

PIPS: Prefetching Instructions with Probabilistic Scouts

Pierre Michaud

Inria / IRISA

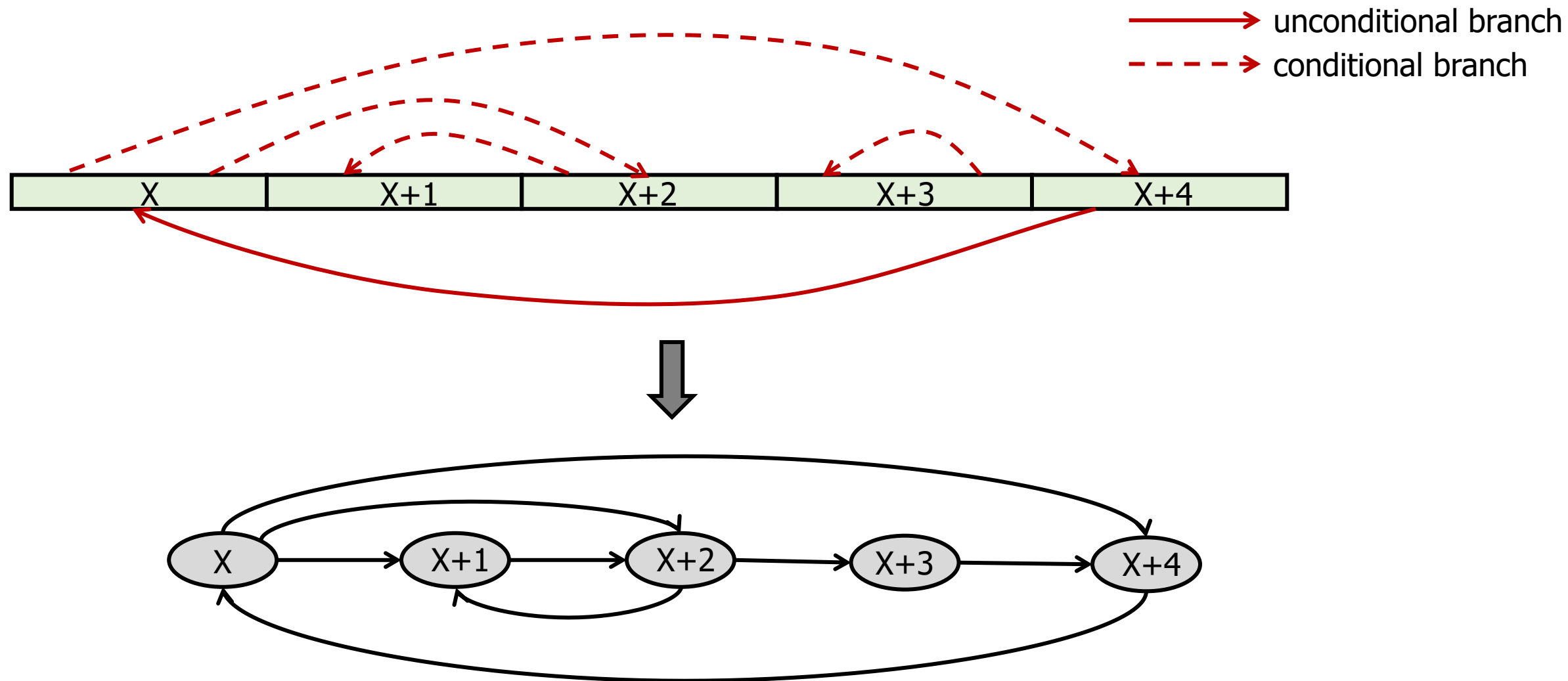
First Instruction Prefetching Championship

May 31, 2020

Control Flow Graph (CFG)

- each node of the CFG is a distinct 64-byte line
- directed edge from line X to line Y if Y is a possible successor of X
 - example: if line X contains no unconditional branch, line X+1 is a possible successor of X
- a line cannot be its own successor ($Y \neq X$)

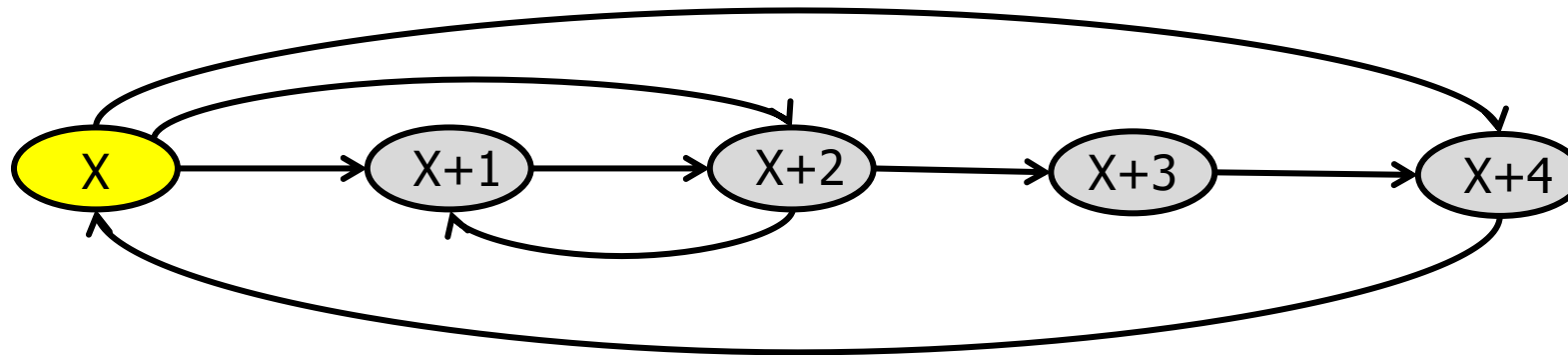
example of CFG



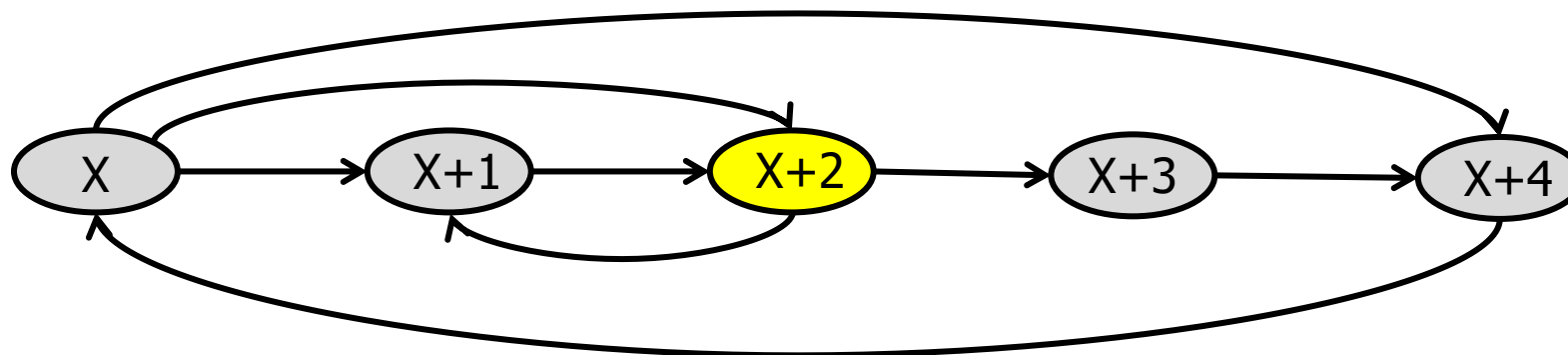
front line

- front line = line where the program counter (PC) currently is
- which PC?
 - at branch prediction?
 - at instruction fetch?
 - at decode?
 - at retirement?
- in PIPS, front line updated at instruction fetch
 - non-speculative, by the magic of trace-driven simulation

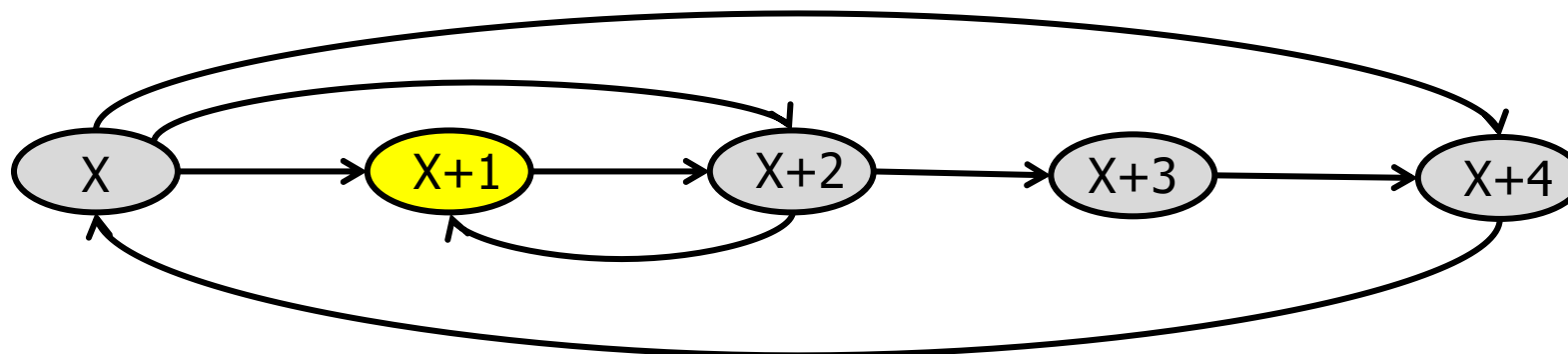
front line moves with PC



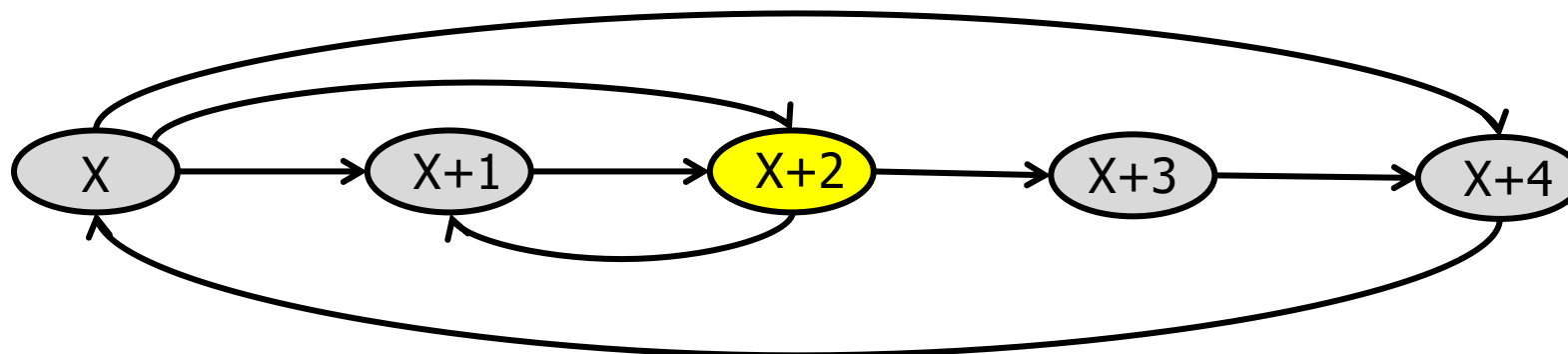
front line moves with PC



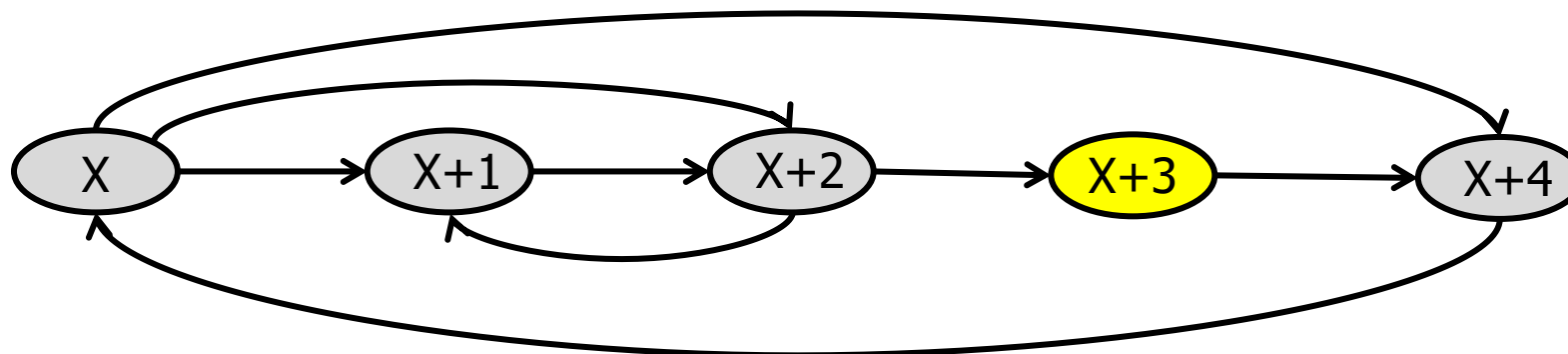
front line moves with PC



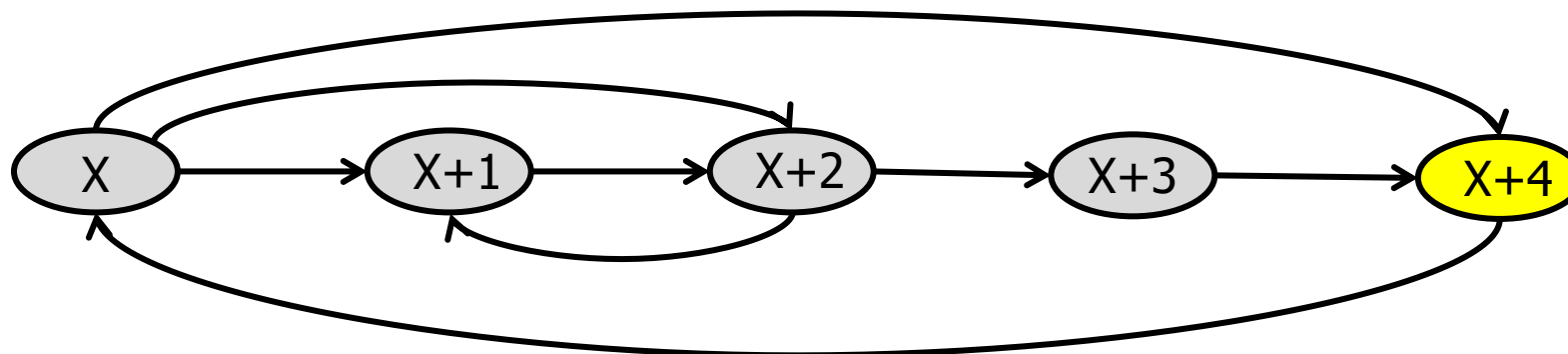
front line moves with PC



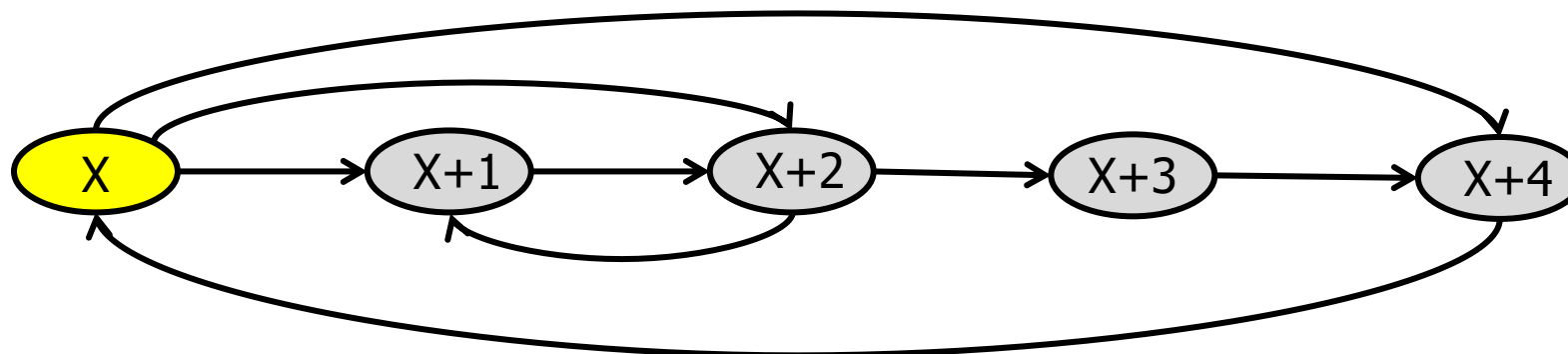
front line moves with PC



front line moves with PC



front line moves with PC



initial idea

- some edges more likely to be taken than others

- estimate edge probability $P(Y|X)$ at runtime

- maintain frequency counts

$$\sum_{Y \in \text{succ}(X)} P(Y|X) = 1$$

- **prefetch all paths whose probability exceeds a threshold**

- path probability = product of edge probabilities on that path
- depth-first traversal of the CFG, starting from front line

- seems too complex for a hardware implementation

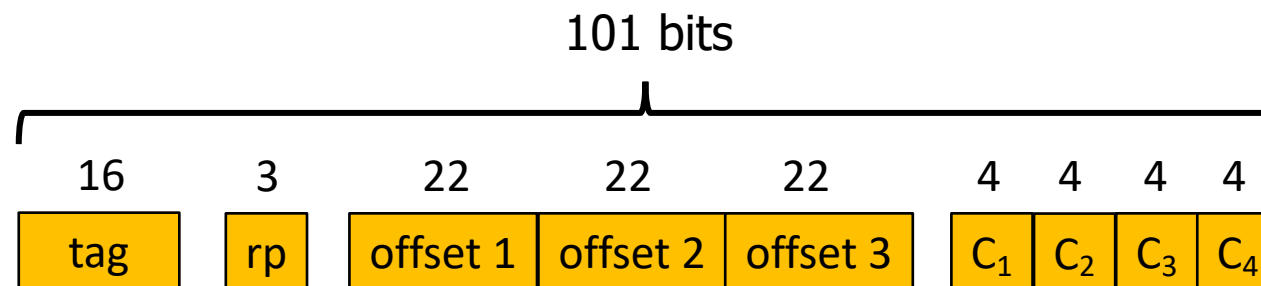
probabilistic scouting

- send **scouts** to explore the CFG, starting from the front line
 - multiple scouts alive at the same time
- a scout follows one path in the CFG, prefetching lines on that path
- **scout moves from node X to successor Y with probability $P(Y|X)$**
- scout dies when ending condition is met (e.g., after fixed time)
- when scout dies, new scout is sent from front line
 - this process repeats indefinitely

Line History Table (LHT)

- accumulates CFG information
- each LHT entry stores information about one node
- organized like a set-associative cache
- accessed with line address
- ideally, want LHT large enough to hold the whole CFG

LHT entry

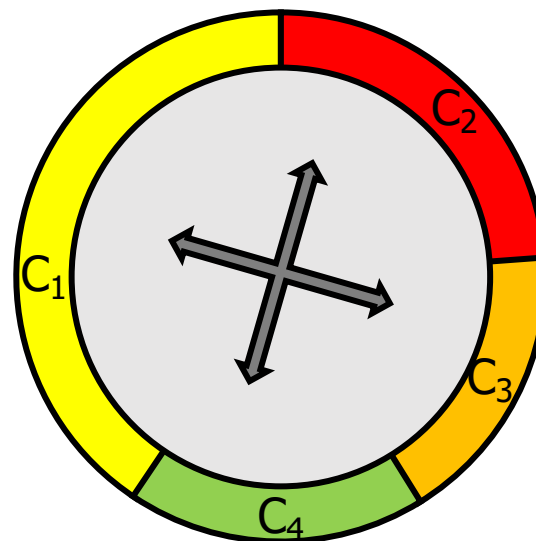


- 4 successors per node X
 - 4 frequency counts C_1, C_2, C_3, C_4
 - $X+1$ is an implicit successor
 - 3 successors Y stored as 3 signed offsets $Y-X$
- when PC takes edge $X \rightarrow Y$, increment the C_i corresponding to Y
 - when one C_i exceeds 15, halve C_1, C_2, C_3 , and C_4

Scouting Cache (SCC)

- SCC acts like a cache for the LHT
 - much smaller than LHT
- scouts access SCC
 - upon SCC miss, access LHT and copy node info into SCC
- temporal locality → scouting speed less impacted by LHT latency
- scout issues prefetch only upon SCC miss
 - reduces redundant prefetches

probabilistic selection?



probably too complex for a hardware implementation....

pseudo-probabilistic selection

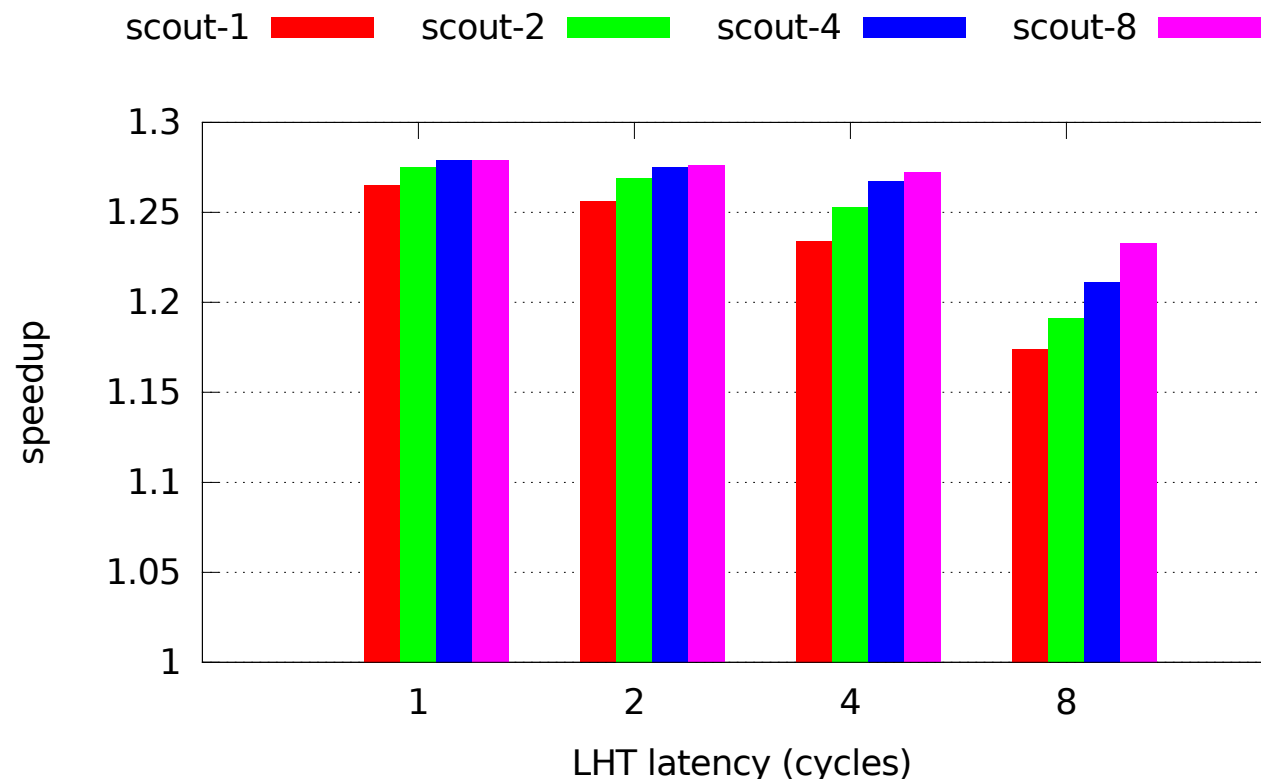
- use SCC information only
- select successor with greatest frequency count
- decrement selected frequency count in the SCC
 - not in the LHT
- when SCC entry has all its frequency counts zeroed, entry kills scouts reaching it
 - reduces over-prefetching

submitted prefetcher: PIPS

	sets	ways	rp	KB
LHT	1024	10	SRRIP	126.25
SCC	32	4	SRRIP	1.56

- 4 parallel scouts
- scouts die when prefetch queue occupancy > 7
- when front line moves, kill one scout
 - victim scout chosen in round-robin fashion
- **LHT & SCC latency < 1 cycle**
 - allowed by championship rules

LHT latency and bandwidth

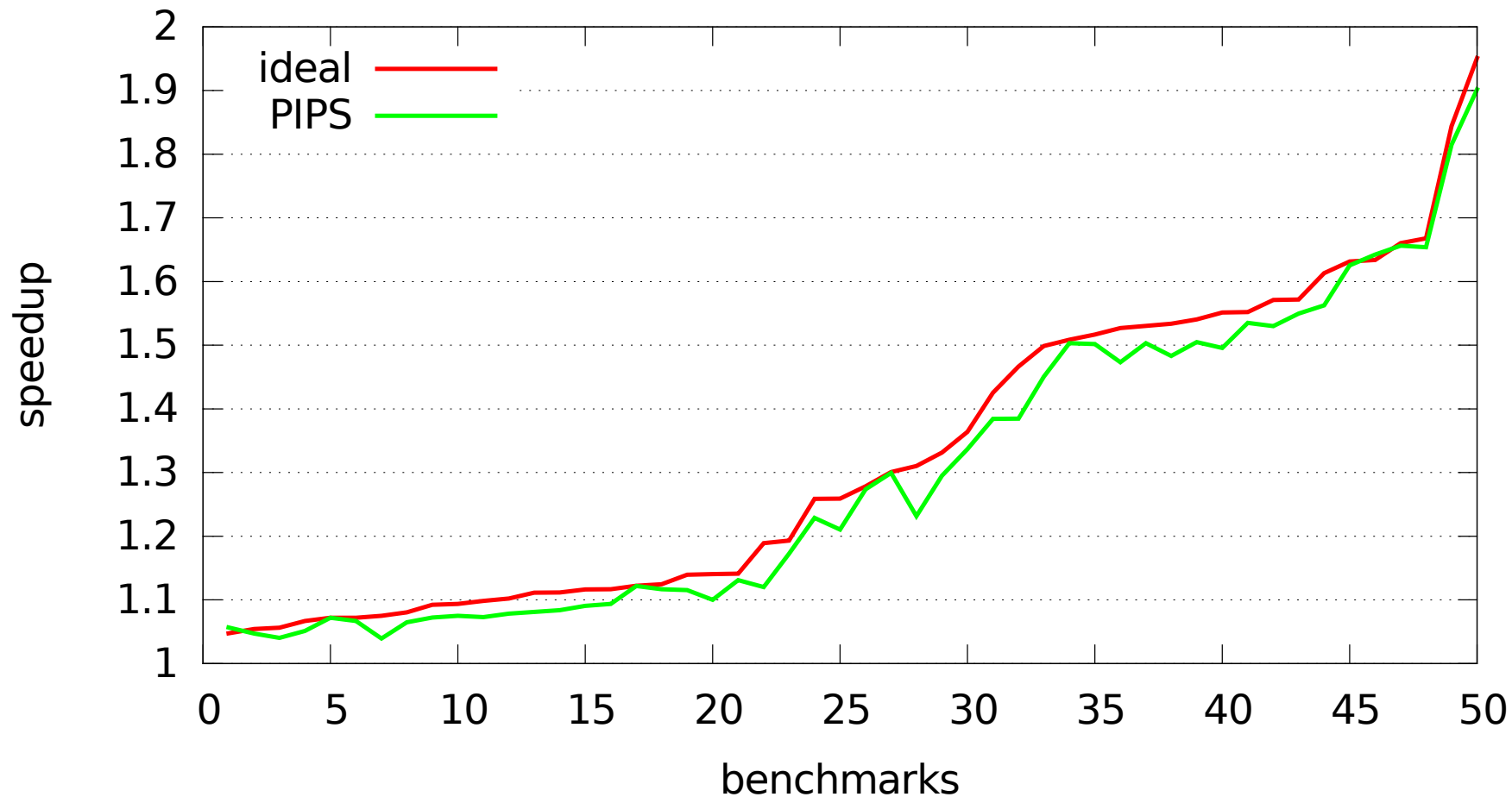


- IPC drops when LHT latency increases
- can be mitigated, somewhat, by increasing number of parallel scouts and LHT bandwidth


How far from ideal prefetch?

- how to estimate ideal-prefetch IPC with simulator?
- huge IL1 cache? → overestimates ideal-prefetch IPC
 - reduced pressure on L2 cache and LLC → fewer data misses
- better: magically turn IL1 misses into hits, but send miss requests to L2
 - approximate upper bound (prefetching changes order of memory accesses)

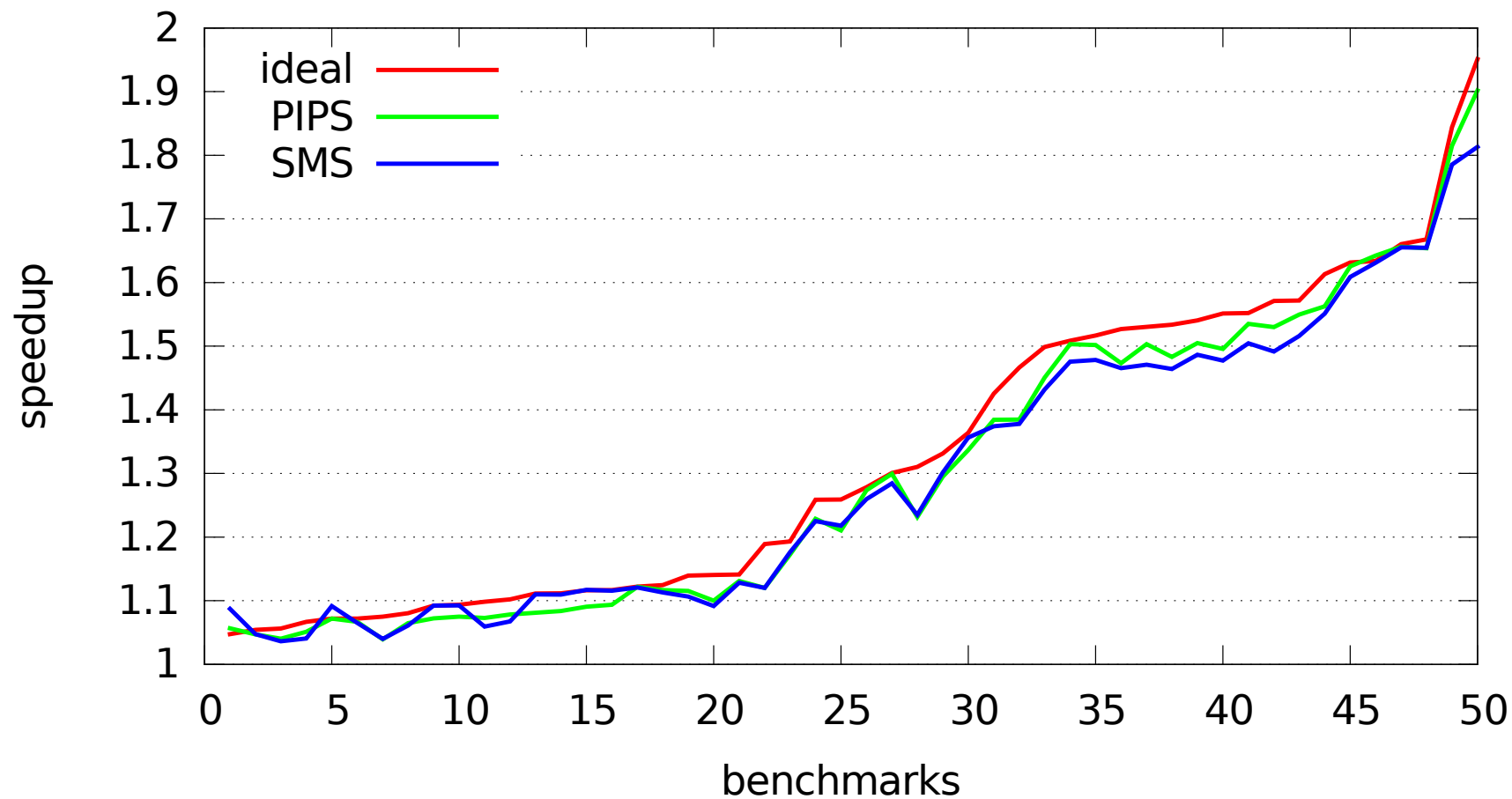
PIPS vs. ideal prefetch



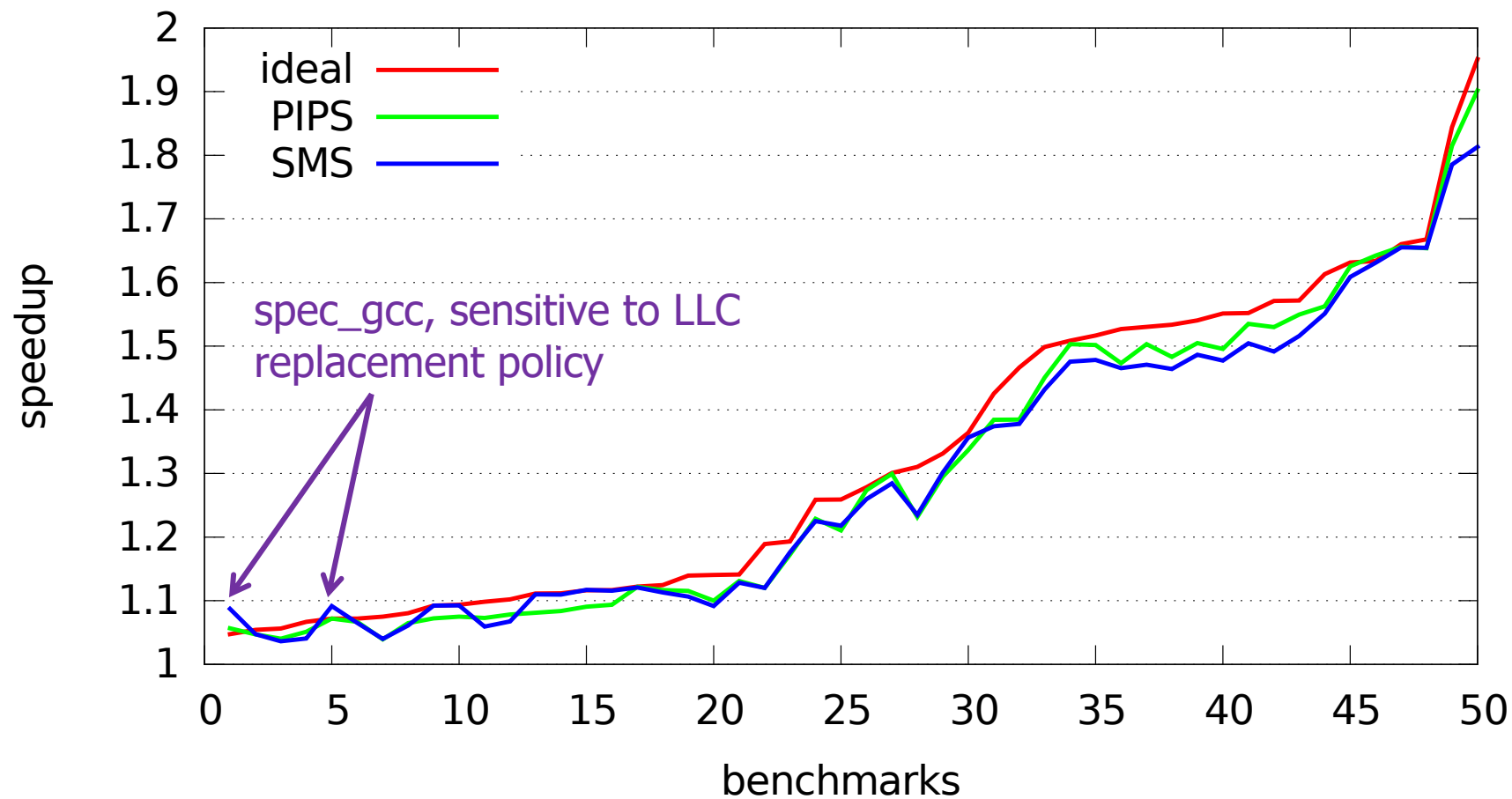
post-deadline experiment

- PIPS does not exploit spatial locality... 
- let's try an **SMS**-like prefetcher
 - “**Spatial Memory Streaming**”, Somogyi et al., ISCA 2006
 - SMS + tuning + predict next spatial region to become active
- probably easier to implement in hardware
 - LHT latency problem in PIPS
- probably more energy-efficient
 - less over-prefetching than PIPS

SMS almost as good as PIPS



SMS almost as good as PIPS



Conclusion

- PIPS is for trying to win the championship, not for real processors
 - maybe there is something to learn from it though...
- suggestion for next championship: provide SRAM model
 - accessing a large table with zero latency is not realistic