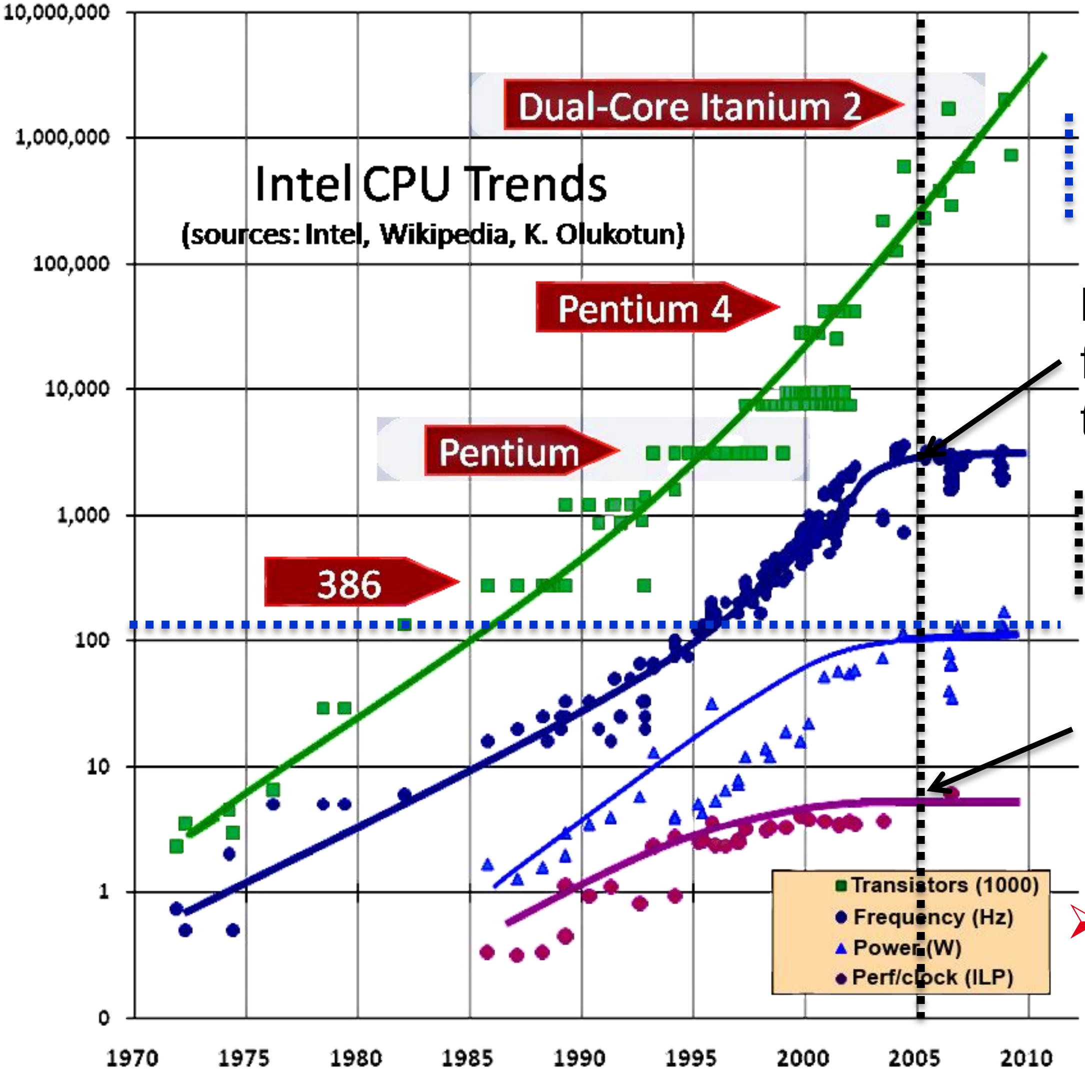
Increasing the Performance of Superscalar Processors through Value Prediction

Arthur Perais

Past and Recent Trends in General Purpose CPUs



Power wall: around 2005.

Power Wall: Stop increasing the frequency if you want your additional transistors.

Dennard scaling* goes to transistor heaven in ~2005/07.

~~**Dennard scaling**~~: Power density is up: Avoid hotspots if you want your additional transistors. Go multicore.

- Transistor count increases, but sequential performance **stagnates**.

*Current and Voltage scale downwards with transistor size.

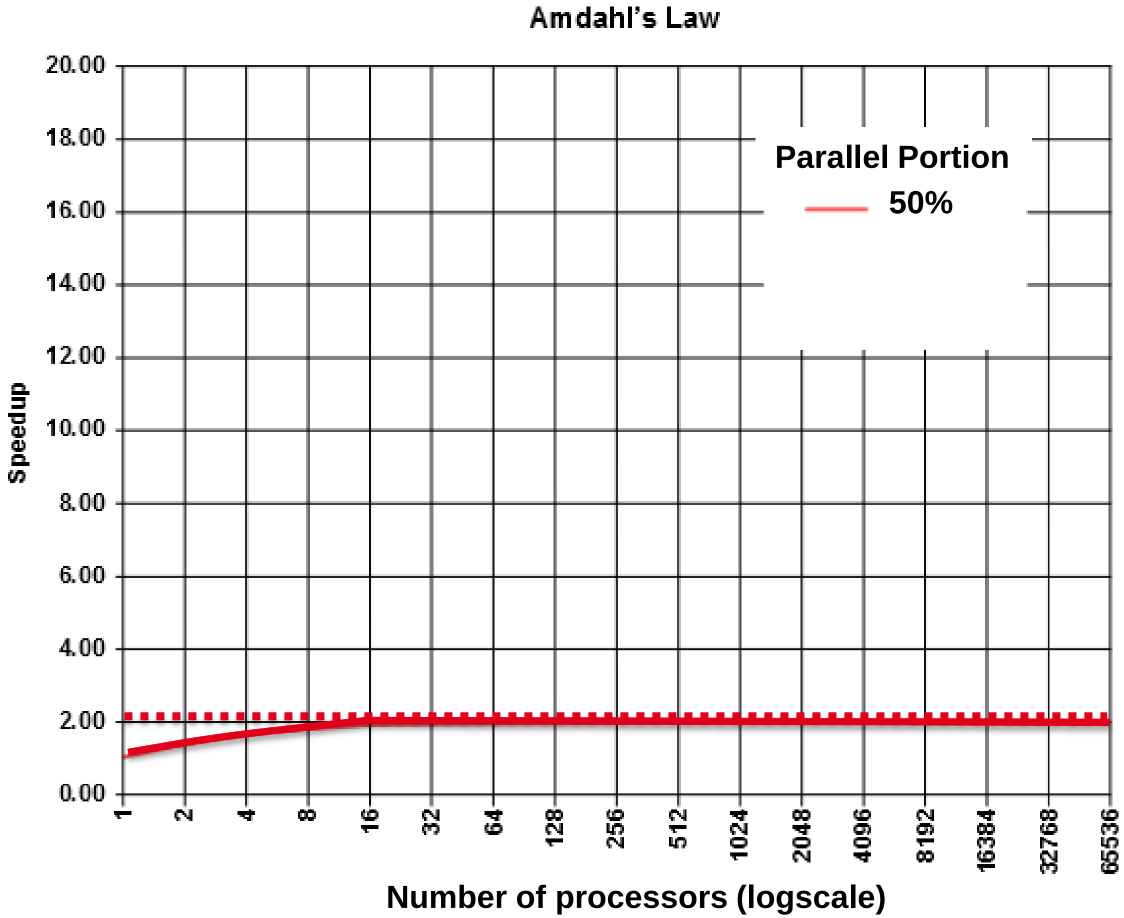
The Multicore Era and Amdah’l Law

- Sequential performance stagnates, but we have several cores:
  - Thread everything.
  - n times faster with n cores (at best).

- What if the code is sequential?

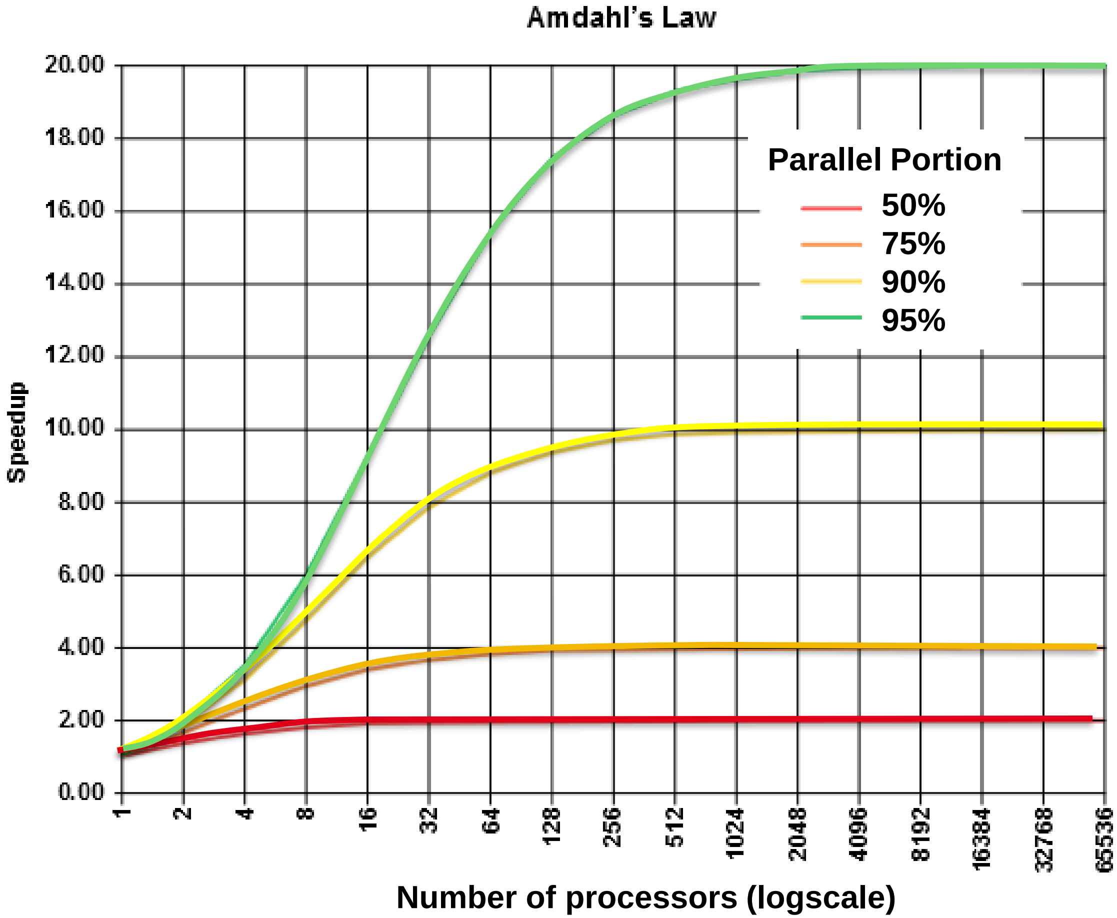
➢ Amdah’l Law (is Forever)

The Multicore Era and Amdah'l Law



- Intrinsically sequential codes do not benefit from multicores. Speedup maxes out at 2.

The Multicore Era and Amdah'l Law



- Fairly parallel codes do not even benefit that much.

Increasing Sequential Performance

$$Perf \sim (Width, ILP^1)$$

- Intuitive way: Increase the processor width.
  1. Super-linear increase in complexity and power.
  2. Timing issues.
  3. Only helps when ILP is already high.

- Instead of increasing “raw” compute capability, try to improve utilization of existing hardware (ILP).

[1] **I**nstruction **L**evel **P**arallelism

Increasing ILP

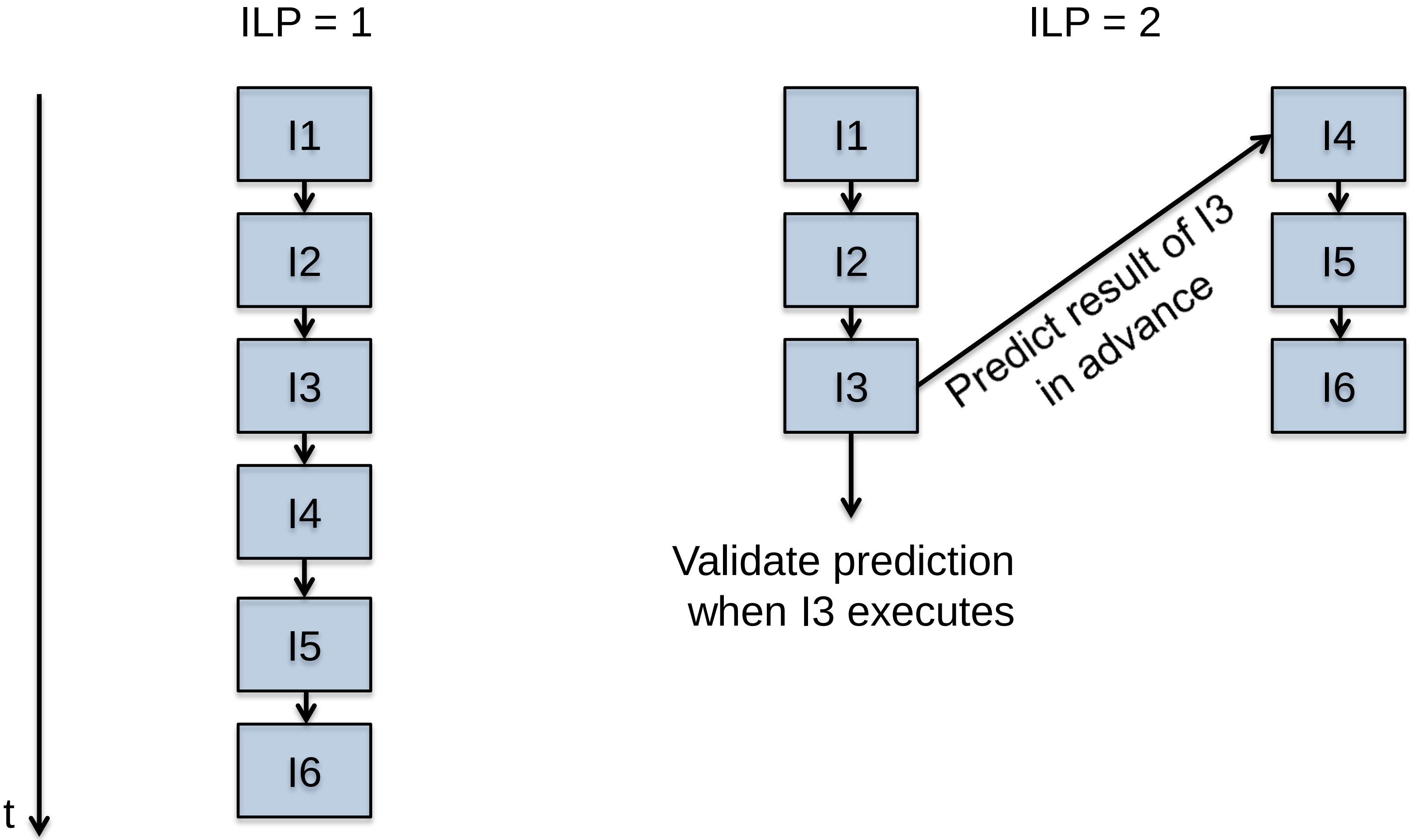
- Processors already do that:
  1. Branch prediction: Control dependencies (speculative).
  2. Renaming: False dependencies (non speculative).
  3. Memory Dependency Prediction: Reveal RAW* dependencies for memory instructions (speculative).

- Performance increases through **better utilization**.

- What if we could **remove** RAW dependencies to further improve utilization?

*Read-after-Write.

Value Prediction (VP) [Lipasti96][Mendelson97]


ILP = 1
ILP = 2
t
I1
I2
I3
I4
I5
I6
I1
I2
I3
I4
I5
I6
Predict result of I3 in advance
Validate prediction when I3 executes

Increasing Performance Through VP - Roadmap

A. Revisiting Value Prediction[1]

1. Existing Predictors and Their Shortcomings
2. Complex Prediction Validation
3. Evaluation

B. Complexity Remains in the Register File[2]

1. VP Requires Additional Ports on the Register File
2. Reducing the Execution Engine Complexity through Value Prediction
3. Evaluation

[1] Perais and Seznec, HPCA’14 & HPCA’15

[2] Perais and Seznec, ISCA’14 & IEEE MICRO’s TP’14

A

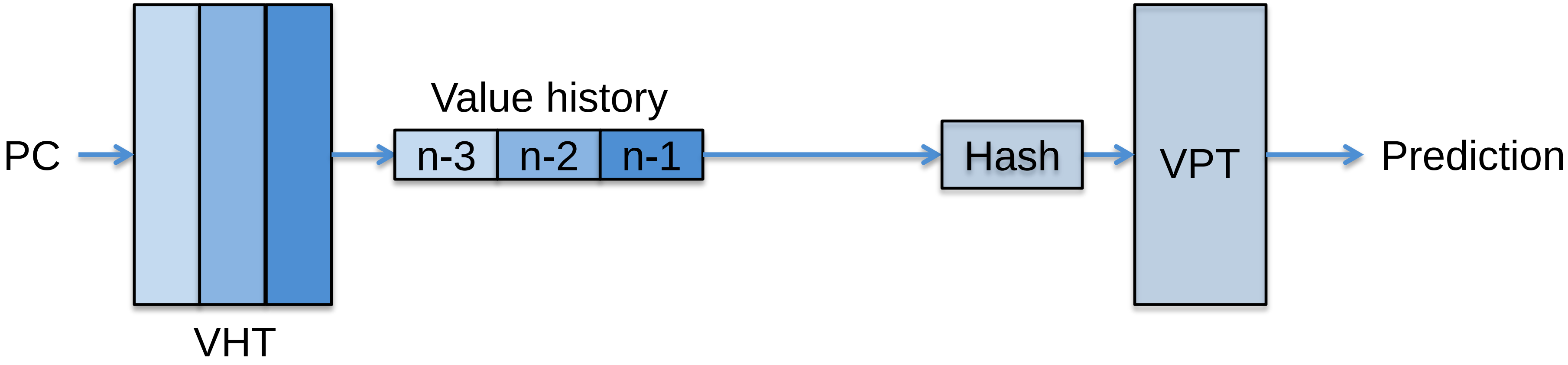
Revisiting Value Prediction

1. Existing Predictors and Their Shortcomings

Different Prediction Schemes

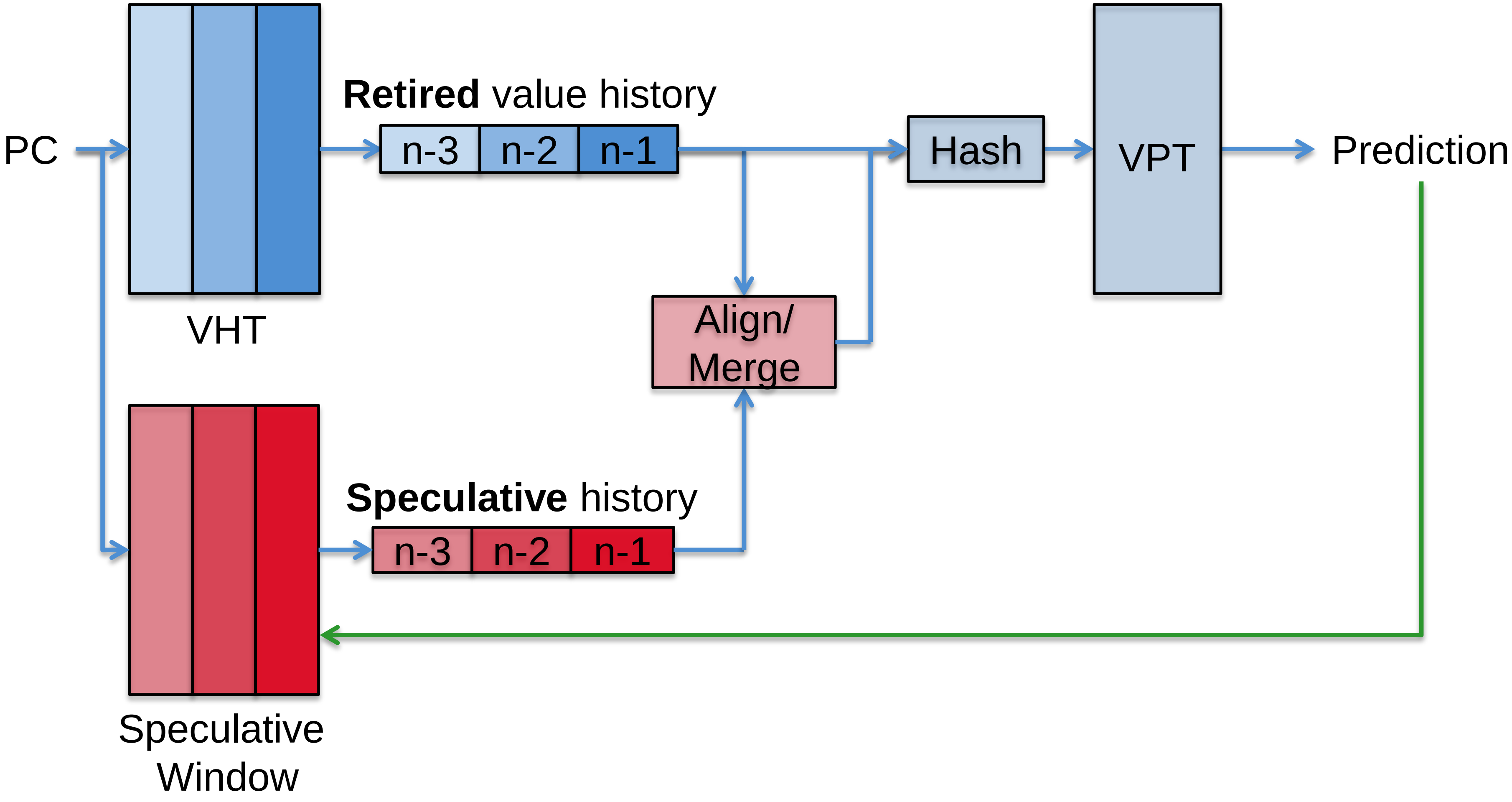
- Context-based: Observe the stream of local values and identify patterns:
  - Finite Context Method (FCM) [Sazeides97&98].

FCM Predictor: Regular Prediction

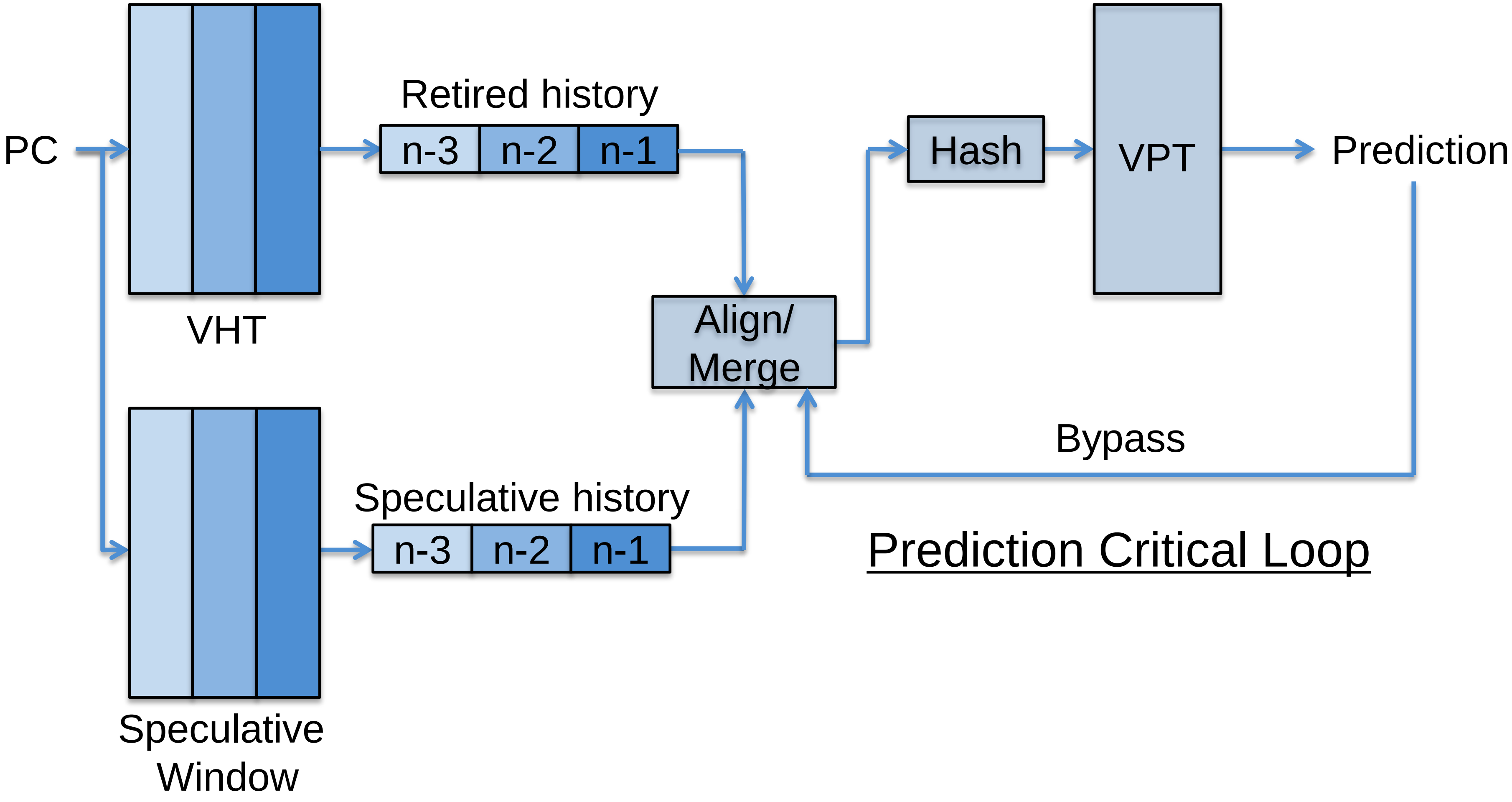


- What if the last value has not been retired or even computed?

FCM Predictor: Inflight Predictions



FCM Predictor: Back-to-back Prediction

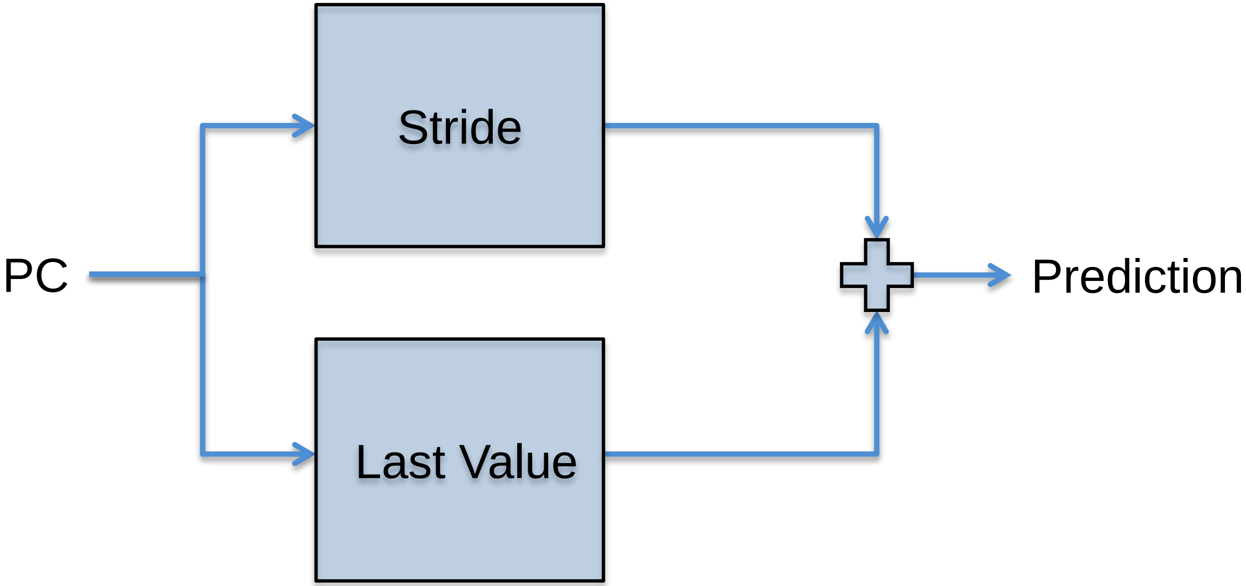


Prediction Critical Loop

Different Prediction Schemes

- Computational: Apply a function to the last result/prediction:
  - Last Value Predictor [Lipasti96].
  - Stride [Gabbay98], 2-delta Stride [Eickemeyer93].

Stride Predictor(s)



Stride Predictor(s)

- What if the last value has not been retired or even computed?
  - A speculative window is required (as in FCM).

- Back-to-back prediction of two instances?
  - Bypass the first prediction to the input of the adder: **Doable** in a single cycle.

Prediction Schemes: Major Shortcomings

- A speculative window is required. How do we actually build it?
- Existing context-based predictors are not adapted in case of predictable tight loops because of the long prediction critical loop.

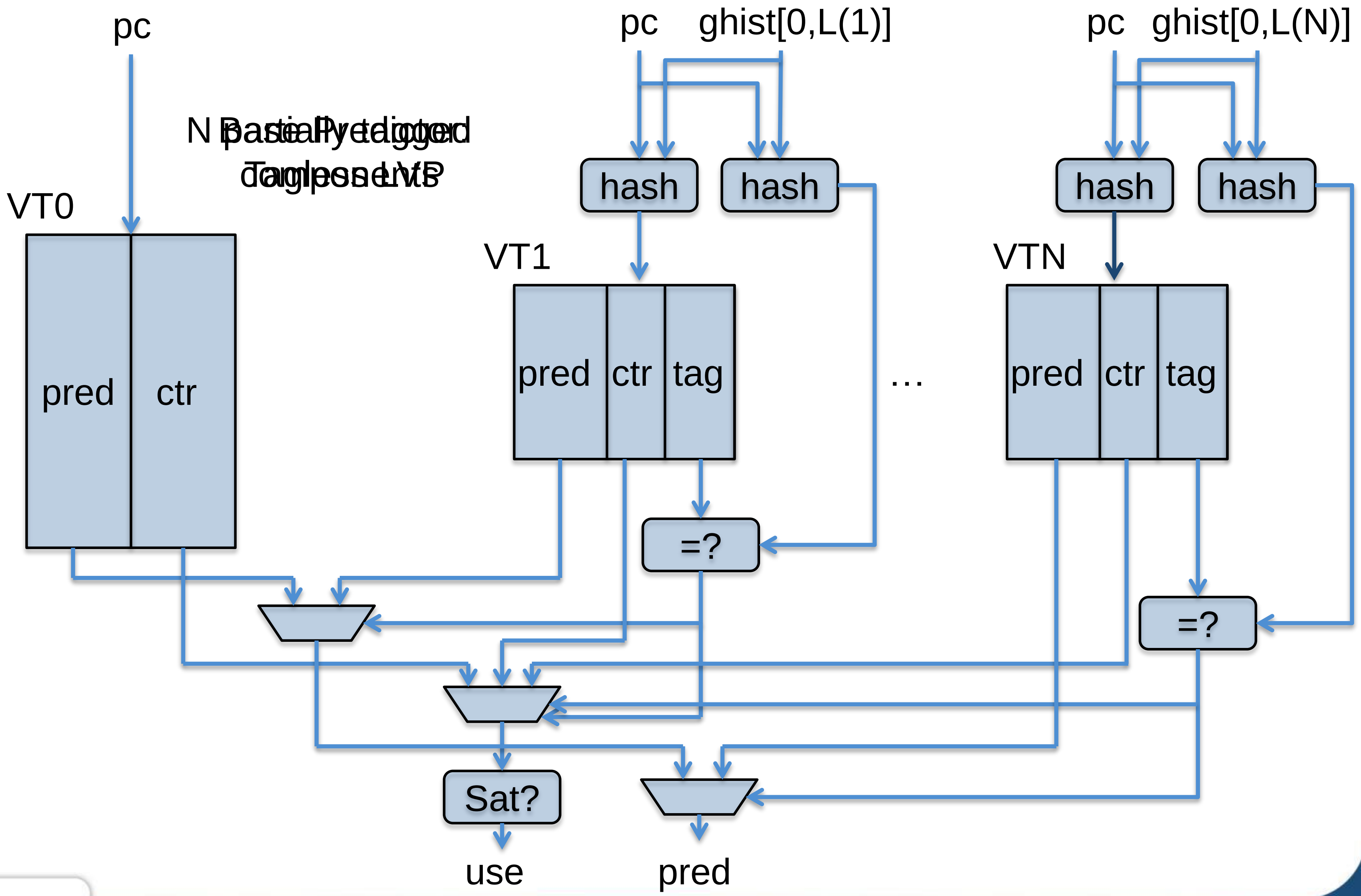
Learning From Branch Prediction

- Remove the need for the previous result(s) to generate a prediction by using control-flow as context.

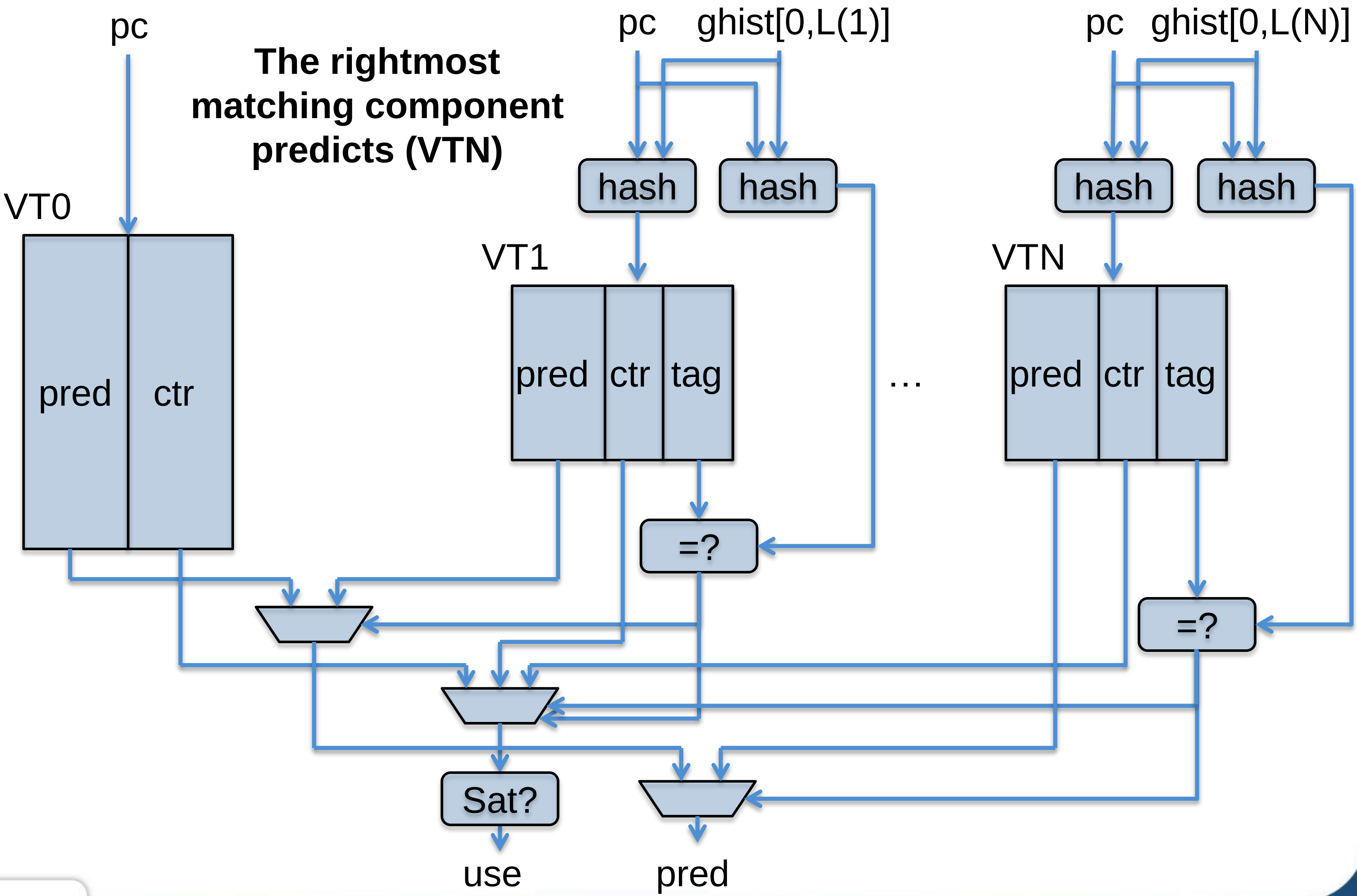
- Indirect Target Prediction is a specific case of Value Prediction.

➢ Modify ITTAGE [Seznec06] to handle all instructions eligible for VP.

The Value TAgged GEometric Predictor (VTAGE)



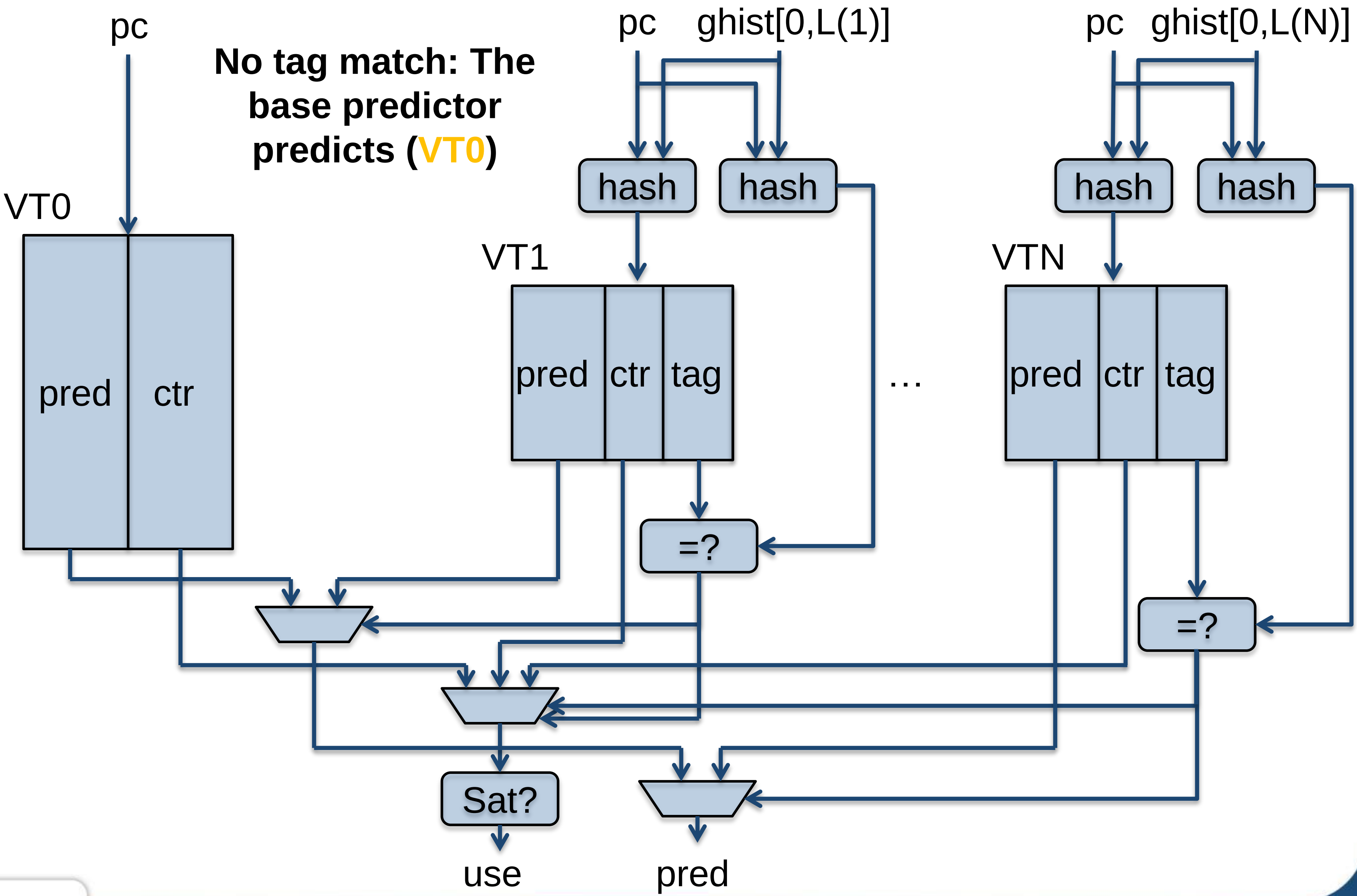
Predicting with VTAGE



The rightmost matching component predicts (VTN)

Predicting with VTAGE

No tag match: The base predictor predicts (VT0)



What VTAGE is Really About

- Context is available and easy to manage because it is **global**.

- The result of the previous instance is **not** required:
  - No speculative window.
  - No prediction critical loop: Back-to-back occurrences can be seamlessly predicted.

VTAGE is not Perfect

- VTAGE requires a lot of storage as a single prediction requires 64 bits (only a few bits for a branch prediction).

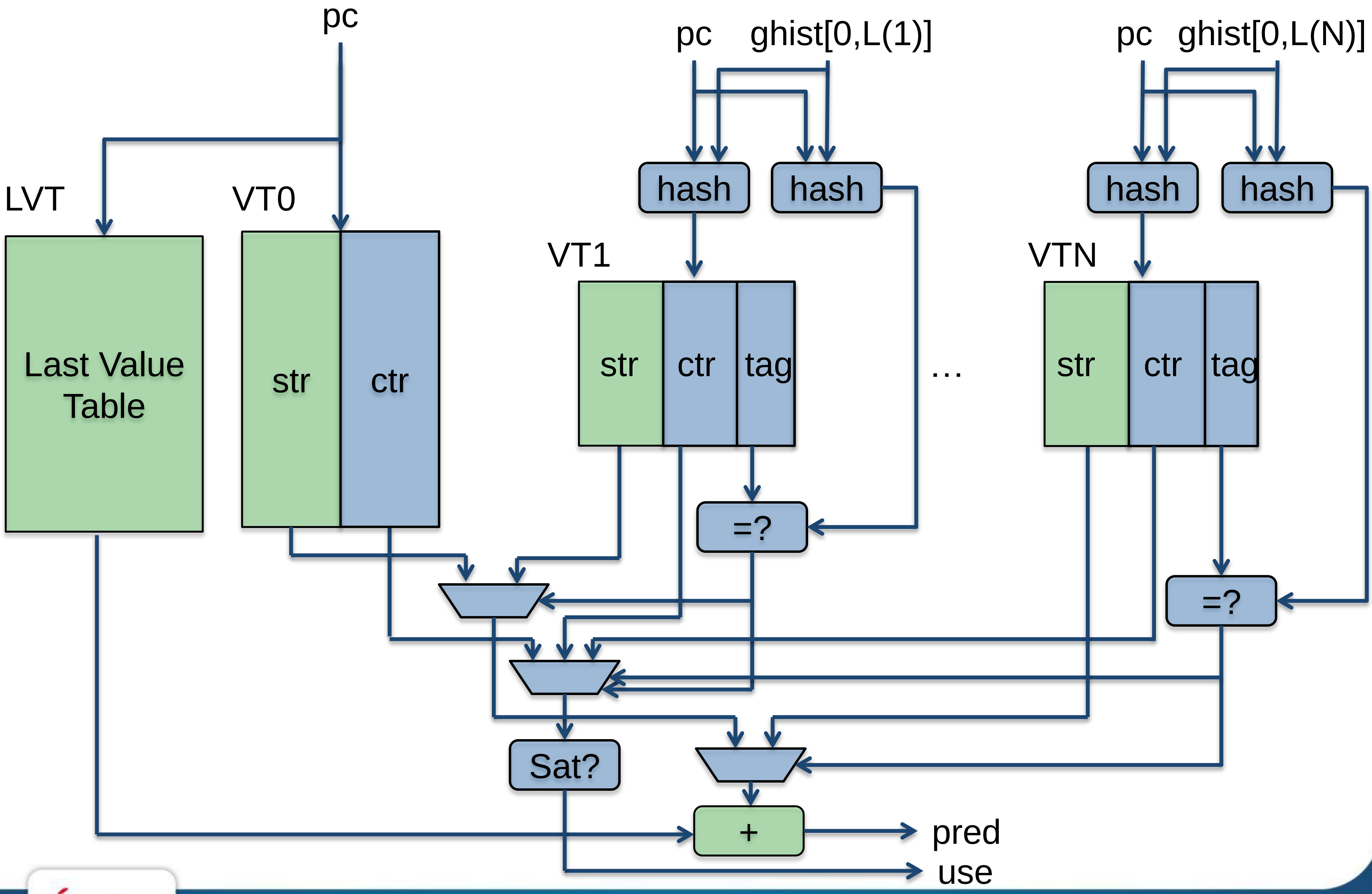
- VTAGE cannot handle **strided patterns** efficiently:
  - Each value in the pattern occupies its own entry (a single entry for the whole pattern in the Stride predictor).

Refining VTAGE: the Differential VTAGE Predictor

- Inspired by the D-FCM of [Goeman01]
  - Store **differences** between sequential results (strides) instead of full 64-bit values in the predictor.
  - Use a table to track last values *(Last Value Table).*

➢ Tightly coupled hybrid of VTAGE and Stride.

What is new in D-VTAGE



Refining VTAGE: the Differential VTAGE Predictor

- Pros:
  - Space efficient: strides can be small (8/16/32 bits).
  - Tightly coupled: **combines** the prediction schemes rather than just **selecting** between them.

- Cons: Needs the previous result.
  - Prediction critical loop (adder and multiplexer, like Stride).
  - A speculative window is needed*.

*A practical implementation is provided in Perais and Seznec, HPCA'15.

A

Revisiting Value Prediction

2. Complex Prediction Validation

Some Important Metrics

- Only confident predictions are actually used in the pipeline. We differentiate:
  - $$Coverage = \frac{Correct\ predictions}{Total\ dynamic\ predictable\ instructions}$$
  - $$Accuracy = \frac{Correct\ predictions}{Correct\ predictions +\ incorrect\ predictions}$$
- One is **meaningless** without the other.
- **Neither** is really conclusive regarding speedup.

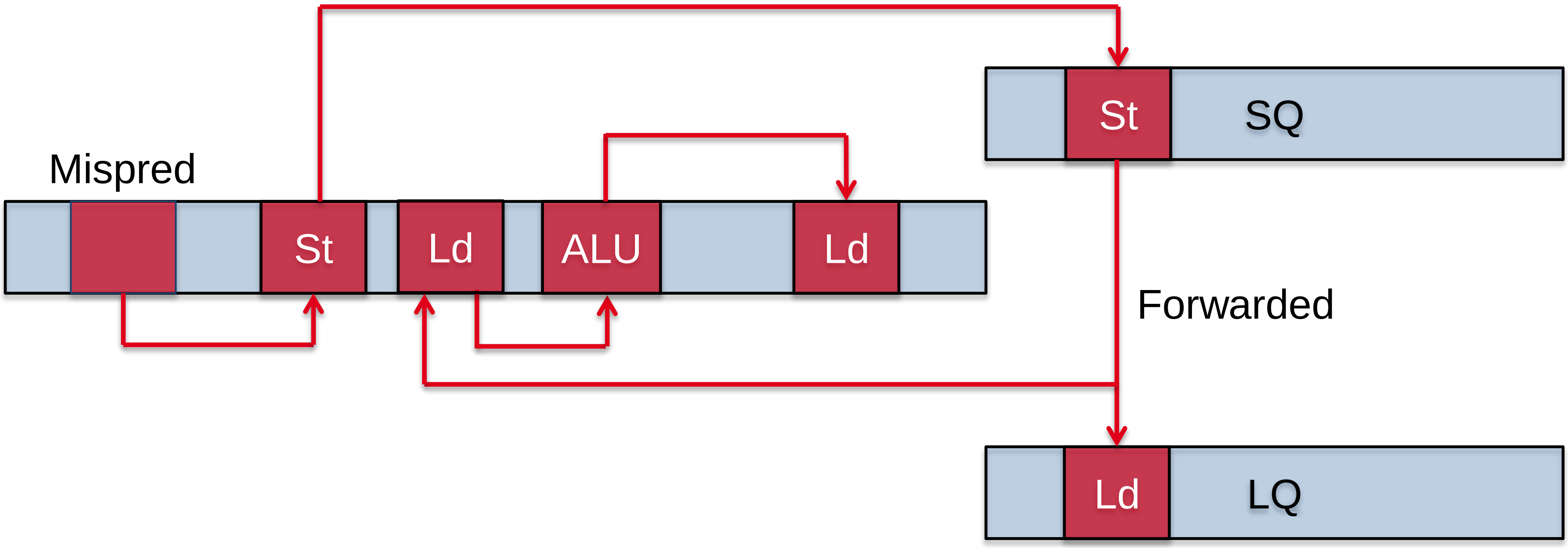
Validation and Recovery for Value Prediction

- Prediction validation is usually done **out-of-order**, at the output of the functional units.
  - Additional hardware.
  - Additional Register File pressure.

- Recovery:
  - Pipeline squashing, simple but slow (~20-30 cycles).
  - Selective replay (reissue), **very complex** but faster (few cycles).

Avoid Selective Replay if Possible

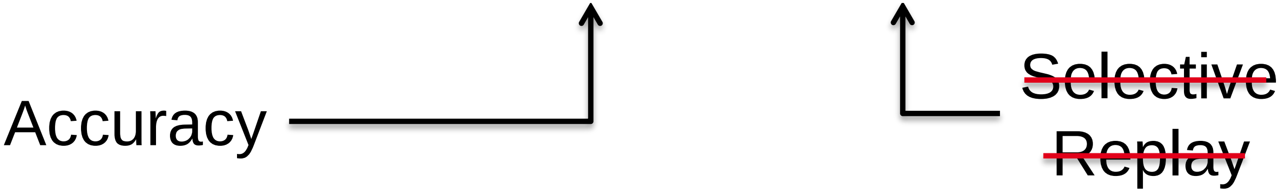
- **Arbitrarily** long dependency chain to replay.
- **Sequential replay**: wrong execution can still continue while replay catches up.



Getting Performance from Value Prediction

- The cycles lost due to **few** mispredictions may offset the cycles won thanks to **many** correct predictions.

$$Overall_{penalty} \approx \; N_{misp} * Misp_{penalty}$$



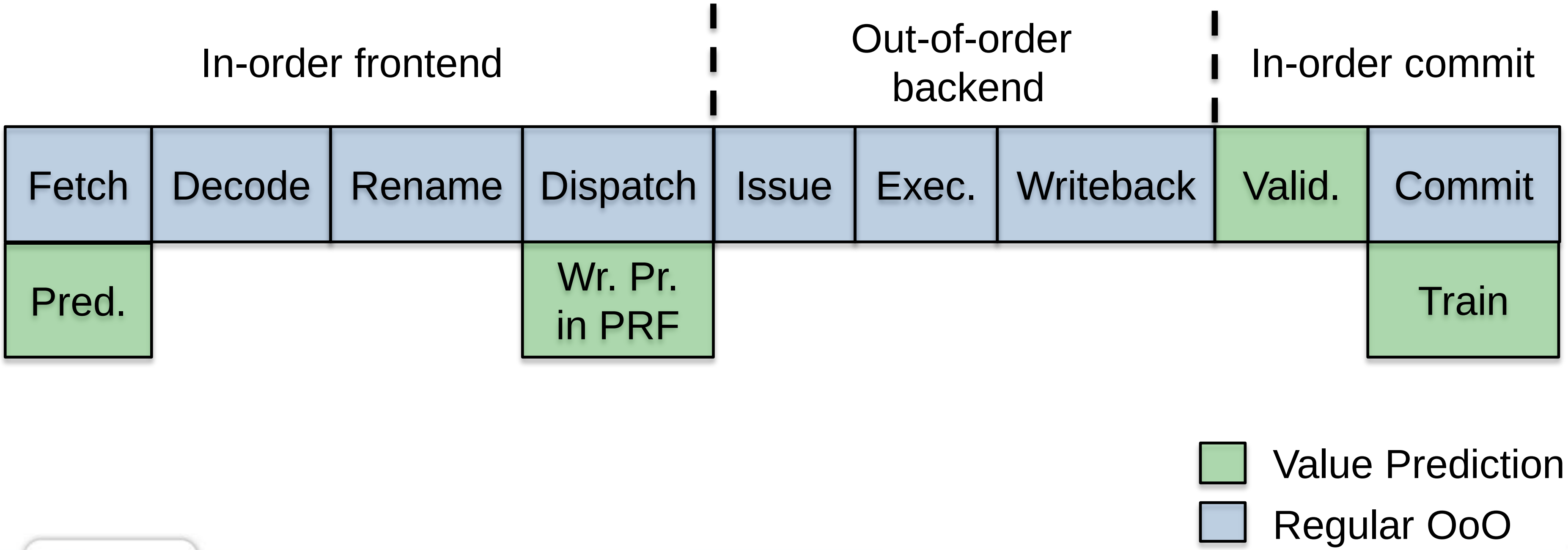
Accuracy

~~Selective Replay~~

➢ Focus on providing **very high accuracy** at reasonable cost in coverage.

Late Validation and Recovery

- Assuming high accuracy, the misprediction penalty is less important.
  - Validate and recover at commit time, **in-order**.



In-order frontend | Out-of-order backend | In-order commit

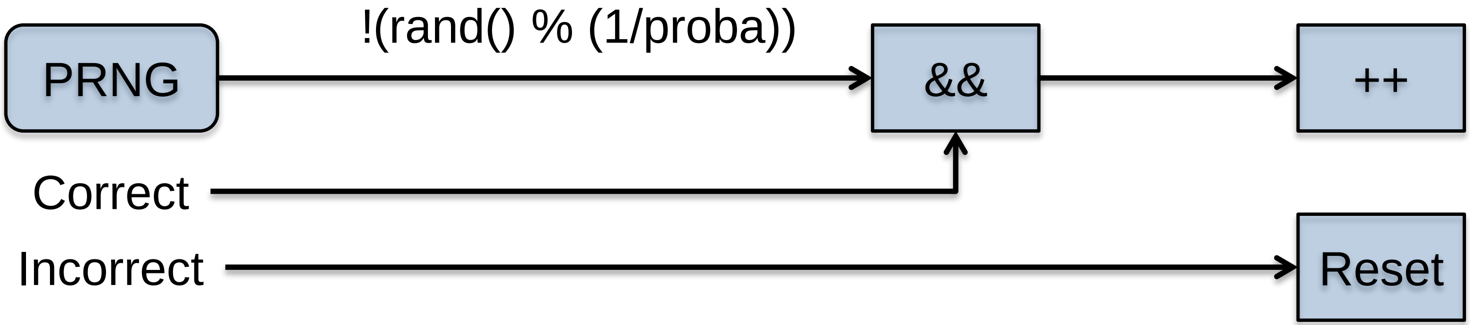
Fetch	Decode	Rename	Dispatch	Issue	Exec.	Writeback	Valid.	Commit
Pred.			Wr. Pr. in PRF					Train

🟩 Value Prediction

🟦 Regular OoO

Providing Very High Accuracy

- Wide (10-bit) saturating counters can do the trick but this takes area.
- Use 3-bit counters and a PRNG to control incrementing [Riley06]: *Forward Probabilistic Counters (FPC).*



$p_{selective} = \{1, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{16}, \frac{1}{16}\}$ $p_{squash} = \{1, \frac{1}{16}, \frac{1}{16}, \frac{1}{16}, \frac{1}{16}, \frac{1}{32}, \frac{1}{32}\}$

A

Revisiting Value Prediction

3. Evaluation

Simulator and Benchmarks

- Cycle-level simulator: gem5 (x86_64).
- 4GHz, 8-wide, 6-issue, 20 cycles Bmispred., 192ROB, 60IQ, 72LQ/42SQ. 32KB L1D/L1I, 1MB unified L2 with stride prefetcher, 4GB DDR3-1600.

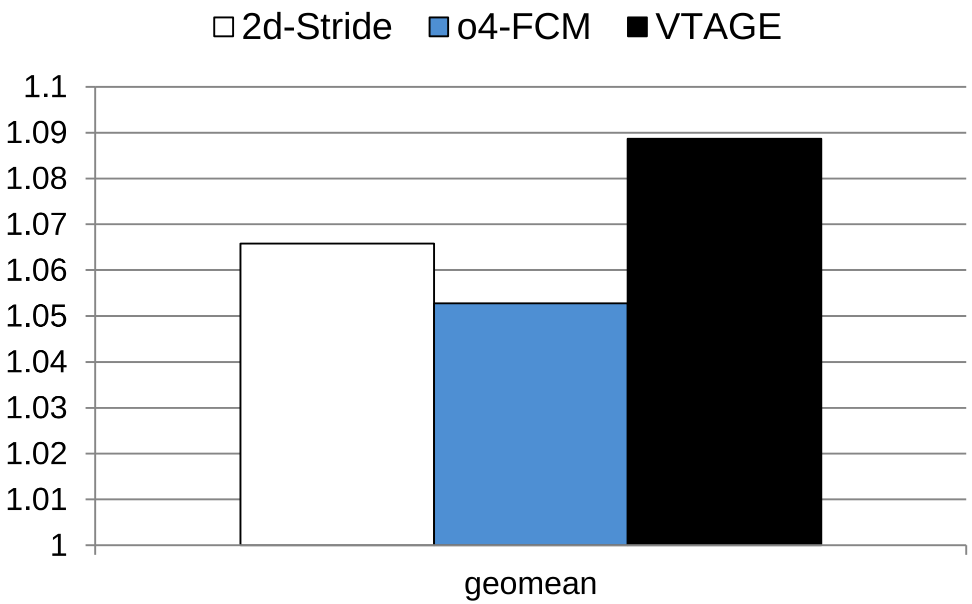
- Single-thread benchmarks: Subset of SPEC’00 and SPEC’06 (36 benchmarks).
- Simpoint: One slice per benchmark, warmup for 50Minsts, run for 100Minsts.

Predictors

- 8K-entry 2-delta Stride predictor [Eickemeyer93].
- 8K-entry VHT/8K-entry VPT order 4 FCM [Sazeides97].
- 6+1 component VTAGE (6 x 1K-entry + 8K-entry tagless LVP). History lengths : 2 to 64 bits.

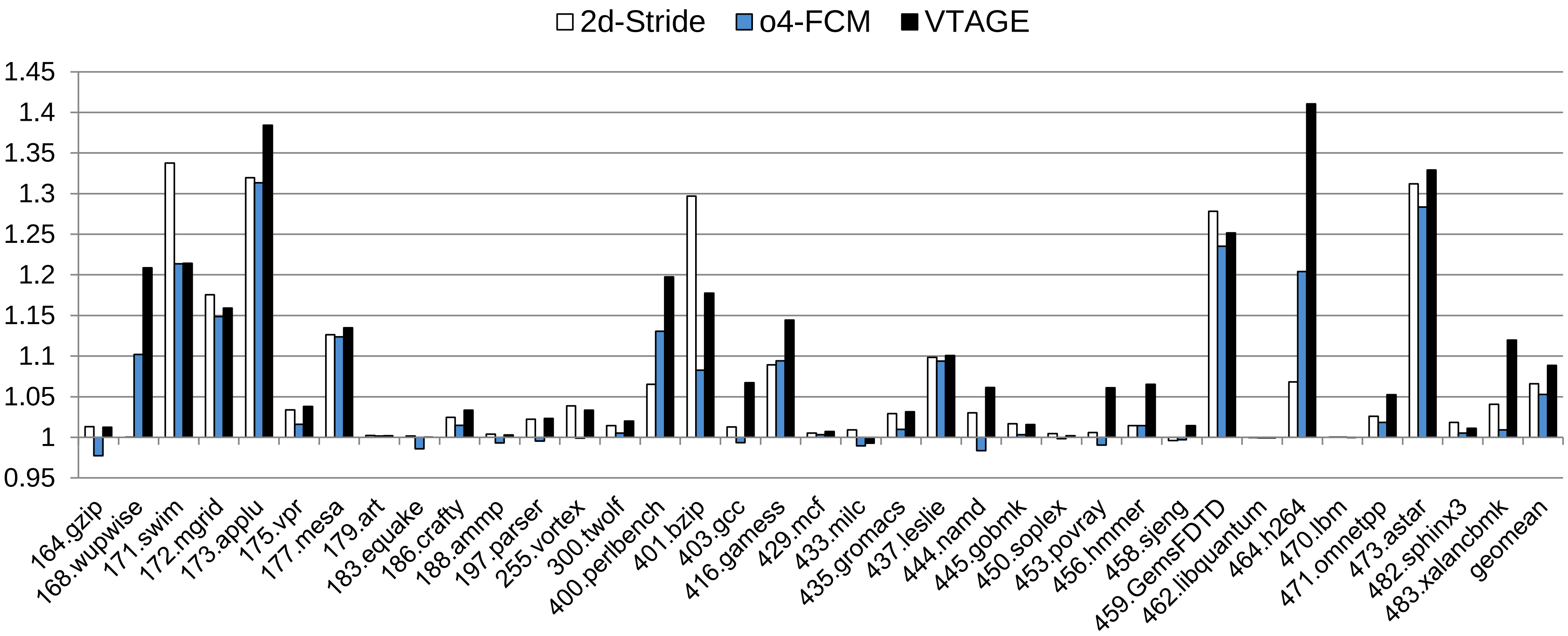
- Regular 3-bit saturating confidence counters/3-bit FPC.

Speedup – Ideal Selective Replay



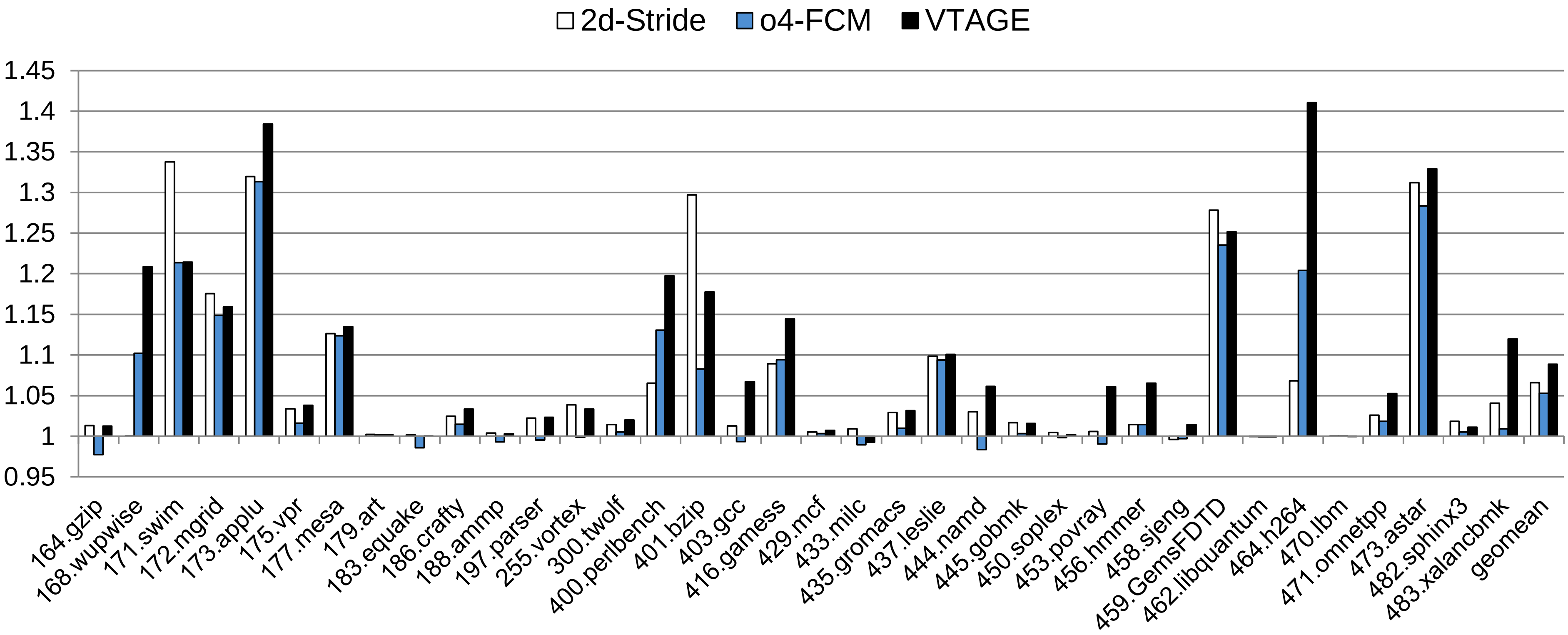
i. 5% to 9% average speedup.
ii. More than 10% speedup in 13 benchmarks out of 36.

Speedup – Ideal Selective Replay



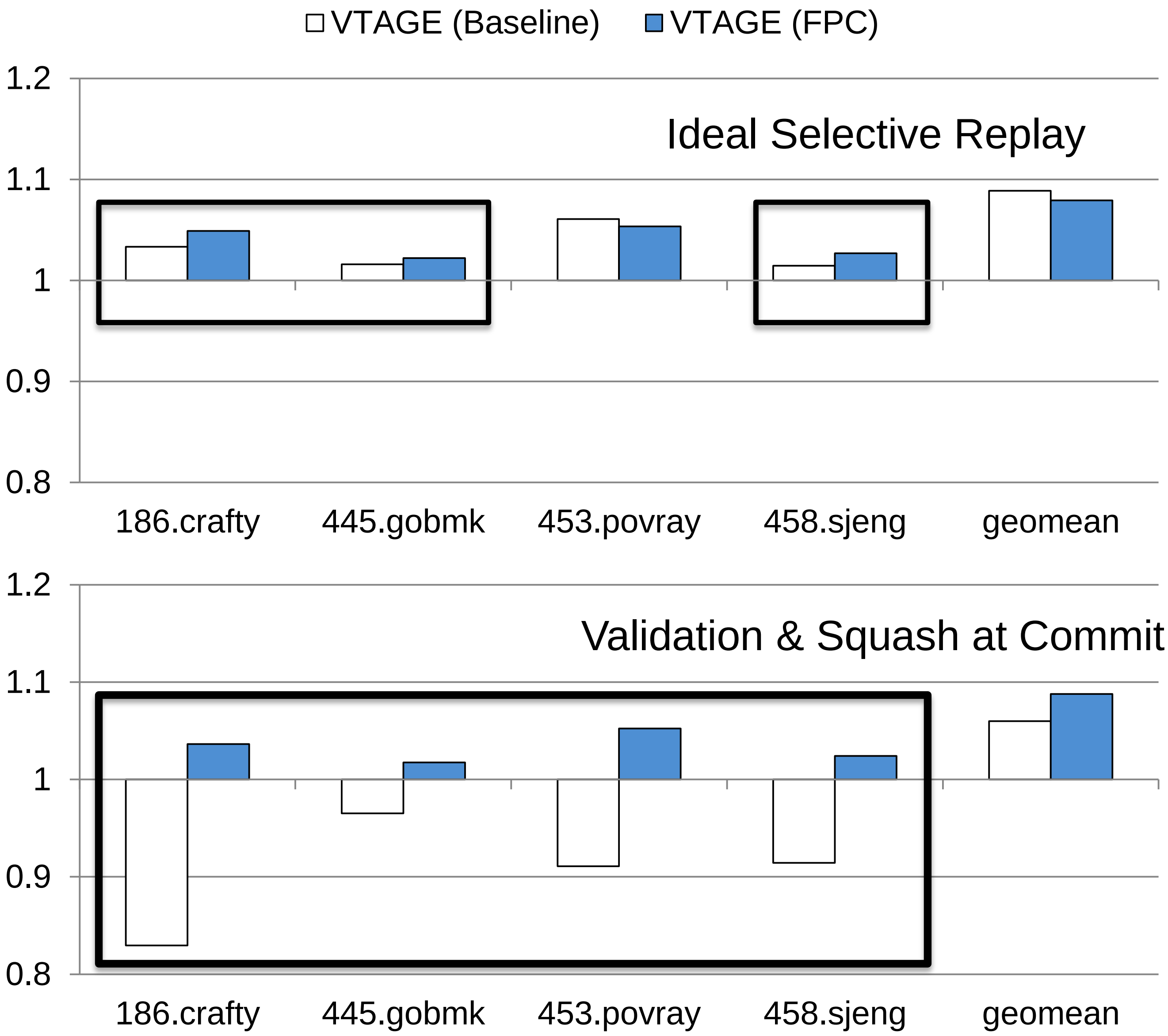
i. More than 10% speedup in 13 benchmarks out of 36.
ii. 5% to 9% average speedup.
iii. Some benchmarks favor the computational 2d-Stride.

Speedup – Ideal Selective Replay



i. More than 10% speedup in 13 benchmarks out of 36.
ii. 5% to 9% average speedup.
iii. Some benchmarks favor the computational 2d-Stride.
iv. Other favor context-based FCM and VTAGE.

Impact of the Recovery Mechanism



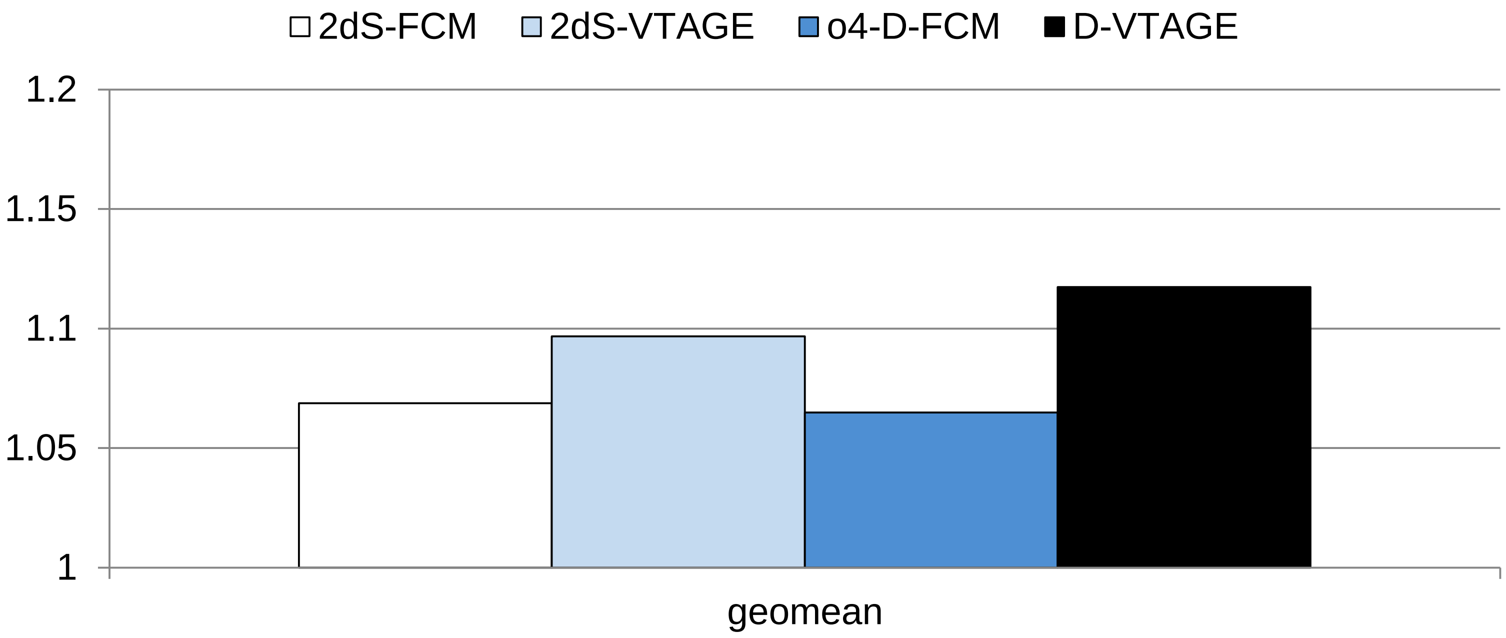
Impact of FPC on Coverage (Squash at Commit)

- No strong correlation between coverage and performance:
  - FPC costs around 10% coverage on average.
  - Yet performance generally increases.

- Also have to consider **accuracy**:
  - Regular counters are > 97%.
  - FPC are > **99.7%**

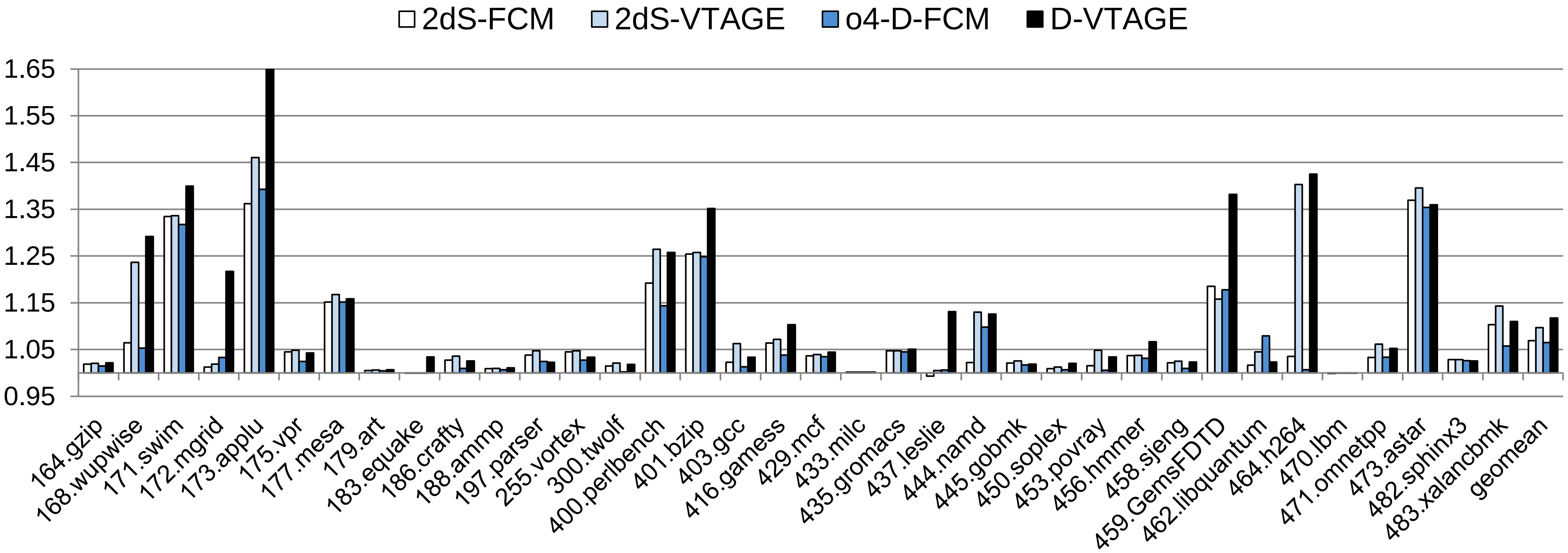
- We really need FPC to cost-effectively push accuracy very high so the cost of recovery can be absorbed.

D-VTAGE vs. Other Hybrids (Squash at Commit, FPC)



- 2dS-FCM, 2dS-VTAGE: Slightly better than the best of both components.
- D-FCM comparable to 2dS-FCM.

D-VTAGE vs. Other Hybrids (Squash at Commit)



- 2dS-FCM, 2dS-VTAGE: Slightly better than the best of both components.
- D-FCM comparable to 2dS-FCM.
- D-VTAGE comparable to 2dS-VTAGE but much better in a few cases.

Increasing Performance Through VP - Roadmap

A. Revisiting Value Prediction

1. ~~Existing Predictors and Their Shortcomings~~
2. ~~Complex Prediction Validation~~
3. Evaluation

D-VTAGE

Validation & Squash at commit.

B. Complexity Remains in the Register File

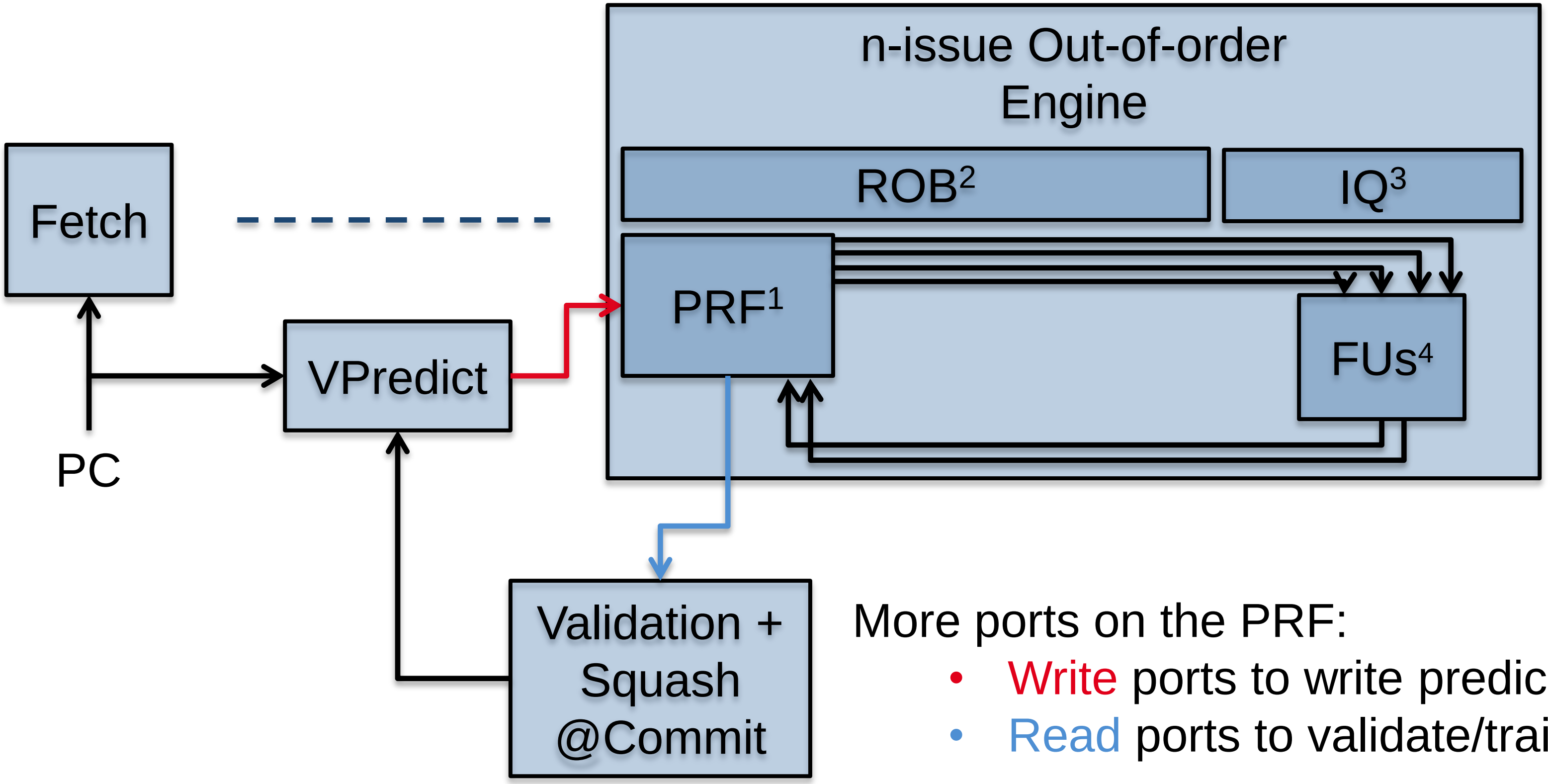
1. VP Requires Additional Ports on the Register File
2. Reducing the Execution Engine Complexity through Value Prediction
3. Evaluation

B

Complexity Remains in the Physical Register File

1. Value Prediction Requires additional Ports on the Register File

The – Slightly – Hidden Costs of VP



More ports on the PRF:

- Write ports to write predictions.
- Read ports to validate/train.

[1] **P**hysical **R**egister **F**ile
[2] **R**e**o**rder **B**uffer
[3] **I**nstruction **Q**ueue (Scheduler)
[4] **F**unctional **U**nits

Let us Count.

- Assume strict RISC-style µ-ops: 2 sources, 1 destination.
- Baseline 6-issue:
  - 12 read (R) ports, 6 write (W) ports.

- VP 8-wide, 6-issue:
  - 12R/6W for execution.
  - 8W to write 8 predictions/cycle in the Physical Register File (PRF).
  - 8R to validate/train 8 instructions/cycle.

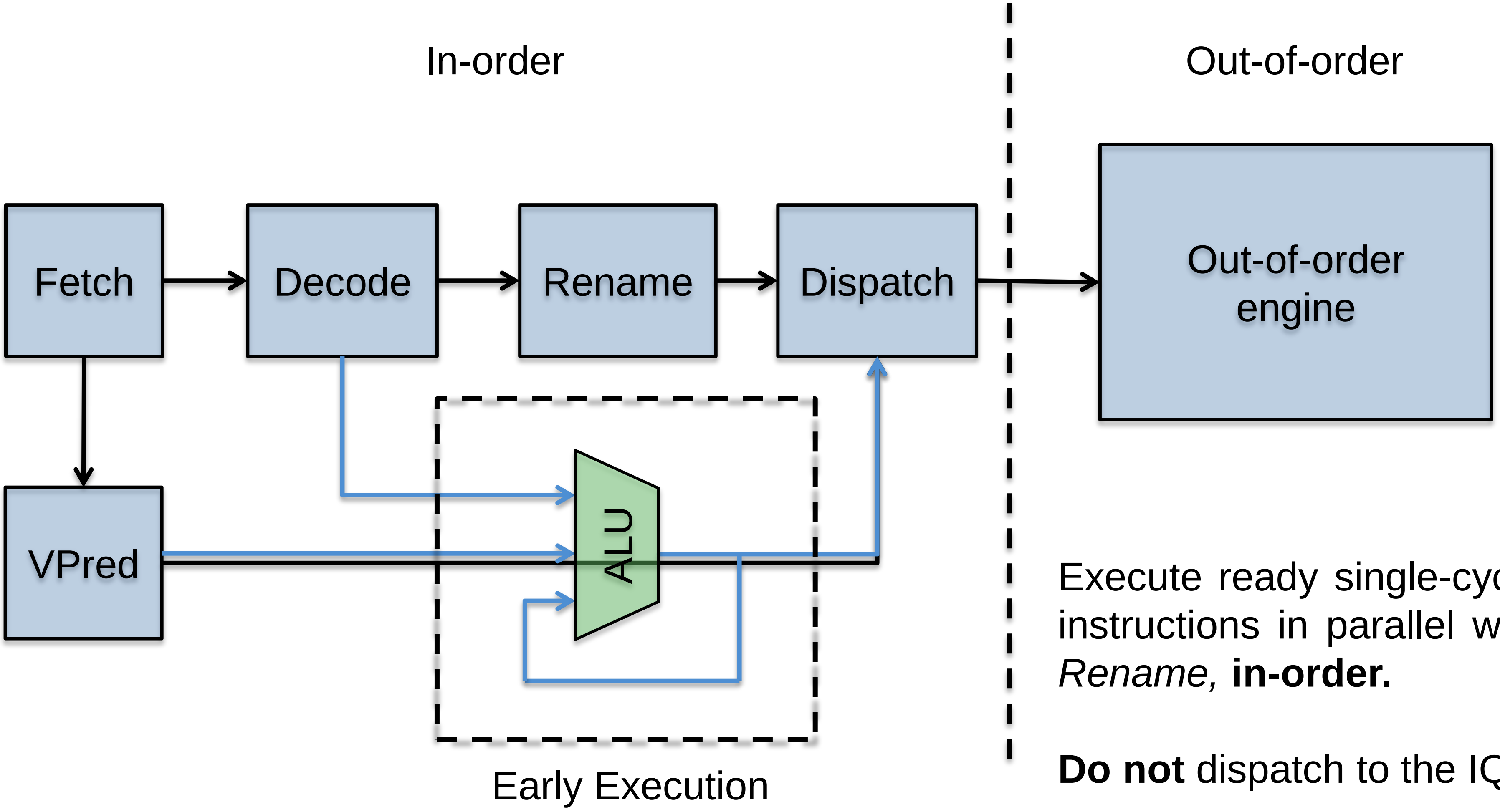
➢ **12R/6W** vs. **20R/14W**! PRF area and power proportional to number of ports **squared** [Zyuban1998].

Leveraging the – slightly – Hidden Benefits of VP

- Value Prediction provides:
  - Instructions with ready operands flowing from the value predictor.
  - Predicted instructions not needing to be executed before retirement.

➢ Offload execution to some other **in-order** parts of the core to reduce complexity in the **out-of-order** core. Save PRF ports in the process.

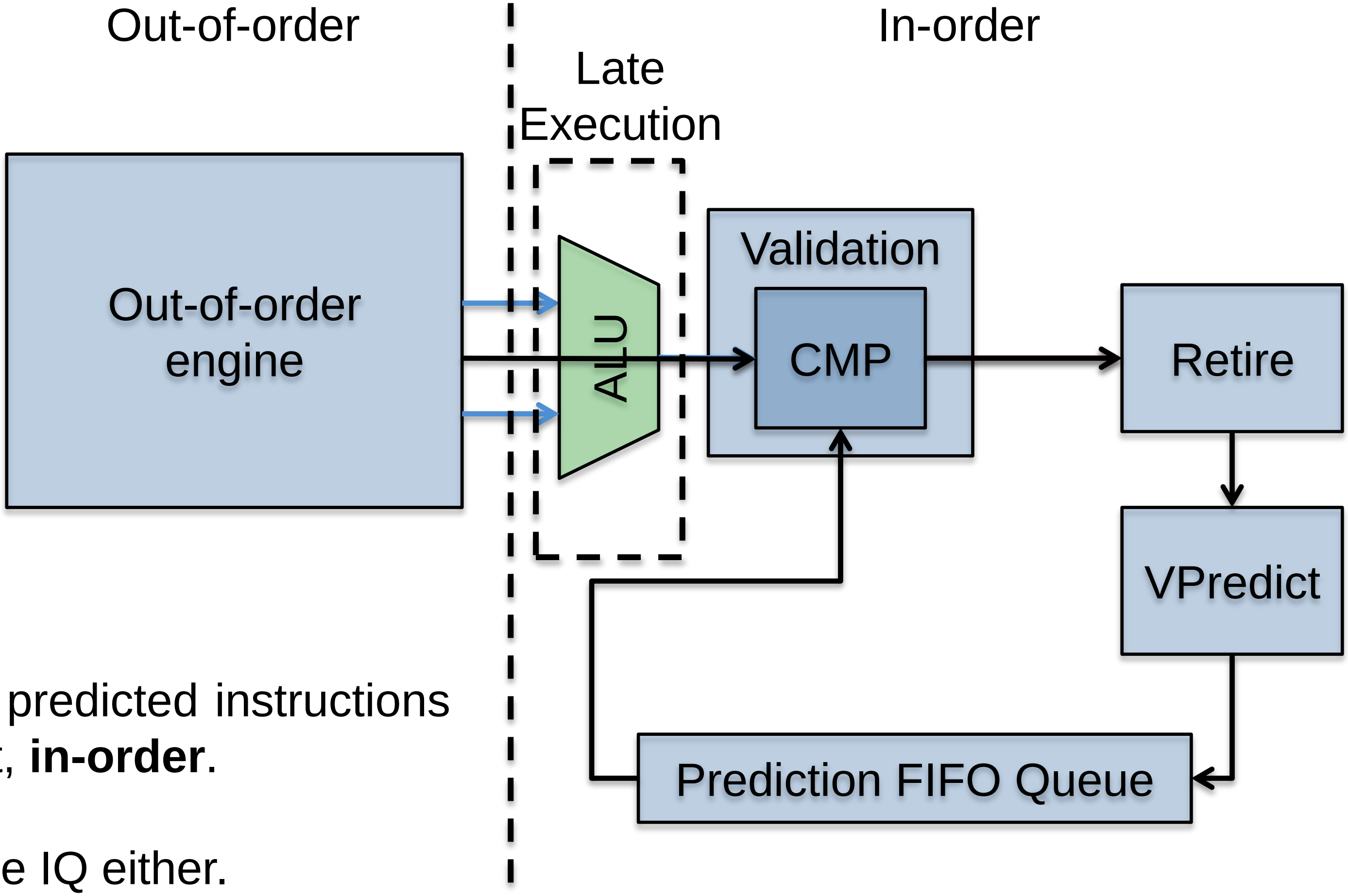
Introducing Early Execution



Execute ready single-cycle instructions in parallel with *Rename*, **in-order.**

Do not dispatch to the IQ.

Introducing Late Execution



Execute single-cycle predicted instructions just before retirement, **in-order**.

Do not dispatch to the IQ either.

{Early | OoO | Late} Execution: EOLE

- Much fewer instructions enter the IQ: We may be able to reduce the issue-width:
  - Less Wakeup logic.
  - Fewer ports on the PRF.
  - Less bypass logic.
  - ➢ **Faster** and **less power hungry** execution engine.

- What about hardware cost?

Hardware Cost of EOLE

- Early Execution:
  - A single rank of simple ALUs.
  - ***No** additional PRF ports**.***
  - Associated bypass network.

- Late Execution & Validation:
  - Rank of simple ALUs and comparators (to validate).
  - No bypass.
  - *n* read ports to validate becomes *2n* to handle *n* instructions per cycle: **16R** for an 8-wide pipeline.

➢ **28R/14W**! Only **12R/6W** for the baseline…

B

Complexity Remains in the Physical Register File

2. Reducing the Execution Engine Complexity through Value Prediction

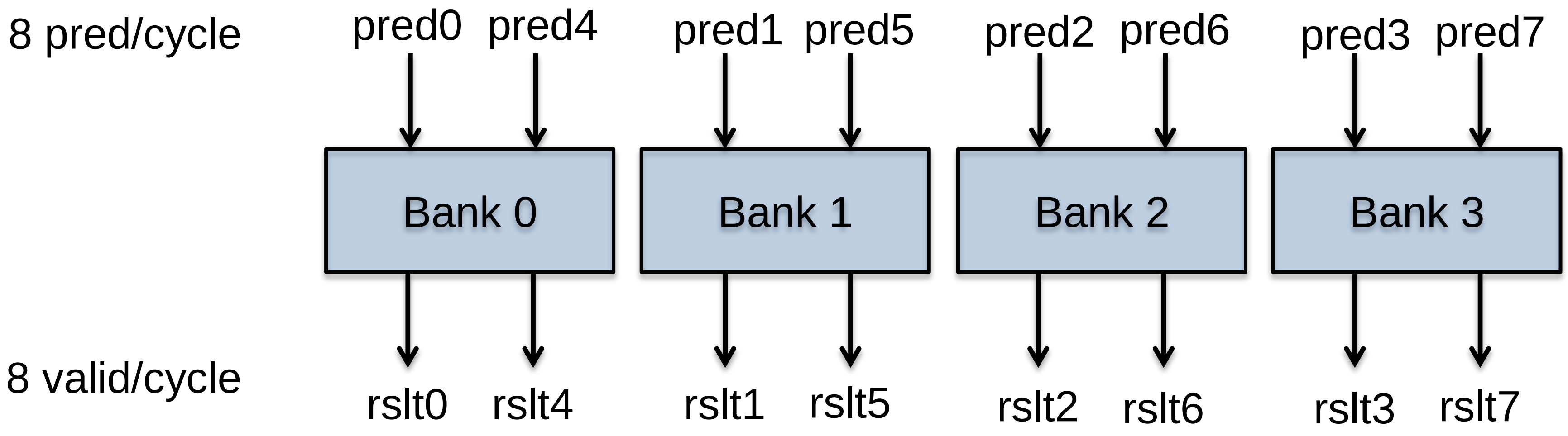
Reducing the Issue Width

- If fewer instructions enter the IQ, then we can reduce the issue width:
  - From 6 to 4 (-4R and -2W): **24R/12W**.

➢ Reduces complexity in the execution engine, but still **too many** ports on the PRF.

Banking the Physical Register File

- Prediction and Validation are done in-order.
  - ➢ Bank the PRF and attribute predictions to consecutive banks.



- ➢ 2 write ports per bank instead of 8 for a 4-bank file.
- ➢ 2 read ports per bank?

Read Port Sharing

- 8 instructions can be validated with 2R per bank…
- …but Late Execution still needs 16R per-bank to process 8 instructions.

- Fortunately, not all instructions are predictable (e.g., *stores)* or late-executable (e.g., *loads*).

➢ Constrain the number of read ports and share them: 4R per-bank is a good tradeoff.

Let us Count, Second Take

- 4-issue out-of-order engine (4W/8R per bank).
- 8 predictions per cycle (2W per bank).
- Constrained late-execution/validation (4R per bank).

➢ 12R/6W per bank in total.

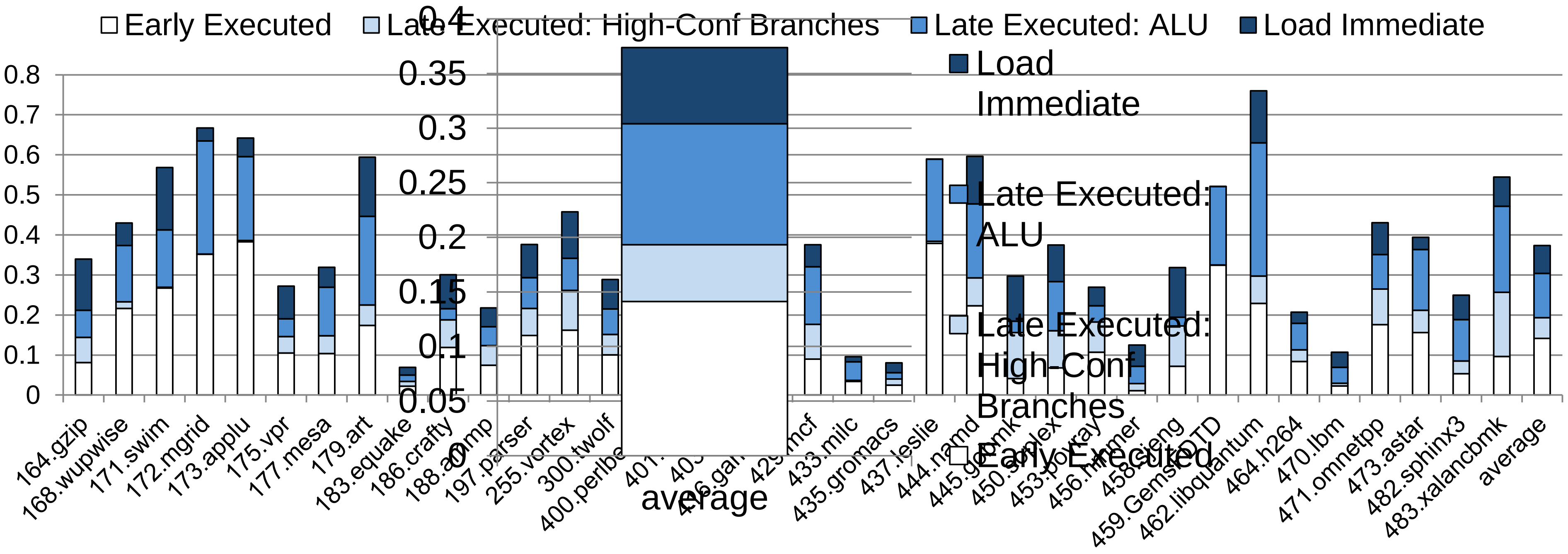
- From **28R/14W**, we now only need **12R/6W**! This is the same amount as the PRF **without VP**, except issue width is **33%** smaller.

B

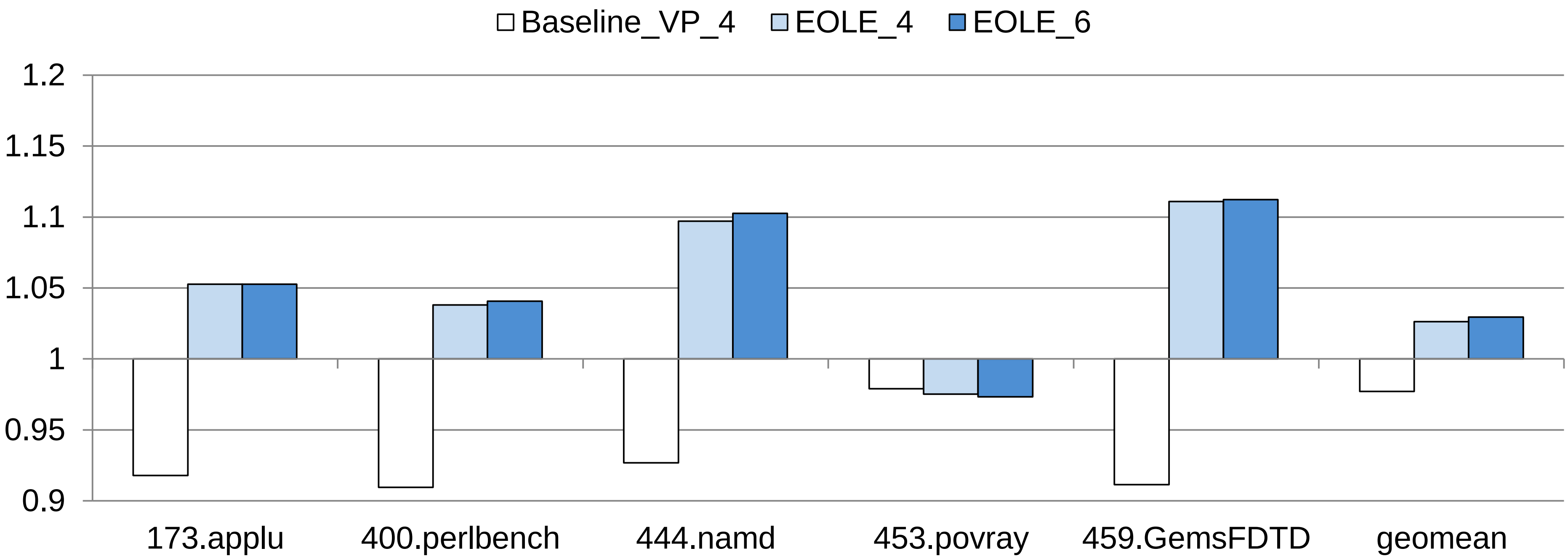
Complexity Remains in the Physical Register File

3. Evaluation

Early Executed – Late Executed

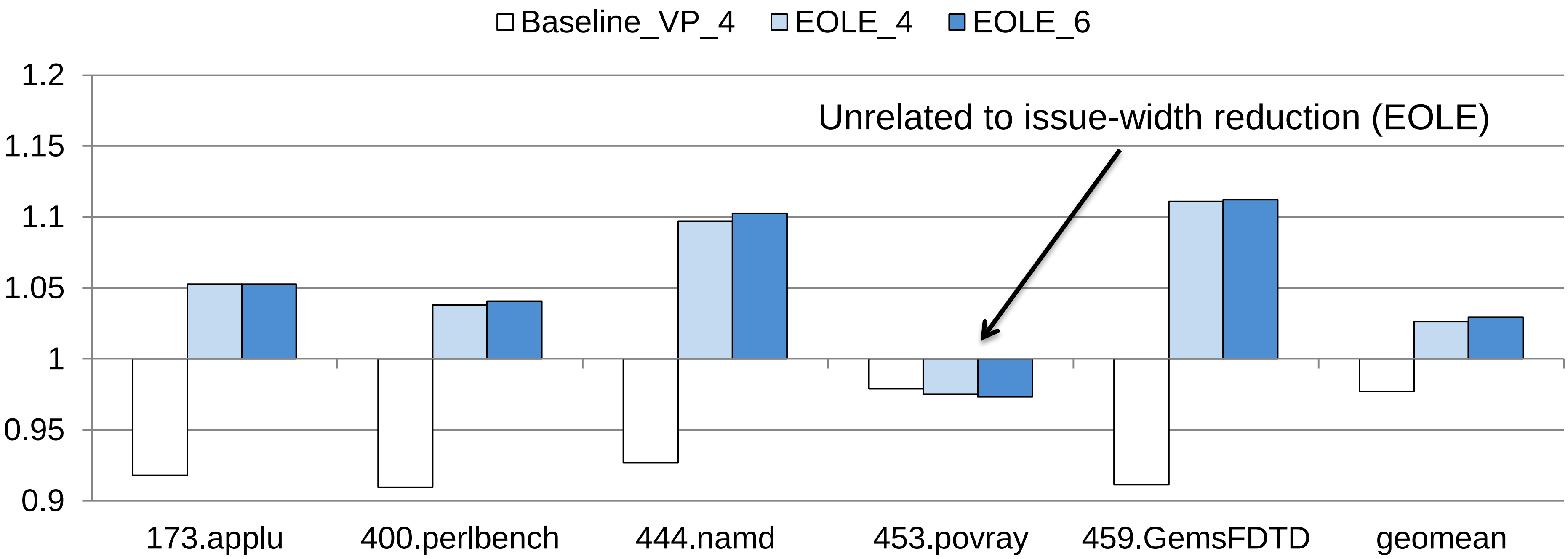


Reducing the Issue Width



- Decreasing the issue-width to 4 without compensating for the lost compute capability has noticeable impact on performance.

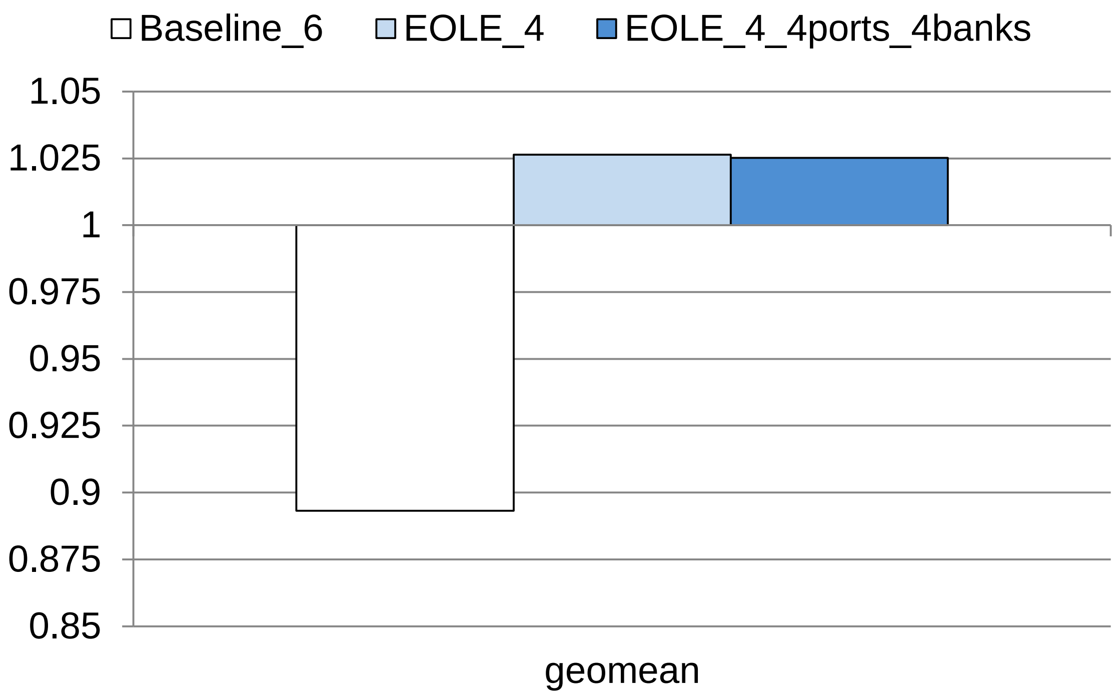
Reducing the Issue Width



Unrelated to issue-width reduction (EOLE)

- Performance is at worst comparable with the 6-issue VP pipeline for EOLE_4.
- The additional compute capability yields a small additional speedup in a few benchmarks.

Limited Issue and PRF Ports



- Constrained Late Execution/Validation provides the same performance as the unconstrained version.
- Value Prediction with **as many PRF ports as the baseline**, and a **simpler** execution engine.

Increasing Performance Through VP - Roadmap

A. Revisiting Value Prediction

1. ~~Existing Predictors and Their Shortcomings~~
2. ~~Complex Prediction Validation~~
3. Evaluation

D-VTAGE

Validation & Squash at commit.

B. Complexity Remains in the Register File

1. ~~VP Requires Additional Ports on the PRF~~
2. ~~Reducing the Execution Engine Complexity through Value Prediction~~
3. Evaluation

EOLE

Increasing Performance Through VP - Roadmap

A. Revisiting Value Prediction
   1. ~~Existing Predictors and Their Shortcomings~~
   2. ~~Complex Prediction Validation~~
   3. Evaluation

D-VTAGE

Validation & Squash at commit.

B. Complexity Remains in the Register File
   1. ~~VP Requires Additional Ports on the PRF~~
   2. ~~Reducing the Execution Engine Complexity through Value Prediction~~
   3. Evaluation

EOLE

C. Practical Value Prediction Infrastructure
   1. ~~n-ported predictor to predict n instructions/cycle~~
   2. ~~D-VTAGE requires a speculative window~~

Perais and Seznec, HPCA’15

Almost Done

Conclusion

Value Prediction In a Processor?

- D-VTAGE: tightly-coupled hybrid having **good performance with reasonable budget** thanks mostly to partial strides (Perais & Seznec, HPCA'14/'15).

- Validation and Squash at commit: **no overhaul of the OoO engine** (Perais & Seznec, HPCA'14).

- EOLE: **avoid additional ports** on the PRF for VP, **reduce the complexity** of the OoO engine (Perais & Seznec, ISCA'14, IEEE MICRO's TP'14).

Future Work

- Better predictors are always required. Especially if they can predict more long-latency instructions (e.g., cache misses).

- Interactions of VP with other mechanisms (“TAGE à tous les étages’’):
  - i. Predicting addresses looks like prefetching.
  - ii. It also looks like memory dependency prediction.

Unrelated Work

- Perais, Seznec, Michaud, Sembrant & Hagersten, ISCA’15. Improve the speculative scheduling of load dependents in the presence of a banked L1 cache.

- Sembrant, Hagersten, Black-Schaffer, Carlson, Perais, Seznec & Michaud, MICRO’15. Park non-critical instructions before they are inserted in the scheduler to improve MLP.

EOF