# Constructing Virtual Architectures on Tiled Processors

David Wentzlaff

Anant Agarwal

MIT

# Emulators and JITs for Multi-Core

David Wentzlaff

Anant Agarwal

MIT

# Why Multi-Core?

# Why Multi-Core?

Future architectures will be on-chip parallel machines

- Moore's Law provides more parallel silicon resources
- Diminishing sequential returns
- Growth applications are parallel

# Why Multi-Core?

Future architectures will be on-chip parallel machines

- Moore's Law provides more parallel silicon resources
- Diminishing sequential returns
- Growth applications are parallel

Future architectures will be optimized for parallel applications

- Hardware compatibility will be broken

# Why Emulators and JITs on Multi-Core?

- Future architectures will be on-chip parallel machines
  - Moore's Law provides more parallel silicon resources
  - Diminishing sequential returns
  - Growth applications are parallel
- Future architectures will be optimized for parallel applications
  - Hardware compatibility will be broken

# Why Emulators and JITs on Multi-Core?

- Future architectures will be on-chip parallel machines
  - Moore's Law provides more parallel silicon resources
  - Diminishing sequential returns
  - Growth applications are parallel
- Future architectures will be optimized for parallel applications
  - Hardware compatibility will be broken
- Future architectures will need to run legacy applications
  - Market forces will require future chips to run 1983 "Frogger" for DOS
  - Software re-verification on new architectures too costly

# Are Emulators and JITs for Multi-Core Different?

# Are Emulators and JITs for Multi-Core Different?

Yes

# Are Emulators and JITs for Multi-Core Different?

Yes

Bountiful parallel resources

- Example: Code optimization cost is reduced
  - “hot spot” analysis may miss sequential performance
- Parallelize client application

# Are Emulators and JITs for Multi-Core Different?

Yes

Bountiful parallel resources

- Example: Code optimization cost is reduced
  - "hot spot" analysis may miss sequential performance
- Parallelize client application

Parameters for Multi-Core are different

- Core-to-Core latencies reduced

# Road Map

Exploit on-chip parallel resources to accelerate emulation

# Road Map

Exploit on-chip parallel resources to accelerate emulation

# Road Map

Exploit on-chip parallel resources to accelerate emulation

Focused on performance

# Road Map

Exploit on-chip parallel resources to accelerate emulation

Focused on performance

Acceleration Mechanisms

# Road Map

Exploit on-chip parallel resources to accelerate emulation

Focused on performance

Acceleration Mechanisms

1. Pipelining Virtual Architectures

# Road Map

Exploit on-chip parallel resources to accelerate emulation

Focused on performance

Acceleration Mechanisms

1. Pipelining Virtual Architectures

2. Speculative Parallel Translation

# Road Map

Exploit on-chip parallel resources to accelerate emulation

Focused on performance

Acceleration Mechanisms

1. Pipelining Virtual Architectures
2. Speculative Parallel Translation
3. Static & Dynamic Architecture Reconfiguration

# Road Map

Exploit on-chip parallel resources to accelerate emulation

Focused on performance

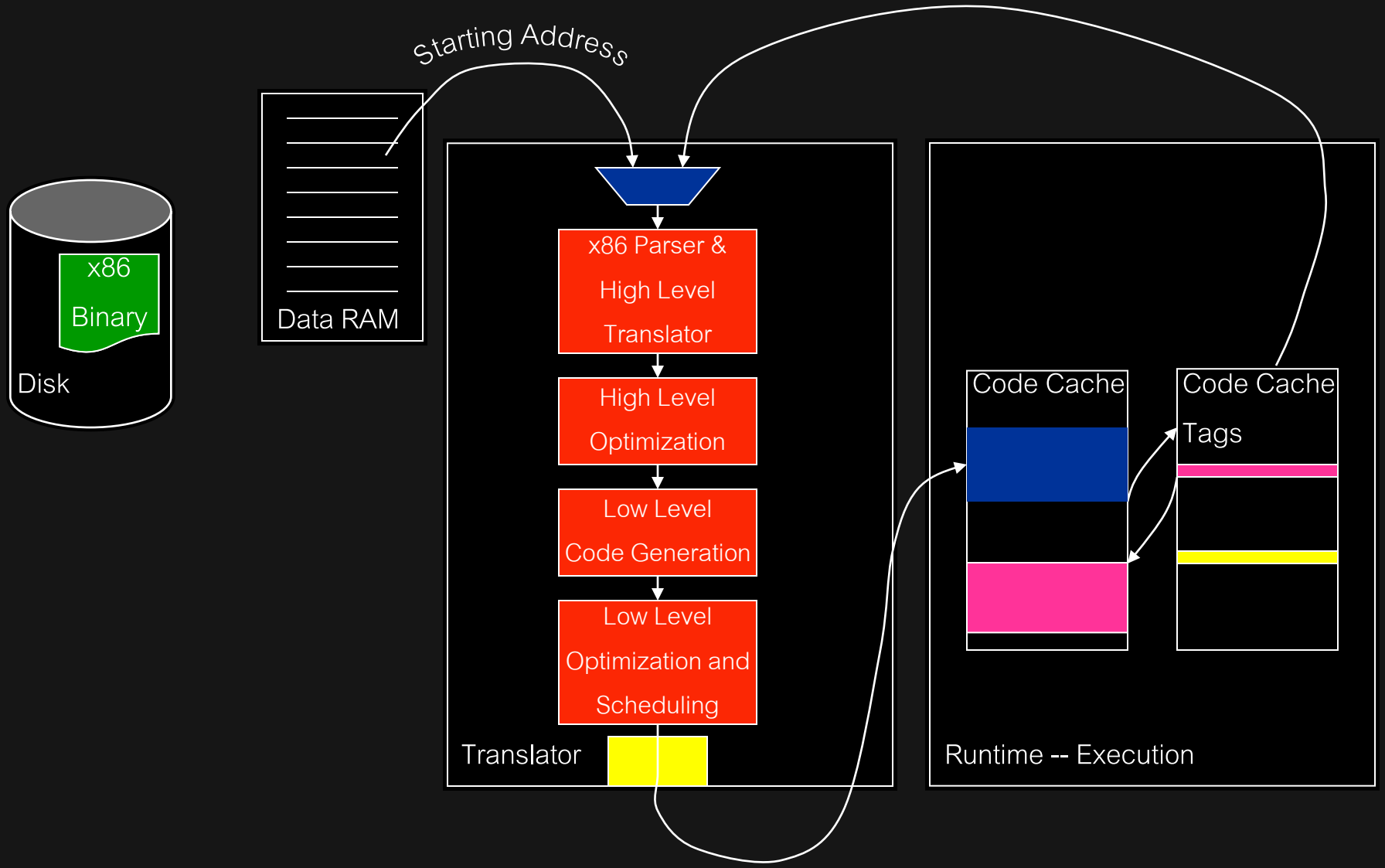
Acceleration Mechanisms

1. Pipelining Virtual Architectures
2. Speculative Parallel Translation
3. Static & Dynamic Architecture Reconfiguration

Proof of concept system

All software parallel translator: x86 on Raw

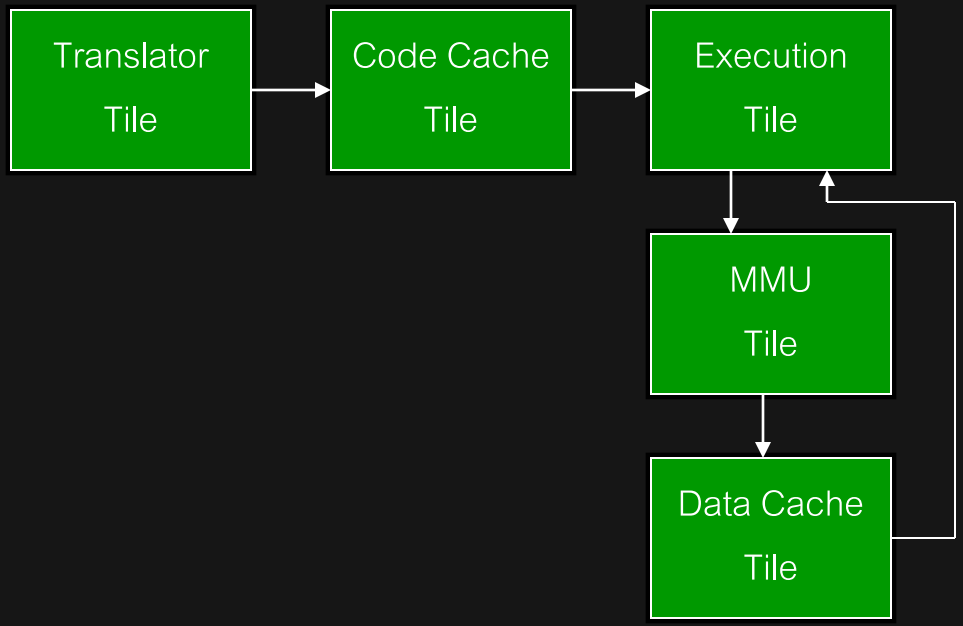
# Background: Translation



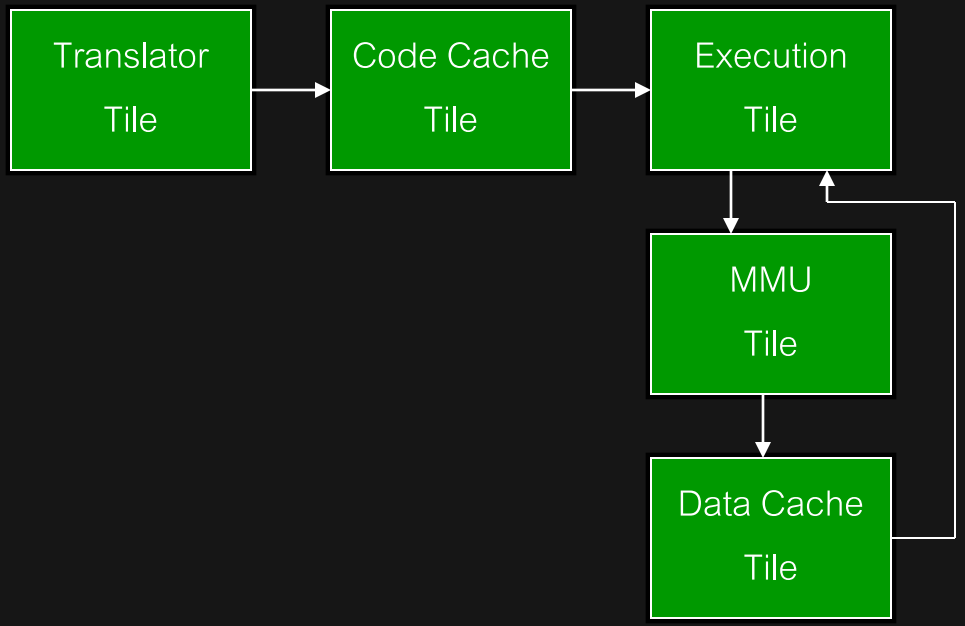
# Background: “Old” Parallel Translation

# 1. Pipelining Virtual Architectures

# 1. Pipelining Virtual Architectures

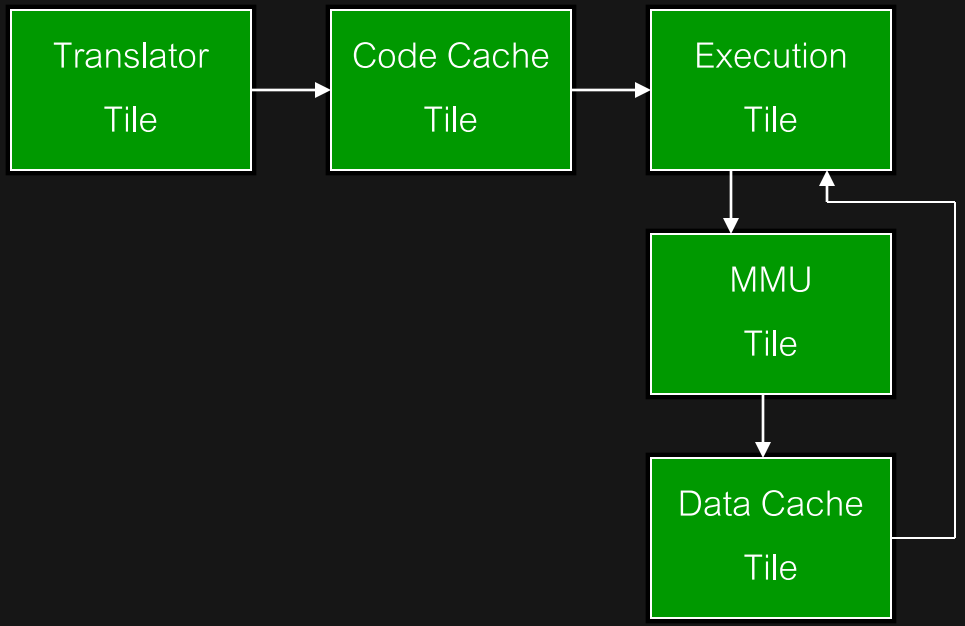


# 1. Pipelining Virtual Architectures




Utilize a Tiled Processor as fabric to construct virtual processor

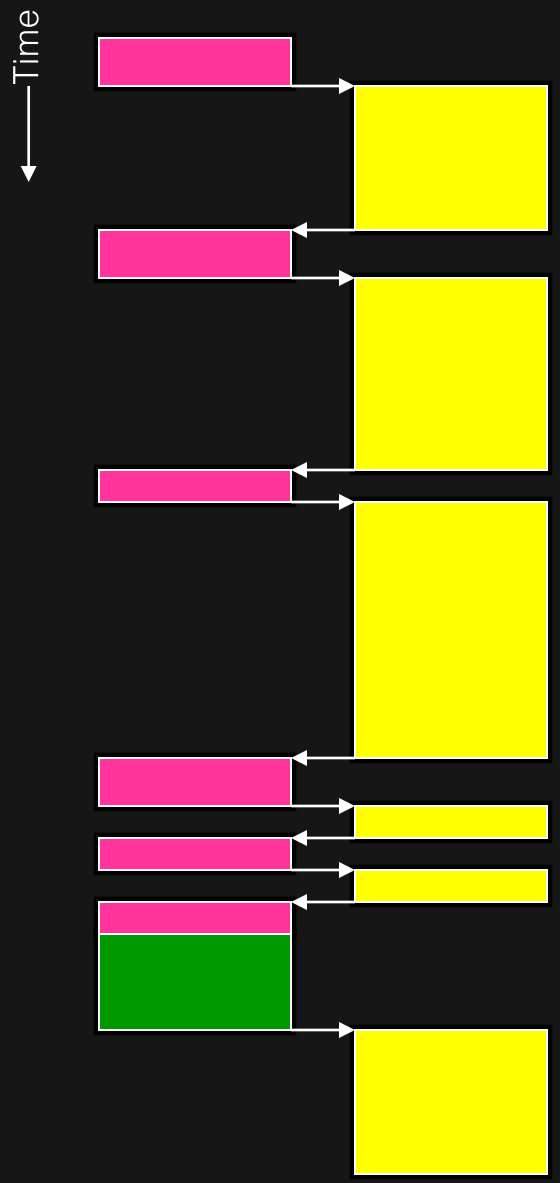
# 1. Pipelining Virtual Architectures

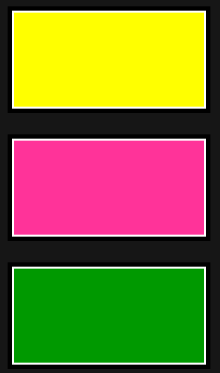



Utilize a Tiled Processor as fabric to construct virtual processor

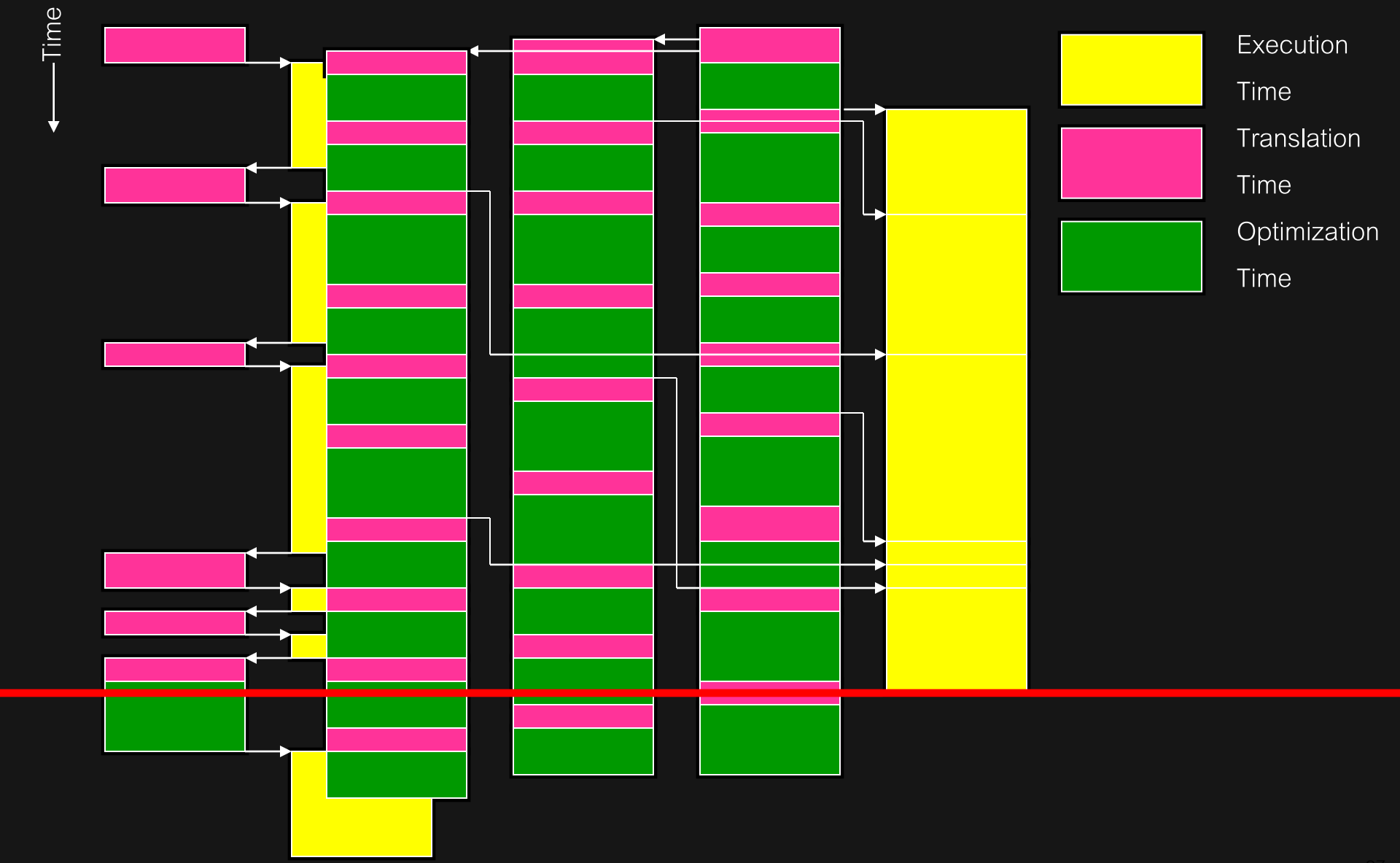
Coarse grain pipelining to exploit parallelism

# Sequential Translation



# 2. Speculative Parallel Translation



# 3. Reconfiguration

Different programs have different characteristics

Processor Architect uses benchmarks to choose “compromise” processor

# 3. Reconfiguration

Different programs have different characteristics

Processor Architect uses benchmarks to choose “compromise” processor

Static Reconfiguration

- Choose different virtual machine configuration based off application

Dynamic Reconfiguration

- Detect phases/programs dynamic needs and reconfigure at runtime
  - Cost to reconfiguration

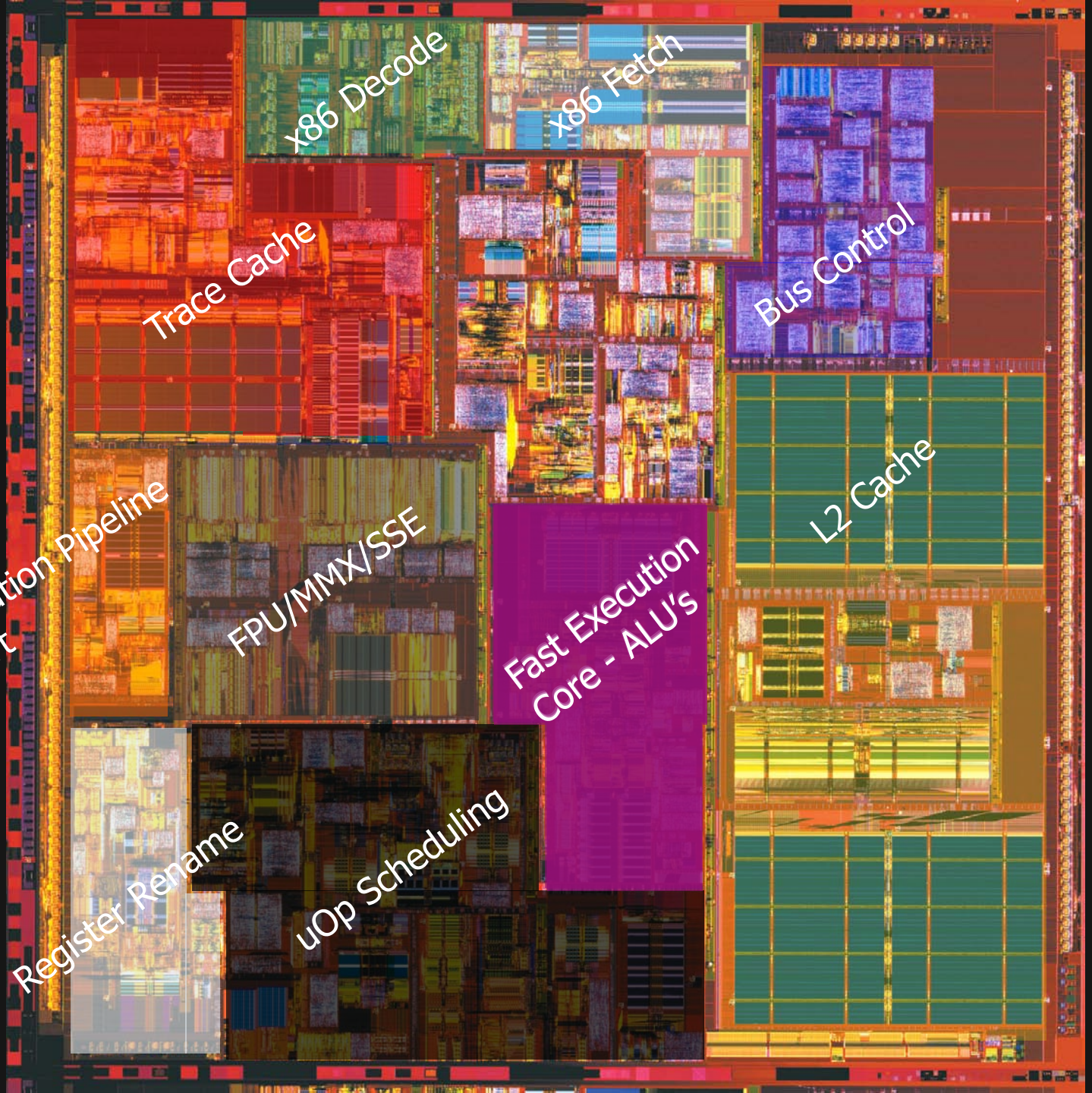
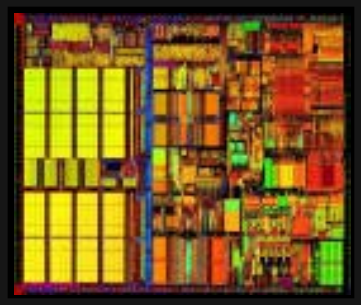


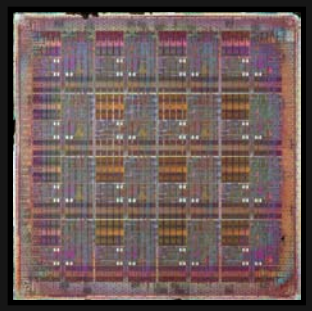

Photo Courtesy Intel Corp.

# Prototype System and Evaluation

# Background: Architectures



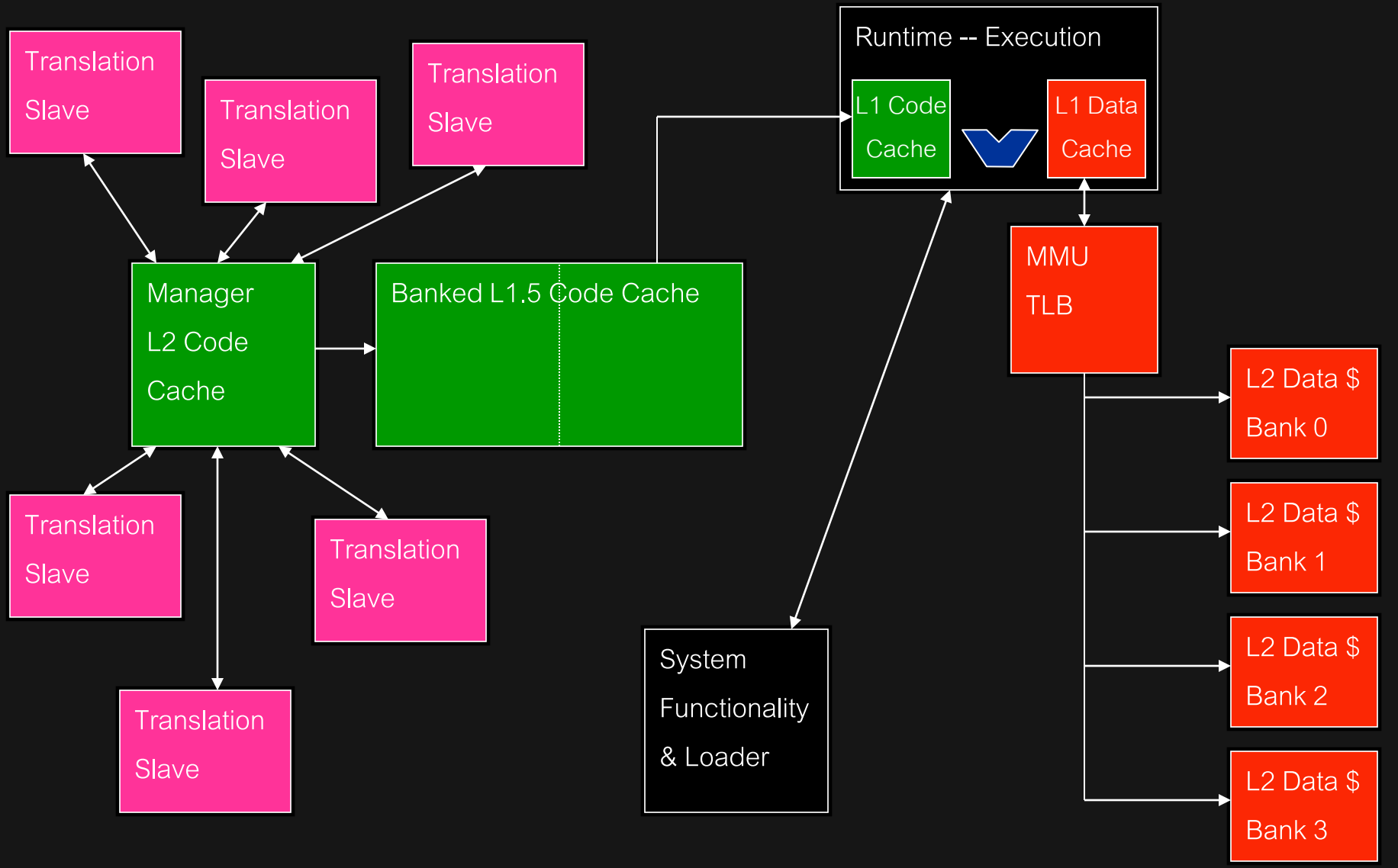
*



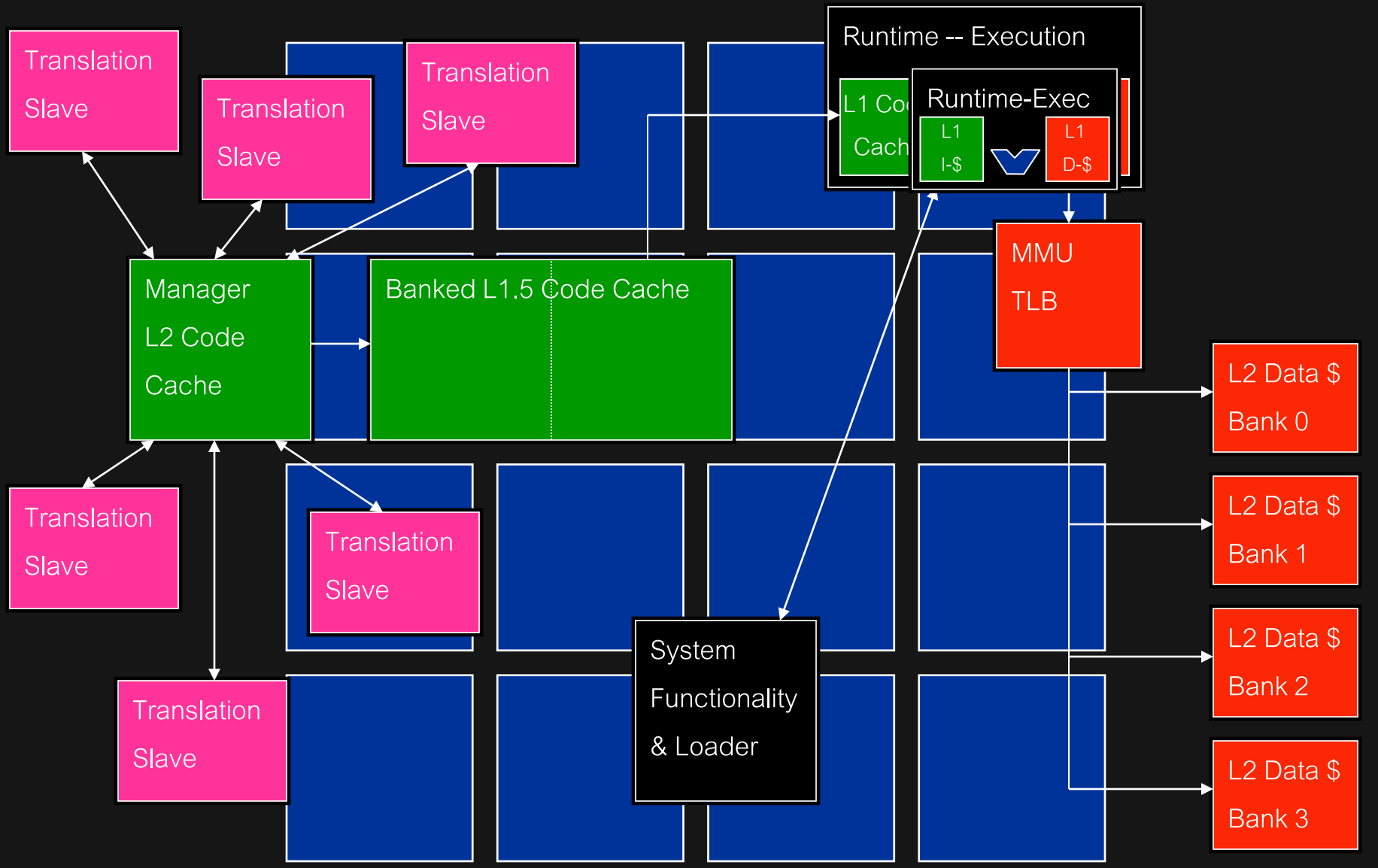
| | x86 (Pentium III) | Raw |
|---|---|---|
| ISA | CISC instruction set | RISC instruction set |
| | Hardware Virtual Memory (VM) | No VM |
| | Hardware Memory Protection | No Memory Protection |
| | Condition Codes used for branching | No Condition Codes |
| | Hardware instruction cache | Software managed instruction memory |
| Implementation | 1 superscalar processor core | 16 Processors arranged in 4x4 mesh |
| | 3-way parallelism | 4 low latency networks |
| | Out-of-order processor | In-order processors |

* Photo Courtesy Intel Corp.

# System Design



# System Design



# Methodology

Cycle comparison of Raw vs. Pentium III

All results collected on real hardware

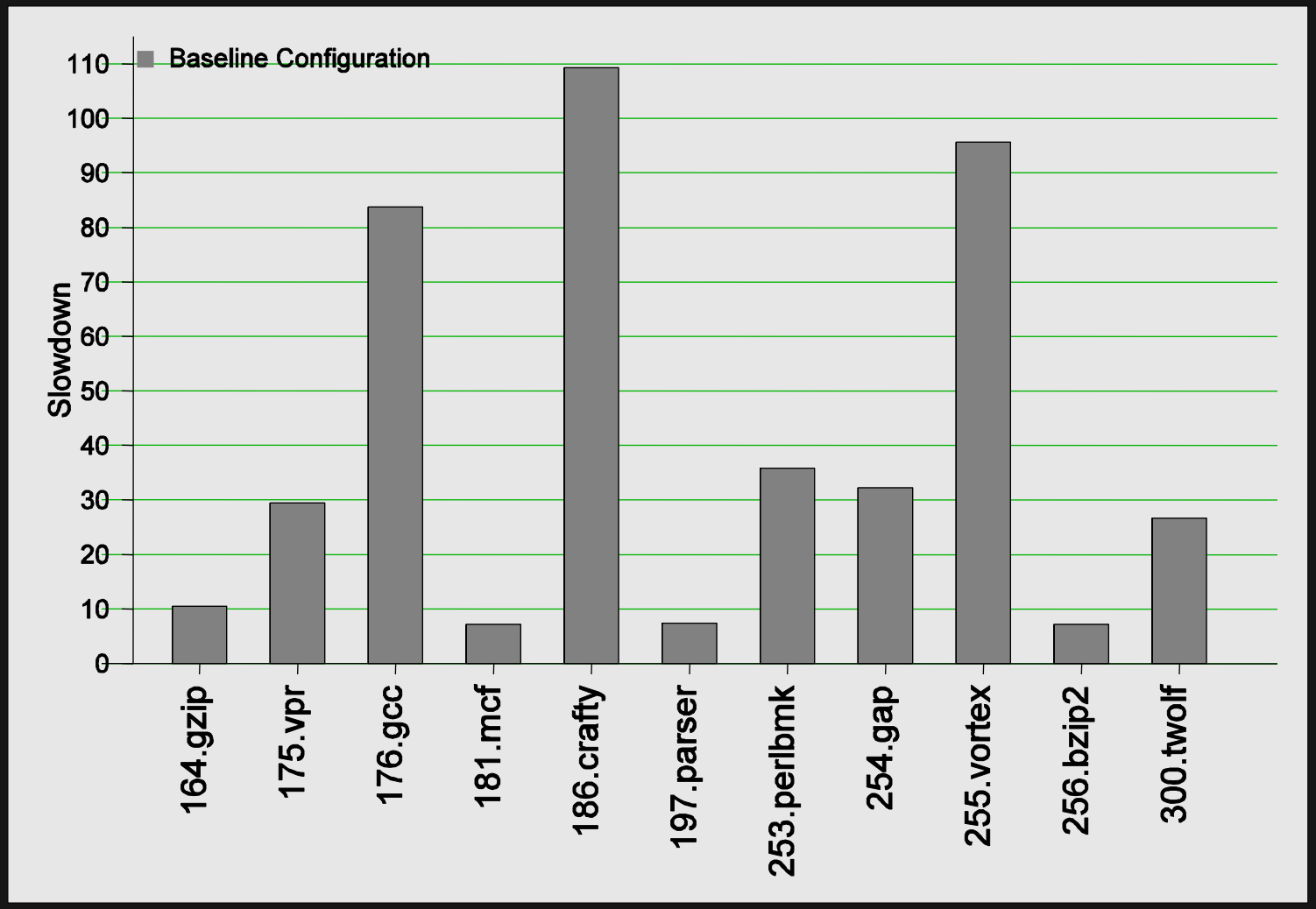
- No hardware added

Same binaries (unmodified)

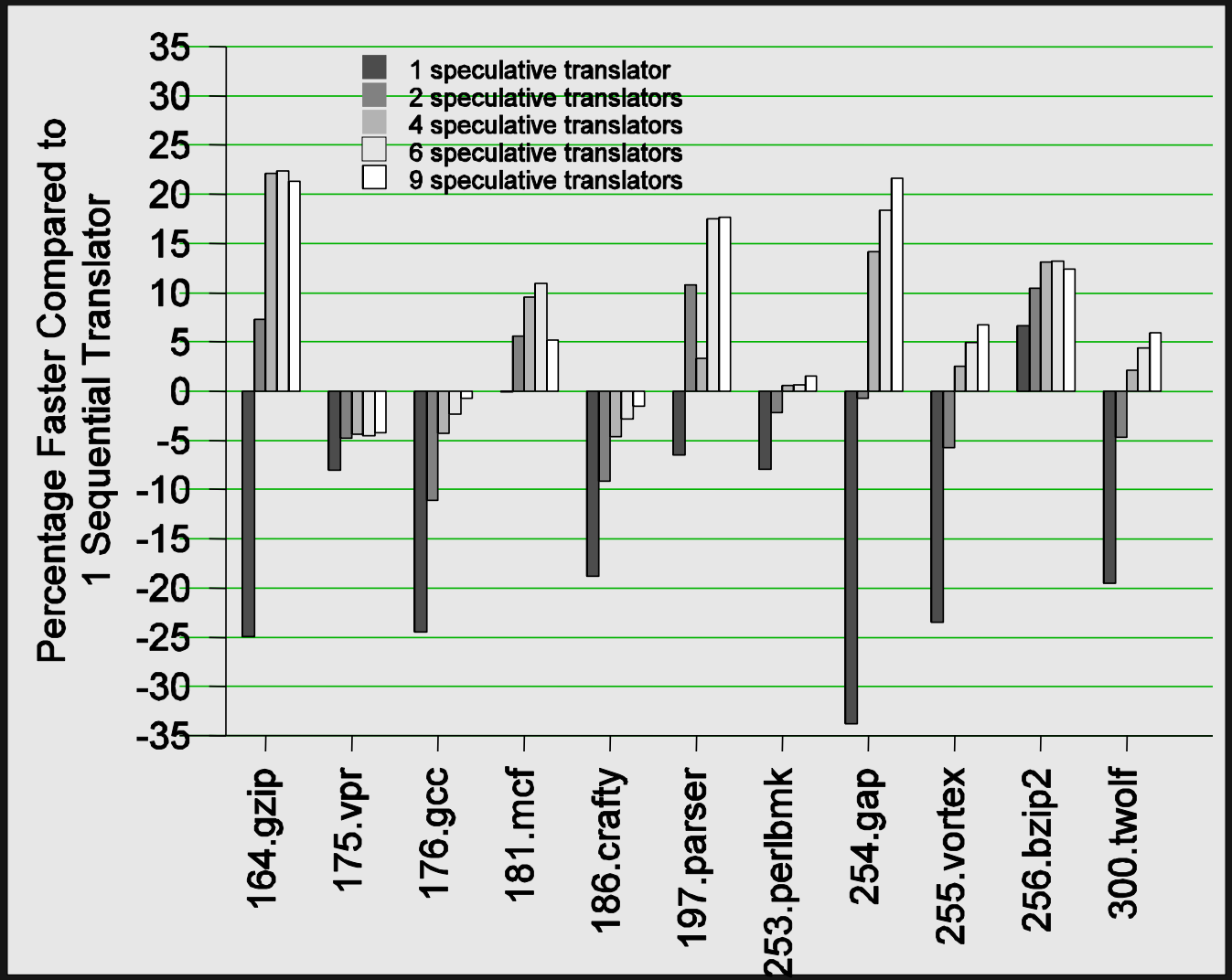
Metric: Slowdown

- Raw executing x86 code compared by cycle against Pentium III

# Baseline Performance

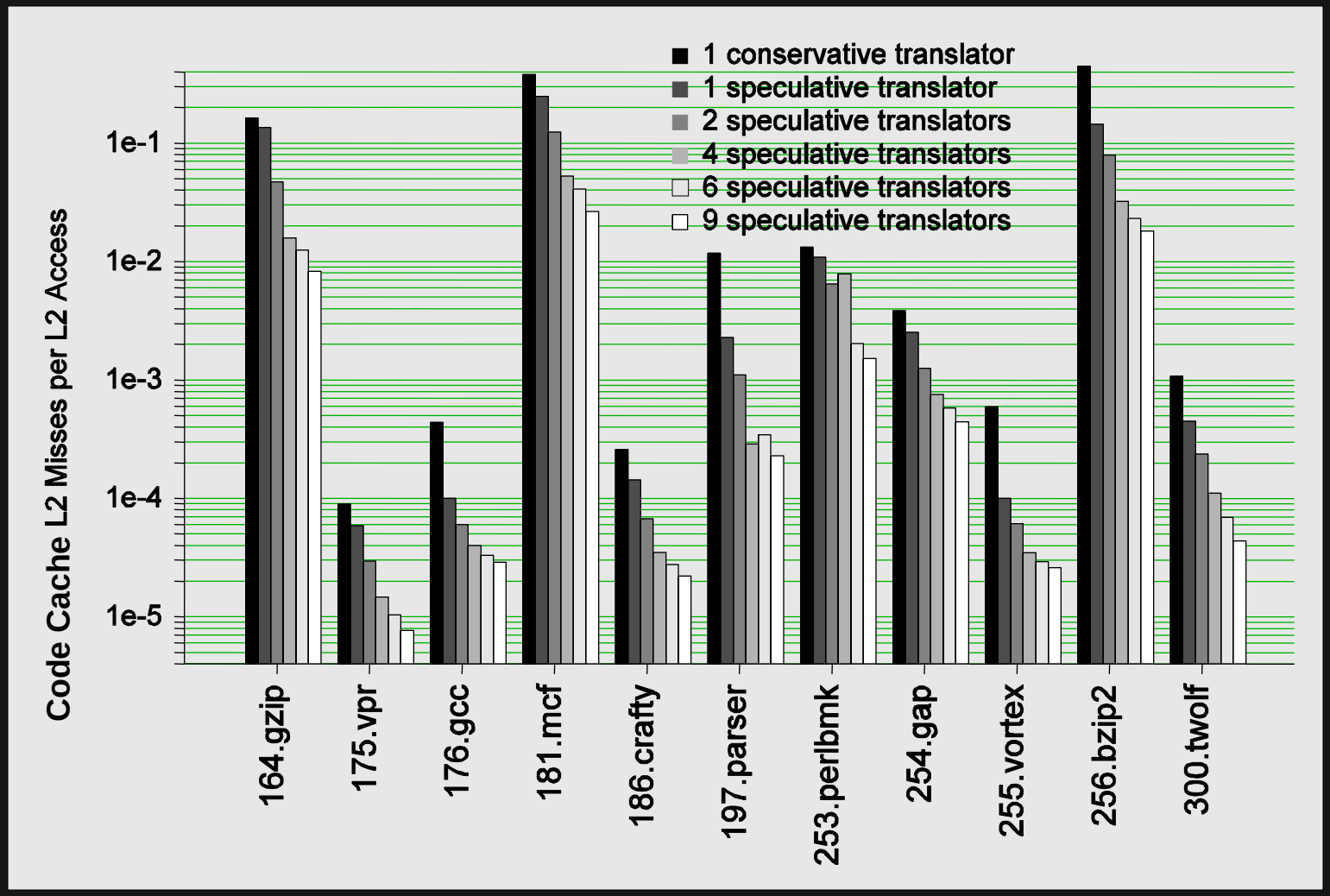


# 2. Speculative Parallel Translation

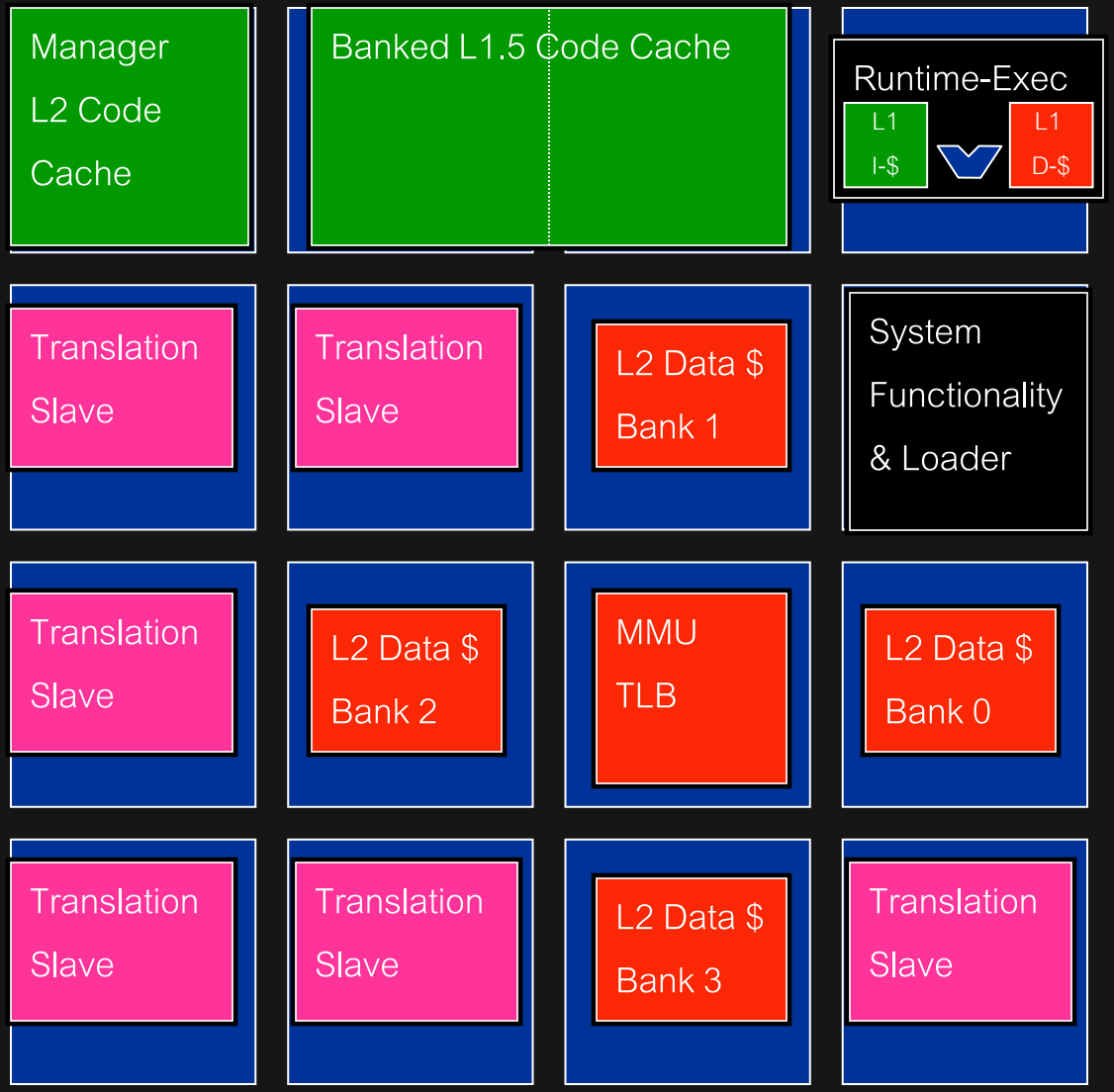


# L2 Code Cache Miss Rate

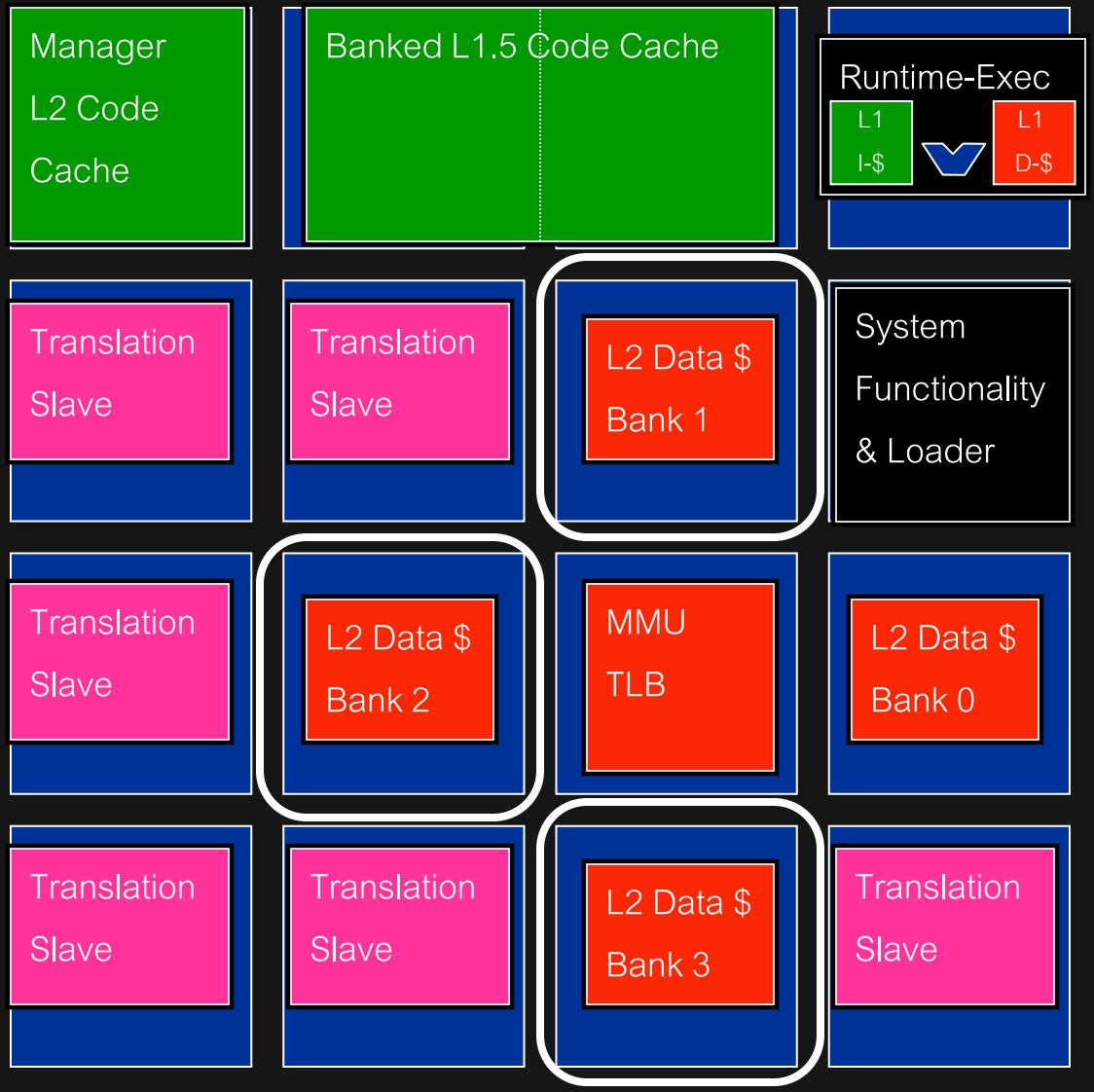
## 2. Speculative Parallel Translation



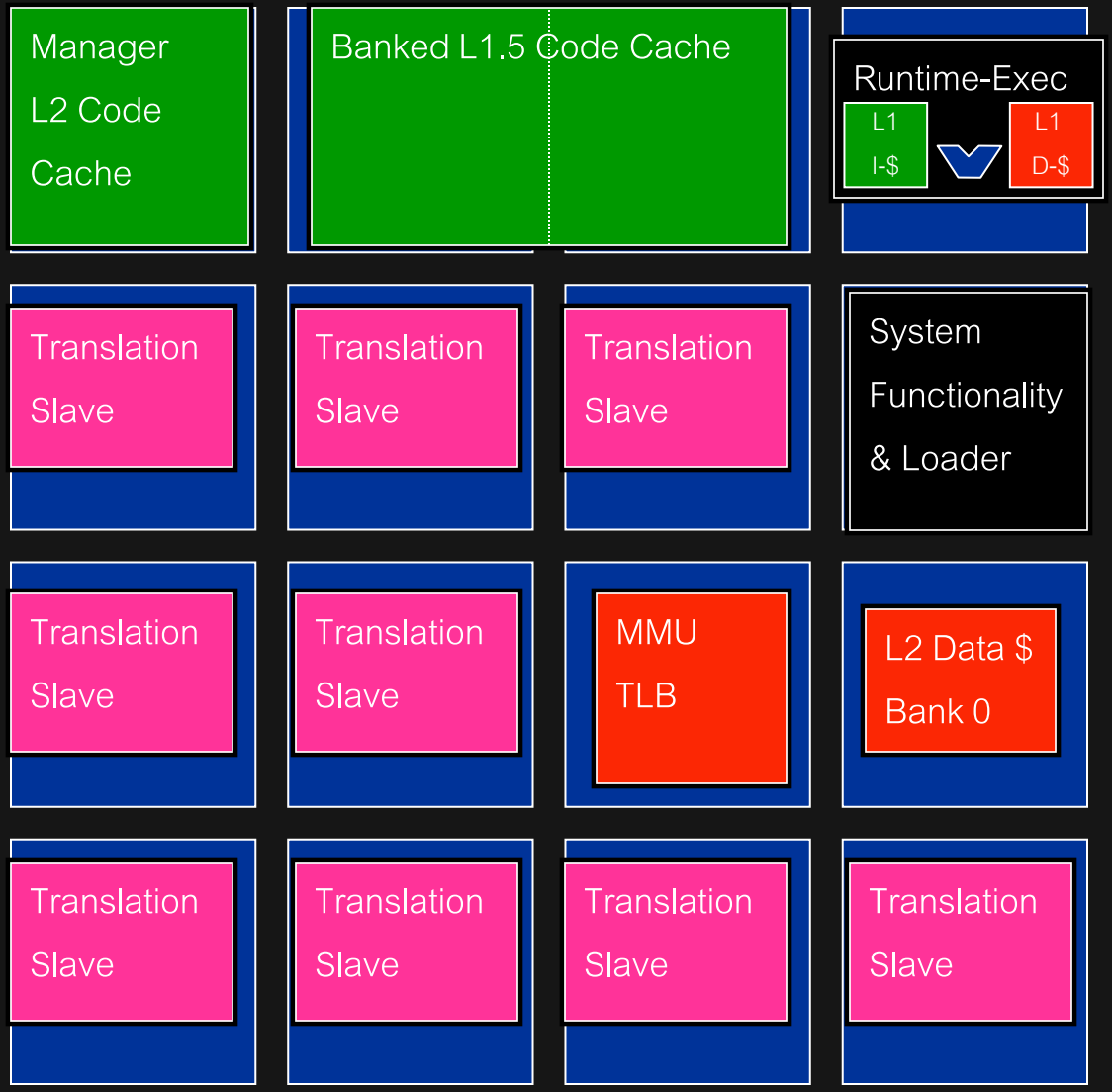
# 3. Static Reconfiguration



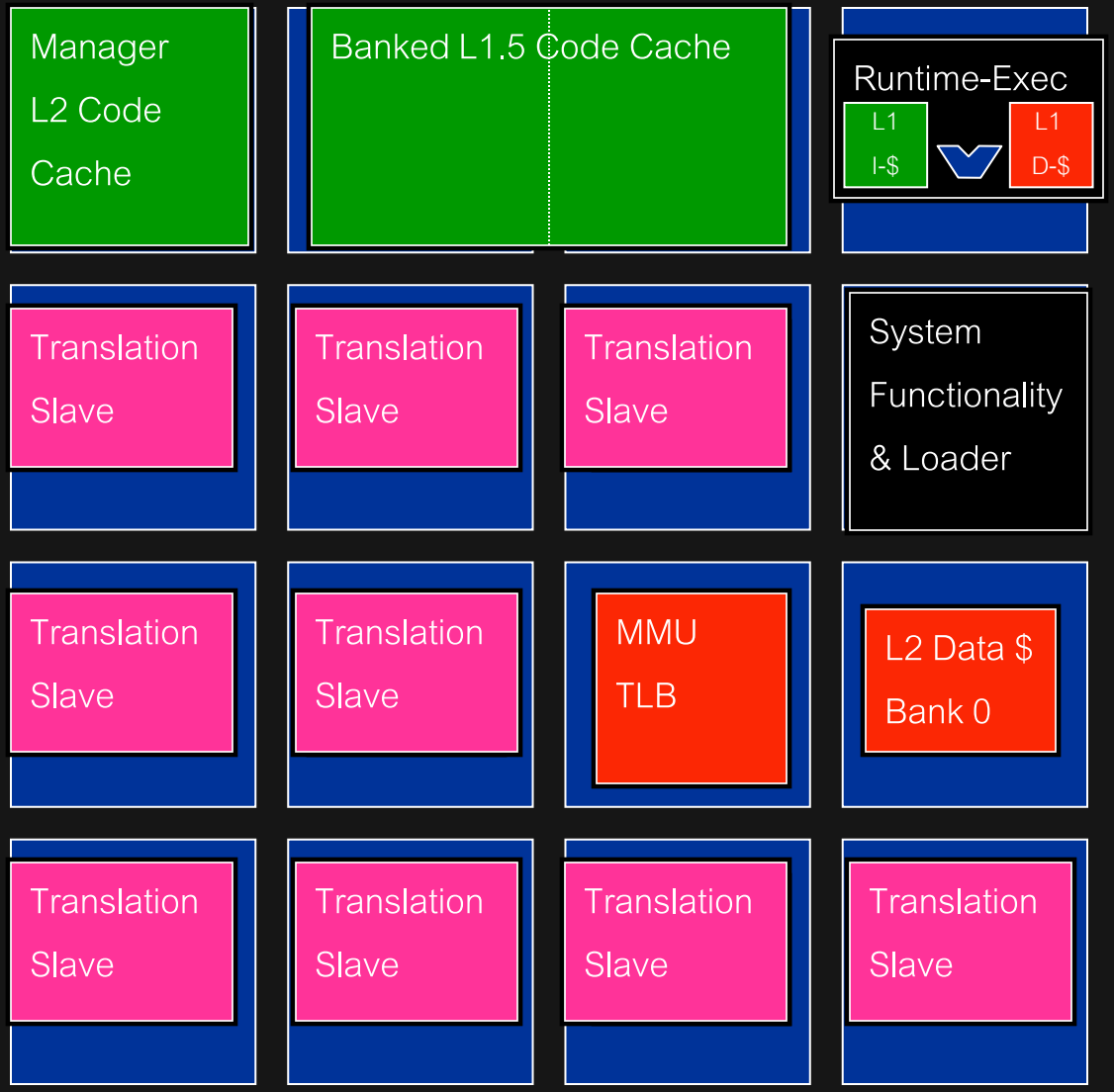
# 3. Static Reconfiguration



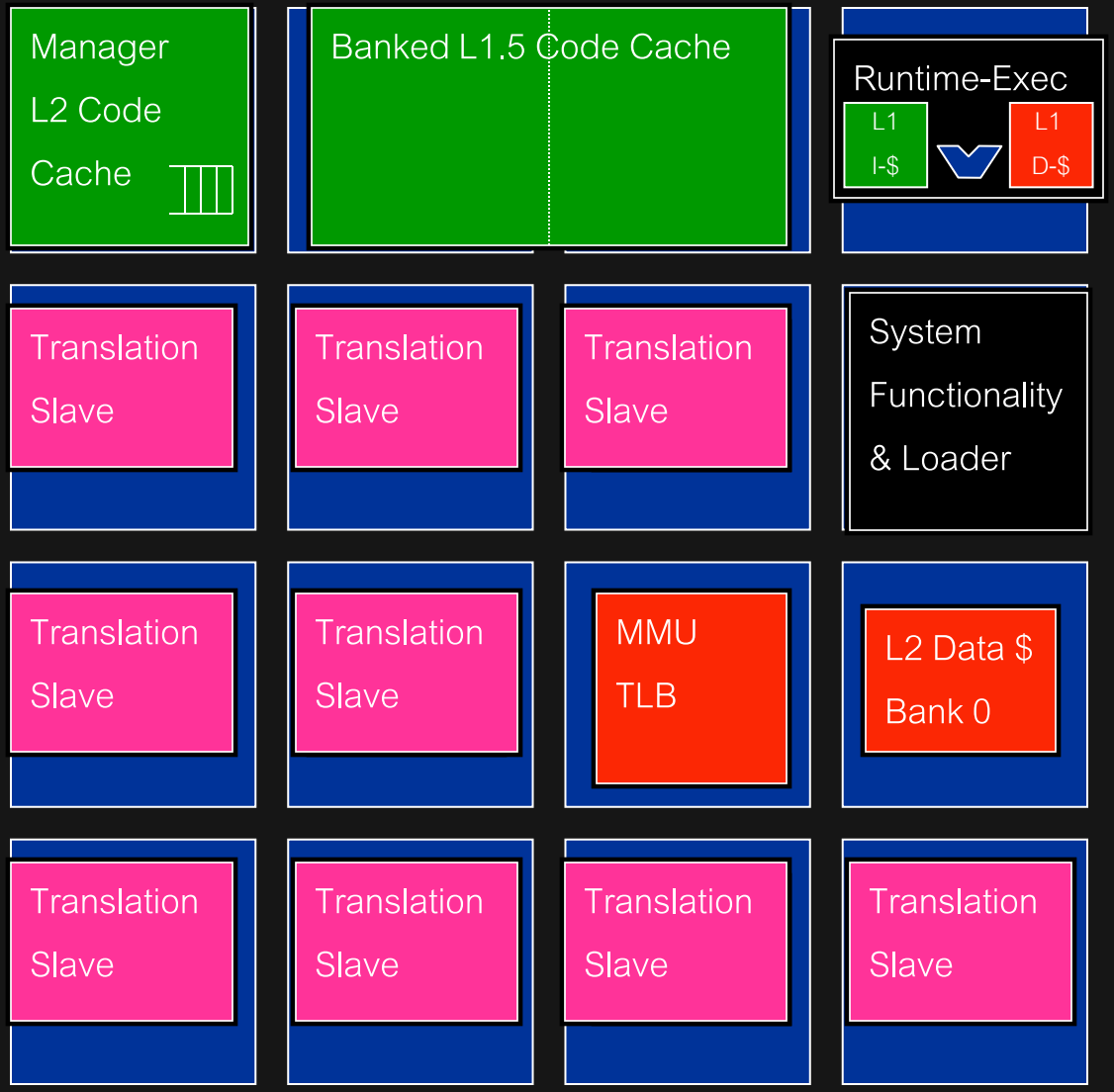
# 3. Static Reconfiguration



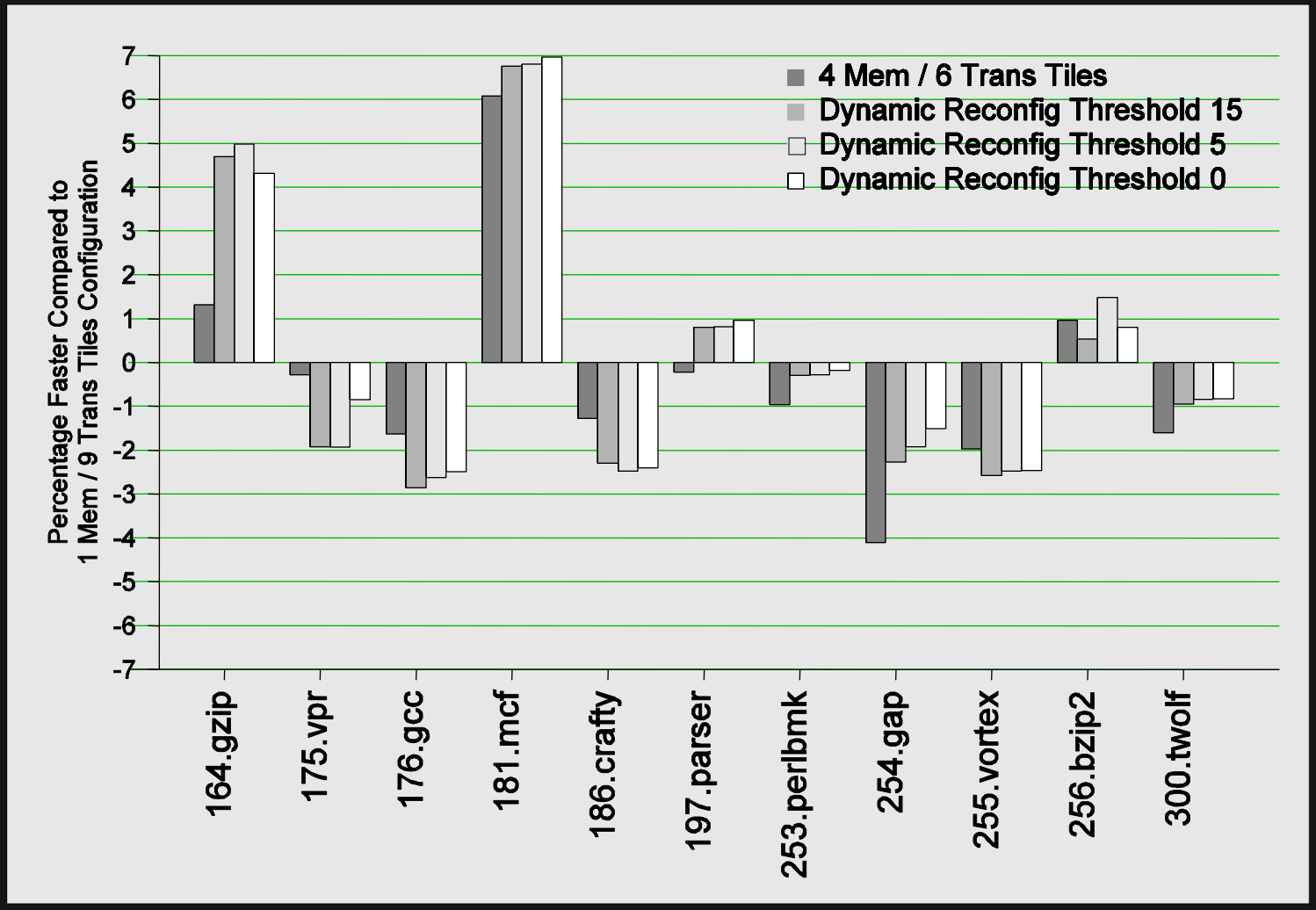
# 3. Dynamic Reconfiguration



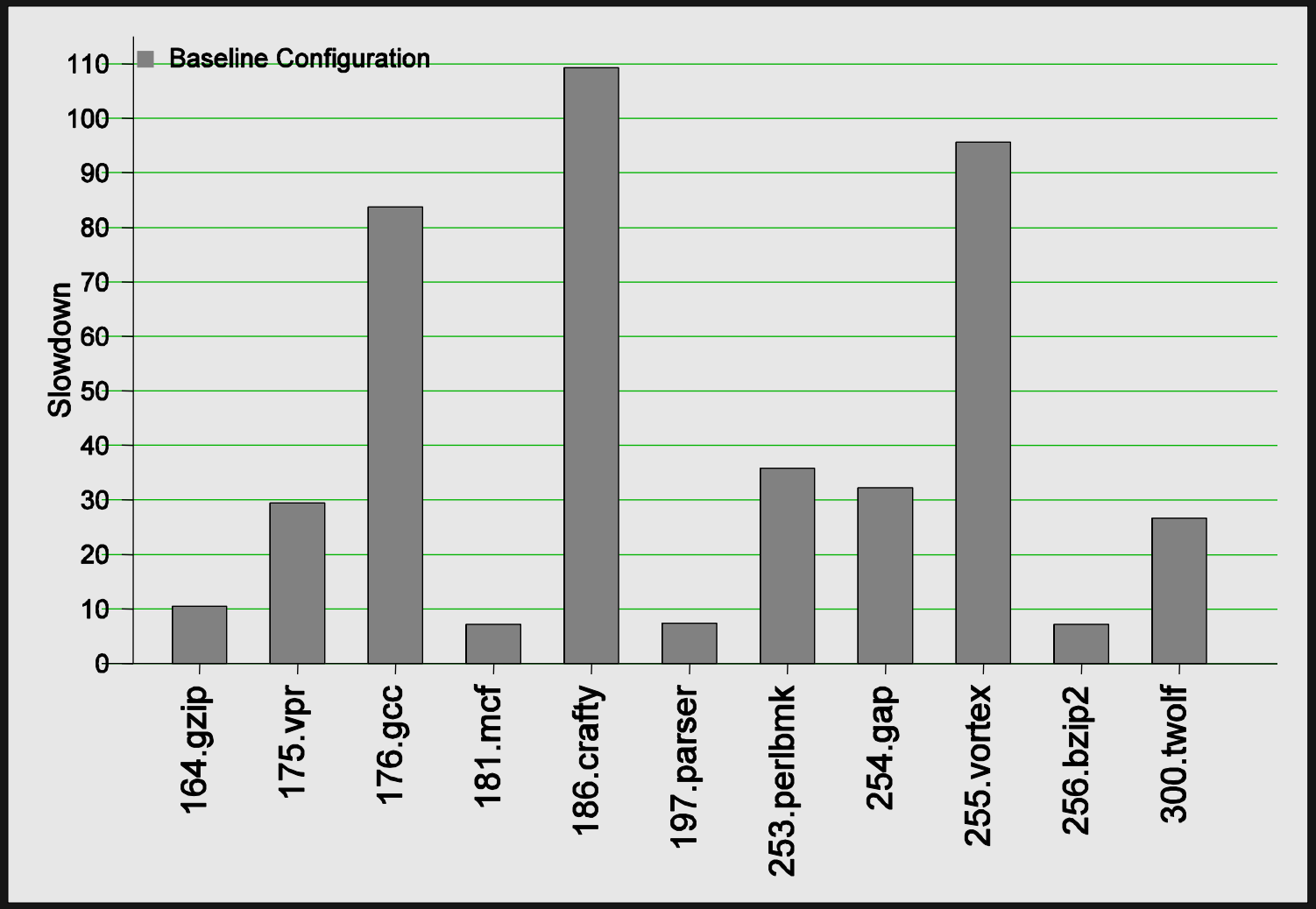
# 3. Dynamic Reconfiguration



# 3. Reconfiguration Zoom



# Baseline Performance Analysis



# Baseline Performance Analysis

# Baseline Performance Analysis

| Intrinsic | Raw Emulator | | Pentium III | |
|---|---|---|---|---|
| | latency | occupancy | latency | occupancy |
| L1 Cache Hit | 6 | 4 | 3 | 1 |
| L2 Cache Hit | 87 | 87 | 7 | 1 |
| L2 Cache Miss | 151 | 87 | 79 | 1 |

# Baseline Performance Analysis

| Intrinsic | Raw Emulator | | Pentium III | |
|---|---|---|---|---|
| | latency | occupancy | latency | occupancy |
| L1 Cache Hit | 6 | 4 | 3 | 1 |
| L2 Cache Hit | 87 | 87 | 7 | 1 |
| L2 Cache Miss | 151 | 87 | 79 | 1 |

CPI = (memory_access_rate * (((1 – L1_miss_rate) * L1_hit_occupancy) + (L1_miss_rate * (((1 – L2_miss_rate * L2_miss_occupancy))))) + ((1 – memory_access_rate) * non_memory_CPI)

# Baseline Performance Analysis

| Intrinsic | Raw Emulator | | Pentium III | |
|---|---|---|---|---|
| | latency | occupancy | latency | occupancy |
| L1 Cache Hit | 6 | 4 | 3 | 1 |
| L2 Cache Hit | 87 | 87 | 7 | 1 |
| L2 Cache Miss | 151 | 87 | 79 | 1 |
| Memory CPI | | 3.9 | | 1 |

CPI = (memory_access_rate * (((1 – L1_miss_rate) * L1_hit_occupancy) + (L1_miss_rate * (((1 – L2_miss_rate * L2_miss_occupancy))))) + ((1 – memory_access_rate) * non_memory_CPI)

# Baseline Performance Analysis

| | |
|---|---|
| Memory System | 3.9x slowdown |
| Lack of ILP | 1.3x slowdown |
| Condition Codes (Flags) | 1.1x slowdown |

# Baseline Performance Analysis

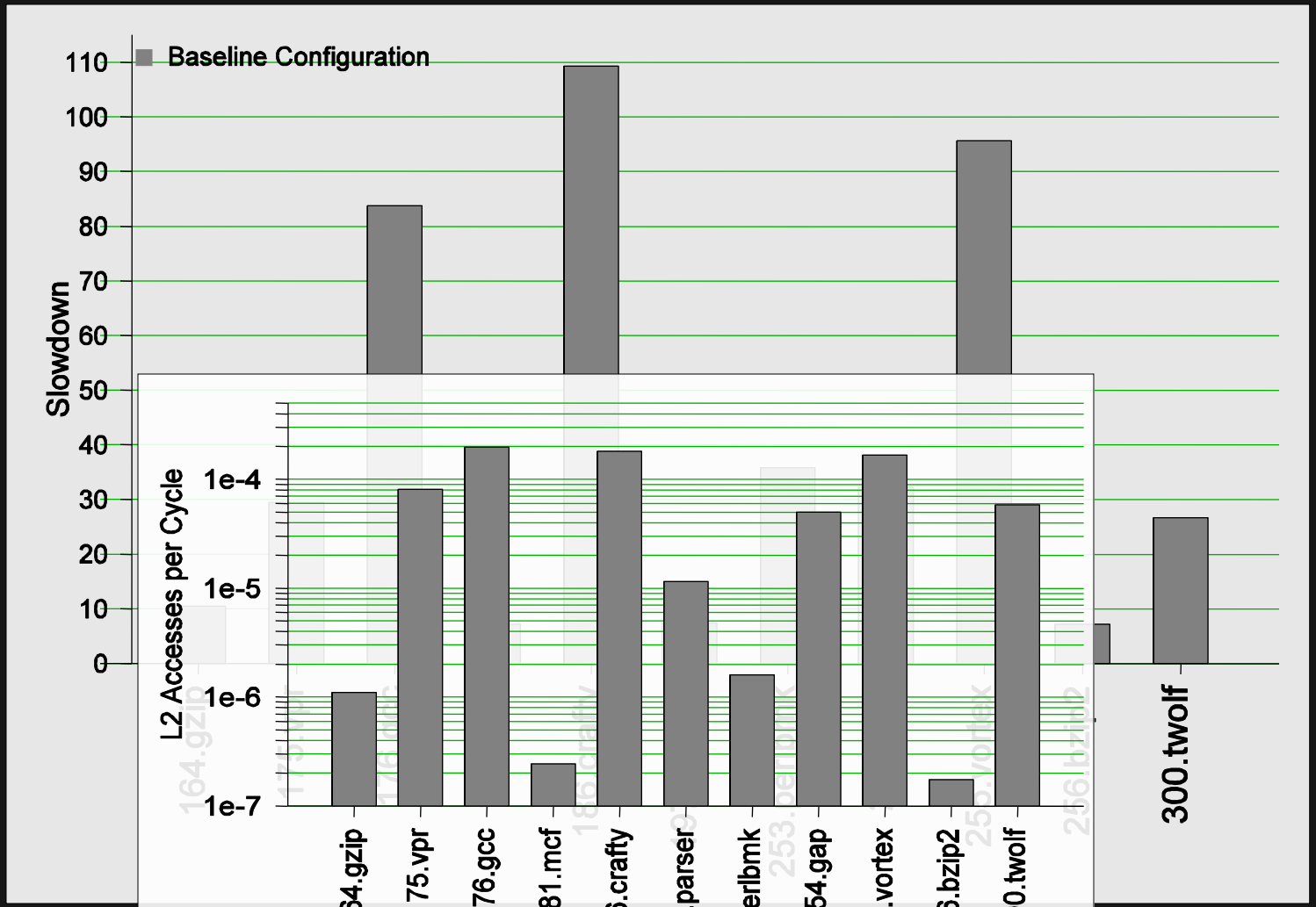
| | | |
|---|---|---|
| Memory System | | 3.9x slowdown |
| Lack of ILP | | 1.3x slowdown |
| Condition Codes (Flags) | x | 1.1x slowdown |
| | | 5.5x slowdown |

# Baseline Performance Analysis

| | | |
|---|---|---|
| Memory System | | 3.9x slowdown |
| Lack of ILP | | 1.3x slowdown |
| Condition Codes (Flags) | x | 1.1x slowdown |
| | | 5.5x slowdown |

Code Cache Misses 1 – 20x slowdown

# Baseline Performance Analysis



# Future Work

Hardware additions to facilitate parallel emulation

x86 Server farm on a chip

- Inter-Virtual Processor dynamic load balancing

Differing forms of Dynamic Reconfiguration

- Vary number of functional units

# Questions ?

# Extras