Instruction´s Queues
    - LSQ
- Cross-pollination with other techniques:
  - "Kilo-processor" and multiprocessor systems
  - "kilo-vector" processors and "kilo-valpred" processors
  - "Kilo-SMT" processor
  - Further Improvements:
    - Branch prediction
    - Control Independence
    - Reuse
    - Predicated and multipath execution
- Conclusion

Motivation: multiprocessors

- Shared-Memory Multiprocessors: increased latencies
  - Traversing interconnection hardware
    - Centralized memory (SMP)
    - Remote memories (DSM)
  - Preserving cache coherence

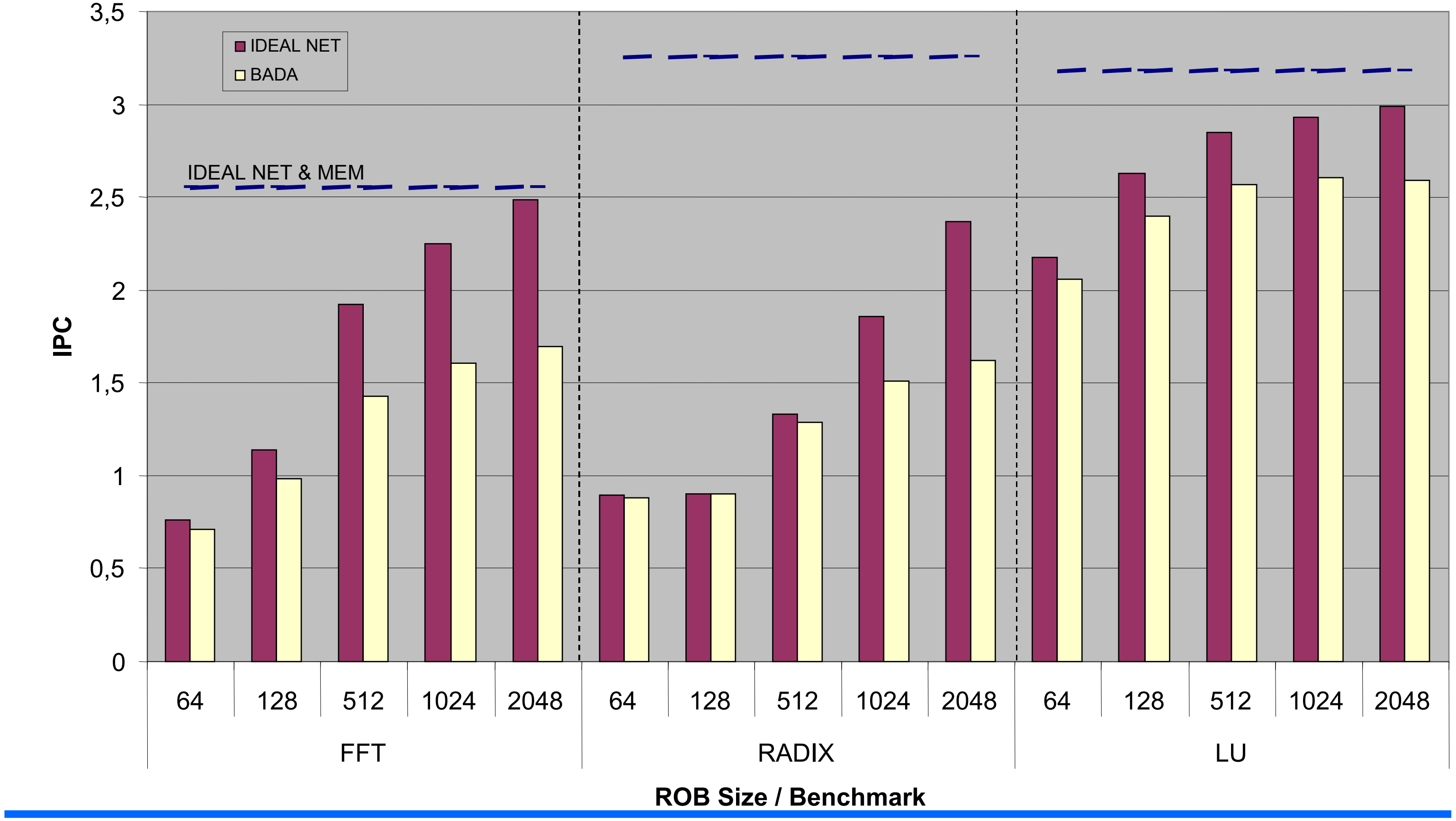


Evaluate potential



"Kilo-processor" and multiprocessor systems

First Results

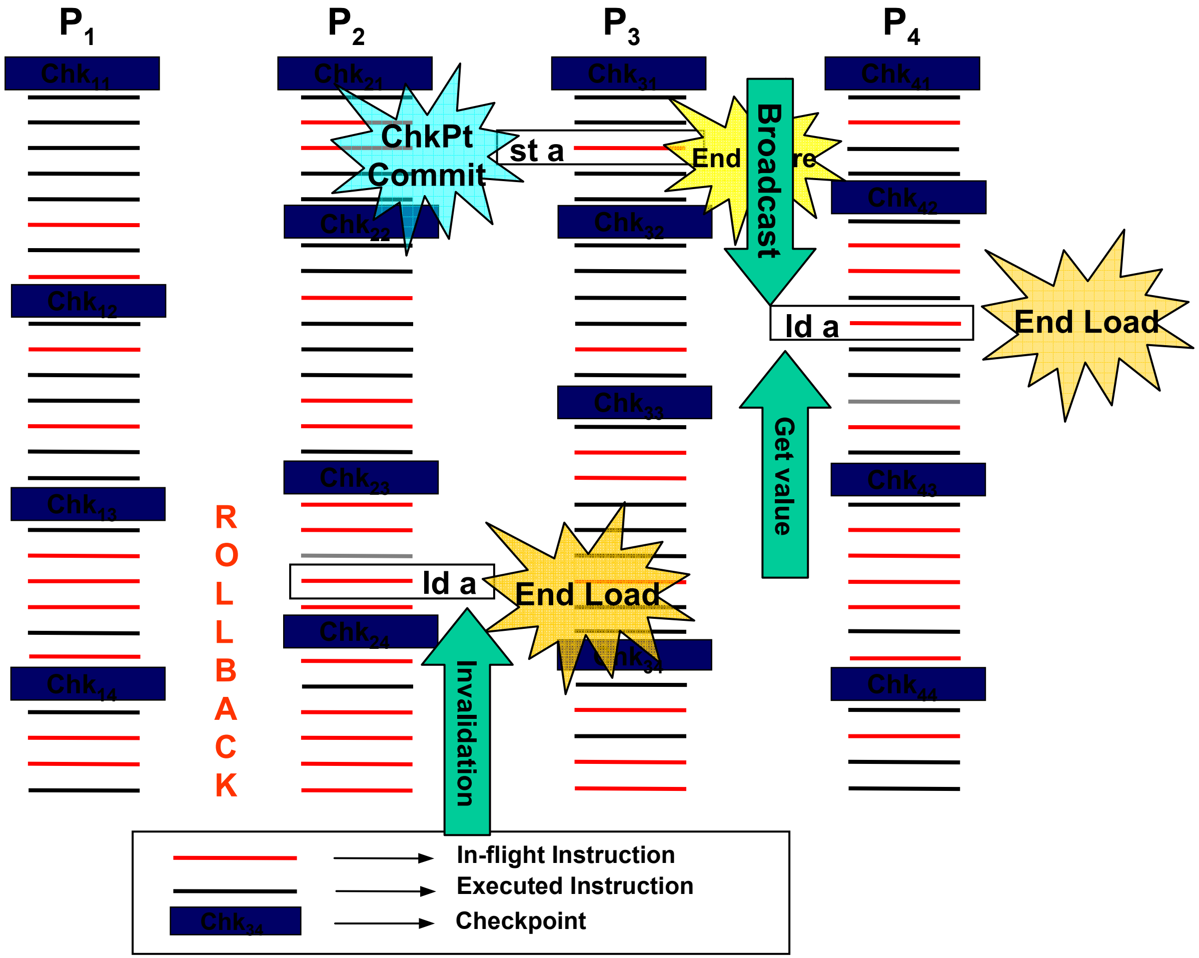


M. Galluzzi et al. " A First glance at Kiloinstruction Based Multiprocessors" Invited Paper. ACM Computing Frontiers Conference,Ischia, Italy, April 10-12, 2004

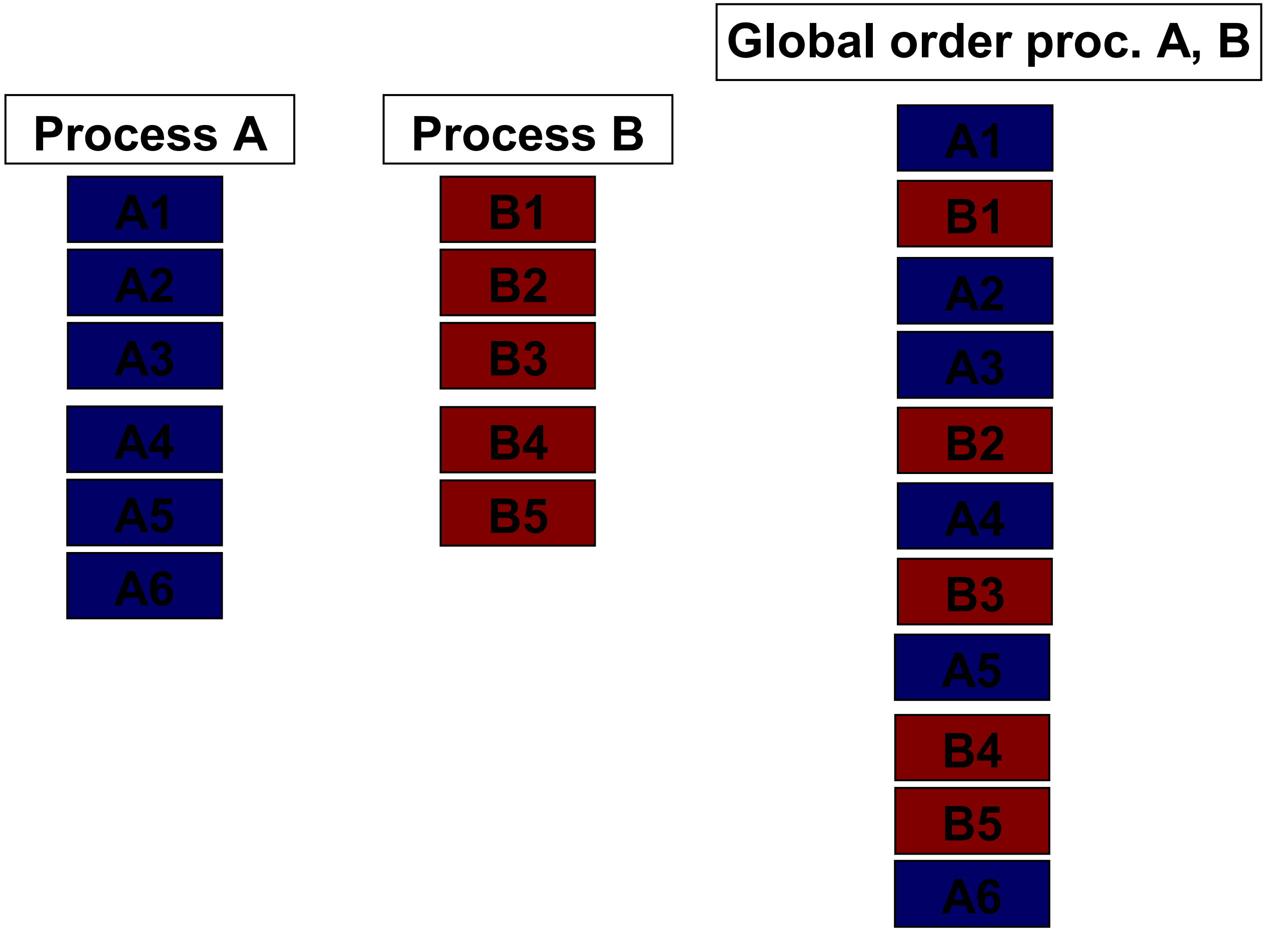
2. Transactional Memory

- Programmer specifies large, atomic tasks
  - atomic { some_work; }
  - Multiple objects, unstructured control-flow, …
  - Declarative: user simply specifies, system implements details
- TM simplifies parallel programming
  - Parallel algorithms: non-blocking sync with coarse-grain code
    - Performance = fine grain locks
  - Sequential algorithms: speculative parallelization
- "All transactions all the time"
  - Stanford Transactional Coherence & Consistency (TCC)
  - Eases performance optimization
  - Makes deterministic replay trivial
- Atomicity & isolation are generally useful
  - For debugging, checkpointing, exception handling, garbage collection, security, compiler optimization

Implicit transactions



SC – explanation



Process A

A1

A2

A3

A4

A5

A6

Process B

B1

B2

B3

B4

B5

Global order proc. A, B

A1

B1

A2

A3

B2

A4

B3

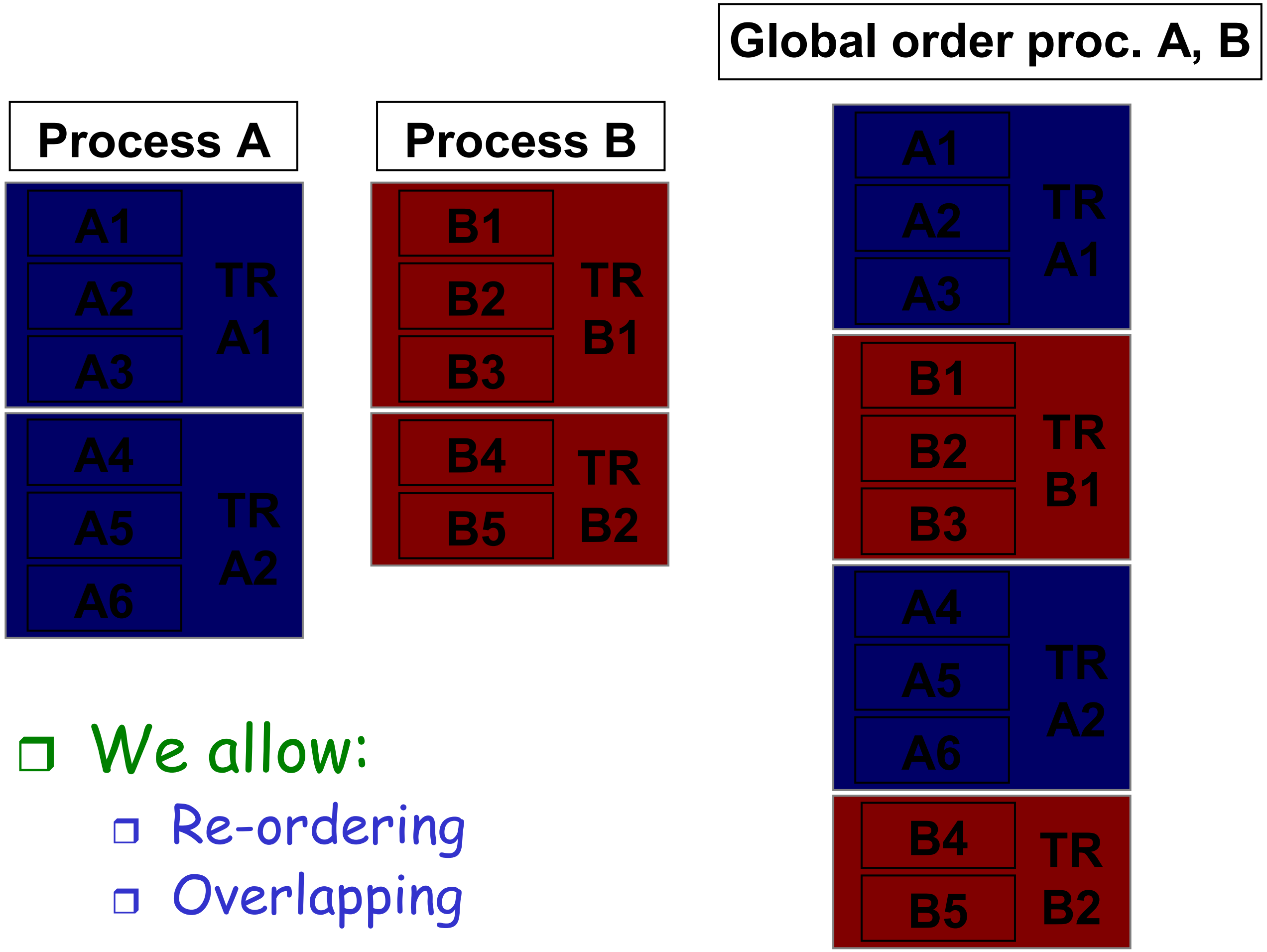
A5

B4

B5

A6

SC – using transactions

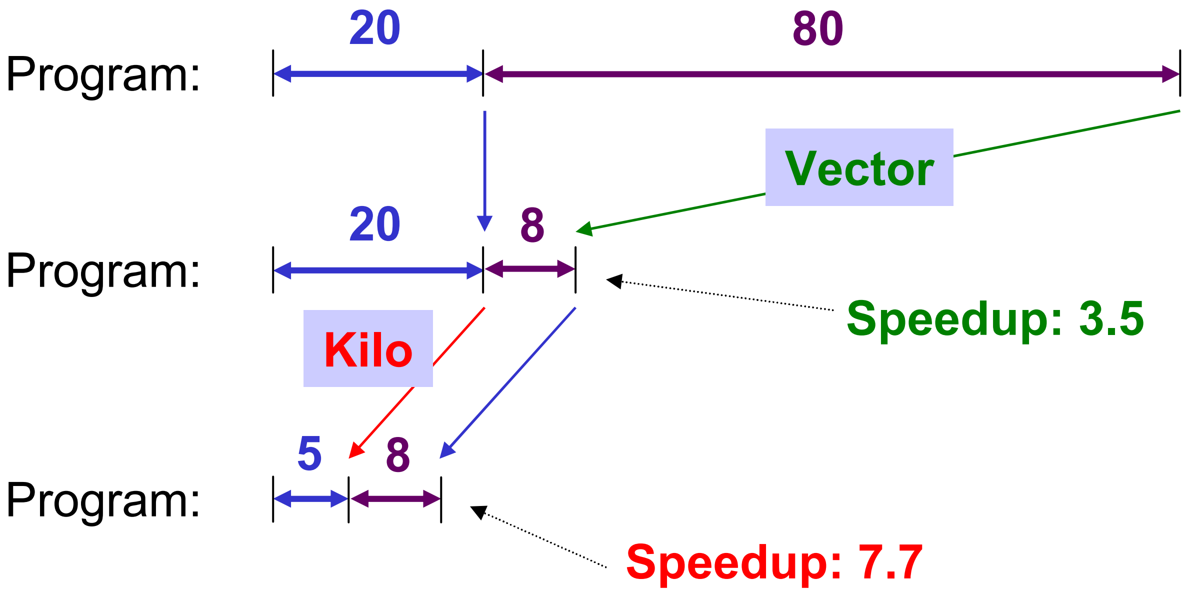


- We allow:
  - Re-ordering
  - Overlapping

Outline

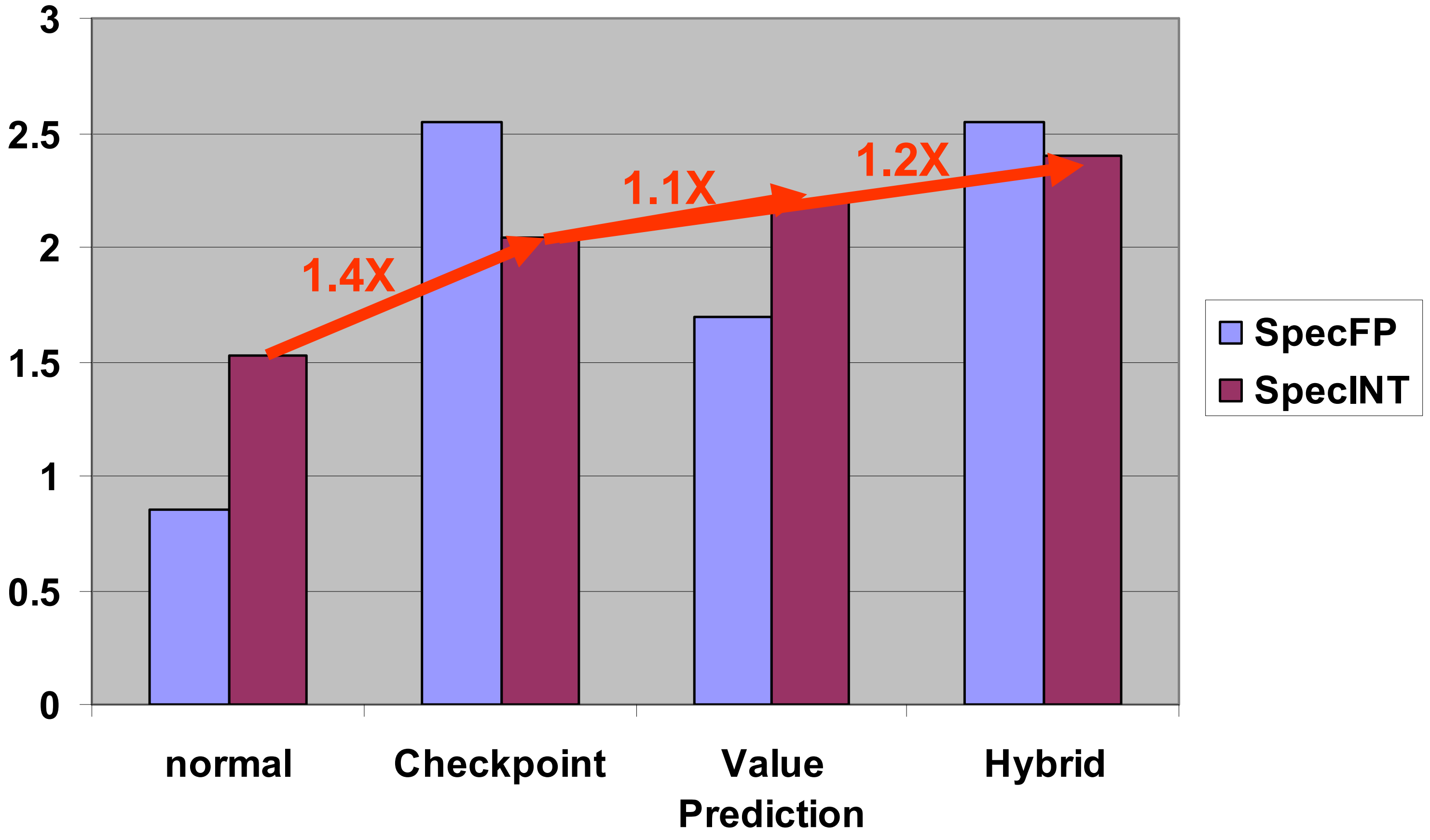
- ❒ Motivation
- ❒ Increasing the number of in-flight instructions
- ❒ Kilo-instruction Processor Ingredients:
  - ❒ Multi-Checkpointing the ROB
    - Out-of-Order Commit
  - ❒ Early Release of Resources
    - Ephemeral Registers
    - Load Queues
  - ❒ Locality Exploitation
    - Instruction´s Queues
    - LSQ
- ❒ Cross-pollination with other techniques:
  - ❒ "Kilo-processor" and multiprocessor systems
  - ❒ "kilo-vector" processors and "kilo-valpred" processors
  - ❒ "Kilo-SMT" processor
  - ❒ Further Improvements:
    - Branch prediction
    - Control Independence
    - Reuse
    - Predicated and multipath execution
- ❒ Conclusion

“Kilo-vector” processor



M. Valero Keynote at HPCA, Madrid-2003

“Kilo-valpred” processor



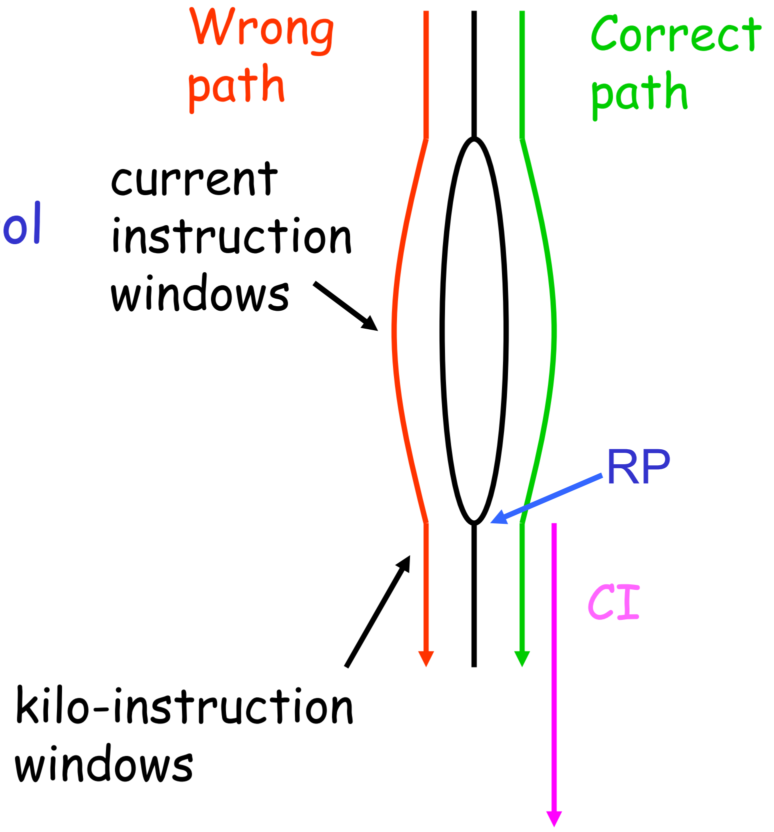
M. Valero Keynote at HPCA, Madrid-2003

Outline

- Motivation
- Increasing the number of in-flight instructions
- Kilo-instruction Processor Ingredients:
  - Multi-Checkpointing the ROB
    - Out-of-Order Commit
  - Early Release of Resources
    - Ephemeral Registers
    - Load Queues
  - Locality Exploitation
    - Instruction´s Queues
    - LSQ
- Cross-pollination with other techniques:
  - "Kilo-processor" and multiprocessor systems
  - "kilo-vector" processors and "kilo-valpred" processors
  - "Kilo-SMT" processor
  - Further Improvements:
    - Branch prediction
    - Control Independence
    - Reuse
    - Predicated and multipath execution
- Conclusion

Kilo and Control Independence

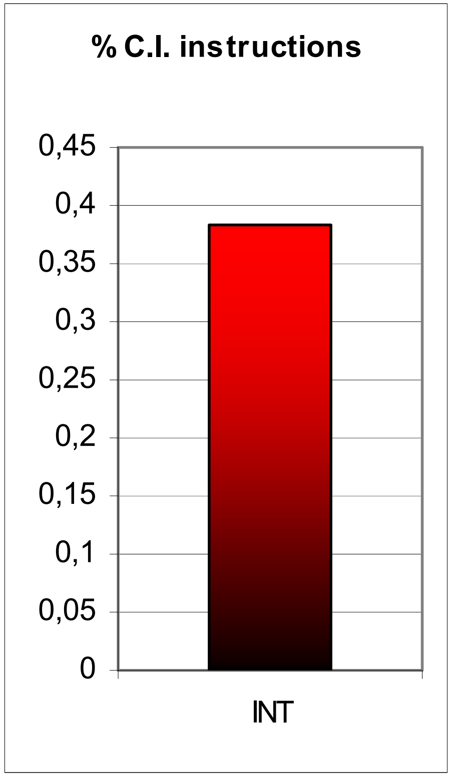
- Larger windows improve:
  - The probability of finding the reconvergence point
  - The correct detection of control independent instructions because the wrong path is completely executed
  - The execution of more control independent instructions for later reuse



M. Valero Keynote at HPCA, Madrid-2003

Kilo and Control Independence

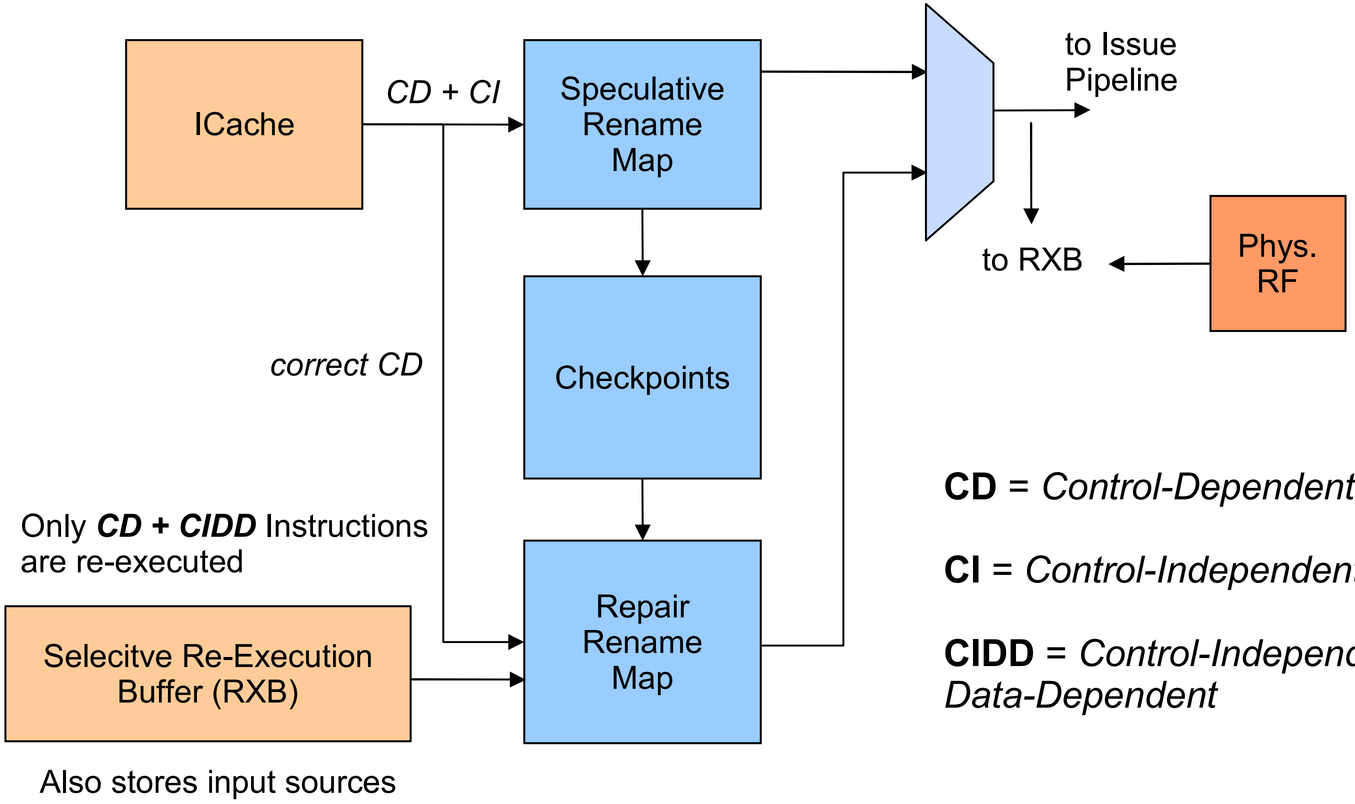
- ❒ More opportunities to find control independent instructions
  - ❒ Squash reuse
  - ❒ Control-independent instruction reexecution removal
  - ❒ Savings:
    - Power/energy
    - Execution bandwidth
    - Resources
  - ❒ Helps to go far ahead in the instruction window faster



M. Valero Keynote at HPCA, Madrid-2003

Some Recent Work

- *Transparent Control Independence* is an interesting approach to overcome the control path problem:



ICache
CD + CI
Speculative Rename Map
to Issue Pipeline
to RXB
Phys. RF
correct CD
Checkpoints
Only **CD + CIDD** Instructions are re-executed
Selecitve Re-Execution Buffer (RXB)
Repair Rename Map
Also stores input sources

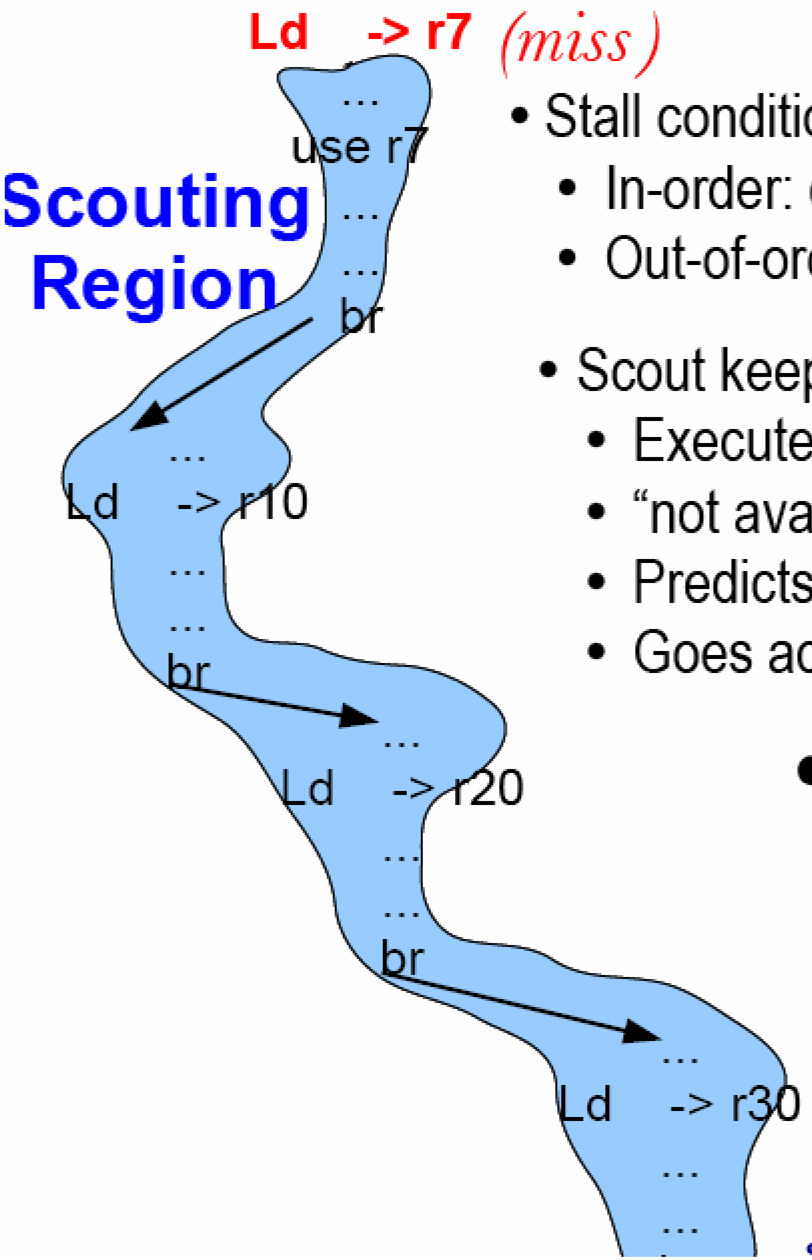
CD = *Control-Dependent*

CI = *Control-Independent*

CIDD = *Control-Independent Data-Dependent*

Al-Zawawi et al., "Transparent Control Independence (TCI)", ISCA-34 (2007)

Rock: Threads for Latency Computing, Scout Threading



Ld -> r7 *(miss)*

Scouting Region

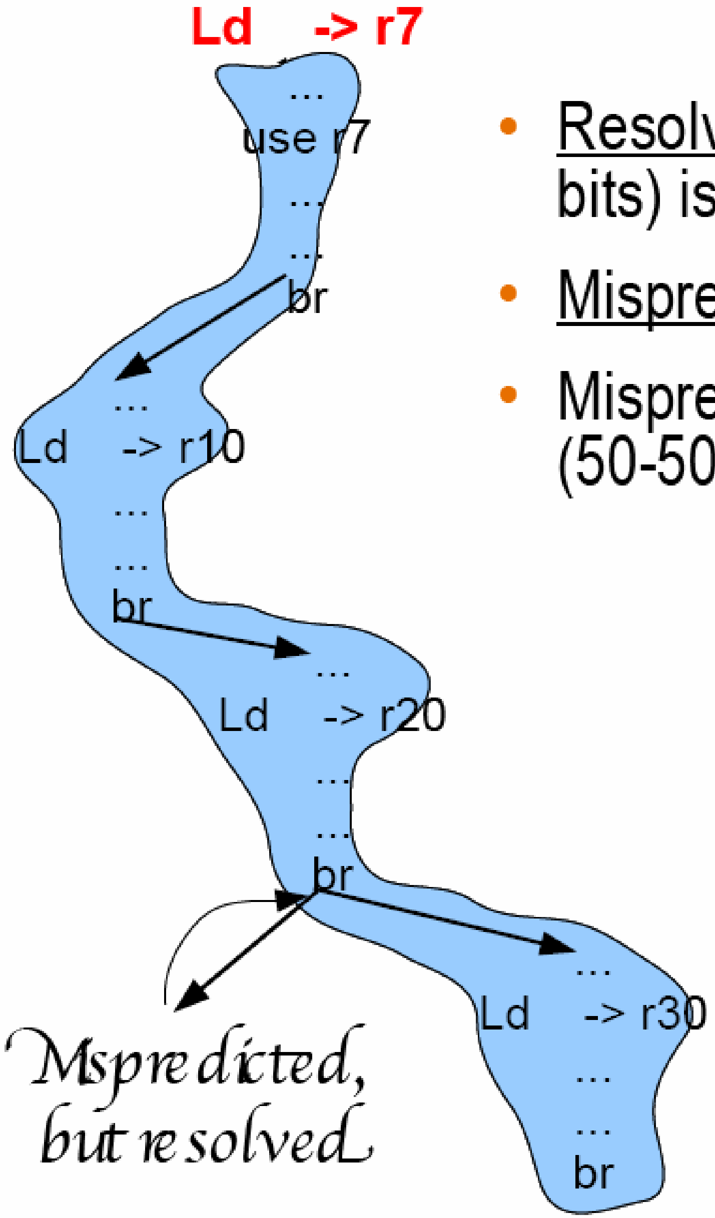
```
...
use r7
...
...
br
...
Ld    -> r10
...
...
br
...
Ld    -> r20
...
...
br
...
Ld    -> r30
...
...
```

- Stall conditions launch a hardware thread – the scout
  - In-order: data dependencies, others
  - Out-of-order: full buffers (e.g. Instruction window)

- Scout keeps going down the instruction stream
  - Executes independent instructions
  - "not available" is propagated
  - Predicts branches
  - Goes across locks, memory barriers

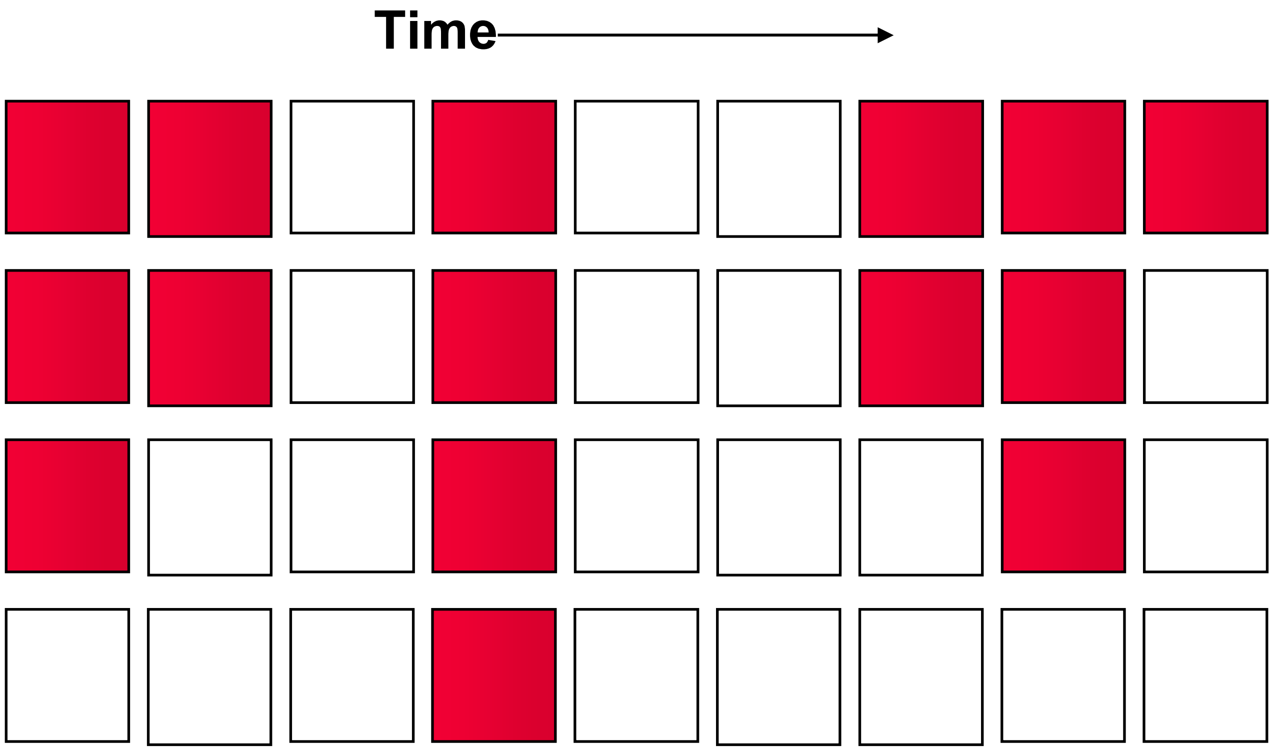
- Goals for the Scout Thread
  - Get to next misses
  - Bring data into caches
  - Warm-up instruction cache
  - Warm-up branch predictor

Runtime Resolution



- Resolved: the data (register or conditional code bits) is available (not depending on a load miss)
- Mispredicted, but resolvable, are fine
- Mispredicted and unresolvable are not always bad (50-50)

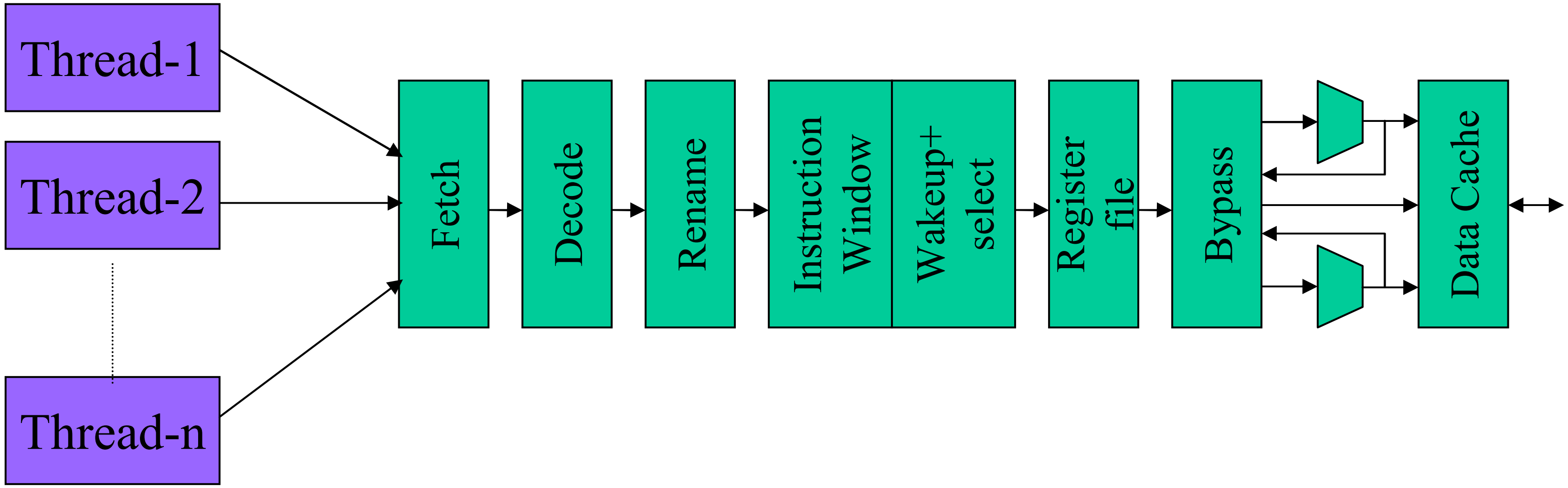
Superscalar Issue



Superscalar leads to more performance, but lower utilization

J. Emer Compaq

Multithreaded Processor

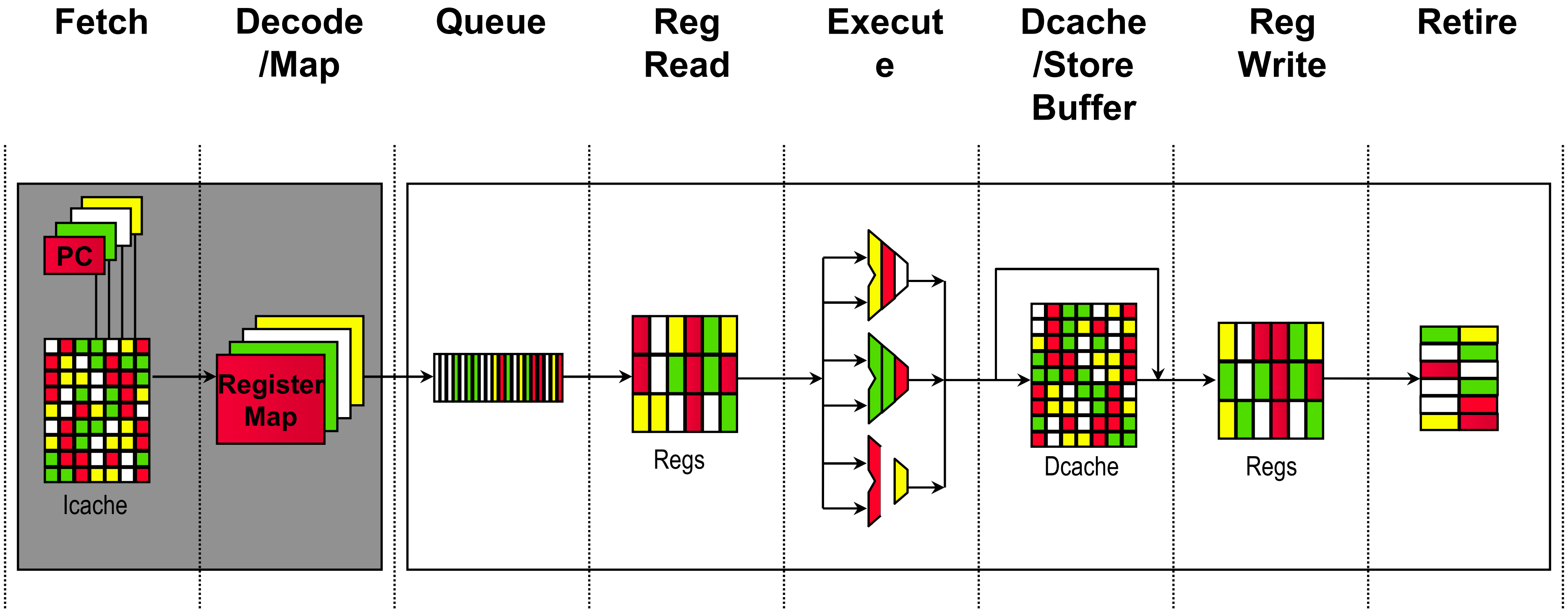


Several Threads executed concurrently

Threads belong to same / different processes

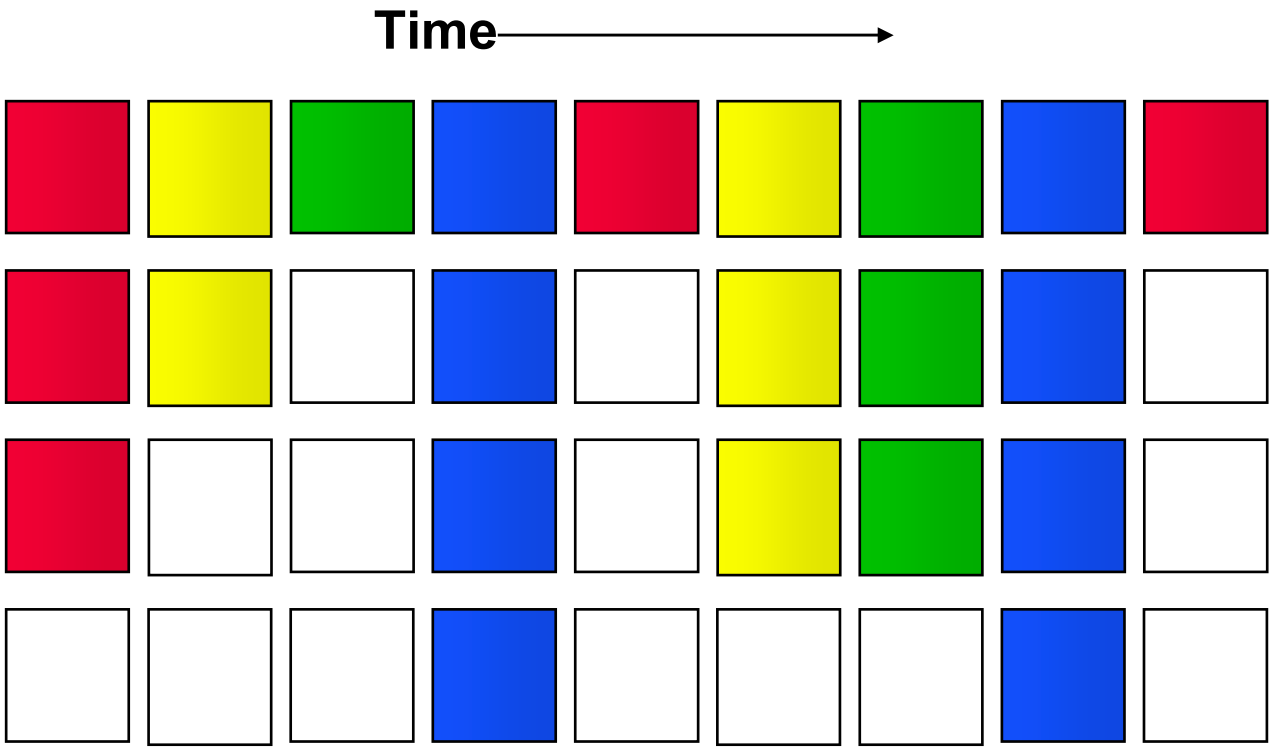
HEP (1978), Alewife , M-Machine , Tera-Computer

SMT out-of-order



J. Emer Compaq

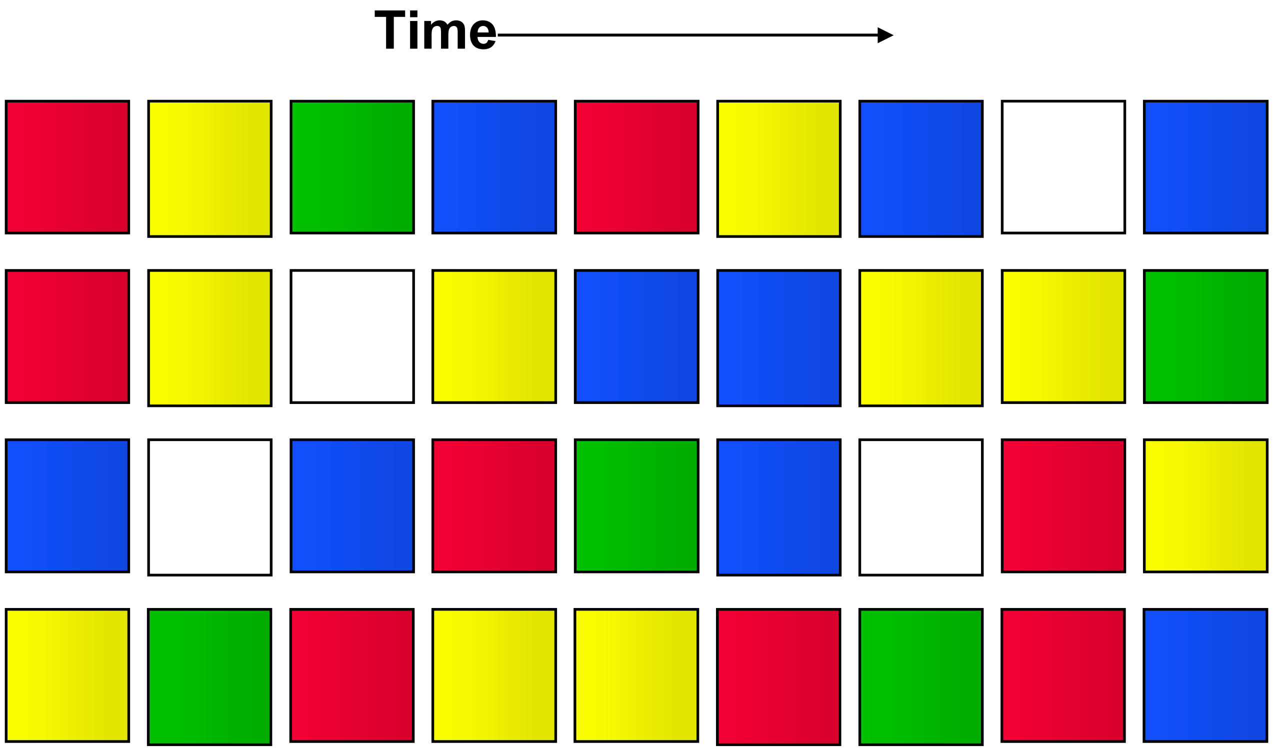
Fine Grained Multithreading



Intra-thread dependencies still limit performance

J. Emer Compaq

Simultaneous Multithreading

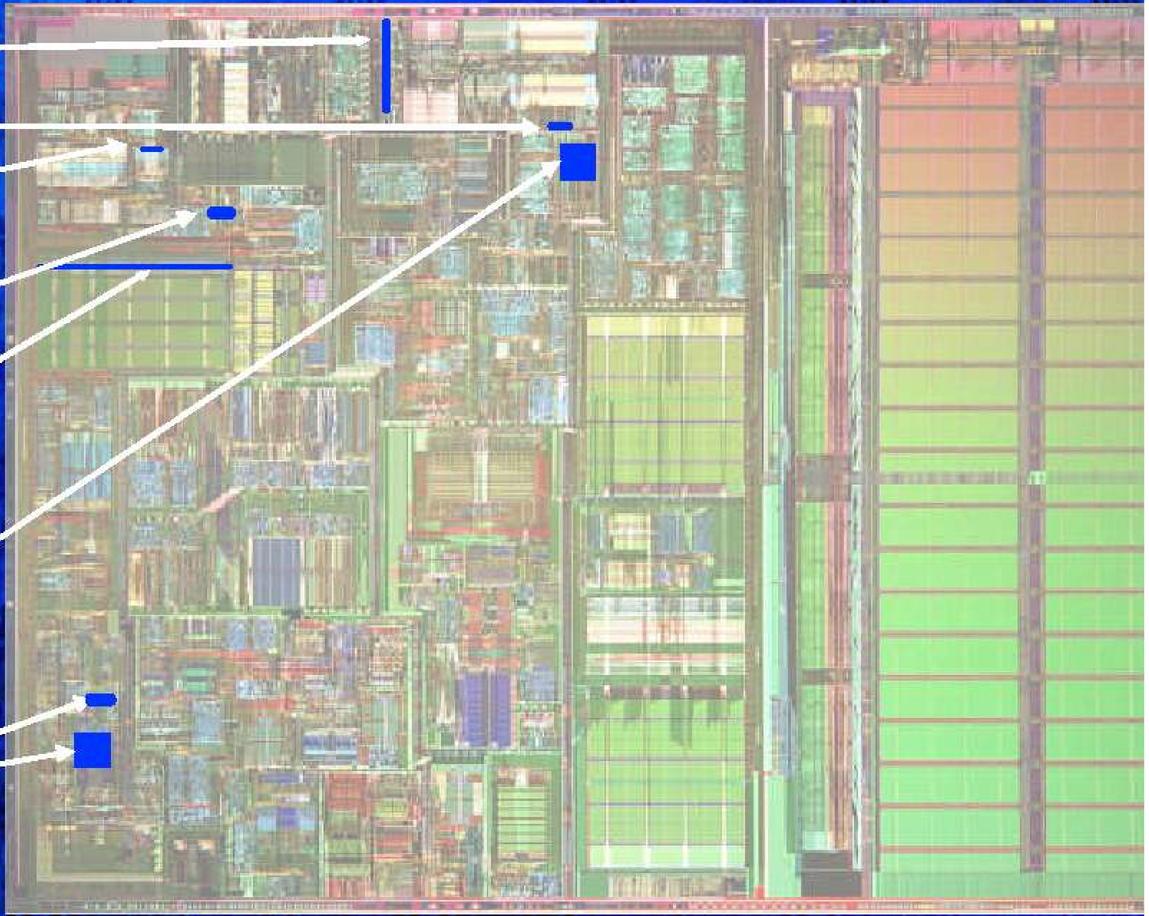


Time

Maximum utilization of function units by independent operations

J. Emer Compaq

HyperThreading Technology: What was added?



Instruction Streaming Buffers

Next Instruction Pointer

Return Stack Predictor

Trace Cache Next IP

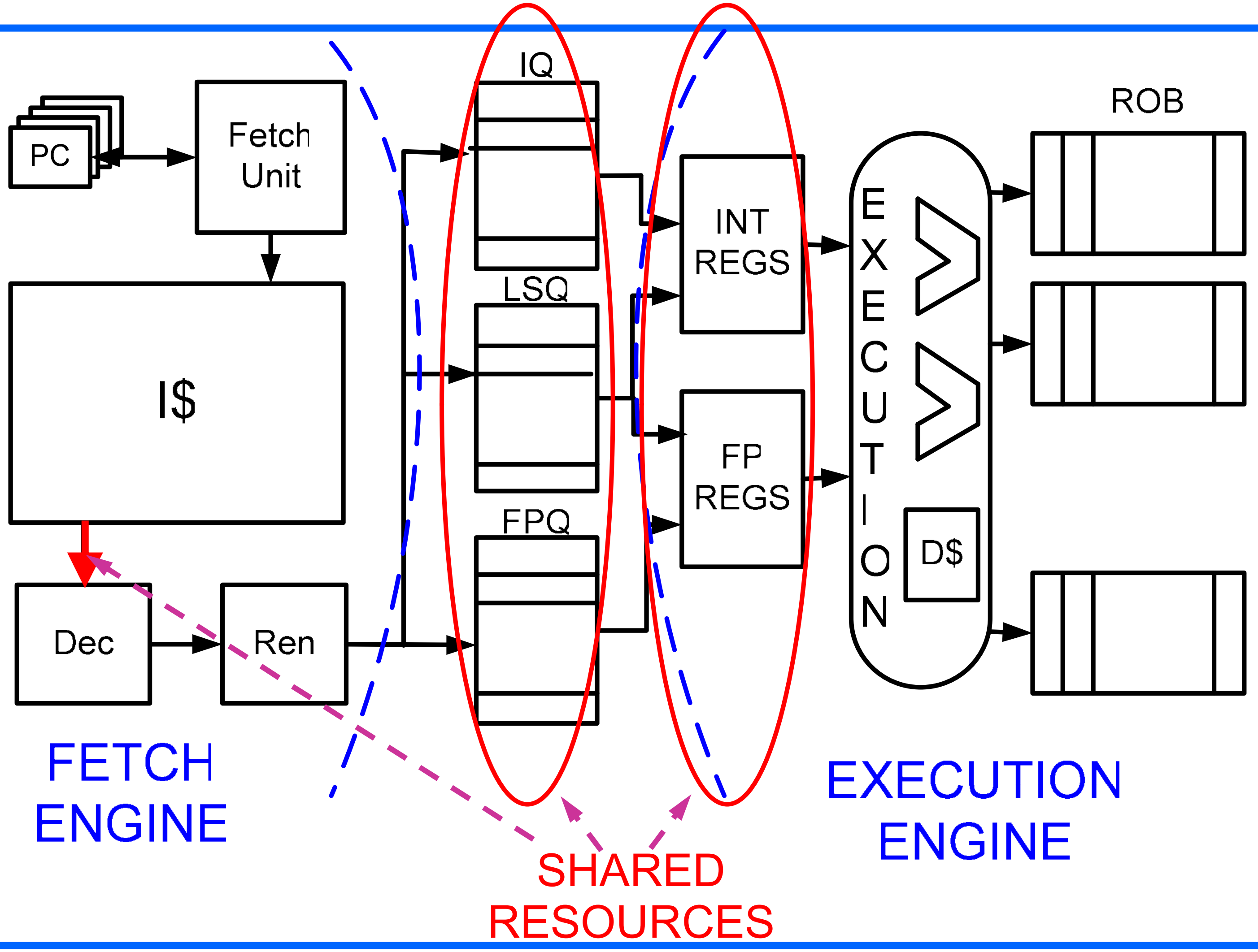
Trace Cache Fill Buffers

Instruction TLB

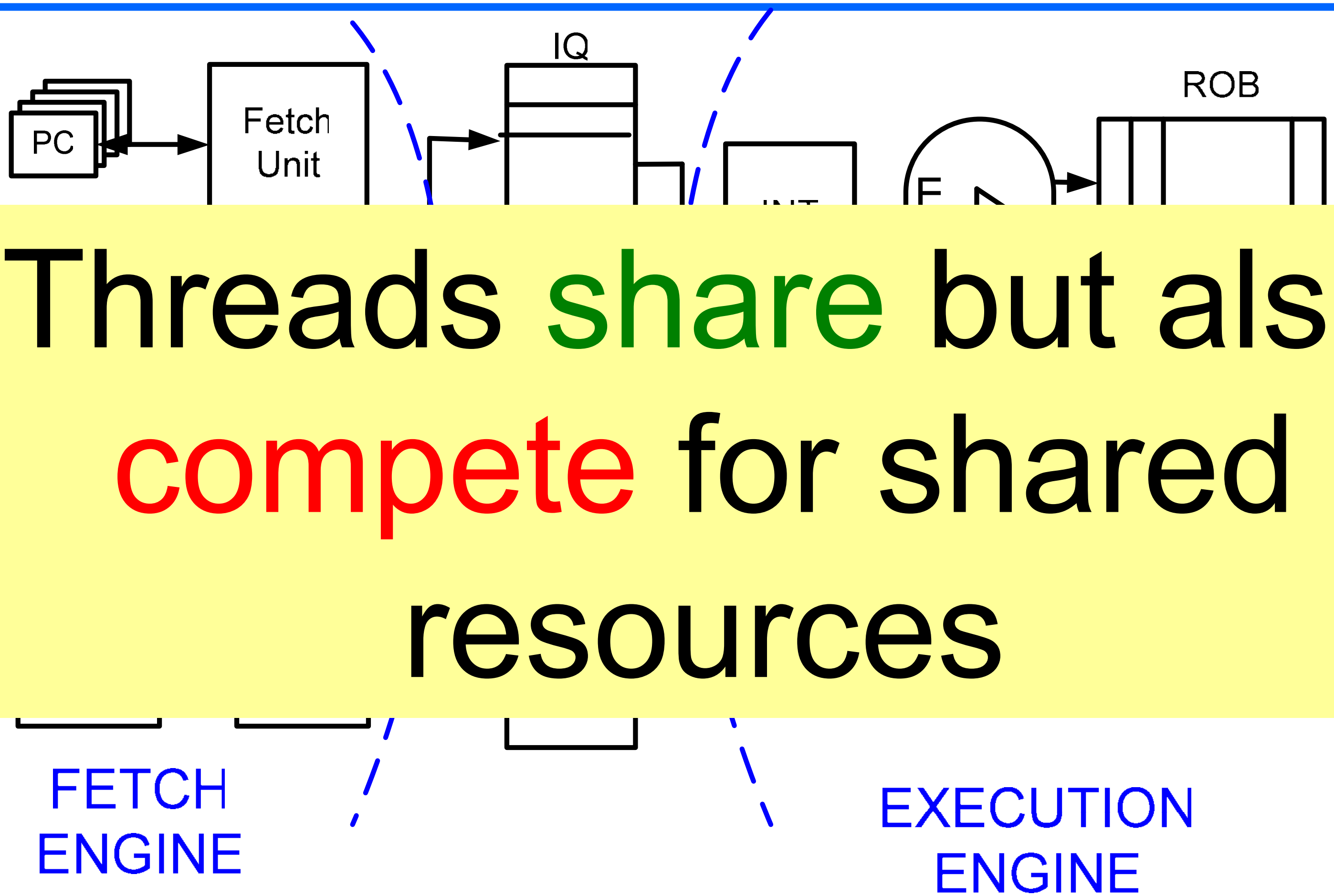
Register Alias Tables

<5% die size (& max power), up to 30% performance increas

SMT Processor



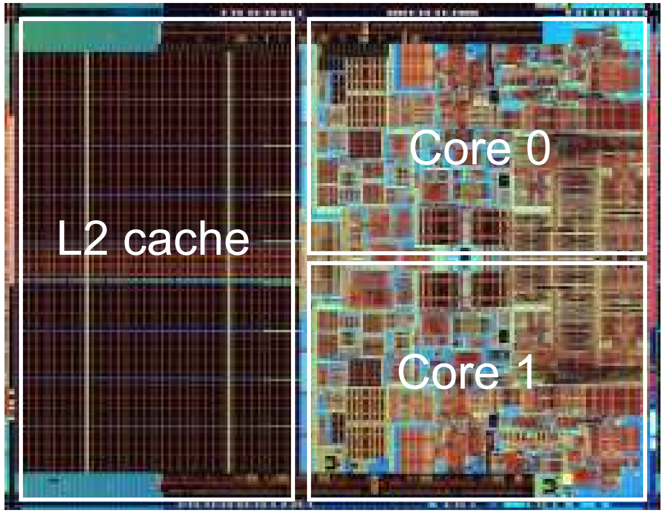
SMT Processor



Threads share but also compete for shared resources

Multi-Threaded Processors

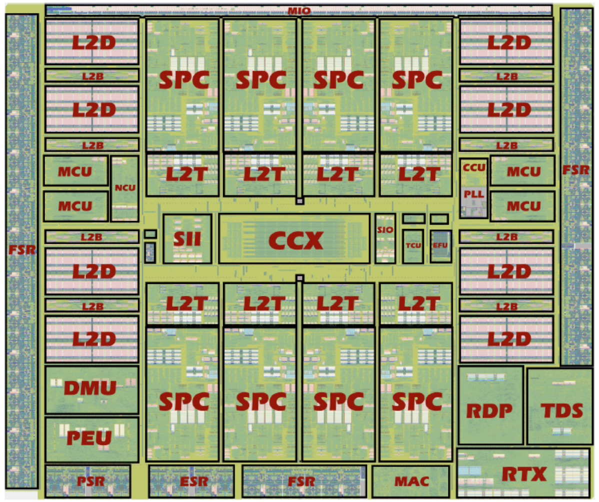
- MT processors are used in a wide range of computing systems:
  - High-performance: IBM Power5, Intel/AMD Quad-Core
  - Real-Time: Infineon Tricore, Imagination Meta
  - Network: Sun Niagara T1, Sun Niagara T2



Intel Core Duo

2cores,

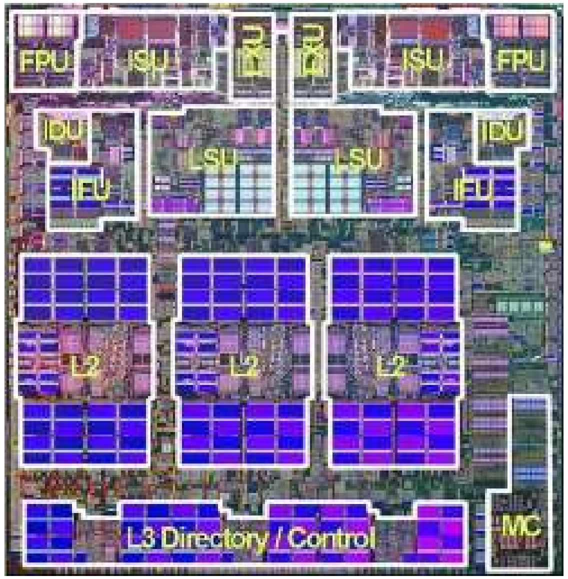
8/16-way 2MB L2 cache



Niagara T2

8 cores 8-way FGMT

12-way 3MB L2 cache



Power5

2 cores 2-way SMT

10-way 2MB L2 cache

Ifetch policies

- Icount[1]:
  - Threads with less instructions in the pre-issue stages are given high priority
  - When a thread experiences an L2 miss:
    - It can monopolize resources
    - In this case all threads are stopped
- Some Ifetch policies try to solve this problem
  - Work on top of Icount
  - Use L1/L2 data cache misses as indicators of resource monopolization,
  - Stalling/squashing threads based on these indicators

[1] Tullsen et al. "Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor " (ISCA 96)

IFetch Policies (II)

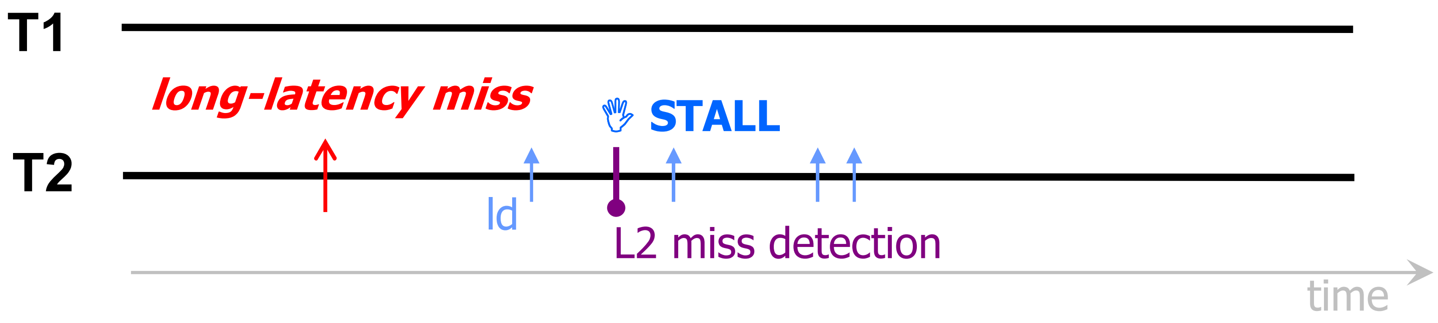
- Main fetch policies:
  - Icount[1]: fetches from threads with fewer instructions in pre-execution stages
  - Stall[2]: stops a thread if it experiences an L2 miss
  - Flush[2]: in addition, flushes all instructions after the missing load
  - Data Gating[3]: stops threads on every L1 data cache miss
- Does not take into account resource occupancy.
- Basically they are a response actions to a given event

[1] Tullsen et al. "Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor" (ISCA 96)

[2] Tullsen et al. "Handling long-latency loads in a simultaneous multithreaded processor" (MICRO 01)

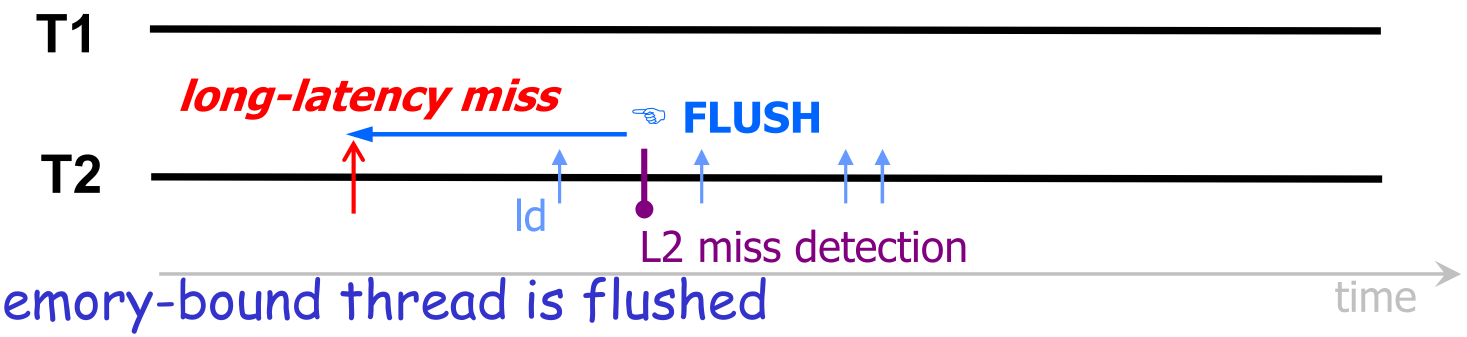
[3] El-Moursy et al. "Front-End Policies for Improved Issue Efficiency in SMT Processors" (HPCA 03)

STALL



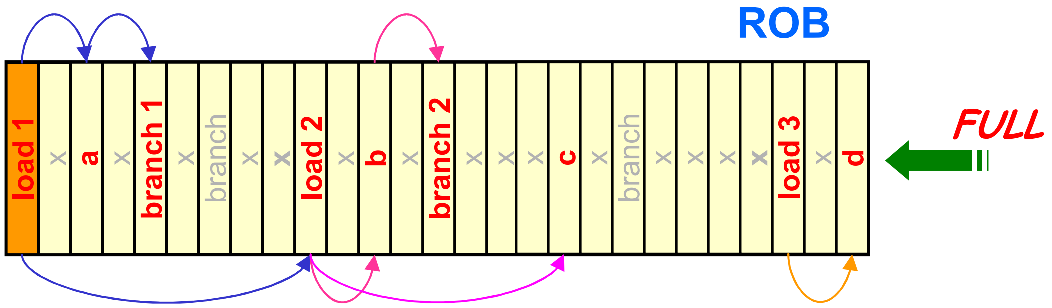
- memory-bound thread is stalled
- thread (T2) **holds the resources**

FLUSH

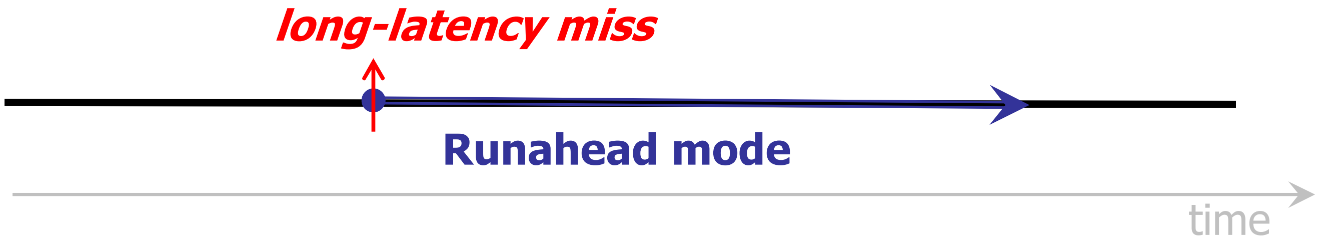


- memory-bound thread is flushed
- thread (T2) frees the resources
- but it is **also stopped**

Enter in Runahead Execution

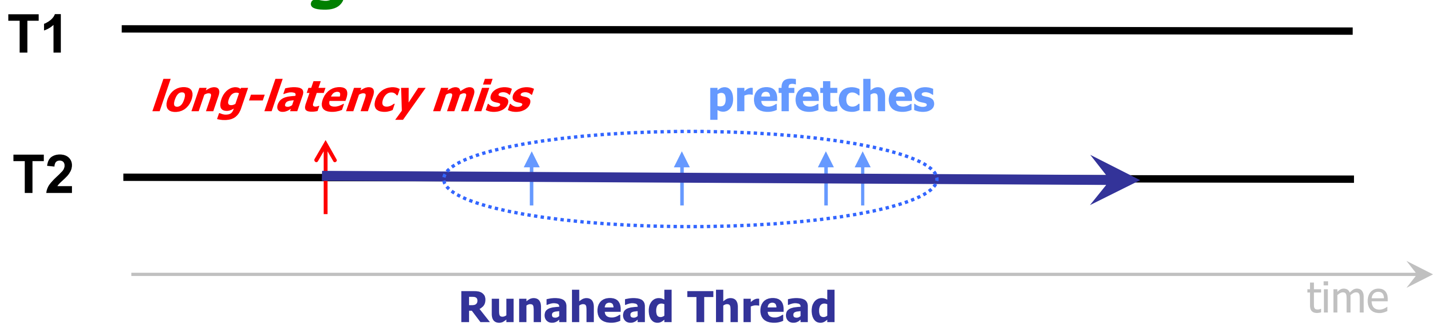


- When L2 miss reaches head of ROB
  - checkpoint architectural state and
  - enter in *runahead mode*



Sources of Improvement

- Prefetching effect

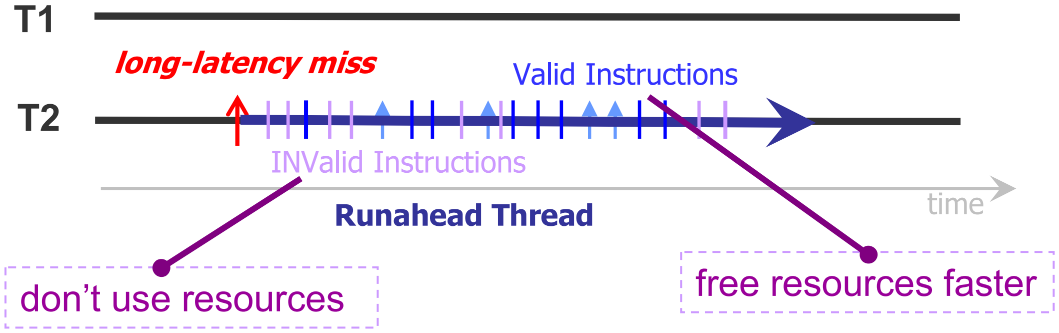


- RaT allows memory-bound threads going speculatively in advance doing prefetching
  - prefetches increase the MLP
  - each thread itself is faster by RaT, without disturbing the other threads.

[1] T. Ramirez et al. “Runahead Threads to Improve SMT Performance”. HPCA 2008

Sources of Improvement

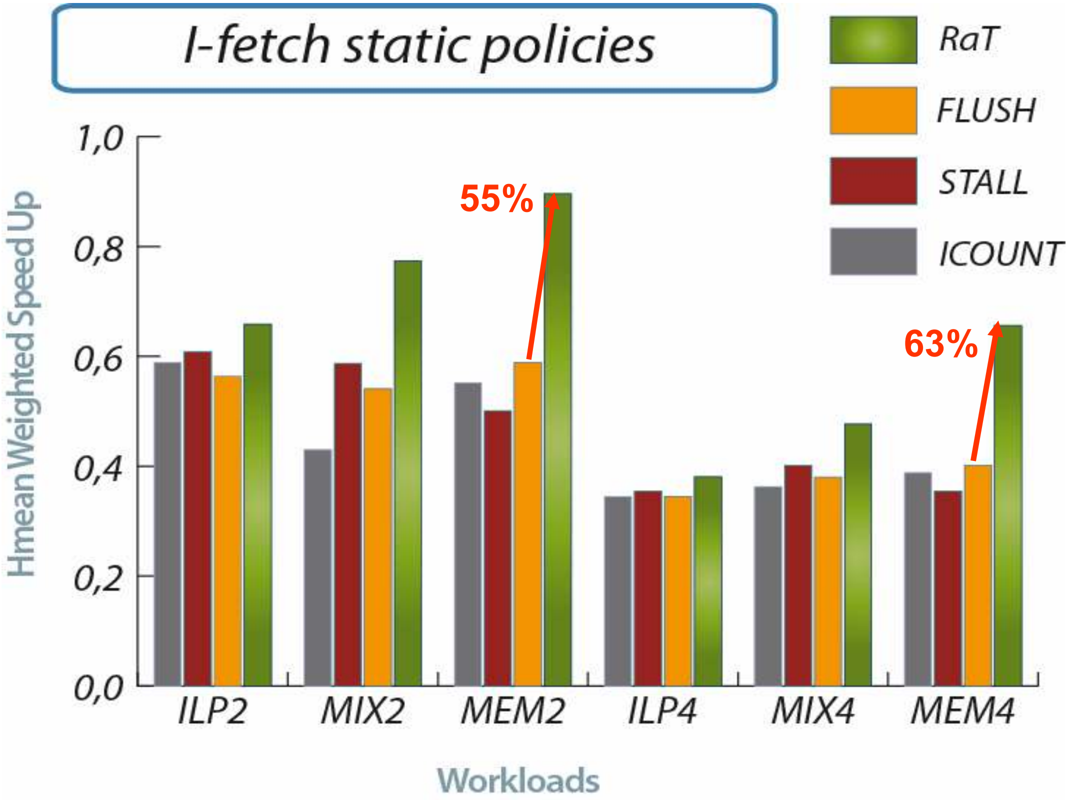
- **Alleviate resource contention**



- RaT are much less aggressive than normal ones with the important SMT resources, allocating them in short periods of time

[1] T. Ramirez et al. "Runahead Threads to Improve SMT Performance". HPCA 2008

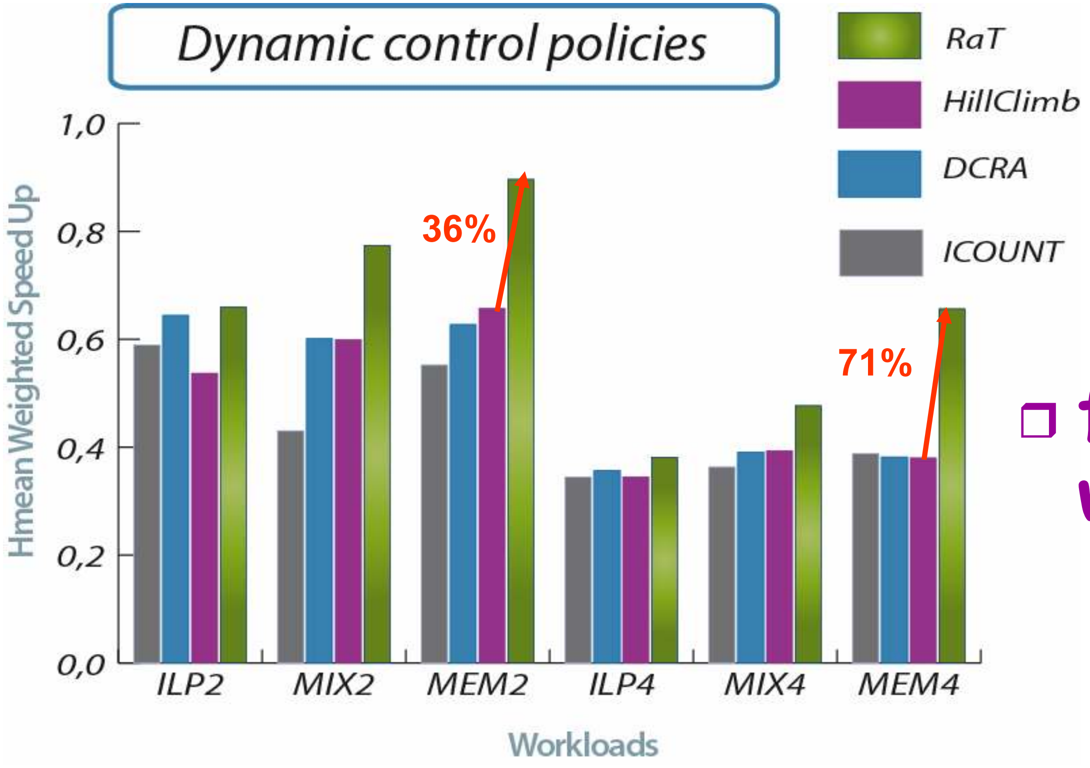
Evaluation - Fairness



- Significant for MEM workloads.

- RaT performs better than flush and stall

Evaluation – Fairness (II)

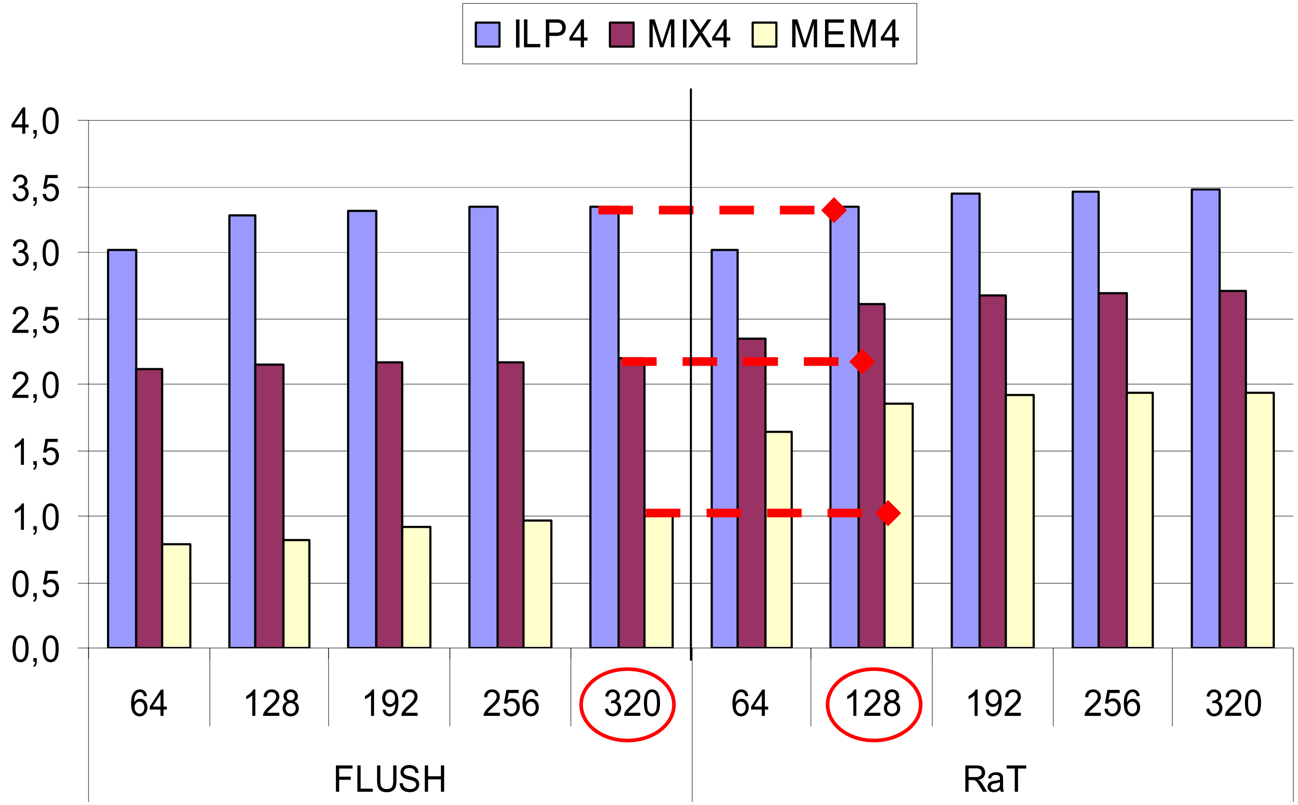


- ❑ for MEM workloads
  - 57% over DCRA
  - 53% over HillClimb

- ❑ RaT again achieves better fairness regarding dynamic control policies

RaT Sources of Improvement

- **RaT benefits** reduce register file up to 60%



UPC contribution to “kilo” processors (1 of 3)

- ❒ We started our work in June 2001
- ❒ Grant proposal to Intel-MRL in January 28th. 2002
- ❒ A. Cristal, et al. “Large virtual ROBs by processor checkpointing” *Technical Report UPC-DAC-2002-39*, July 2002. (Rejected for Micro-2002)
  - Multiple Checkpointing
  - Out-of-order Commit, No need for ROB
  - Early release of registers and loads
- ❒ A. Cristal and M. Valero, ”ROBs virtuales utilizando checkpointing”. Spanish Workshop on Parallelism. Lleida, Sept., 2002
  - Same as the previous report, but in Spanish
- ❒ A. Cristal, J. Martínez, M. Valero and J. Llosa, “Ephemeral Registers”, *Technical Report CSL-TR-2003-1035* , 2003. Rejected for ISCA 2003 and Micro 2003
  - **Ckeckpoint** + Early Release + Late allocation of registers
- ❒ A. Cristal, J. Martínez, J. LLosa and M. Valero, “ A case for resource-conscious out-of-order processors”, IEEE TCCA Computer Architecture Letters, Vol. 2, October 2003
  - Underutilization of resources

UPC contribution to “kilo” processors (2 of 3)

- ❒ A. Cristal, et al. “ A case for resource-conscious out-of-order processors: Towards Kilo-instruction in-flight processors”. MEDEA Workshop, Sept 2003 and ACM-CAN, March 2004
- ❒ A. Cristal et al. “Kilo-instruction Processors”. Invited paper. ISHPC-V.Tokyo, LNCS-2858. October 20-22th, 2003
- ❒ A. Cristal et al. “Future ILP Processors”. Invited paper. IJHPCN, to be published
- ❒ A. Cristal, et al. “Out-of-Order Commit Processors” *Technical Report UPC-DAC-2003-44*, July 2003. HPCA-10, Madrid, Feb. 2004
  - Unified mechanism for Out-of-Order Commit and IQs management
  - Use of Checkpointing and Ephemeral Registers
- ❒ M. Galluzzi et al. “ A First glance at Kiloinstruction Based Multiprocessors” Invited Paper. ACM Computing Frontiers Conference. Ischia, Italy, April 10-12, 2004
- ❒ E. Vallejo, M. Galluzzi, A. Cristal, F. Vallejo, R. Beivide, Per Stenström, James E. Smith and Mateo Valero. “Implementing Kilo-Instruction Multiprocessors”. Invited paper. IEEE Conference on Pervasive Services, ICPS-05. Santorini, Greece. July 11-14, 200
- ❒ M. Galluzzi, E. Vallejo, A. Cristal, F. Vallejo, R. Beivide, P. Stenstrom, J. Smith and M. Valero. “Implicit Transactional Memory in Kilo-Instruction Multiprocessor”. Invited paper. ACSAC-2007. The Twelfth Asia-Pacific Computer Systems Architecture Conference. Seoul, Korea, August 23-25, 2007. LNCS 4697, pp.339-353

UPC contribution to “kilo” processors (3 of 3)

- Pericas et al. “A decoupled KILO-Instruction Processor”, HPCA-12, Austin, Feb 2006
  - Execution Locality Concept
  - Decoupled Processor (Cache/Memory Processor)
- Pericas et al. “A Flexible Heterogeneous MultiCore Architecture”, PACT-16, Brasov, Sept 2007
  - DataFlow Memory Processor
  - Shared Memory Processor
- Pericas et al. “A Two-Level Load/Store Queue based on Execution Locality”, ISCA-35, June 2008
  - Locality-based LSQ + Local/Global Disambiguation
  - Completes execution-locality based KILO-Instruction Processor
- More work being conducted at this point

Talks about “Kilo” processors, from UPC

- ❐ Presentation in Barcelona, to Intel-MRL in February 2002
- ❐ Spanish Workshop on Parallelism. Lleida, Sept., 2002
- ❐ Presentation to Intel-MRL in March 2003
- ❐ Invited presentation. NSF Panel “On the Future of Computer Architecture Research: Wise Views and Fresh Perspectives”. San Diego, June 2003
- ❐ Invited Lecture. PA3CT Conference. Edegem, Belgium, September 22-23, 2003
- ❐ MEDEA Workshop. New Orleans, September 2003
- ❐ Invited Lecture. ISHPC-V. The 5th International Symposium on High Performance Computing. Tokyo, Japan, October 20-22, 2003
- ❐ Keynote lecture. Seminar on Compilers and Architecture. IBM Haifa. November 11th., 2003.
- ❐ Invited lecture. Intel MRL. Haifa., Israel. Nov. 12th., 2003
- ❐ HPCA-10, Madrid, February 14-18, 2003
- ❐ Keynote lecture. HPCA-10. Madrid, February 14-18, 2003
- ❐ Invited lecture. ACM Computing Frontiers. Ischia, April, 2004
- ❐ ACM Invited lecture. ENCAR México, May 2004
- ❐ Keynote Lecture. Europar. Pisa, September 2004
- ❐ Distinguish lecture, Irvine, January 2006
- ❐ HPCA-12, Austin, February 2006
- ❐ PACT-16, Brasov, September 2007
- ❐ … several keynotes and Distinguish Lectures from ACM (India, México,..)
- ❐ Onassis Foundation. Haraklion, Crete, July 2008

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.

- ❒ Tejas Karkhanis and James E. Smith. "A Day in the Life of a Data Cache Miss", 2nd Annual Workshop on Memory Performance Issues (WMPI), June, 2002.

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 resources allocated for 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.

Checkpointing

- J.E. Smith and A.R. Plezskun "Implementing Precise Interrupts in Pipelined Processors", ISCA-12, 1985
- 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, No need for ROB
        - 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, Nov. 2002.
        - One checkpoint
        - Early release of resources

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,* 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* , 2003.
  - **Ckeckpoint** + Early Release + Late allocation of registers
- ❐ T. Monreal et al., “Late allocation and early release of physical registers”, IEEE-TC (to appear in October 2004)

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

References for LSQ for Large ROB

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

- 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

- H. Akkari, R. Rajwar and S. T. Srinivasan “Checkpointing Processing and Recovery: Towards Scalable Large Instruction Window Processors” Micro-36, 2003

- S. Sethumadhavan, R. Desikan, D. Burger, C.R. Moore and S. W. Keckler “Scalable Hardware Memory Disambiguation for High ILP Processors” Micro-36, 2003

- H. W. Cain et al. “Memory Ordering: A Value-based approach”, ISCA-32, 2004

- A. Roth, “Store Vulnerability Window (SVW): Re-Execution Filtering for Enhanced Load Optimization”, ISCA‘-33, 2005

Conclusion

- Affordable "Kilo-instruction Processors"
- Checkpointing and resource-conscious architectures
  - Out-of- order commit
  - Ephemeral registers
  - Two-level instruction queues
  - Early release of loads
  - Load/store queue management
- New ideas to watch for
  - Better branch predictors
  - Predication and Multi-path execution
  - Control and data independent instructions
  - Reuse of large blocks of instructions
- New processor paradigms:
  - "Kilo-based" multiprocessor systems
  - "Kilo-vector" processors
  - "Kilo-SMT" processors
  - "Kilo-valpred" processors

Acknowledgments

- Adrián Cristal
- José Martínez

- Josep Llosa
- Daniel Ortega
- Fran Cazorla
- Enrique Fernández
- Oliver Santana
- Ayose Falcón
- Alex Pajuelo
- Marco Galluzzi
- Tanausu Ramírez
- Miquel Pericas
- Jim Smith

- Yale Patt
- Alex Veidenbaum
- Guri Sohi
- Mark Hill
- Wen-mei Hwu
- "Mon" Beivide
- Valentín Puente
- José Angel Gregorio
- Teresa Monreal
- Victor Viñals

- Intel, Konrad Lai and Ronny Ronen

Thank you very much ☺