

A Co-Designed Virtual Machine for Instruction Level Distributed Processing

Ho-Seop Kim

Ph.D. Defense

Electrical and Computer Engineering
University of Wisconsin—Madison

December 2nd, 2004

Contents

- Introduction
- ILDP Instruction Set Architecture
- ILDP Microarchitecture
- Dynamic Binary Translation for ILDP
- Control Transfer Support Mechanisms
- Evaluation
- Conclusions

Technology Trends: Pipelining

Basic P6 Pipeline

1	2	3	4	5	6	7	8	9	10
Fetch	Fetch	Decode	Decode	Decode	Rename	ROB Rd	Rdy/Sch	Dispatch	Exec

Basic Pentium® 4 Processor Pipeline

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
TC Nxt IP	TC Fetch	Drive	Alloc	Rename	Que	Sch	Sch	Sch	Disp	Disp	RF	RF	Ex	Flgs	Br Ck	Drive			

- Deeper pipeline (= fewer logic levels per pipe stage):
 - + Higher clock speed
 - Higher power consumption, lower IPC, design complexity, more aggressive circuit and fabrication techniques
- Note “Drive” stages – global wire delays

Technology Trends: Pipelining

Basic P6 Pipeline

1	2	3	4	5	6	7	8	9	10
Fetch	Fetch	Decode	Decode	Decode	Rename	ROB Rd	Rdy/Sch	Dispatch	Exec

Basic Pentium® 4 Processor Pipeline

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
TC Nxt IP	TC Fetch	Drive	Alloc	Rename	Que	Sch	Sch	Sch	Disp	Disp	RF	RF	Ex	Flgs	Br Ck	Drive			

- Can we design a moderate depth *and* high clock frequency pipeline?

1	2	3	4	5	6	7	8	9	10
Fetch	Fetch	Decode	Decode	Decode	Rename	ROB Rd	Rdy/Sch	Dispatch	Exec

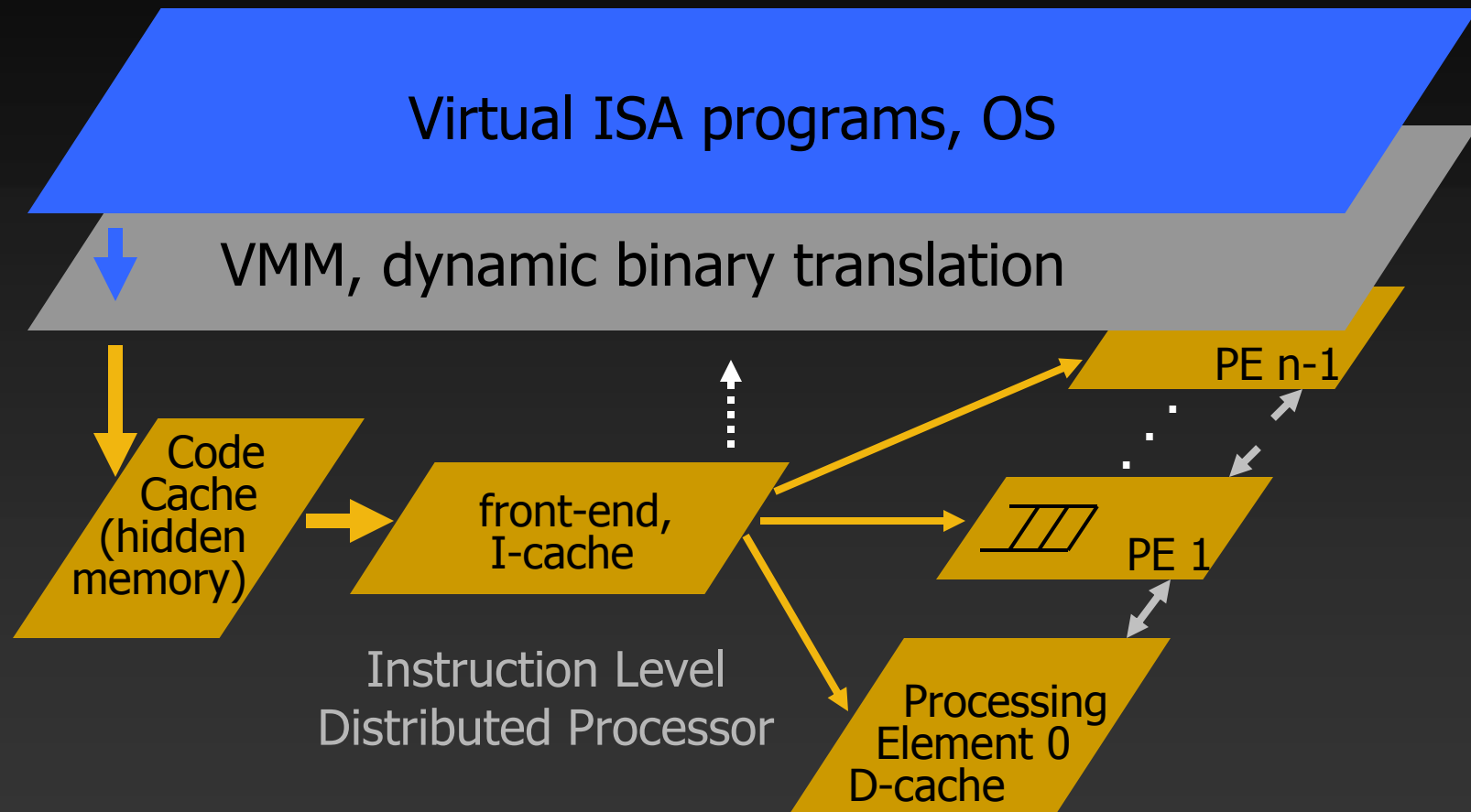
Desired Microarchitecture Properties

- Reduce requirements on important pipeline structures
 - Higher clock speed and/or lower power consumption
- Simpler design
 - Less low-level hardware-based speculation techniques
 - Less aggressive circuit design techniques
 - Reduced design time
- Tolerant to on-chip global wire latency

Instruction Level Distributed Processing

- Consider microarchitecture as a distributed computing problem
- Account for communication as well as computation
 - Localize communication to small units while managing the overall structure for communication
- Conventional ISAs are not the best fit
- Should work well with irregular programs, not just with high ILP programs

Big Picture: Co-Designed Virtual Machine



- Similar to Crusoe, DAISY but CD-VM \neq VLIW
- Alpha (V-ISA) \rightarrow ILDP (accumulator-oriented I-ISA)

ILD P Summary

- Rediscover accumulator-oriented ISA: better suited for today's technology constraints
- Simple, high clock speed/low power, wire latency tolerant processor
- Stay *even* on IPC, *gain* on clock speed (or reduce power consumption)
- Compared to VLIW-based CD VMs, HW/SW trade off point is closer to HW → low-overhead DBT
- **Simplicity is the keyword**

Thesis Summary

- A co-designed VM using an accumulator ISA and microarchitecture is an effective approach for implementing out-of-order superscalar processors
- Contributions:
 - Develop three key components: ISA, microarchitecture, dynamic binary translator
 - Evaluate performance and complexity with a correct-by-design simulation framework

Contents

- Introduction
- **ILDP Instruction Set Architecture**
- ILDP Microarchitecture
- Dynamic Binary Translation for ILDP
- Control Transfer Support Mechanisms
- Evaluation
- Conclusions

Observations

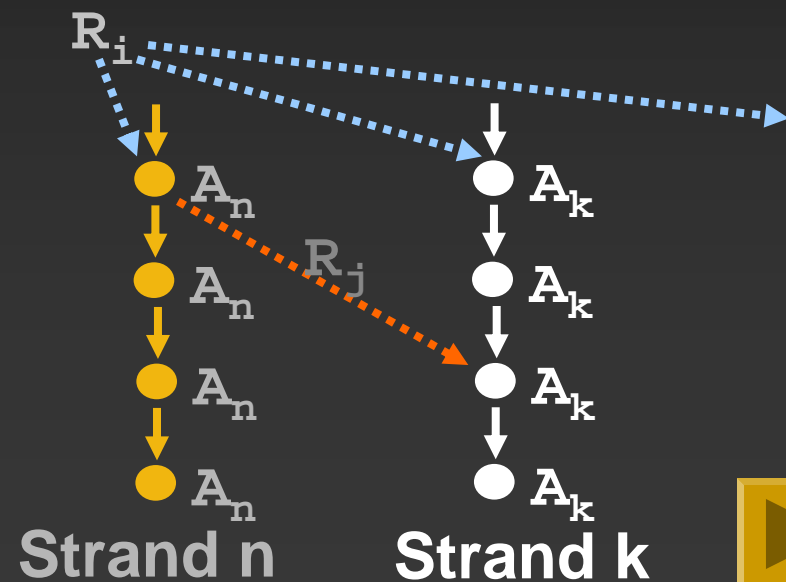
- Register value usage locality
 - Most register values are used just once (>70%)
 - Small number of scratchpad registers can handle those temporary values
- Dependent instructions can't issue in parallel
 - Unless value speculation techniques are used
 - Why not *exploit* dependences if not easily avoidable?
- Dynamic register dependence graph is not as complex as a 3-operand RISC ISA implies
 - 50 to 62% instructions have only one or zero inputs

Desired ISA Properties

- Separate registers
 - Global: values that live long or are used by multiple instructions
 - Local: values that are used once, quickly
- Express inter-instruction dependences and their lifetime more explicitly
- Close enough to the traditional ISAs to allow efficient dynamic binary translation

ILD P ISA

- Accumulators (A_n): temporary values inside dependent chains of instructions (*strands*)
- General-purpose registers (GPRs; R_j): values that live long or are used by multiple strands
- Offloading temporary values to accumulators reduces pressure on GPRs



Example: 164.zip

RTL notation

```
L1: R3 ← mem[R16]
    R17 ← R17 - 1
    R16 ← R16 + 1
    R3 ← R1 xor R3
    R1 ← R1 >> 8
    R3 ← R3 and 0xff
    R3 ← 8*R3 + R0
    R3 ← mem[R3]
    R1 ← R3 xor R1
    P ← L1, if (R17 != 0)
```

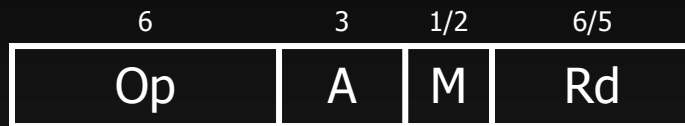
L2:

ILDP translation

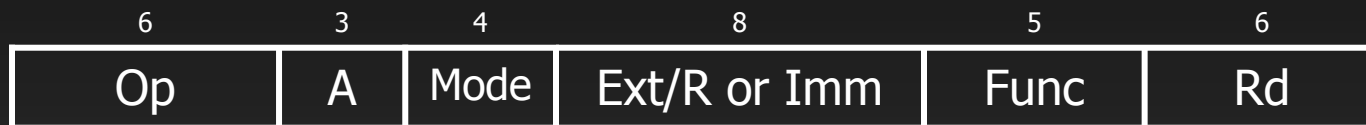
```
L1: R3(A0) ← mem[R16]
    R17(A1) ← R17 - 1
    R16(A2) ← R16 + 1
    R3(A0) ← A0 xor R3
    R1(A3) ← R1 >> 8
    R3(A0) ← A0 and 0xff
    R3(A0) ← 8*A0 + R0
    R3(A0) ← mem[A0]
    R1(A3) ← R3 xor A3
    P ← L1, if (A1 != 0)
    P ← L2
```

L2:

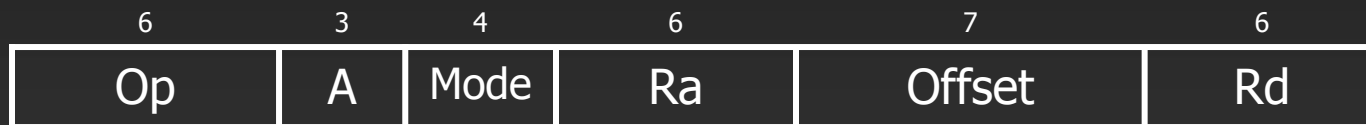
ILD P ISA Formats



Short format



Operate format



Memory format

- Mode and ext field:

$Rd/A \leftarrow A \text{ op } R$ $Rd/A \leftarrow R \text{ op } A$ $Rd/A \leftarrow A \text{ op } Imm$

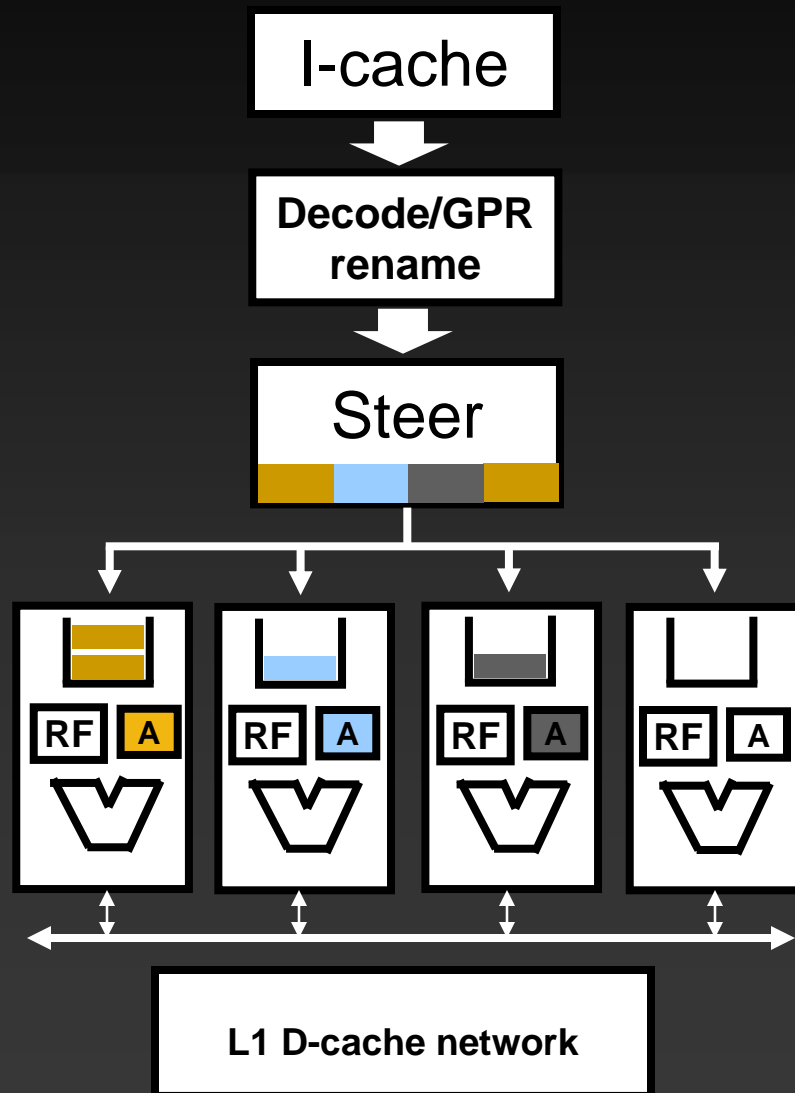
$Rd/A \leftarrow R \text{ op } Imm$ (not supported)

- End-of-strand bit: Mode[22]

Contents

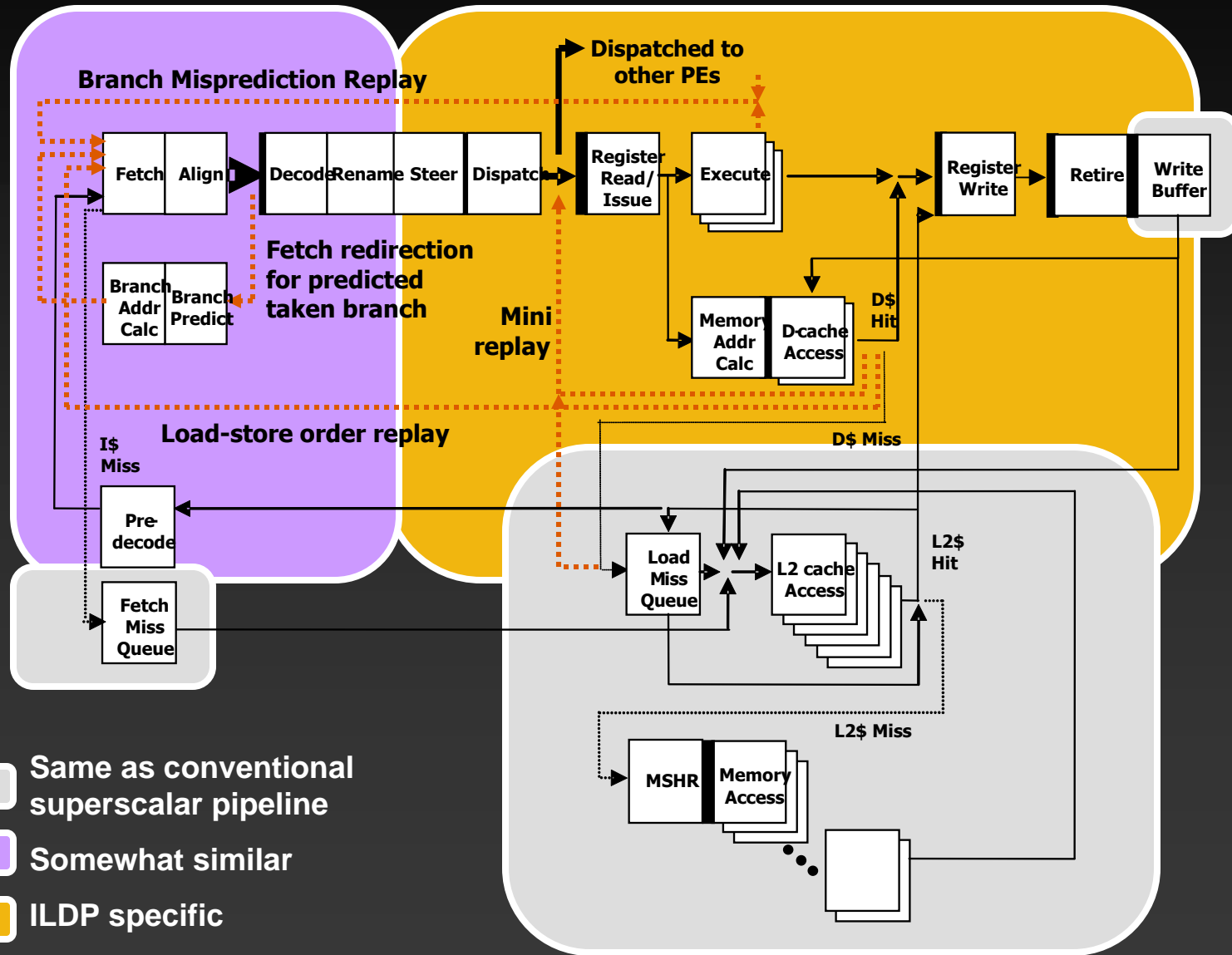
- Introduction
- ILDP Instruction Set Architecture
- **ILDP Microarchitecture**
- Dynamic Binary Translation for ILDP
- Control Transfer Support Mechanisms
- Evaluation
- Conclusions

Overall Microarchitecture



- Fetch mechanism is optimized for multiple sequential basic blocks
- Only GPRs are renamed to physical registers (reduce rename bandwidth)
- Accumulators are renamed as instructions are steered (dependence-based steering)
- Out-of-order dispatch (no wakeup)
- In-order instructions issue (reduce issue logic complexity)
- Non-pipelined functional units
- Replicated register file & L1 D-cache (reduce number of ports)
- Value communication takes additional cycles

ILDP Pipeline



ILDP Operand Capture Model

**Physical register file
(Capture after issue)**

Disp	Issue	Reg read	Addr Calc	D\$ 1	D\$ 2
-------------	--------------	-----------------	------------------	--------------	--------------

+ Single source of register operands

– Load latency speculation generates mini-replays

**Reorder buffer
(Capture before issue)**

Disp/ ROB read	Bypass/ Issue	Exec	Reg write	Retire
-------------------------------	--------------------------	-------------	----------------------	---------------

+ No load latency speculation

– Multiple sources, depends heavily on bypass (CAM)

**ILDP Physical register file
(Capture before issue)**

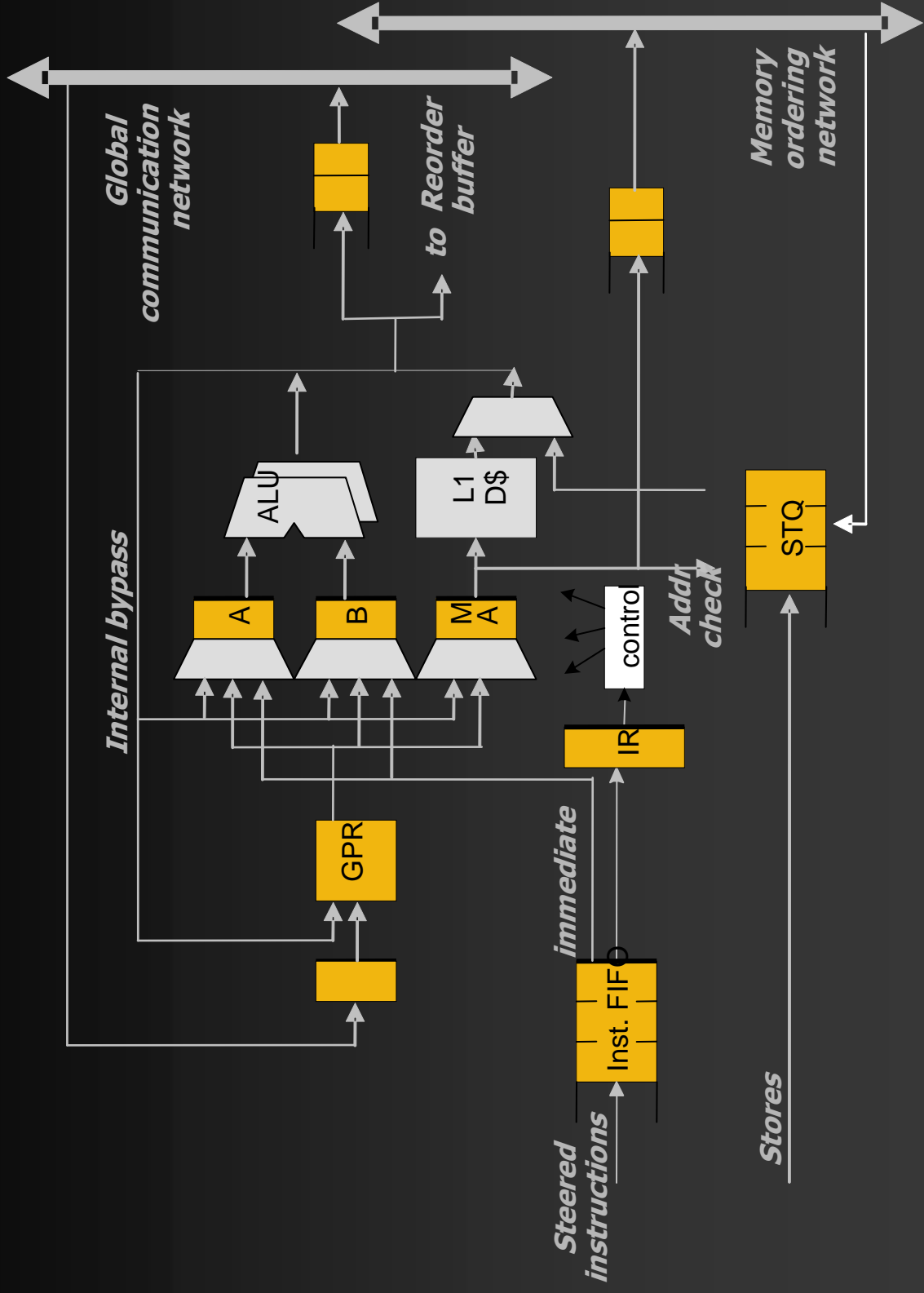
Disp	Reg/Acc read/ Issue	Exec	Reg write	Retire
-------------	------------------------------------	-------------	----------------------	---------------

+ Reduced dependence on bypass

+ No load latency speculation

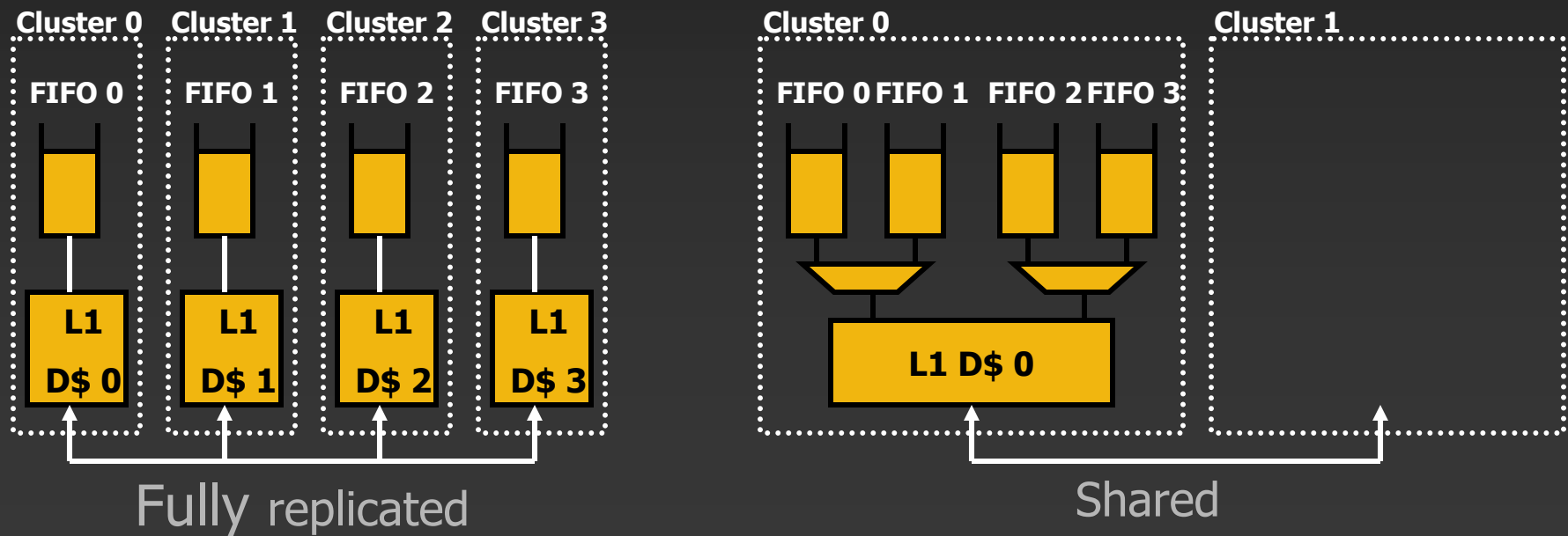
- ILDP pipeline combines the best of both models

Processing Element



D-cache Subsystem

- No silver bullet here
 - ILDP microarchitecture was built around register dependences
 - Partitioning L1 D cache is hard in ILDP microarchitecture
 - Use more traditional sharing



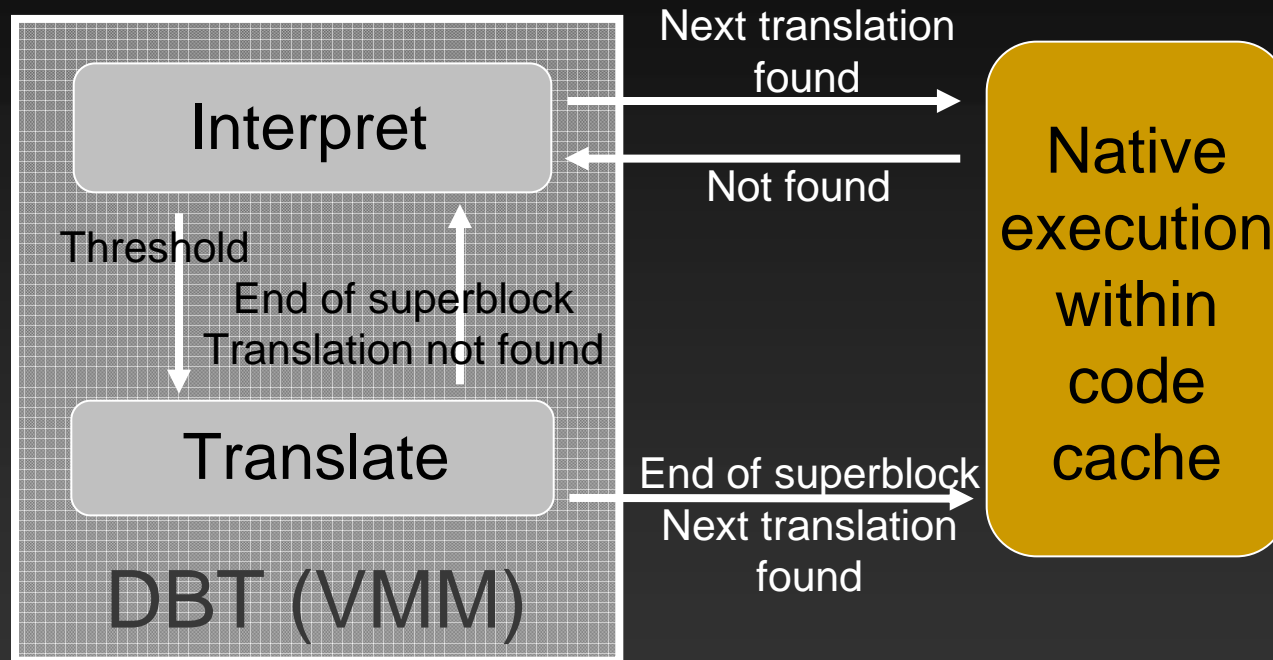
Memory Instruction Ordering

- Memory dependence speculation is already an important technique
- Distributed PEs naturally lead to out-of-order (speculative) memory instruction issue
- Replicated store queues (and a shared load queue) enforce ordering
- Essentially the same mechanism used in modern out-of-order processors
- Binary translation does not change instruction ordering – hardware does all scheduling

Contents

- Introduction
- ILDP Instruction Set Architecture
- ILDP Microarchitecture
- **Dynamic Binary Translation for ILDP**
- Control Transfer Support Mechanisms
- Evaluation
- Conclusions

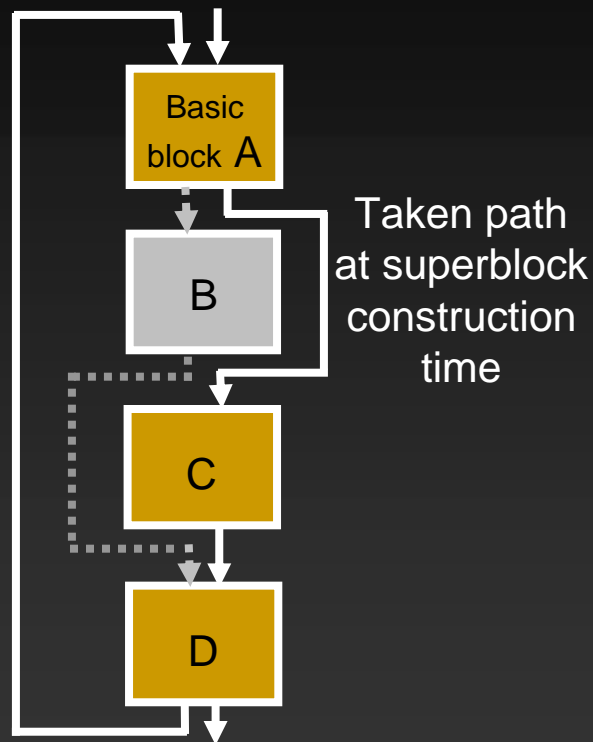
Dynamic Binary Translation Framework



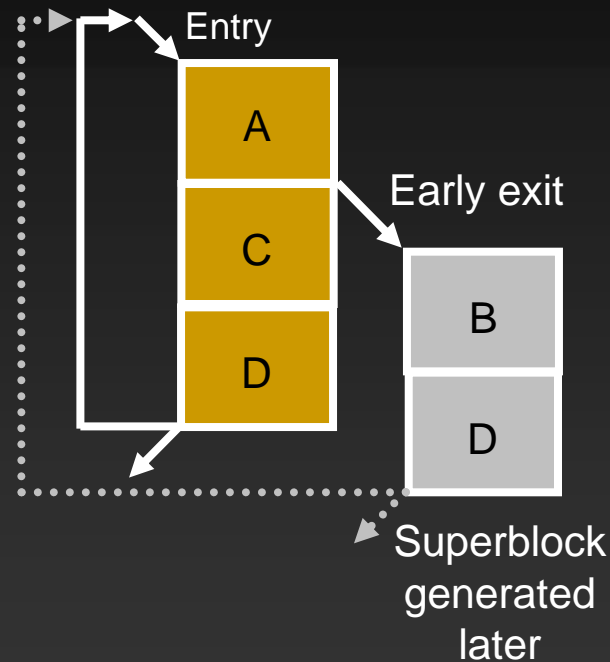
- Standard hot-path based translation
 - Modified Most Recently Executed Tail (HP Dynamo)

Translation Unit: Superblock

Original program layout



Generated superblocks

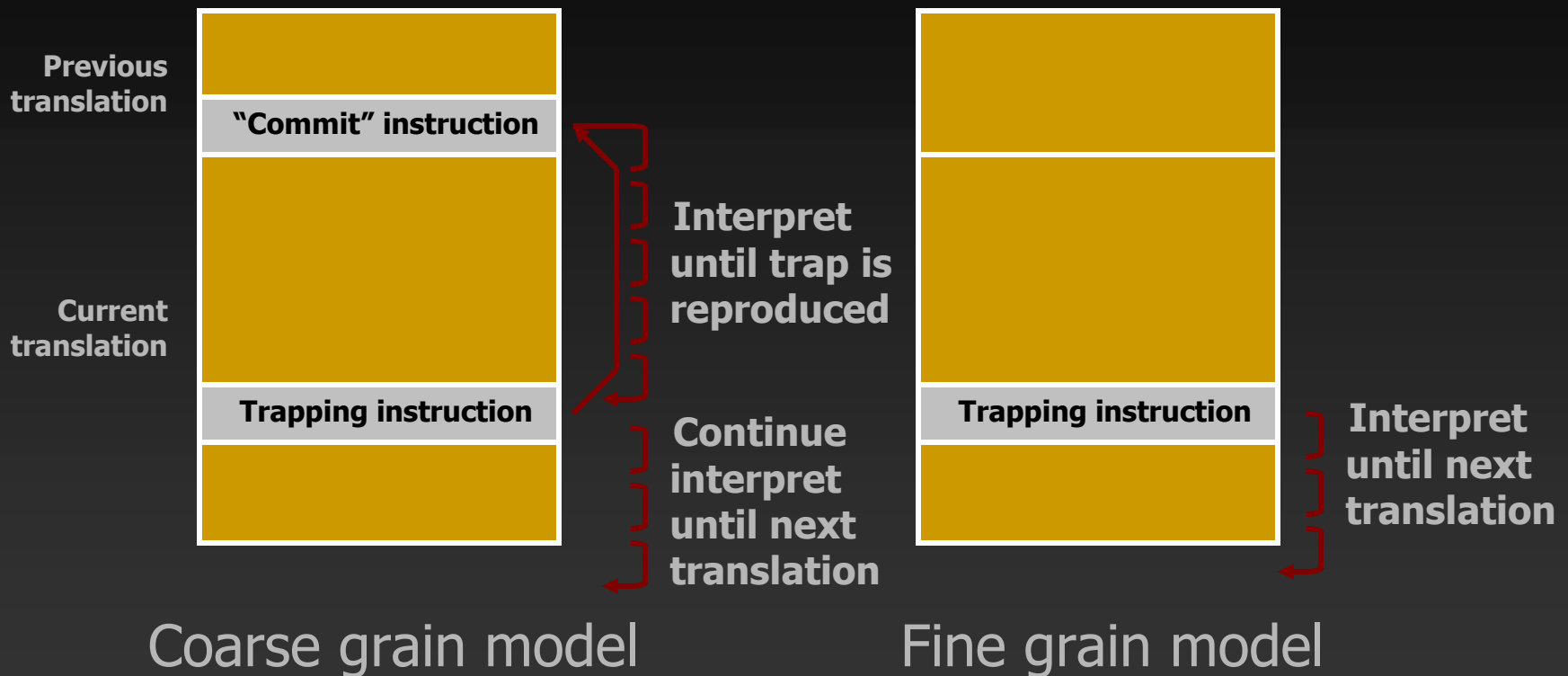


- Basic blocks are put in the most frequently executed order: dynamic code re-layout

Precise State Recovery

- Must restore all V-ISA registers
 - Including the ones that were assigned to accumulators
- Keep the destination GPR identifier in translated instructions
 - Separate register file for maintaining architected state (off the pipeline critical path)
 - Instruction format distinguishes two GPR writes
 - Values that are used by other strands → physical & arch. RF
 - Values only necessary for trap recovery → arch. RF only

State Maintenance Granularity



- Coarse grain model (Transmeta) can be "painful" in pathological cases
- ILDP DBT does *not* reschedule instructions → Fine grain model



Dynamic Code Expansion

- One of the biggest problem in any dynamic translation/optimization systems
- ILDP DBT should do better than VLIW
- Efforts
 - Use 16-bit format to make up for instruction count expansion
 - Keep one-to-one mapping
 - Exploit known V-ISA idioms
 - Use special instructions/hardware (especially control transfer support)

Translation Algorithm

- No aggressive optimizations/instruction rescheduling
 - Identify strands and re-map intra-strand temporary values to accumulators
 - Still provides code re-layout
- Dependence/usage identification (linear scan)
 - Local/no-user values → accumulator
 - Live-in/live-out/communication global values → GPR
- Strand formation: identify lifetime of dependences
 - Zero local input → start a new strand
 - One local input → follow the producer's strand
 - Two local inputs → one of two strands is terminated

Contents

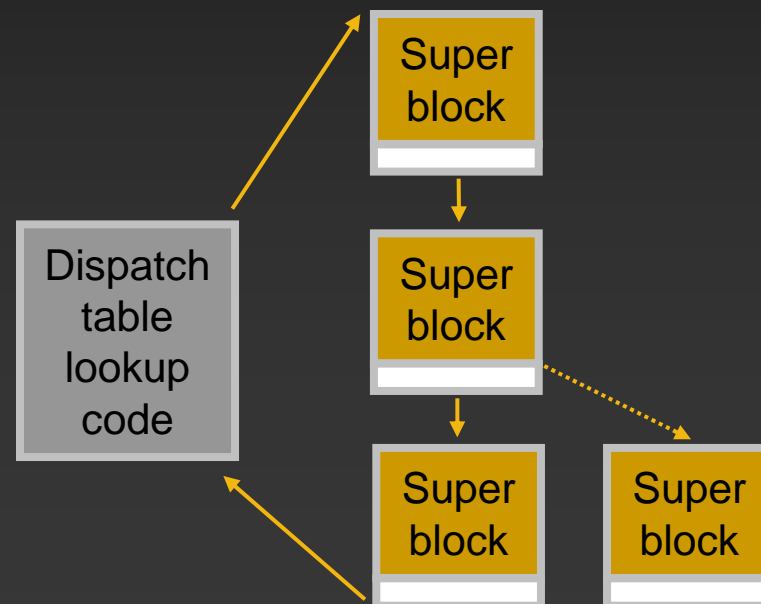
- Introduction
- ILDP Instruction Set Architecture
- ILDP Microarchitecture
- Dynamic Binary Translation for ILDP
- **Control Transfer Support Mechanisms**
- Evaluation
- Conclusions

Reality of Dynamic Code Layout

- Dynamo: 6% speed up just by dynamic code re-layout
 - Accounts for about half of total speed up
- Not true in modern pipeline with BTB
 - Instead, performance loss
 - Specialized hardware support mechanisms help
- For evaluation, a same-ISA code cache system was modeled

PC Translation Problem in Code Caches

- Translated code cache PC (TPC) is different from Source binary PC (SPC)
 - Need branch/jump target address translation
 - (Direct) branches are easier; target address is fixed



Register Indirect Jumps

- Target address (SPC in a register) can change
 - SPC needs to be translated at run-time
- Conventional solution: software prediction using superblock construction-time target address

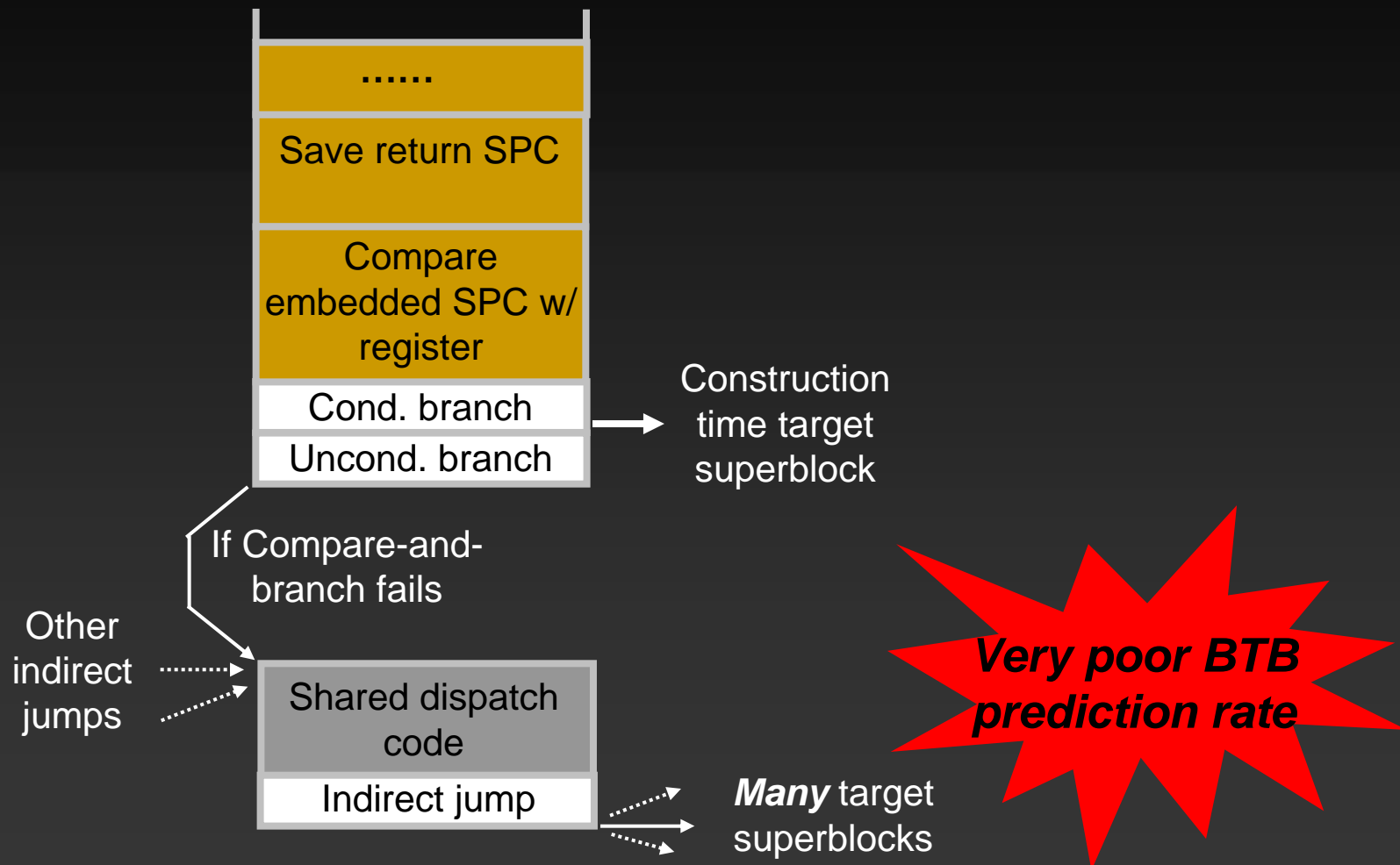
```
If Rx == #target_SPC goto #target_TPC
```

```
Else dispatch_table_lookup(Rx)
```

- One of the biggest overhead in code caches
 - Compare-and-branch: 6 instructions
 - Hash table lookup: 15 instructions } in Dynamo x86



Why Conventional Chaining Performs Poorly

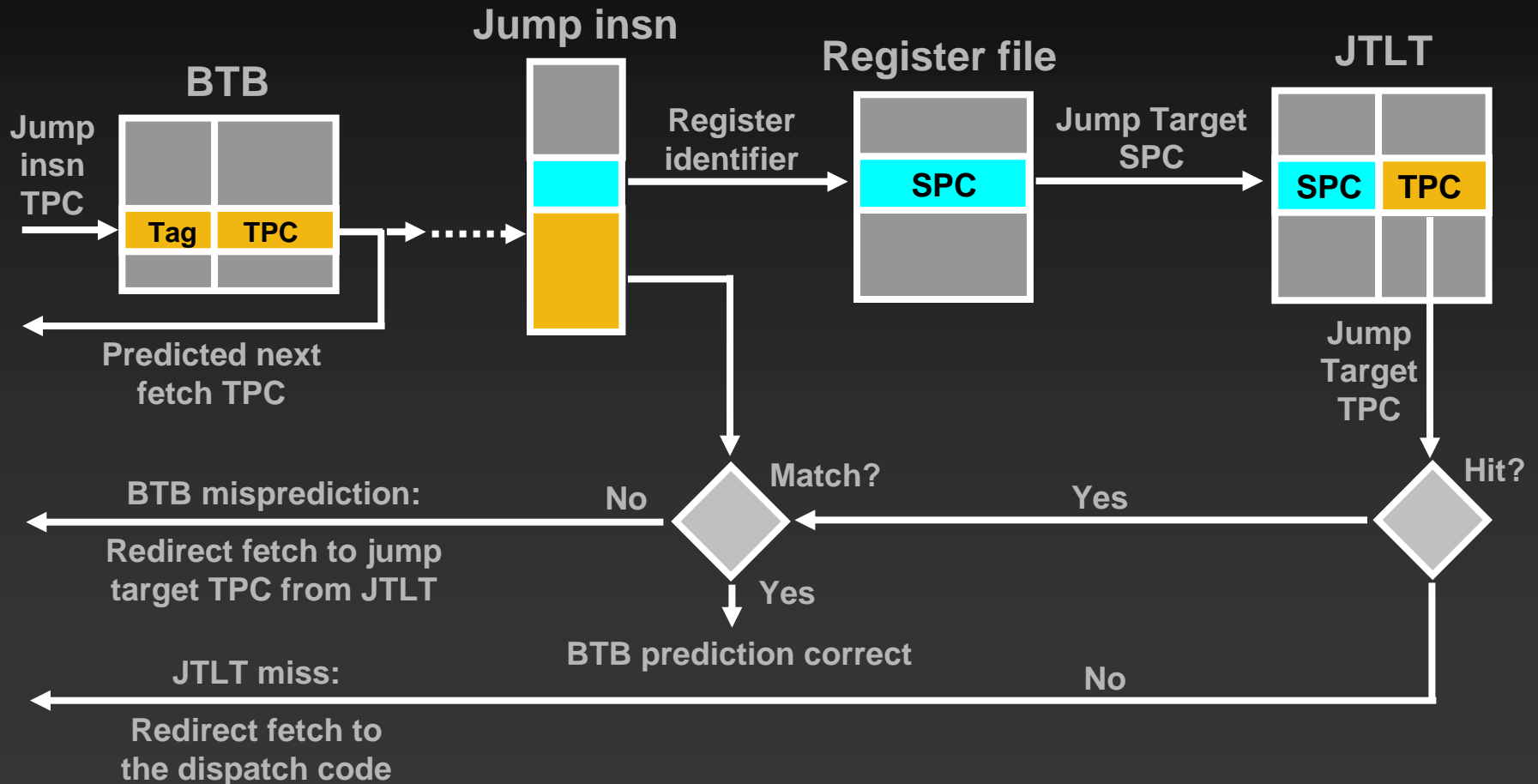


Translated code for a indirect function call instruction

Jump Target-address Lookup Table

- A hardware cache of dispatch table entries
- Similar to TLBs in virtual memory
 - Managed by software
 - If hit, always correct
- A jump is not translated to compare-and-branch code sequence
 - Reduce code expansion

Jump Target-address Lookup Table

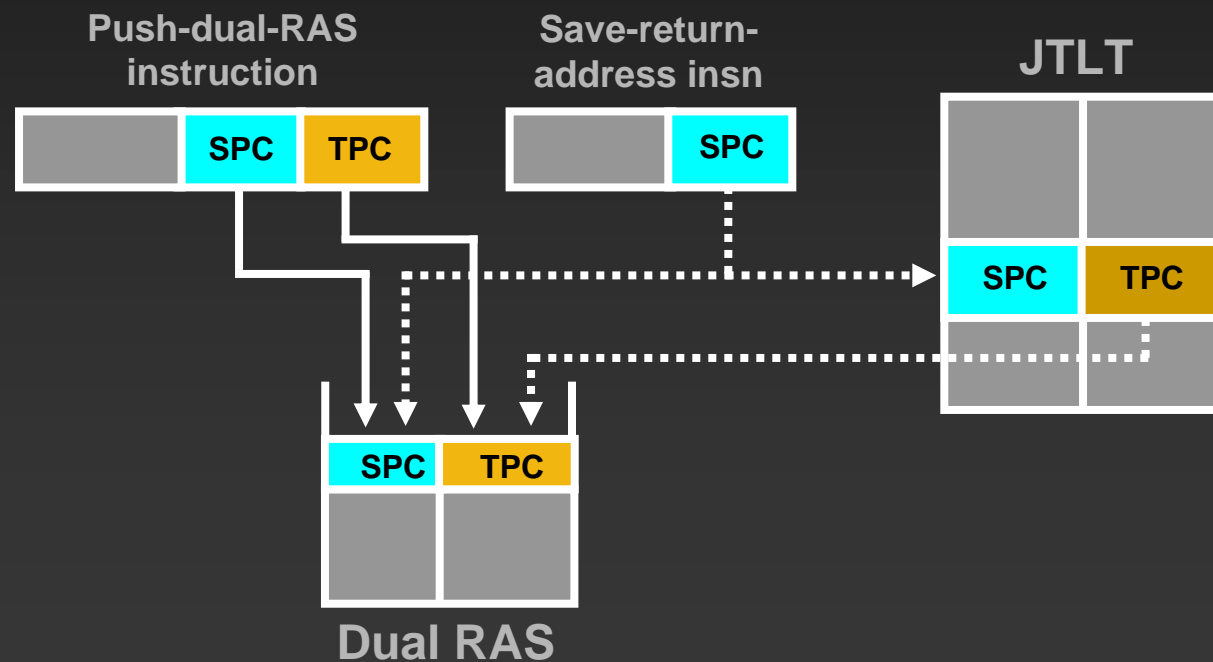


Conventional RAS Cannot Be Used

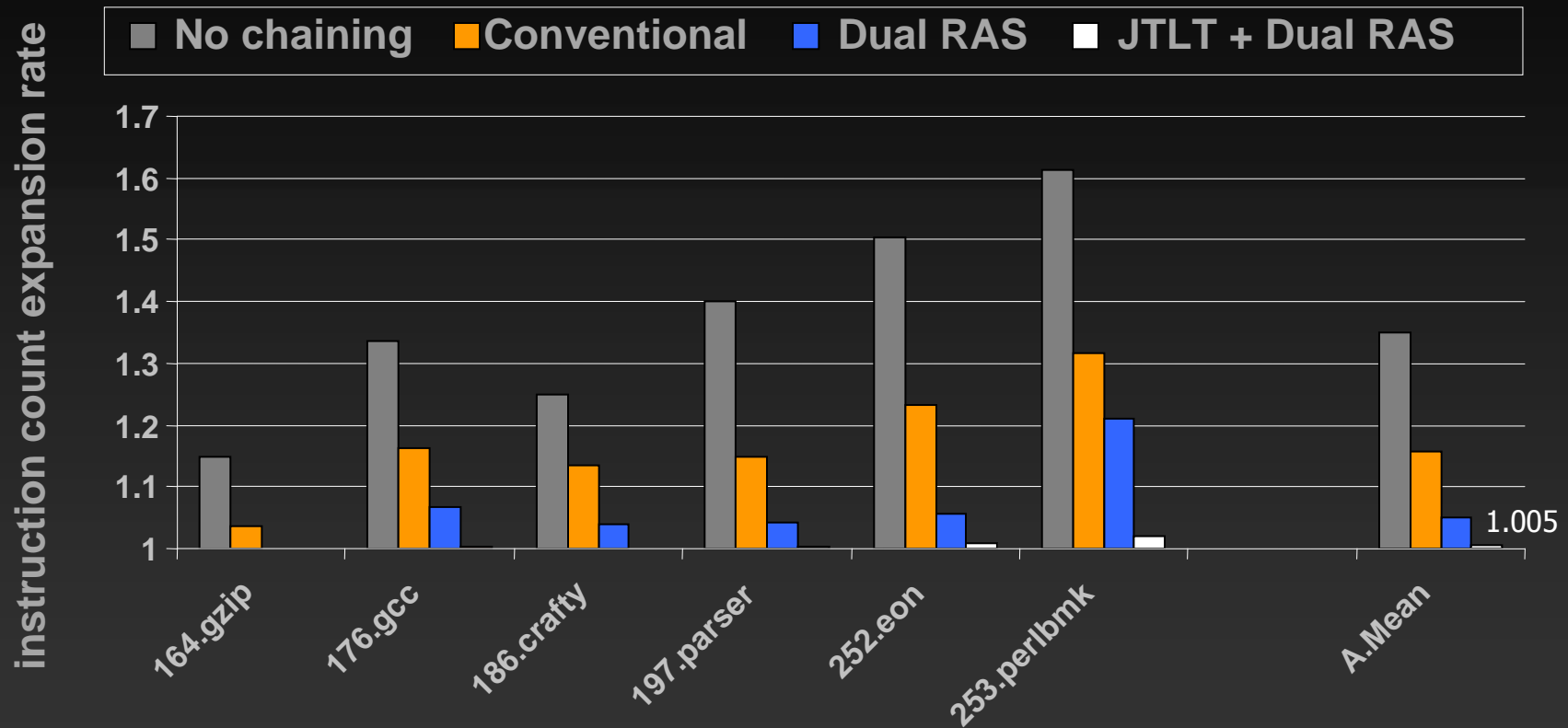
- Push: the next TPC after a function call instruction is NOT the return TPC
- Pop: how to verify the popped TPC is correct?
 - Only return SPC is available in register
 - Saving return TPC breaks precise exception semantics
- Nonetheless, a (highly accurate and simple) RAS-type prediction mechanism is desirable

Dual RAS

- Solution:
 - Push both SPC and TPC [Gschwind]
 - Use TPC for prediction, SPC for check

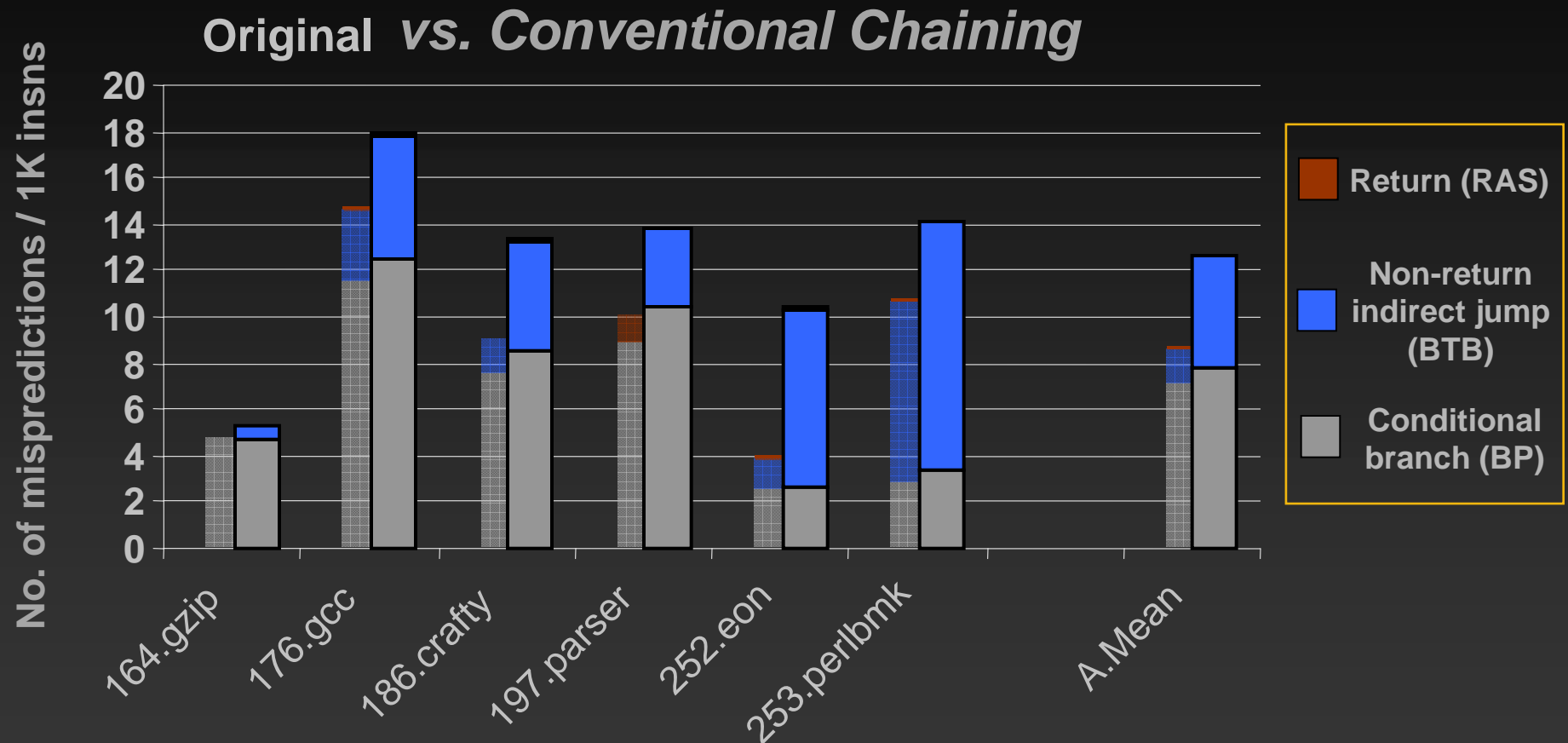


Dynamic Instruction Count Expansion



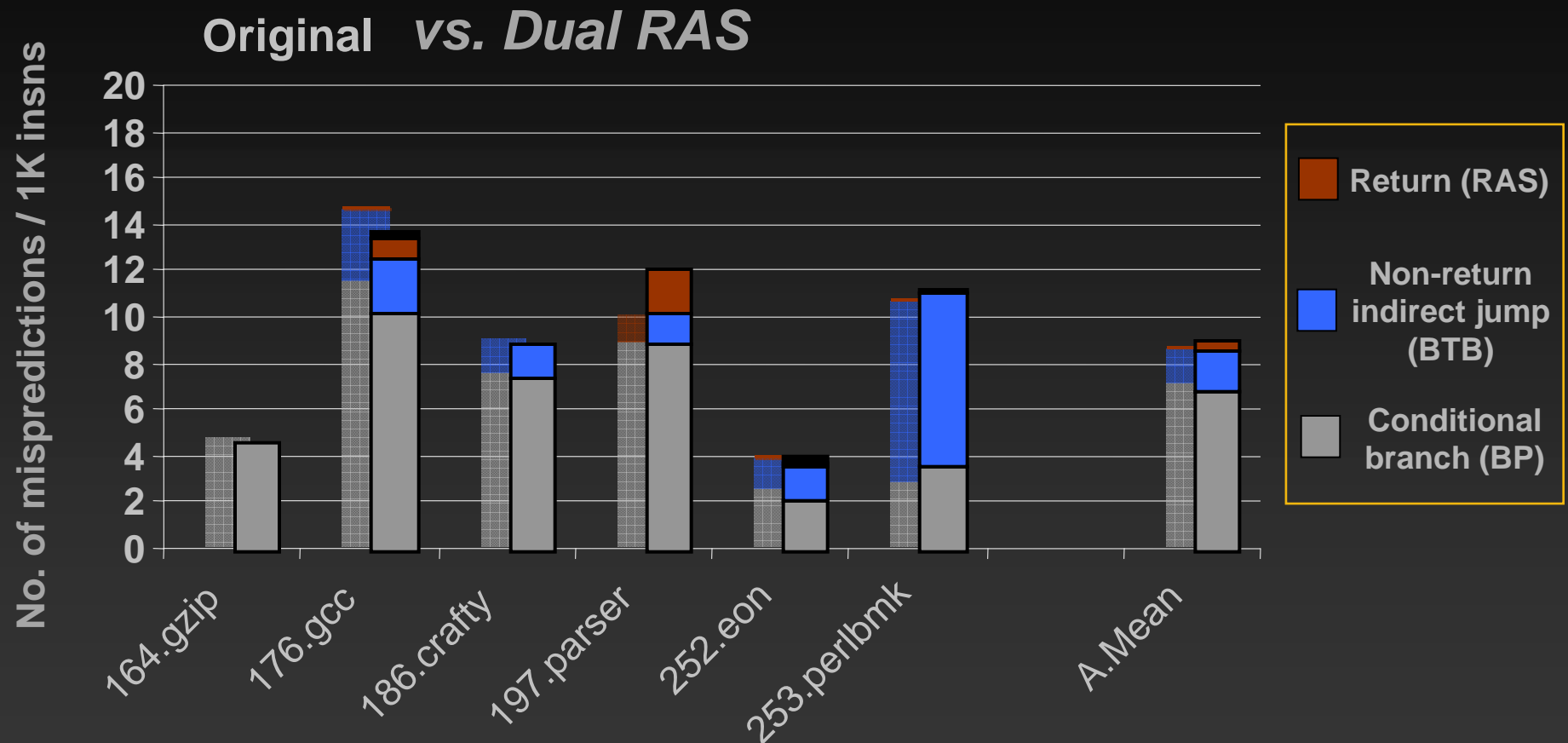
- Hardware mechanisms help reduce code expansion
- 2x increase in number of instructions between taken control transfers → improved fetch efficiency

Branch Prediction Performance



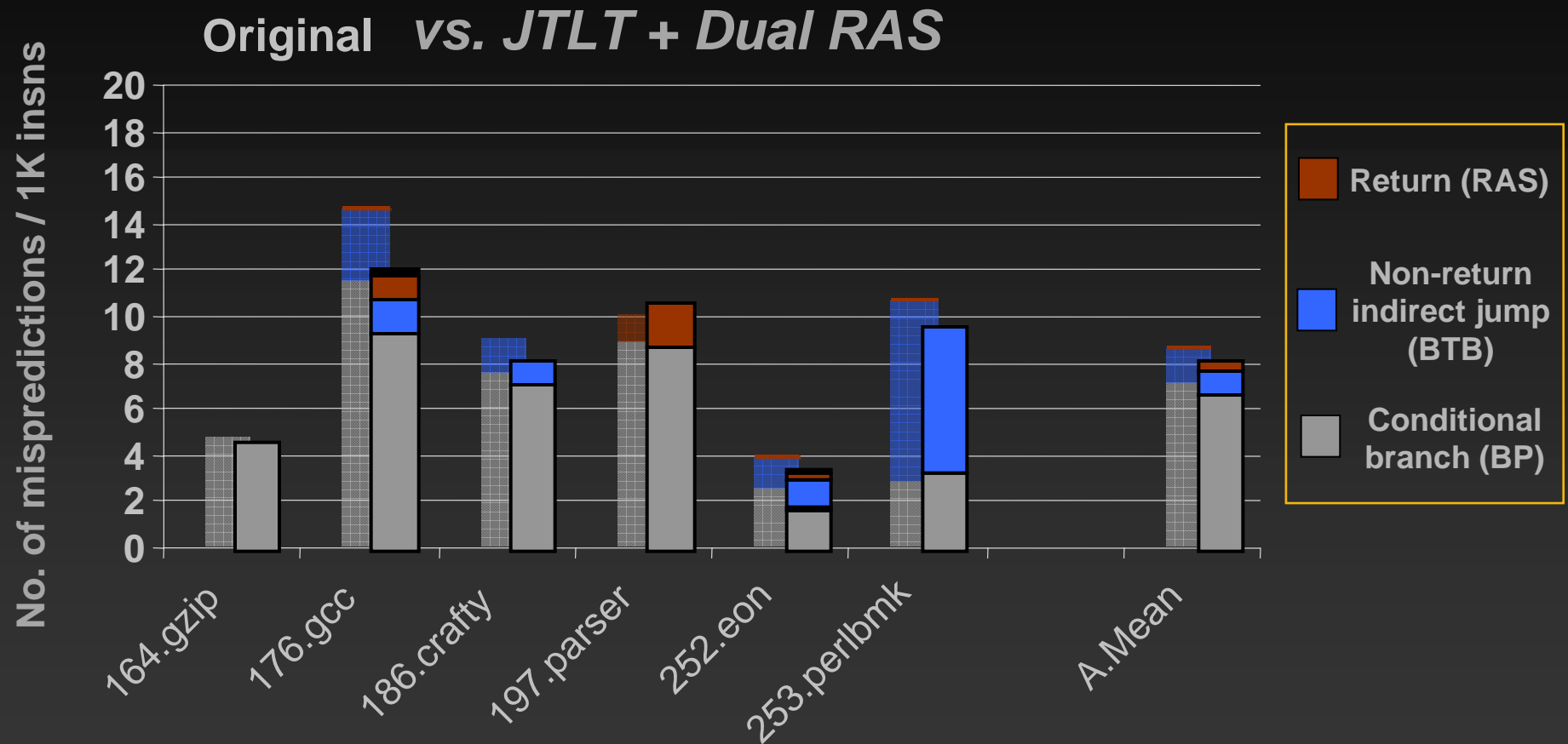
- Dual RAS has the biggest impact
- JTTL + Dual RAS achieves *fewer* mispredictions

Branch Prediction Performance



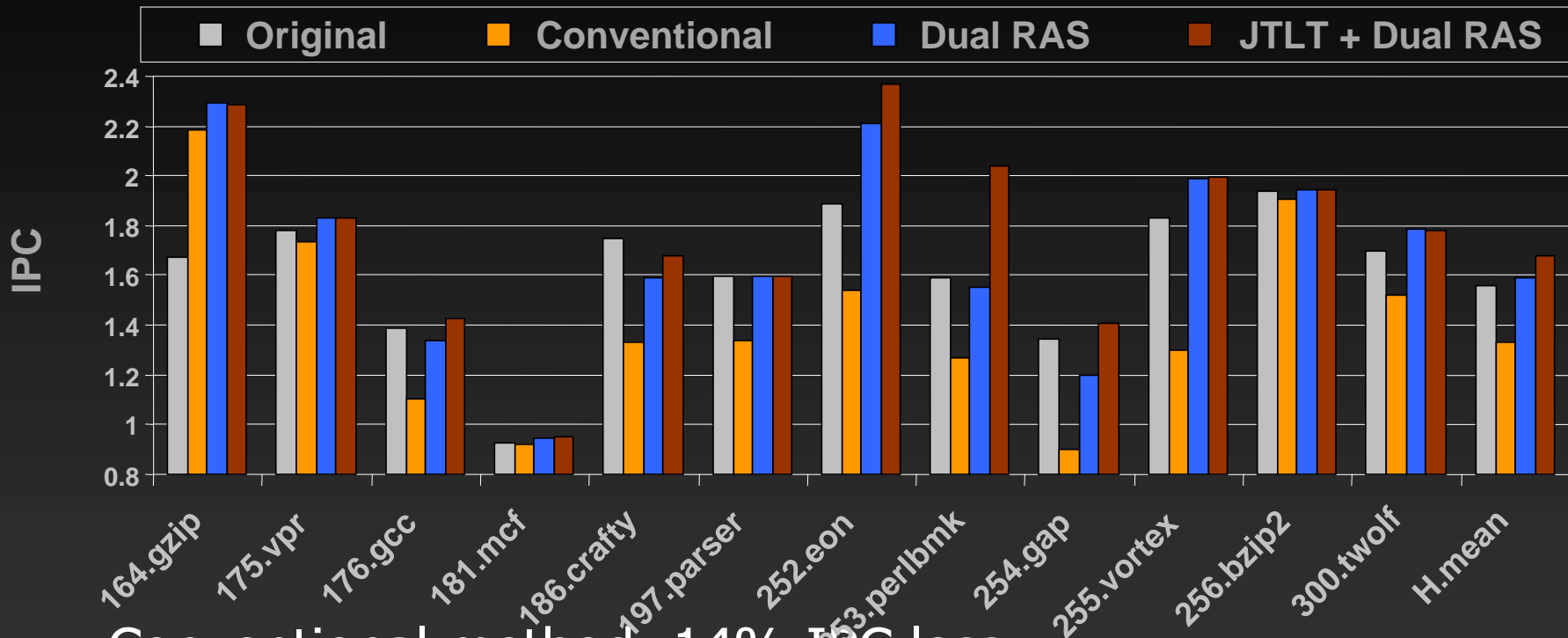
- Dual RAS has the biggest impact
- JTLT + Dual RAS achieves *fewer* mispredictions

Branch Prediction Performance



- Dual RAS has the biggest impact
- *JTLT + Dual RAS* achieves *fewer* mispredictions

Effect of Support Mechanisms



- Conventional method: 14% IPC loss
 - PA8000 stalls fetch at jumps → compare-and-branch conversion helps
- Dual RAS provides the most benefit
- Using both JTLT & dual RAS, 7.7% IPC improvement

Contents

- Introduction
- ILDP Instruction Set Architecture
- ILDP Microarchitecture
- Dynamic Binary Translation for ILDP
- Control Transfer Support Mechanisms
- Evaluation
- Conclusions

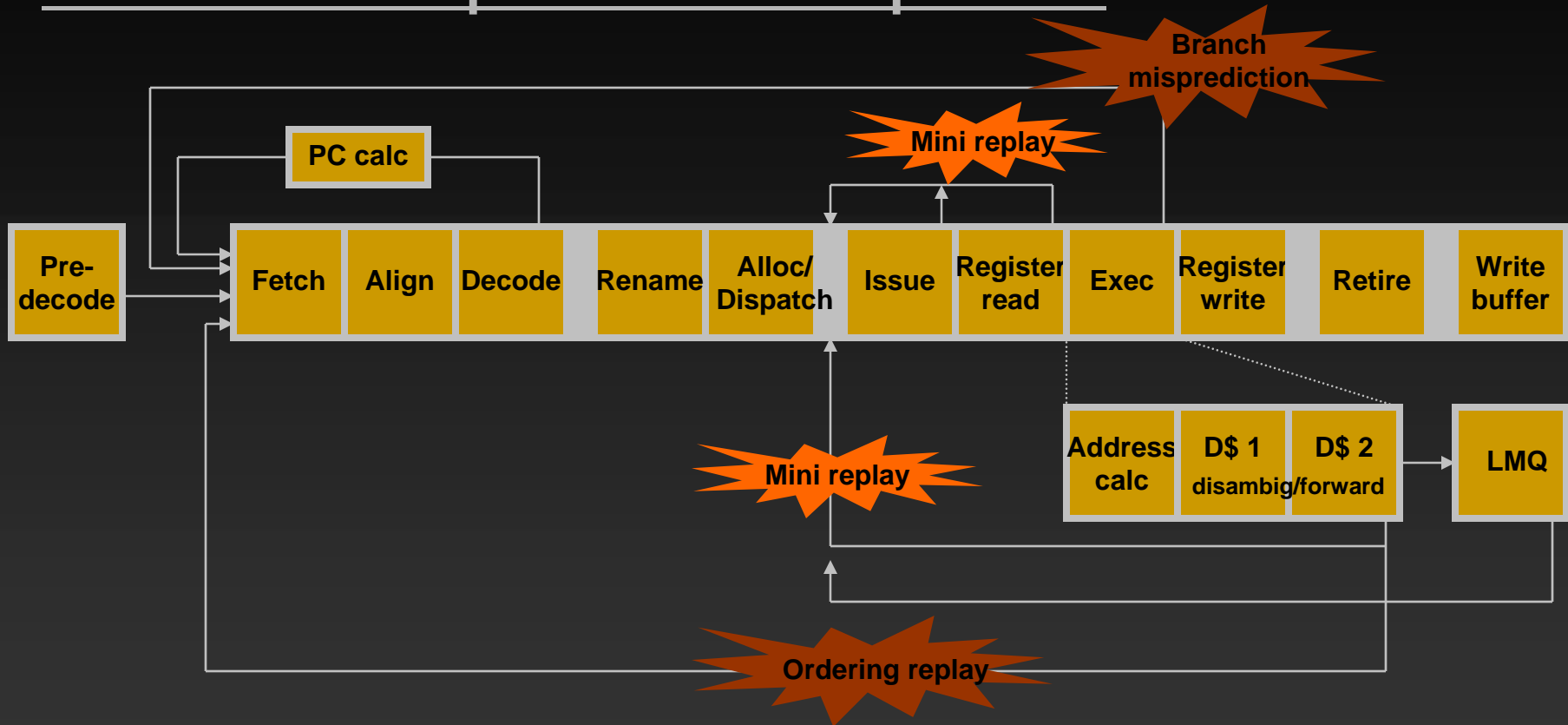
Evaluation Methodology

- DBT/microarchitecture simulator hybrids
 1. Baseline: native Alpha execution, modern superscalar
 2. ILDP: Alpha → ILDP ISA, ILDP microarchitecture
- SPEC CPU 2000 integer benchmarks
 - Alpha EV6, base optimization level (-O3)
 - **T**est or **T**rain input set, ran up to 4.3 billion instructions

Research Infrastructure

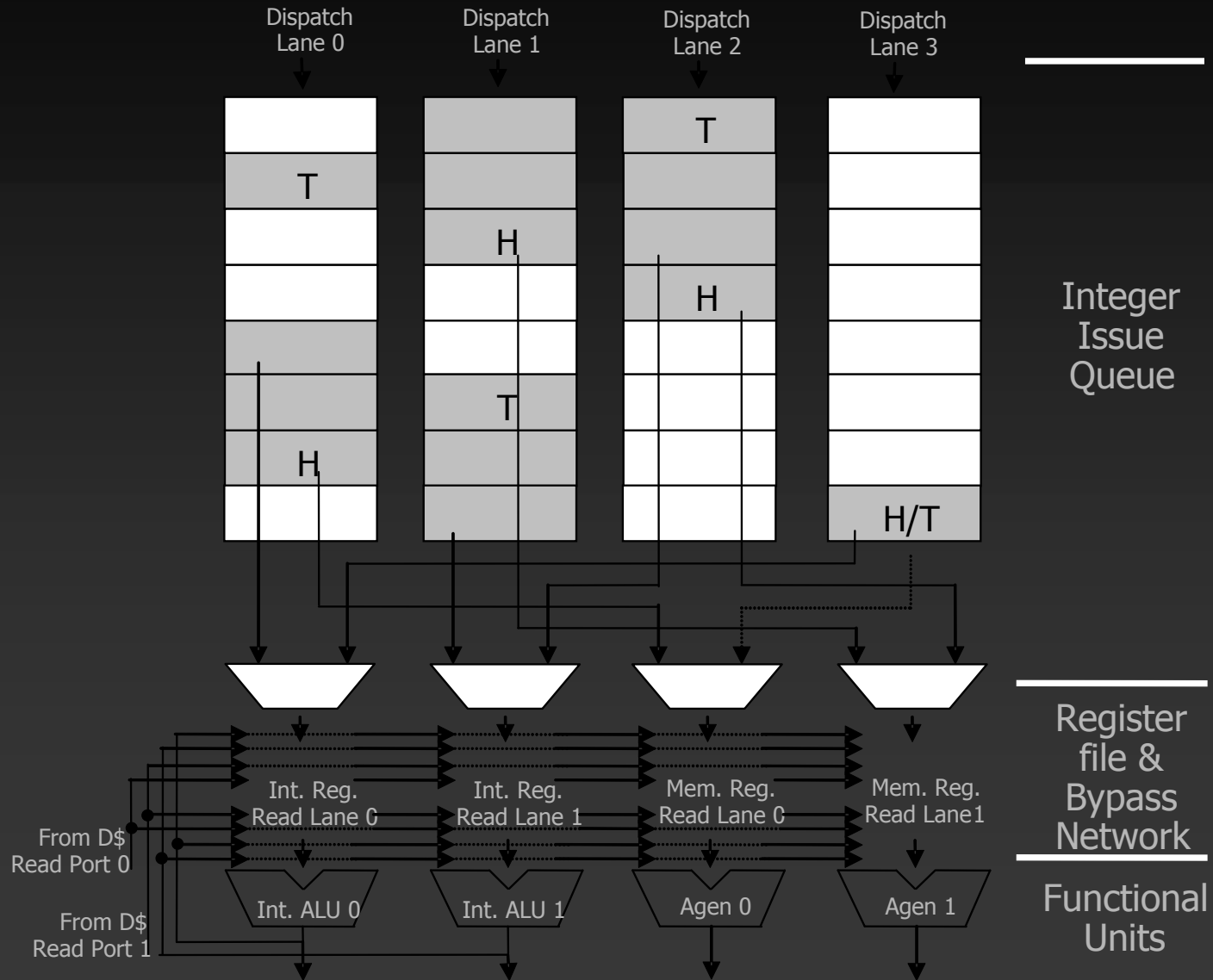
- Baseline simulator
 - A detailed 12-stage simulator built from (almost) scratch
 - Largely modeled after IBM POWER4 microarchitecture
 - Correct-by-design
 - Puts the calculated values in the bypasses/physical register file/caches
 - Prevents obscure timing bugs and “cheats” by design
 - “Golden” simulator verifies architected state at retire stage
 - ILDP timing simulator is built on top of baseline simulator
- Binary translation mechanism
 - Emulate interpret/translate modes
 - Timing simulate native execution mode

Baseline Superscalar Pipeline



- Instructions and data *values* actually flow through the pipeline “lanes”
- The hardest part: Race conditions between caches, LMQ, WB, MSHR
- Out-of-order branch resolution is not easy to get right either

Baseline Issue Path



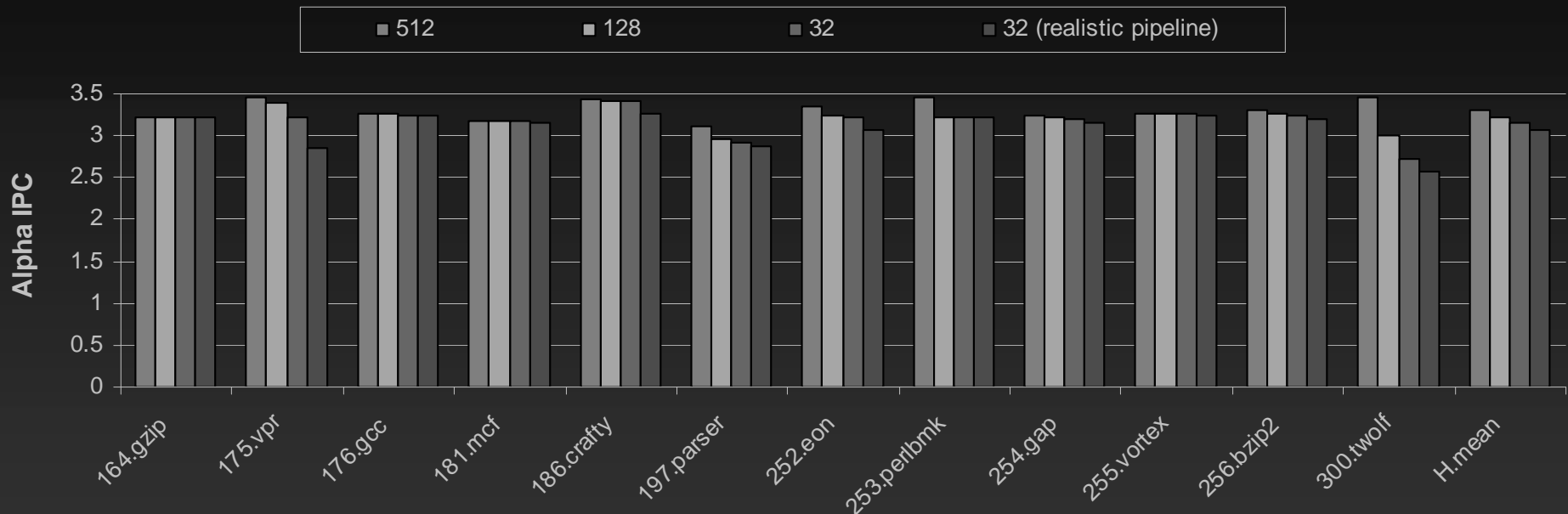
A Co-Designed Virtual Machine for ILDP

Simulated Microarchitectures

	Baseline superscalar	ILDP microarchitecture
Branch predictor	16K entry, 12-bit global history g-share predictor, up to 2 predictions per cycle, 16-entry RAS, 512-entry, 4-way BTB, (optional) 256-entry JTTL	
L1 I-cache	128-byte line size, direct-mapped, 32KB	
L1 D-cache	Write-through, write-no-allocate, 128-byte line size, 4-way, 32KB, 2-cycle, 2 read ports/1 write port	Same or 16KB, twice replicated
L2 cache	Unified, write-back, write-allocate, 128-byte line size, 4-way, 1MB, 8-cycle	
decode/retire BW	4 Alpha instructions	4 ILDP instructions
ROB size	128 Alpha instructions	128 ILDP instructions
Issue window size	4*8 (int/mem), 4*8 (fp)	4/6/8 (FIFO heads)
Issue bandwidth	2 int, 2 mem, 2 fp	4/6/8
Execution resources	2 int, 2 mem, 2 fp	4/6/8 fully symmetric functional units
Global bypass latency	0 cycle	2/1/0 cycle

Validation of Baseline Simulator

Effect of window size and pipeline inefficiencies (perfect disambiguation/BP/I\$/D\$)



- Steady state (perfect predictions/caches) IPC is close to 4
- The baseline design is complexity-effective

Validation of Baseline Simulator

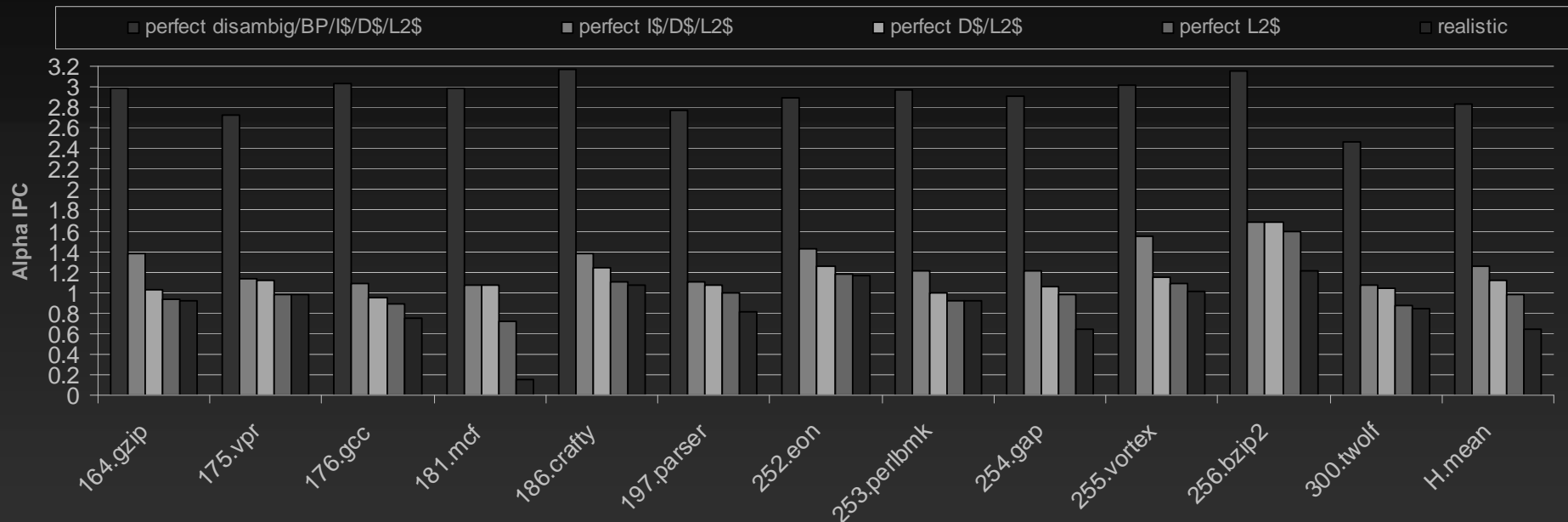
Effect of memory dependence speculation



- Steady state (perfect predictions/caches) IPC is close to 4
- The baseline design is complexity-effective
- Memory ordering speculation is important

Summary of Baseline Simulations

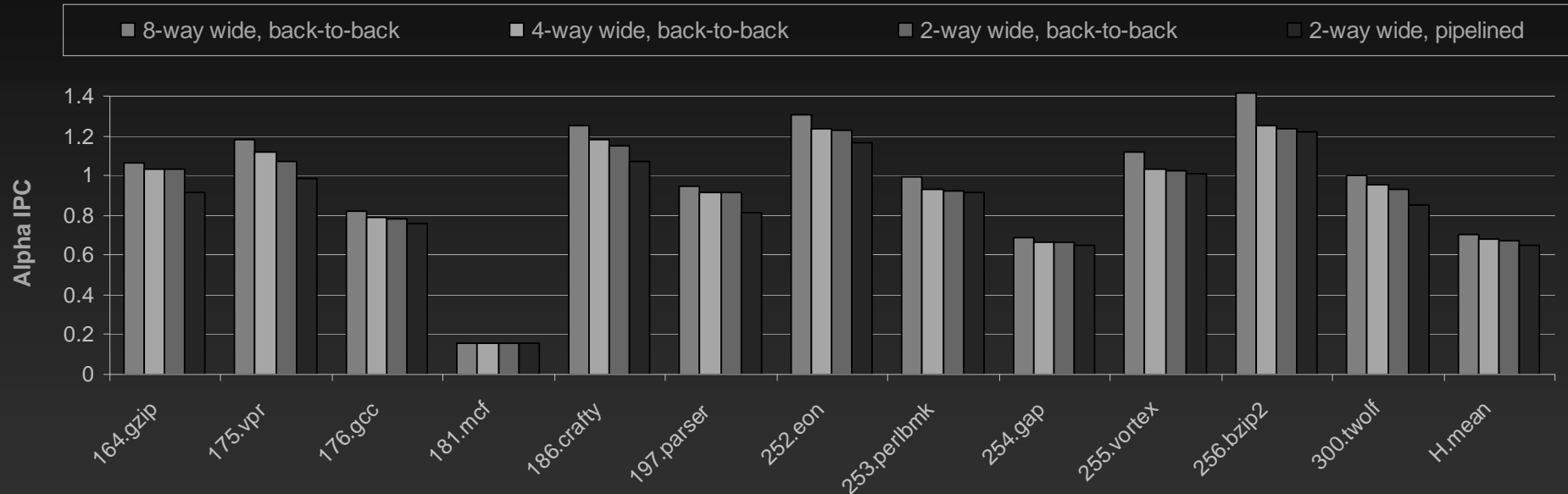
Effect of prediction and caches



- Branch mispredictions and L2 cache misses largely determine the overall performance

Summary of Baseline Simulations

Effect of issue logic



- Branch mispredictions and L2 cache misses largely determine the overall performance
- Issue mechanism is of secondary importance

DBT Characteristics

% instructions executed in Superblocks	Relative no. of dynamic instructions	% of copy instructions among extra instructions	Average instruction size	Relative code size
99.74%	1.33x	57.88%	3.59 B	1.32x

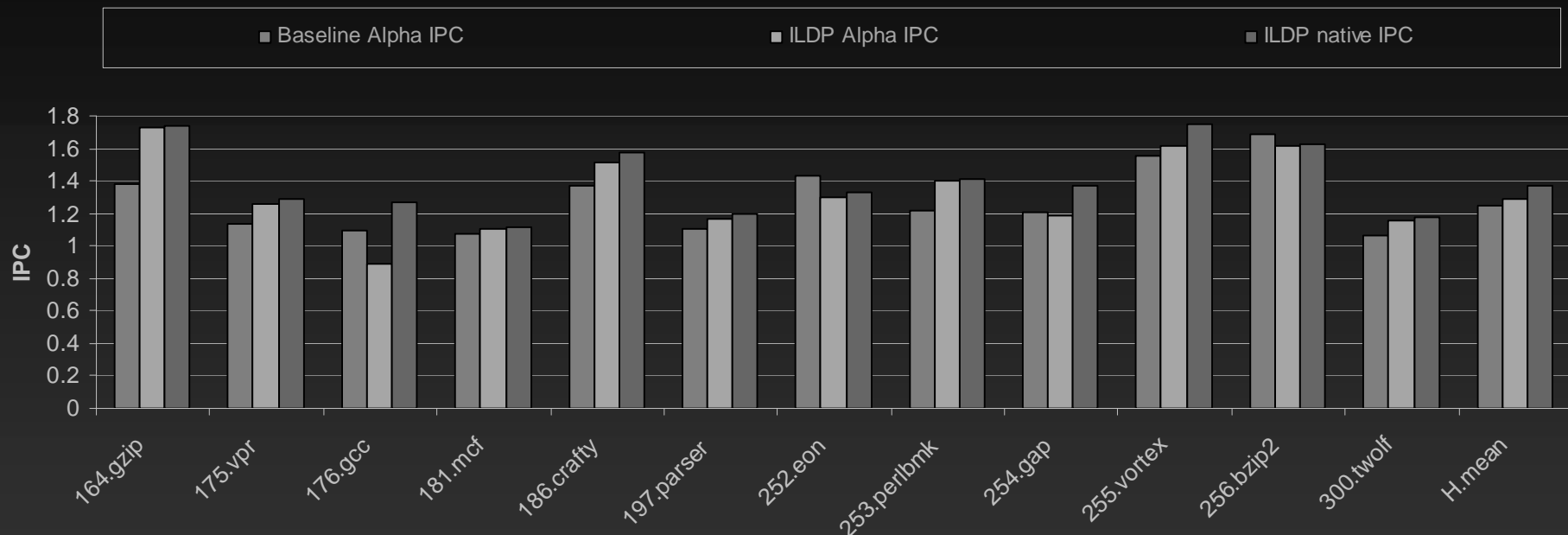
- Lack of `R op Imm` mode hurts
- If native execution rate is below 99%, interpretation overhead becomes significant
- Little code expansion
 - DAISY: 1.27x to 7.95 for SPECint95

Translation Overhead

- Profile the DBT/simulator using Atom tools
- Depends on both V-ISA and I-ISA
- 860 Alpha instructions to translate a single Alpha instruction
 - Simple nature of the DBT → no aggressive optimization
- DAISY report 4,000+ PowerPC instructions
 - VLIW → heavy optimization and scheduling

IPC Performance

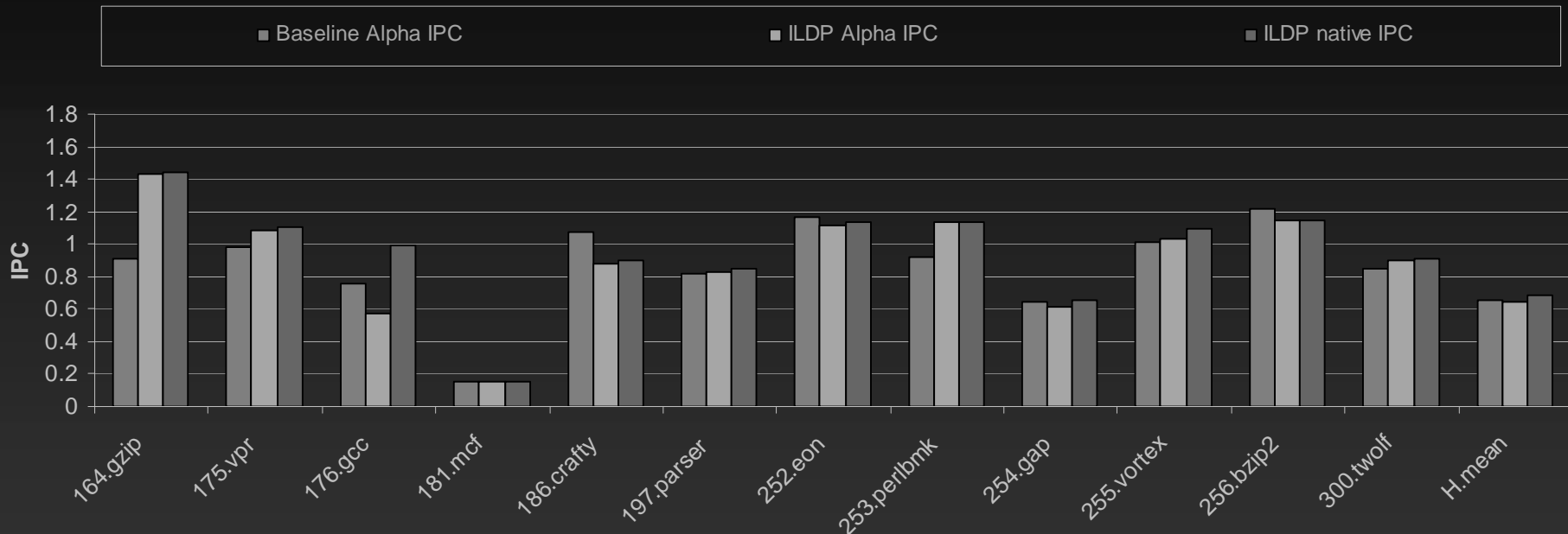
Perfect I\$/D\$/L2\$



- Alpha IPC includes interpretation/translation overheads
- ILDP system performs slightly better
 - Ignores clock frequency and design simplicity advantages

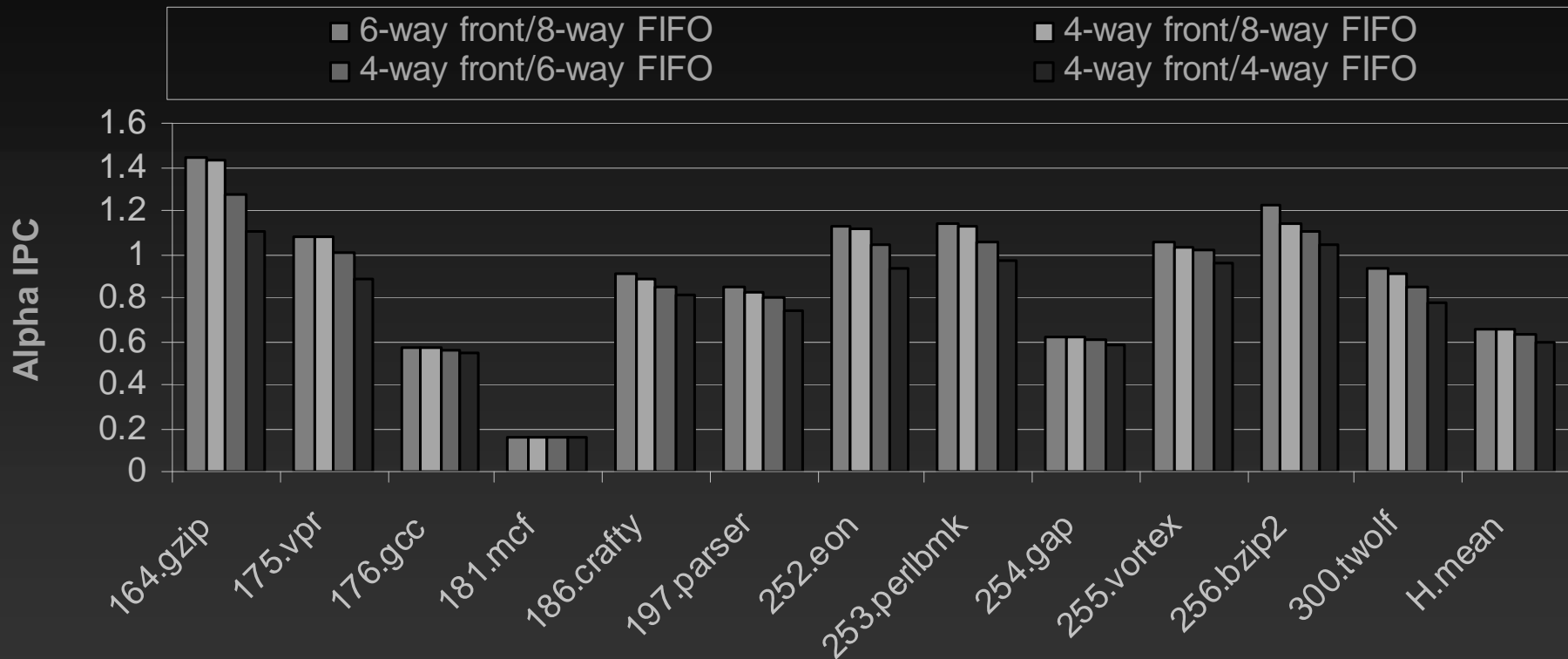
IPC Performance

Realistic I\$/D\$/L2\$



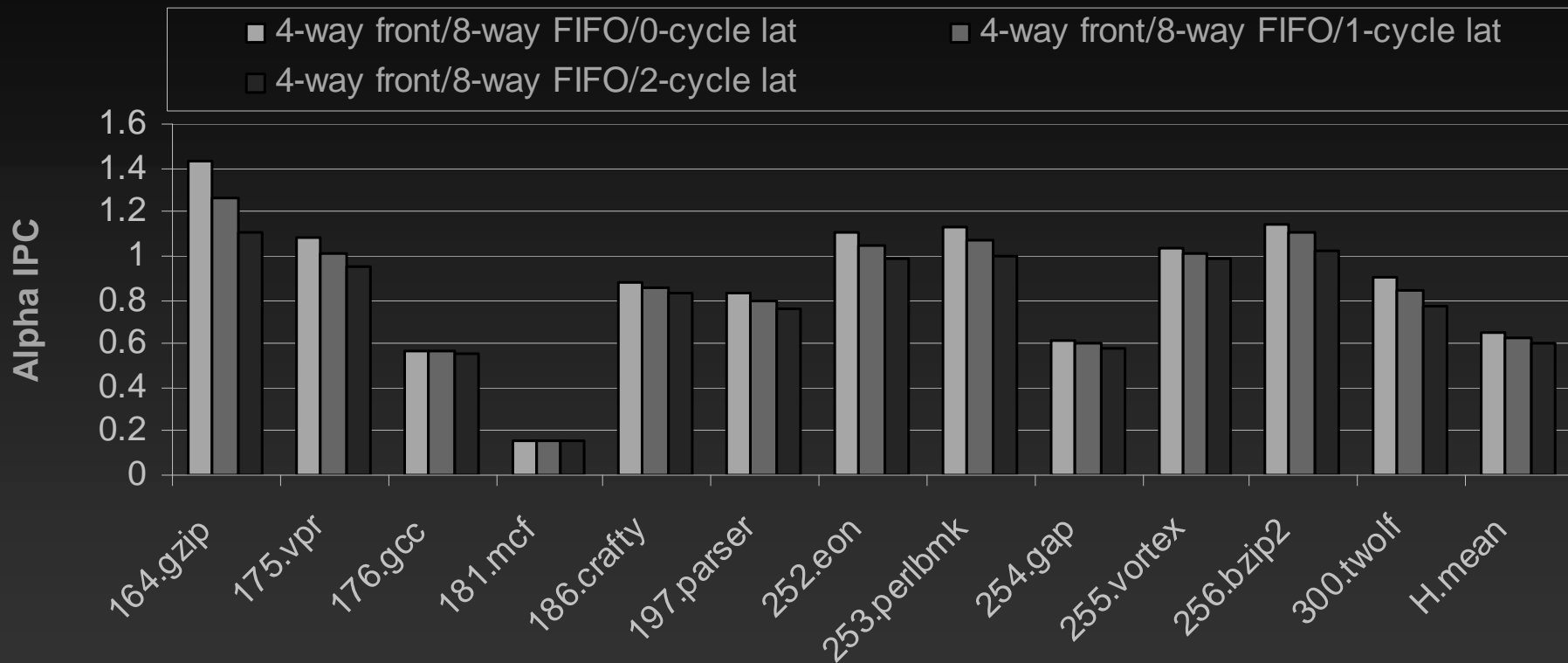
- Alpha IPC includes interpretation/translation overheads
- ILDP system performs slightly better
 - Ignores clock frequency and design simplicity advantages
- Performances eventually converge due to L2 cache misses

IPC Variation over Machine Parameters



- 6-way front-end makes up for dynamic instruction count expansion
- 6-PE holds up fairly well (3% IPC loss from 8-PE)

IPC Variation over Machine Parameters



- 6-way front-end makes up for dynamic instruction count expansion
- 6-PE holds up fairly well (3% IPC loss from 8-PE)
- 3.1% and 6.8% IPC loss from 1- or 2-cycle comm. latency

Complexity Estimates

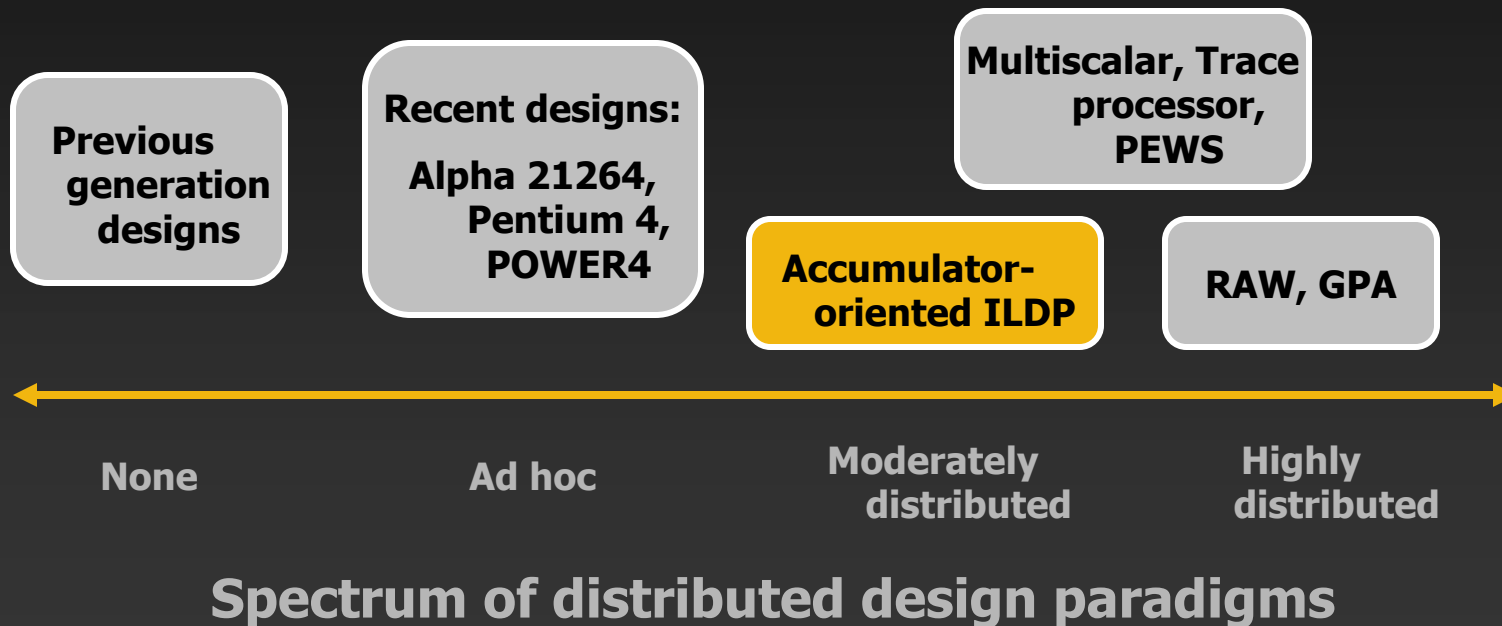
	Rename	Issue	Register File	Functional Units	Single cycle bypass
<i>4-wide superscalar</i>	12 R, 4 W ports	2/2/2 ooo 4*8 window	7 R, 5 W ports	pipelined	6 results 8 to 12 fan-out
<i>4-wide ILDP</i>	8 R, 4 W ports	1-way io x8	1 R, 4 W ports x8	sequential	1 results 4 fan-out

Contents

- Introduction
- ILDP Instruction Set Architecture
- ILDP Microarchitecture
- Dynamic Binary Translation for ILDP
- Control Transfer Support Mechanisms
- Evaluation
- Conclusions

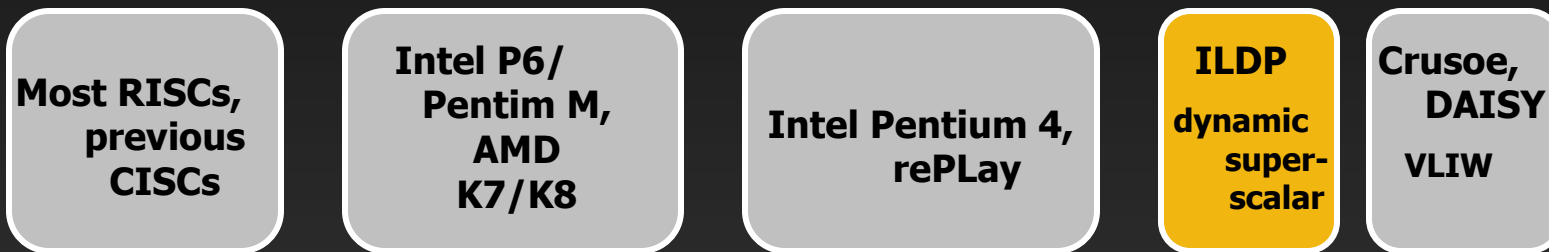
ILDP Design Points

- It's all about exploring new design trade-off points



ILDP Design Points

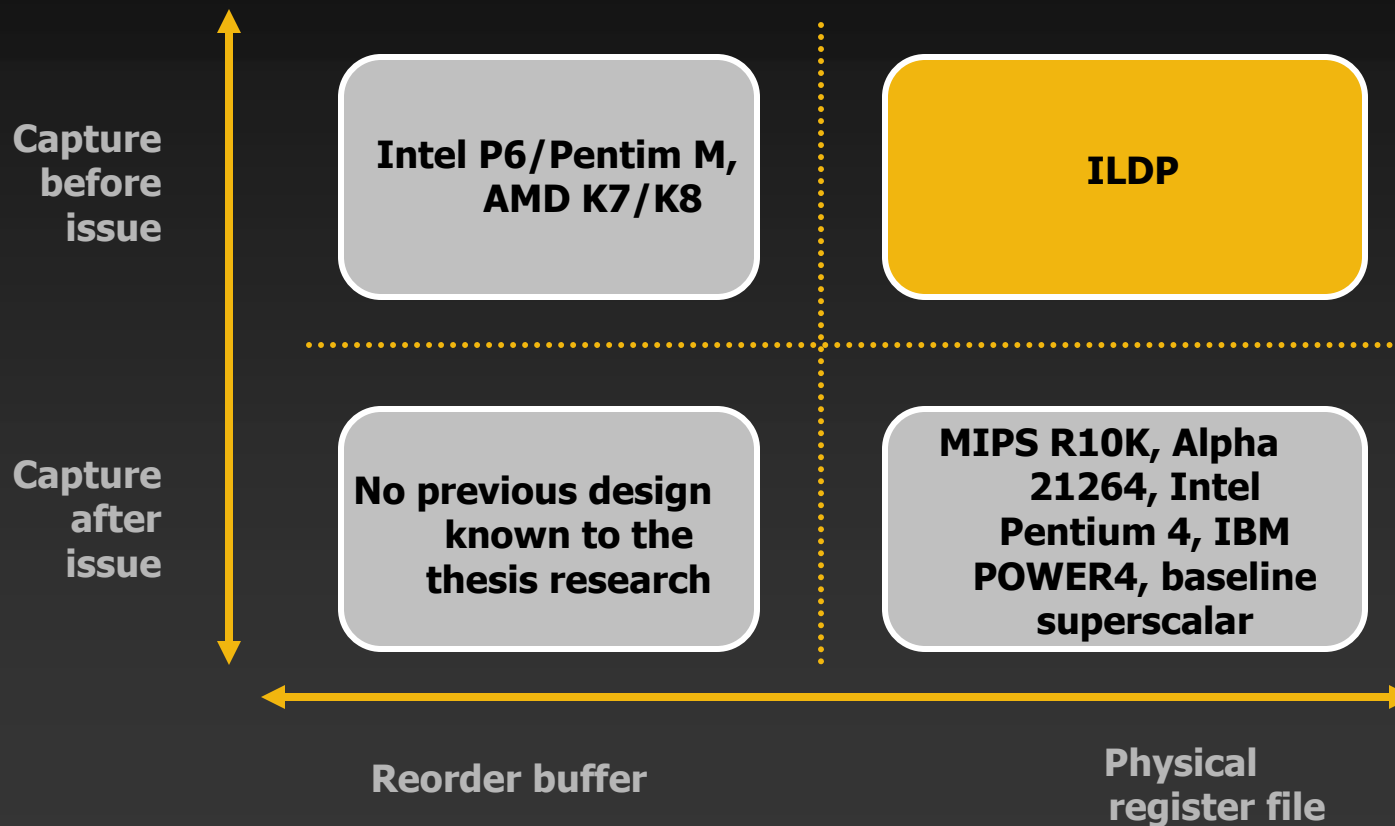
- It's all about exploring new design trade-off points



Spectrum of dynamic translation mechanisms

ILD P Design Points

- It's all about exploring new design trade-off points



Spectrum of register operand capture models

Summary

- A simple, distributed, yet dynamic superscalar microarchitecture
- Accumulator-oriented I-ISA is the key enabler
 - Reduces complexities in key hardware structures and binary translation
- Simple, low-overhead dynamic binary translation
 - Special hardware support for efficient control transfer is essential to reduce DBT overhead
- ILDP system performs similarly as conventional superscalar designs
 - Reduced complexity can be used for higher clock frequency or reduced power consumption
- Correct-by-design simulation infrastructure was built

Future Research Directions

- Reducing interpretation overhead
- Distributed D-cache and disambiguation mechanisms
- Static compilation
- Other virtual ISAs (x86)
- High ILP workloads
- Further I-ISA/DBT optimization opportunities