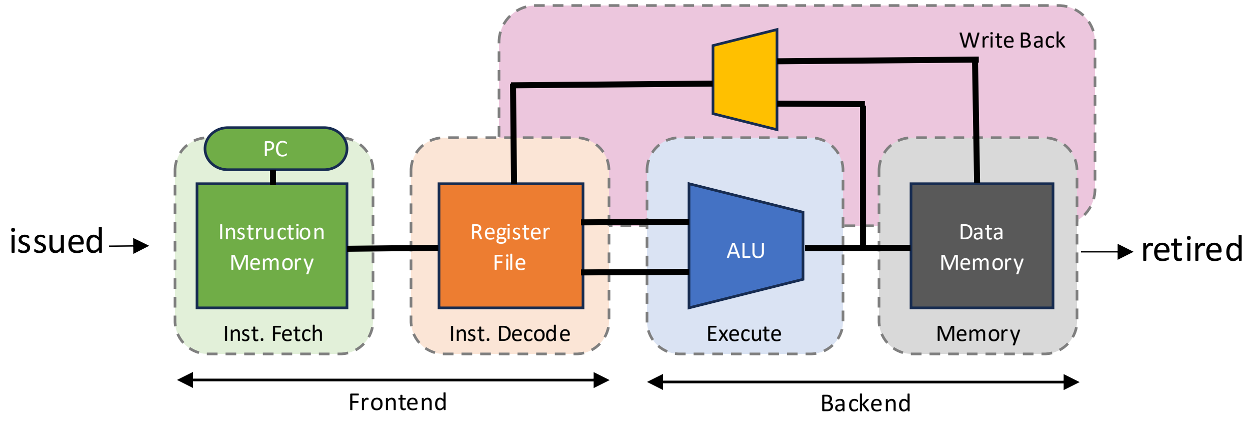
UDP: Utility-Driven Fetch Directed Instruction Prefetching

Presented in ISCA 2024

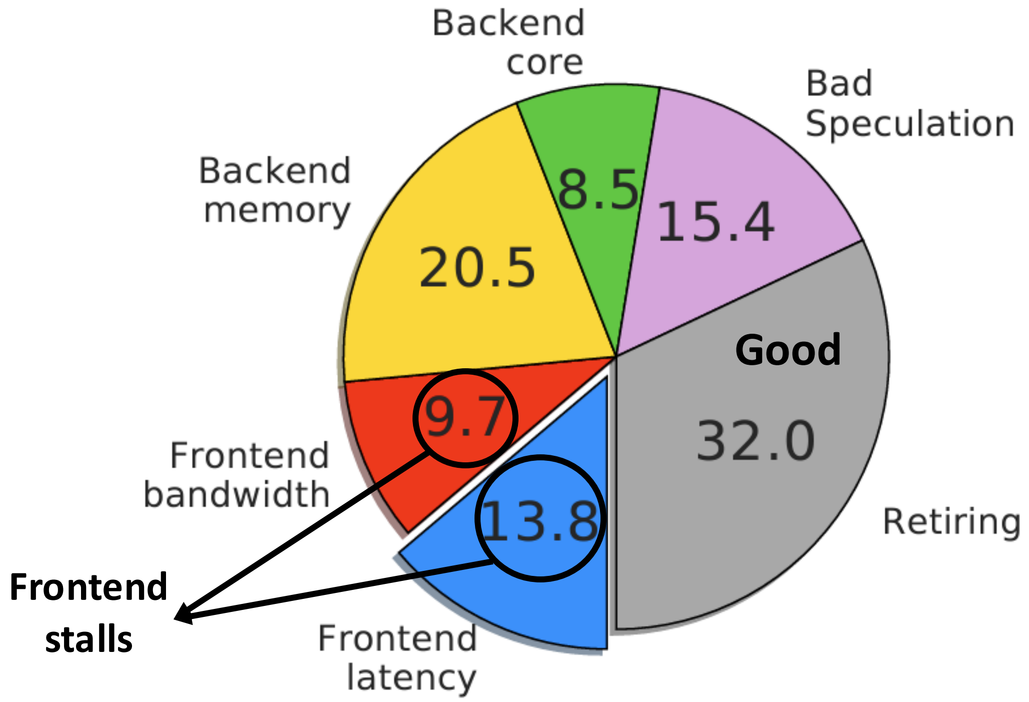
Surim Oh*, Mingsheng Xu*, Tanvir Ahmed Khan**, Baris Kasikci***, Heiner Litz*

*University of California Santa Cruz
**Columbia University
***University of Washington

The large instruction footprint responsible for a quarter of pipeline stalls!



How many stalls
spent in each part?



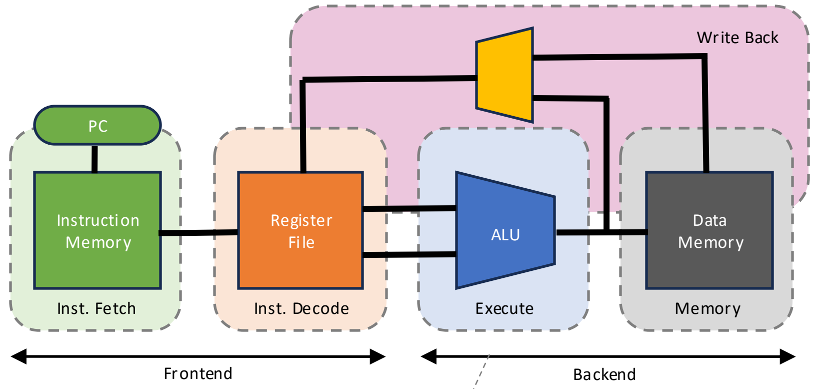
CPU Performance of Google Web Search

1) Processors in Google's fleet spend almost a quarter of their cycles due to frontend stalls.

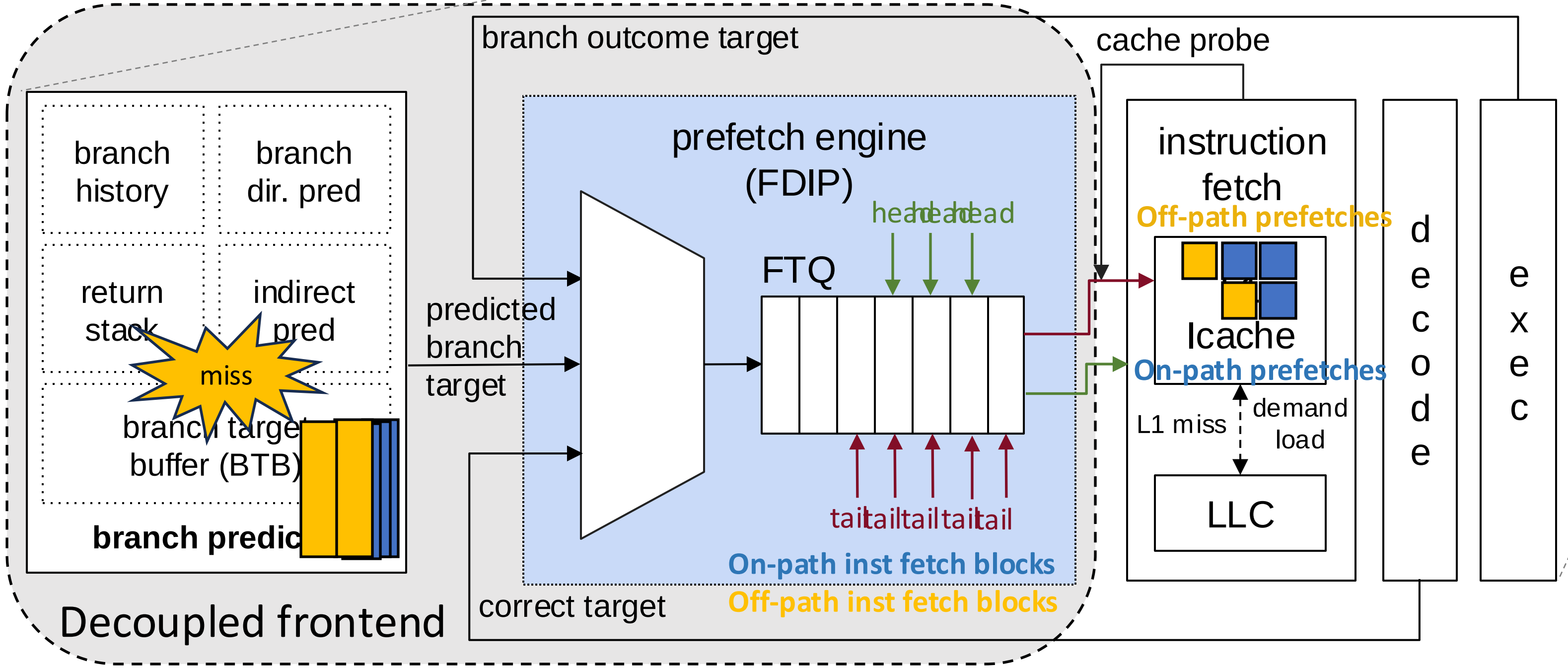
2) Instruction footprint >>>> Instruction cache

[G. Ayers et al. 2019]

Zoom into Today's Frontend



Issuing off-path prefetches on a BTB miss → Icache pollution with unuseful cache lines

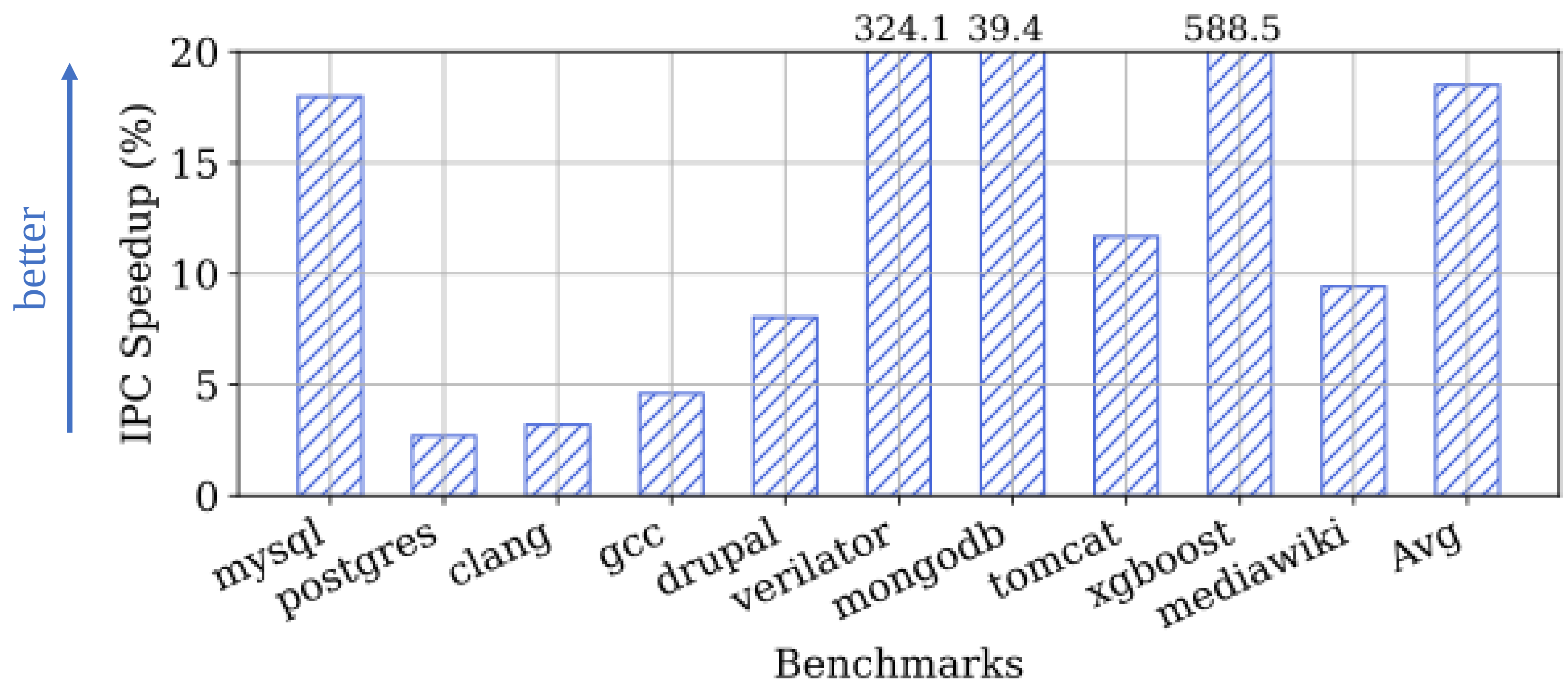


Frontend stalls

FDIP (Fetch-Directed Instruction Prefetching) a hardware prefetching technique leveraging the branch predictor to prefetch instructions

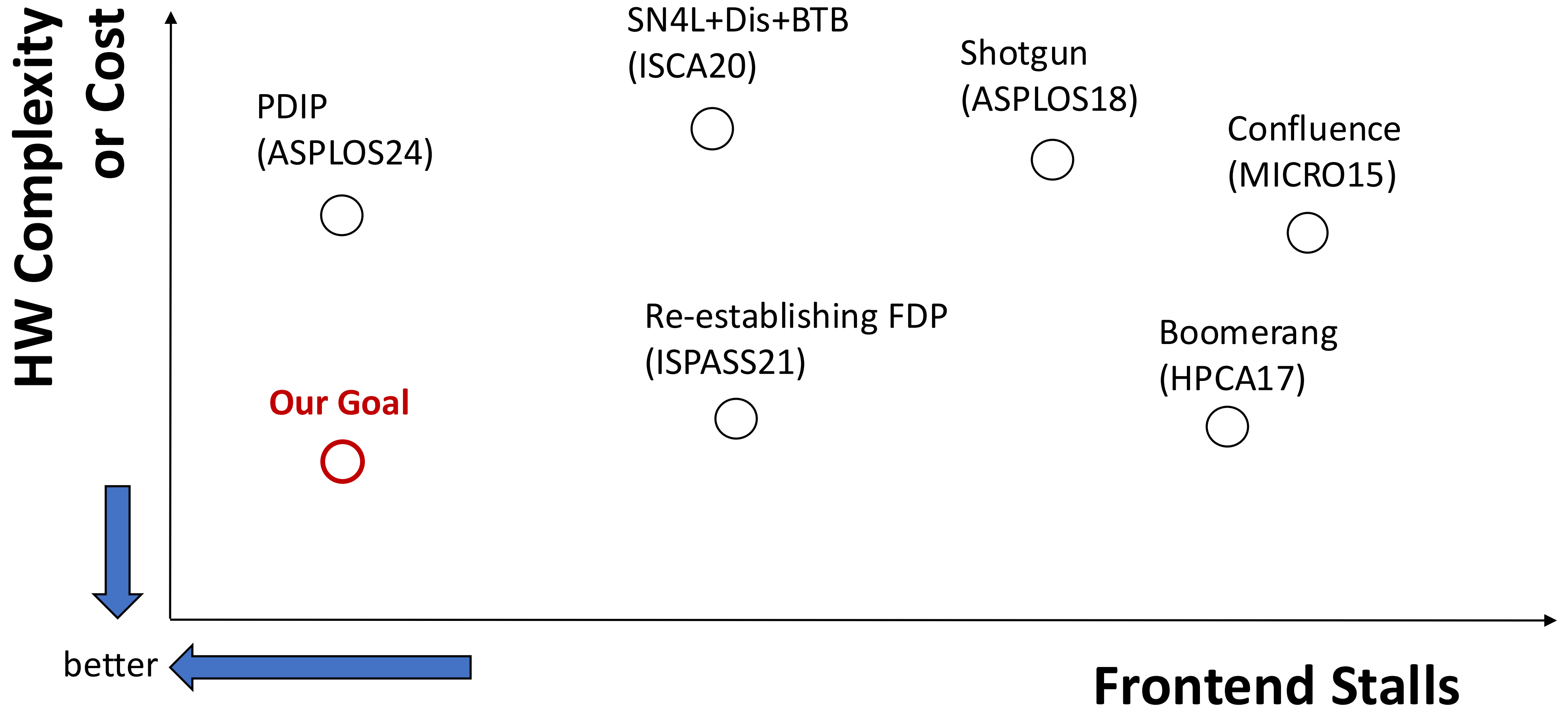
FDIP Falls Short of a Perfect Instruction Cache

Large code footprint $\rightarrow$ Many branches exceeding the BTB size $\rightarrow$ BTB misses $\rightarrow$ Limit the accuracy of FDIP



Ideal icache's speedup over today's FDIP

Related Work and Our Goal



Research Problems

- **Why** does FDIP fail to eliminate frontend stalls?

  A detailed analysis of the state-of-the-art FDIP

- **How** can we leverage insights to reduce frontend stalls?

  UFTQ: Dynamically adapts the FTQ size

  UDP: Prefetches only useful instructions

Introduction

- Optimizing data center applications

Analysis

- Why FDIP falls short of the ideal?

UFTQ

- Dynamically adapting FTQ size

UDP

- Utility-Driven Prefetching

Conclusion

- Utility study reduces cache pollution and improves timeliness.

Future Works

Workload Study

- 10 Data center applications

- Application-specific 10M instruction simpoints
- Aggregated simpoints based on their weights
- Warmed up with 10M instructions

MySQL PostgreSQL

MySQL, PostgreSQL:
`sysbench` OLTP-like database benchmark

GCC

Clang, GCC:
Building SPEC CPU 2017

Drupal™

HHVM OSS-perf benchmark

VERILATOR

Verilog simulator

mongoDB®

mongo-perf benchmark

Apache Tomcat

MediaWiki

dmlc XGBoost

Gradient Boosting Library

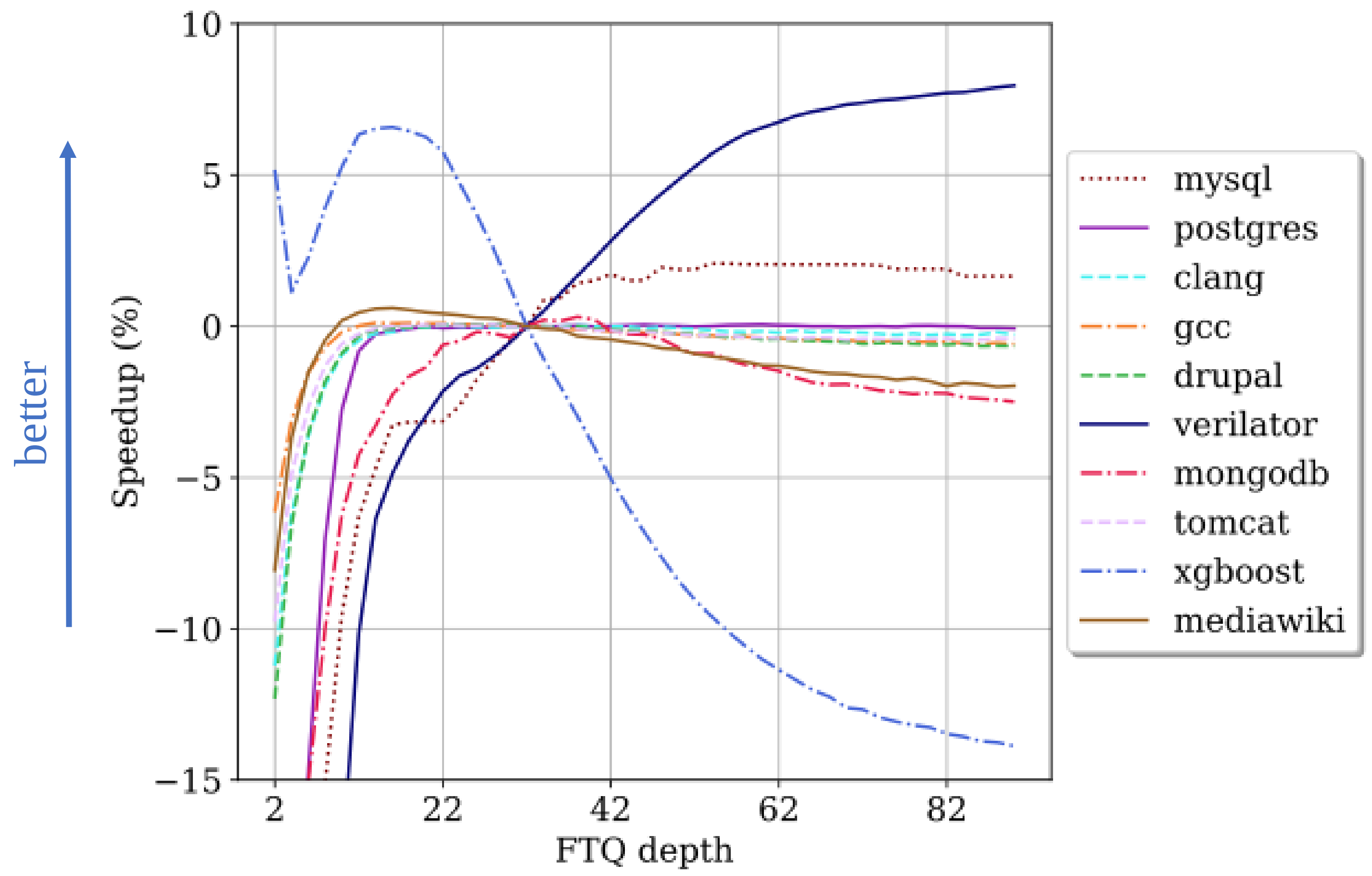
Simulation Environment

- Open-source, cycle-accurate Scarab simulator

Hardware Parameter	Value
CPU	Sunny-Cove-like
Frontend width and retirement	6-way
Functional Units	4 ALU, 2 Load, 2 Store
Branch Predictor	TAGE-SC-L
Branch Target Buffer (BTB)	8K entries
Instruction Prefetcher	**FDIP**
Data Prefetcher	Stream
L1 instruction cache	32 KiB, 8-way, 3 cycles
L1 data cache	48 KiB, 12-way, 4 cycles
L2 unified cache	512 KiB, 8-way, 13 cycles
LLC unified cache	Shared 2MiB/core, 16-way, 36 cycles

Decoupled Frontend Parameter	Value
FTQ blocks per cycle	2
FTQ block size	32 Bytes

Analysis: Optimal Run-ahead Distance (FTQ depth)



Large FTQ depth:

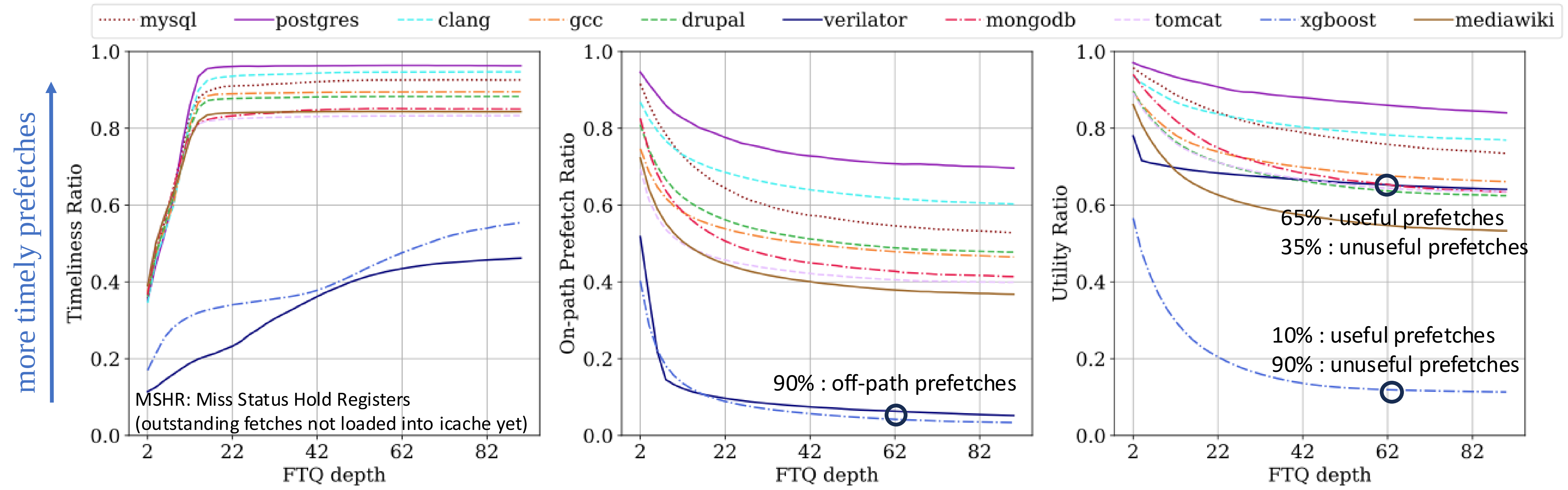
- Improves prefetch timeliness
- Increases off-path prefetches

Analysis: Timeliness, Off-path vs. On-path, Usefulness

$$Timeliness\ Ratio = \frac{icache_{pref.hits}}{icache_{pref.hits} + MSHR_{pref.hits}}$$

$$Onpath\ Prefetch\ Ratio = \frac{Prefs_{on}}{Prefs_{on} + Prefs_{off}}$$

$$Utility\ Ratio = \frac{Prefs_{useful}}{Prefs_{useful} + Prefs_{unuseful}}$$



Applications require different FTQ depth to achieve timeliness

More off-path prefetches with larger FTQ depth

Limiting off-path prefetches through bandwidth throttling and FTQ depth is insufficient.

Analysis: Usefulness of Off-path Prefetches

```
1   if (cond) a++;
2   else a--;
3   b += a; //Merge Point
```

Line 3 is a useful off-path candidate after a merge point.

Off-path prefetches are

Very Useful

VERILATOR

Somewhat useful/harmful

MySQL™

Apache Tomcat

MediaWiki

PostgreSQL

clang

Drupal™

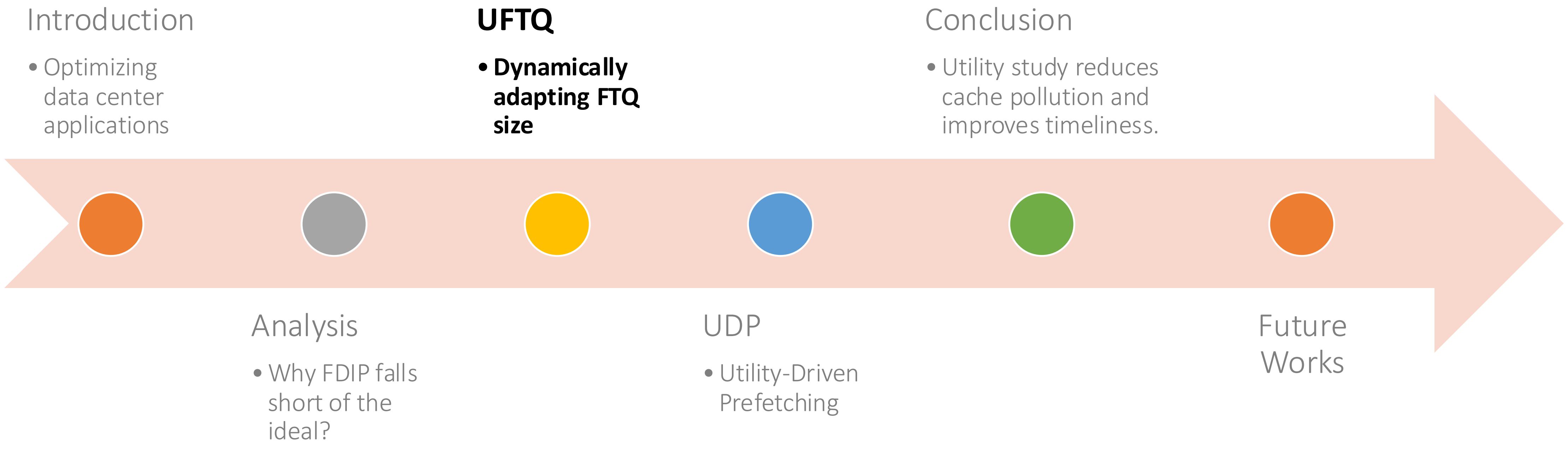
GCC

Harmful

mongoDB®

dmlc XGBoost

No one-size-fits-all FTQ ?



Introduction

- Optimizing data center applications

Analysis

- Why FDIP falls short of the ideal?

UFTQ

- **Dynamically adapting FTQ size**

UDP

- Utility-Driven Prefetching

Conclusion

- Utility study reduces cache pollution and improves timeliness.

Future Works

UFTQ : Application-specific FTQ Size

$$Utility\ Ratio = \frac{Prefs_{useful}}{Prefs_{useful} + Prefs_{unuseful}}$$

$$Timeliness\ Ratio = \frac{icache_{pref.hits}}{icache_{pref.hits} + MSHR_{pref.hits}}$$

Dynamically adjusting the FTQ size for a given workload by leveraging utility ratio and timeliness ratio

No one-size-fits-all FTQ

Application	Optimal FTQ	Utility	Timeliness
mysql	56	0.77	0.93
postgres	76	0.85	0.96
clang	54	0.79	0.95
gcc	60	0.72	0.93
drupal	28	0.64	0.85
verilator	84	0.64	0.46
mongodb	38	0.69	0.85
tomcat	24	0.69	0.82
xgboost	12	0.30	0.31
mediawiki	18	0.62	0.83
Average (Geomean)	42	**0.65**	**0.75**
Correlation Coefficient	-	**0.63**	**0.21**

- **UFTQ-AUR**: find Queue Depth satisfying AUR (QDAUR) and adjust FTQ size

1) start from Average FTQ (39)
2) measure Utility Ratio of the next 1000 prefetches
3) if Utility Ratio < AUR, reduce FTQ size

- **UFTQ-ATR**: find Queue Depth satisfying ATR (QDATR) and adjust FTQ size

- **UFTQ-ATR-AUR**: find QDAUR then QDATR

- Trained on 80% of randomly selected simpoints $\longrightarrow$

$$\begin{aligned} FTQ_{size} = &- 0.34 \cdot QDAUR + 0.64 \cdot QDATR \\ &+ 0.008 \cdot QDAUR^2 + 0.01 \cdot QDATR^2 \\ &- 0.008 \cdot QDAUR \cdot QDATR \end{aligned}$$

- Evaluated on non-trained 20% of simpoints

UFTQ: AUR-ATR (FTQ size based on Utility & Timeliness)

- Measure utility & timeliness with

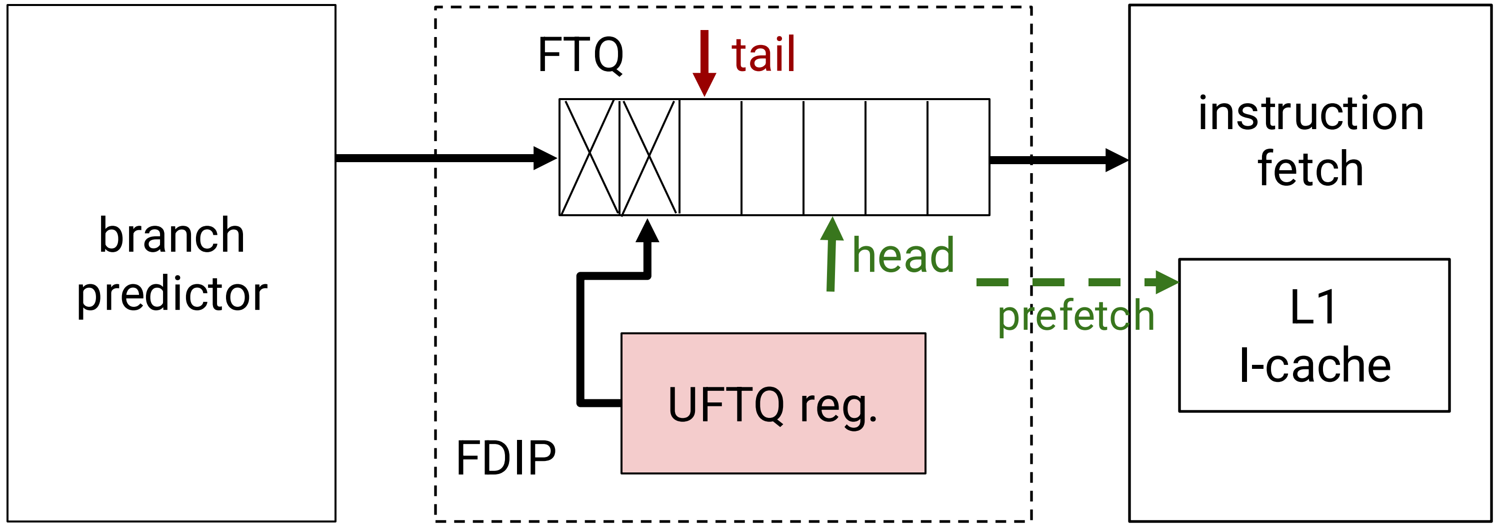
  four 10-bit counters

  two 32-bit fixed point registers (ratios)

- Adapt FTQ size dynamically

$$\boxed{Timeliness\ Ratio} = \frac{icache_{pref.hits}}{\boxed{icache_{pref.hits}} + \boxed{MSHR_{pref.hits}}}$$

$$\boxed{Utility\ \ Ratio} = \frac{Prefs_{useful}}{\boxed{Prefs_{useful}} + \boxed{Prefs_{unuseful}}}$$



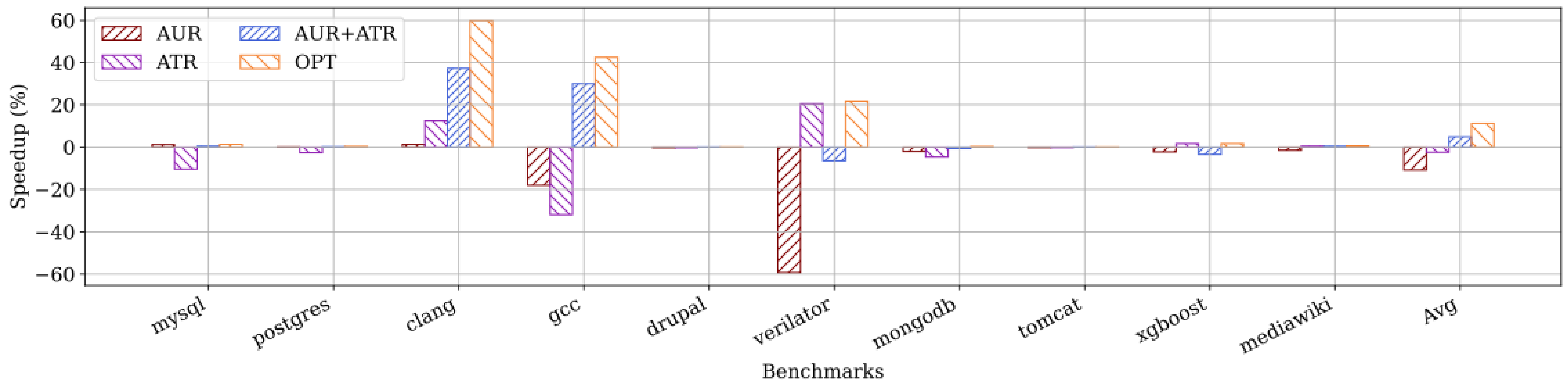
UFTQ : Evaluation - Speedup

4.9% (up to 37.2%)

IPC speedup

1.2% (up to 28%)

Icache miss reduction



UFTQ-ATR-AUR only prevents either inaccurate or untimely prefetches.

Introduction

- Optimizing data center applications

Analysis

- Why FDIP falls short of the ideal?

UFTQ

- Dynamically adapting FTQ size

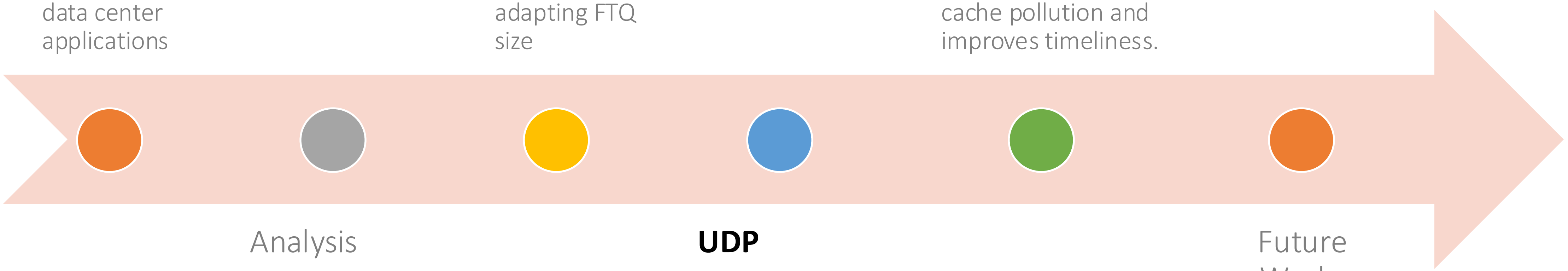
UDP

- **Utility-Driven Prefetching**

Conclusion

- Utility study reduces cache pollution and improves timeliness.

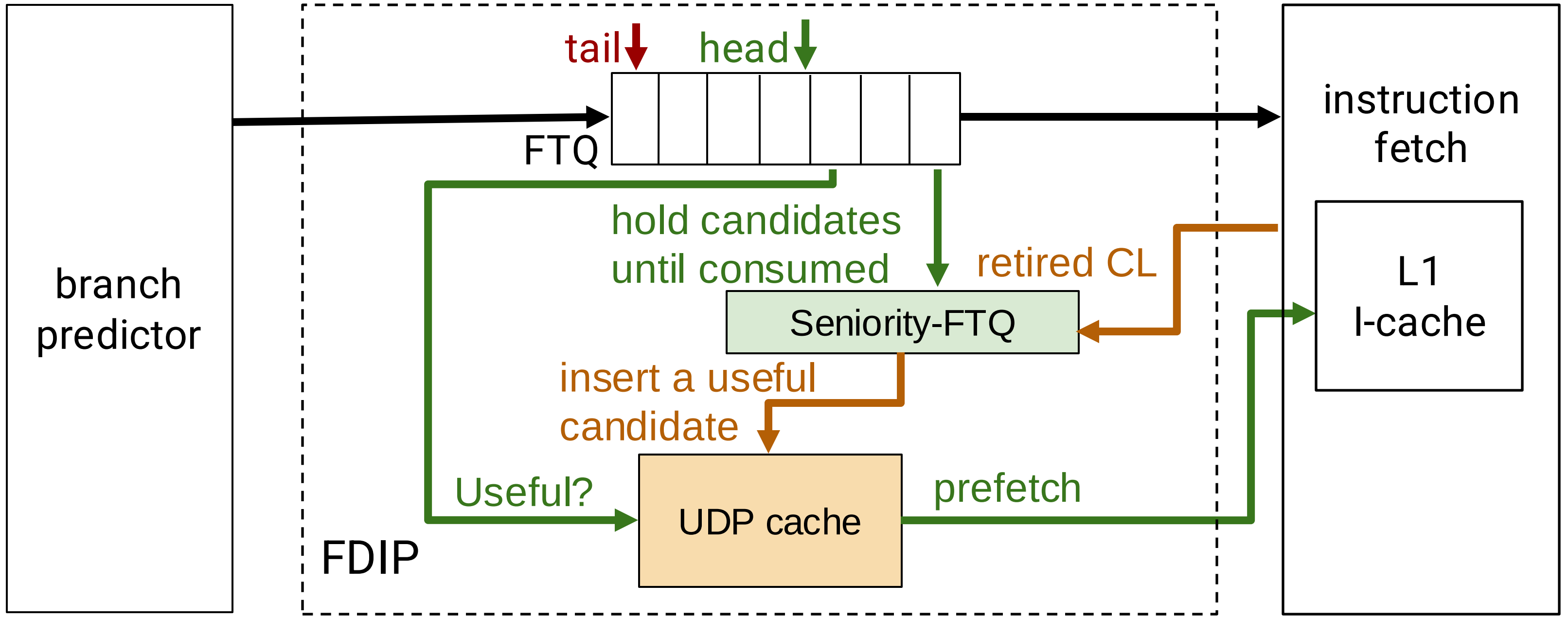
Future Works



Reached an optimal FTQ, however, still have unuseful prefetches polluting icache.
→ Can we filter out unuseful prefetches and improve timeliness of useful prefetches even more?

UDP : Utility-Driven Instruction Prefetch

- Learn usefulness of each prefetch candidate
- Predict if FDIP is on or off-path based on branch confidence
- On-path: Always prefetch
- Off-path: No prefetch unless address is in bloom filter



Seniority FTQ

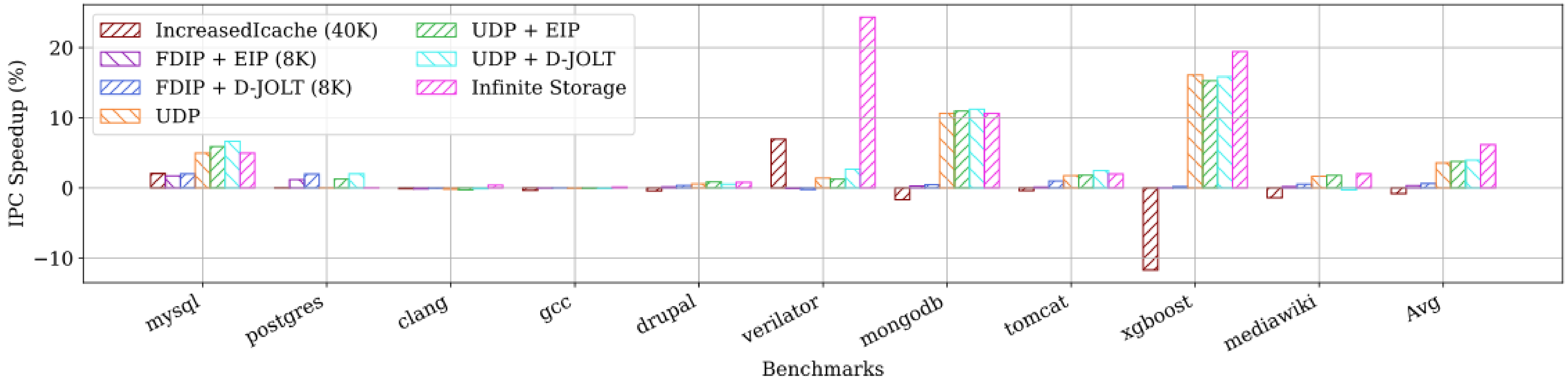
smaller than reorder buffer (ROB)

UDP cache (useful-set)

8KB Bloom Filter:
1-block/2-block/4-block filters

Filter Management:
Clear the filter when full and the unuseful ratio reaches 75%

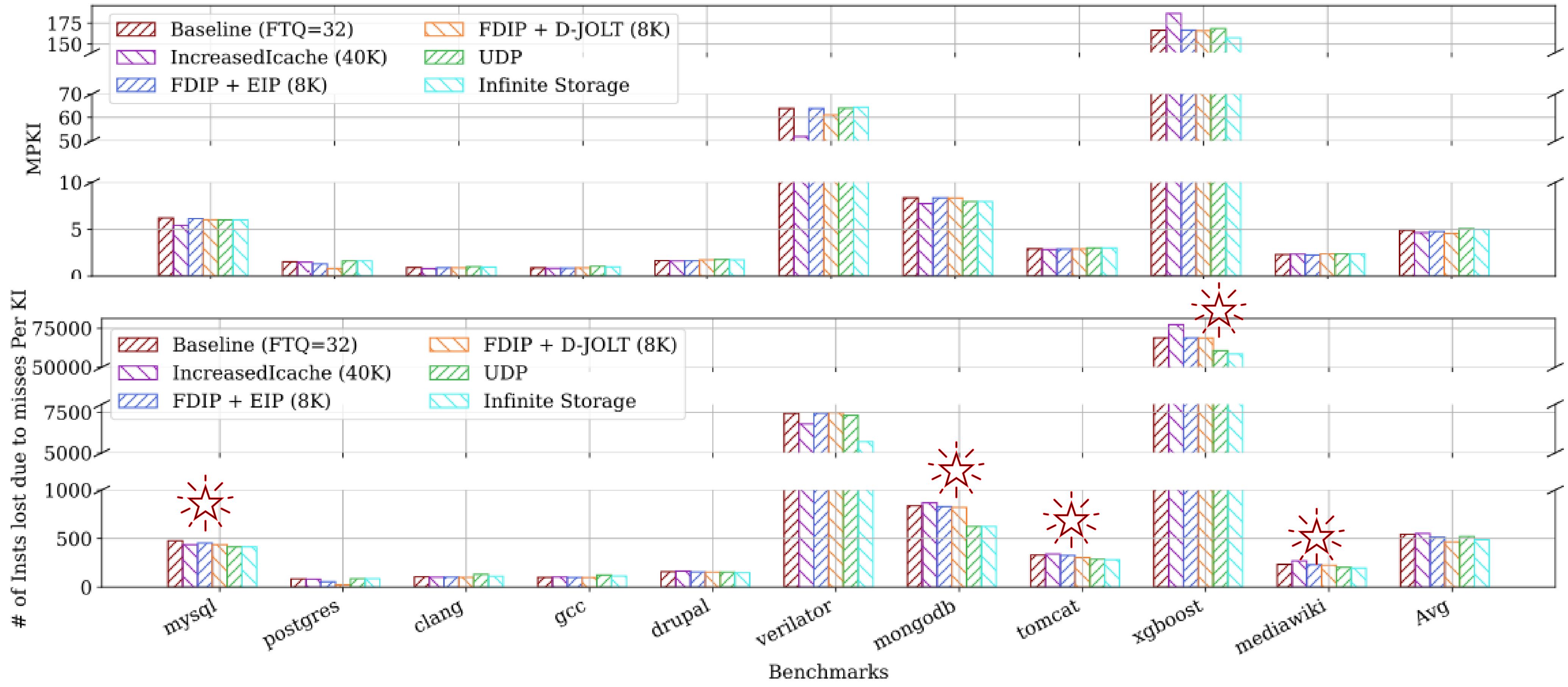
UDP : Evaluation - Speedup



UDP provides substantial performance gains of up to 16.1% for xgboost and 3.6% on average.

UDP : Evaluation – Icache Misses & Inst lost due to misses

MPKI: Misses Per Kilo Instructions



Introduction

- Optimizing data center applications

Analysis

- Why FDIP falls short of the ideal?

UFTQ

- Dynamically adapting FTQ size

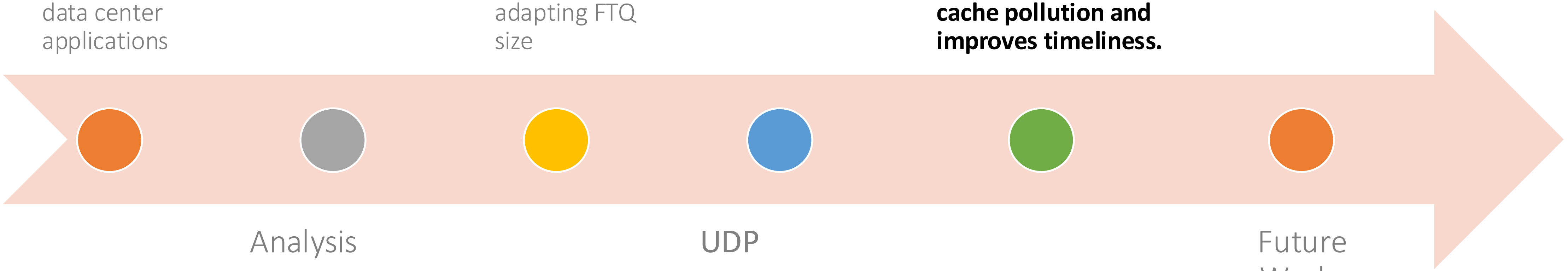
UDP

- Utility-Driven Prefetching

Conclusion

- **Utility study reduces cache pollution and improves timeliness.**

Future Works



Contribution

FDIP fails to eliminate all icache misses

Quantify the effect of untimely prefetches & inaccurate off-path prefetches

UFTQ, dynamically adapt the prefetch aggressiveness of FDIP

UDP, only prefetch useful prefetches

Introduction

- Optimizing data center applications

Analysis

- Why FDIP falls short of the ideal?

UFTQ

- Dynamically adapting FTQ size

UDP

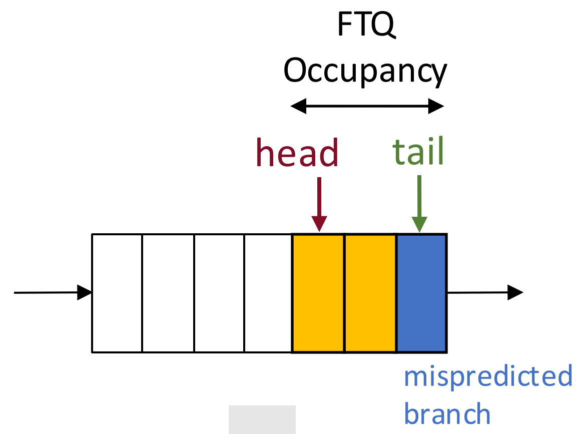
- Utility-Driven Prefetching

Conclusion

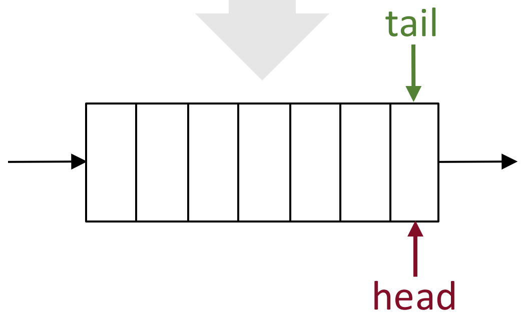
- Utility study reduces cache pollution and improves timeliness.

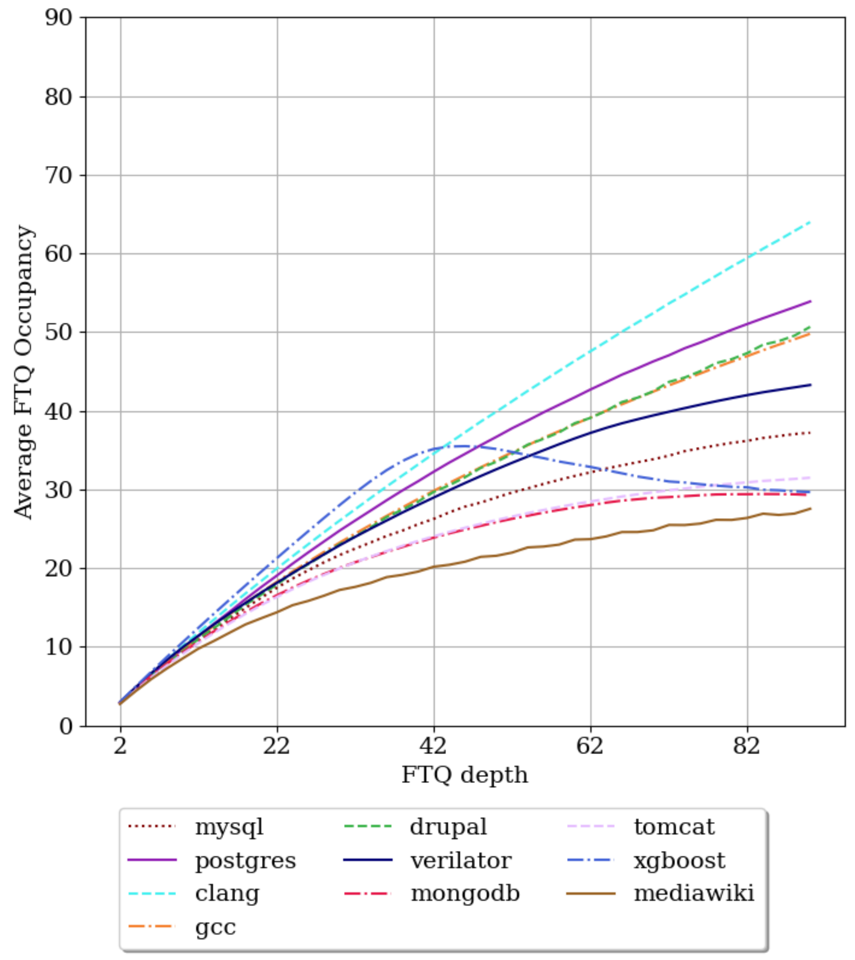
Future Works

Analysis: FDIP Recovery Frequency



Flush FTQ on a branch resolution
(recovery)
for a mispredicted branch





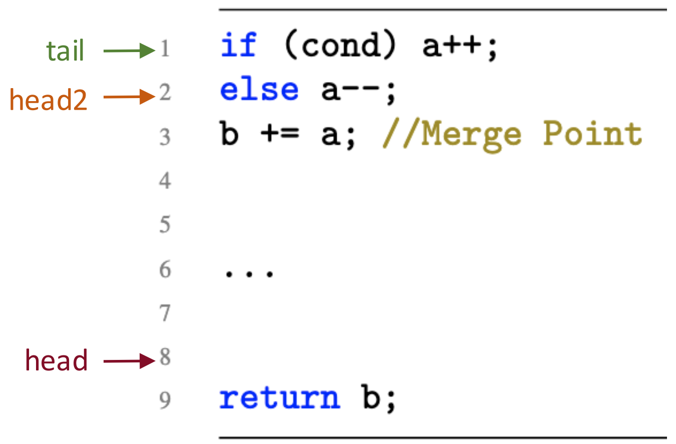
Recoveries act as a natural throttling condition for FDIP.

FDIP cannot run ahead (fully exploit FTQ) due to BTB misses and BP mispredictions.

Merge Point Study

- How to increase the run-ahead distance even with frequent recoveries?

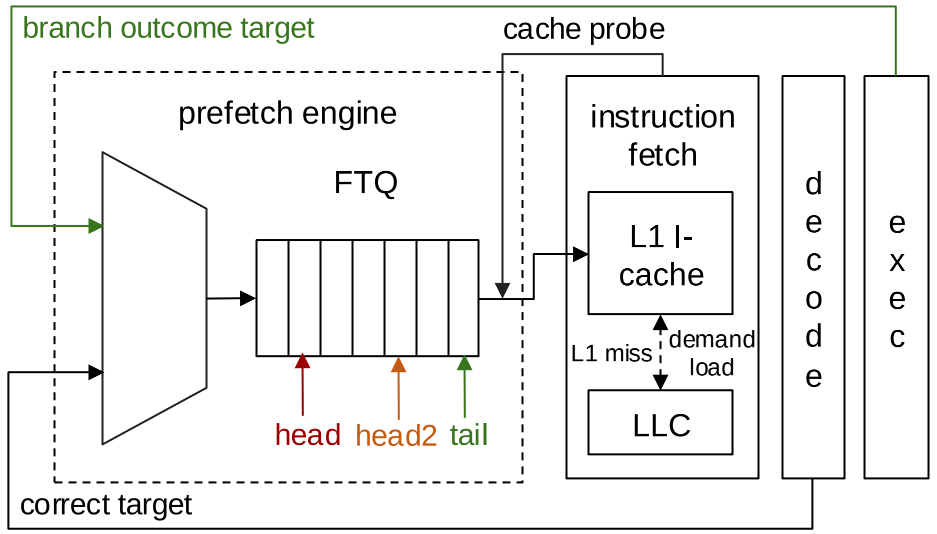
On a recovery, can FTQ not be flushed?



```
if (cond) a++;
else a--;
b += a; //Merge Point


...


return b;
```



Please check out our resources!

- Resources
  - Scarab + Decoupled frontend: https://github.com/Litz-Lab/scarab
  - Scarab Infrastructure: https://github.com/Litz-Lab/Scarab-infra
    - Dockerfiles for data center workloads ready to run with DynamoRIO (collecting traces) and Scarab simulation

- Email: soh31@ucsc.edu

Questions ?