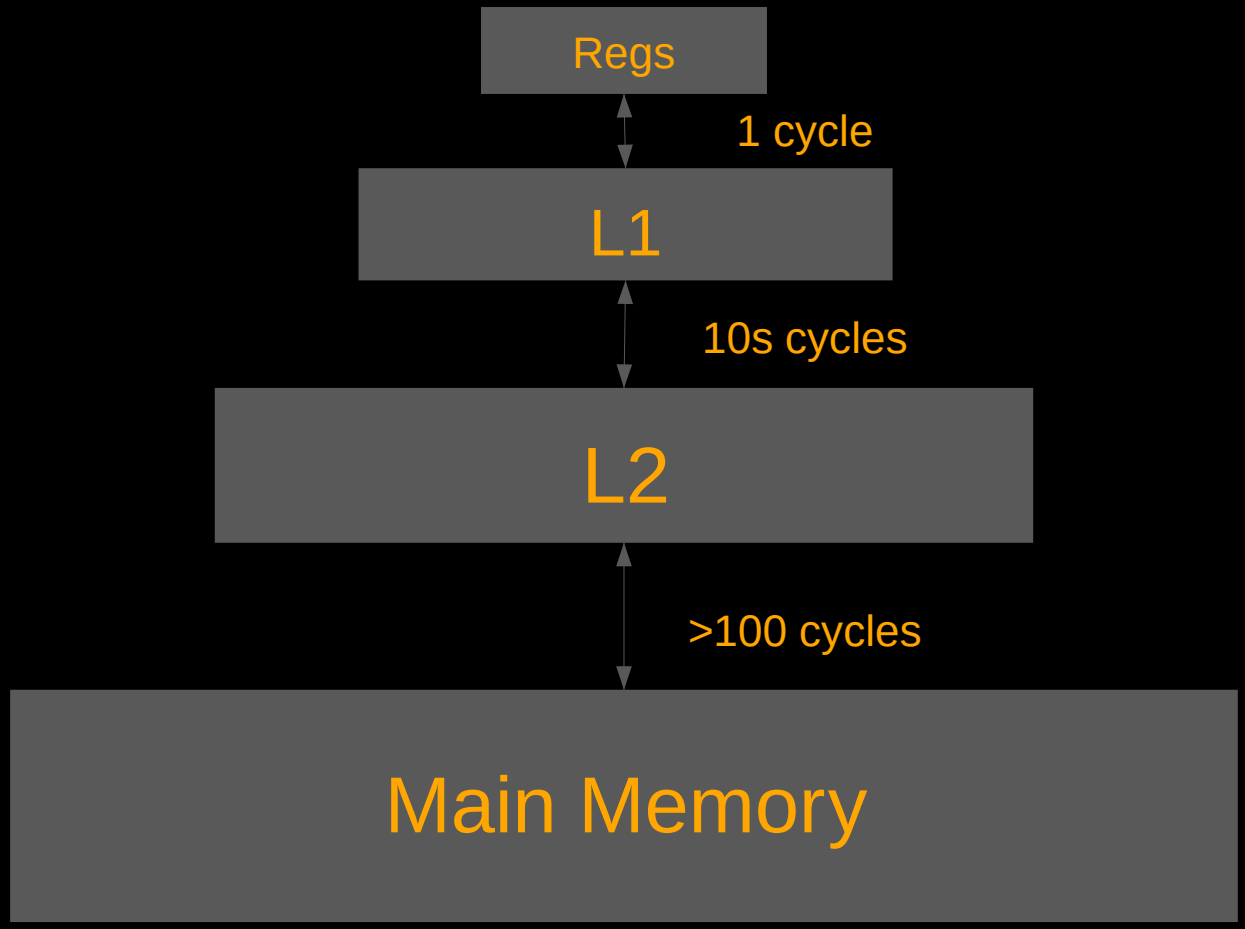
Cost-Effective Compiler Directed Memory Prefetching and Bypassing

Daniel Ortega,[†], Eduard Ayguadé[†], Jean-Loup Baer[‡] and Mateo Valero[†]

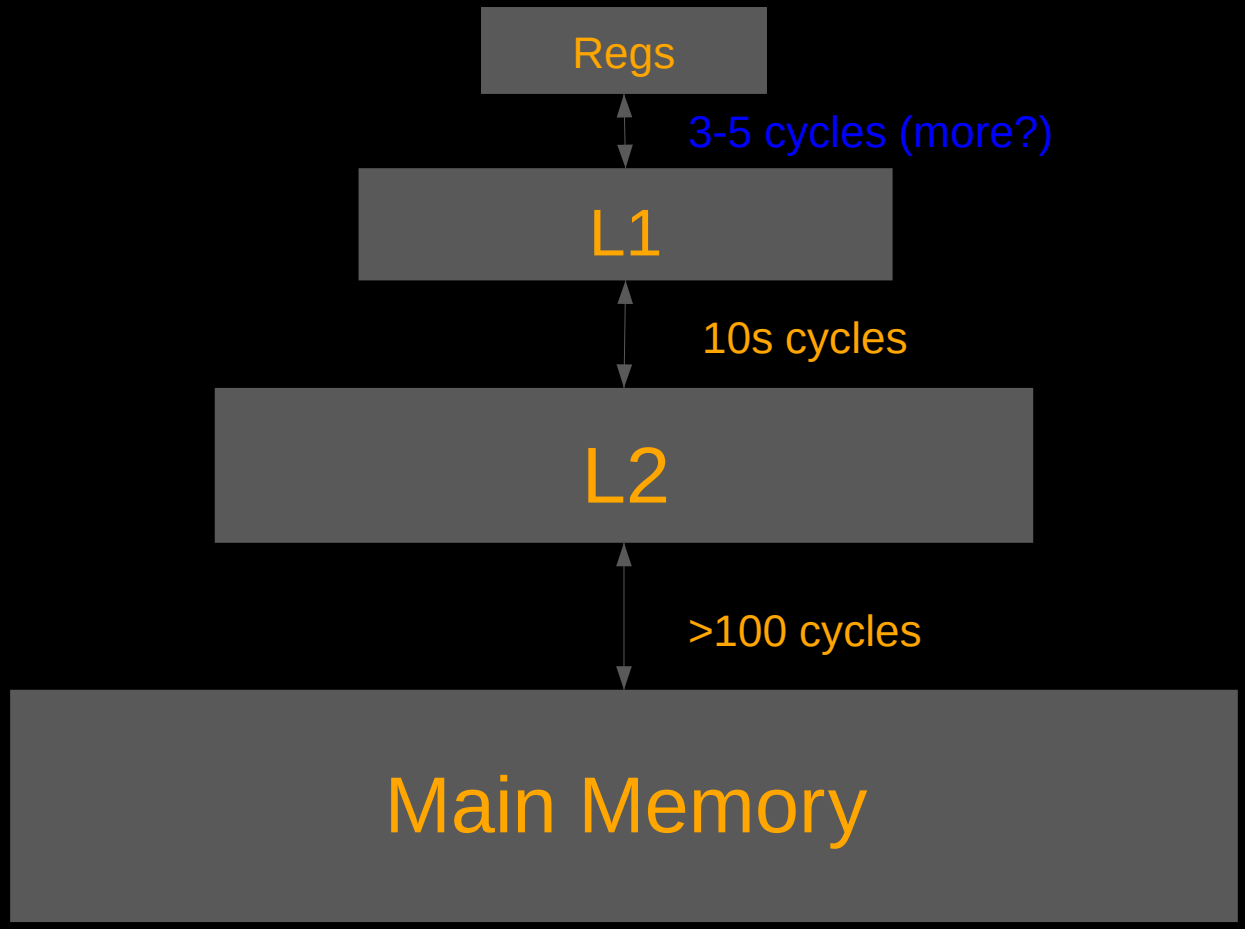
[†]Departamento de Arquitectura de Computadores,
Universidad Politécnica de Cataluña – Barcelona
{dortega,eduard,mateo}@ac.upc.es

[‡]Department of Computer Science and Engineering,
University of Washington – Seattle
baer@cs.washington.edu

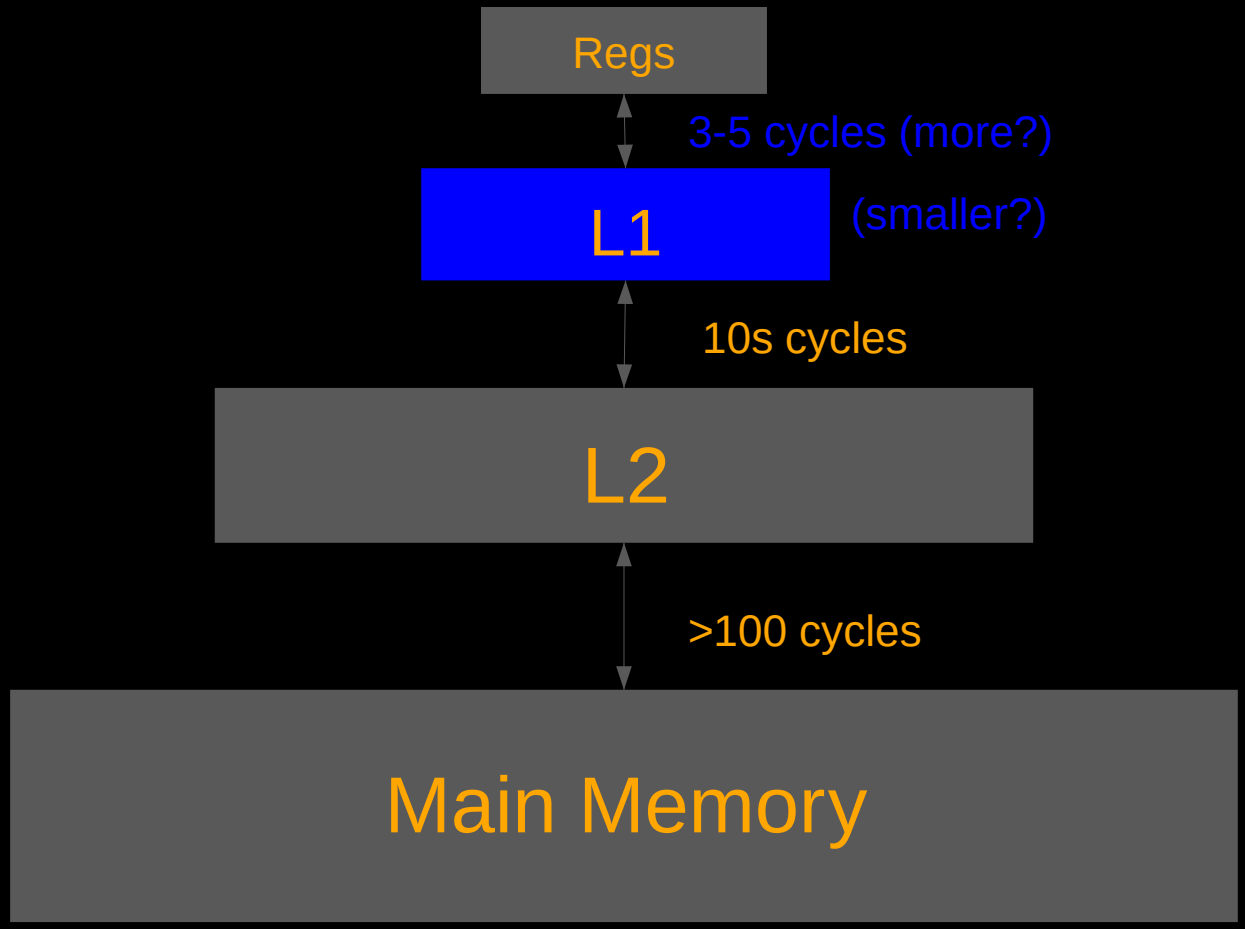
Conventional Cache Hierarchy



Conventional Cache Hierarchy



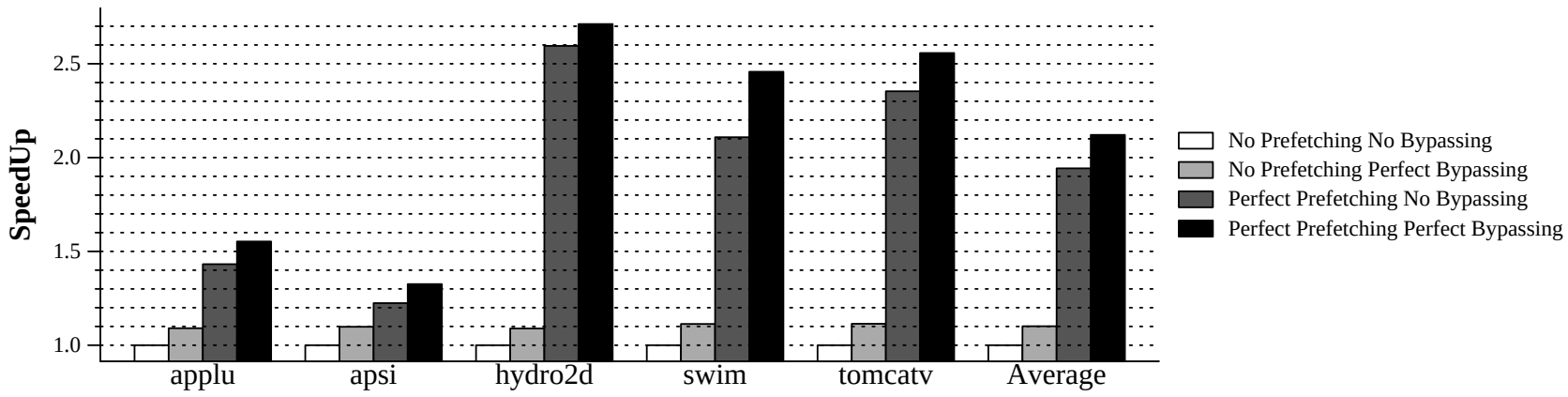
Conventional Cache Hierarchy



Our approach

- Attack register-L1 gap with memory instruction bypassing
- Use hardware prefetcher for L1-L2 gap
  - ⋆ Directed by the software
  - ⋆ thus simple Hardware
  - ⋆ no recovery needed in case of misprediction

Limit Study

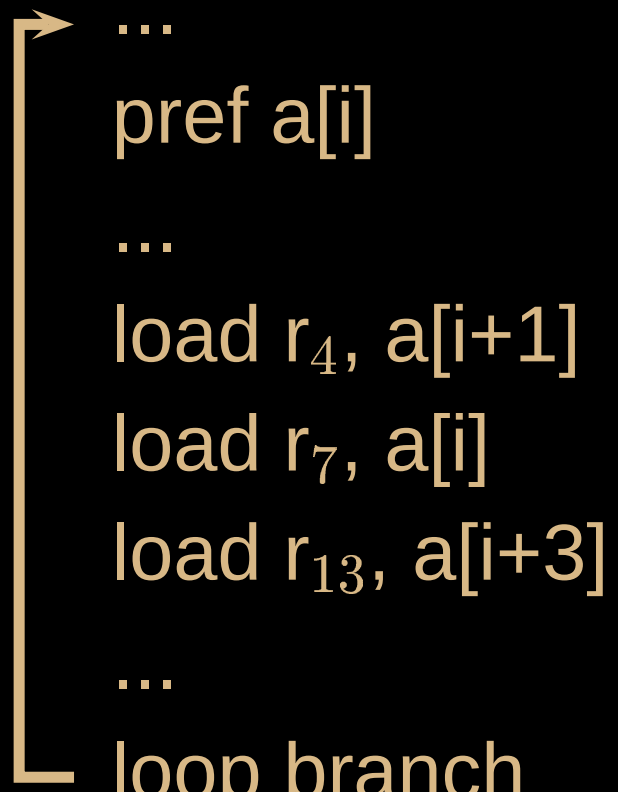


Limit values for a 4-way machine

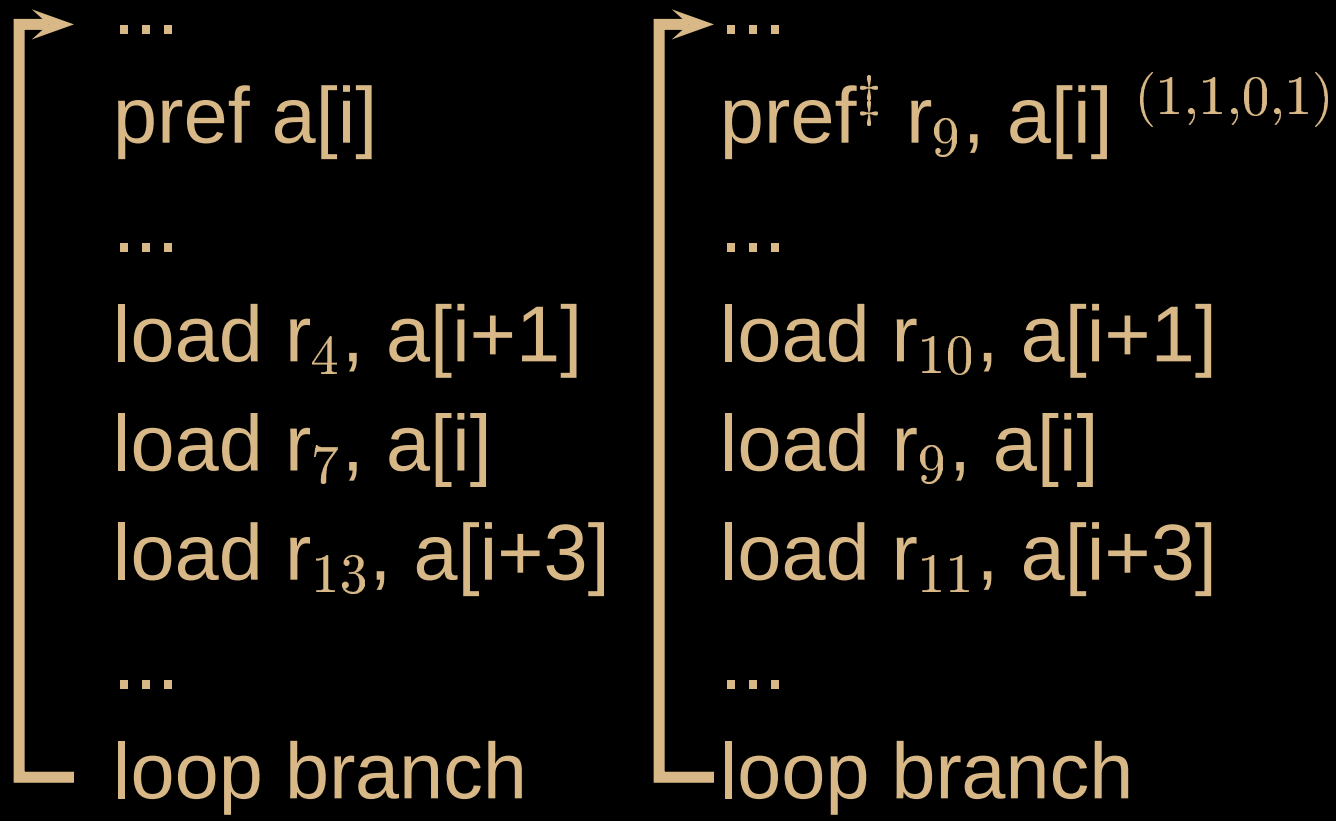
Index

- Motivation
- Memory Instruction Bypassing
- Compiler Directed Memory Prefetcher
- Comparison with APDP

MIB through Renaming I



MIB through Renaming I



MIB through Renaming II

Renaming Table

$\ldots$	$\rightarrow$	$\ldots$
r_9	$\rightarrow$	f_{10}
r_{10}	$\rightarrow$	f_{14}
r_{11}	$\rightarrow$	f_9
$\ldots$	$\rightarrow$	$\ldots$

Special Renaming Table

$\ldots$	$\rightarrow$	$\ldots$
r_9	$\rightarrow$	$\ldots$
r_{10}	$\rightarrow$	$\ldots$
r_{11}	$\rightarrow$	$\ldots$
$\ldots$	$\rightarrow$	$\ldots$

MIB through Renaming II

...

pref‡ r_9, a[i] $^{(1,1,0,1)}$

Renaming Table

...	$\rightarrow$	...
r_9	$\rightarrow$	f_{10}
r_{10}	$\rightarrow$	f_{14}
r_{11}	$\rightarrow$	f_9
...	$\rightarrow$	...

Special Renaming Table

...	$\rightarrow$	...
r_9	$\rightarrow$	f_{21}
r_{10}	$\rightarrow$	f_{32}
r_{11}	$\rightarrow$	f_{36}
...	$\rightarrow$	...

MIB through Renaming II

...

pref‡ r_9, a[i] $^{(1,1,0,1)}$

...

load r_{10}, a[i+1]

Renaming Table

...	→	...
r_9	→	f_{10}
r_{10}	→	f_{32}
r_{11}	→	f_9
...	→	...

Special Renaming Table

...	→	...
r_9	→	f_{21}
r_{10}	→	...
r_{11}	→	f_{36}
...	→	...

MIB through Renaming II

...

pref‡ r_9, a[i] $^{(1,1,0,1)}$

...

load r_{10}, a[i+1]

load r_9, a[i]

Renaming Table

...	→	...
r_9	→	f_{21}
r_{10}	→	f_{32}
r_{11}	→	f_9
...	→	...

Special Renaming Table

...	→	...
r_9	→	...
r_{10}	→	...
r_{11}	→	f_{36}
...	→	...

MIB through Renaming II

...

pref‡ r_9, a[i] $^{(1,1,0,1)}$

...

load r_{10}, a[i+1]

load r_9, a[i]

load r_{11}, a[i+3]

Renaming Table

...	→	...
r_9	→	f_{21}
r_{10}	→	f_{32}
r_{11}	→	f_{36}
...	→	...

Special Renaming Table

...	→	...
r_9	→	...
r_{10}	→	...
r_{11}	→	...
...	→	...

MIB through Renaming II

...

pref‡ r_9, a[i] $^{(1,1,0,1)}$

...

load r_{10}, a[i+1]

load r_9, a[i]

load r_{11}, a[i+3]

...

loop branch

Renaming Table

...	→	...
r_9	→	f_{21}
r_{10}	→	f_{32}
r_{11}	→	f_{36}
...	→	...

Special Renaming Table

...	→	...
r_9	→	...
r_{10}	→	...
r_{11}	→	...
...	→	...

Index

- Motivation
- Memory Instruction Bypassing
- Compiler Directed Memory Prefetcher
- Comparison with APDP

Decoupled Prefetcher

- Compiler inserts prefetching operations
- Instructions bring data closer to the processor
- Memory instruction bypassing takes care of bringing data to the register file from L1
- Compiler also instructs prefetching hardware to prefetch ahead (to L1)
- No.s of prefetches are minimised by compiler control

Prefetching Mechanism

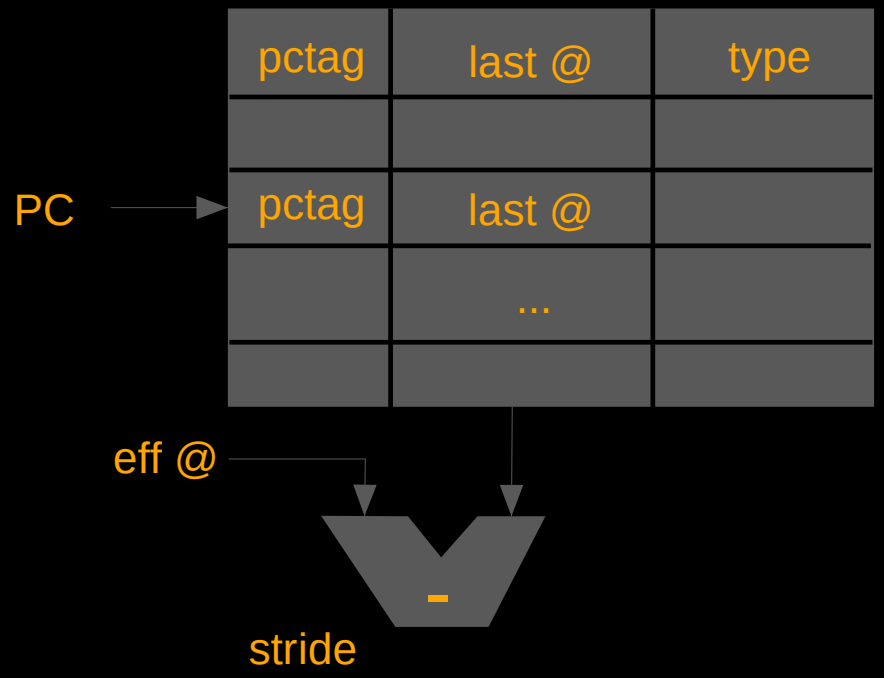
pctag	last @	type
	...	

Prefetching Mechanism

- Decode phase

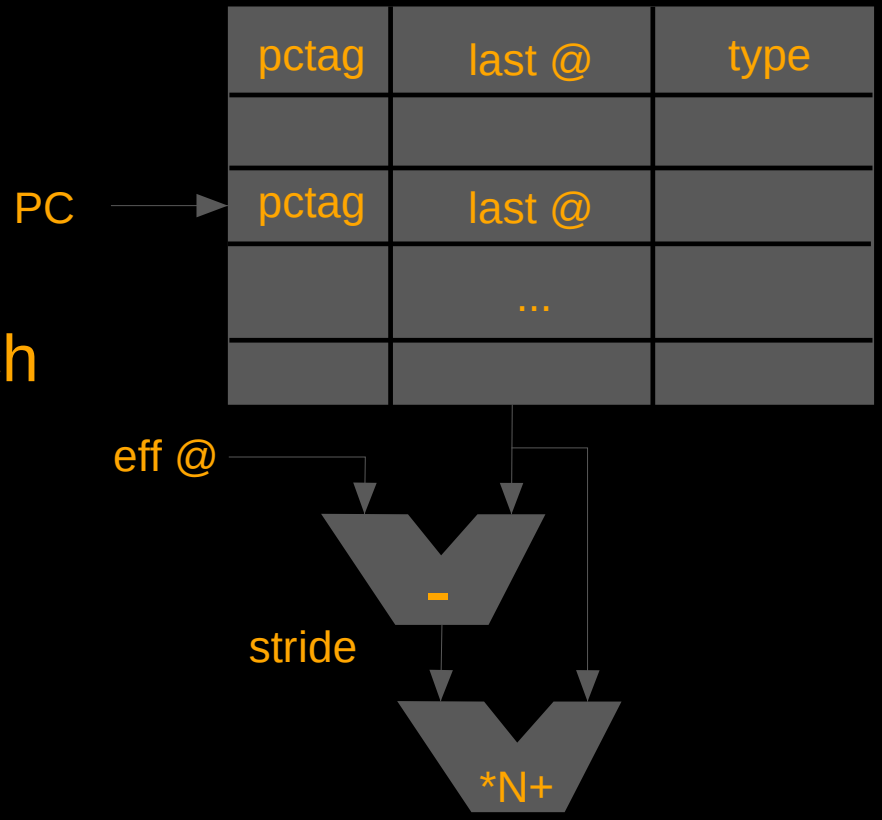
Prefetching Mechanism

- Decode phase
- Address calc.



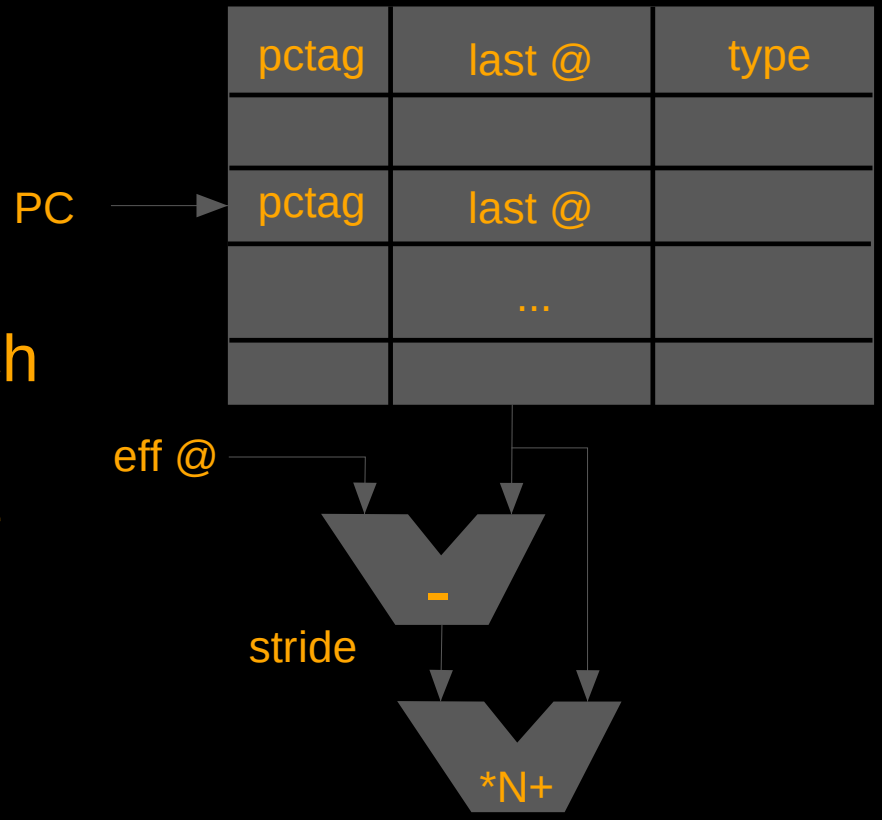
Prefetching Mechanism

- Decode phase
- Address calc.
- Generate new prefetch



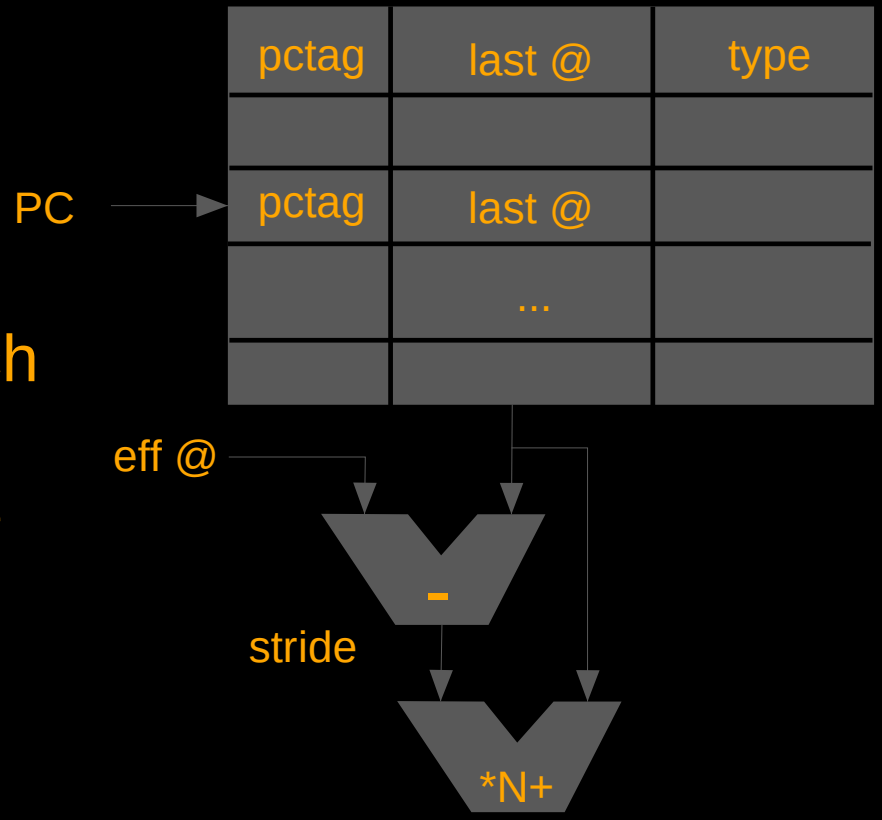
Prefetching Mechanism

- Decode phase
- Address calc.
- Generate new prefetch
  - ⋆ N depends on type

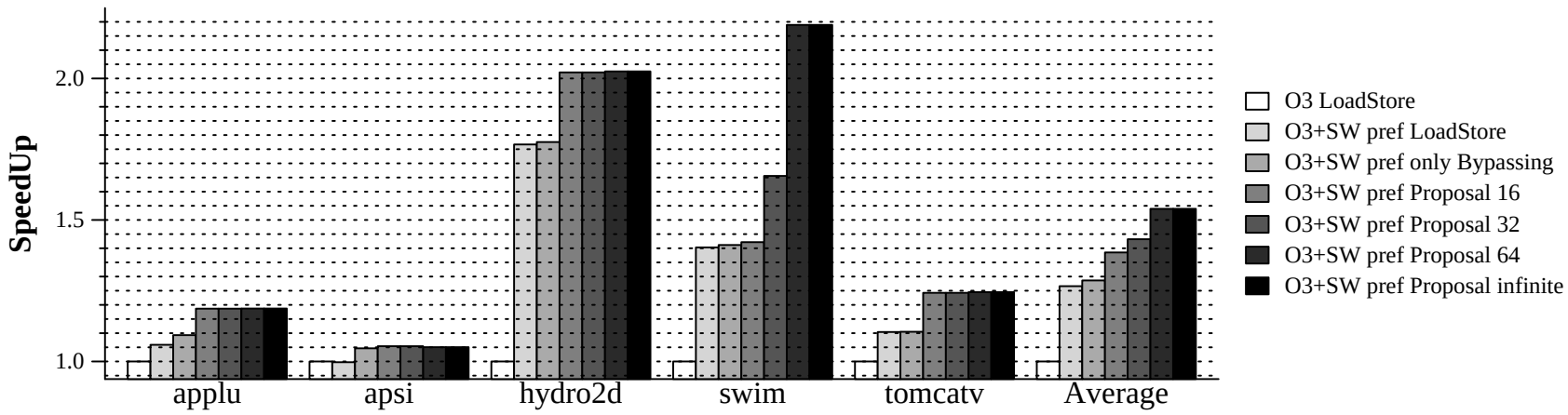


Prefetching Mechanism

- Decode phase
- Address calc.
- Generate new prefetch
  - ⋆ N depends on type
  - ⋆ wait for free port

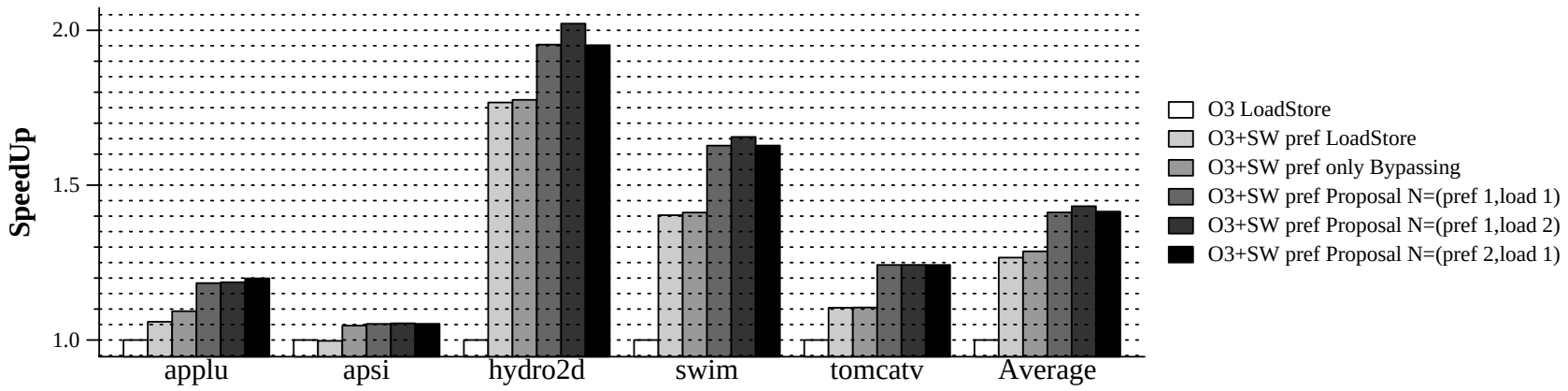


Performance Results



Effect of number of streams in a 4-way machine

Performance Results II



Effect of lookahead policy in a 4-way machine with 32 entries

Index

- Motivation
- Memory Instruction Bypassing
- Compiler Directed Memory Prefetcher
- Comparison with APDP

Comparison with APDP‡

- Address Prediction for Data Prefetching
  - predicts addresses of memory operations
  - makes prediction available as soon as load arrives to decoding
  - needs recovery mechanism in case of misprediction
  - dynamic mechanism

‡J. González and A. González, ICS 97

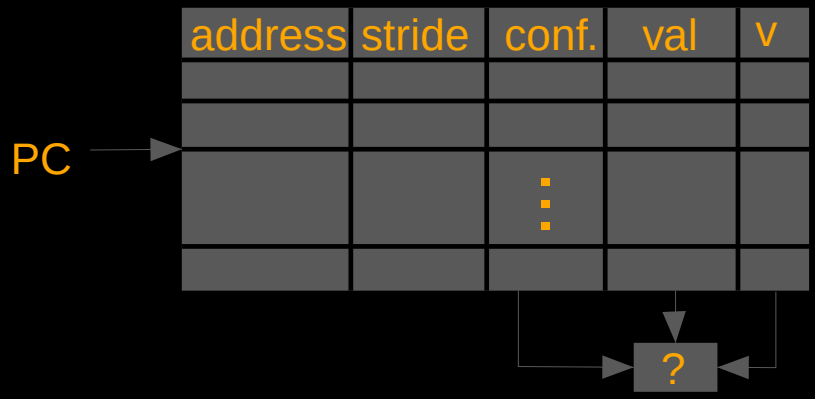
Prefetching Mechanism

Prefetching Mechanism

- Decode phase

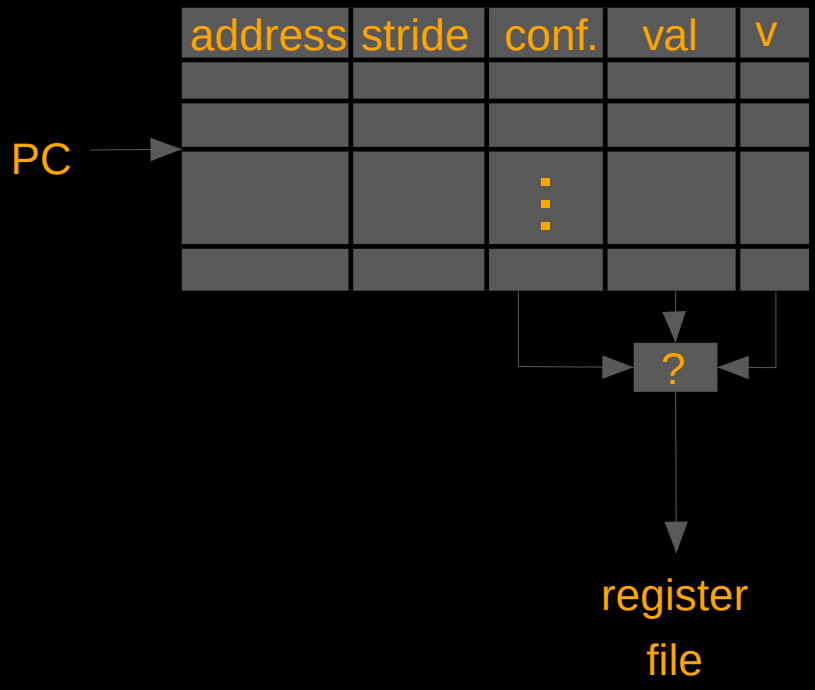
Prefetching Mechanism

- Decode phase
- Is value correct?



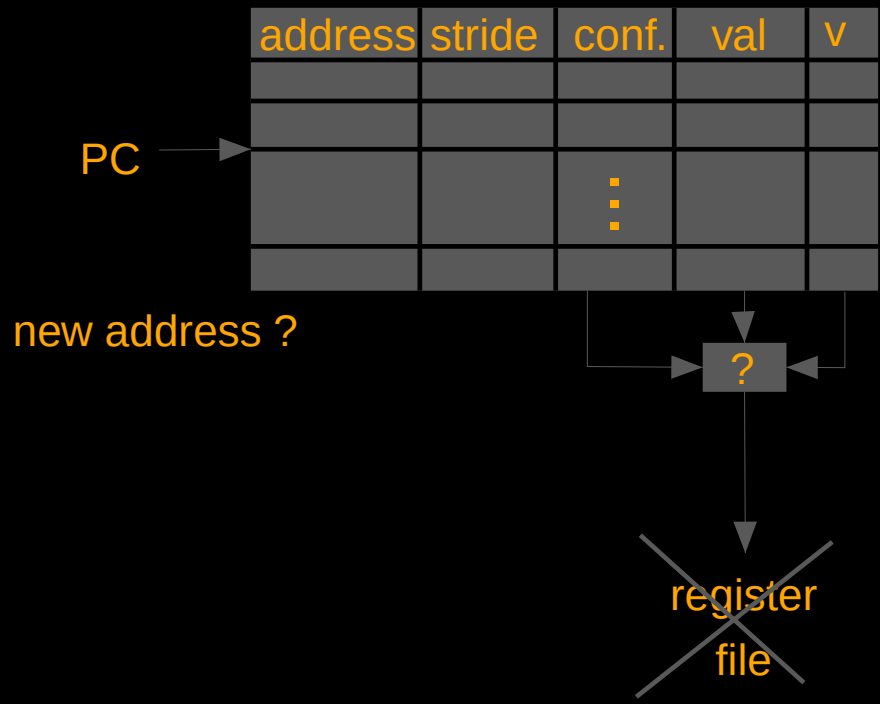
Prefetching Mechanism

- Decode phase
- Is value correct?
- Yes $\rightarrow$ bypass



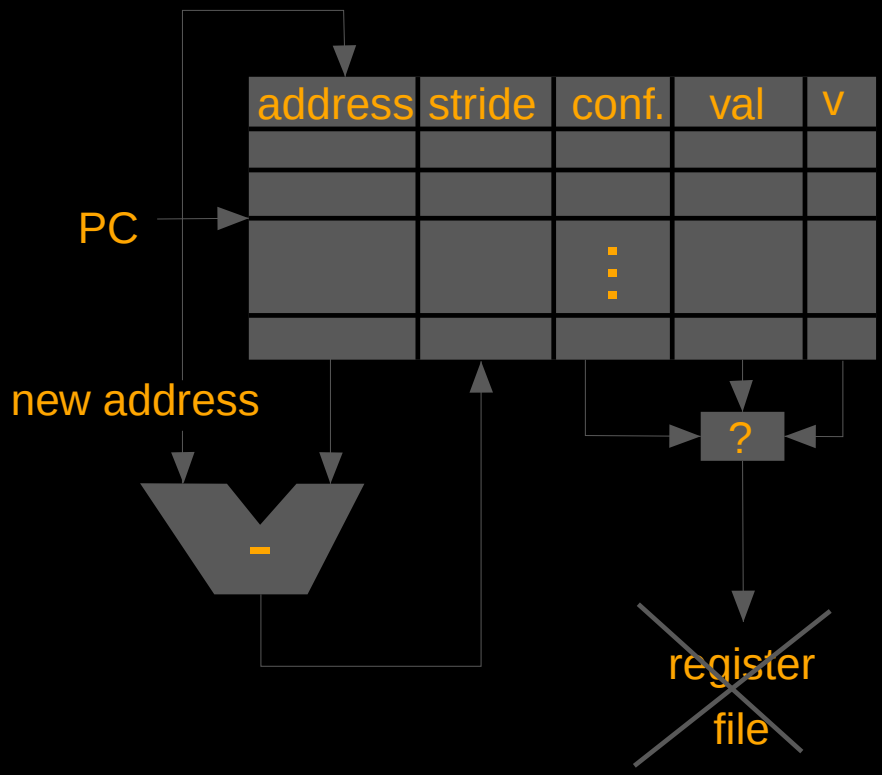
Prefetching Mechanism

- Decode phase
- Is value correct?
- Yes $\rightarrow$ bypass
- Address calc.



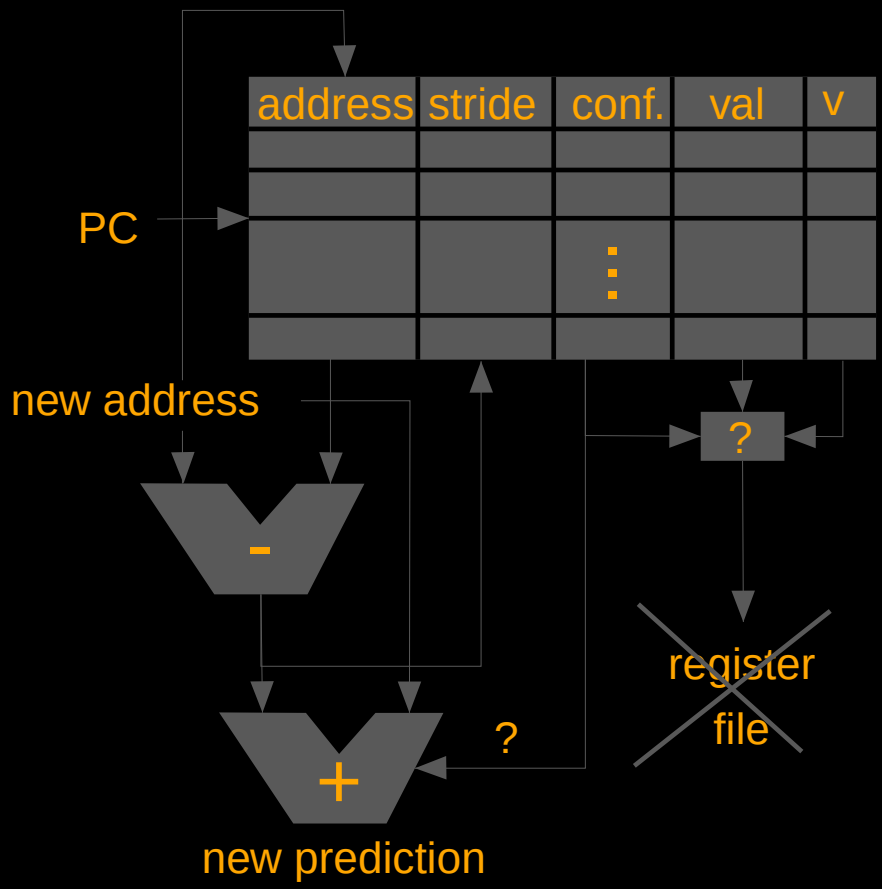
Prefetching Mechanism

- Decode phase
- Is value correct?
- Yes $\rightarrow$ bypass
- Address calc.
- Update table

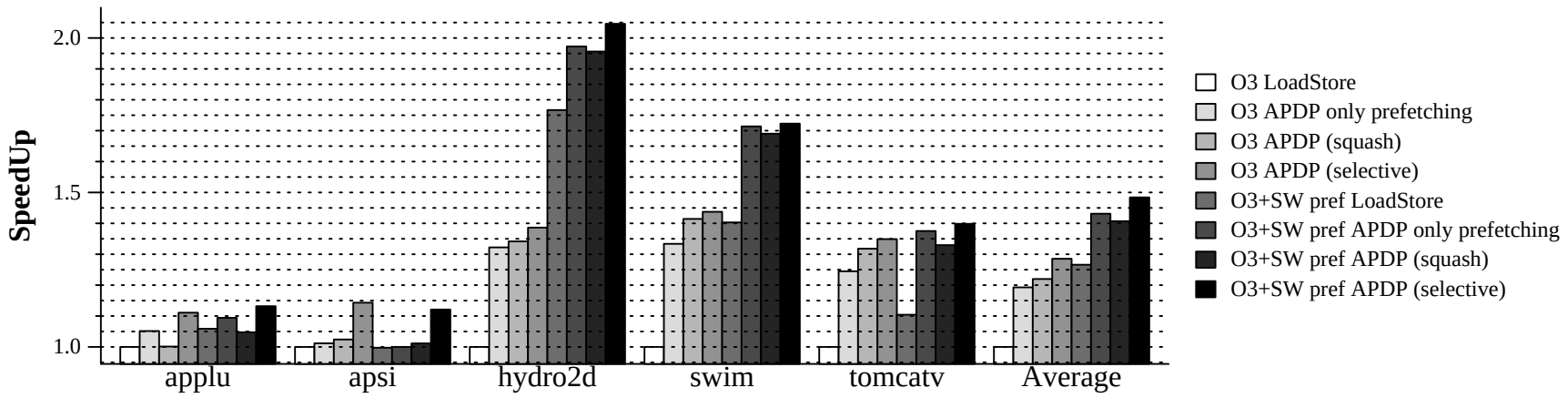


Prefetching Mechanism

- Decode phase
- Is value correct?
- Yes $\rightarrow$ bypass
- Address calc.
- Update table
- Generate prefetch?

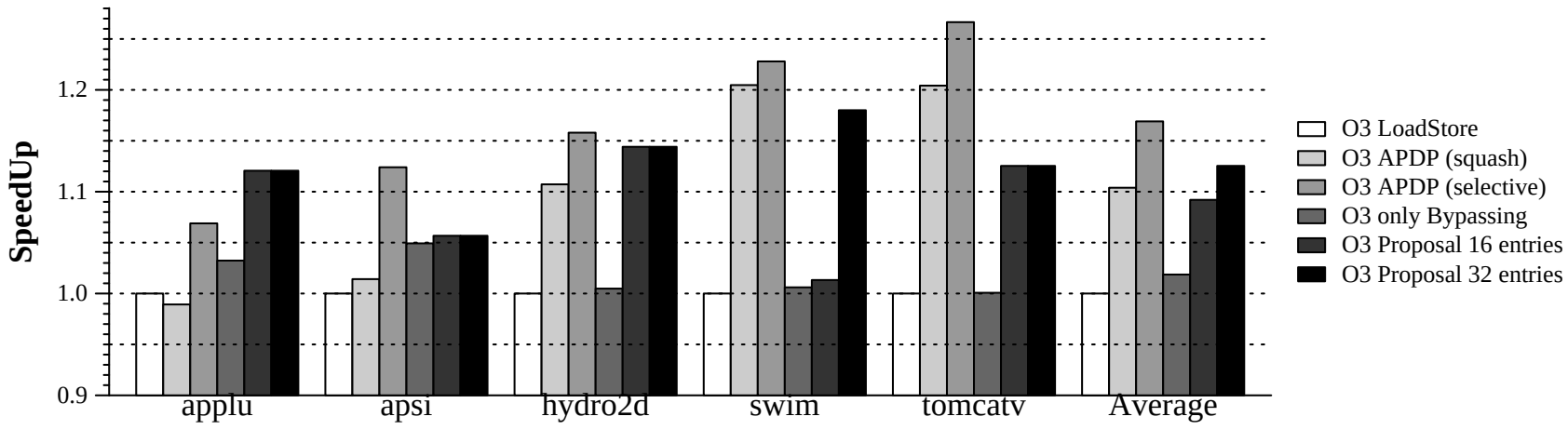


Comparison Results



Effect of only prefetching in APDP (4-way and 2 ports)

Comparison Results

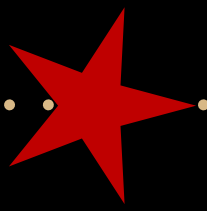


4 way machine comparison against APDP

Memory traffic

	applu	apsi	hydro2d	swim	tomcatv
LoadStore	1.00	1.00	1.00	1.00	1.00
APDP sel.	1.14	1.14	1.24	1.21	1.21
APDP sq.	1.14	1.15	1.24	1.21	1.22
Bypassing	0.90	0.74	0.82	0.72	0.89
Proposal	0.98	0.85	0.95	0.83	0.94

Any questions?



Thank you

dortega@ac.upc.es