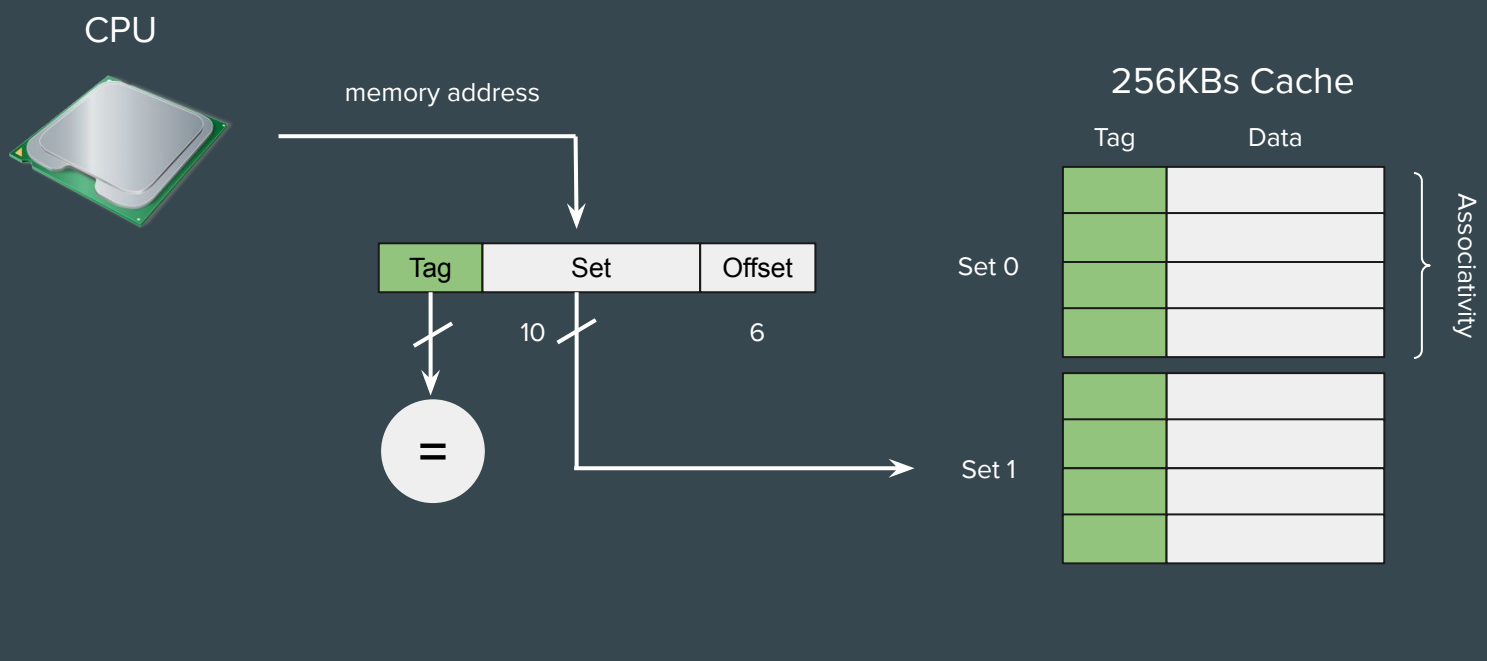
# CacheQuery: Learning Replacement Policies from Hardware Caches

...

**Pepe Vila**, Pierre Ganty, Marco Guarnieri, and Boris Köpf
*IMDEA Software Institute* *Microsoft Research*

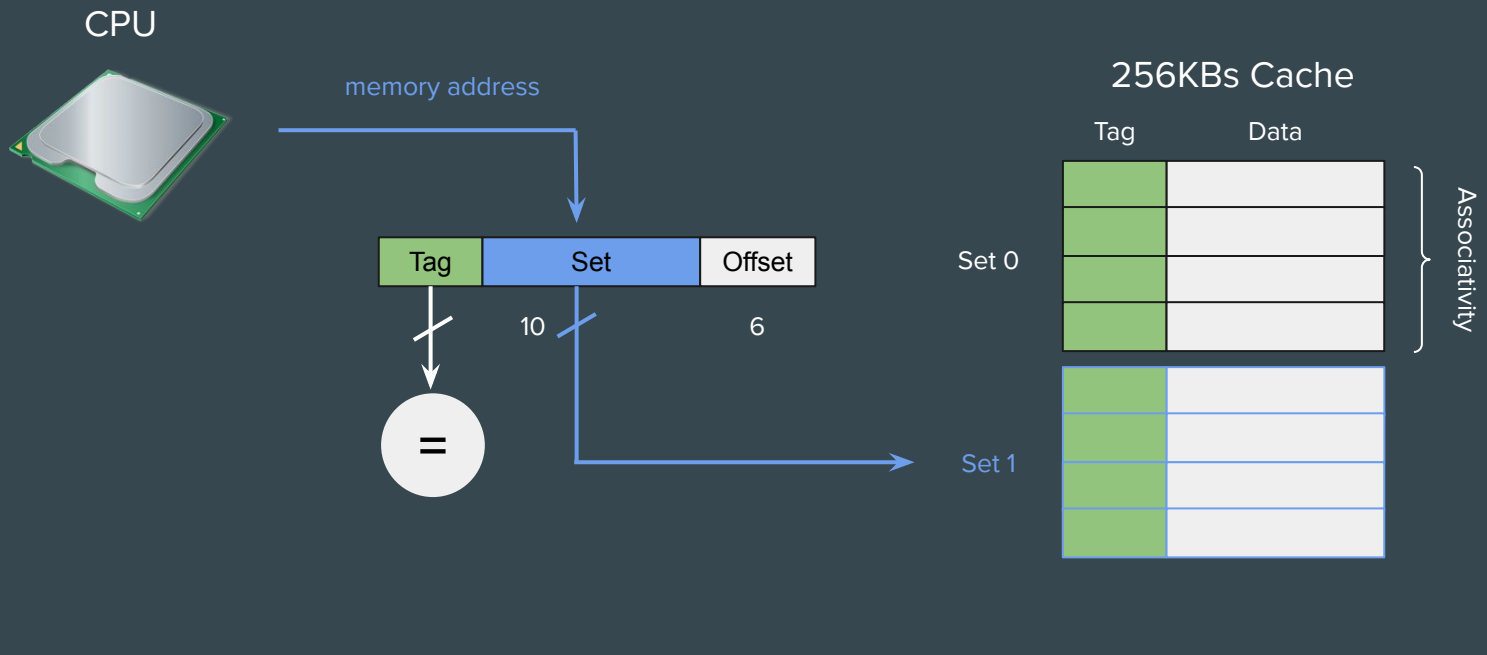
PLDI 2020 Synthesis II

# Caches: those little although faster friends...



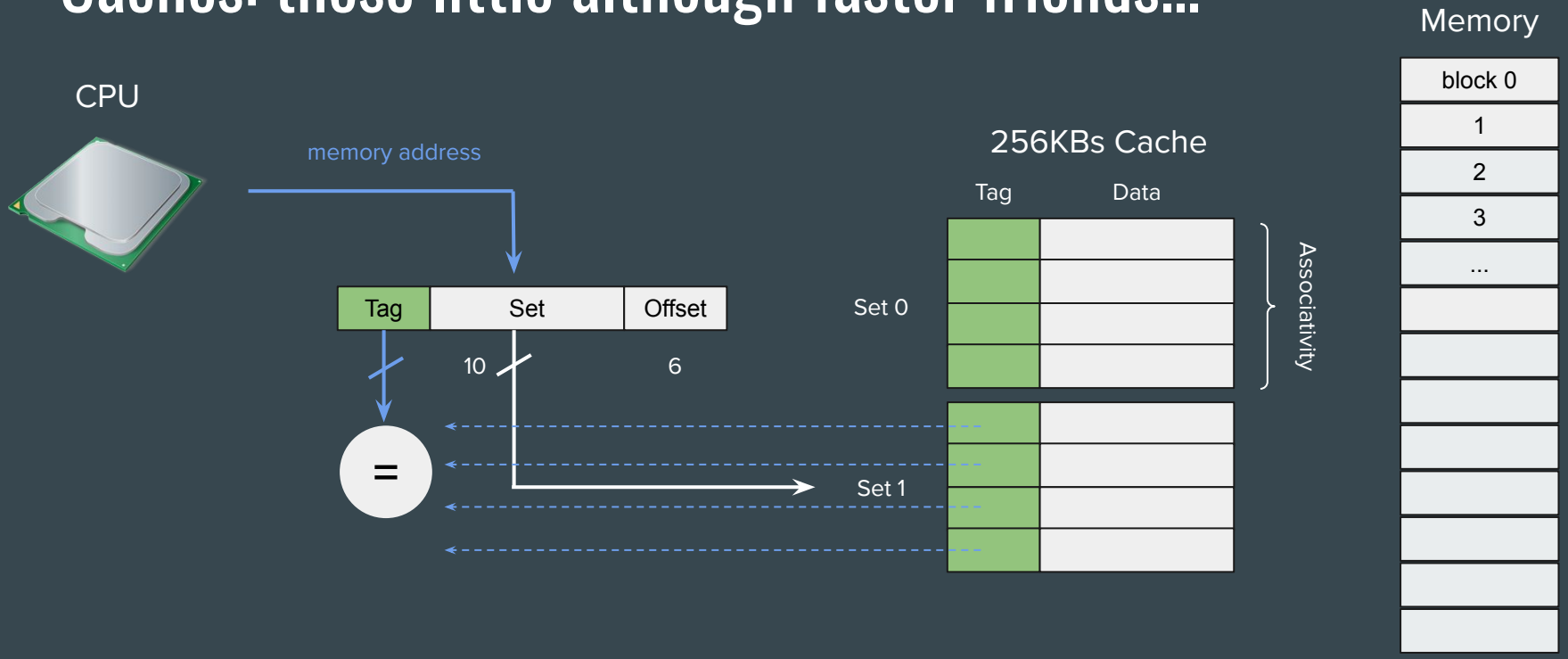

- Memory partitioned in **memory blocks** (64 bytes = $2^6$)
- Cache partitioned in equally sized **cache sets** (1024 = $2^{10}$ = 256KB / (64 * 4)
- Cache sets have capacity for N **cache lines** (also known as ways or **associativity**)

# Caches: those little although faster friends...



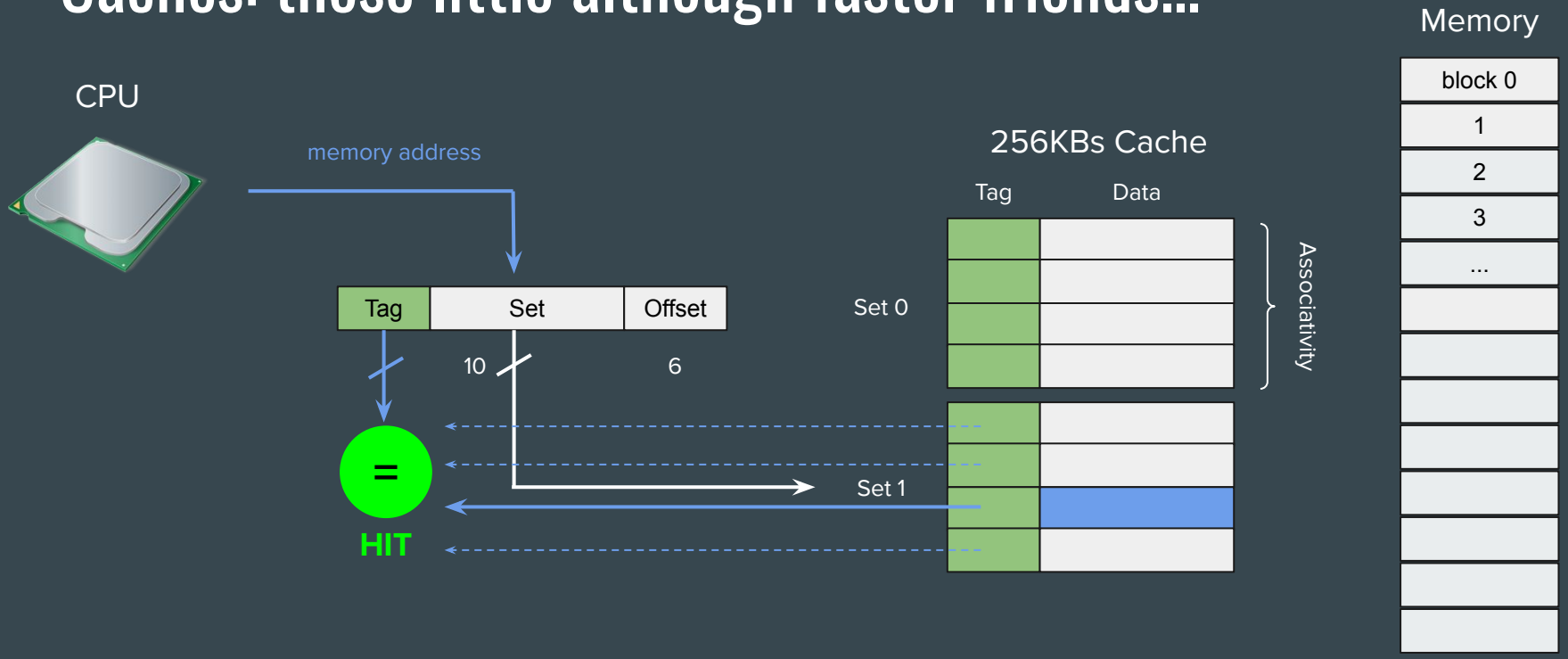

- Memory partitioned in **memory blocks** (64 bytes = $2^6$)
- Cache partitioned in equally sized **cache sets** (1024 = $2^{10}$ = 256KB / (64 * 4)
- Cache sets have capacity for N **cache lines** (also known as ways or **associativity**)

# Caches: those little although faster friends...



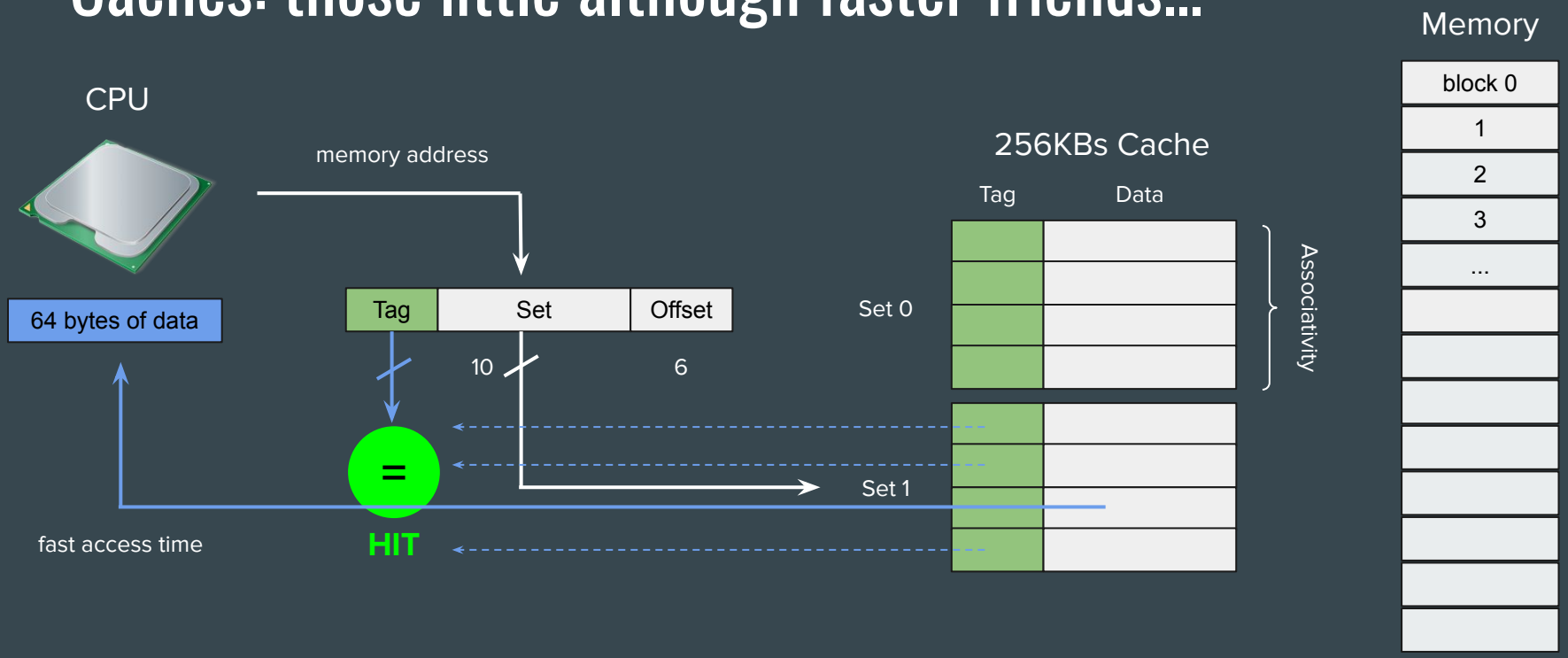

- Memory partitioned in **memory blocks** (64 bytes = $2^6$)
- Cache partitioned in equally sized **cache sets** (1024 = $2^{10}$ = 256KB / (64 * 4)
- Cache sets have capacity for N **cache lines** (also known as ways or **associativity**)

# Caches: those little although faster friends...



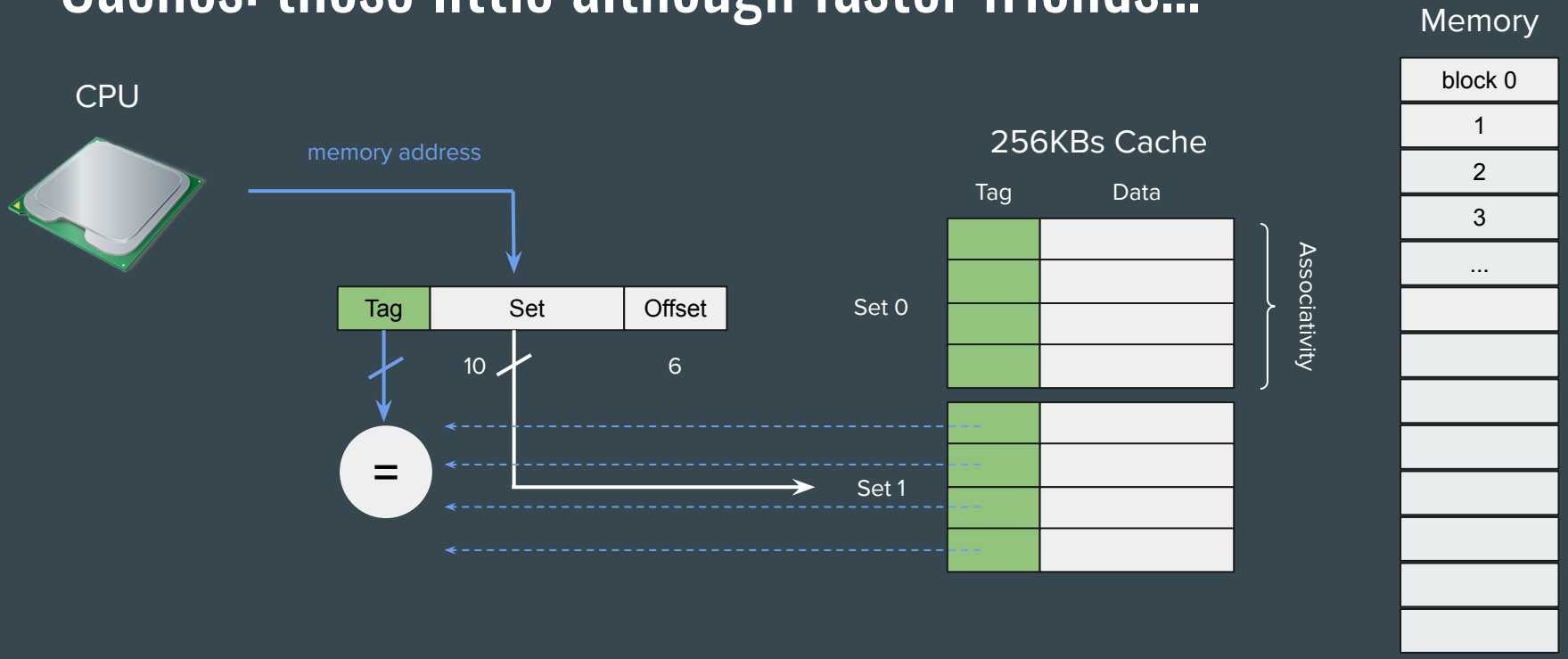

- Memory partitioned in **memory blocks** (64 bytes = $2^6$)
- Cache partitioned in equally sized **cache sets** (1024 = $2^{10}$ = 256KB / (64 * 4)
- Cache sets have capacity for N **cache lines** (also known as ways or **associativity**)

# Caches: those little although faster friends...



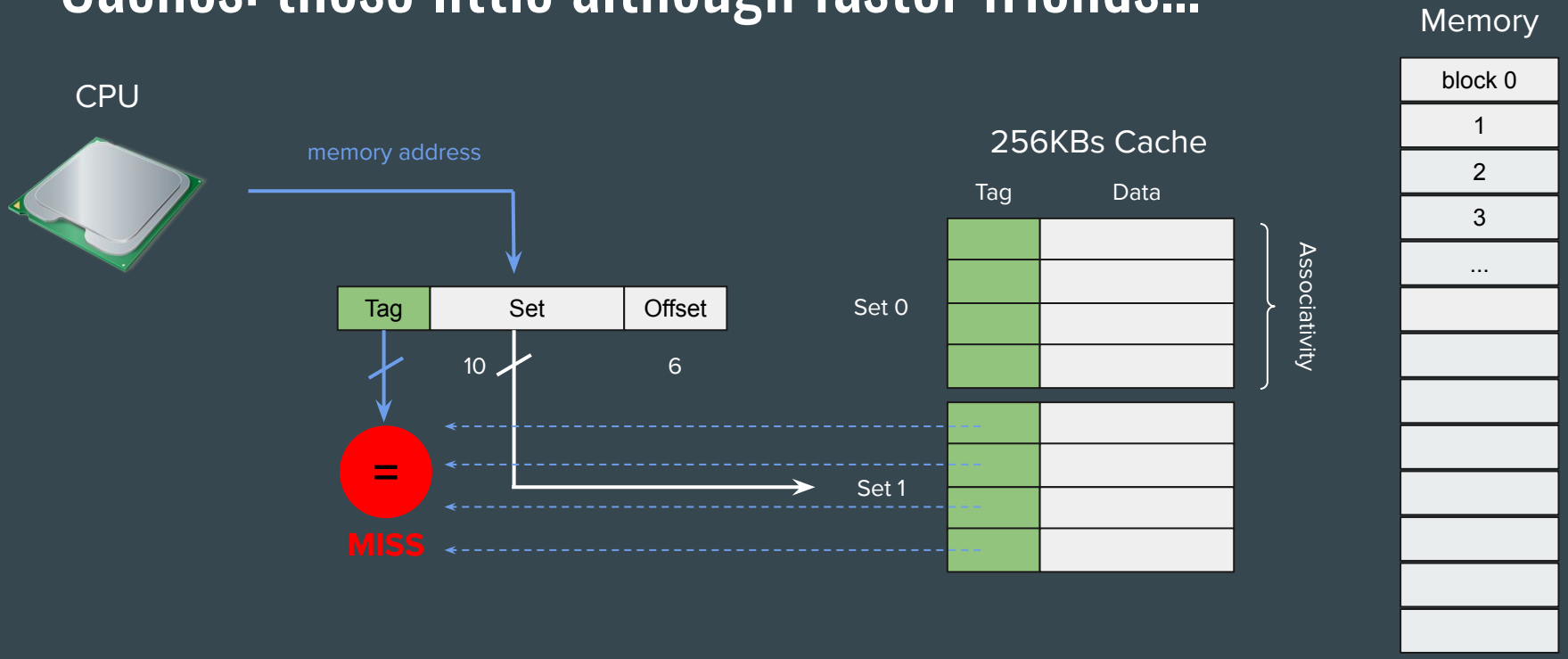

- Memory partitioned in **memory blocks** (64 bytes = $2^6$)
- Cache partitioned in equally sized **cache sets** (1024 = $2^{10}$ = 256KB / (64 * 4)
- Cache sets have capacity for N **cache lines** (also known as ways or **associativity**)

# Caches: those little although faster friends...



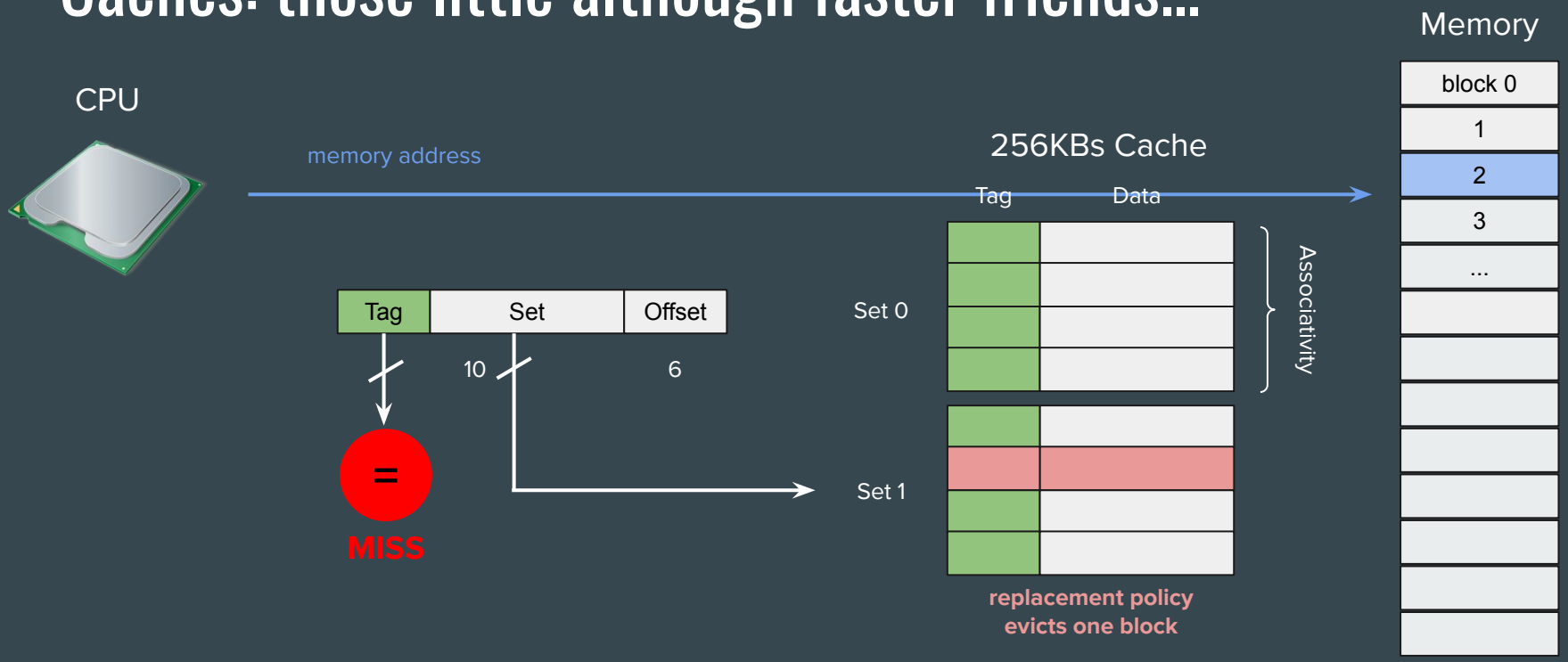

- Memory partitioned in **memory blocks** (64 bytes = $2^6$)
- Cache partitioned in equally sized **cache sets** (1024 = $2^{10}$ = 256KB / (64 * 4)
- Cache sets have capacity for N **cache lines** (also known as ways or **associativity**)

# Caches: those little although faster friends...



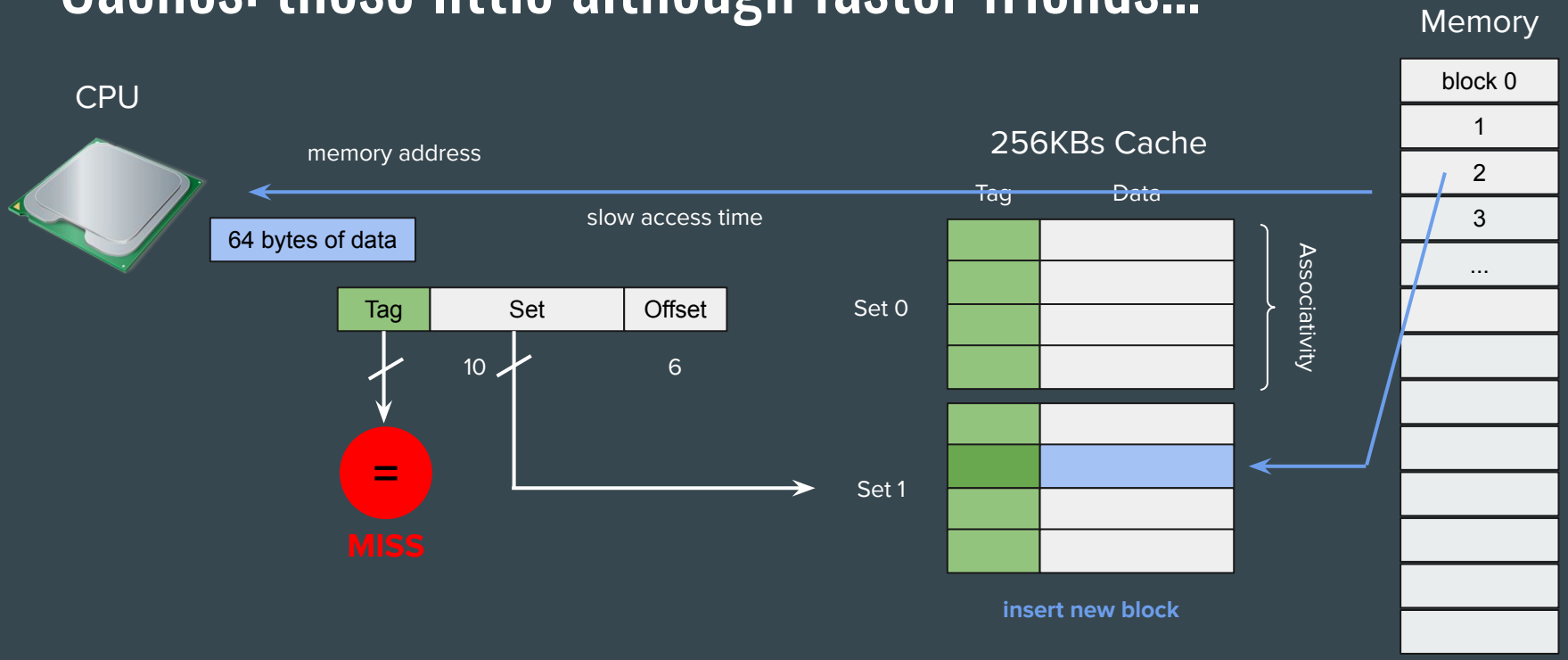

- Memory partitioned in **memory blocks** (64 bytes = $2^6$)
- Cache partitioned in equally sized **cache sets** (1024 = $2^{10}$ = 256KB / (64 * 4)
- Cache sets have capacity for N **cache lines** (also known as ways or **associativity**)

# Caches: those little although faster friends...



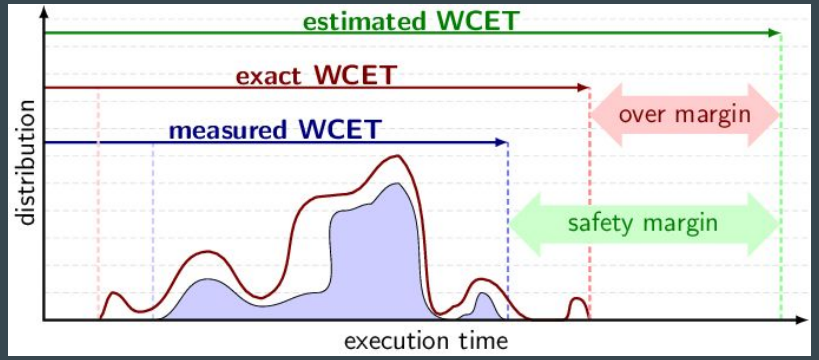

- Memory partitioned in **memory blocks** (64 bytes = $2^6$)
- Cache partitioned in equally sized **cache sets** (1024 = $2^{10}$ = 256KB / (64 * 4)
- Cache sets have capacity for N **cache lines** (also known as ways or **associativity**)

# Caches: those little although faster friends...




- Memory partitioned in **memory blocks** (64 bytes = $2^6$)
- Cache partitioned in equally sized **cache sets** (1024 = $2^{10}$ = 256KB / (64 * 4)
- Cache sets have capacity for N **cache lines** (also known as ways or **associativity**)

# Caches: their importance and impact




# Problem: cache as a black box

MEMORY ADDRESSES

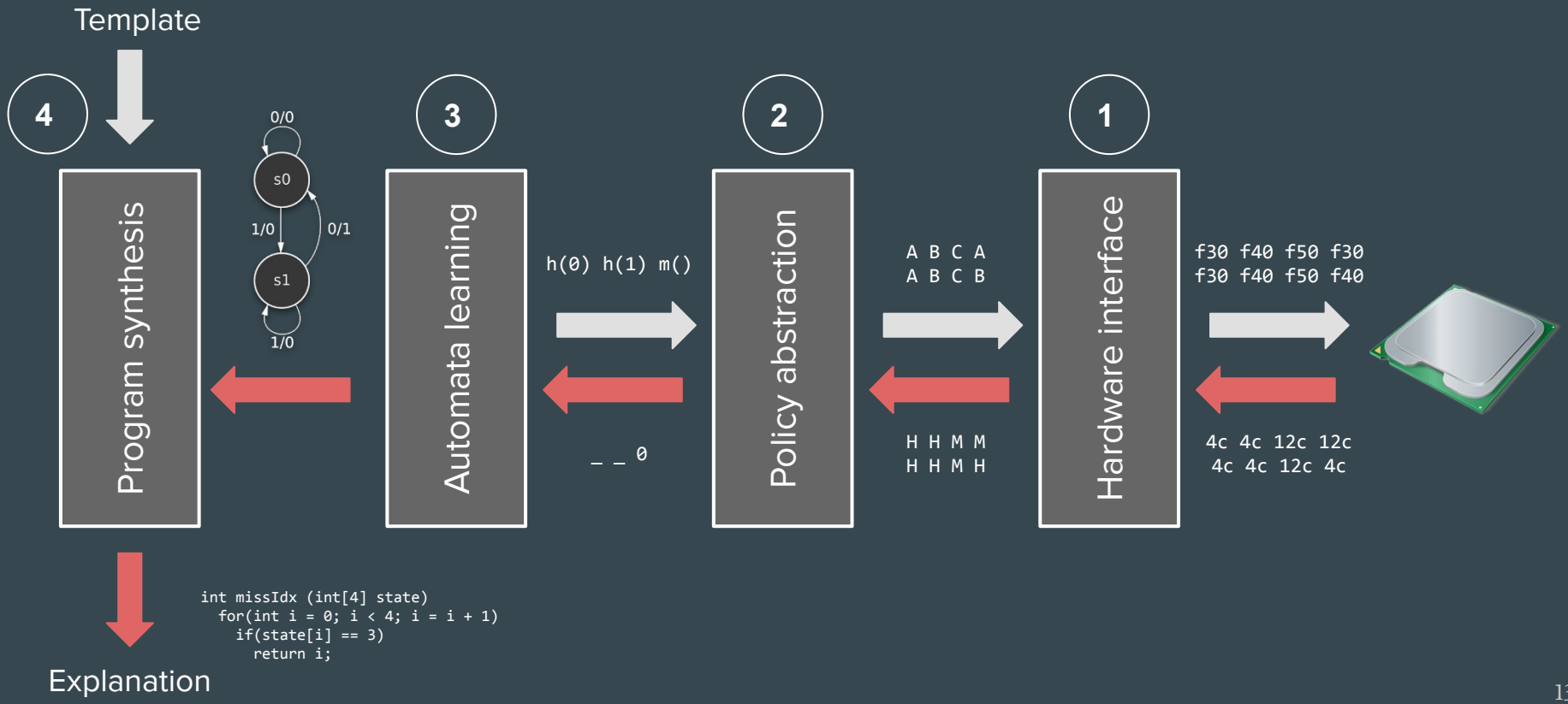
f30 f40 f50 f30

BLACKBOX CACHE

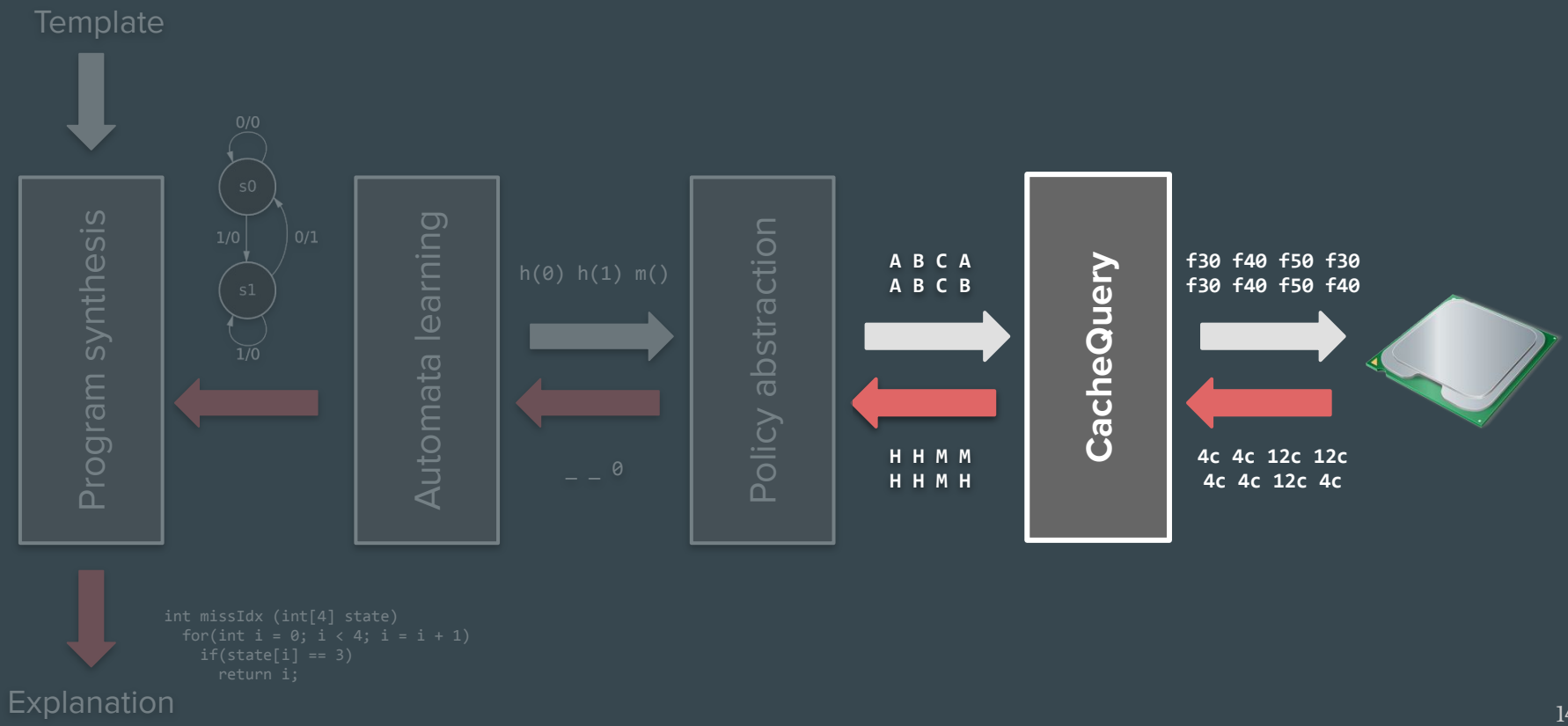
TIME MEASUREMENTS

15 16 14 4

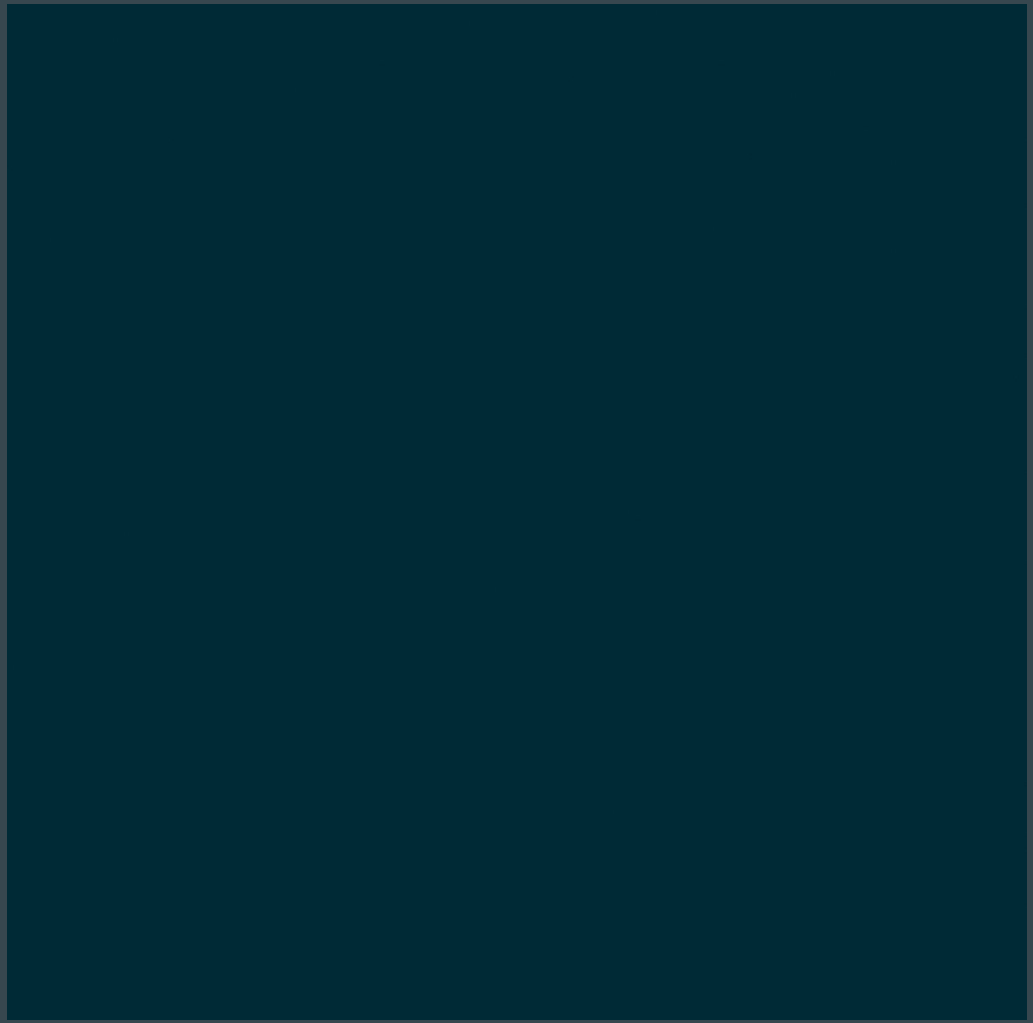
# Our approach for learning replacement policies



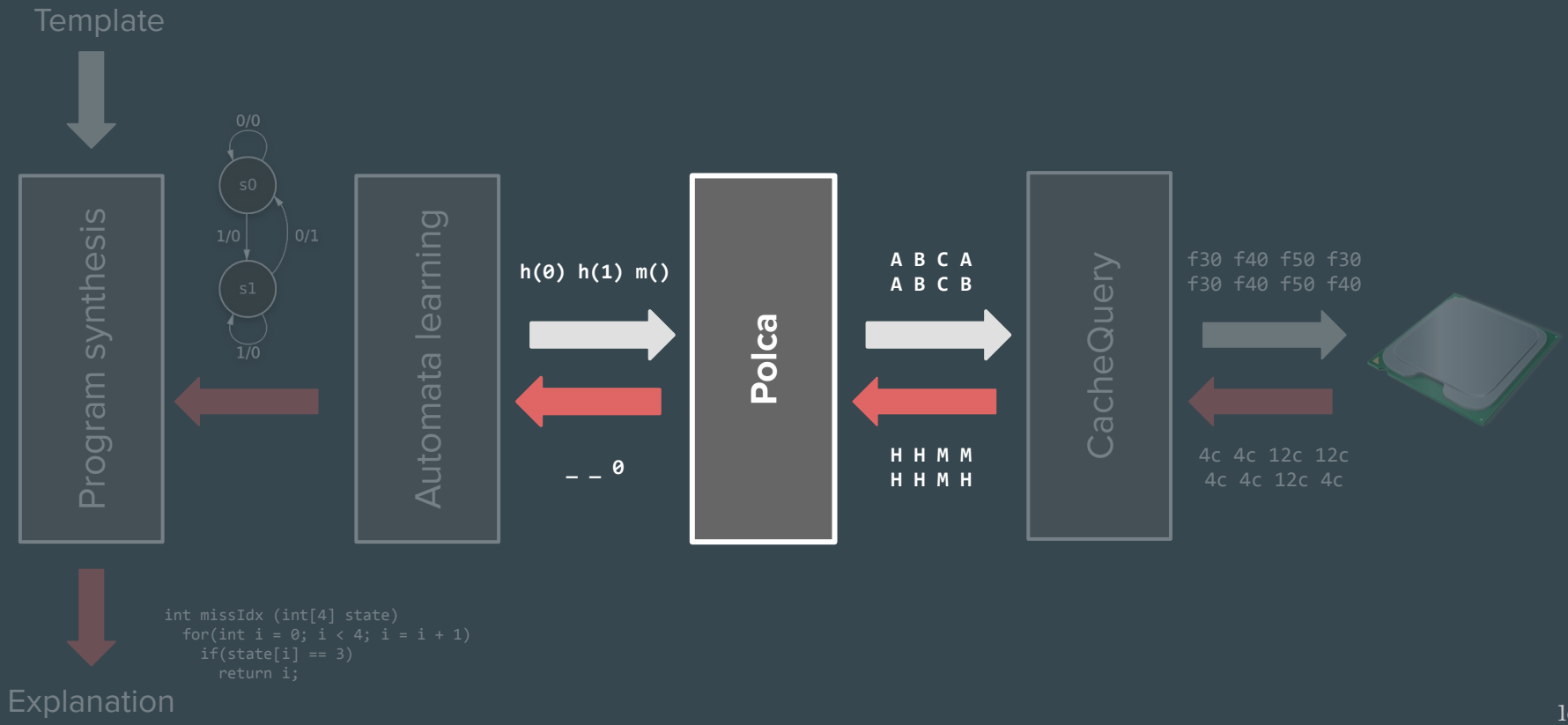
# CacheQuery: a hardware interface



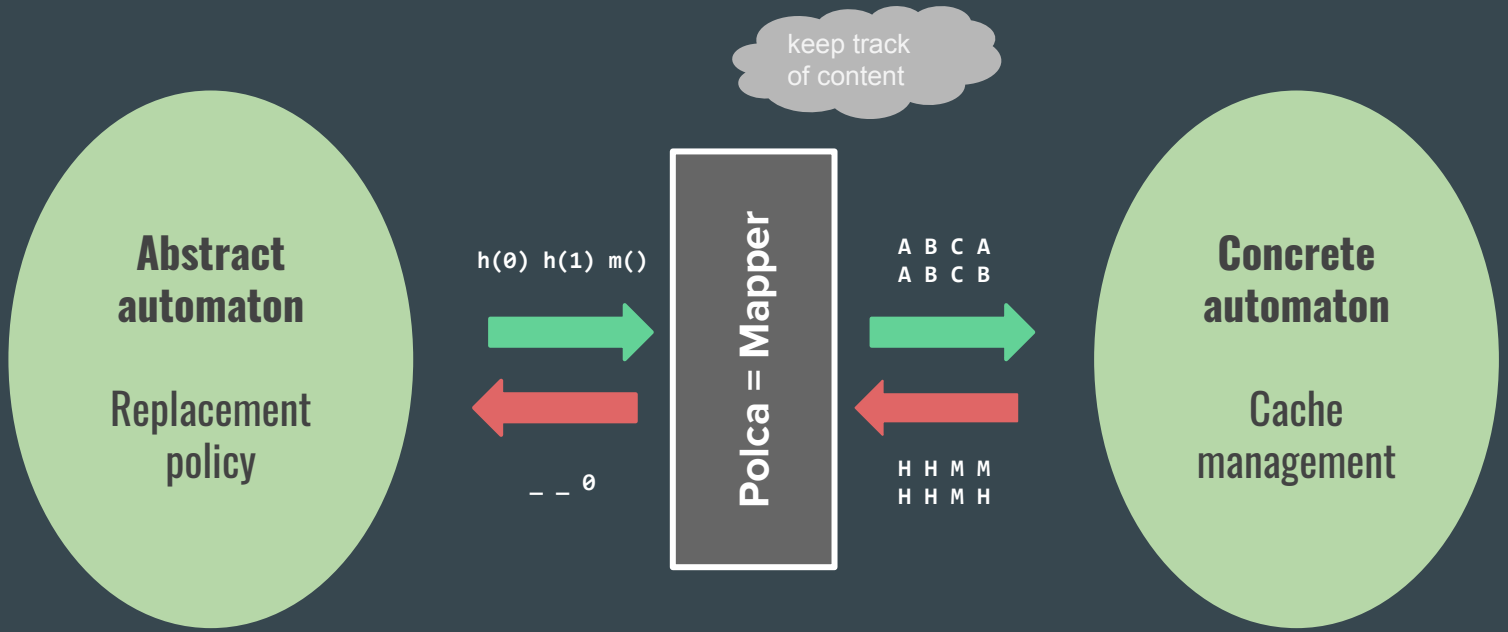
# CacheQuery: a hardware interface



# Polca: a cache policy automaton abstraction



# Polca: a cache policy automaton abstraction



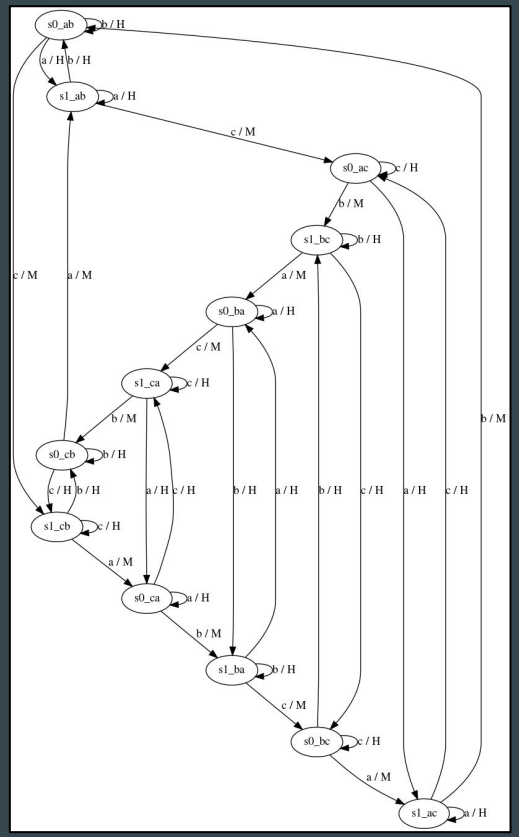

Input: {h(0), h(1), ..., h(n-1), m()}

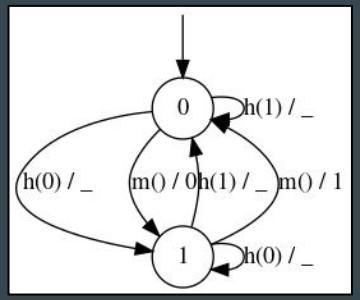
Output: {_, 0, 1, ..., n-1}

{A, B, C, ....}

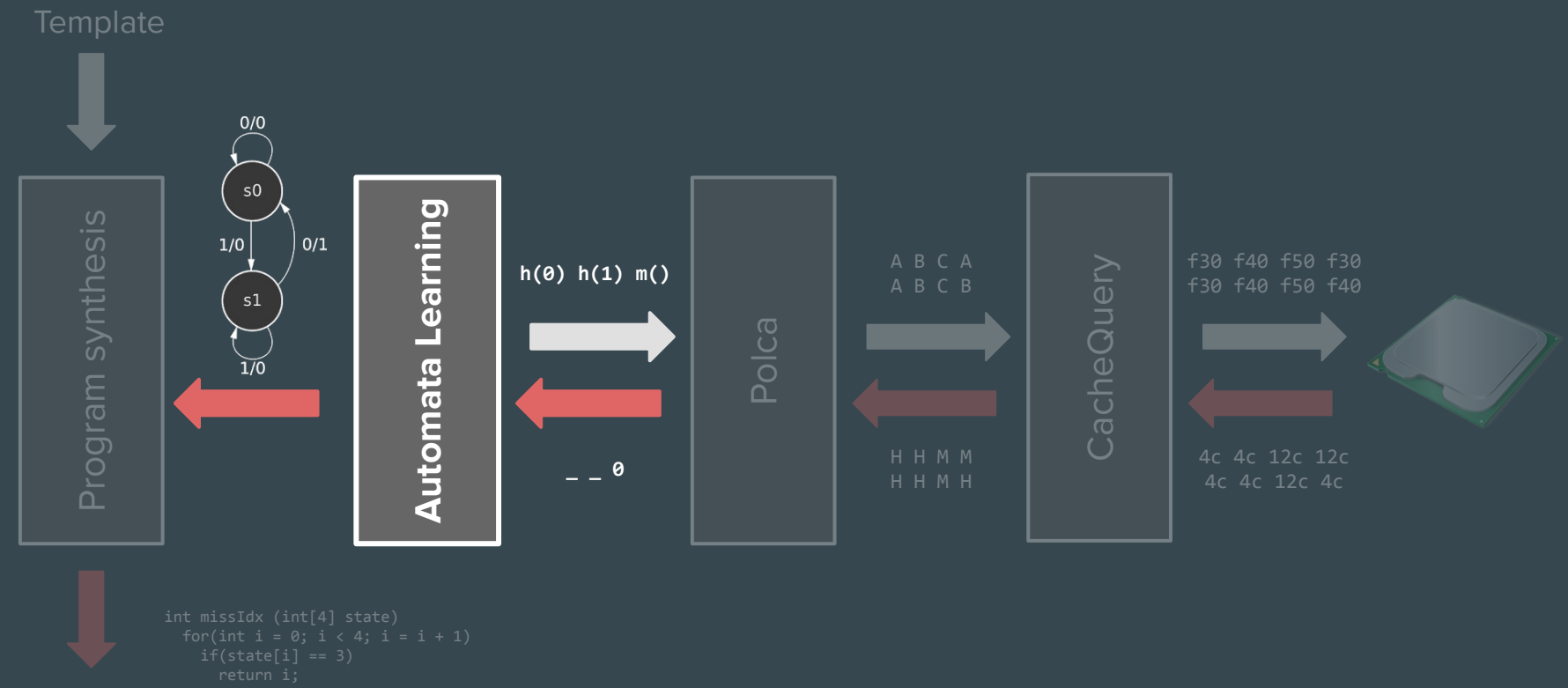
{H, M}

# Caches: those little although faster friends...



# LearnLib: an automata learning framework

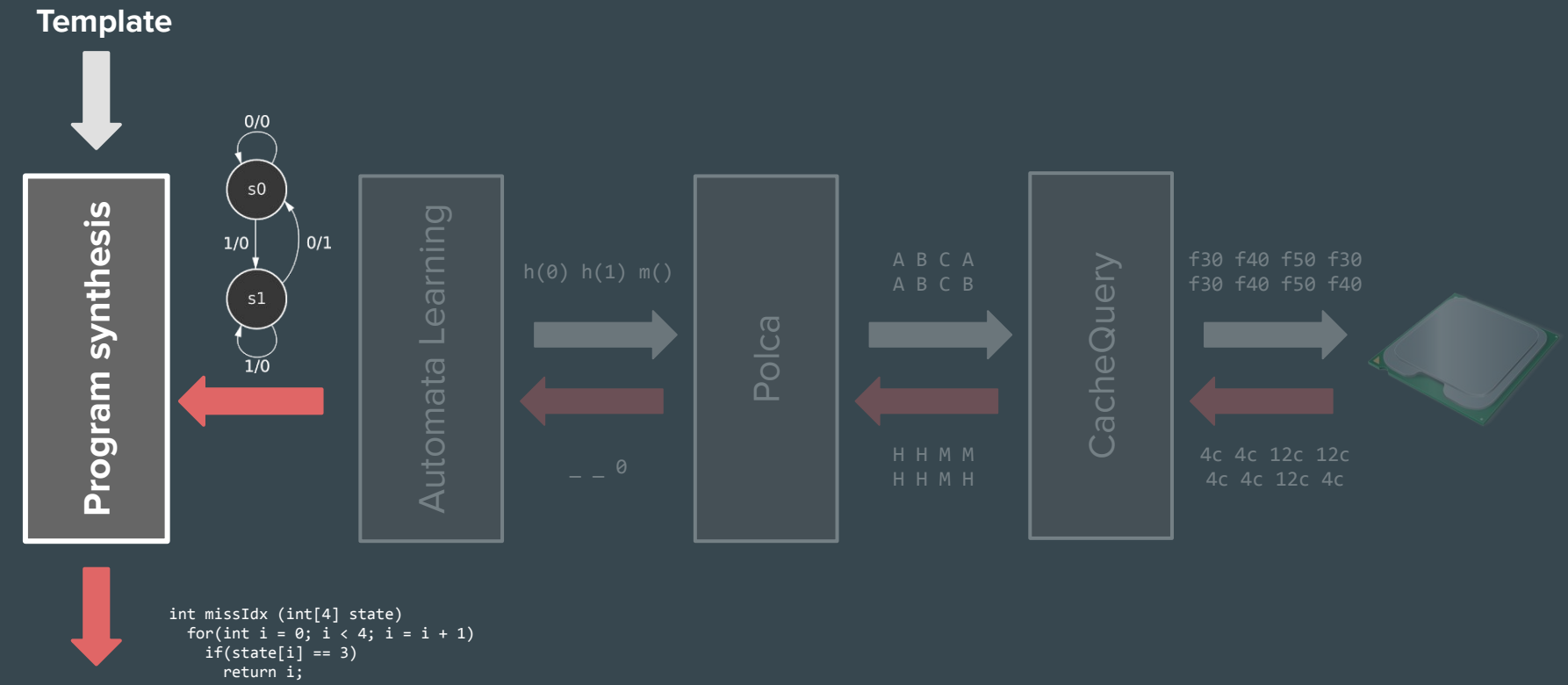


# LearnLib: an automata learning framework

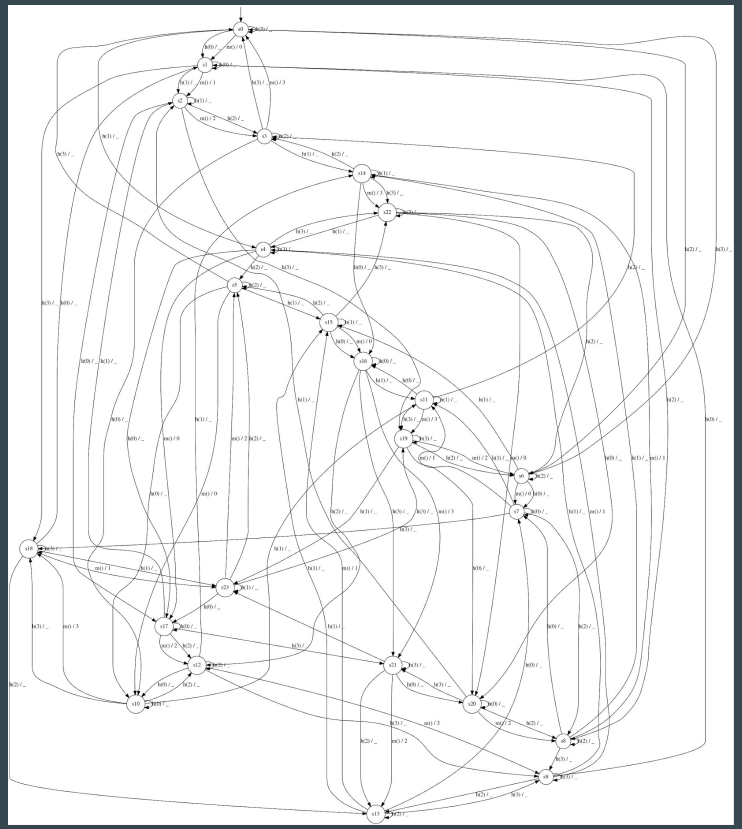
- **LearnLib** is an open source Java framework for automata learning developed at the TU Dortmund - https://learnlib.de/
- Angluin's L* algorithm has been extended to **Mealy machines**:
  - Membership queries replaced by **output queries**
  - Equivalence queries **approximated by test sequences** for conformance testing
  - **Reset sequence** is bootstrapping problem, we solve it with Flush+Refill

**WP-method:** test sequence selection - given an upper bound on the number of states of the System Under Learning (SUL), guarantees equivalence

# Sketch: synthesizing programs as explanations



# Sketch: synthesizing programs as explanations



# Sketch: synthesizing programs as explanations

Domain knowledge or high-level view of a replacement policy:

- Each block has an associated **age**
- **Promotion** rule decides how the ages are updated upon a hit
- **Replacement** rule decides which block is evicted upon a miss
- **Insertion** rule decides the age of a new block

We use it to “sketch” a template for replacement policies
and encode the automaton’s output and transition functions as constraints!

# Sketch: synthesizing programs as explanations

```
hit (state, line) :: States×Lines → States
  state = promote(state, line)
  state = normalize(state, line)
  return state
```

```
miss (state) :: States → States×Lines
  Lines idx = -1
  state = normalize(state, idx)
  idx = evict(state)
  state[idx] = insert(state, idx)
  state = normalize(state, idx)
  return ⟨state, idx⟩
```

# Sketch: synthesizing programs as explanations

```
hit (state, line) :: States×Lines → States
  state = promote(state, line)
  state = normalize(state, line)
  return state
```

```
miss (state) :: States → States×Lines
  Lines idx = -1
  state = normalize(state, idx)
  idx = evict(state)
  state[idx] = insert(state, idx)
  state = normalize(state, idx)
  return ⟨state, idx⟩
```

```
promote (state, pos) :: States×Lines → States
  States final = state
  if (??{boolExpr(state[pos])})
    final[pos] = ??{natExpr(state[pos])}
  for(i in Lines)
    if(i != pos ∧ ??{boolExpr(state[pos], state[i])})
      final[i] = ??{natExpr(state[i])}
  return final
```

# Results

# Results

## Description of Skylake/Kaby Lake L3's (*New2*):

Initial insertion on a flushed cache set:

```
int[4] s0 = {3,3,3,3};
```

```
int[4] hitState (int[4] state, int pos)
  int[4] final = state;
  // Promotion
  if (final[pos] > 1)
    final[pos] = 1;
  else
    final[pos] = 0;
  // Is there a block with age 3?
  bit found = 0;
  for(int j = 0; j < 4; j = j + 1)
    if(!found)
      for(int i = 0; i < 4; i = i + 1)
        if(!found && final[i] == 3)
            found = 1;
    // If not, increase all blocks
    if(!found)
      for(int i = 0; i < 4; i = i + 1)
          final[i] = final[i] + 1;
  return final;
```

```
int[4] missState (int[4] state)
  int[4] final = state;
  int replace = missIdx(state);
  // Insertion
  final[replace] = 1;
  // Is there a block with age 3?
  bit found = 0;
  for(int j = 0; j < 4; j = j + 1)
    if(!found)
      for(int i = 0; i < 4; i = i + 1)
        if(!found && final[i] == 3)
          found = 1;
    // If not, increase all blocks
    if(!found)
      for(int i = 0; i < 4; i = i + 1)
          final[i] = final[i] + 1;
  return final;


// Replace first block with age 3 starting from the left
int missIdx (int[4] state)
  for(int i = 0; i < 4; i = i + 1)
    if(state[i] == 3)
      return i;
```

# Thank you for listening! Questions?


https://github.com/cgvwzq/cachequery

https://github.com/cgvwzq/polca

https://arxiv.org/pdf/1912.09770.pdf