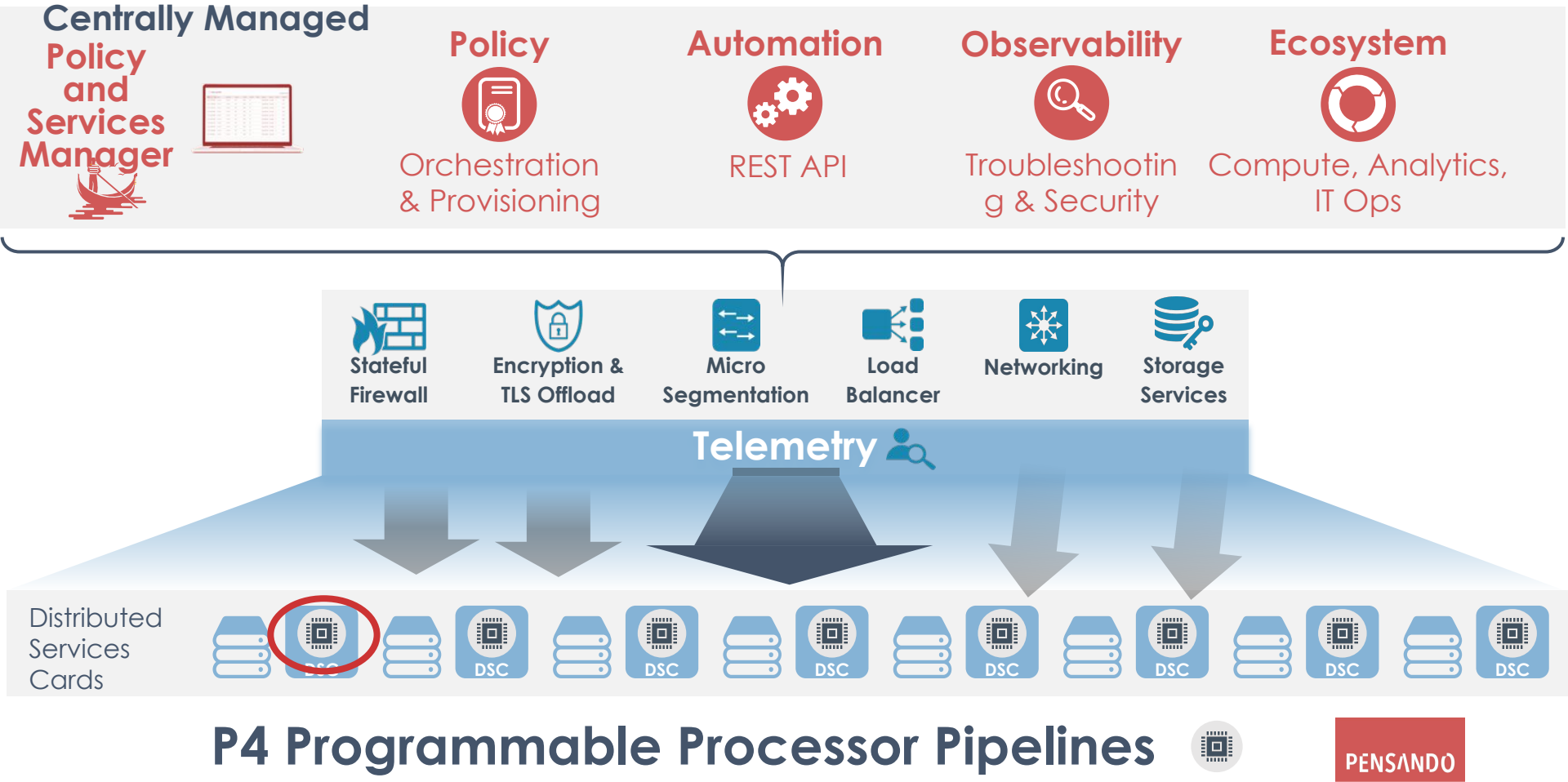
Flexible and Efficient Memory System for a High Performance Processing Pipeline

Mario Baldi
Distinguished Technologist

Mike Galles
ASIC Architect

Pensando Systems

Pensando Distributed Services Platform



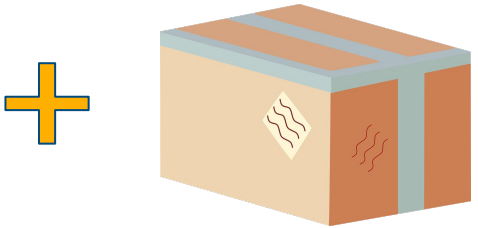
Why Pipeline? [Why P4]

PENSANDO

What is specific of a pipeline?

Each stage executes a limited task

- Small number of instructions
  - E.g., an action in P4
- Simple data structures
  - E.g., a table in P4



- Similar processing
  - Limited set of operations
- “Well-defined” data access

PENSANDO

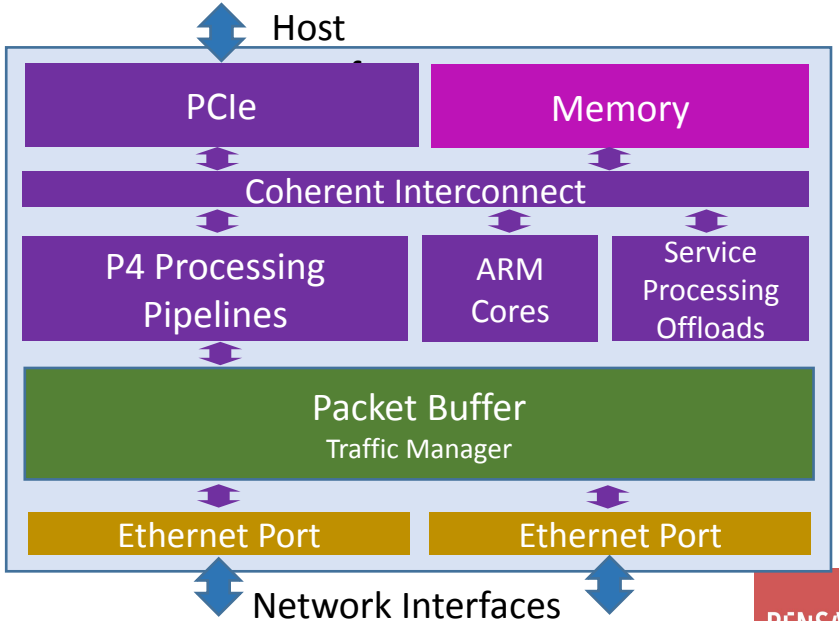
How do we get high performance?

- Optimize execution of the limited set of operations
- Specialized hardware for table lookup
- Fast local instruction memory for small programs
- Fast local data memory for small tables

PENSANDO

Memory is key to achieve expected performance

- Pipeline
- P4 programs
- [Overall solution]



PENSANDO

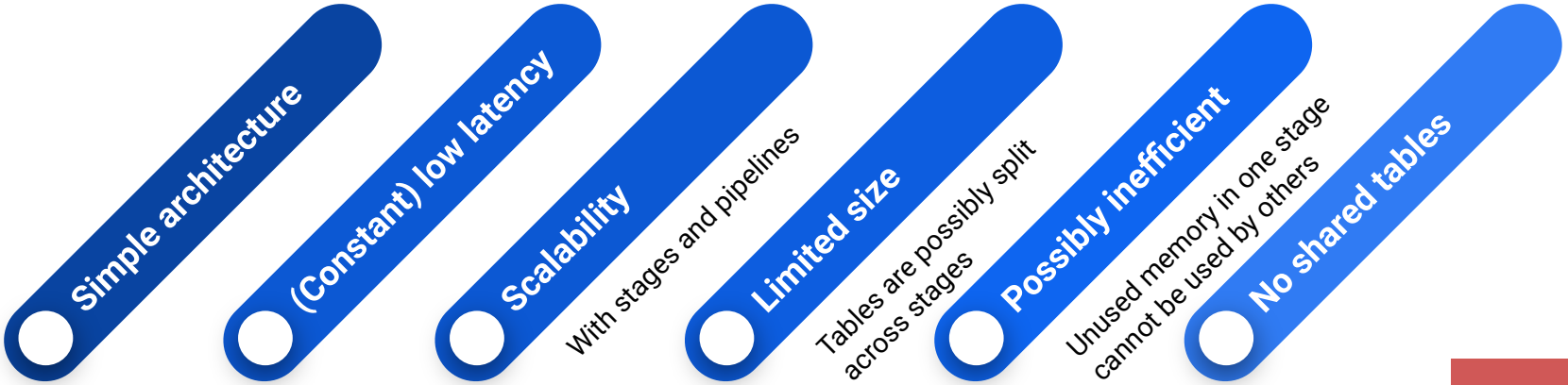
Memory Design Options

PENSANDO

Local Memory

Each stage has a dedicated amount of memory

- Instruction and data (table) memory
- Possibly writeback memory (e.g., registers)



PENSANDO

Central Memory

Stages share a common memory

- Instruction and data (table) memory
- Possibly writeback memory (e.g., registers)
- Possibly off-chip

👍 Size can be large

👍 Efficient and flexible usage

👍 Shared by stages and devices

👎 High latency

👎 Potential bottleneck

PENSANDO

Central Memory with Local Cache

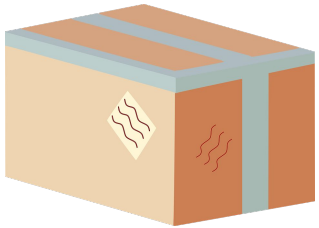
Mask access latency and limit contention

↓

possibly best of breed

In traditional, general purpose computing

- Spatial locality
- Temporal locality



In pipelined packet processing, also

Functional locality

PENSANDO

Functional Locality

A given function accesses a limited number of instructions and amount of data

Instructions and data for the particular function are in the cache of the stage where execution takes place

- Short program at each stage
- Small tables are common

With larger tables

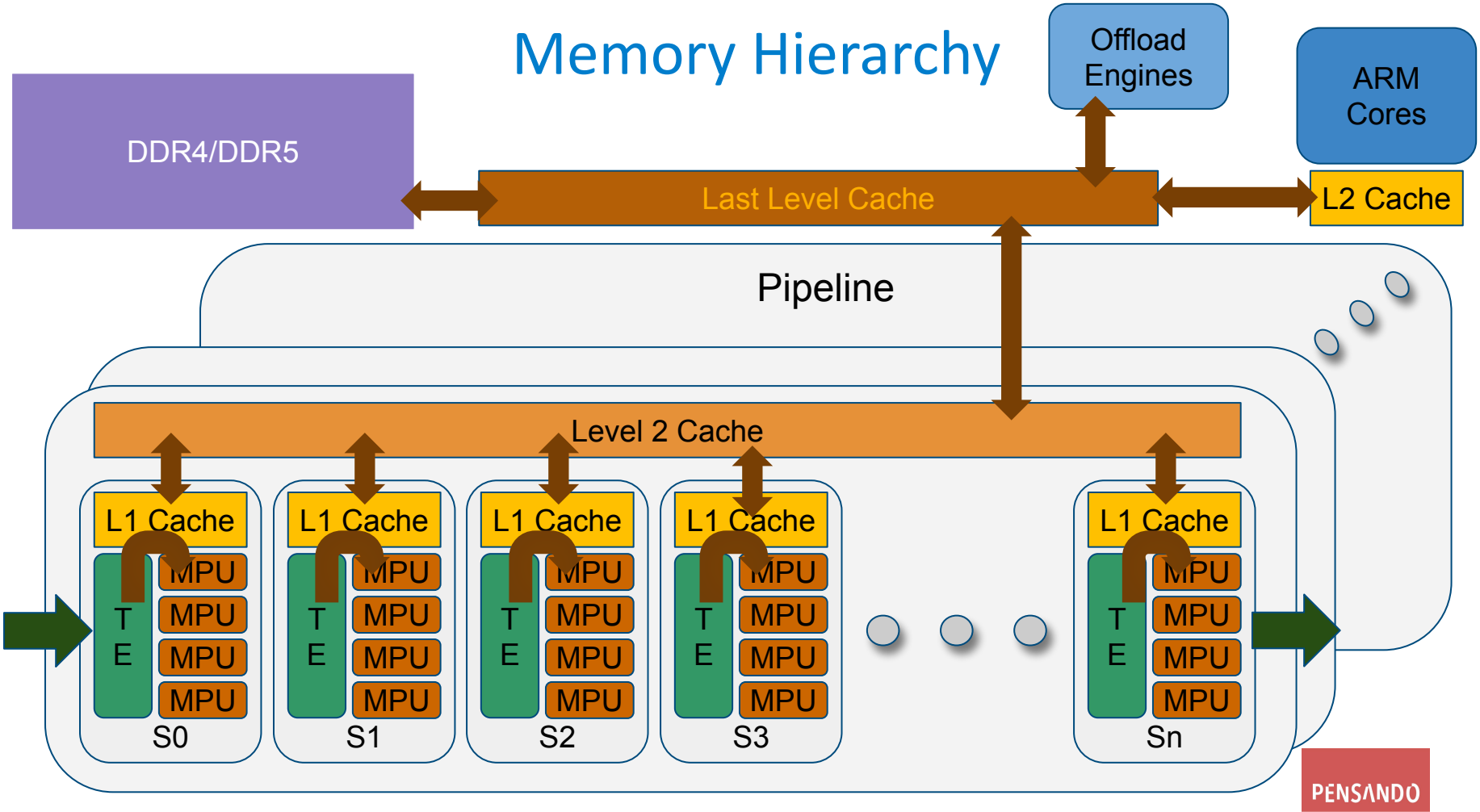
👍 Packets of the same flow access the same entries and are processed close in time

- Temporal locality

👎 Large number of active flows may result in misses

PENSANDO

Memory Hierarchy



Variable Access Latency

- Low on cache hits
- High on cache misses

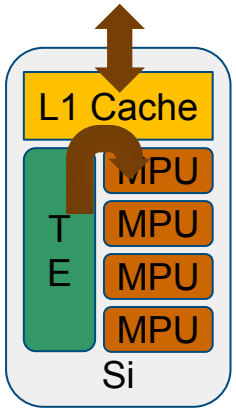
Options for handling high latency

- Stall the pipeline
  - Affect throughput → performance
- Process another packet

PENSANDO

Handling Variable Latency

- Table Engine (TE) can issue up to 28 reads in parallel
  - Possibly related to multiple packets
- As each completes, action data is provided to one of the Match Processing Units (MPUs)
  - MPU fetches instructions
    - Likely to still be in cache
- Packet ordering preserved



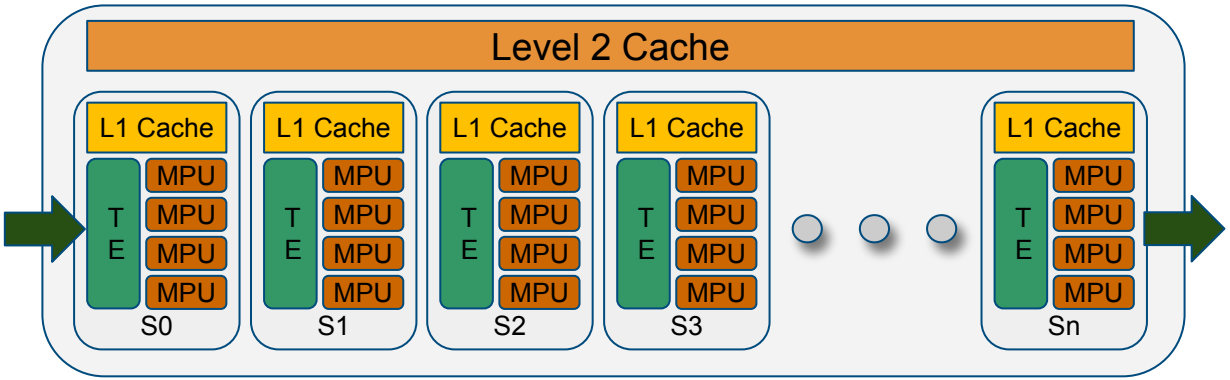
PENSANDO

Experimental Evaluation

PENSANDO

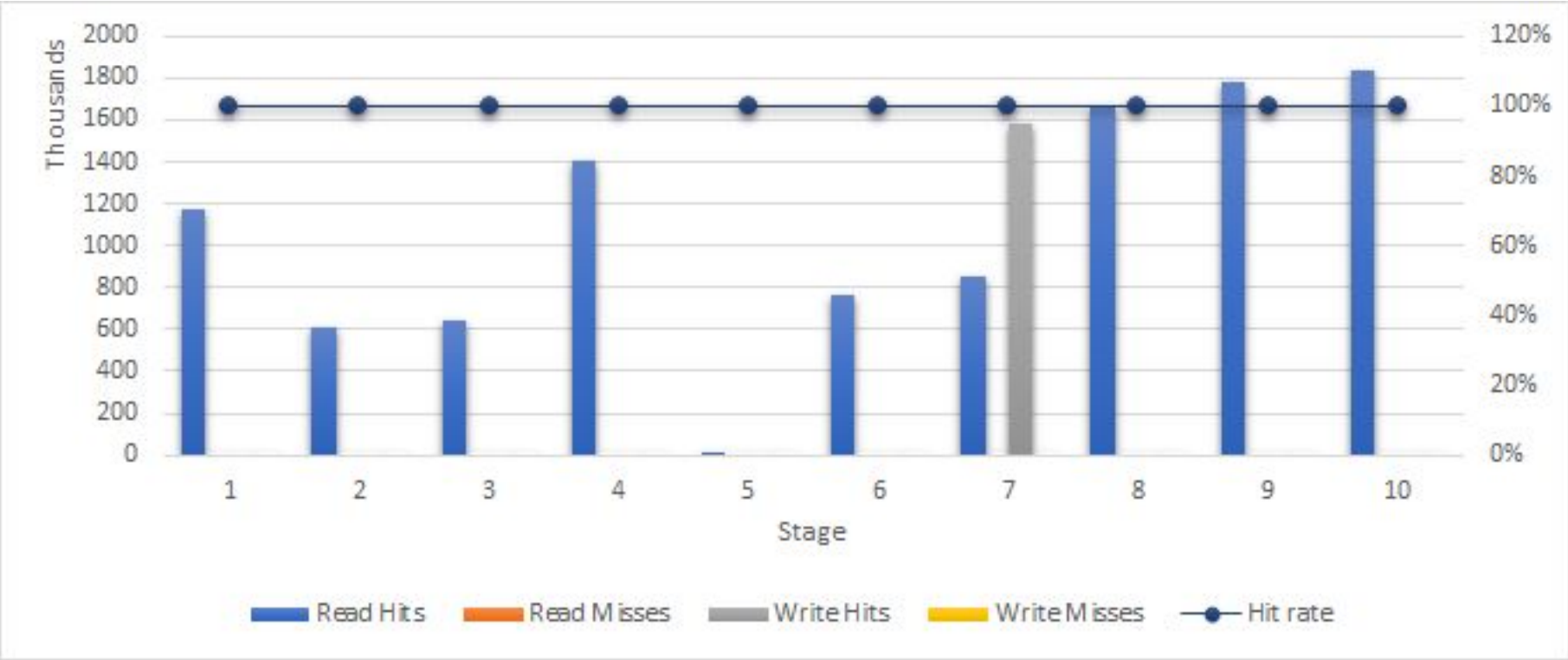
Experimental Setup

- Production P4 code for cloud data center
  - Two different P4 programs
- Various configuration and traffic scenarios
- L1 data cache hits and misses per stage
- Instruction cache hits and misses per MPU



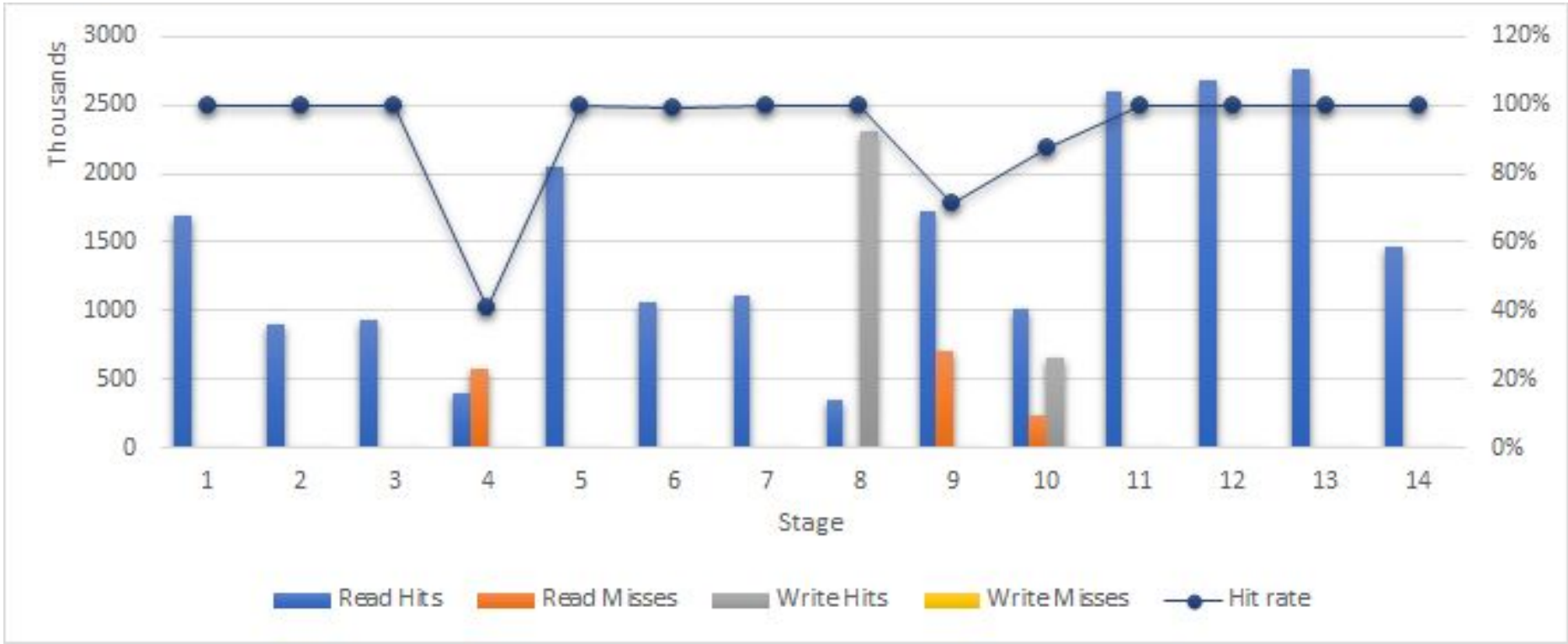
PENSANDO

1,000 active flows



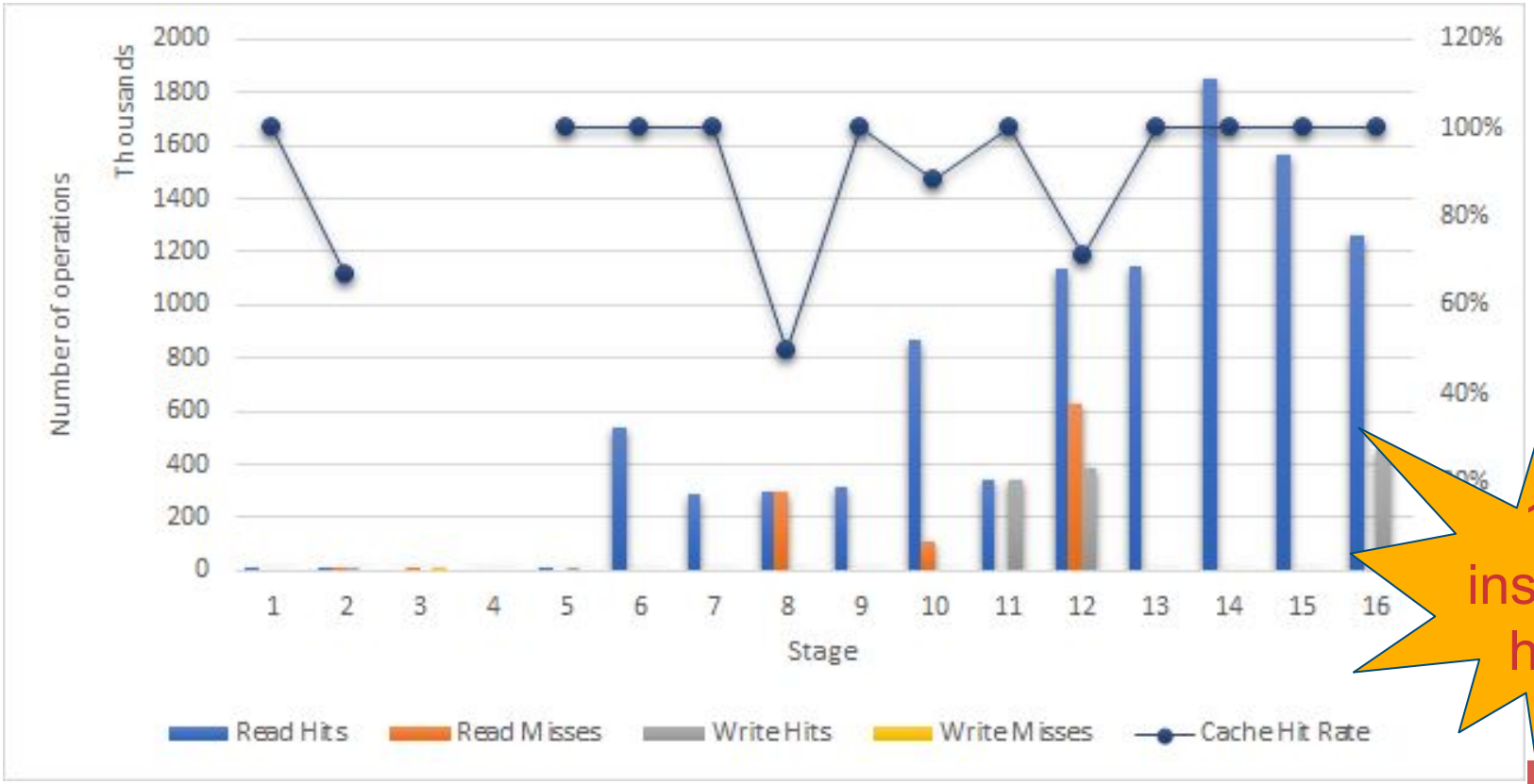
PENSANDO

1,800,000 active flows



PENSANDO

Other P4 Program



100% instruction hit rate

PENSANDO

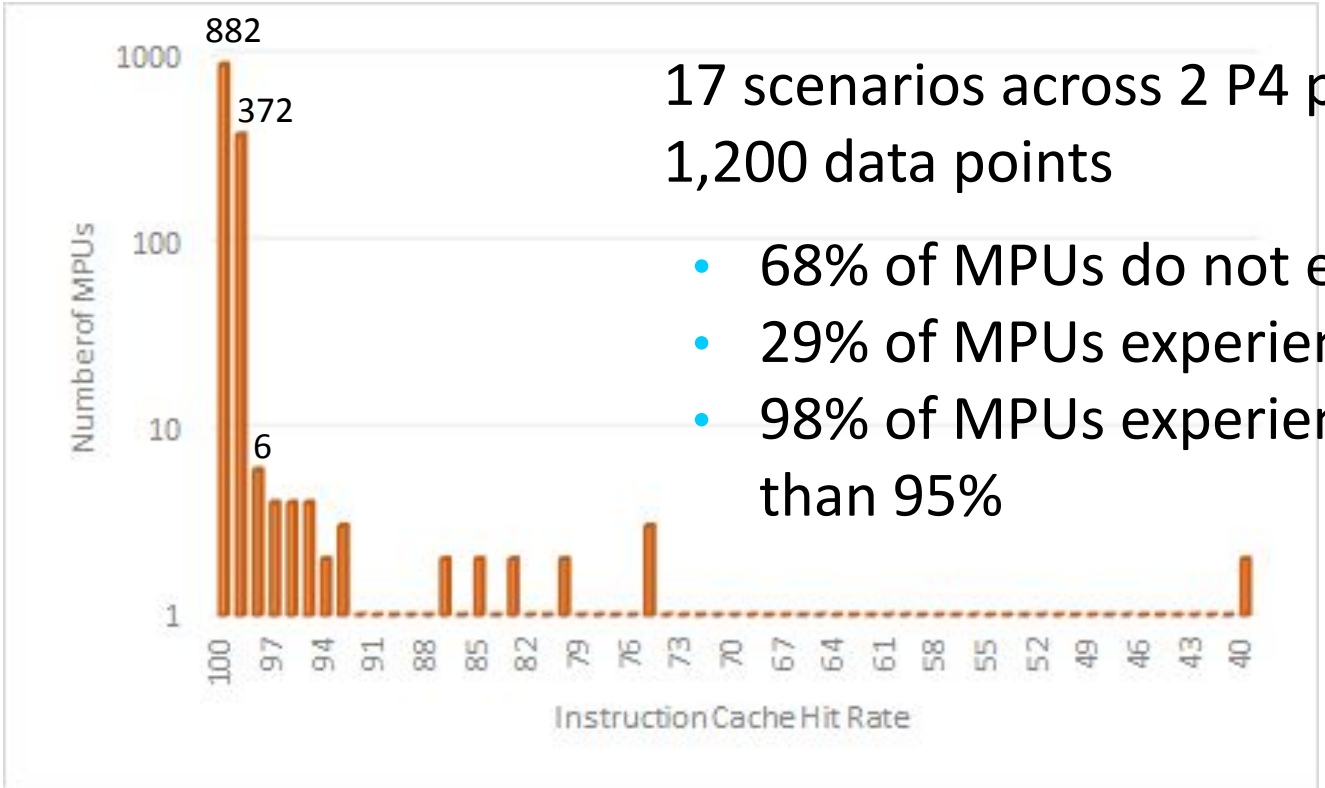
Instruction Cache

- Common actions compile to 30-50 instructions
- L1 cache can contain 2k 8-byte instructions
- Potentially 40-60 actions per stage

Instruction misses are extremely rare

PENSANDO

Instruction Cache



17 scenarios across 2 P4 programs, over 1,200 data points

- 68% of MPUs do not experience misses
- 29% of MPUs experience 1% misses
- 98% of MPUs experience hit rate higher than 95%

PENSANDO

Concluding Remarks

- P4 over pipelines for performance and flexibility
- Memory plays a key role in the performance and flexibility of target architectures
- A large central memory with a caching system successfully provides both
  - At least for a device at the network edge

PENSANDO

Thank You

baldi[at]pensando.io
pensando.io/blog

PENSANDO