Continual Flow Pipelines

Srikanth Srinivasan, **Ravi Rajwar**, Haitham Akkary,
Amit Gandhi, Mike Upton

Intel Corporation

ASPLOS-XI October 2004

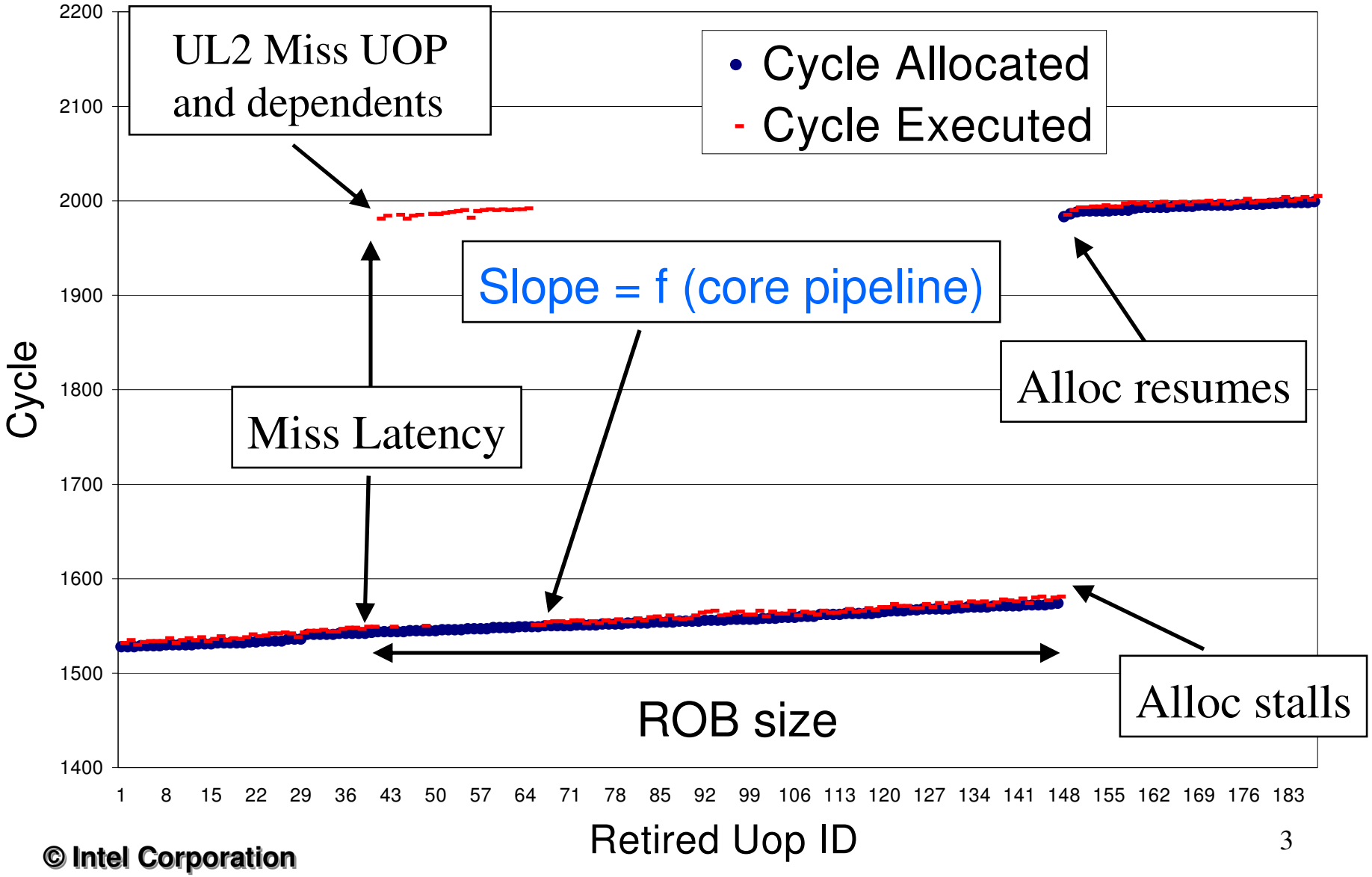
MRL

Problem: Memory latency

- Long latency memory operations
  - disrupt pipeline and stall back-end
- Very large caches
  - inefficient performance
  - negatively impact die size
- Very large instruction window (>4K)
  - memory parallelism and latency tolerance
  - conventional methods intractable

Need high performance using small caches and buffers

TPC-C profile



What happened?

- L2 miss and dependents (blocked)
  - comprise small fraction of window
  - block ROB and other resources
- Miss-independents
  - comprise large fraction of window
  - can proceed but don't have resources
- ROB cannot overlap L2 miss
  - memory latency exposed
  - pipeline disrupted—backend stalls

Solution: Continual Flow Pipelines

Treat Blocked & Independents differently

- Blocked (Miss-dependents, Slice)
  - create self-contained slice and drain out
  - free critical resources
  - execute when miss returns
- Independents
  - execute and “pseudo-retire”
  - retain ability to undo using checkpoint
- Automatic result integration
  - no re-execution!

Outline

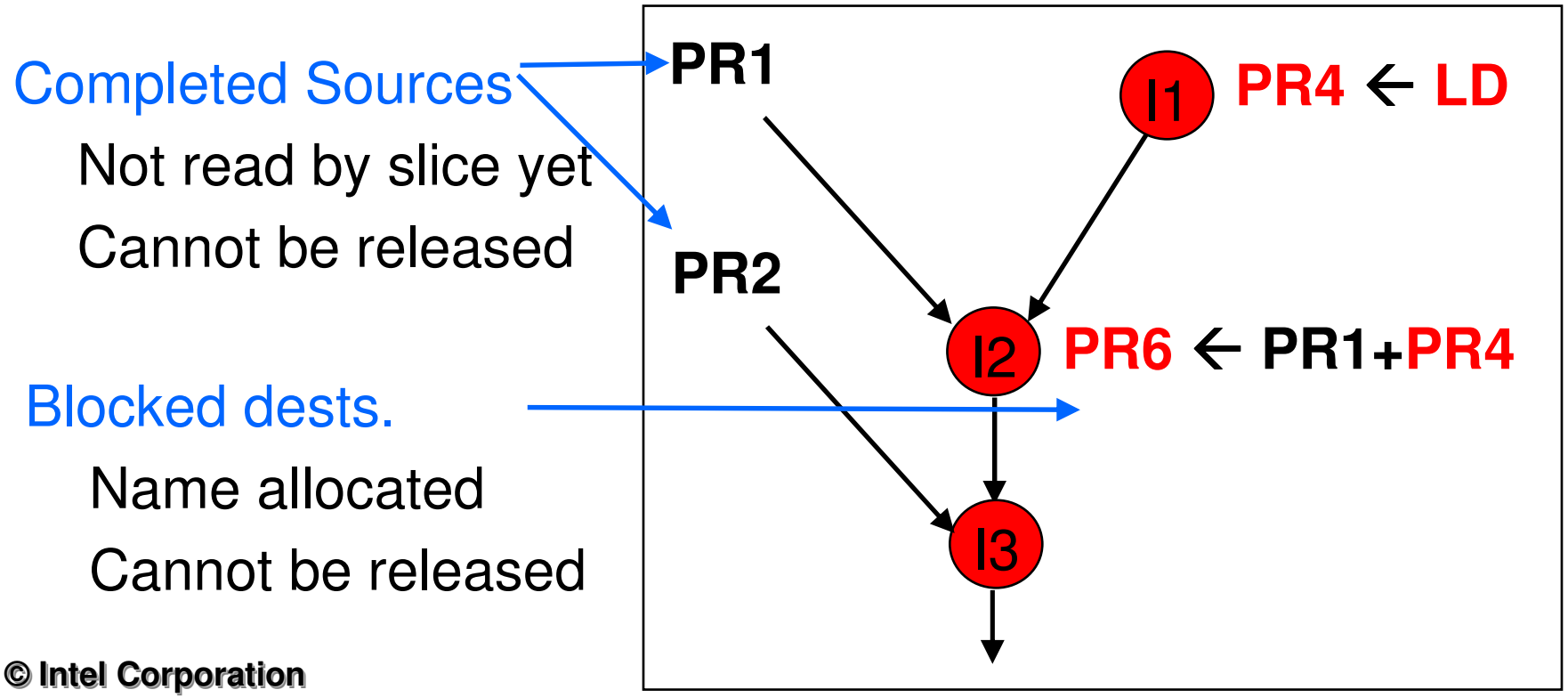
- Introduction
- Motivation
- Continual Flow Pipelines
  - Concepts
  - Performance
  - Analysis
- Summary

Motivation (1/3)

- No misses → no problem
  - instructions execute quickly, free resources
  - critical resources sized for L2 hit
- Long latency miss → stalls backend
  - blocked instructions cannot execute
    - occupy and block resources
  - large window needed (>4K instructions)

Motivation (2/3)

- Blocked instructions cannot execute
  - may continue to occupy scheduler entries
  - put pressure on register file



Motivation (3/3)

- Scheduler
  - non-blocking proposals exist
    - Pentium 4-style replay
    - WIB (as large as instruction window)
- Registers
  - no solution for completed source registers
  - late allocation of blocked destinations
    - significant pipeline changes

No unified non-blocking solution exists

Outline

- Introduction
- Motivation
- Continual Flow Pipelines
  - Concepts
  - Performance
  - Analysis
- Summary

Continual Flow Pipelines

Treat Blocked & Independents differently

1. Blocked
   - create **self-contained slice** and drain out
   - free critical resources
2. Independents
   - execute and “pseudo-retire”
   - free critical resources
   - can undo
3. Automatic result integration
   - **no re-execution**

CFP key actions

Detect L2 miss and save state

1. Drain slice (incl. completed sources)
2. Process, execute, “retire” independents
3. When L2 miss serviced, process slice
4. Merge outputs of slice and independents

1. Identifying and draining slice

- Propagate poison (NAV) bits to consumers
  - registers
  - store queue (for memory poisoning)
- Blocked instructions (Slice)
  - treat NAV source reg. as “ready” and “read”
  - read completed source reg. and mark “read”
  - flow through pipeline
    - allow release of registers and scheduler entry
  - enter buffer along with completed source value
    - record renamed names for registers

2. Processing slice & independents

- Independents execute normally
  - "pseudo retire"
  - release critical resources
- Rename map filter
  - track live outs (independent or slice)
  - "all" allocated instructions update filter

3. Executing slice

- When L2 miss data returns
- Blocked instructions in slice buffer
  - have completed source “value”
    - don’t re-read register file for completed sources
    - become immediates
  - re-acquire register names
    - physical-to-physical register renaming
    - filter read to ensure appropriate live-out linking
  - re-acquire scheduler entry
  - execute normally through pipeline

4. Merging outputs

- Automatic integration
  - no explicit operation required
  - rename map filter tracks live outs continuously

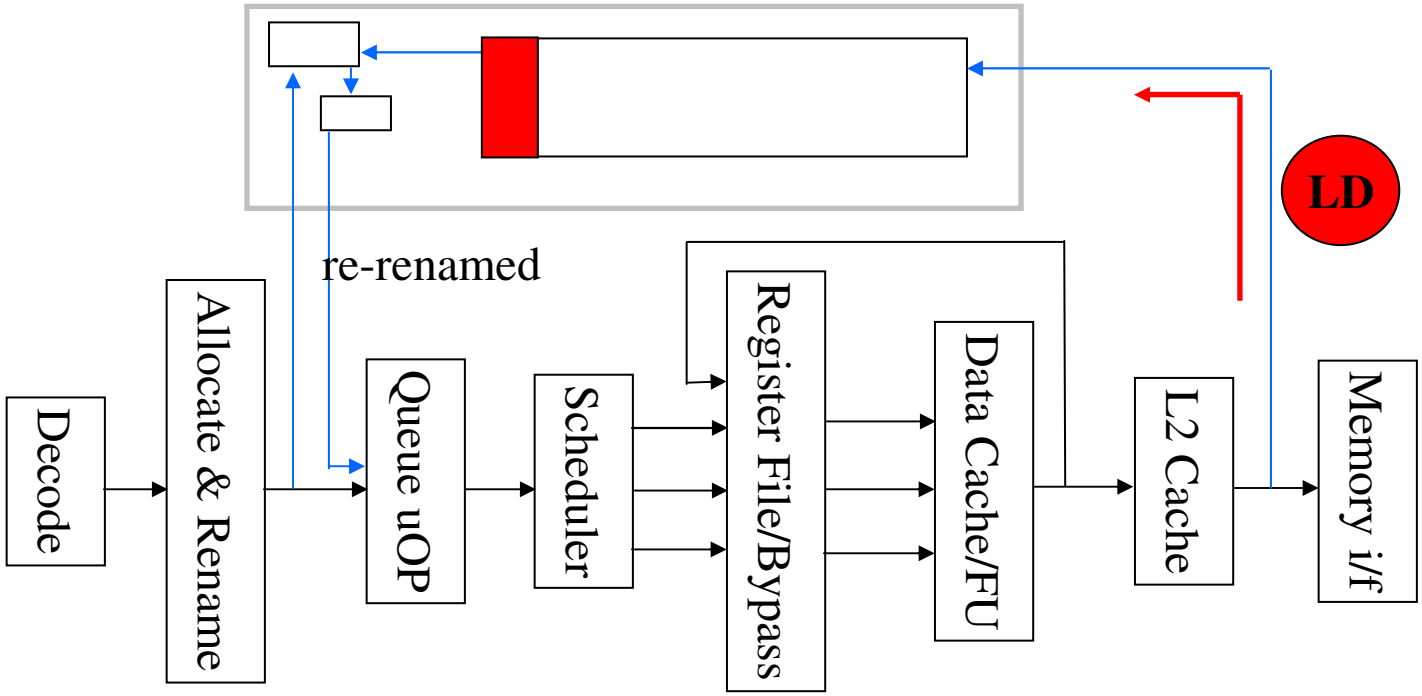
Wait for next miss…

Key components

- Slice Data Buffer
  - dense FIFO SRAM structure
  - significantly smaller than target window
- Slice filter and remapper
  - fixed size structure
- Checkpoints
  - for recovering if necessary
- L2 load buffer and L2 store queue
  - for memory buffering and ordering

Simple dense structures off the critical path

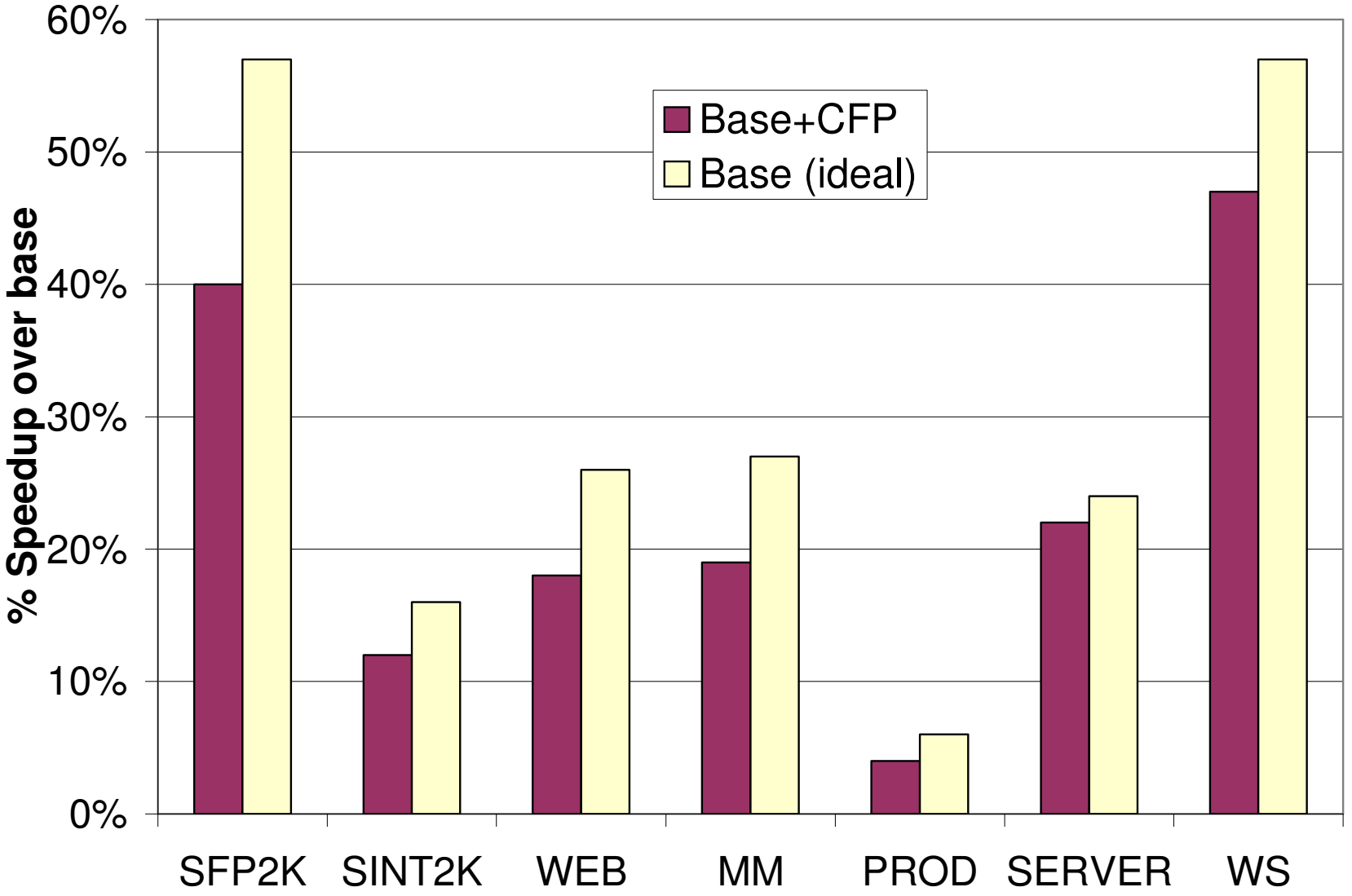
Block diagram for CFP



Outline

- Introduction
- Motivation
- Continual Flow Pipelines
  - Concepts
  - Performance
  - Analysis
- Summary

CFP performance

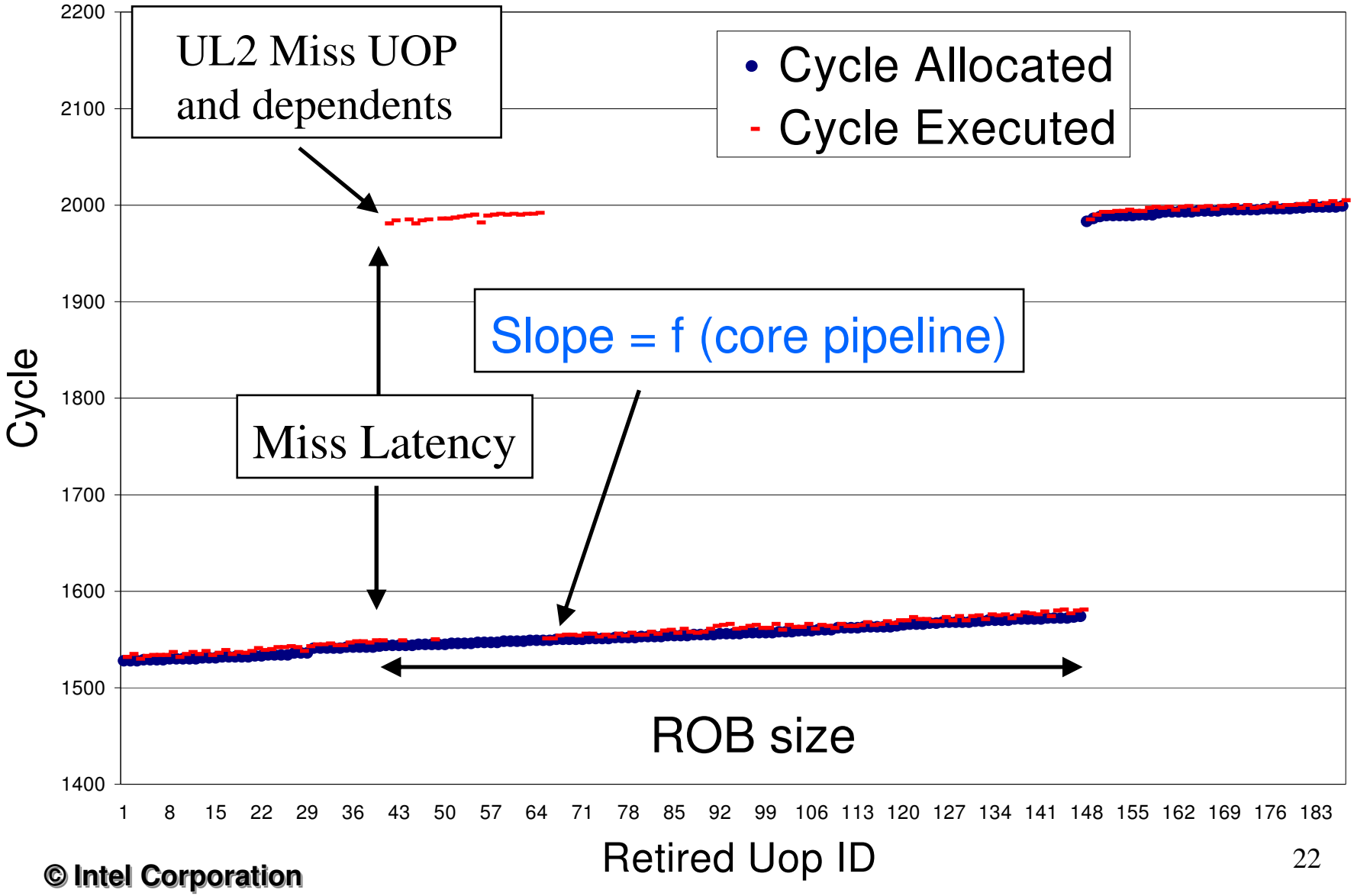


CFP performance intuition

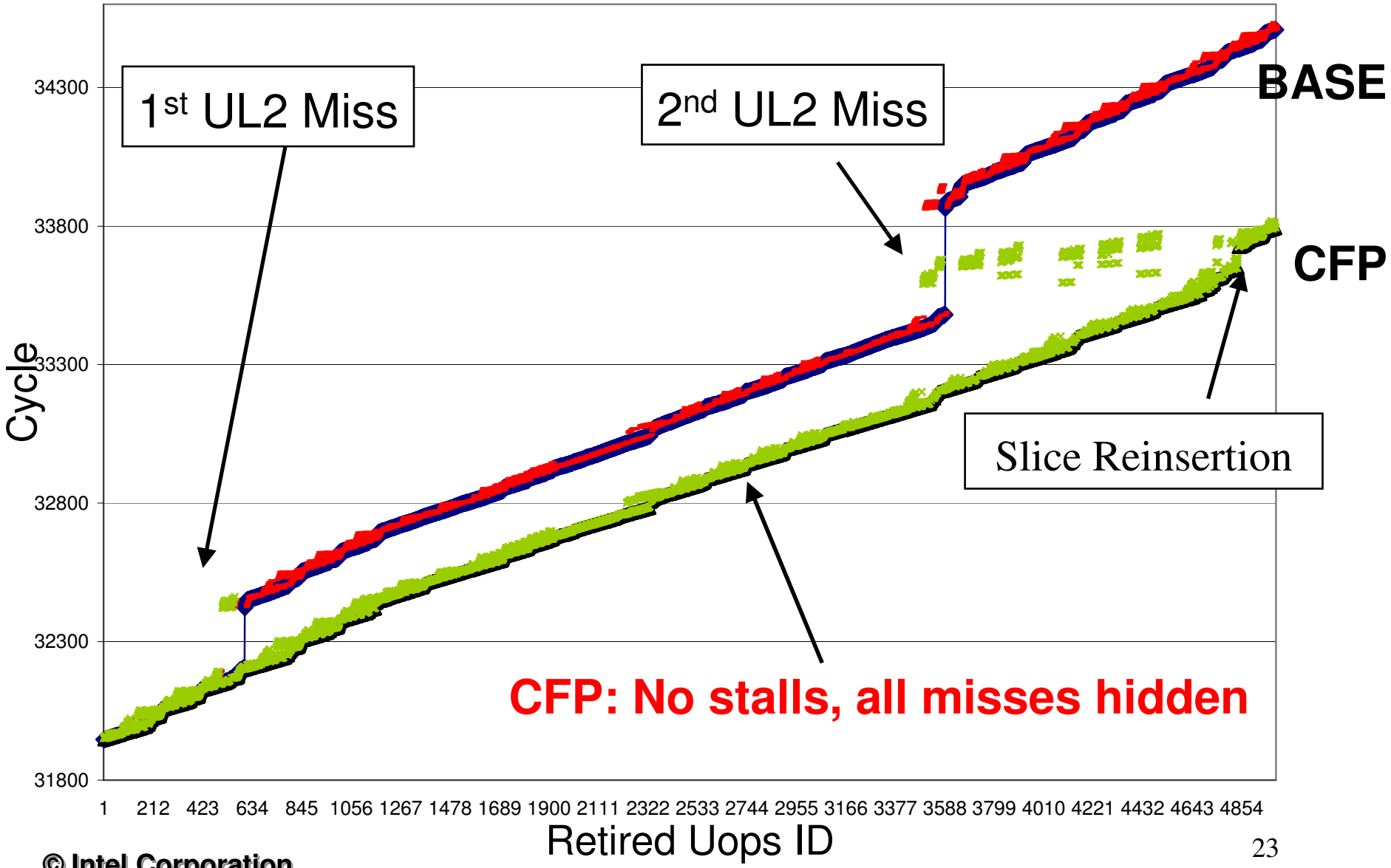
- Memory latency tolerance
  - significant useful independent work
  - no re-execution, automatic result integration
  - can tolerate first and isolated misses
- Memory level parallelism
  - when clustered misses occur
  - overlap multiple misses

CFP achieves BOTH

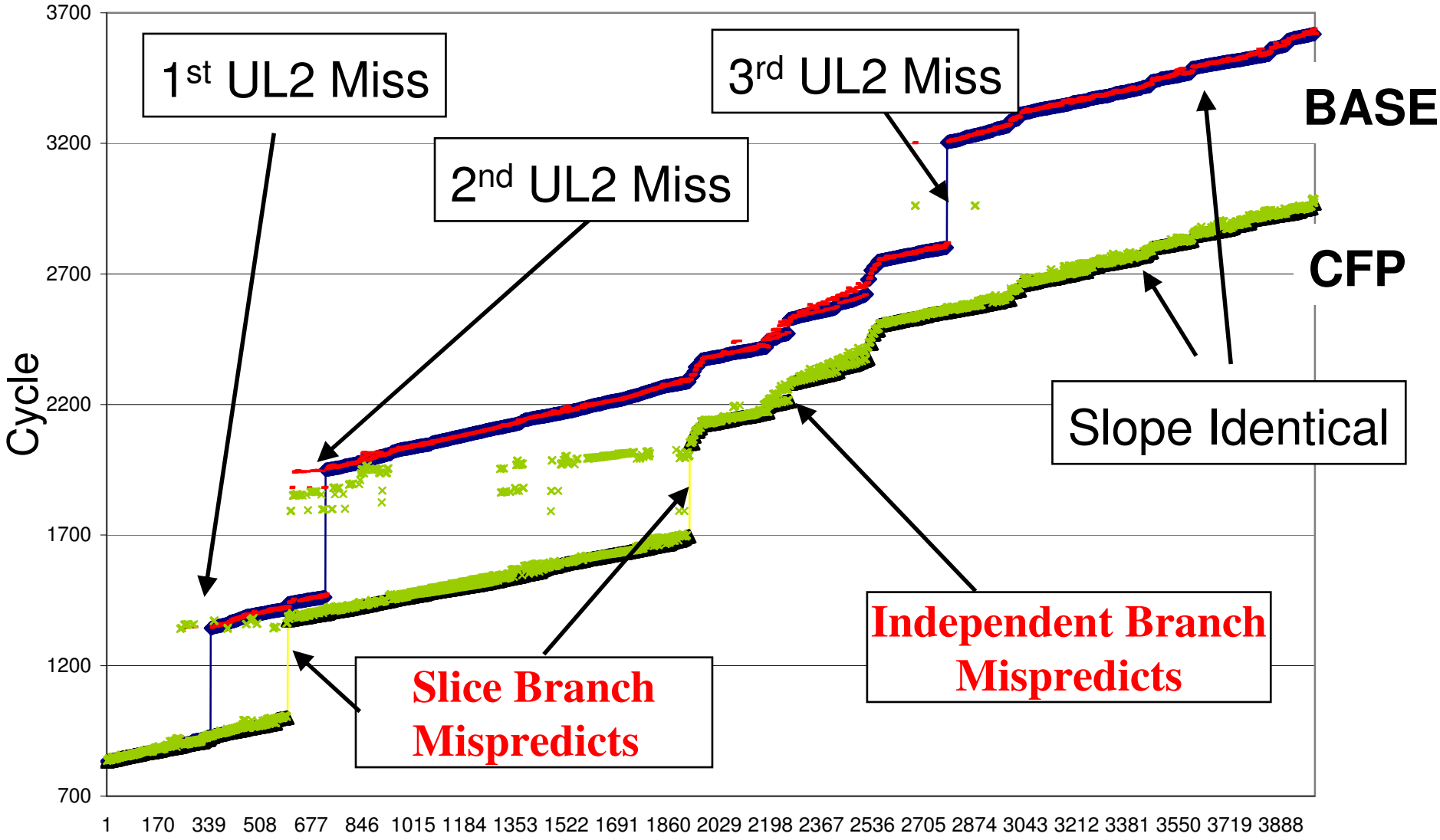
Analysis: Base description



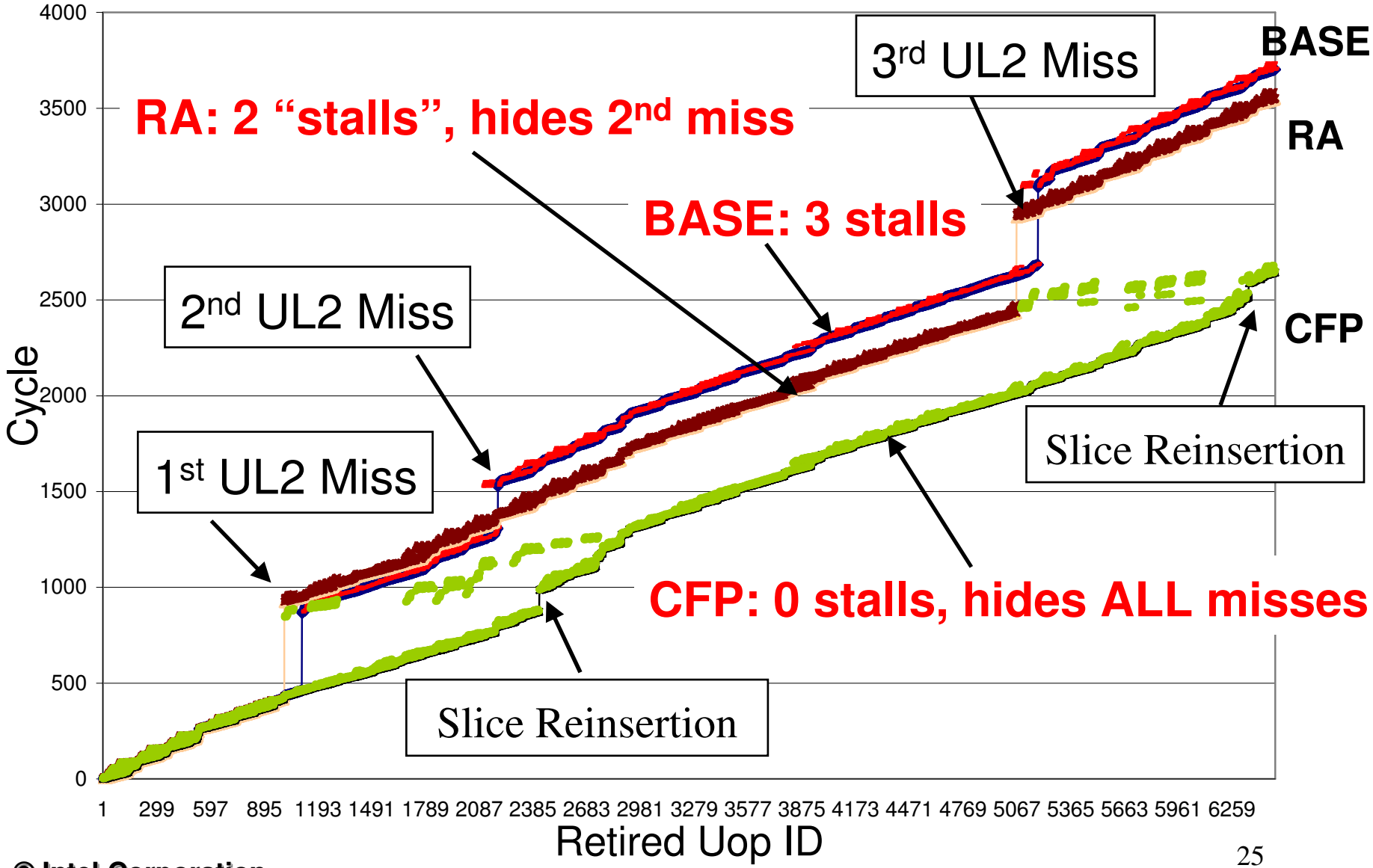
Analysis: SFP2K overlay



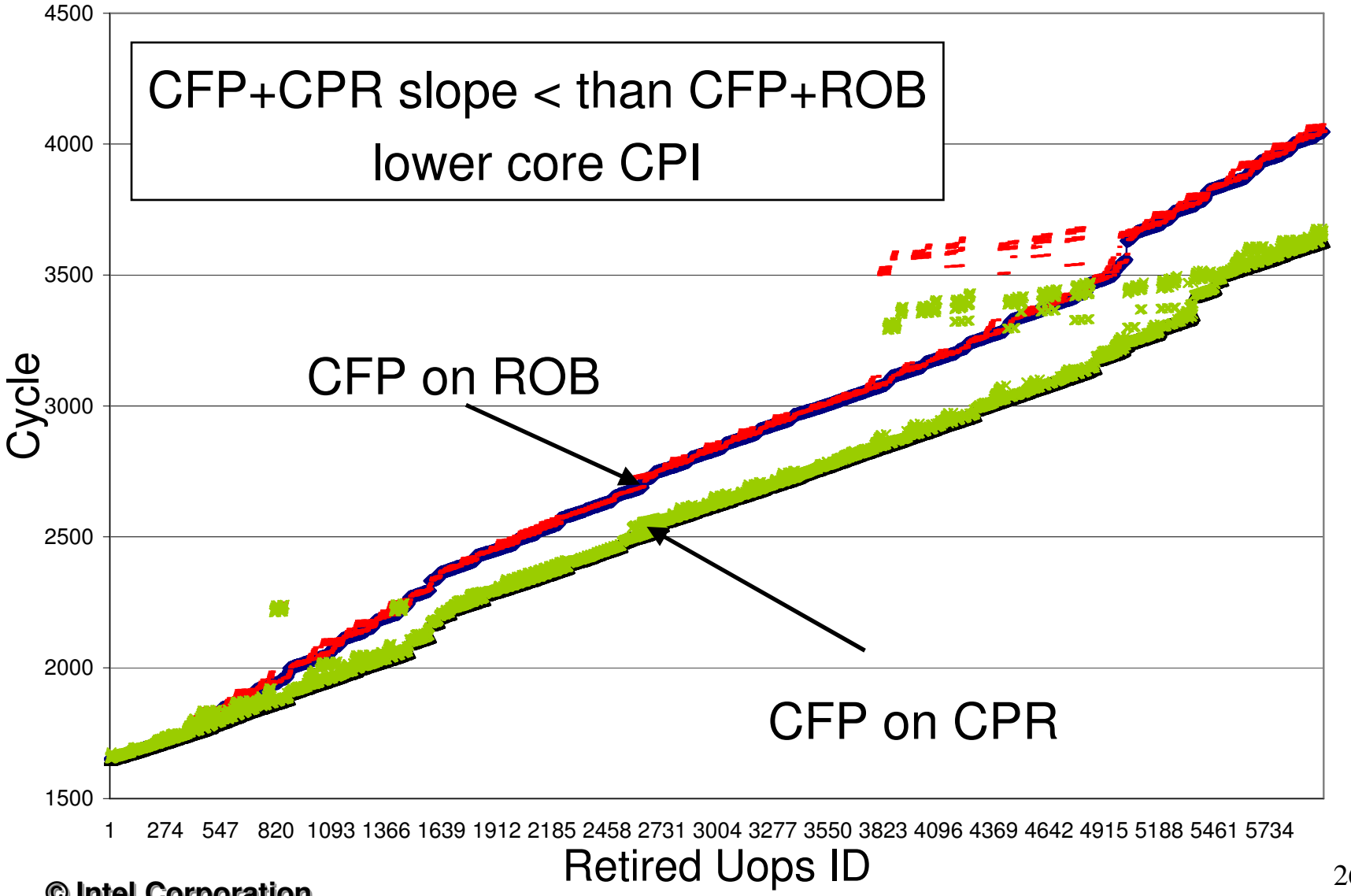
Analysis: TPC-C overlay (events)



Analysis: CFP and Runahead



Impact for core changes



Summary

- Treat miss-dependents differently
  - buffer “self-contained” slice outside pipeline
  - execute, retire indeps., don’t re-execute
- Unified non-blocking proposal
  - high memory-latency tolerance
    - enables high cache efficiency
  - high single-thread performance
  - enables more cores instead of more cache