It is the Instruction Fetch front-end

Stupid !!

André Seznec

Single thread performance

- Has been driving architecture till early 2000's
  - And that was fun !!
    - Pipeline
    - Caches
    - Branch prediction
    - Superscalar execution
    - Out-of-order execution

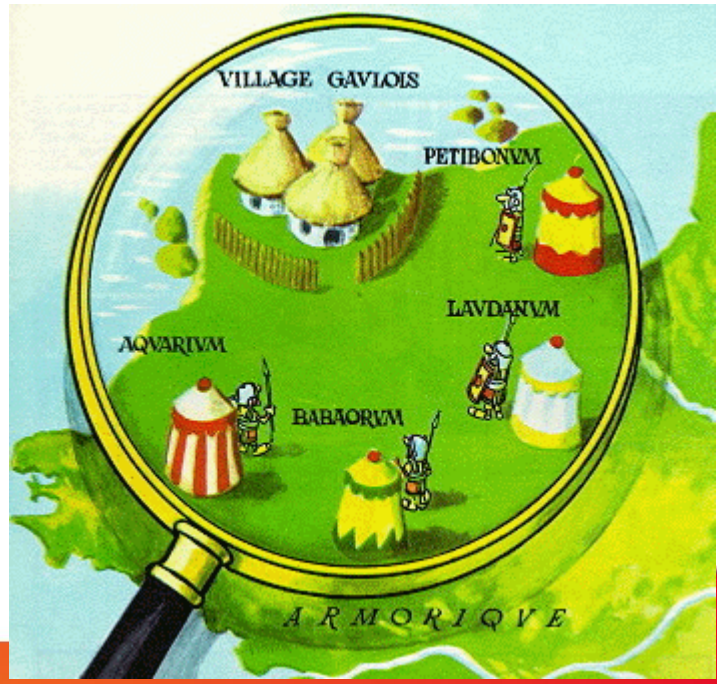
Winter came on the architecture kingdom

- Beginning 2003:
  - The terrible “multicore era”
  - The tragic GPGPU era
  - The Deep learning architecture
  - The quantum architecture

The world was full of darkness

In those terrible days

- Parallelism zealots were everywhere.
- Even industry had abandoned the “Single Thread Architecture” believers
- Among those few:
  - A group at INRIA/IRISA



But “Amdahl’s Law is Forever”

- The universal parallel program did not appear

- Manycores are throughput oriented;
  - The user wants short response time

Could it be that the old religion (single thread architecture) was not completely dead ?

And spring might come back

- Everyone is realizing that single thread performance is the key.
- Companies are looking for microarchitects:
  - Intel, Amd, ARM, Apple, Microsoft, NVIDIA, Huawei, Ampere Computing, ..
- But a nightmare for publications:
  - One microarchitecture session at Micro 2019

So we definitely need

A very wide-issue aggressively speculative supercalar core

Ultra High Performance Core (1)

- Very wide issue superscalar c
  - >= 8-wide
  - Out-of-order e
  - 300-500
    - I ons ?
      dencies ?
      egister file access ?

It's the OoO engine !
Stupid !!

Ultra High Performance Core (2)

- Main memory latency:
  - 200-500 cycles
- Cache hierarch
  - L3-L4:
  - L2
  - 64K, 2-4 cycles
  - tion ?
  - refetch ?
    - Compressed ?

It's the memory hierarchy ! Stupid !!

Ultra High Performance Cor

- 8-instructions per cycle
  - with 500 inst.
  - with 10-1
  - with
  - 8 inst./cycle ?
  - s/memory dependencies ?
  - alues ?

It's the Instruction Front-End ! Stupid !!

A block in the instruction front-end

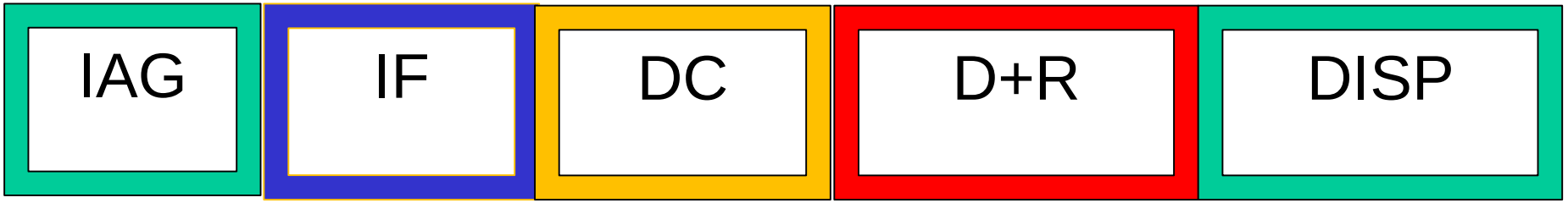
Prediction

I-fetch

Decode

Dependencies
+renaming

Dispatch



+ memory dependency prediction
+ move elimination
+ value prediction (?)

Instruction address generation

- One block per cycle

In practice, not sufficient

- Speculative: accuracy is critical

4 MPKI/ 500 inst window:
75 % wrong pathes

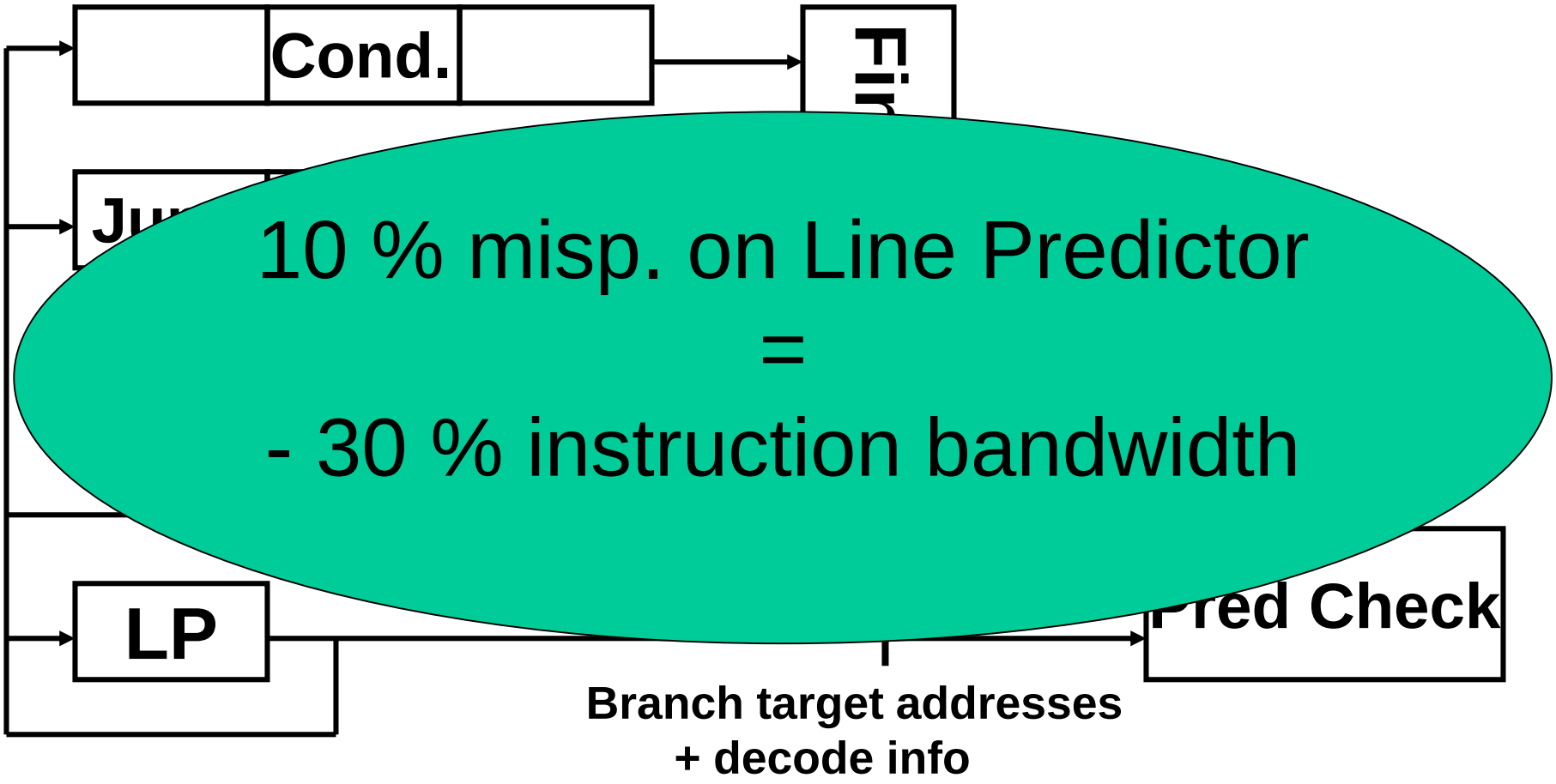
- Accuracy c
  -
  - Seque
  - Return address stack read
  - J
  - Final addres

Will not fit in a single cycle

Hierarchical IAG (example)

- Fast IAG + Complex IAG
- Conventional IAG spans over four cycles:
  - 3 cycles for conditional branch prediction
  - 3 cycles for I-cache read and branch target computation
  - Jump prediction , return stack read
  - + 1 cycle for final address selection
- Fast IAG: Line prediction:
  - a single 2Kentry table + 1-bit direction table
  - select between fallthrough and line predictor read

Hierarchical IAG (2)



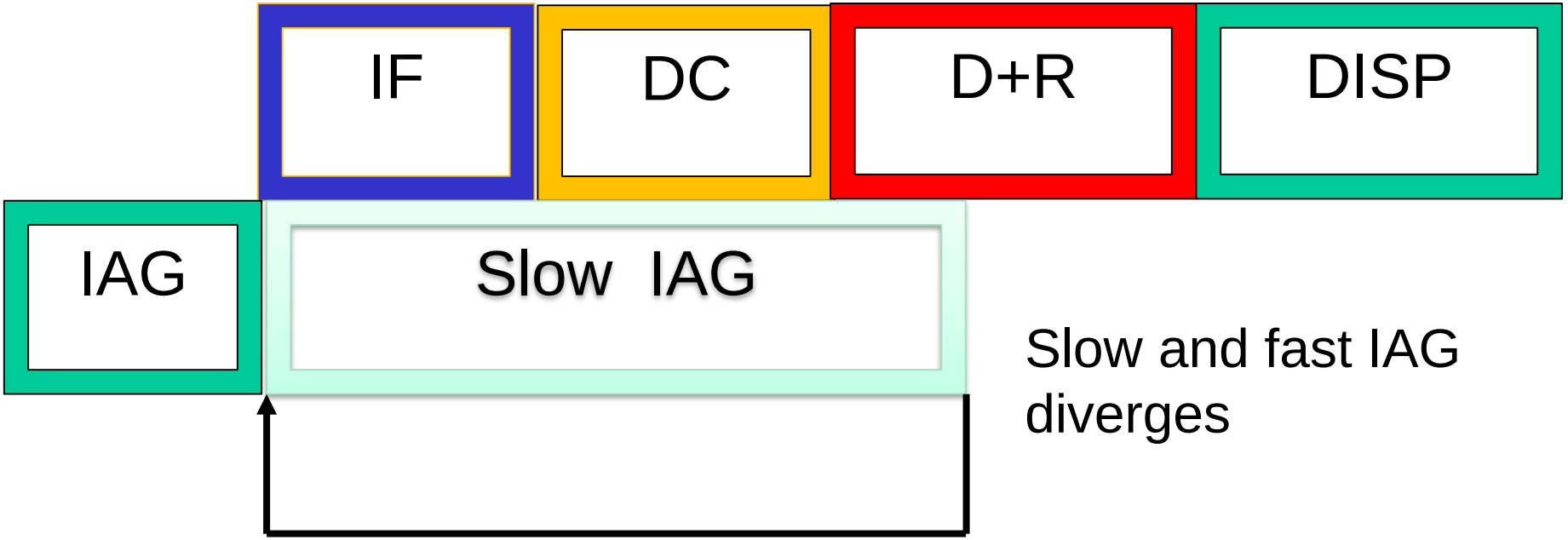
So ?

- You should fetch as much as possible:
  - Contiguous blocks
    - Across contiguous cache blocks !
    - Bypassing not-taken branches !
  - More than one block par cycle ?

Example: Alpha EV8 (1999)

- Fetches up to two, 8-instruction blocks per cycle from the I-cache:
  - a block ends either on an aligned 8-instruction end or on a taken control flow
  - up to 16 conditional branches fetched and predicted per cycle
- Next two block addresses must be predicted in a single cycle

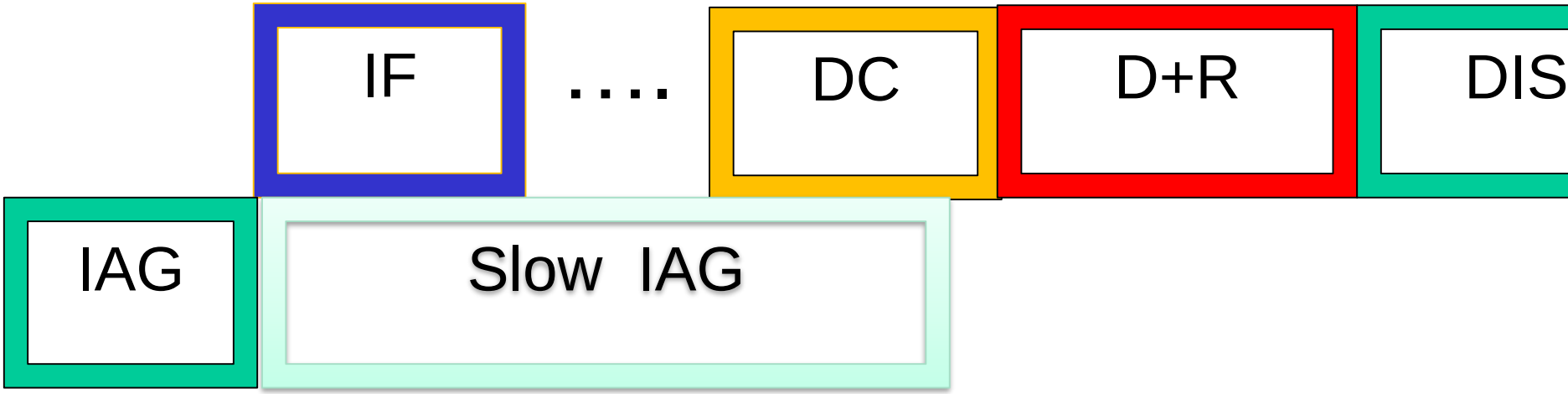
A block in the instruction front-end



Slow and fast IAG diverges

If you overfetch ..

- Add buffers;



Decode is not an issue

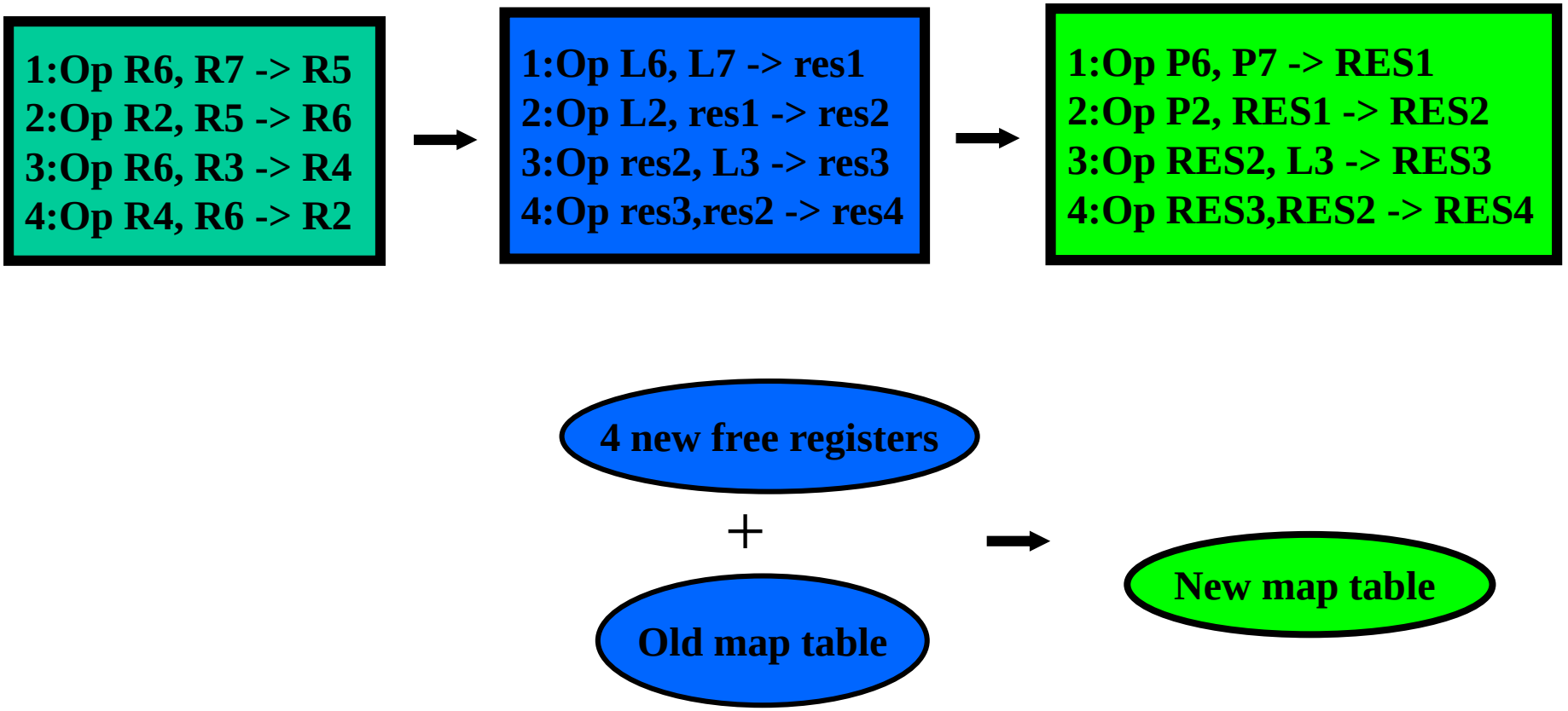
- If you are using a RISC ISA !!
- Just a nightmare on x86 !!

Dependencies marking and register renaming

- Just need to rename 8 (or more) inst per cycle:
  - Check/mark dependencies within the group
  - Read old map table
  - Get up to 8 free registers
  - Update the map table

The good news:
It can be pipelined

Dependencies marking and register renaming (2)



OK, where are we ?

- Very long pipeline:
  - ≈ 15-20 cycles before execution stage
  - Misprediction is a disaster
- Very wide-issue
  - Need to fetch/decode/rename ≧ 8 inst/cycles
  - mis(Fast prediction) is an issue
  - Misses on I-caches/BTB also a problem

Why branch prediction ?

- 10-30 % instructions are branches
- Fetch more than 8 instructions per cycle
- Direction and target known after cycle 20
  - Not possible to lose those cycles on each branch
  - PREDICT BRANCHES
    - and verify later !!

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 per 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 ?!?

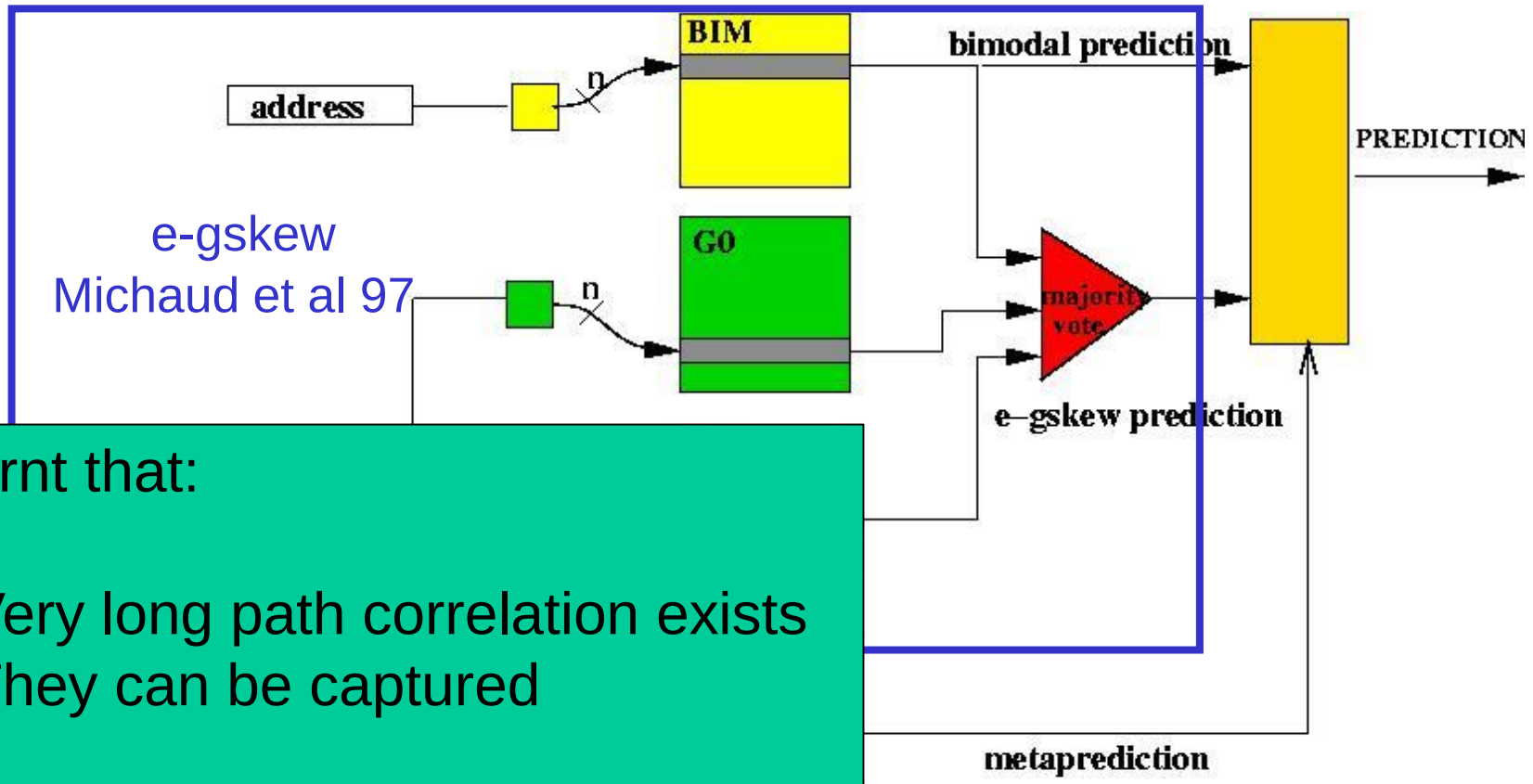
- Global history:
  - Append a bit on a single history register
  - Use of a circular buffer and just a pointer to speculatively manage the history

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

Essentially relied on 2-bit counters

EV8 predictor (1999): (*derived from*) 2bc-gskew

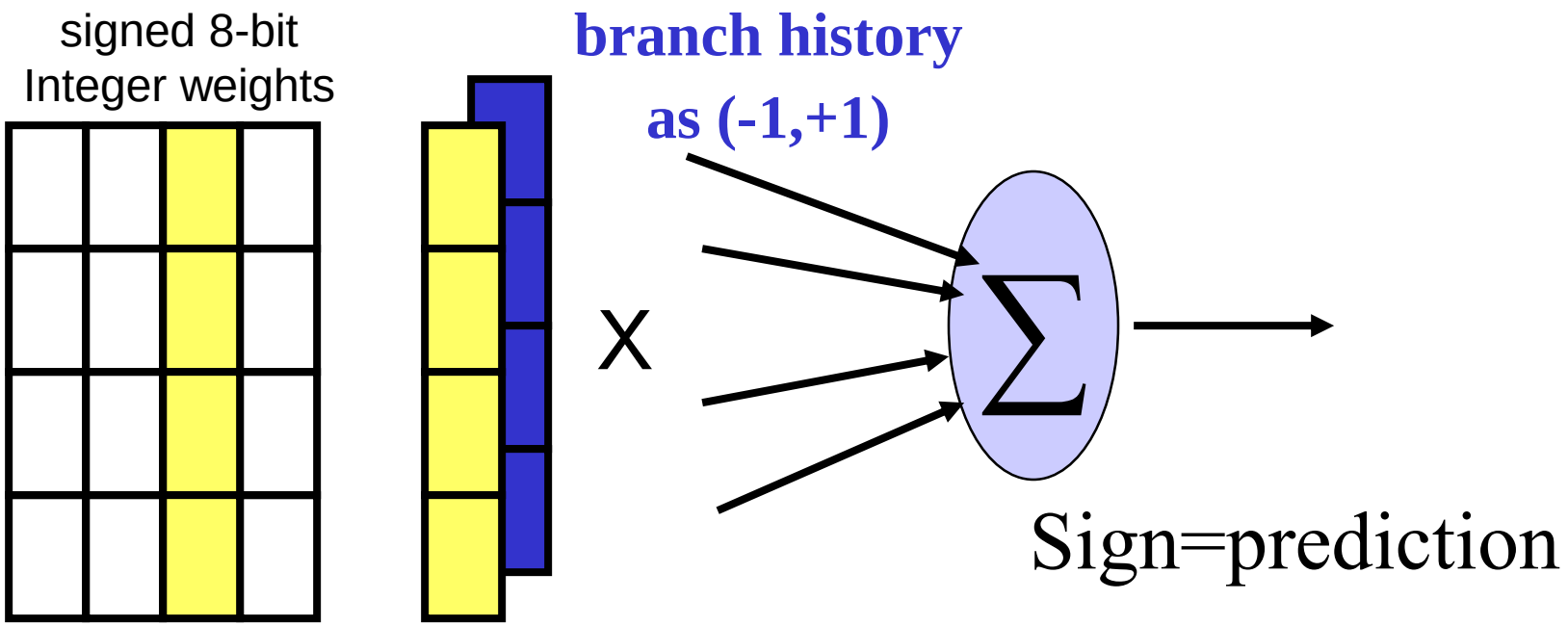


Learnt that:

- Very long path correlation exists
- They can be captured

In the new world

A UFO : The perceptron predictor
Jiménez and Lin 2001

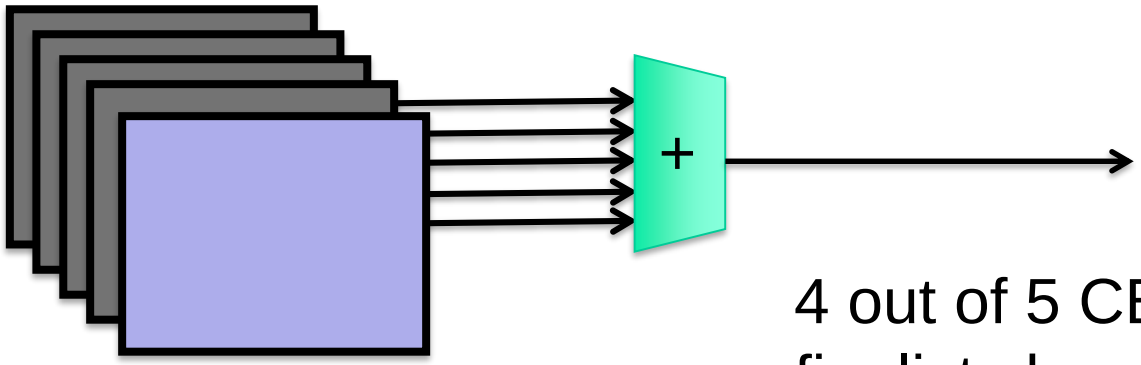


Update on mispredictions or if |SUM| < θ

(Initial) perceptron predictor

- Competitive accuracy
- High hardware complexity and latency
- Often better than classical predictors
- Intellectually challenging

Rapidly evolved to



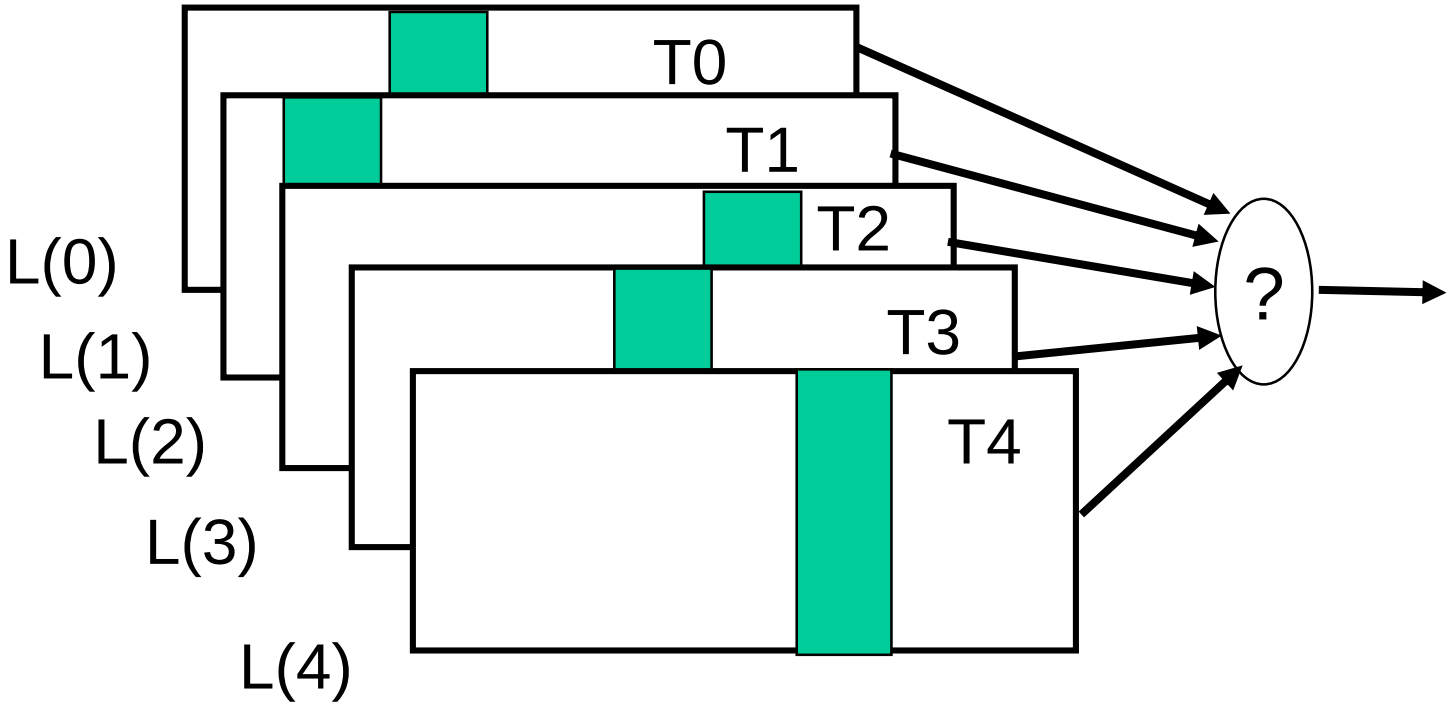
4 out of 5 CBP-1 (2004) finalists based on perceptron,

Can combine predictions:
-global path/branch history
-local history
-multiple history lengths
- ..

An answer

- The geometric length predictors:
  - GEHL and TAGE

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. $L(1) \leq N < L(2)$
- Branches (A,H) and (A,H')

  biased in opposite directions

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

GEometric History Length predictor

The set of history lengths forms a geometric series

$$L(0) = 0$$

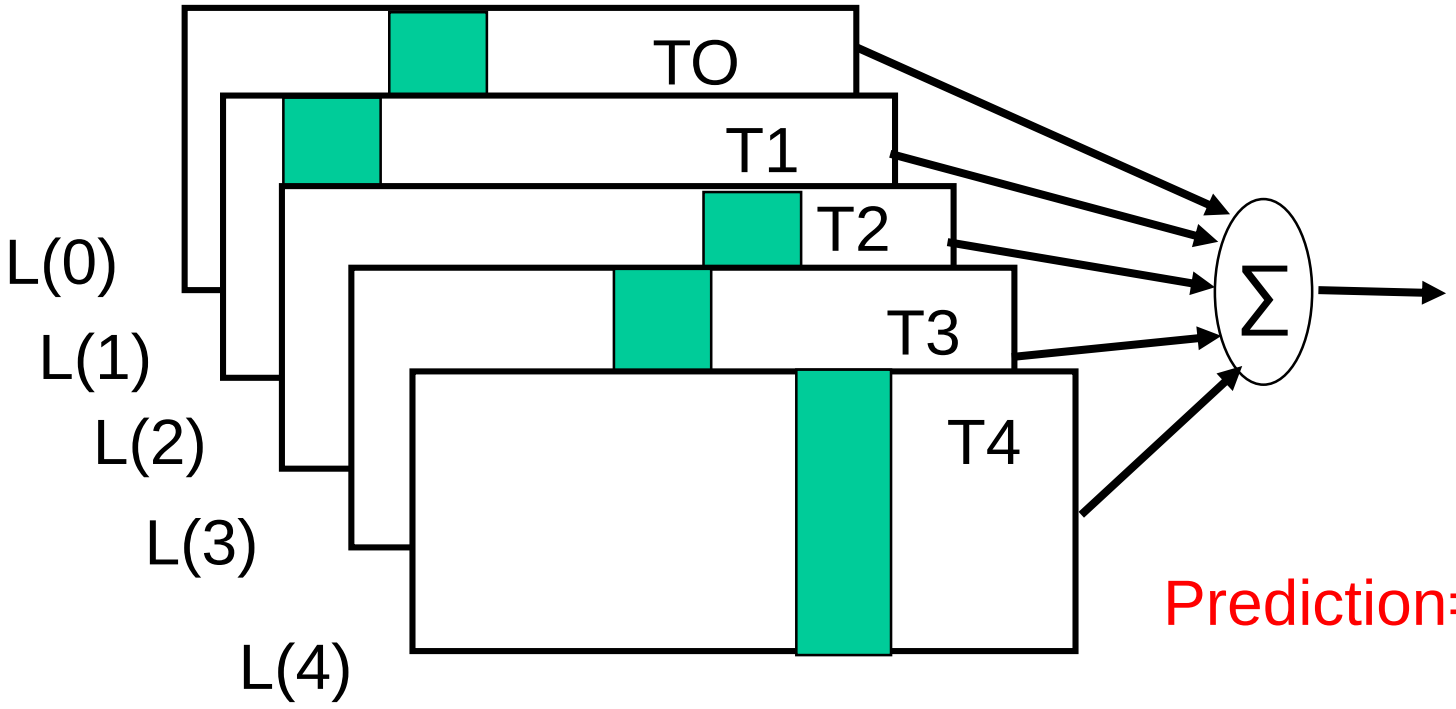
$$L(i) = a^{i-1} L(1)$$

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

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

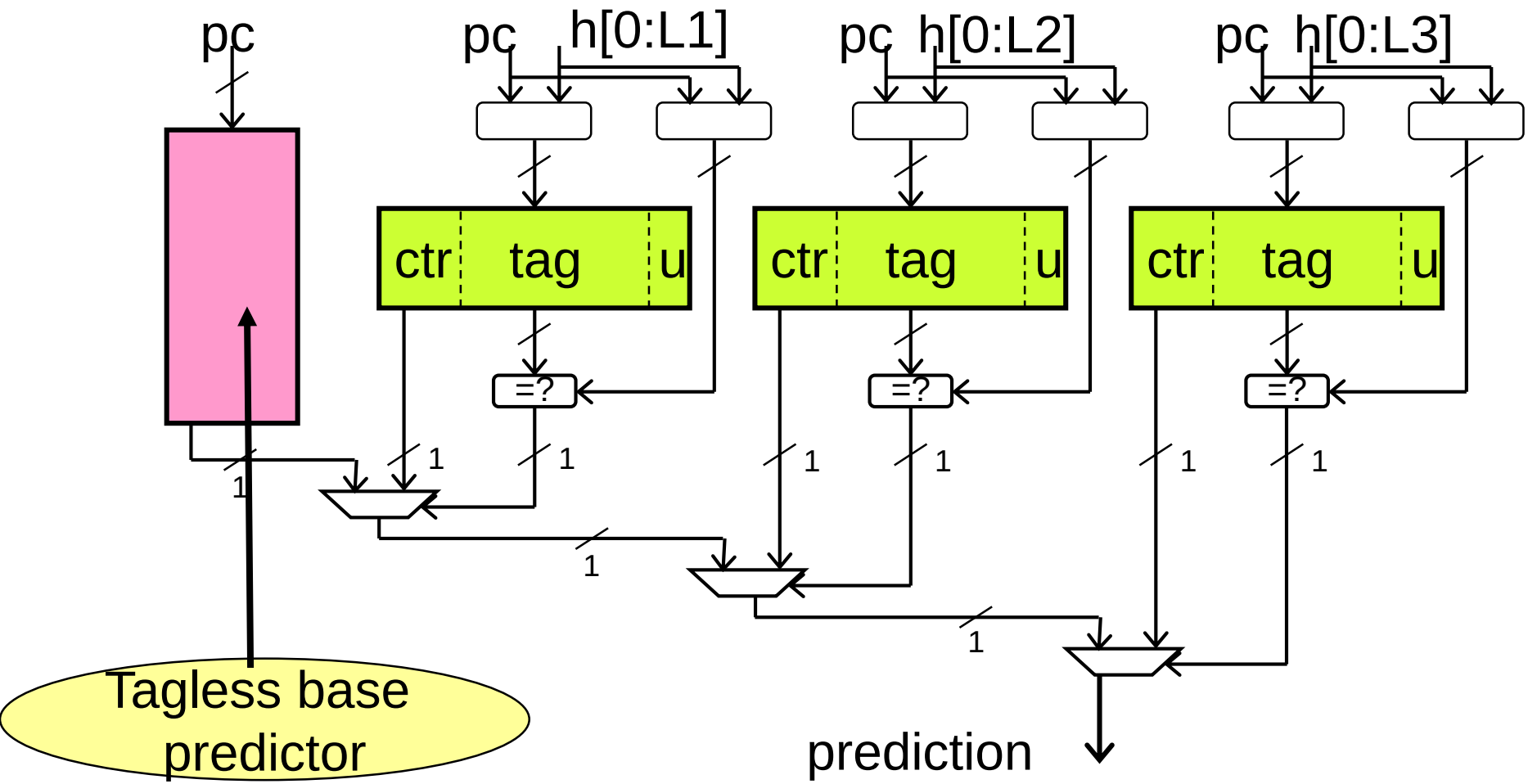
Spends most of the storage for short history !!

GEHL (2004) prediction through an adder tree



Using the perceptron idea with geometric histories

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
    + geometric length
    + optimized update policy
    - Basis of the CBP-2,-3,-4,-5 winners
- Inspiration for many (most) current effective designs

A BP research summary (CBP1 traces)

- 2bit counters 1981: 8.55 misp/KI
- Gshare 1993: 5.30 misp/KI
- EV8-like 2002 (1999): 3.80 misp/KI
- CBP-1 2004: 2.82 misp/KI
- TAGE 2006: 2.58 misp/KI
- TAGE-SC 2016: 2.36 misp/KI

No real work before 1991:
win 37 %

Hot topic, heroic efforts:
win 28 %,

The perceptron era, a few actors:
win 25 %

TAGE introduction:
win 10%,

A hobby for AS and DJ :
win 10%,

And indirect jumps ?

TAGE principles to indirect jumps:

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

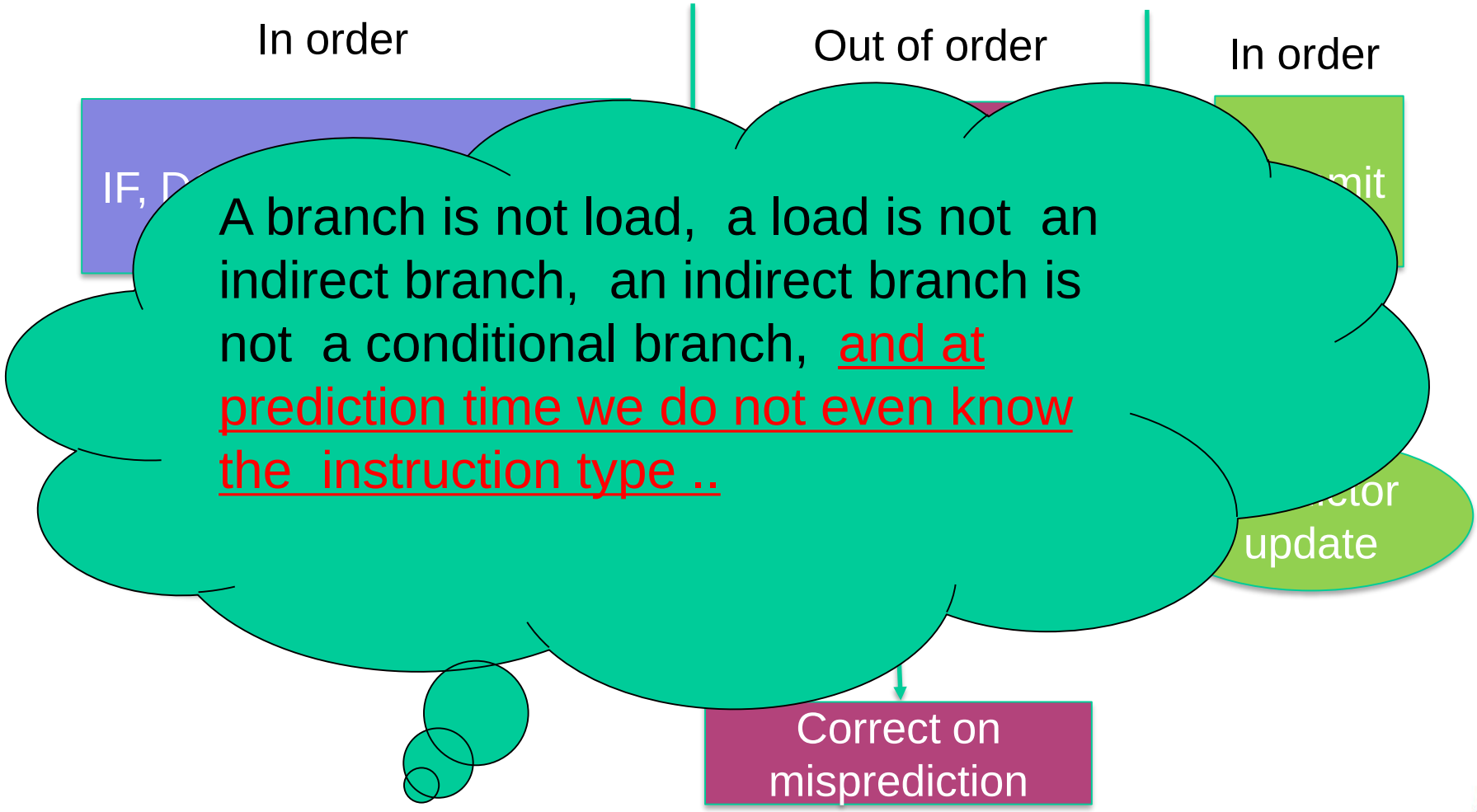
The 3 first ranked predictors at 3rd CBP in 2011 were ITTAGE predictors

Memory (in)dependencies predictors

To allow load and stores to execute out-of-order

- Naive: dependent/independent
- Wait: e.g. Store sets
- Store forwarding: bypass the cache
- Register producer to consumer forwarding

A speculation opportunity on RISC ISA



The Omnipredictor (PACT 2018)

- Consolidating several types of speculation in a single predictor structure : **TAGE**.

- Memory dependency prediction and indirect target prediction through TAGE and the BTB at **zero storage overhead**.

- **Omnipredictor:** a good fit for mid-range cores with constrained hardware budget

Value Prediction ?

- Also in the front-end ..
  - Predictions should be done in the front-end
  - Control-flow could be used to predict
    - Values
    - Value equality
    - Register equality

Issues in Front-End

- High instruction footprint applications (servers, cloud, web browsers, ..)
  - Instruction cache misses
  - BTB misses

Summary

- Single thread performance was, is and will be a major issue:
  - Industry is eager to deliver, but limited progress

- More « a la grand papa » microarchitects needed

A few relevant publications

- A. Seznec, S. Felix, V. Krishnan, Y. Sazeides , "Design trade-offs on the EV8 branch predictor", ISCA 2002
- A. Seznec, P. Michaud, **" A case for (partially) tagged Geometric History Length Branch Prediction"**, JILP, Feb. 2006,
- A. Perais ,A. Seznec. Practical Data Value Speculation for Future High-end Processors. HPCA 2014
- A. Perais, F.A. Endo, A.Seznec. Register Sharing for Equality Prediction. Micro 2016,
- A. Perais, A. Seznec, Cost Effective Speculation with the Omnipredictor *PACT '18*