Prediction and Speculation usage in microarchitecture

André Seznec

IRISA/INRIA

- Prediction and Speculation are everywhere

- Branch prediction

- Revisiting Value Prediction

PREDICTION AND SPECULATION ARE EVERYWHERE

Microarchitecture is about performance

- First implementation constraint:
  - CORRECTNESS

- Then : PERFORMANCE
  - In the respect of constraints:
    - Silicon area
    - Power/temperature constraints

Microarchitecture performance is about

- Technology
- Implementation:
  - Minimize the critical path
- Prediction and Speculation

Prediction vs Speculation

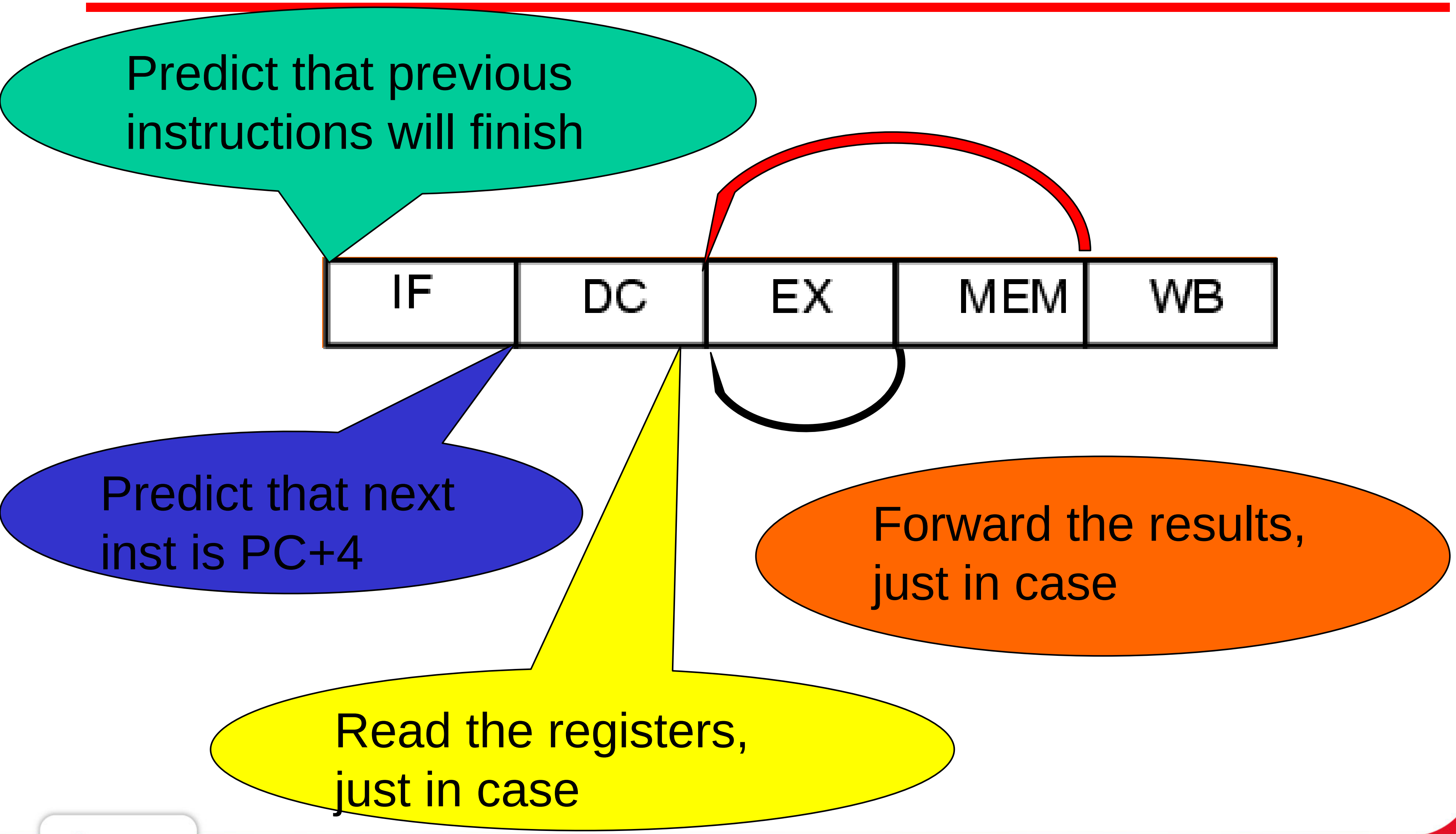
- Prediction:
  - Predict some events:
    - Predict the direction of a branch
    - Predict the address of a future load
- Speculation:
  - Computes based on a prediction
    - Fetch/decode/execute intructions

Necessitate repair if uncorrect

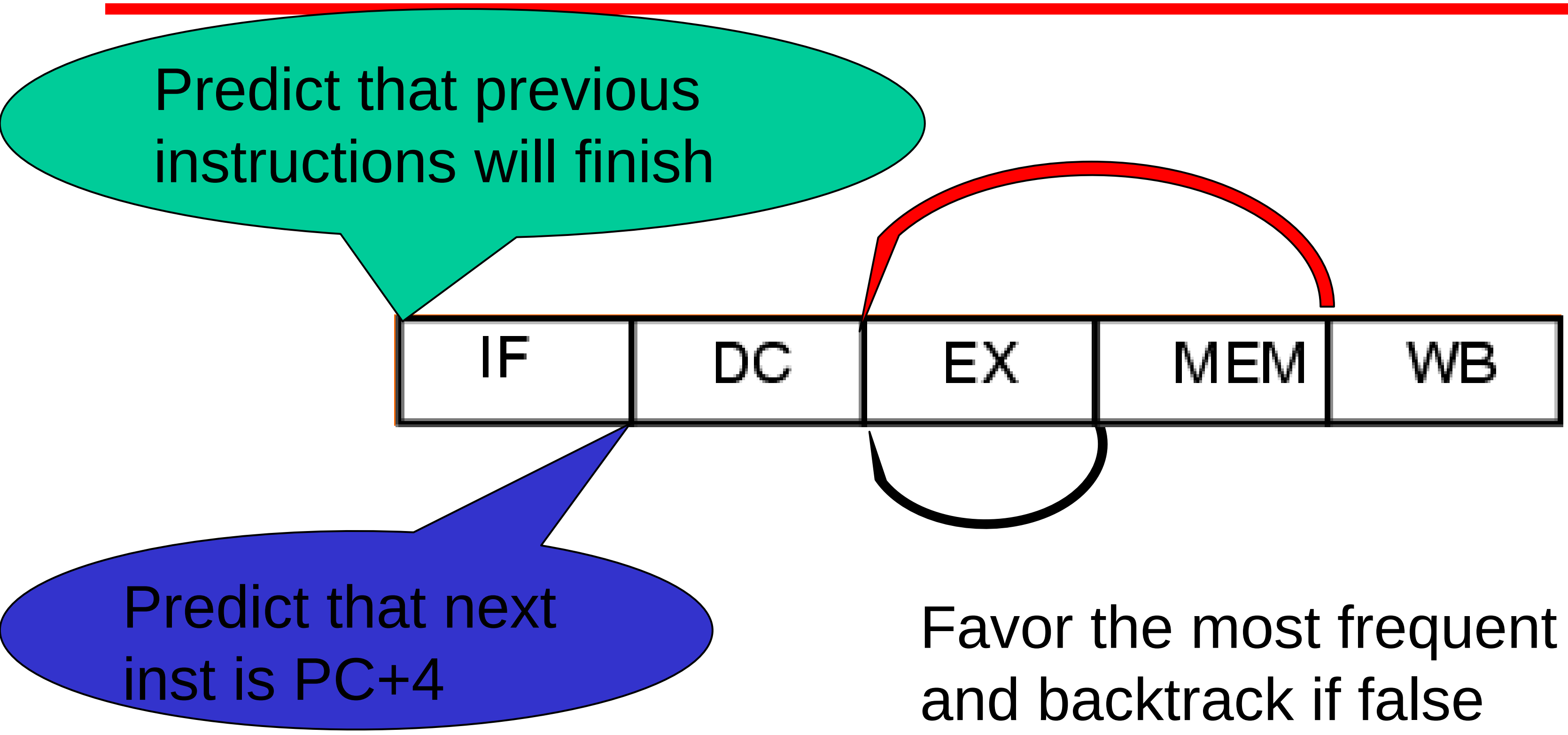
Prediction/speculation everywhere

- Pipeline
- Out-of-order execution
- Branch prediction
- Cache memories
- Dependence/independence prediction
- Data prediction
- Coherence transaction prediction
- Confidence prediction
- SMT instruction allocation policies

Pipelining is already speculating

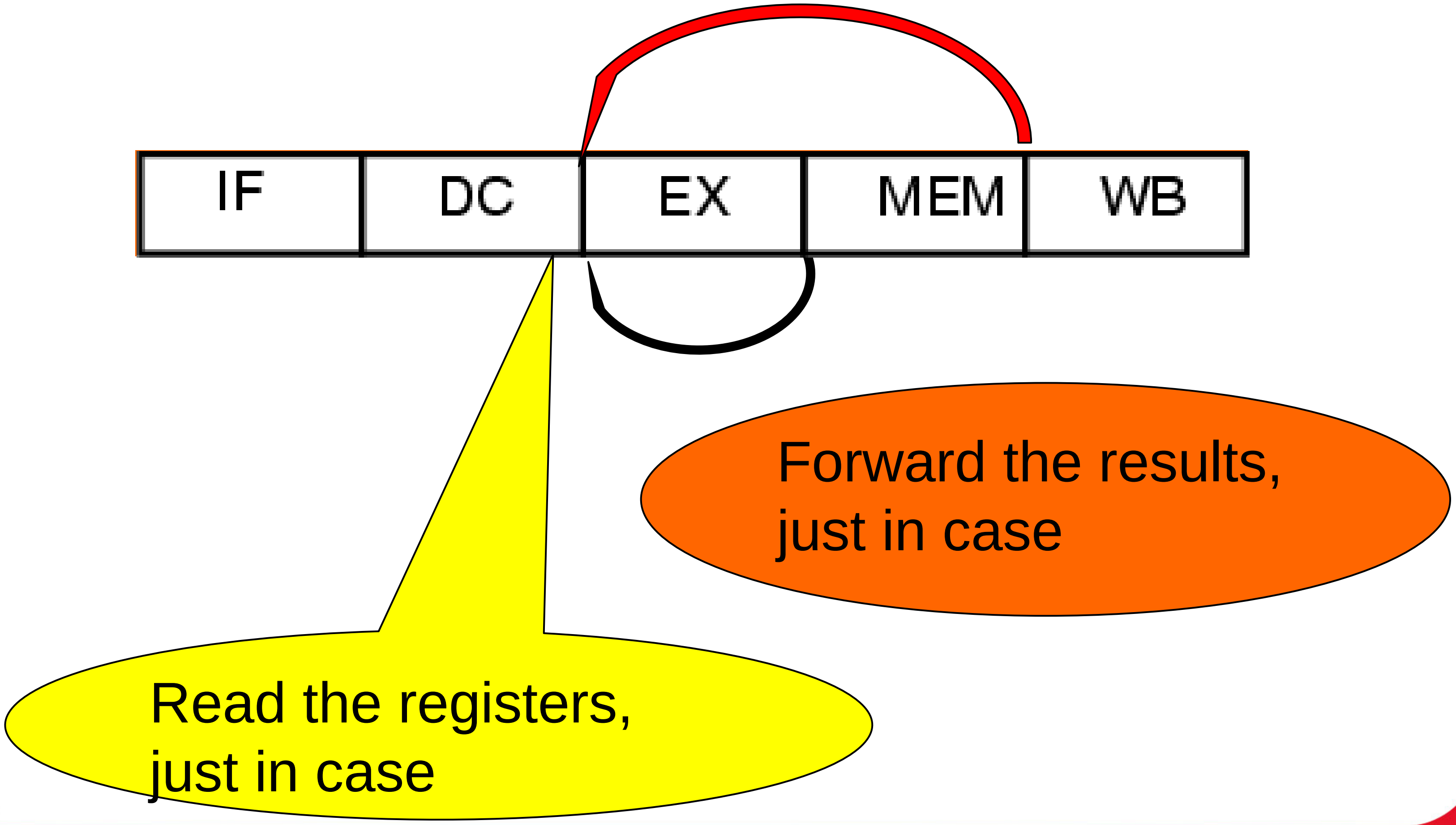

Predict that previous instructions will finish
IF
DC
EX
MEM
WB
Predict that next inst is PC+4
Forward the results, just in case
Read the registers, just in case

First principle


Predict that previous instructions will finish
IF
DC
EX
MEM
WB
Predict that next inst is PC+4

Favor the most frequent case, and backtrack if false

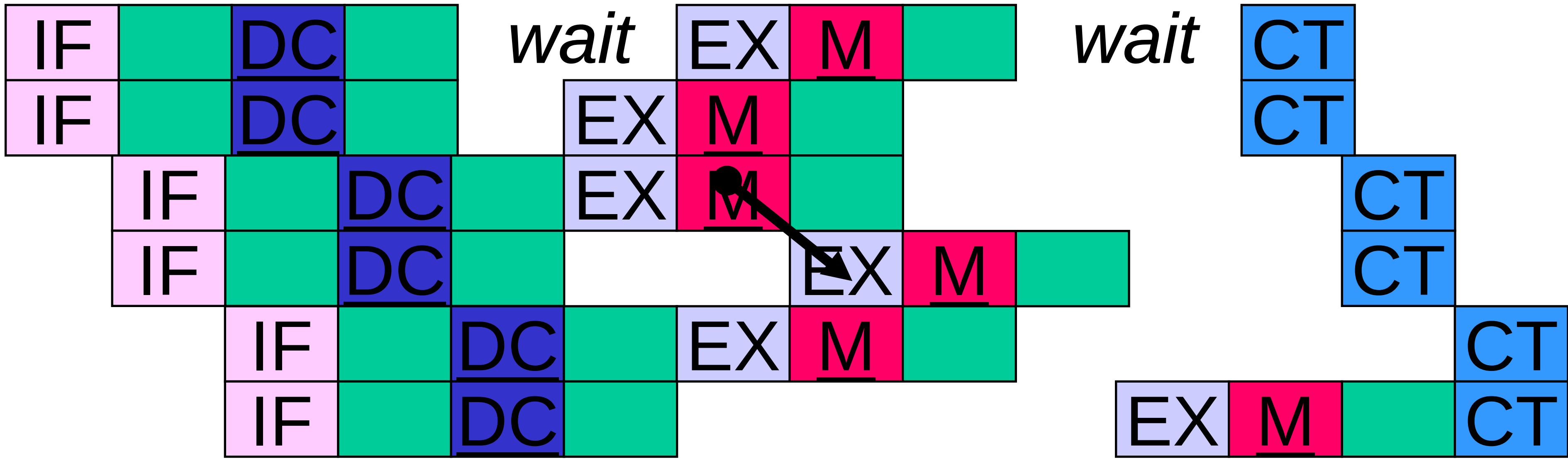
Anticipate



Forward the results, just in case

Read the registers, just in case

Out-of-order execution



An instruction may be fetched
100’s cycles before its validation

Out-of-order execution: speculation at multiple levels

- 100’s of *speculative* instructions in flight
  - Any instruction can abort due to:
    - Exception
    - Wrong instruction address
    - Wrong instruction
    - Wrong operand(s)

Exception

- An exception from time to time:
  - 1,000,000’s of cycles interval
    - Who cares ?
      - Speculate no exception
      - Just ensure correctness:
        - Flush the whole pipeline
  - Apart TLB misses !!
    - In some applications count by MPKI (miss per kilo instructions)

Why wrong instruction address

- With pipelining already an issue
  - PC of next instruction known late:
    - Decode for non-branch instructions
    - Execute for cond. branches or indirect jumps
- Predict the instruction flow and speculatively fetch/execute along the predicted instruction flow

Instruction flow prediction:

predict the address of the next instruction block

1. Within current block predict branches and branches types
2. Predict for all branches:
    1. Targets
    2. Directions
3. Select the correct next block address

On Alpha EV8 processor (cancelled 2001):

Two 8-instruction blocks per cycle

Up to 16 branches predicted per cycle

Why wrong instruction

- Instruction is read on the I-cache:
  - Needed ASAP for decode then verifying target, type etc.

➔ read without tag check:

- Line predictor on Alpha EV8
- Way predictor on Sun UltraSparc

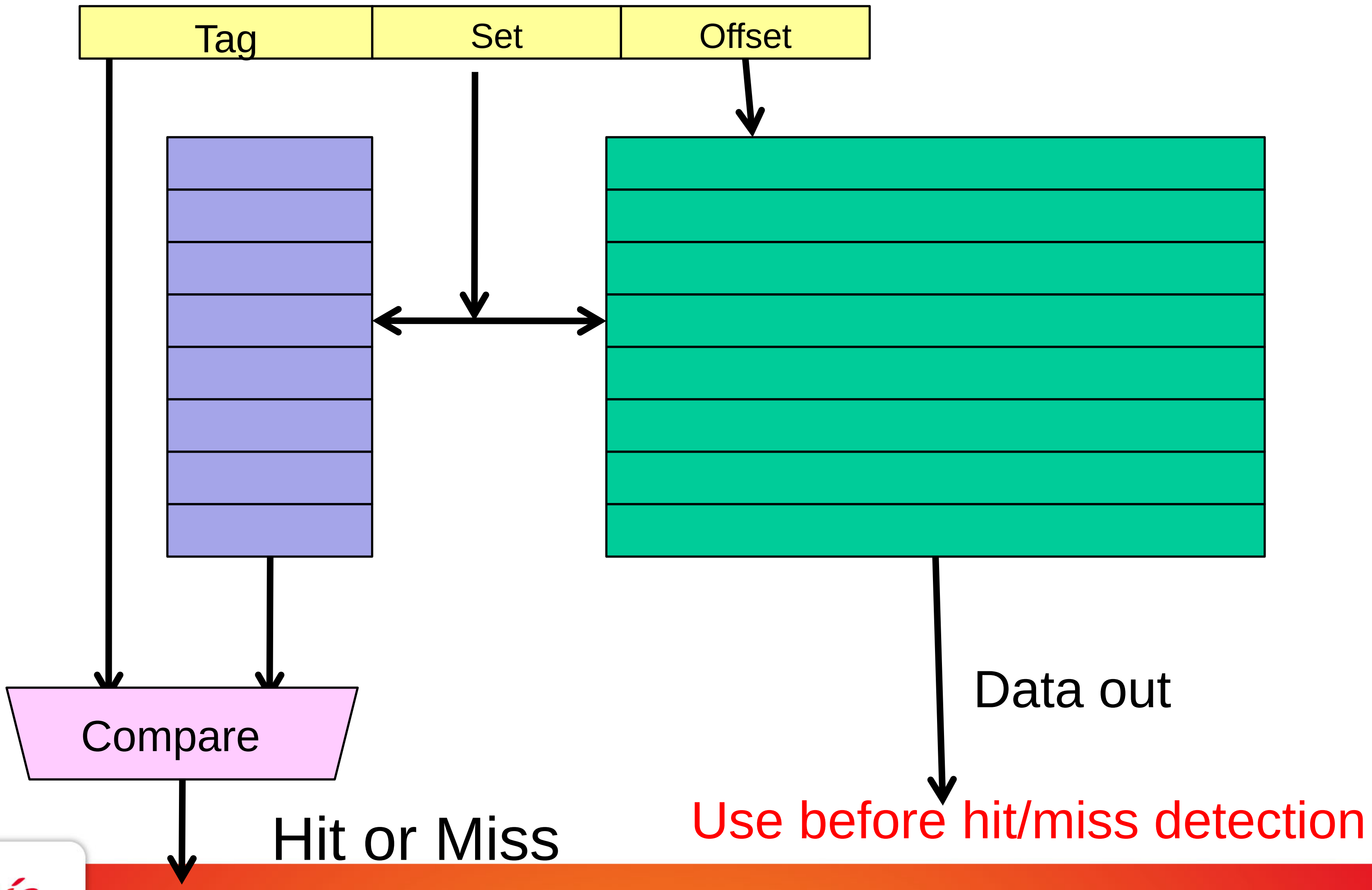
Wrong instruction repair

- Learned very early in the pipeline:
  - at tag check (way prediction)
  - end of instruction address generation (LP on EV8)
- Just flush two or three pipeline stages and resume fetching

Why wrong operands

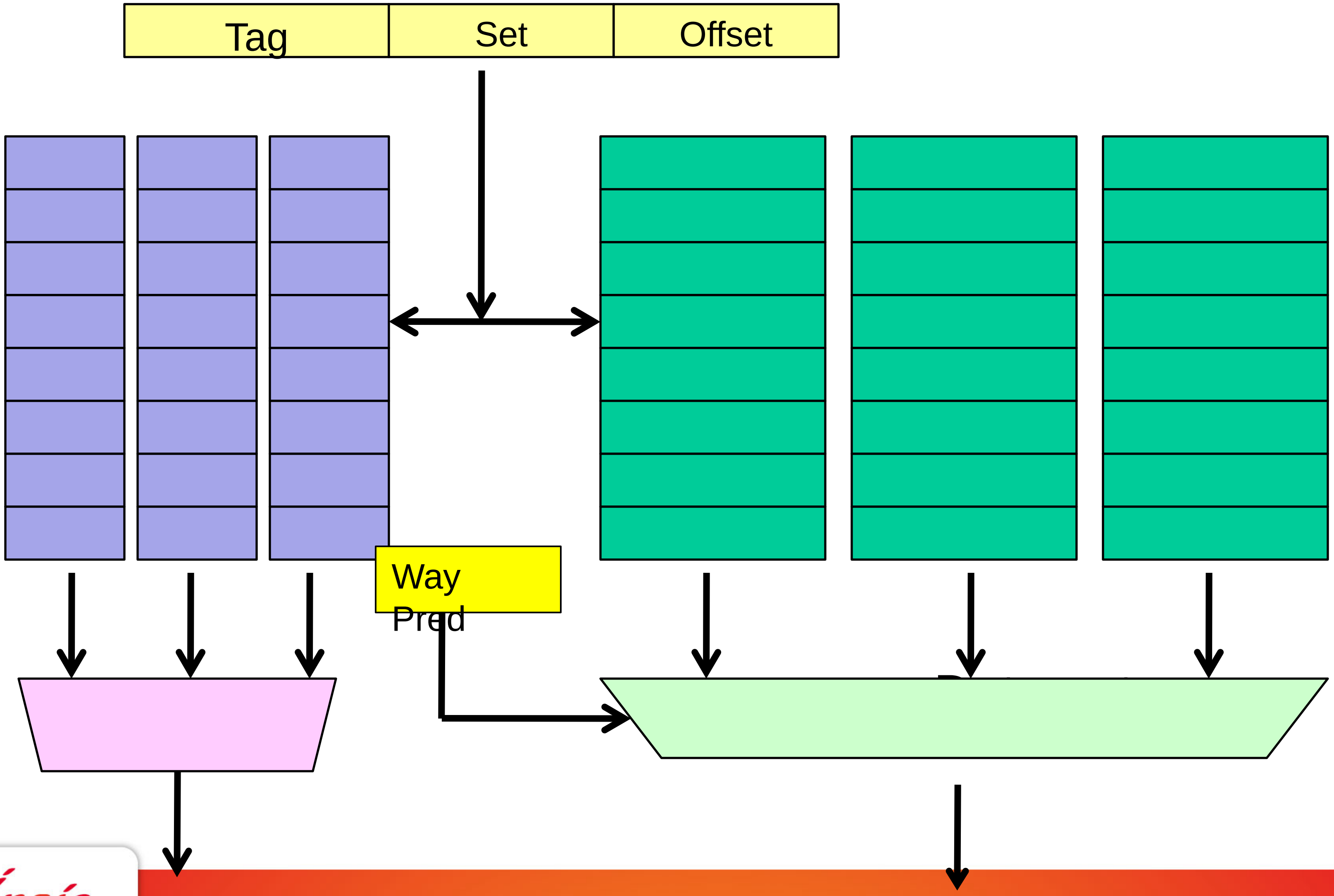
- Many papers on reducing critical path in data cache
  - Way prediction on data cache
  - Optimistic use on direct mapped cache

Optimistic cache result use

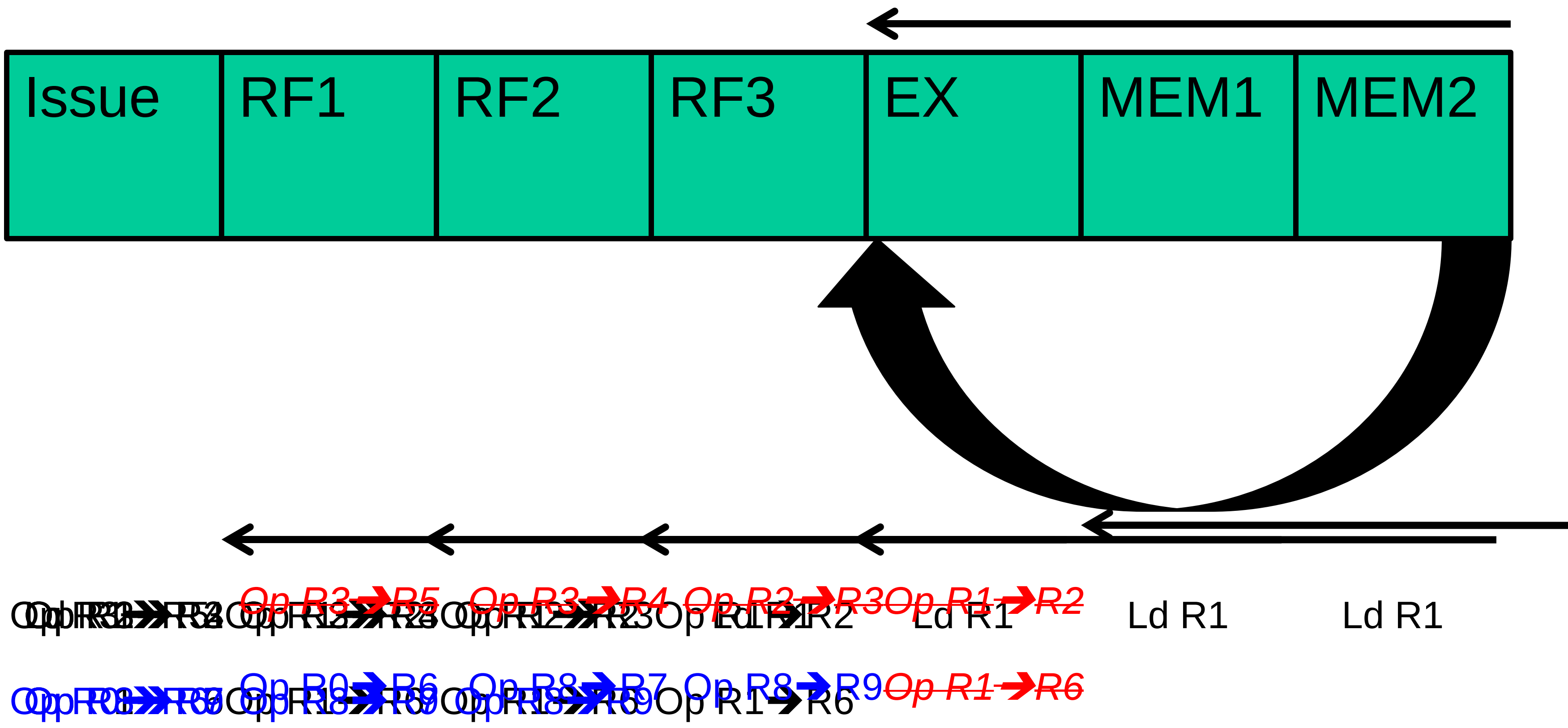


Use before hit/miss detection

Optimistic cache result use



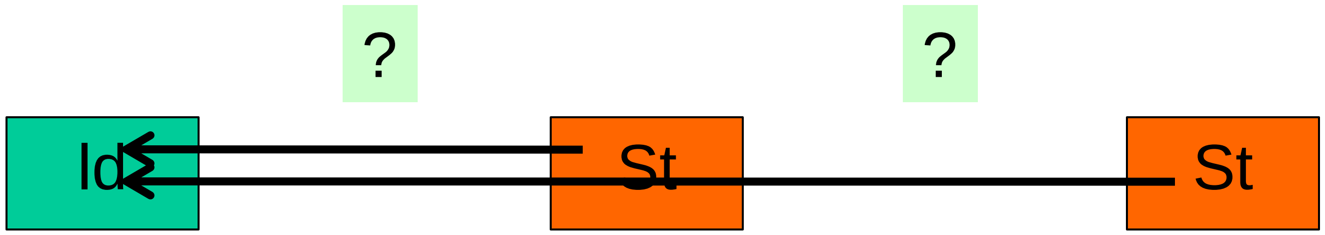
Why wrong operands: the implicit hit prediction



Oops: miss

Why wrong operand: False (in)dependencies

Loads potentially dependent on any preceeding store



On all processors, predict effective ld/st (in)dependencies and repair if uncorrect

Bypassing the stores

- At decode, allocate an entry in the load/store queue
- At execution:
  - For loads check all the older stores with computed addresses:
    - if match then grab the data
  - For stores, check all the younger already executed loads
    - if match then repair

Bypassing the stores

- Systematic bypass is not performant:
  - Cost of repair

- Dependence prediction:
  - i.e. bypassing when independence is predicted

Predicting dependencies

- This load is dependent on a previous store

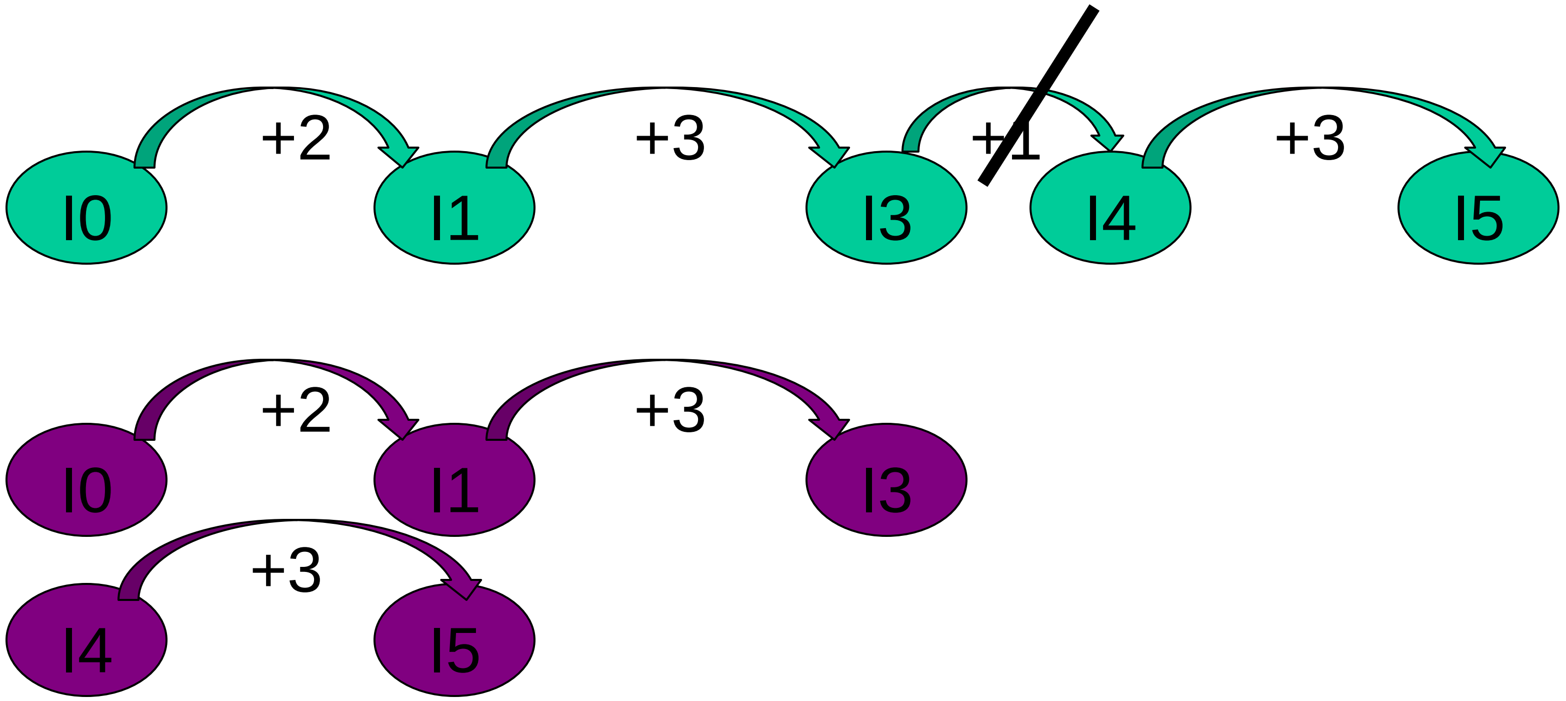
- This load is dependent on THIS previous store

Wrong data value prediction ?

Lipasti et al, Gabbay and Mendelson 1996

Basic idea:

- Eliminate (some) true data dependencies through predicting instruction results



Value Prediction:

- Large body of research 96-02

- Quite efficient:
  - Surprisingly high number of predictable instructions

- <u>**Not implemented so far:**</u>
  - High cost : *is it still relevant now ?*
  - High penalty on misp.: **don’t lose all the benefit**

Wrong operand: the selective repair issue

- Realize that an instruction has used a wrong operand
  - All dependent instructions have to be cancelled and reissued:
    - May be 10’s of instructions
    - Implemented but not very documented

Other prediction usages

- Cache prefetching
- Coherence transaction prediction
- Confidence estimation
- SMT steering policies

Cache prefetching is no speculation

- Bring a memory block (likely to be accessed) close to the processor core:
  - L1 cache ? L2 cache ? Prefetch buffer
- Does not modify the memory block :
  - no action needed for repairing

Cache prefetching issues

- Which block to prefetch ?

- Aggressiveness:
  - How much to prefetch in advance ?
    - In-time or to late ?
  - Bandwidth wasting and/or demand miss delaying:
    - How to control ?

Cache prefetching

- Predicting the addresses of next cache blocks to be touched
  - Next block prefetching
  - Stride prefetching
  - Stream prefetching
  - Markov prefetching
- 100's of propositions
- Implemented in many processors

Next block prefetching

- On a miss, prefetch next block:
  - (potentially) save one of two misses
- On a hit on a prefetched block, prefetch
  - Create streams of prefetchs
- Generally too late on modern processors
  - Latency of several hundred cycles

Stream prefetching

- Detect series of consecutive blocks
- Prefetch the successors
  - Resolve the distance issue

Stride prefectching

- Accesses by the same load/store instruction:
  - A, A +S, A+ 2S, ➔ A+ 3S, A+4S (probably)

- Issues:
  - Seen only at L1 levels
  - Only on virtual address space

Markov prefetching

- Series of coorelated misses:
  - Miss on B follows miss on A which follows miss on Z etc:
    - See miss on Z: prefetch A, prefetch B

- Issue: very memory demanding for storing patterns

General issue on prefetching

- Multi- manycore era:
  - Memory hierarchy shared by many cores:
    - How to hande the sharing of bandwidth, memory hierarchy and so on

Coherence transaction prediction

On coherent cache multiprocessors:

- Try to learn pattern of the data usage:
  - Try to push the data towards its consumer
    - And avoid coherence traffic

A very particular prediction

- Confidence estimation:
  - Predict whether the prediction is “likely” to be correct
    - For evaluating the potential tradeoff benefit/cost
      - Prefetch (benefit against wasting bandwidth)
      - Branch prediction (saving power)

Confidence estimation

- Quite particular:
  - Each predictor needs its own adapted confidence estimation
  - Each usage requires its own confidence estimation level

SMT instruction steering policies

- Share all the resources among several threads:
  - Functional units, caches, ..
- Fetch for thread T0 or thread T1
- Different strategies to “predict” the benefits:
  - Number of pending instructions
  - Number of low confidence branches

Many other predictions

- E.g. usage of hardware structures:
  - To save power:
    - Dynamic instruction window sizes

BRANCH PREDICTION

Why branch prediction ?

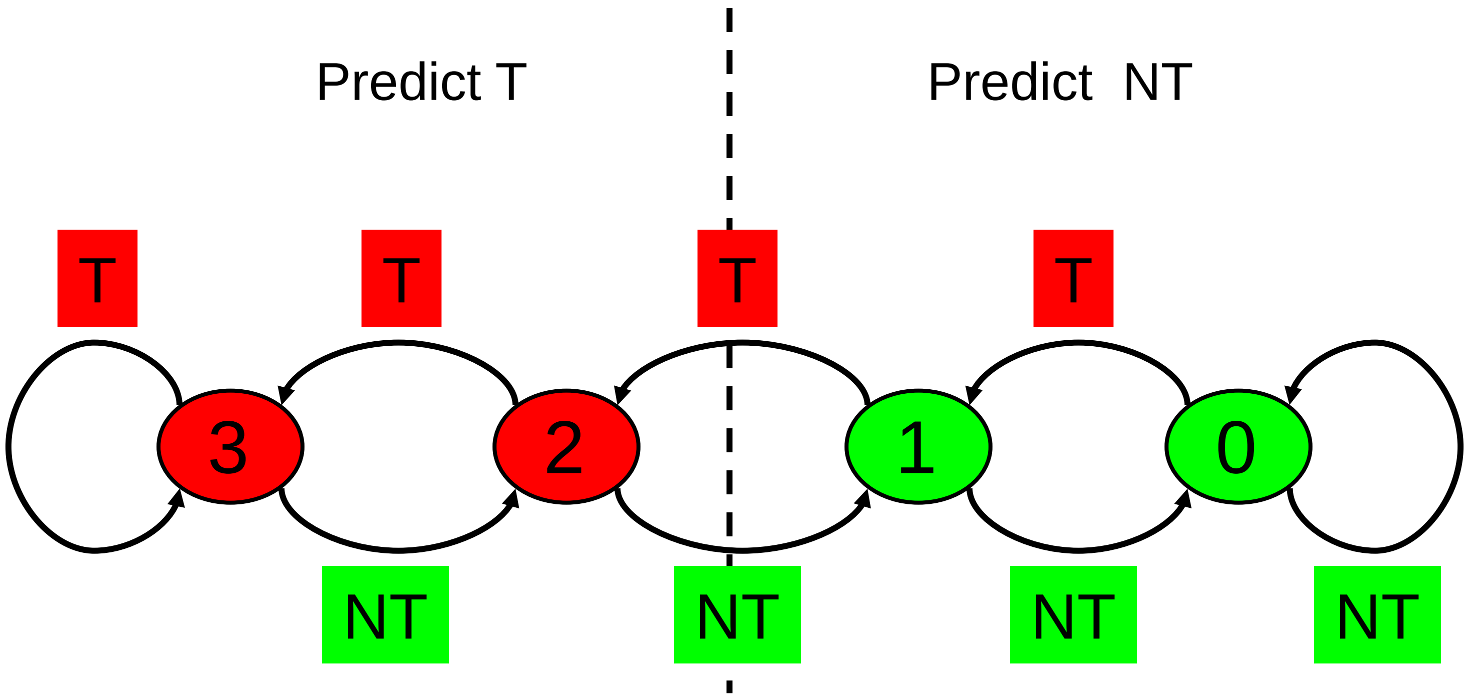
- 10-30 % instructions are branches
- 4 instructions per cycle
- Direction and target known around cycle 20
  - Not possible to lose 20 cycles on each branch
  - PREDICT BRANCHES
    - and verify later !!

Prediction through the PC
Smith 1981

- Most branches biased towards taken or not taken

- Use a table to predict the branches:
  - Record the output of the branch
  - Use the last behavior
    - + add some memory through a 2 bit counter

The 2-bit counter automaton



global branch history

Yeh and Patt 91, Pan, So, Rameh 92

```
B1: if cond1

B2: if cond2

B3: if cond1 and cond2
```

B1 and B2 outputs determine B3 output

Exploiting local history
Yeh and Patt 91

Look at the 3 last occurrences:

If all loop backs then loop exit

otherwise: loop back

```
for (i=0; i<100; i++)
    for (j=0;j<4;j++)
        loop body
```

- **A local history <u>per</u> branch**
- **Table of counters indexed with PC + local history**

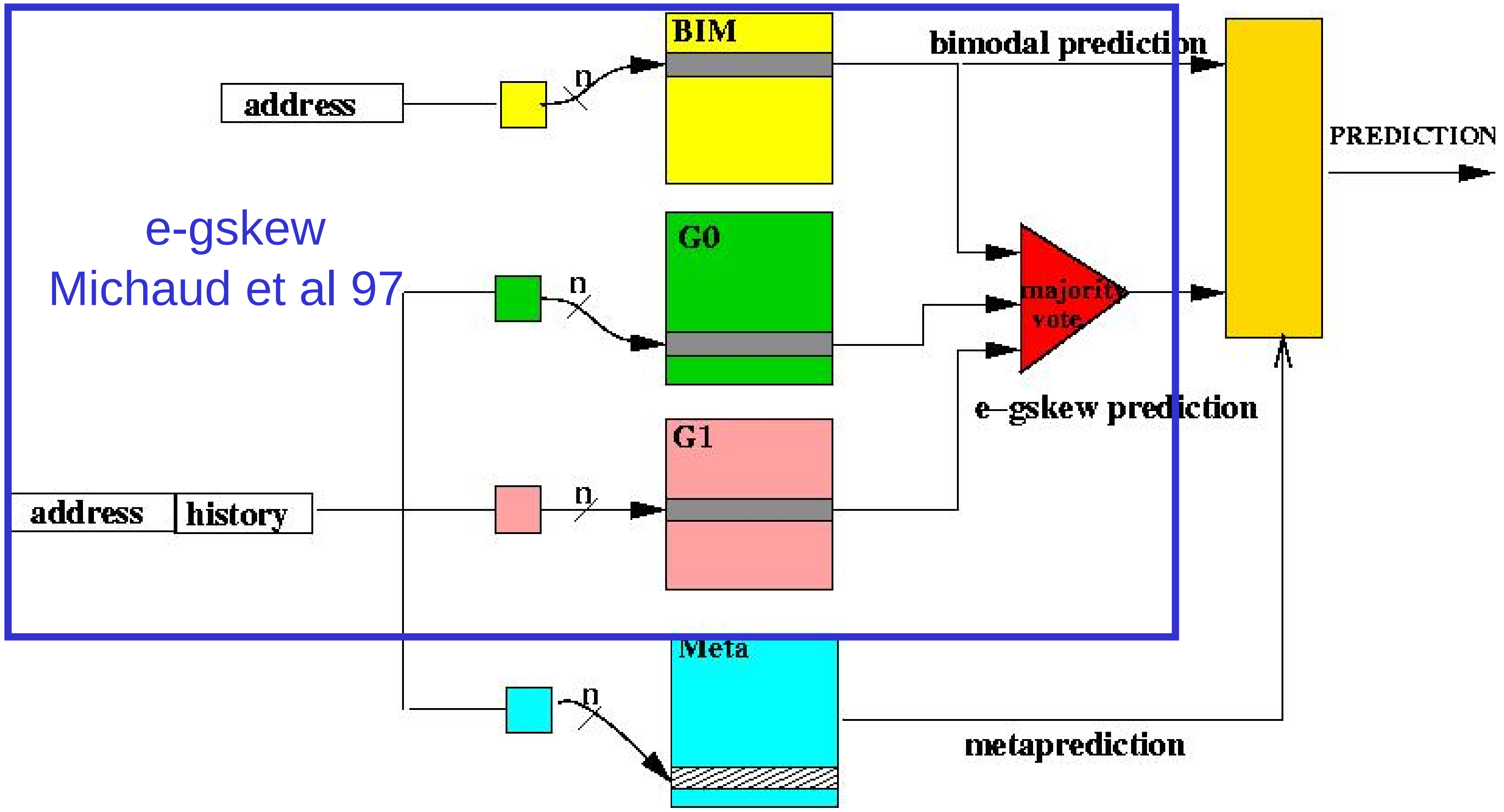
Speculative history must be managed !?

- Local history:
  - table of histories (unspeculatively updated)
  - must maintain a speculative history per inflight branch:
    - Associative search, etc ?!?
- Global history:
  - Append a bit on a single history register
  - Use of a circular buffer and just a pointer to speculatively manage the history

Branch prediction: Hot research topic in the late 90's

- McFarling 1993:
  - Gshare (hashing PC and history) +Hybrid predictors
- « Dealiased » predictors: reducing table conflicts impact
  - Bimode, e-gskew, Agree 1997

EV8 predictor: (*derived from)* 2bc-gskew Seznec et al, ISCA 2002 (1999)



Retrospectively

For 64Kbits predictors on CBP-1 traces (2004)
assuming 2 inst/cycle, 20-cycle misprediction penalty

- 2bit counters 1981: 8.55 misp/KI 671 cycles/KI

No real work before 1991:
win 37 % misp, 10 % perf

- Gshare 1993: 5.30 misp/KI 606 cycles/KI

Hot topic, heroic efforts:
win 28 % misp, 5 % perf

- EV8-like 2002 (1999): 3.80 misp/KI 576 cycles/KI

Still worth to enhance branch prediction ?

Replacing the branch predictor by a more accurate one

- Improves directly the performance

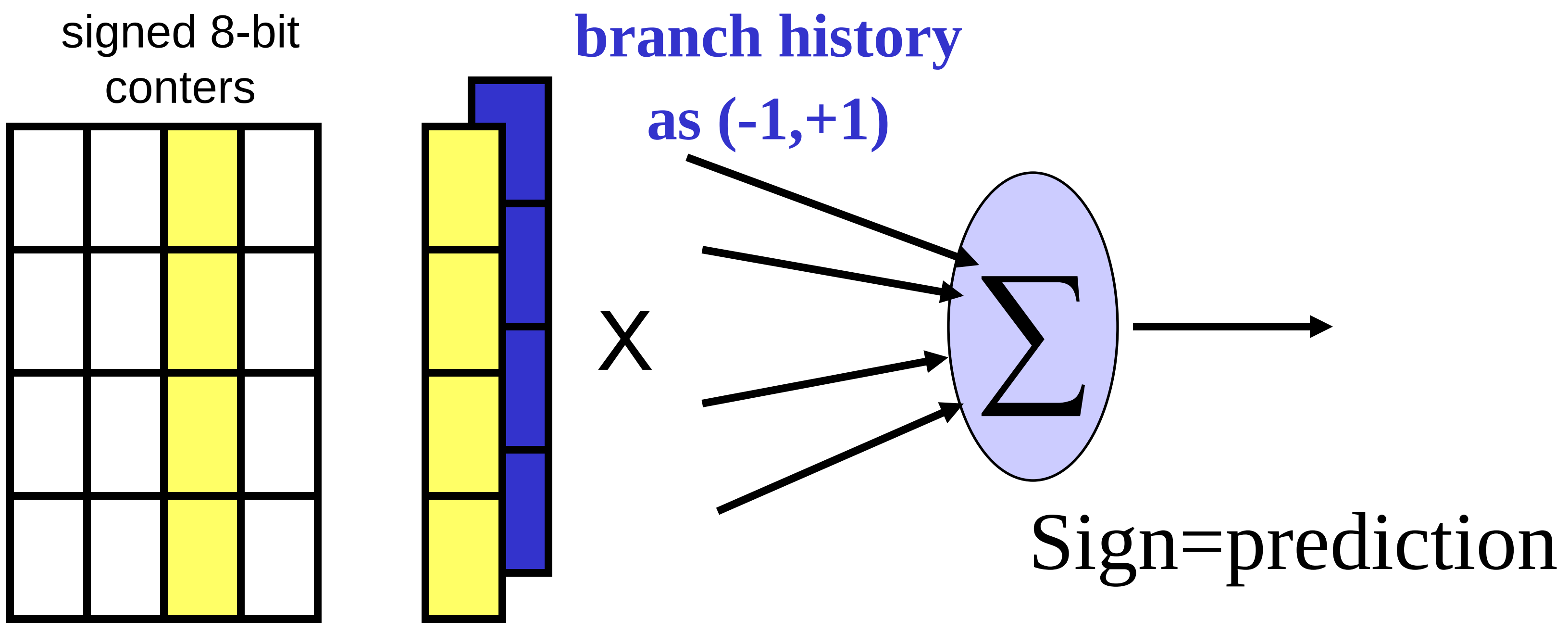
- Does not affect the rest of the design
  - Can be considered for a new release of the processor

After 2000, interest from computer architecture community faded

THE MULTICORE ERA

But ..

Perceptron predictor Jimenez and Lin 2001



Update on mispredictions or if |SUM| < θ

Perceptron predictor

- Not that accurate
- High hardware complexity

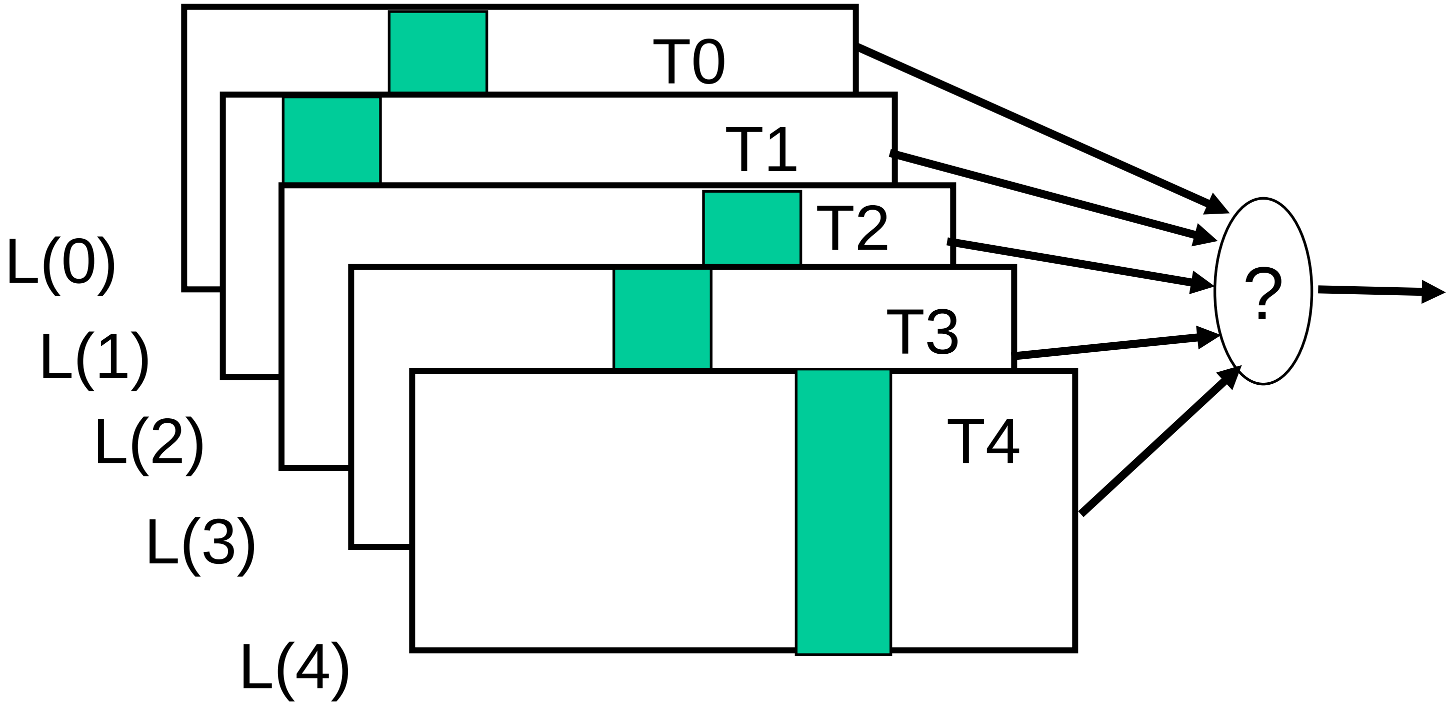
but

- Sometimes better than classical predictors
- Intellectually challenging

Back around 2003

- 2bcgskew (EV8) state-of-the-art, but:
  - but was lagging behind neural inspired predictors on a few benchmarks
- Just wanted to get best of both behaviors, plus
  - Reasonable implementation cost:
    - Use only global history
    - Medium number of tables
  - In-time response: a taken branch per cycle

The basis : A Multiple length global history predictor



With a limited number of tables

Underlying idea

- H and H' two history vectors equal on N bits, but differ on bit N+1
  - e.g. L(1)≤N<L(2)
- Branches (A,H) and (A,H')

  biased in opposite directions

Table T2 should allow to discriminate between (A,H) and (A,H')

From my “old experience” 2bcgskew and EV8

- Some applications benefit from 100+ bits histories
  - Generally only a few branches
- Other don’t !!
  - And it is a loss of storage
  - Should not “waste” too much space for long histories

GEometric History Length predictor

The set of history lengths forms a geometric series

$$L(0) = 0$$

$$L(i) = \alpha^{i-1} L(1)$$

{0, 2, 4, 8, 16, 32, 64, 128}

What is important: L(i)-L(i-1) is drastically increasing

Spends most of the storage for short history !!

Selecting between multiple predictions

- Use of a meta predictor
  - ~~“wasting” storage !?!~~
  - ~~chosing among 5 or 10 predictions ??~~

Poor storage efficiency

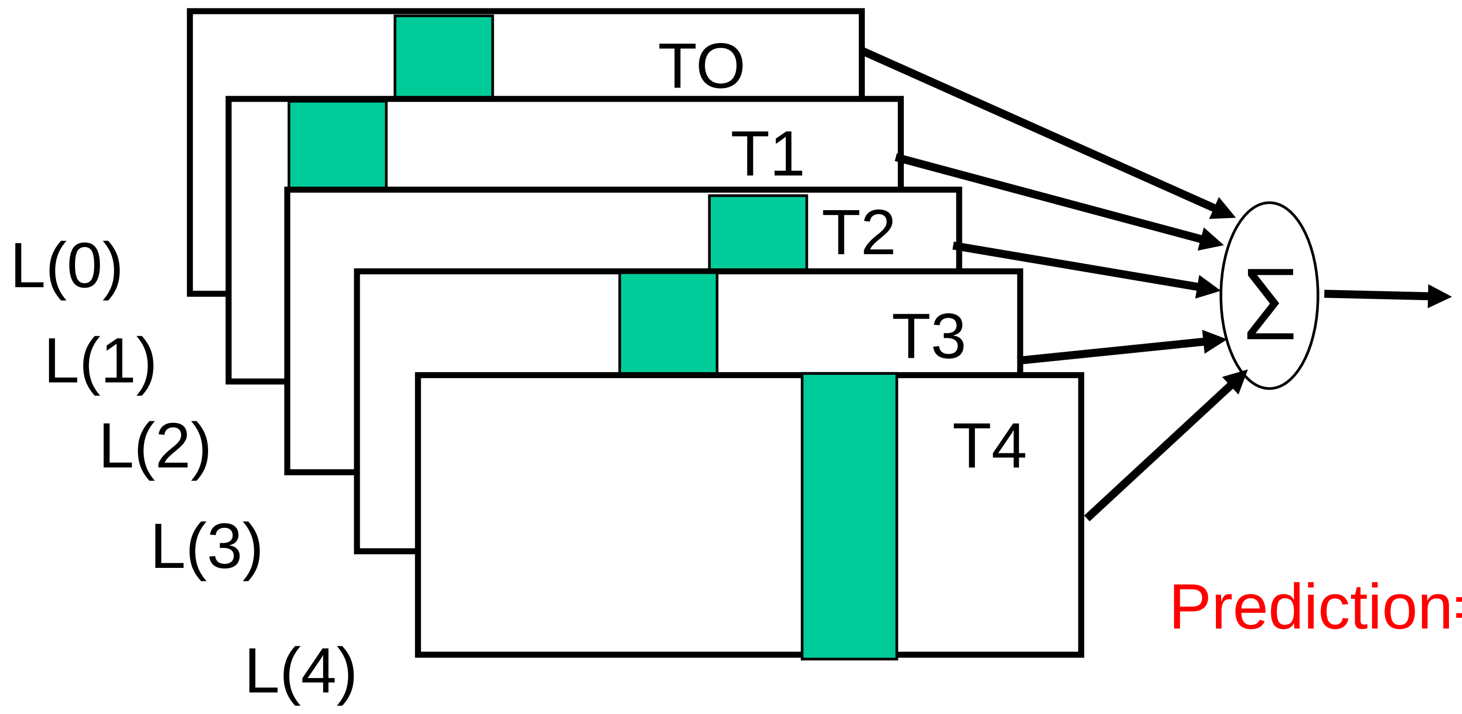
- Neural inspired predictors:
  - Use an adder tree instead of a meta-predictor

  Jimenez and Lin 2001

- Partial matching:
  - Use tagged tables and the longest matching history

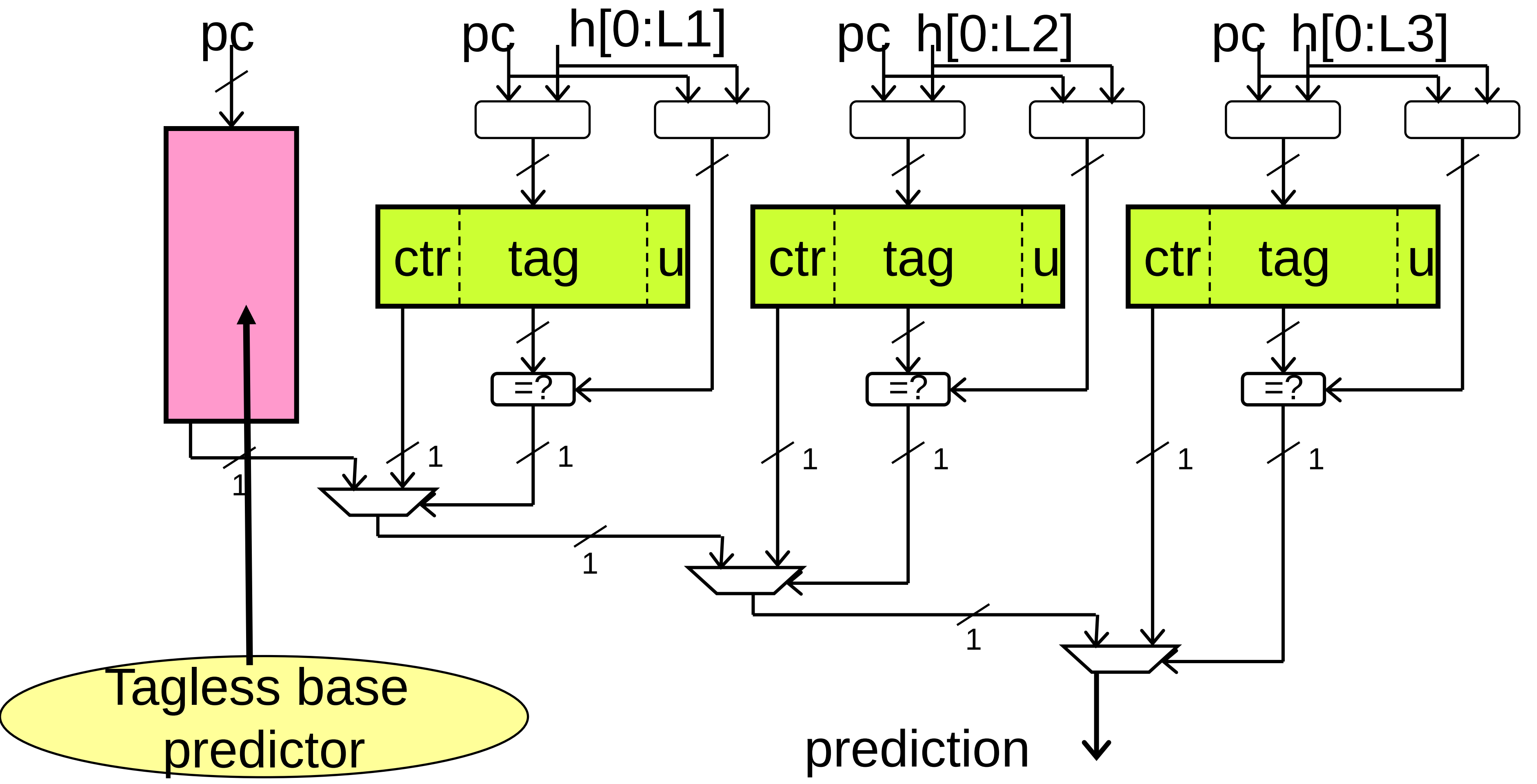
  Chen et al 96, Eden and Mudge 1998, Michaud 2005

GEHL (2004) prediction through an adder tree



Prediction=Sign

TAGE (2006) prediction through partial match



The Geometric History Length Predictors

- Tree adder:
  - O-GEHL: Optimized GEometric History Length predictor
    - CBP-1, 2004, best practice award
- Partial match:
  - TAGE: TAgged GEometric history length predictor
    - Inspired from PPM-like, Michaud 2004

      + geometric length

      + optimized update policy
    - Base of the CBP-2, 2006 winner
    - Base of the CBP-3, 2011 winner

GEHL

- Geometric history length: 4 to 12 tables
- Perceptron inspired threshold based update policy:
  - Perceptron-like threshold does not work ☹
  - Experimentally:
    - The number of tables is a reasonable threshold
  - *Dynamic threshold fitting*
- 4-bit or <u>5-bit counters </u>(against 8-bit on perceptron)

Dynamic update threshold fitting

On an O-GEHL predictor, best threshold depends on

- the application ☹
- the predictor size ☹
- the counter width ☹

By chance,

on most applications, for the best fixed threshold,

updates on mispredictions ≈ updates on correct predictions

<u>**Monitor the difference**</u>

<u>**and adapt the update threshold**</u>

2004 Championship Branch Prediction O-GEHL

- O-GEHL = GEHL + a trick

  Dynamic history length fitting

  - *Brings (marginal) extra accuracy, but complex logic cost for medium storage budgets*

NOT TO BE IMPLEMENTED

Evaluation framework

1st Championship Branch Prediction traces:

- 20 traces including system activity
- Floating point apps : loop dominated
- Integer apps: usual SPECINT
- Multimedia apps
- Server workload apps: very large footprint

64 Kbits configuration
2004 Championship Branch Prediction

- 8 tables:
  - Medium number of tables
  - 5 bit counters for T0 and T1, 4 bit counters otherwise

- L(1) =3 and L(10)= 200
  - {0,3,5,8,12,19,31,49,75,125,200}

The O-GEHL predictor

- 2nd at CBP: 2.82 misp/KI
- Best practice award:
  - *The predictor the closest to a possible hardware implementation*
  - Does not use exotic features:
    - Various prime numbers, etc
    - Strange initial state
- Very short warming intervals:
  - Chaining all simulations: 2.84 misp/KI

OGEHL predictor (in 2004)

- State-of-the-art before CBP
  - 1Mbit 2bcgskew (9,9,36,72): 3.19 misp/KI
  - 1888 Kbits PBNP (58): 3.23 misp/KI

- OGEHL
  - 32Kbits (3,150): 3.41 misp/KI
  - 1Mbits (5,300): 2.27 misp/KI

Robustness of the OGEHL predictor

- Robustness to variations of history lengths choices:
  - L(1) in [2,6], L(10) in [125,300]
  - misp. rate < 2.96 misp/KI
- Geometric series: not a bad formula !!
  - best geometric L(1)=3, L(10)=223, 2.80 misp/KI
  - best overall {0, 2, 4, 9, 12, 18, 31, 54, 114, 145, 266} 2.78 misp/KI

OGEHL scalability

- 4 components — 8 components
  - 64 Kbits: 3.02 -- 2.84 misp/KI
  - 256Kbits: 2.59 -- 2.44 misp/KI
  - 1Mbit: 2.40 -- 2.27 misp/KI
- 6 components — 12 components
  - 48 Kbits: 3.02 – 3.03 misp/KI
  - 768Kbits: 2.35 – 2.25 misp/KI

4 to 12 components bring high accuracy ☺

At CBP-1, all finalists were using tree adders apart the PPM-like predictor: 3.24 misp/KI

but ..

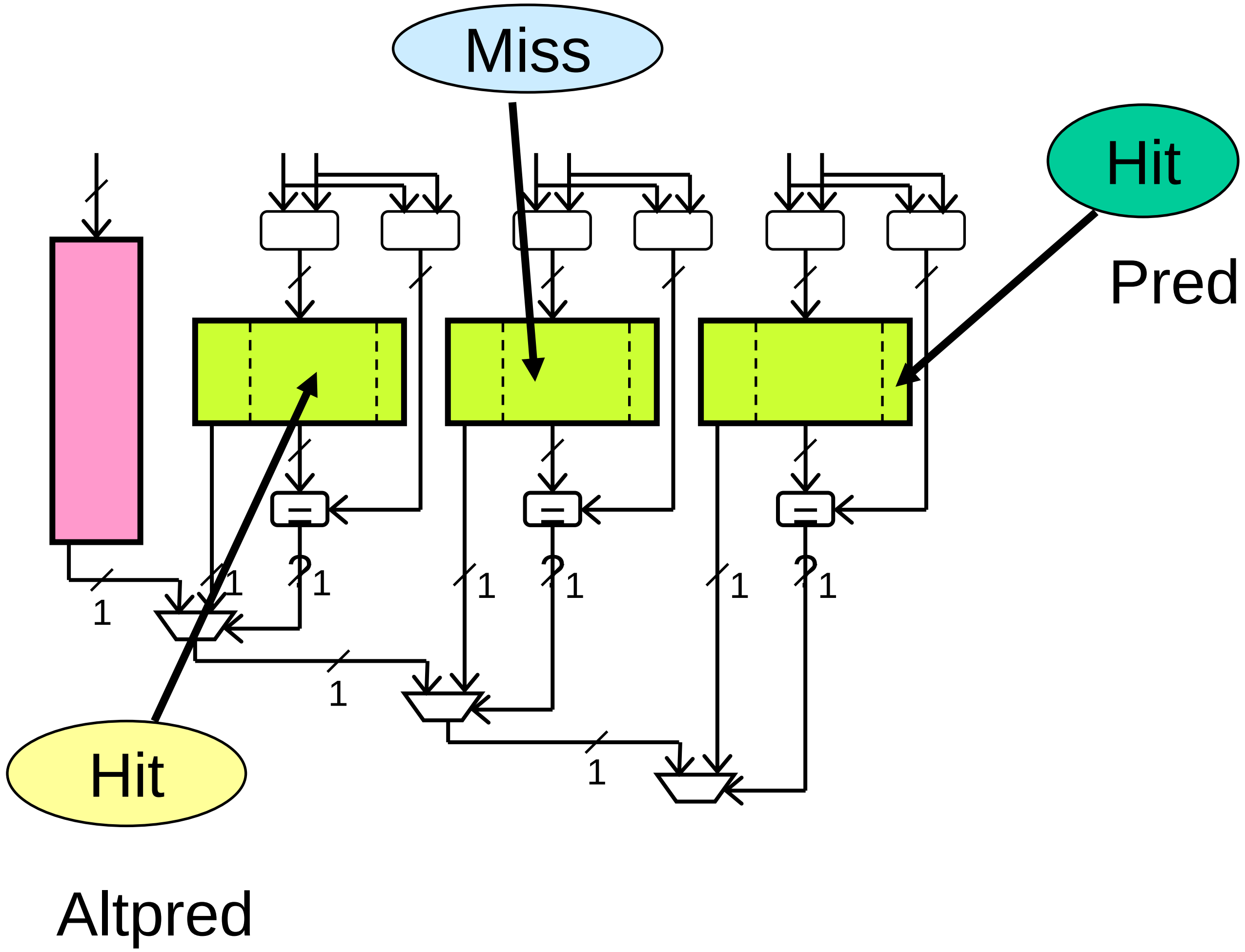
The update policy was poor

TAGE

- Partial tag match
  - almost ..

- Geometric history length

- Very effective update policy


Miss
Hit
Pred
Hit
Altpred
1
1
=?
1
1
1
1
1
1
1

Prediction computation

- General case:
  - Longest matching component provides the prediction

- Special case: newly allocated entries
  - Very high misprediction rate on :
    - *weak Ctr :* 42 % mispredictions

*In many cases, **Altpred** more accurate than **Pred***

- *Property dynamically monitored through a single 4-bit counter:*
  - *weak Ctr: 34 % mispredictions*

TAGE update policy

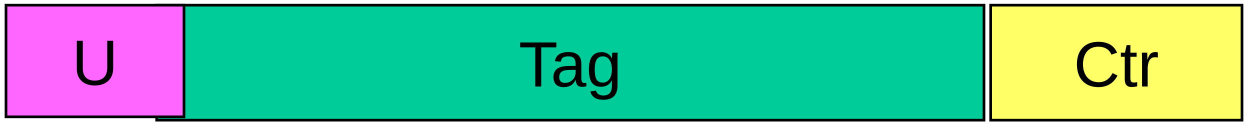
- General principle:

  <u>Minimize the footprint of the prediction</u>.

  - Just update the longest history matching component and allocate at most one entry on a misprediction

A tagged table entry

- Ctr: 3-bit prediction counter
- U: 2-bit useful counter
  - Was the entry recently useful ?
- Tag: partial tag



Update policy

- Update the matching component
  - or the base predictor

- Allocate at most one new entry on a misprediction:
  - A single entry
  - In place of an otherwise useless entry

Usefulness of an entry

If (Altpred ≠ Pred) then

- Pred = taken : U= U + 1 becomes *useful*

Graceful aging:

Periodic reset of 1 bit in all counters

Useful = <u>avoided a misprediction quite recently</u>

Allocating a new entry on a misprediction

- Find a single *useless* entry (U=0) with a longer history:
  - Priviledge the smallest possible history
    - To minimize footprint
  - But not too much
    - To avoid ping-pong phenomena

- Initialize Ctr as weak and U as zero:
  - *Can be replaced till it becomes useful*

    (not so many entries become useful)

TAGE vs OGEHL

8 comp. OGEHL	8 comp. TAGE	5 comp. TAGE
64K: 2.84 misp/KI	64K: 2.58 misp/KI	64K: 2.70 misp/KI
128K: 2.54 misp/KI	128K: 2.38 misp/KI	128K: 2.45 misp/KI
256K: 2.43 misp/KI	256K: 2.23 misp/KI	256K: 2.28 misp/KI
512K: 2.33 misp/KI	512K: 2.12 misp/KI	512K: 2.19 misp/KI
1M : 2.27 misp/KI	1M: 2.05 misp/KI	1M: 2.12 misp/KI

Same trend on other benchmark traces

Partial tag matching is more effective than adder tree

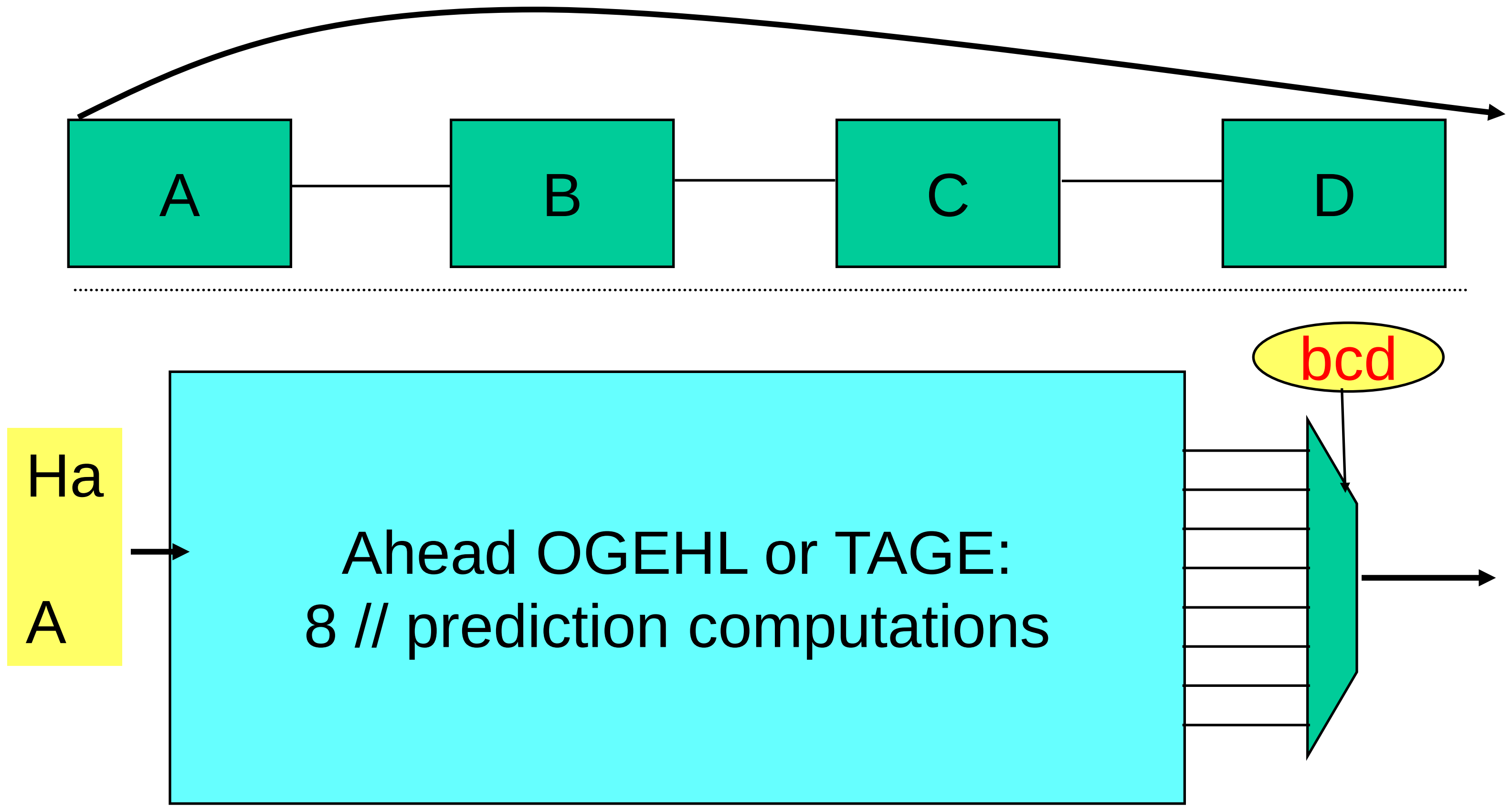
- At equal table numbers and equal history lengths for limited storage budgets (<= 1 Mbit)
  - TAGE better than GEHL on each of the 20 benchmarks

- Accuracy limit (with unlimited storage budget) is higher on GEHL than TAGE
  - Shown in the limit study for CBP-2

Prediction computation time ?

- 3 successive steps:
  - Index computation:
    - a few entry XOR gate
  - Table read
  - Adder tree or (tag check + mux traversal)
- Will not fit on a single cycle:
  - Overriding predictor
  - or can be ahead pipelined !

Ahead pipelining a global history branch predictor



Ahead Pipelined 64 Kbits OGEHL or TAGE

- 3-block ahead 64Kbits TAGE:
  - 2.70 misp/KI vs 2.58 misp/KI
- 3-block ahead 64Kbits OGEHL:
  - 2.94 misp/KI vs 2.84 misp/KI

Not such a huge accuracy loss ☺

And indirect jumps ?

TAGE principles to indirect jumps:

"A case for (partially) tagged branch predictors", JILP Feb. 2006

The 3 first ranked predictors at 3rd CBP in 2011 were ITTAGE predictors

Geometric History Length predictors

- state-of-the-art accuracy using only global information:
  - *Very long history: 200+ bits !!*
- can be ahead pipelined: in-time prediction
- many effective design points
  - GEHL or TAGE ☺
  - Nb of tables, history lengths:
    - Tradeoffs on accuracy against complexity, power
- prediction computation logic complexity is low

(compared with concurrent predictors ☺)

Recent advances

- Storage free and efficient confidence estimator

  HPCA 2011

- Capturing some local history-based extra accuracy

  Micro 2011

  - for large predictors
    - 512Kbits: 1.96 vs 2.12 MPKI

A BP research summary

- 2bit counters 1981: 8.55 misp/KI 671 cycles/KI

No real work before 1991:
win 37 % misp, 10 % perf

- Gshare 1993: 5.30 misp/KI 606 cycles/KI

Hot topic, heroic efforts:
win 28 %, 5 % perf

- EV8-like 2002 (1999): 3.80 misp/KI 576 cycles/KI

Boring topic, a very few actors:
win 33 %, 4 % perf

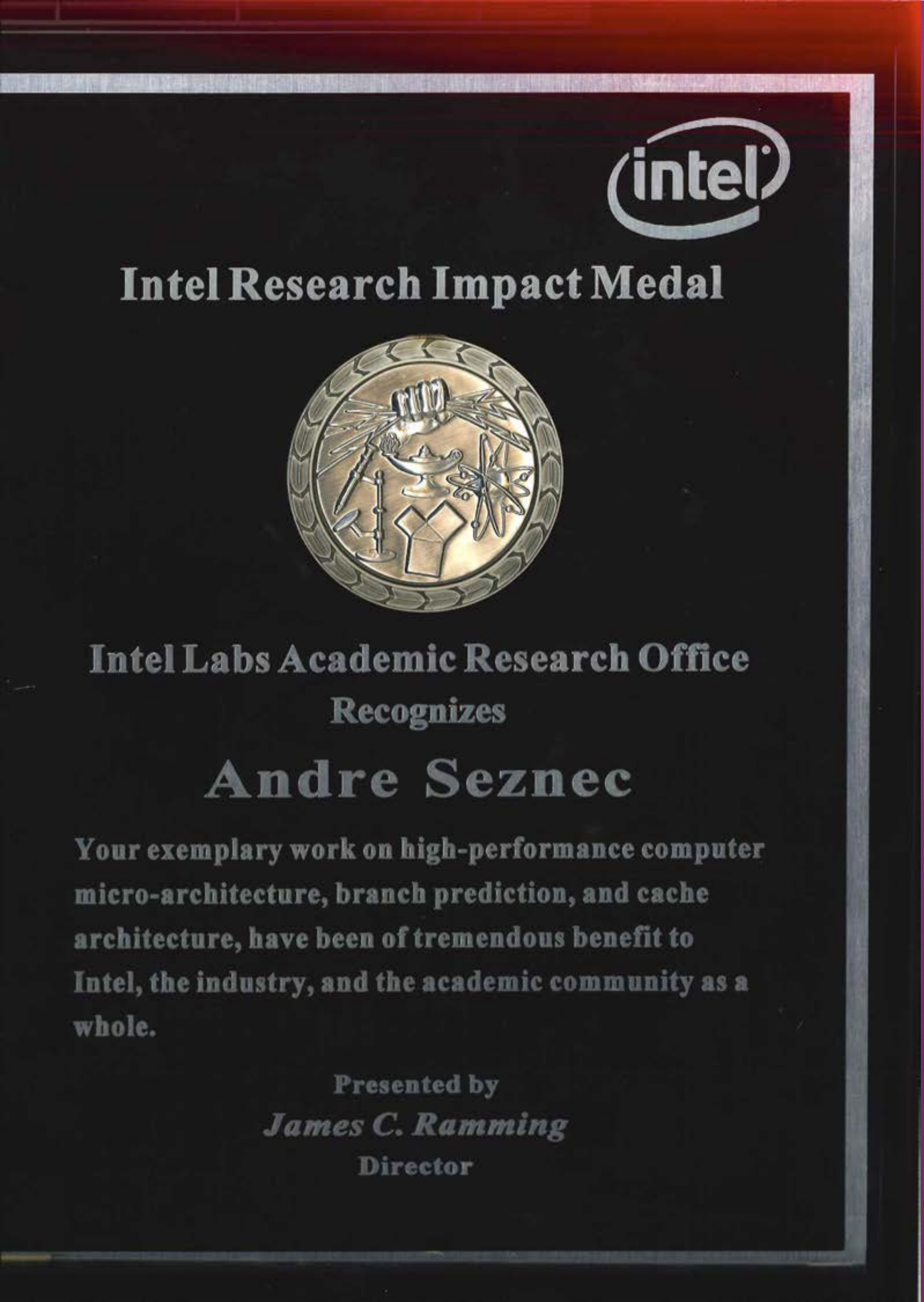
- TAGE 2006: 2.58 misp/KI 551 cycles/KI

But what does real processors use ?

- No precise disclosure

- But :

  1st Intel Research Impact Medal in 2012
  to André Seznec



Intel Research Impact Medal

Intel Labs Academic Research Office

Recognizes

Andre Seznec

Your exemplary work on high-performance computer micro-architecture, branch prediction, and cache architecture, have been of tremendous benefit to Intel, the industry, and the academic community as a whole.

Presented by

James C. Ramming

Director

The End of Branch Prediction research ?

- See the limit study at CBP-2
- Need other new ideas to go further
  - Information source ??

Revisiting Value Prediction (Ongoing Work)

with Arthur Pérais

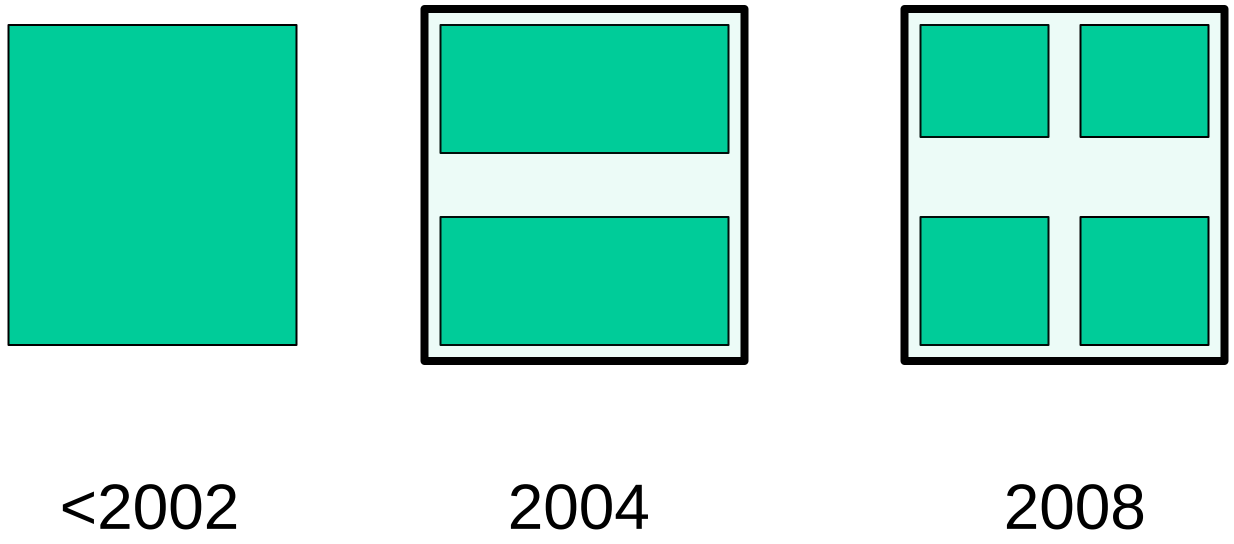
Value Prediction

- Large body of research 96-02

- Disappeared and was not implented

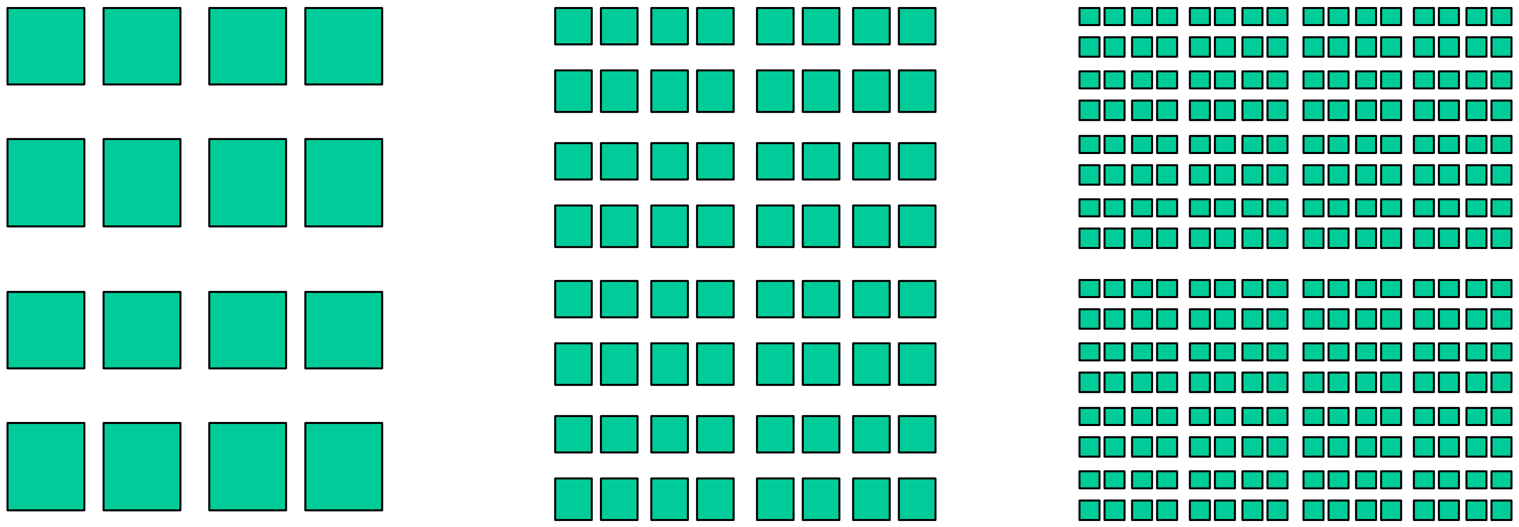
But a new context:
The multi-many core era

The multicore era 2002- ..



GREAT !!

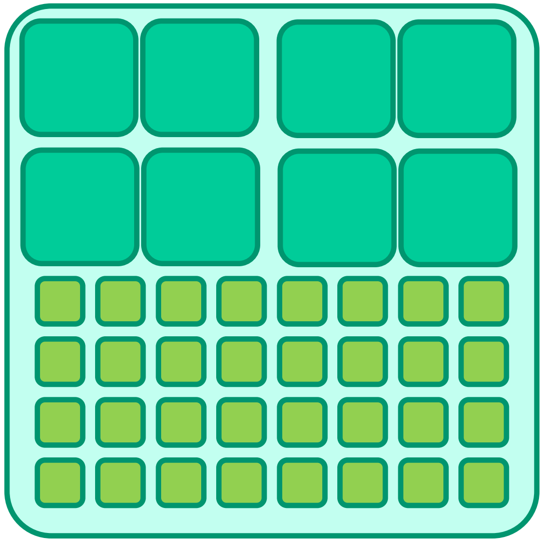
And now ?



Not that great:

- •Amdahl's Law
- •Lots of codes still sequential

May be rather heterogeneous ?



Resource of 10 current cores to one ultra complex core

Objective:

High sequential performance

How:

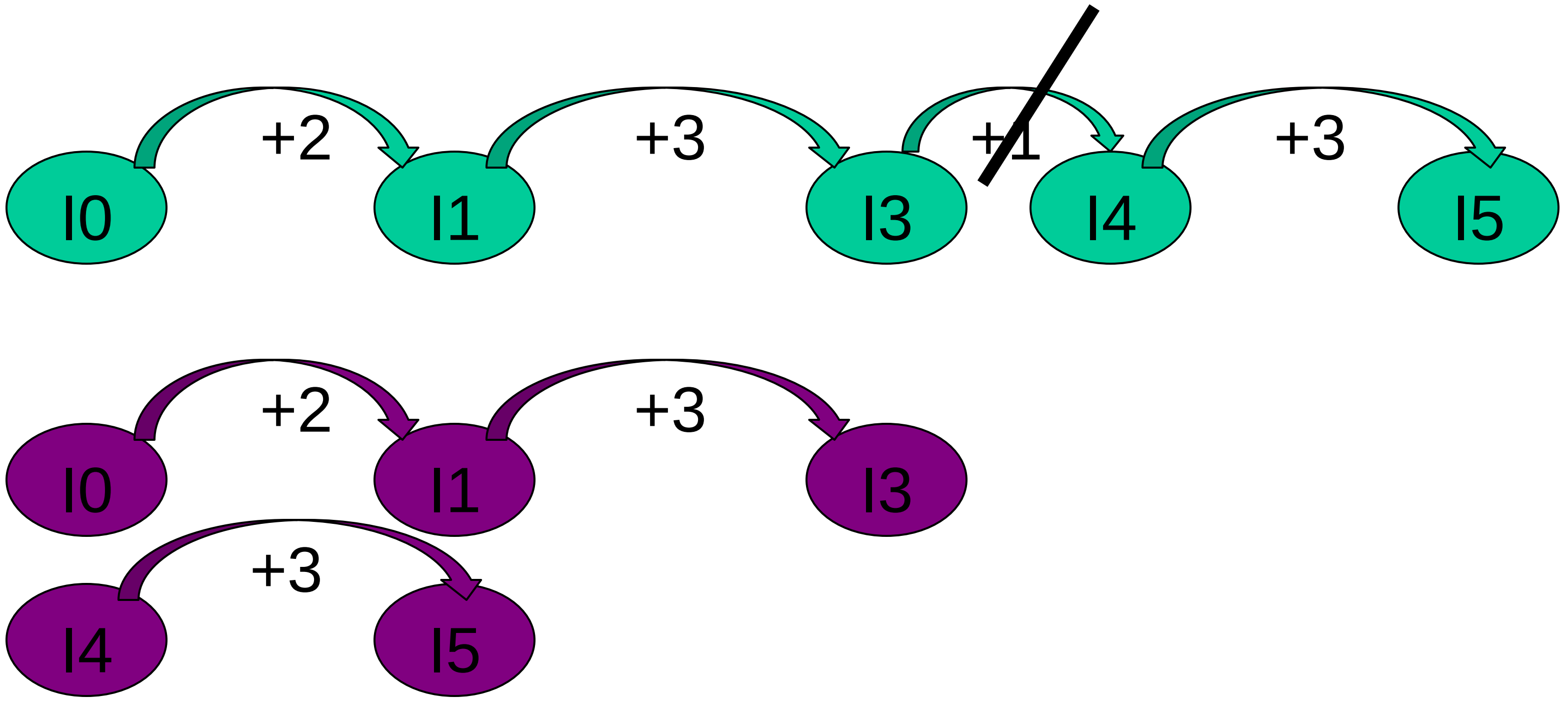
Why not value prediction ?

Value prediction ?

Lipasti et al, Gabbay and Mendelson 1996

Basic idea:

- Eliminate (some) true data dependencies through predicting instruction results



Value Prediction:

- Large body of research 96-02

- Quite efficient:
  - Surprisingly high number of predictable instructions

- Not implemented so far:
  - High cost : *is it still relevant now ?*
  - High penalty on misp.: **don’t lose all the benefit**

What is new ?

- Billions of transistors:
  - And not worth to multiply cores
- Better understanding of confidence issues
  - 95 % accuracy not sufficient
  - >>99 % is the objective
- Better understanding of branch prediction
  - Use it to predict general values

Different value predictors

- Context-based predictors:
  - Use (value) history to predict
  - *Use (branch) history to predict*
    - *This presentation*

- Computational predictors:
  - Apply a function on the previous value(s)

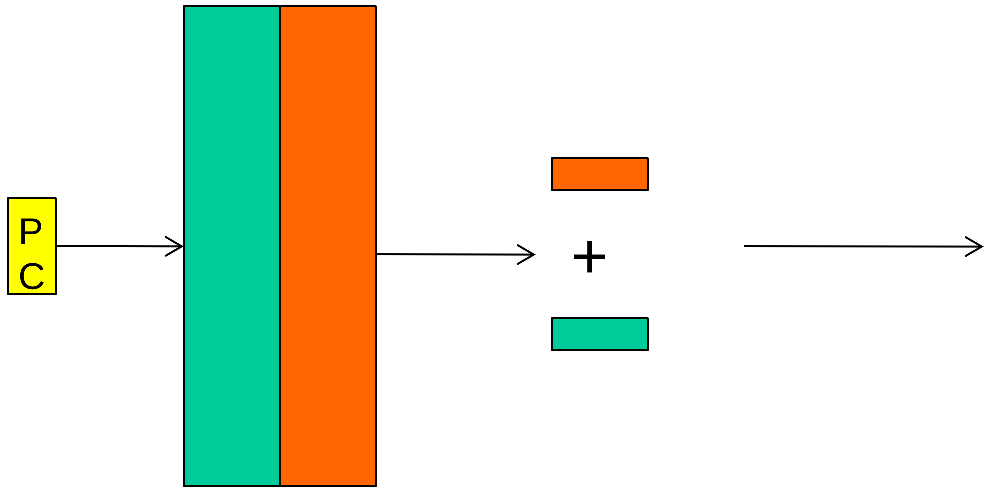
Last Value Predictor

- Just predict the last produced value
  - Set Associative Table
  - Use confidence counters

Analogy with PC-based branch prediction

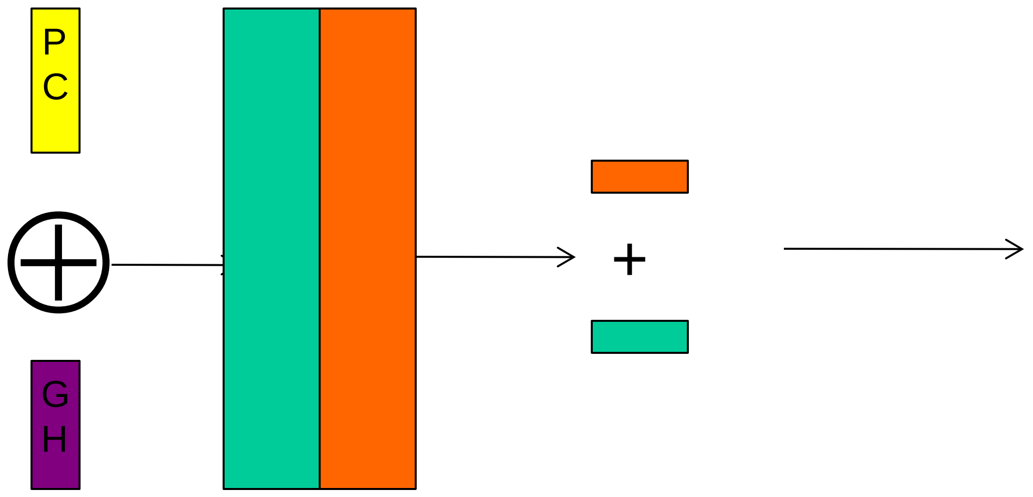
Stride value predictor

- Add last value + (last difference)



Analogy with stride predictor, but also with loop predictor

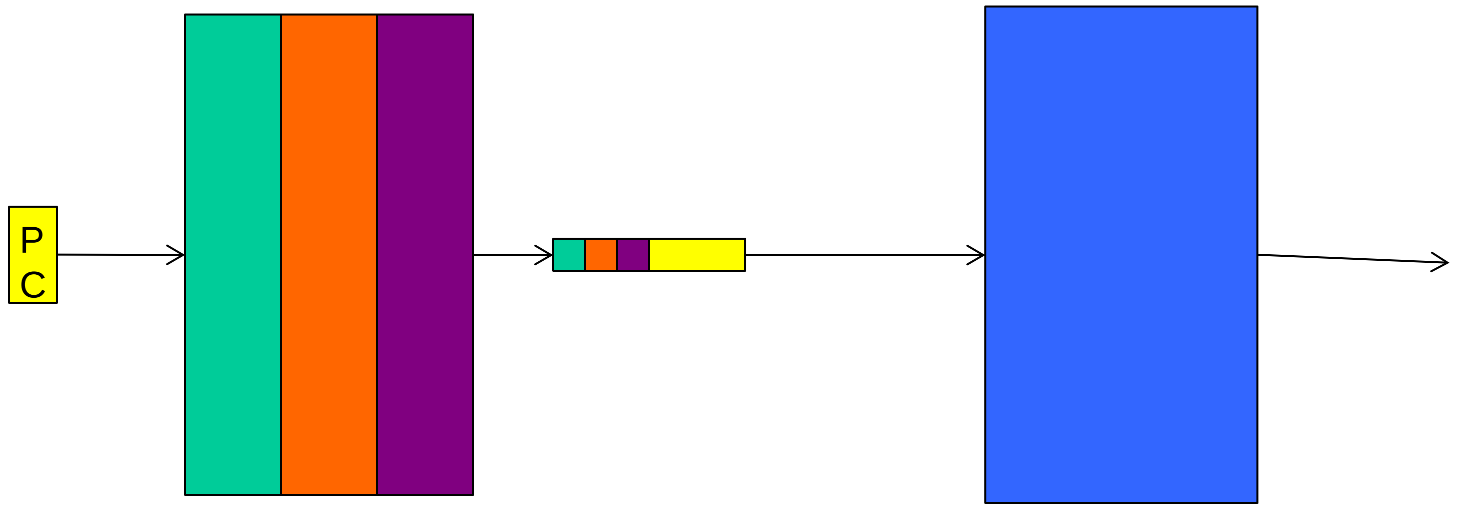
Per-Path Stride



Allows to capture nested loops (without inner body branch)

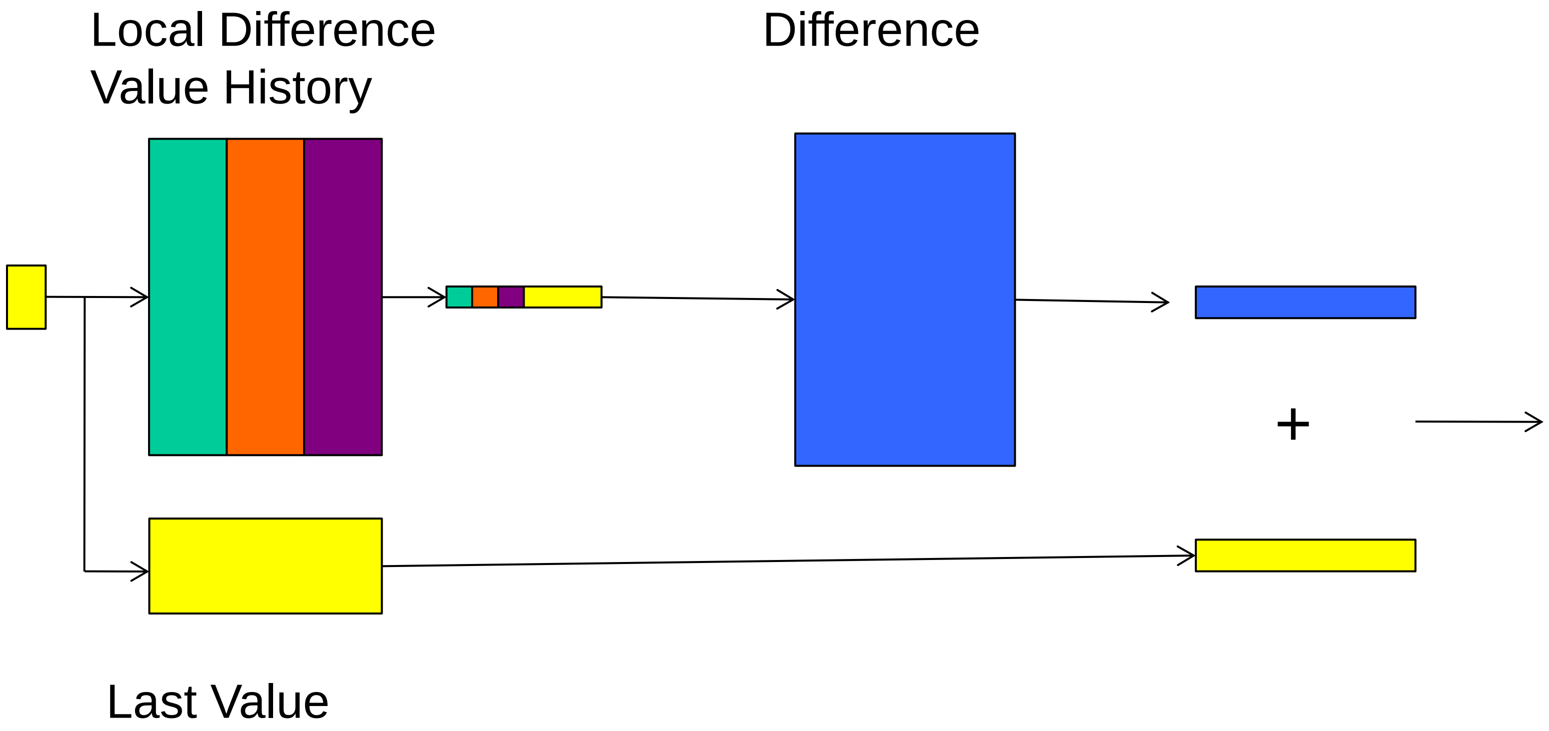
Finite Context Method predictors

Use history of the last values by the instruction



Analogy with local history branch predictor

Differential FCM predictor



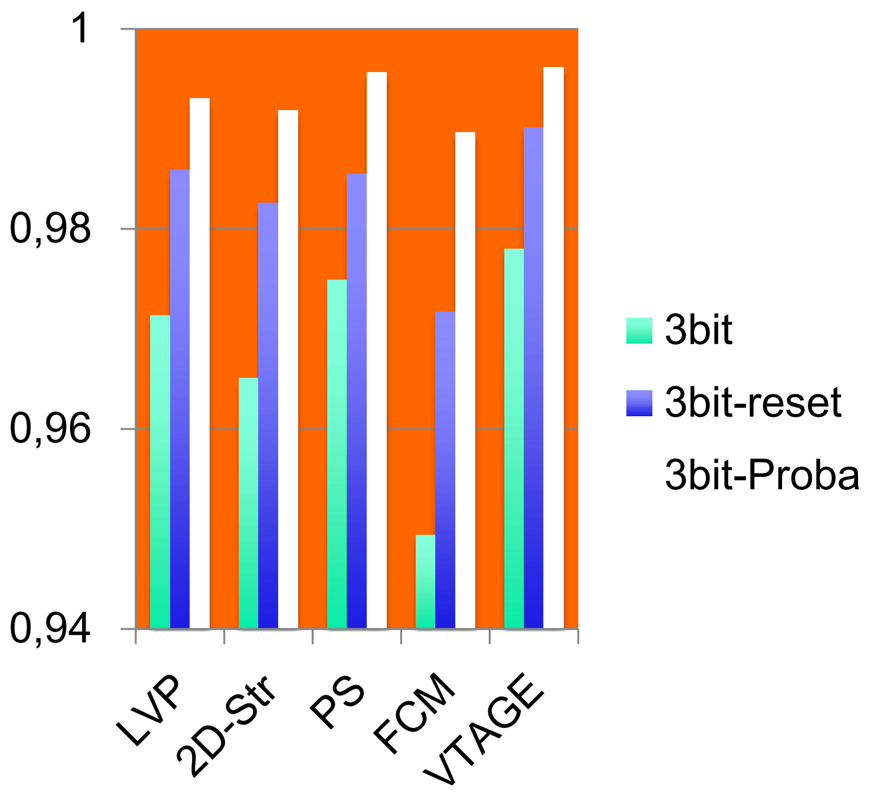
Somekind of hybrid computational/context

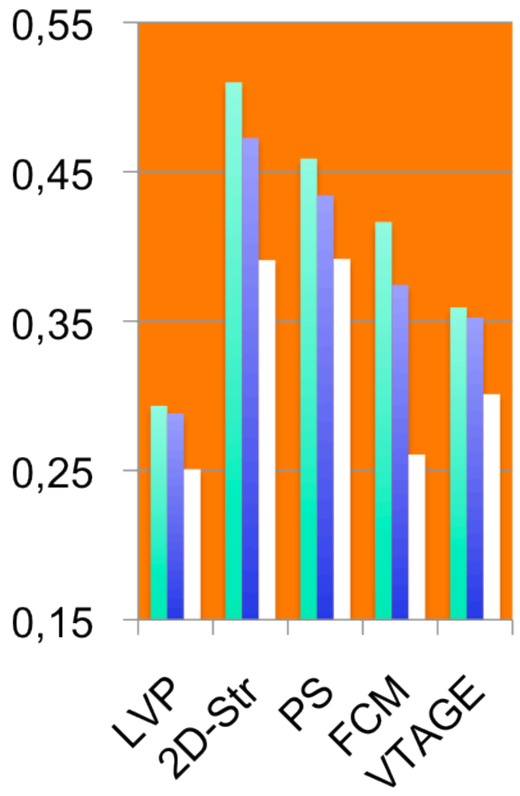
And global ~~value~~ history

- Just no sense !
  - Need the history of the last instructions
    - Too late !!

- But global branch history !?!
  - ITTAGE is the state-of-the-art indirect branch predictor !!
  - And it predicts values !

Accuracy and Coverage (average)





Issues on repair strategies

- Flush the pipeline

- Selective repair

Flush the pipeline

- Branch misprediction like
  - Well understood
- High misprediction cost
  - Need very high confidence accuracy

Selective repair

- Repair and replay on the mispredicted instruction and its dependency chain
  - Complex implementation
  - Complex artefacts

- But certainly already implemented
  - Hit/Miss L1 misprediction

Other artefacts with Value Prediction

Selective replay
Possible cascade of mispredictions

Scenario:

1. Several predictions inflights for Inst I
2. A misprediction is repaired:
   - But the other chained predictions are not

     FCM, strides

Conclusion

- Fresh look at Value Prediction
  - Transpose branch prediction techniques
- High accuracy with high coverage possible
- Local Value Predictors: just unrealistic
- Performance gain ? On going work

General conclusion

Microarchitecture is about

- correctness
- prediction and speculation