# Exploiting Modern Hardware Features via Lightweight Profiling

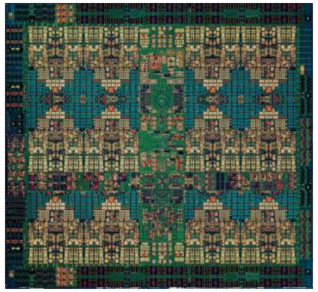
Probir Roy

Scalable Tools Workshop’19

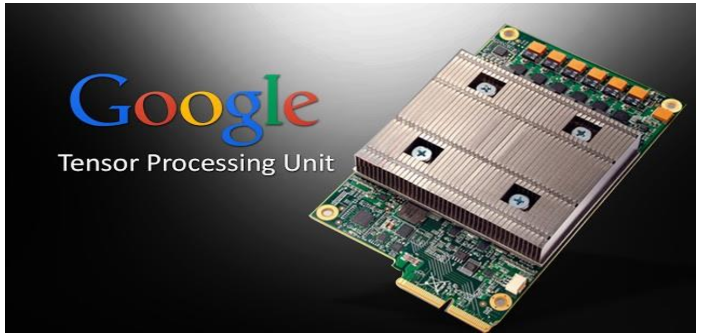
WILLIAM & MARY

CHARTERED 1693

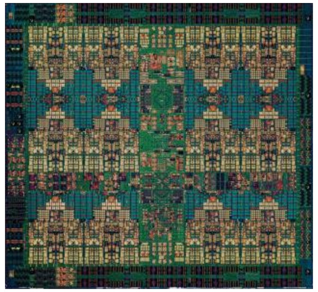
# High performance and challenges



IBM POWER 9 CPU

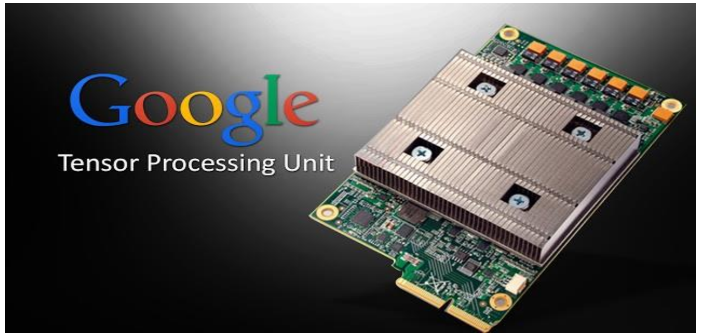


# High performance and challenges

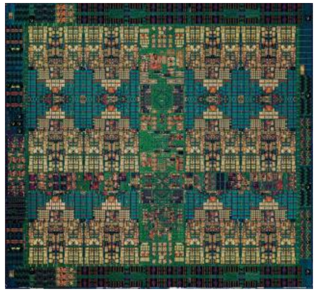


IBM POWER 9 CPU

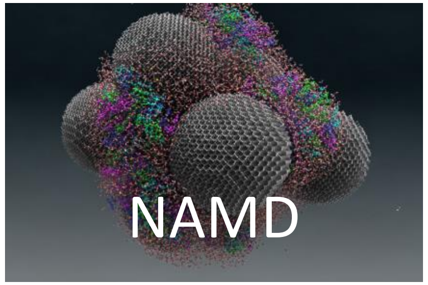
Amazon CloudFront



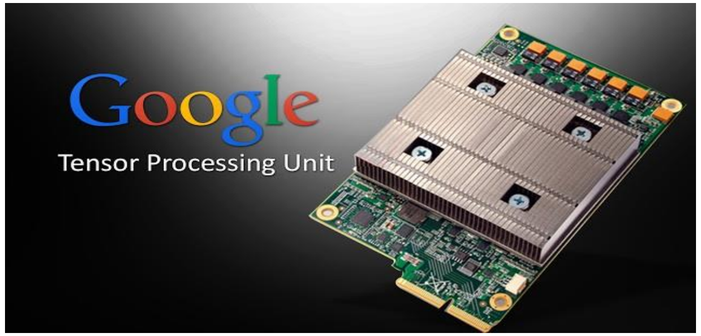
# High performance and challenges

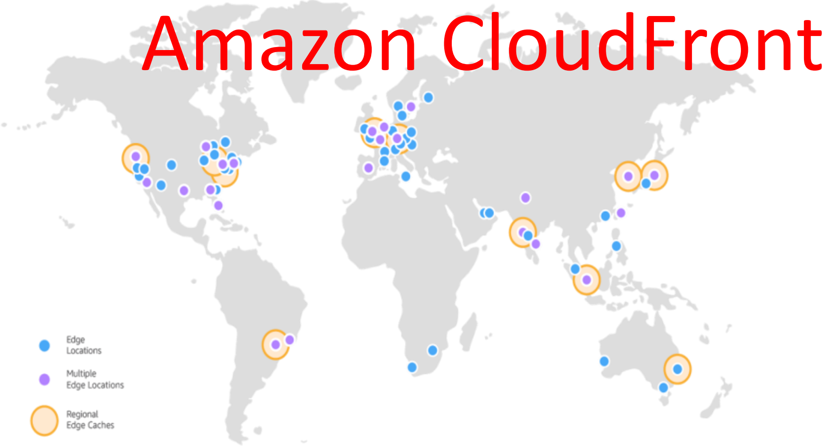


IBM POWER 9 CPU



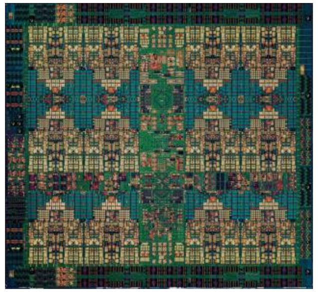
NAMD



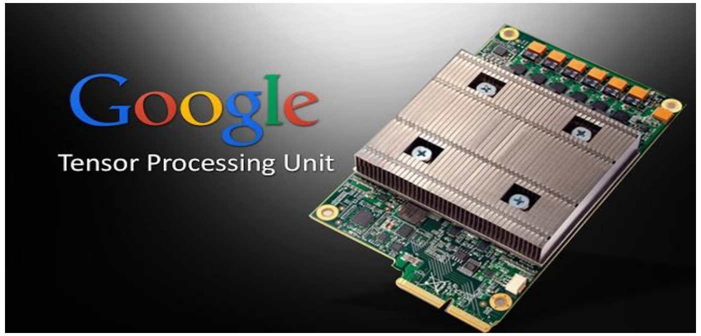


Amazon CloudFront

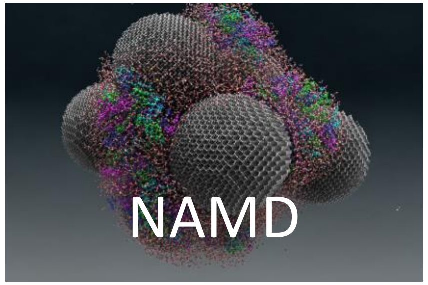
# High performance and challenges



IBM POWER 9 CPU



Amazon CloudFront



OpenMP

MPI

cassandra

redis

julia

TensorFlow

# High performance and challenges

Common Goal: Performance

IBM POWER 9 CPU

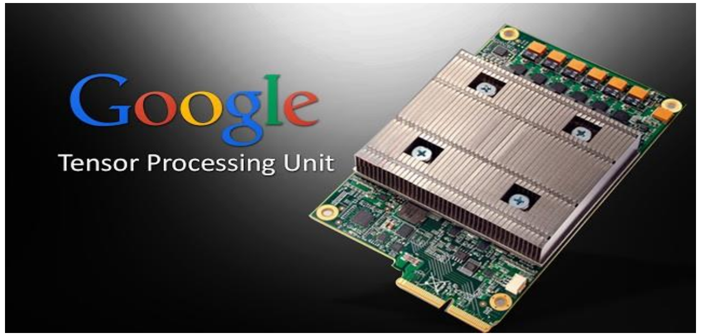
OpenMP

MPI

Amazon CloudFront

cassandra

redis



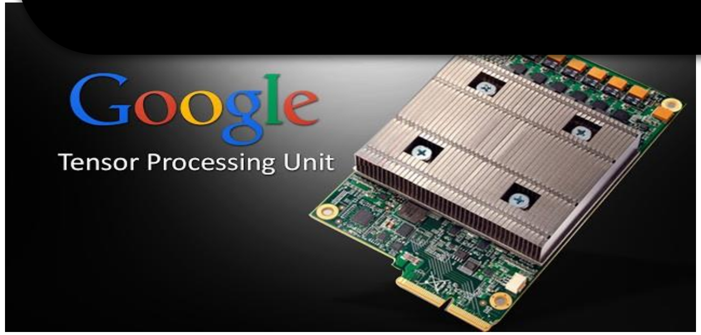
julia

TensorFlow

# High performance and challenges

Common Goal: Performance

Variable characteristics of hardware and software is a challenge



julia

TensorFlow

# High performance and challenges

Common Goal: Performance

Variable characteristics of hardware and software is a challenge

Deep insights

# High performance and challenges

http://sites.utexas.edu/jdm4372/2016/11/22/sc16-invited-talk-memory-bandwidth-and-system-balance-in-hpc-systems/

# High performance and challenges

Peak FLOPS per socket increasing at **50%-60% per year**

http://sites.utexas.edu/jdm4372/2016/11/22/sc16-invited-talk-memory-bandwidth-and-system-balance-in-hpc-systems/

# High performance and challenges

Peak FLOPS per socket increasing at **50%-60% per year**

Memory bandwidth increasing at **~23% per year**

http://sites.utexas.edu/jdm4372/2016/11/22/sc16-invited-talk-memory-bandwidth-and-system-balance-in-hpc-systems/

# High performance and challenges

Peak FLOPS per socket increasing at **50%-60% per year**

Memory bandwidth increasing at **~23% per year**

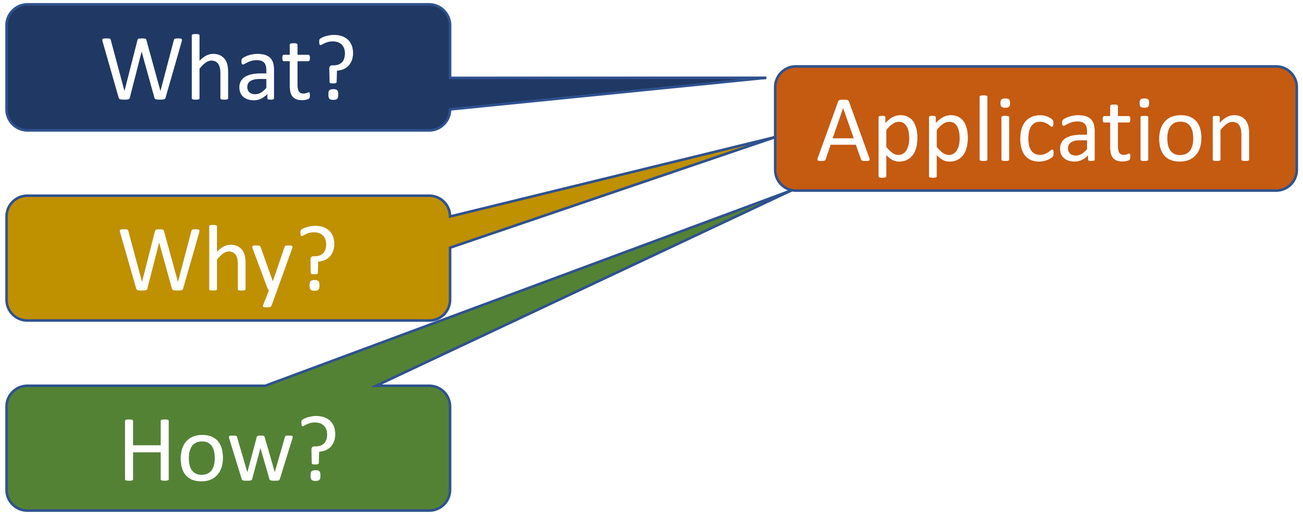
Memory latency increasing at **~4% per year**

http://sites.utexas.edu/jdm4372/2016/11/22/sc16-invited-talk-memory-bandwidth-and-system-balance-in-hpc-systems/

# Steps of performance analysis

Application

# Steps of performance analysis



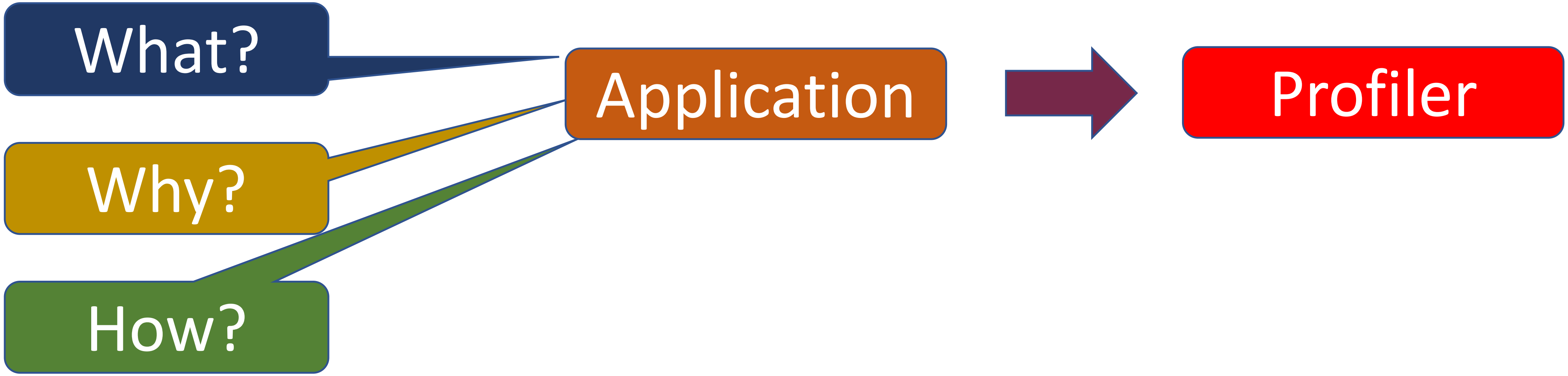
What?

Why?

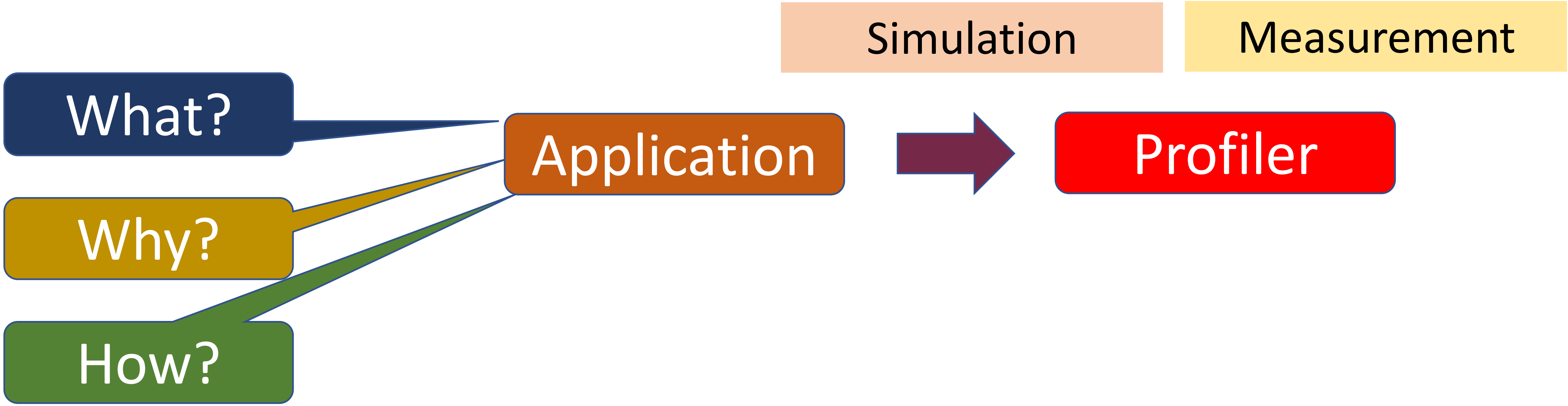
How?

Application

# Steps of performance analysis



# Steps of performance analysis



Simulation

Measurement

What?

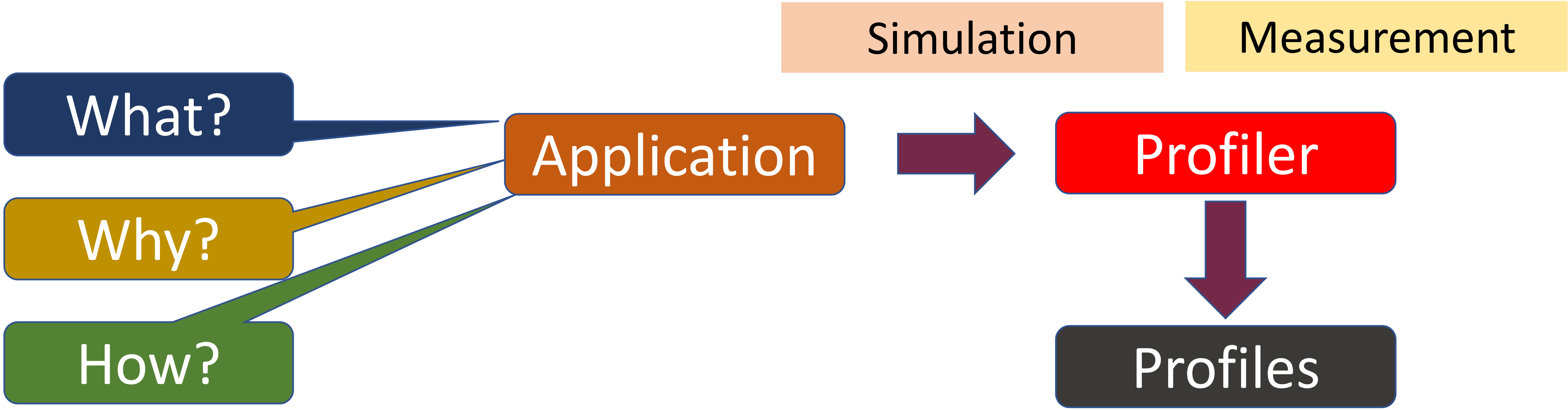
Why?

How?

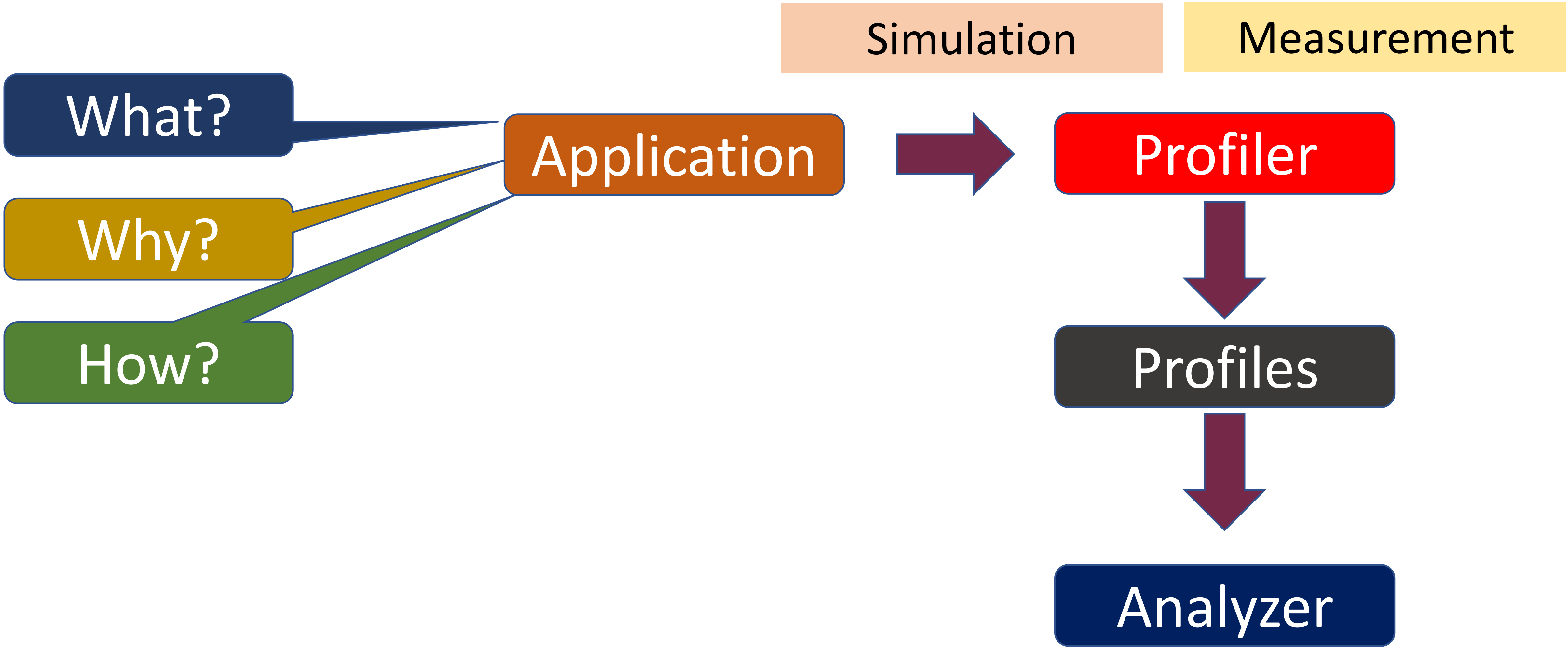
Application

Profiler

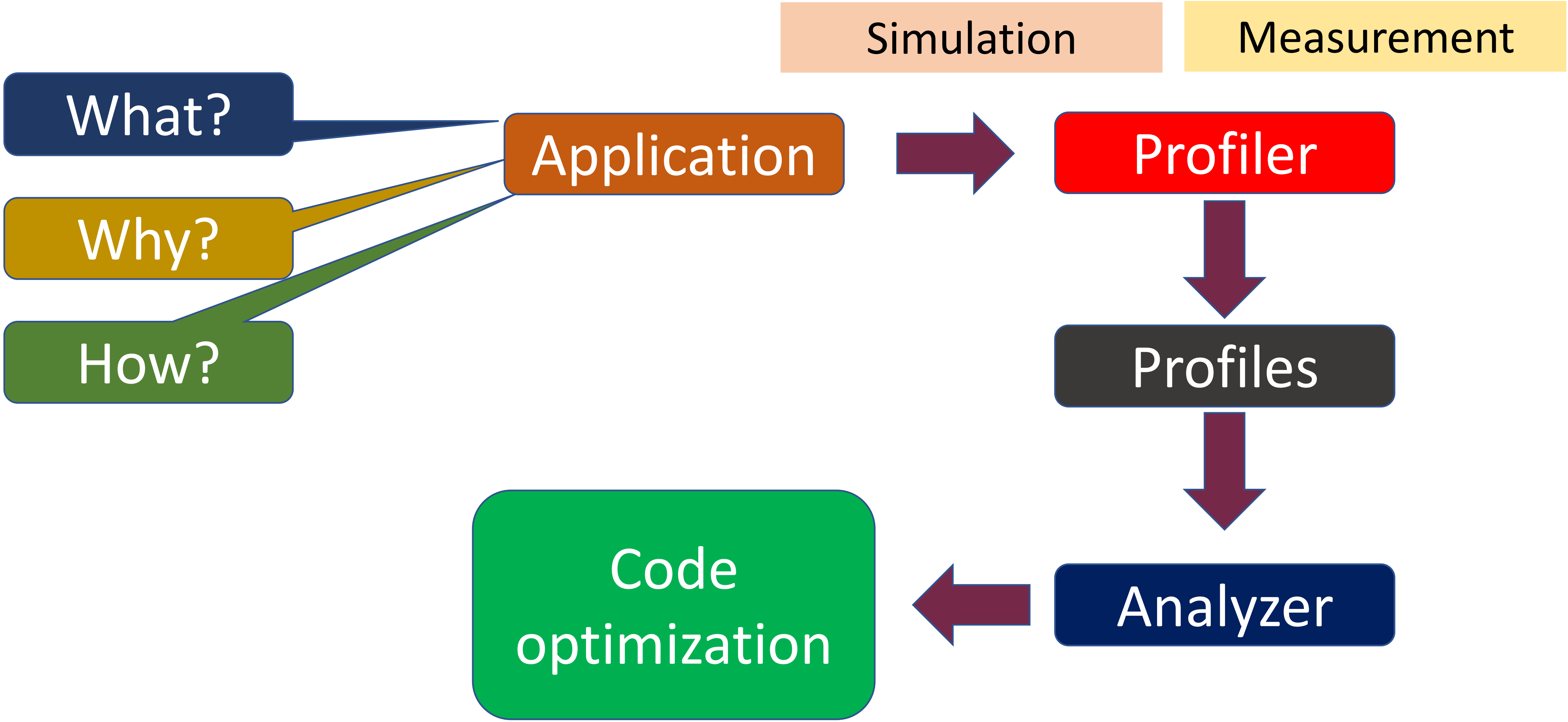
# Steps of performance analysis



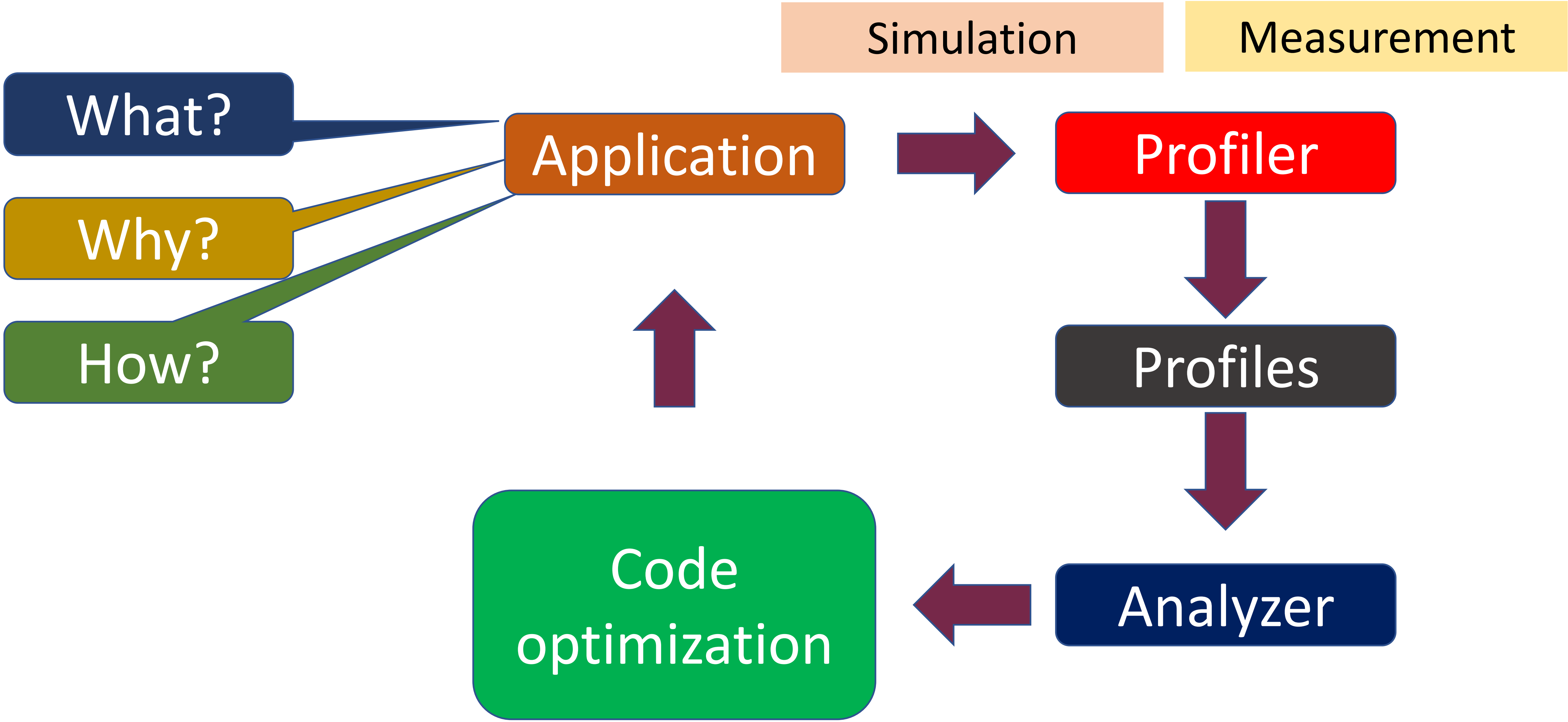
# Steps of performance analysis



# Steps of performance analysis



# Steps of performance analysis



Simulation

Measurement

What?

Why?

How?

Application

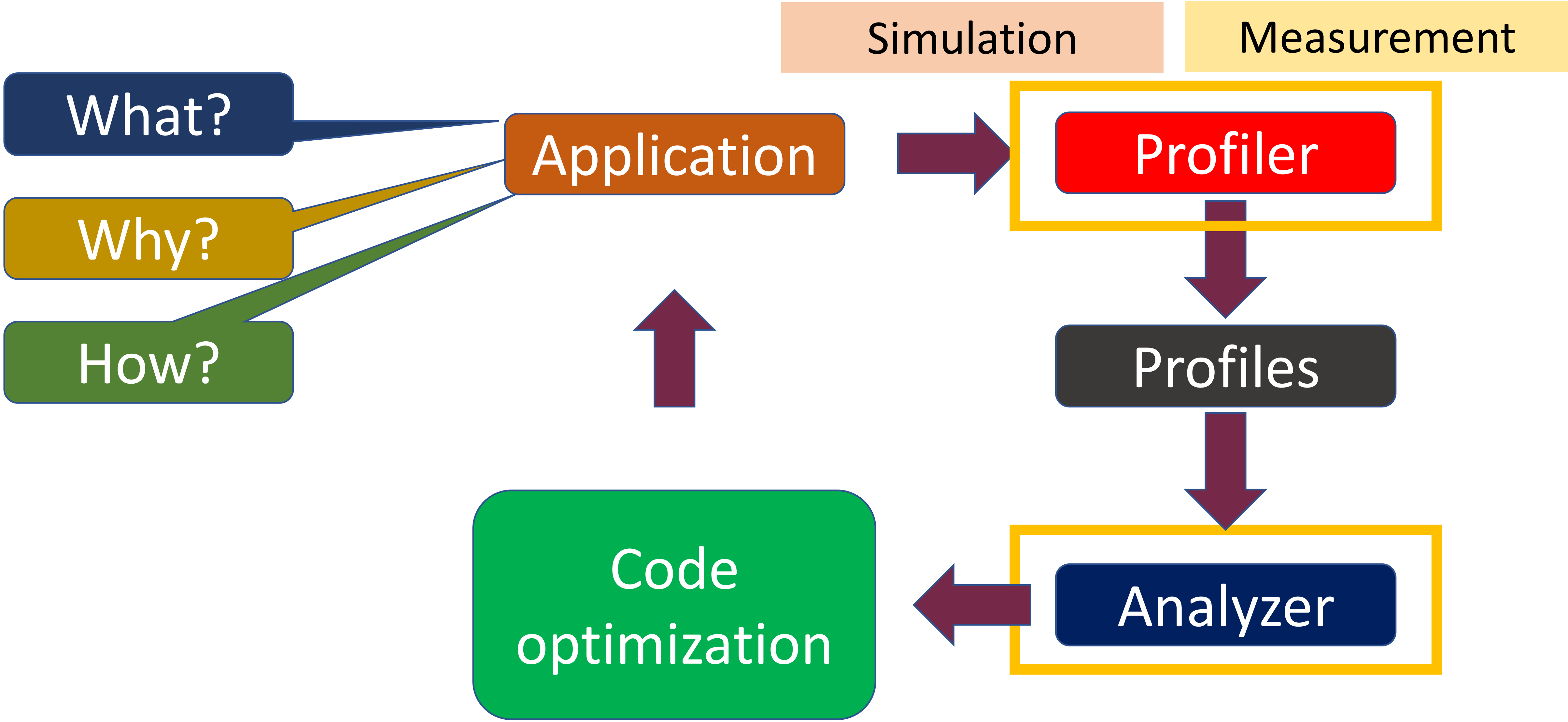
Profiler

Profiles

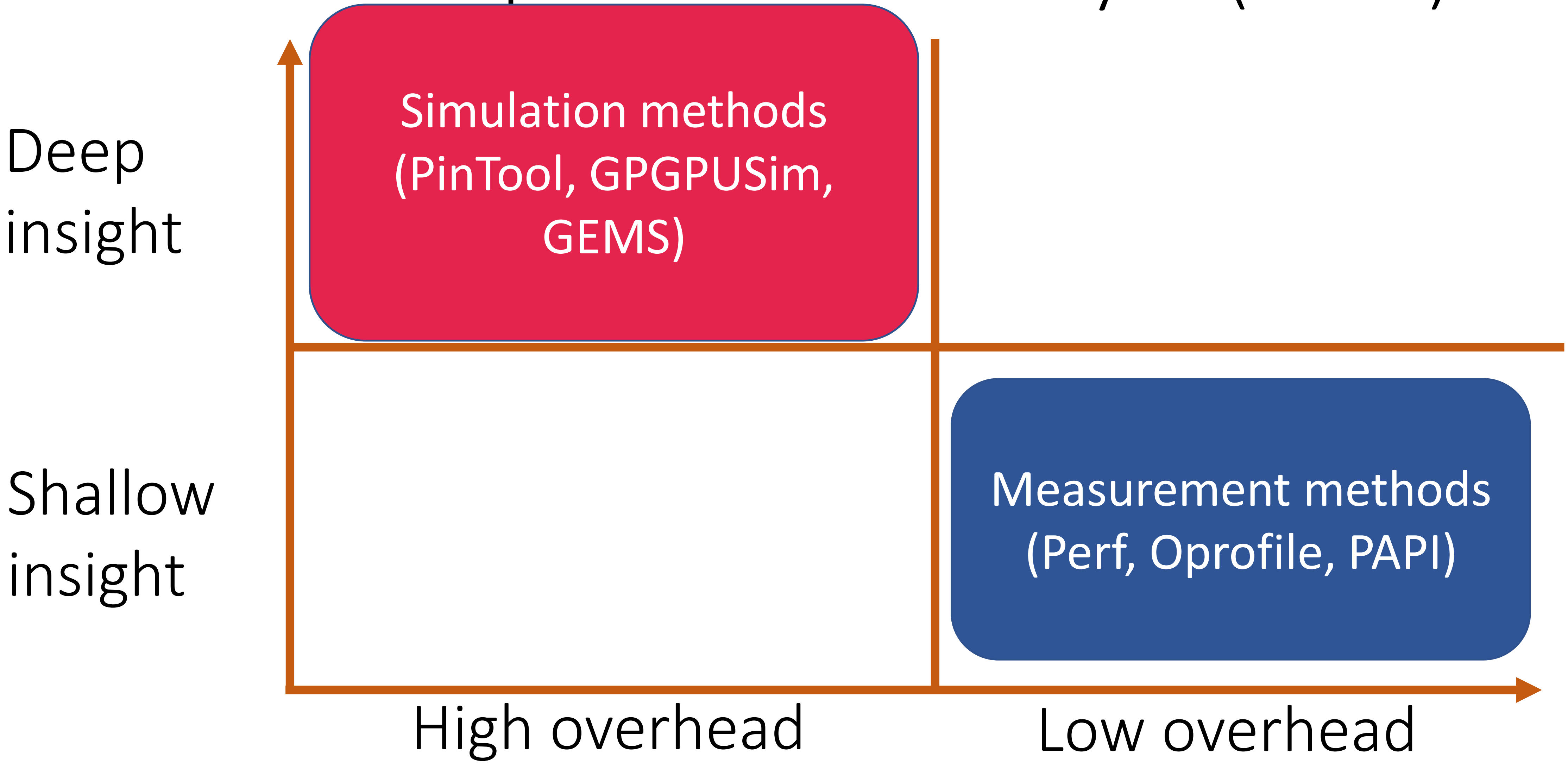
Analyzer

Code optimization

# Steps of performance analysis



# Limitations of performance analysis (Cont.)



Deep insight

Shallow insight

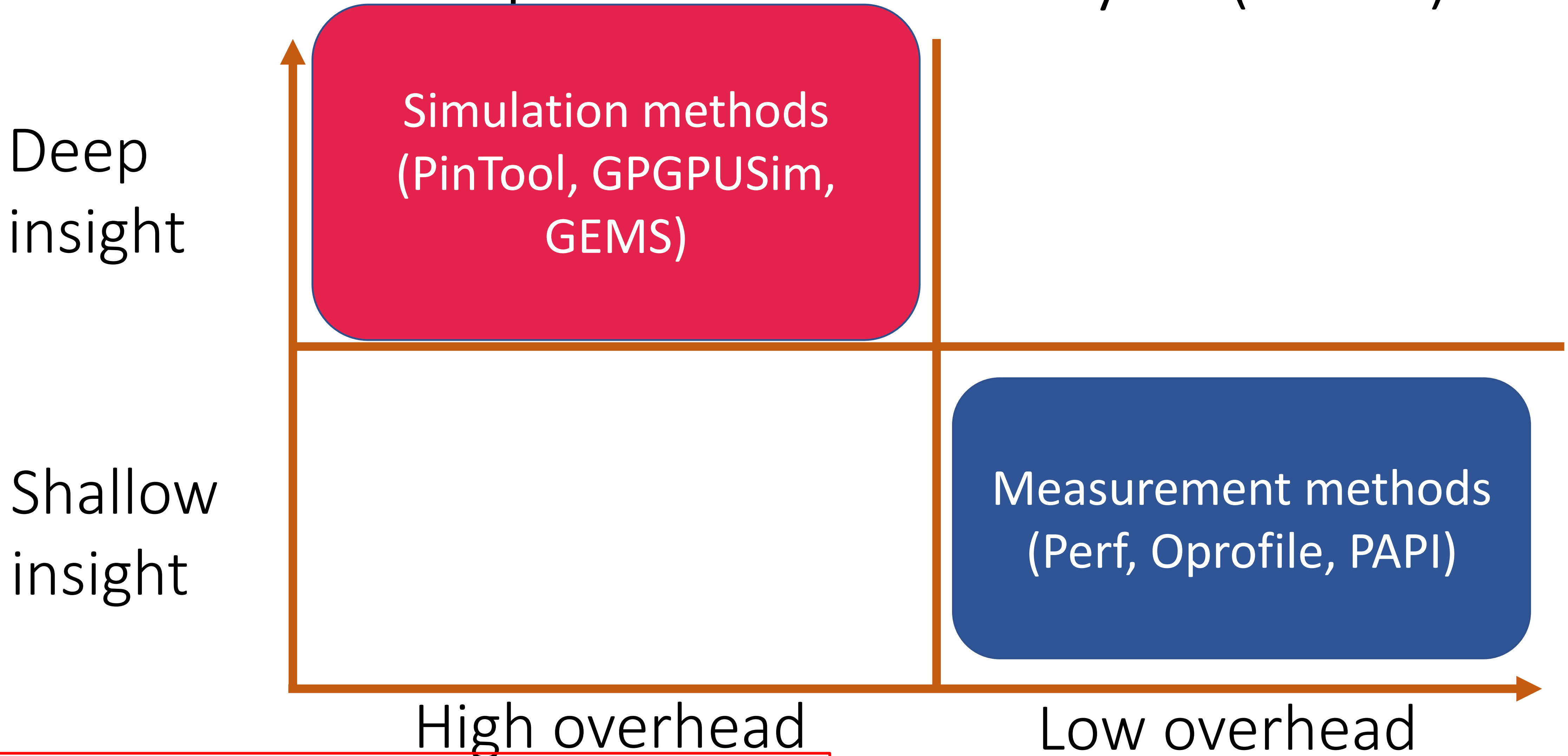
Simulation methods (PinTool, GPGPUSim, GEMS)

Measurement methods (Perf, Oprofile, PAPI)

High overhead

Low overhead

# Limitations of performance analysis (Cont.)



Deep insight

Shallow insight

Simulation methods (PinTool, GPGPUSim, GEMS)

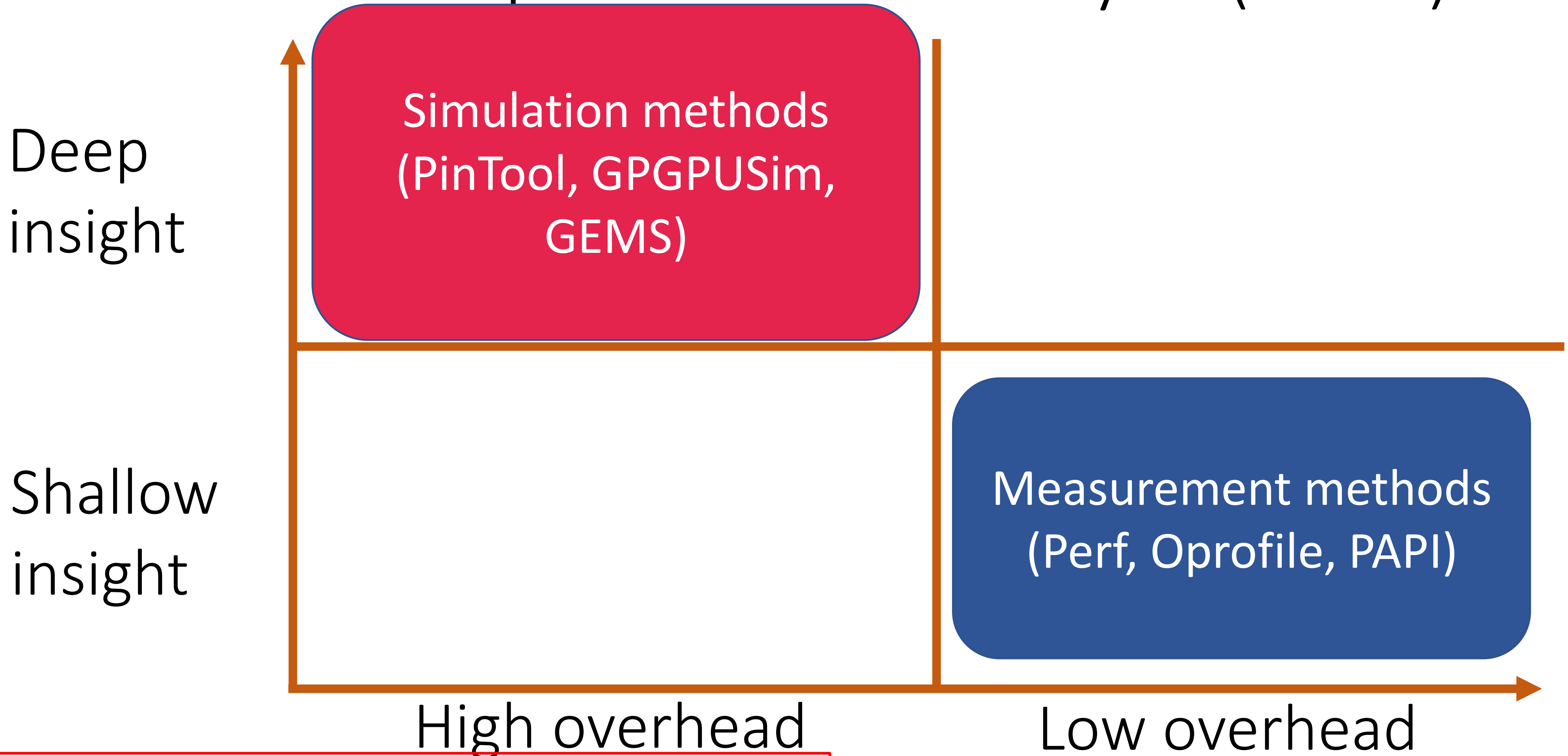
Measurement methods (Perf, Oprofile, PAPI)

High overhead

Low overhead

***Cache simulation:*** average 38x (*Xiang et al.* A higher order theory of locality )

# Limitations of performance analysis (Cont.)



Deep insight

Shallow insight

Simulation methods (PinTool, GPGPUSim, GEMS)

Measurement methods (Perf, Oprofile, PAPI)

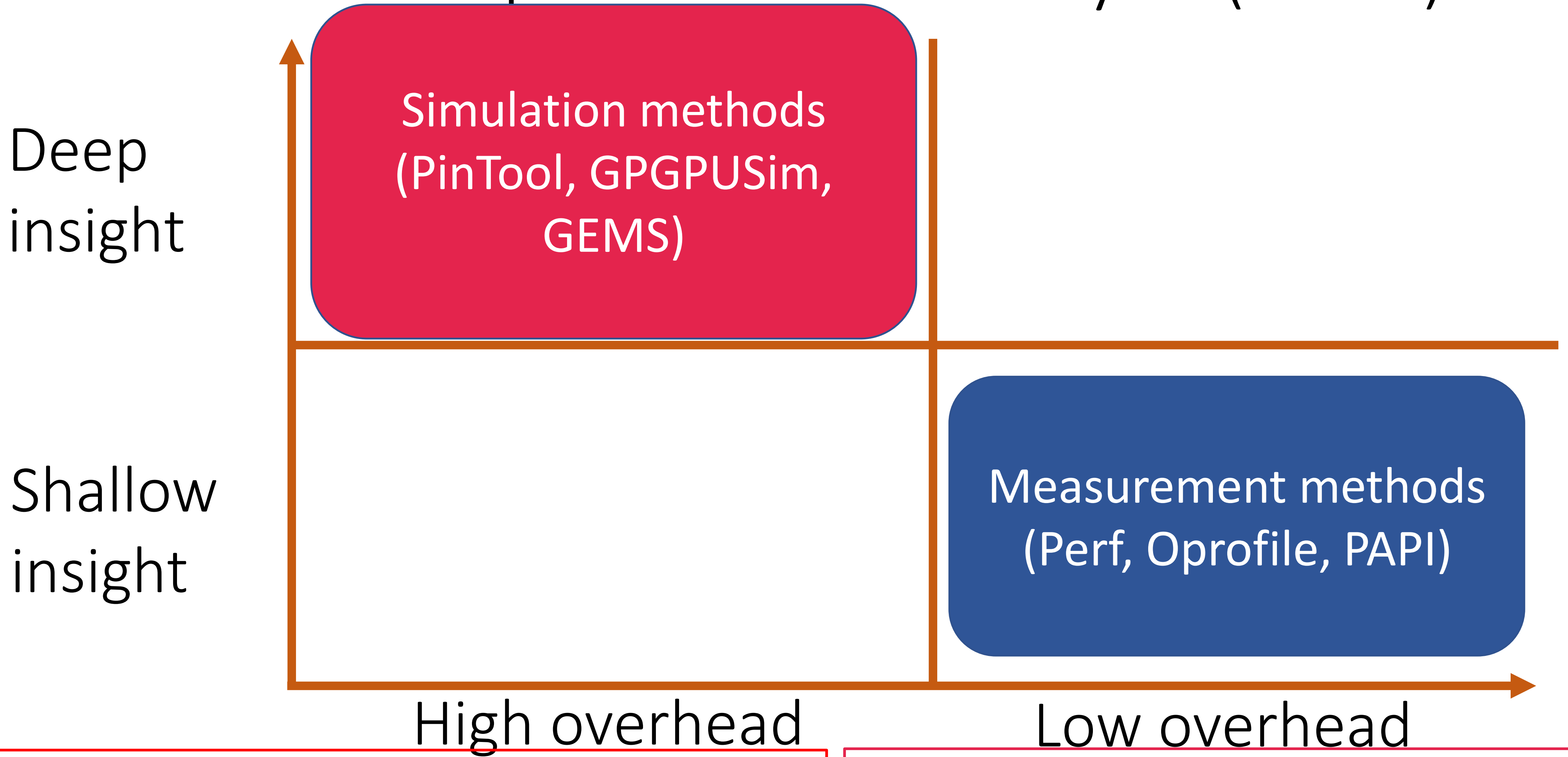
High overhead

Low overhead

***Cache simulation:*** average 38x (*Xiang et al.* A higher order theory of locality )

***Selective instrumentation***: 2x - 5x (Rane et al. MACPO)

# Limitations of performance analysis (Cont.)



Deep insight

Simulation methods (PinTool, GPGPUSim, GEMS)

Shallow insight

Measurement methods (Perf, Oprofile, PAPI)

High overhead

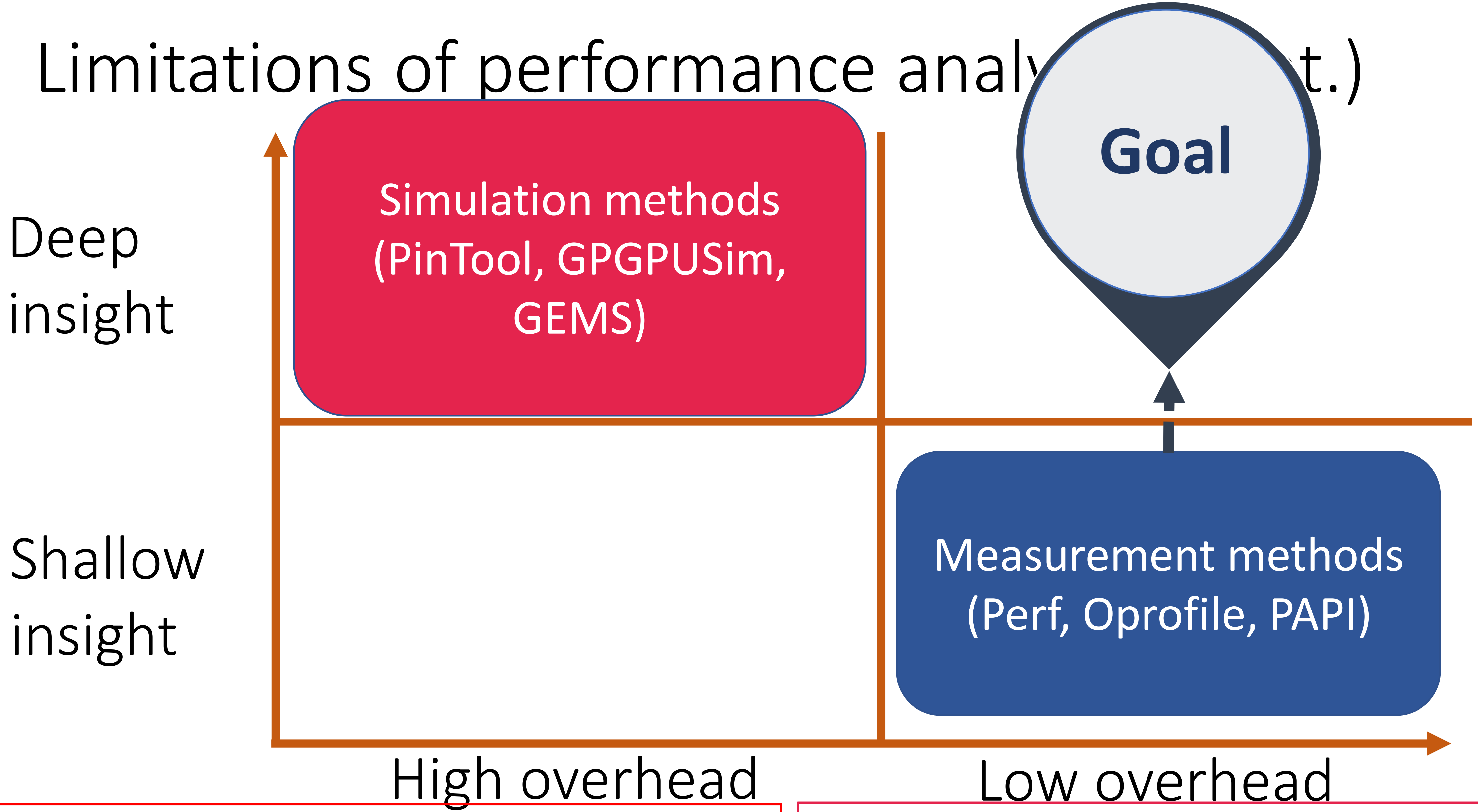
Low overhead

***Cache simulation:*** average 38x (*Xiang et al.* A higher order theory of locality )

***Profiling:*** < 10% (Liu et al. A Data-centric Profiler for Parallel Programs)

***Selective instrumentation***: 2x - 5x (Rane et al. MACPO)

# Limitations of performance analy t.)



Goal

Deep insight

Simulation methods (PinTool, GPGPUSim, GEMS)

Shallow insight

Measurement methods (Perf, Oprofile, PAPI)

High overhead

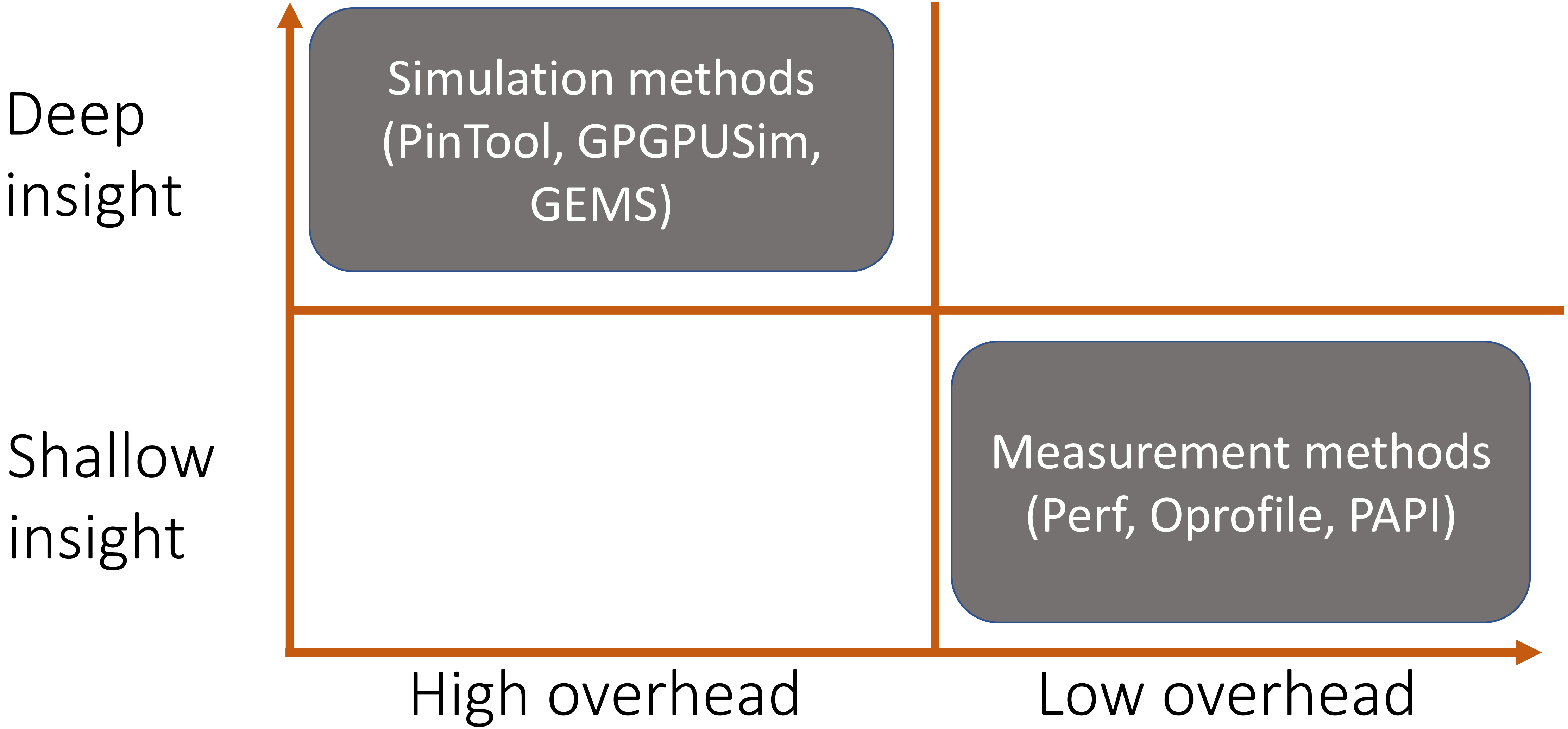
Low overhead

***Cache simulation:*** average 38x (*Xiang et al.* A higher order theory of locality )

***Profiling:*** < 10% (Liu et al. A Data-centric Profiler for Parallel Programs)

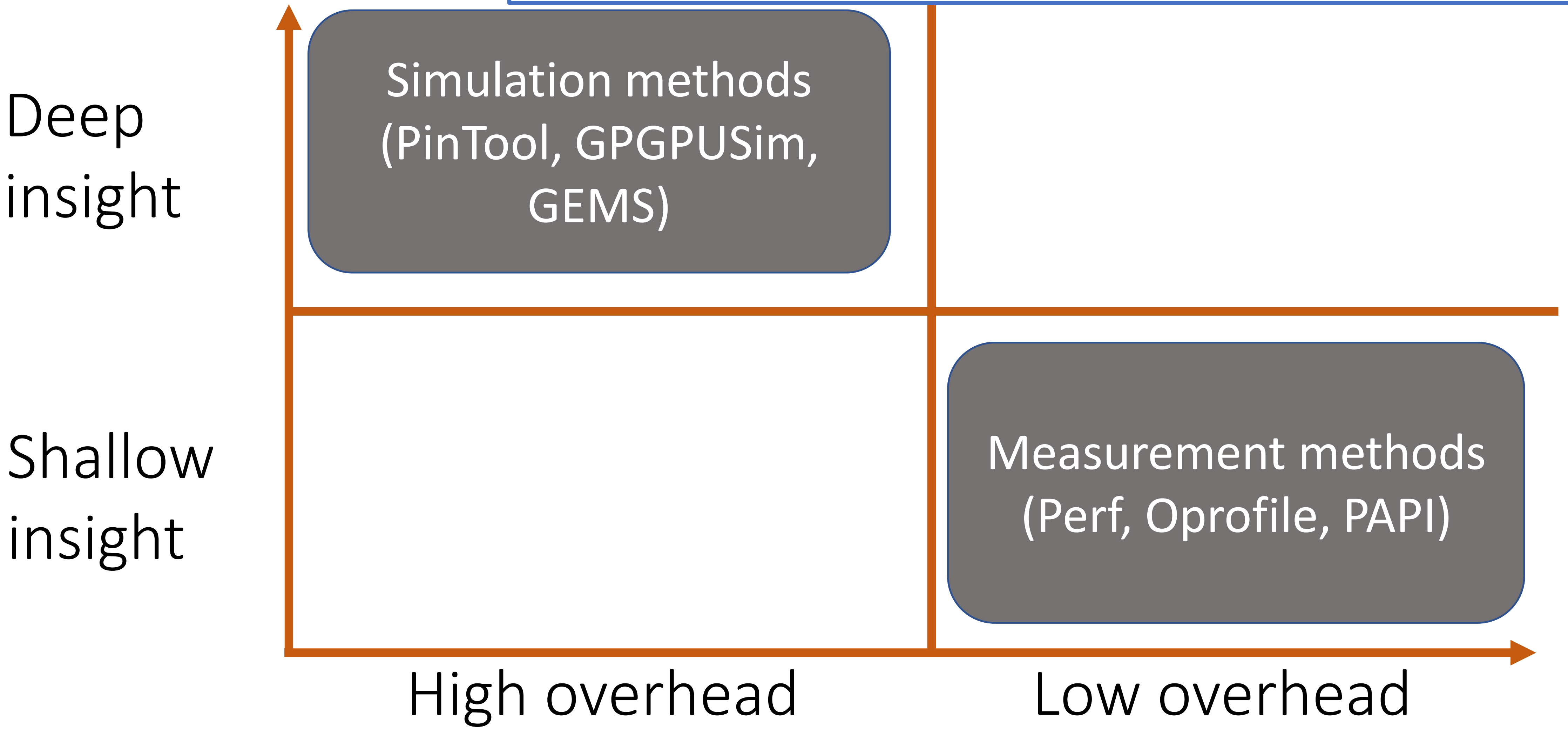
***Selective instrumentation***: 2x - 5x (Rane et al. MACPO)

# Research statement


Deep insight
Shallow insight
Simulation methods (PinTool, GPGPUSim, GEMS)
Measurement methods (Perf, Oprofile, PAPI)
High overhead
Low overhead

# Research statement

**Lightweight profiling with PMUs can provide deep insights into performance issues caused by memory hierarchies and poor algorithm choice**



Deep insight

Shallow insight

Simulation methods (PinTool, GPGPUSim, GEMS)

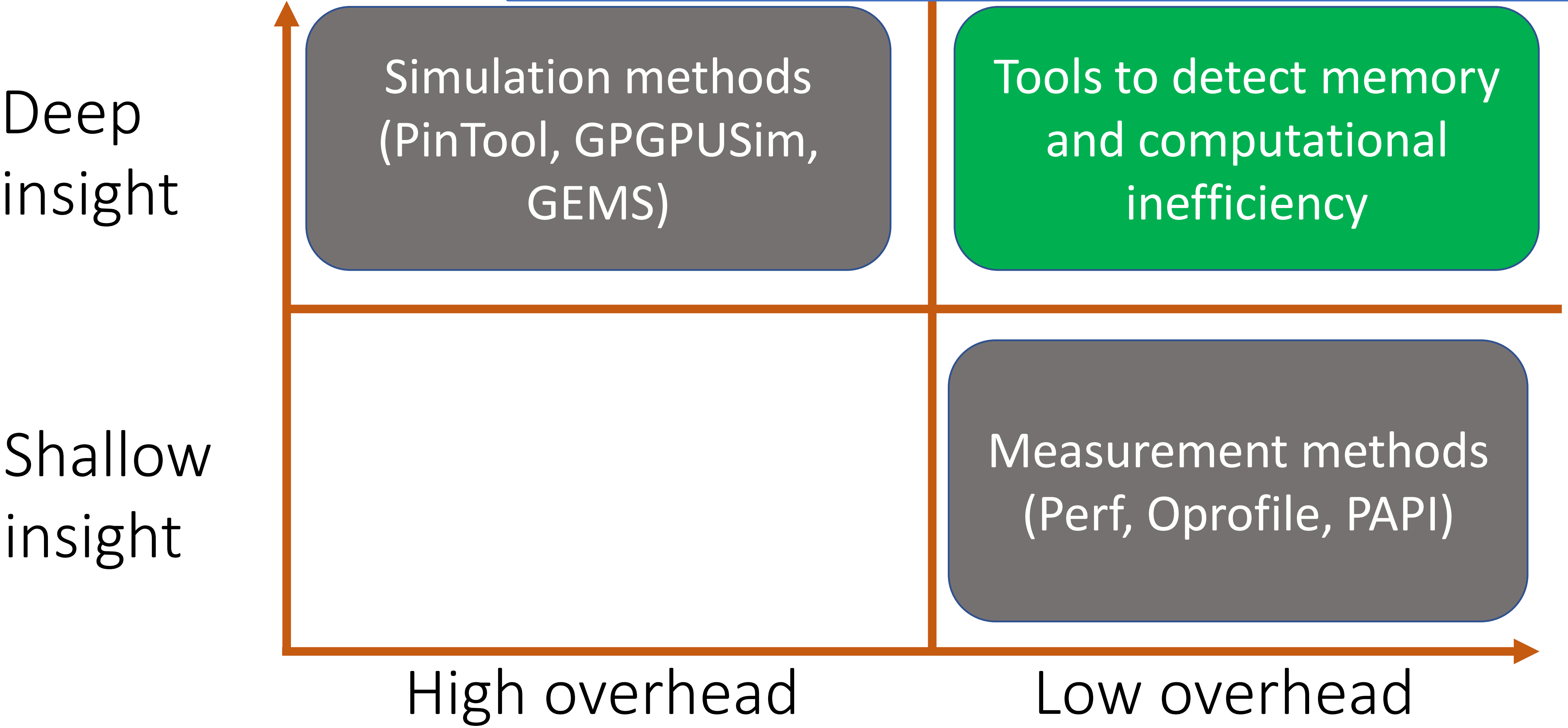
Measurement methods (Perf, Oprofile, PAPI)

High overhead

Low overhead

# Research statement

**Lightweight profiling with PMUs can provide deep insights into performance issues caused by memory hierarchies and poor algorithm choice**



Deep insight

Shallow insight

High overhead

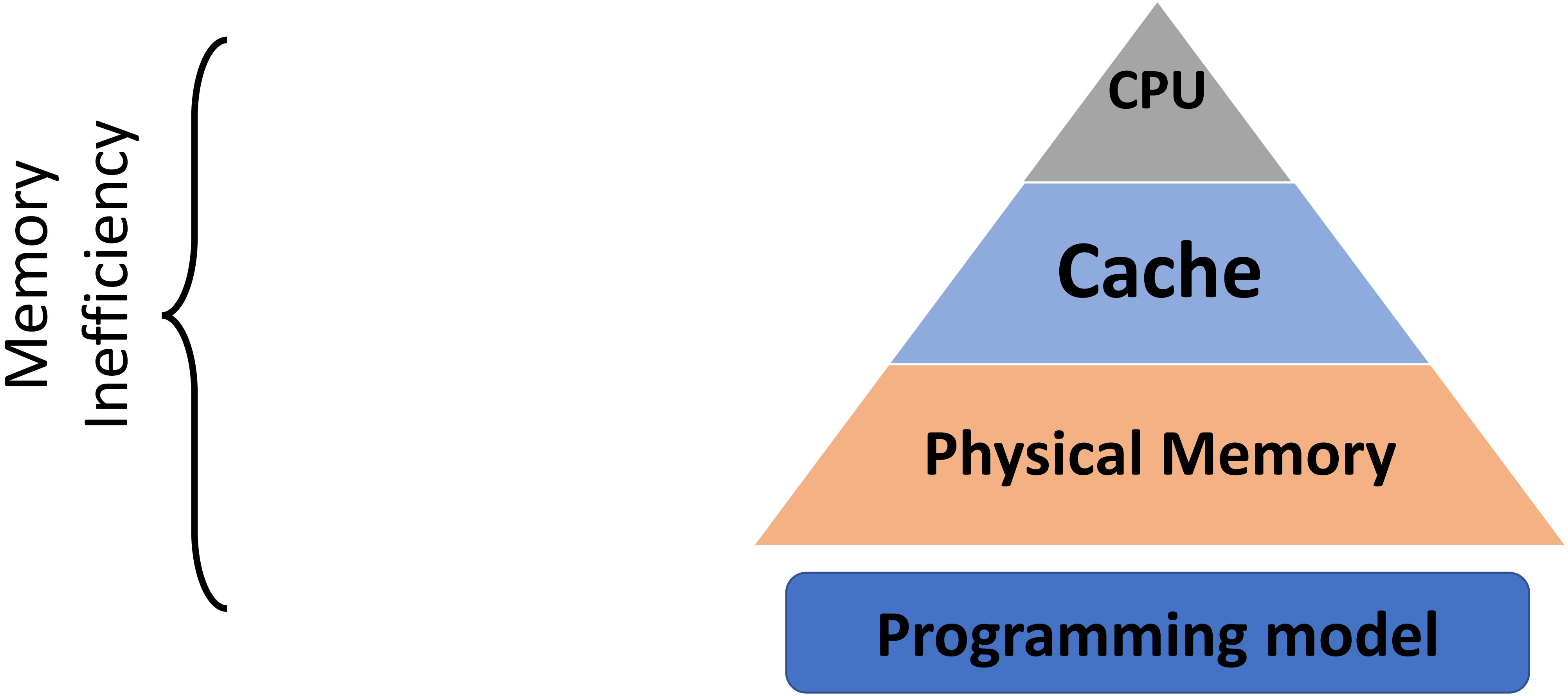
Low overhead

Simulation methods (PinTool, GPGPUSim, GEMS)

Tools to detect memory and computational inefficiency

Measurement methods (Perf, Oprofile, PAPI)

# My research at a glance



Memory Inefficiency

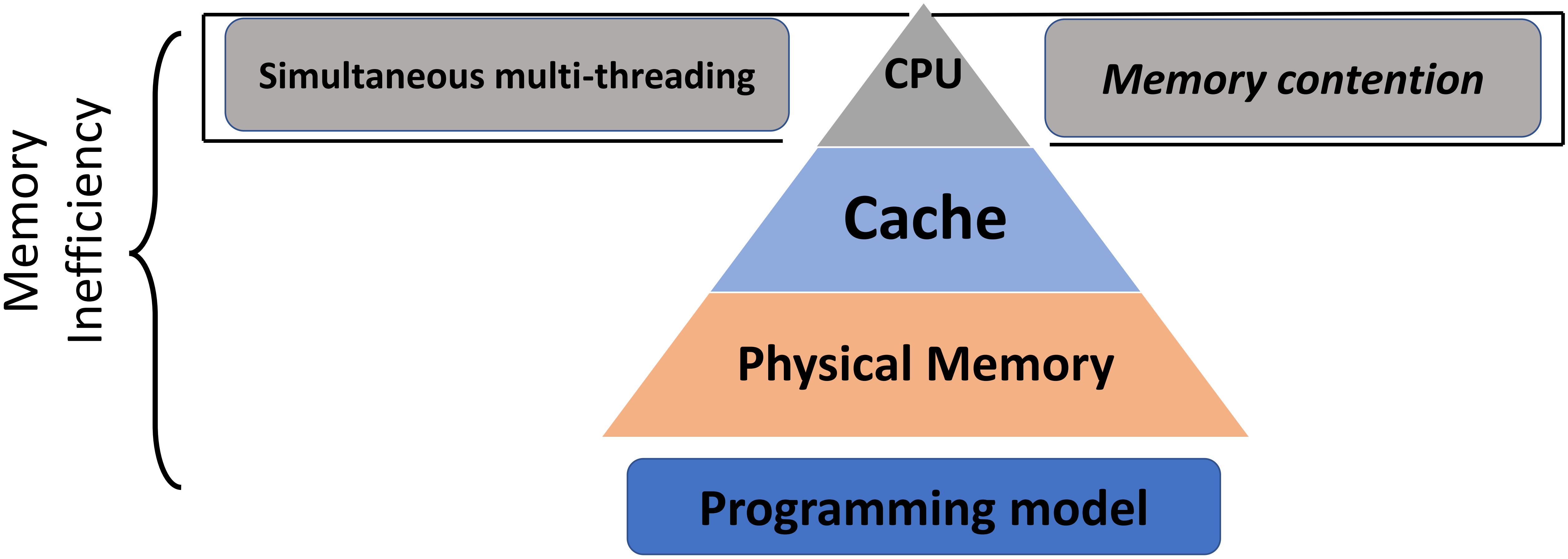
CPU

Cache

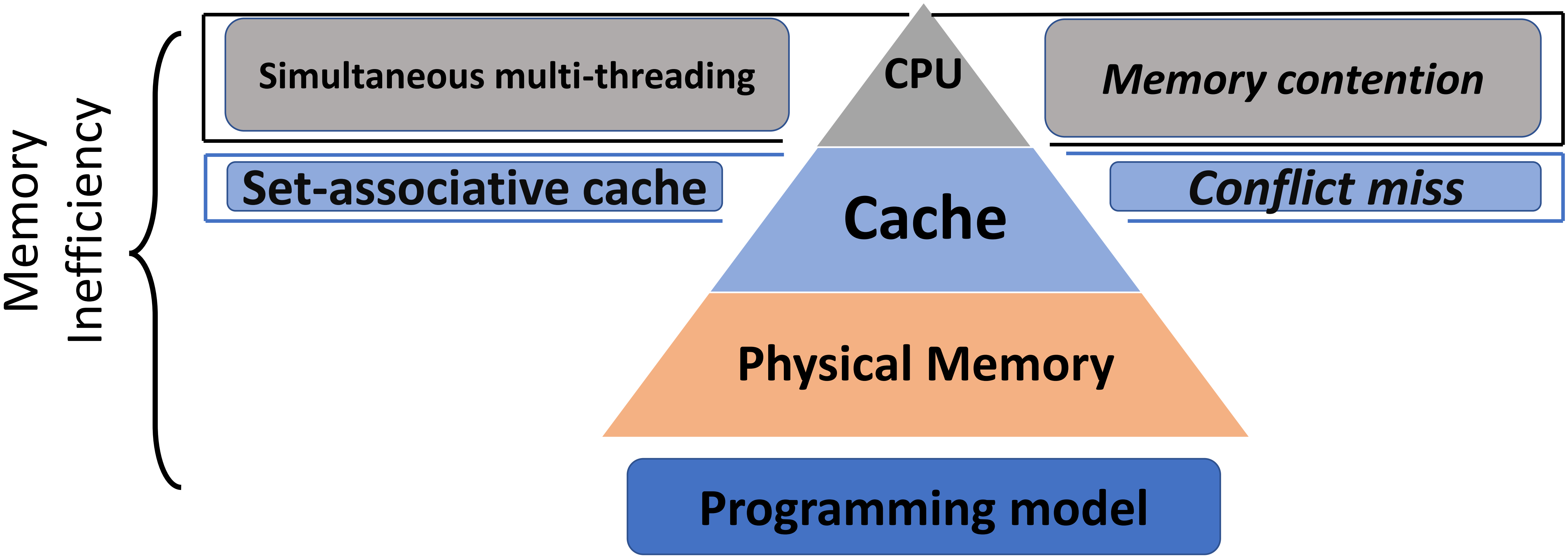
Physical Memory

Programming model

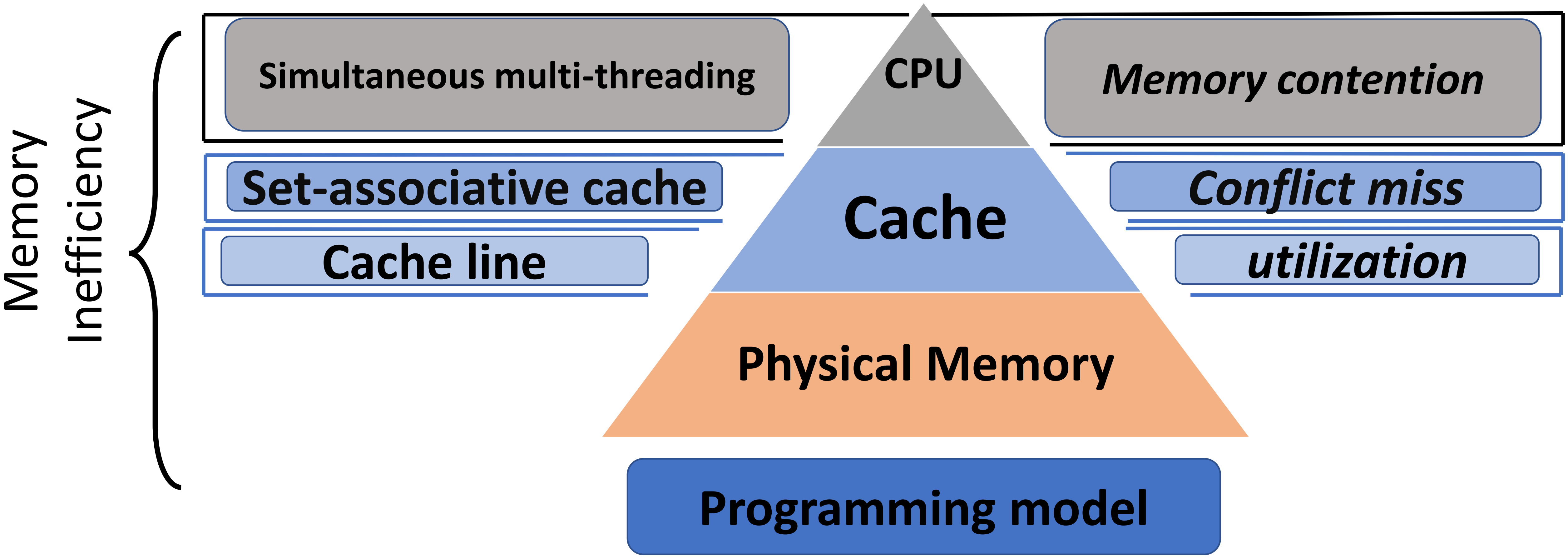
# My research at a glance



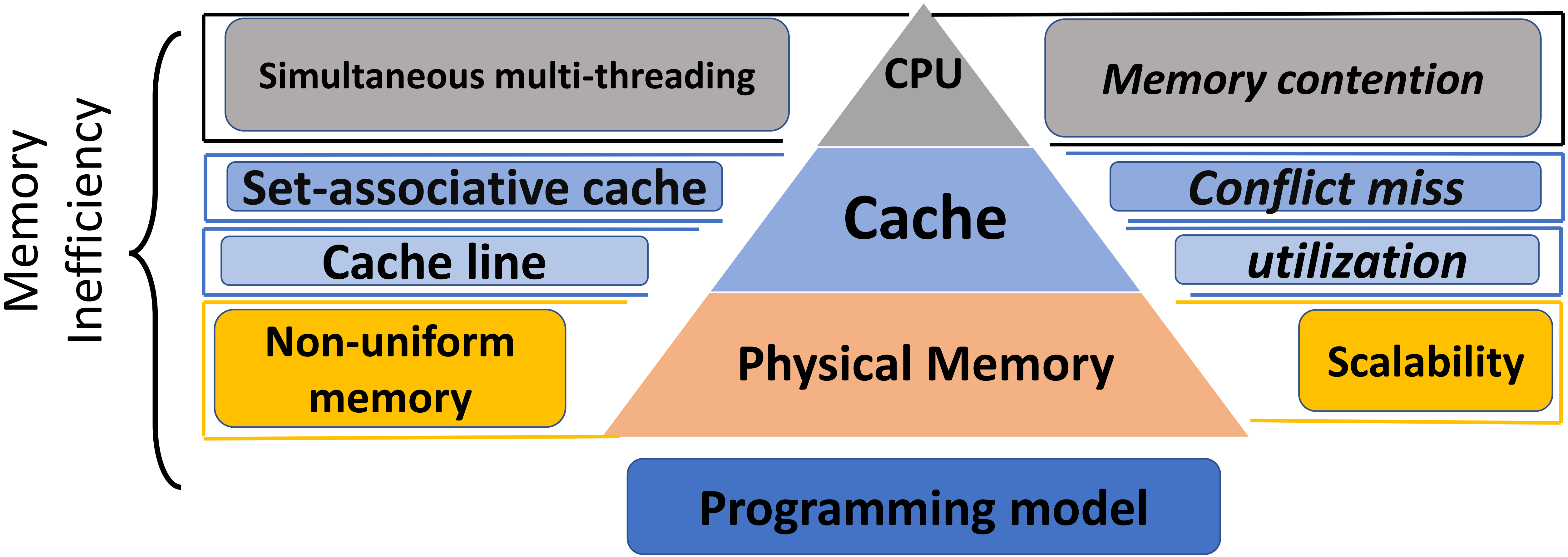
# My research at a glance



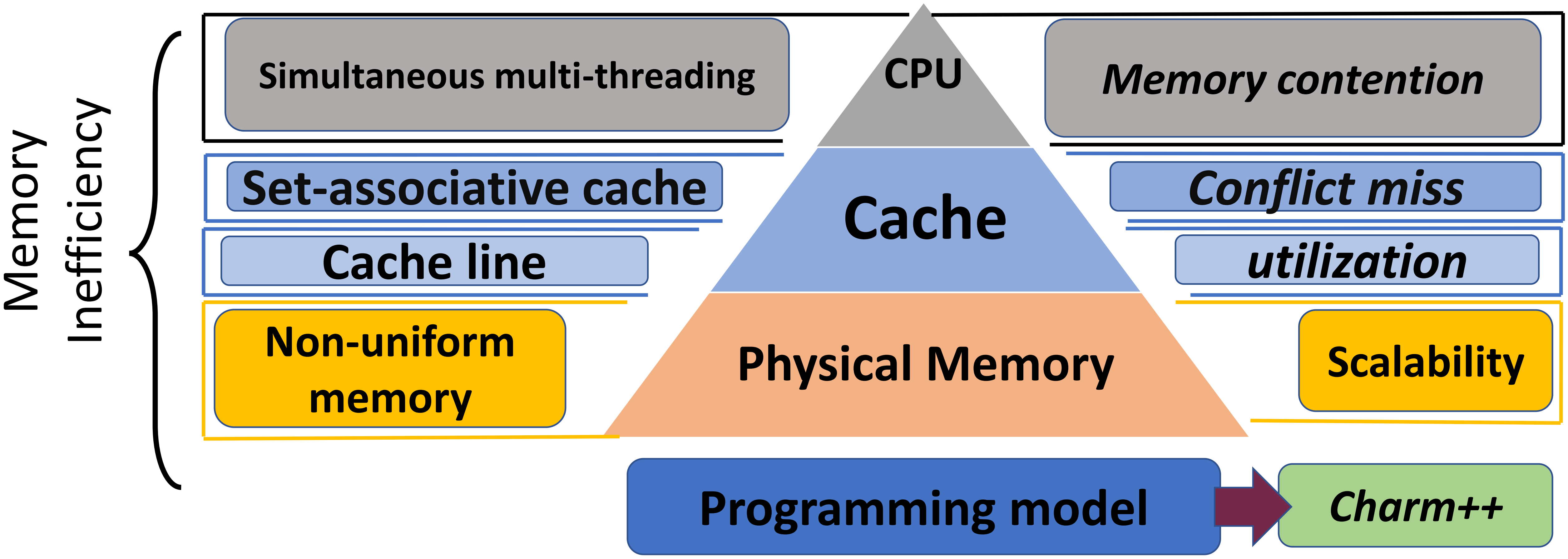
# My research at a glance



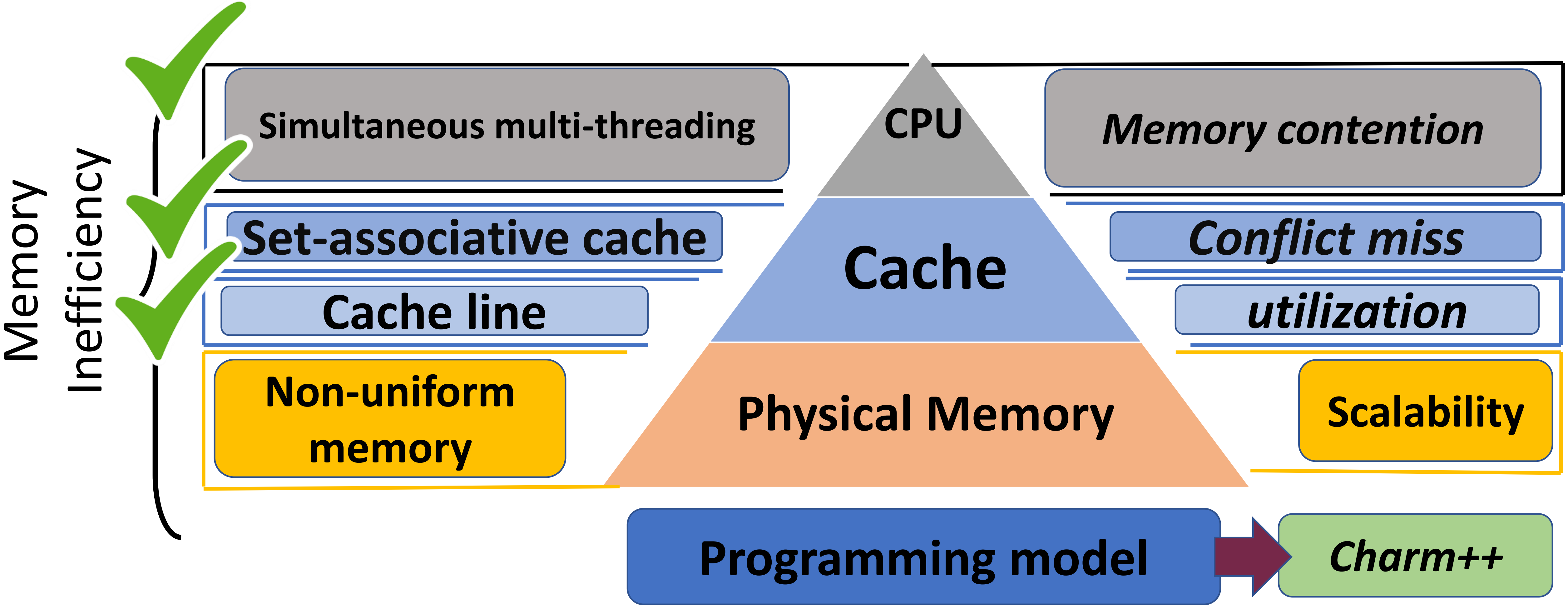
# My research at a glance



# My research at a glance



# My research at a glance

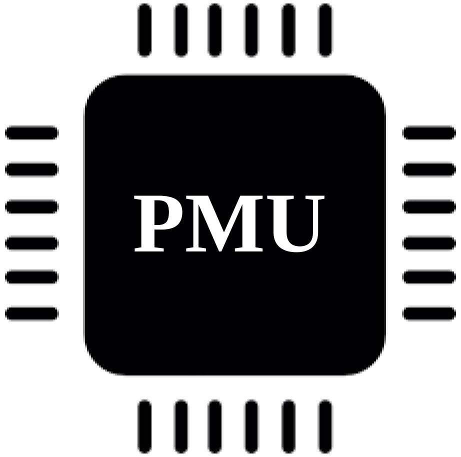


# Outline

- Lightweight profiling
- SMT-aware optimization
- Detection of cache conflicts
- Guiding data-structure layout transformation

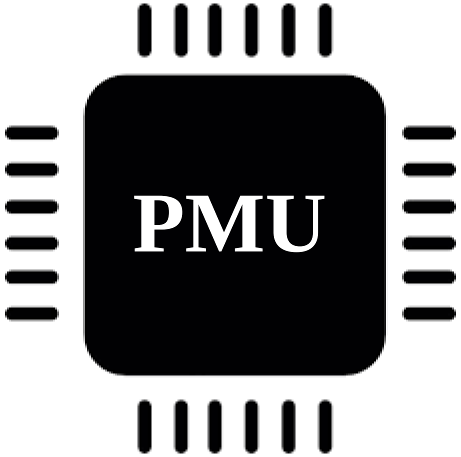
# Lightweight memory profiling

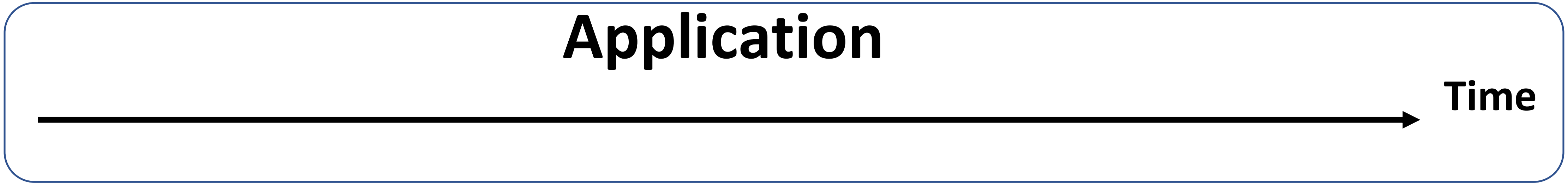
- Hardware profiling
- Event based sampling
  - Intel (Precise event based sampling - PEBS)
  - AMD (Instruction based sampling - IBS)
  - IBM (Marked event sampling - MRK)



# Lightweight memory profiling

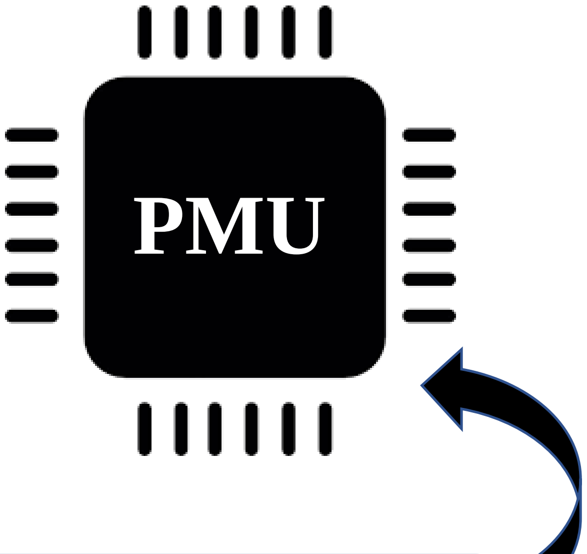
- Hardware profiling
- Event based sampling
    - Intel (Precise event based sampling - PEBS)
    - AMD (Instruction based sampling - IBS)
    - IBM (Marked event sampling - MRK)





# Lightweight memory profiling

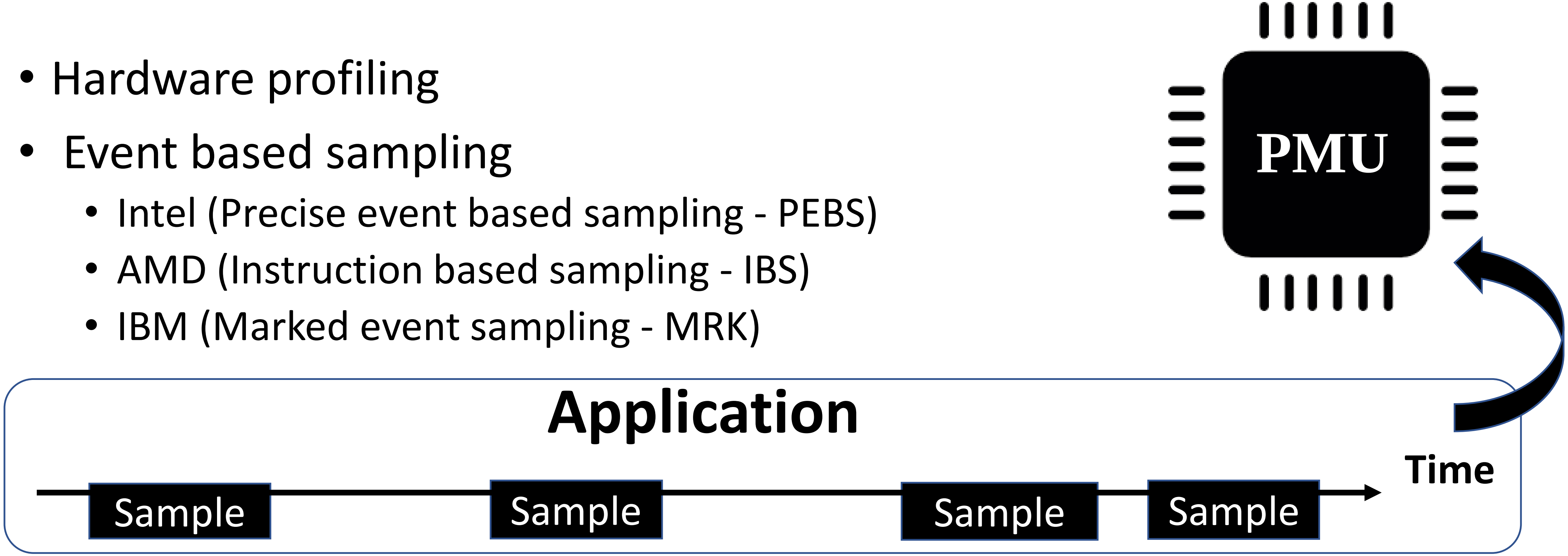
- Hardware profiling
- Event based sampling
  - Intel (Precise event based sampling - PEBS)
  - AMD (Instruction based sampling - IBS)
  - IBM (Marked event sampling - MRK)



Application
Time

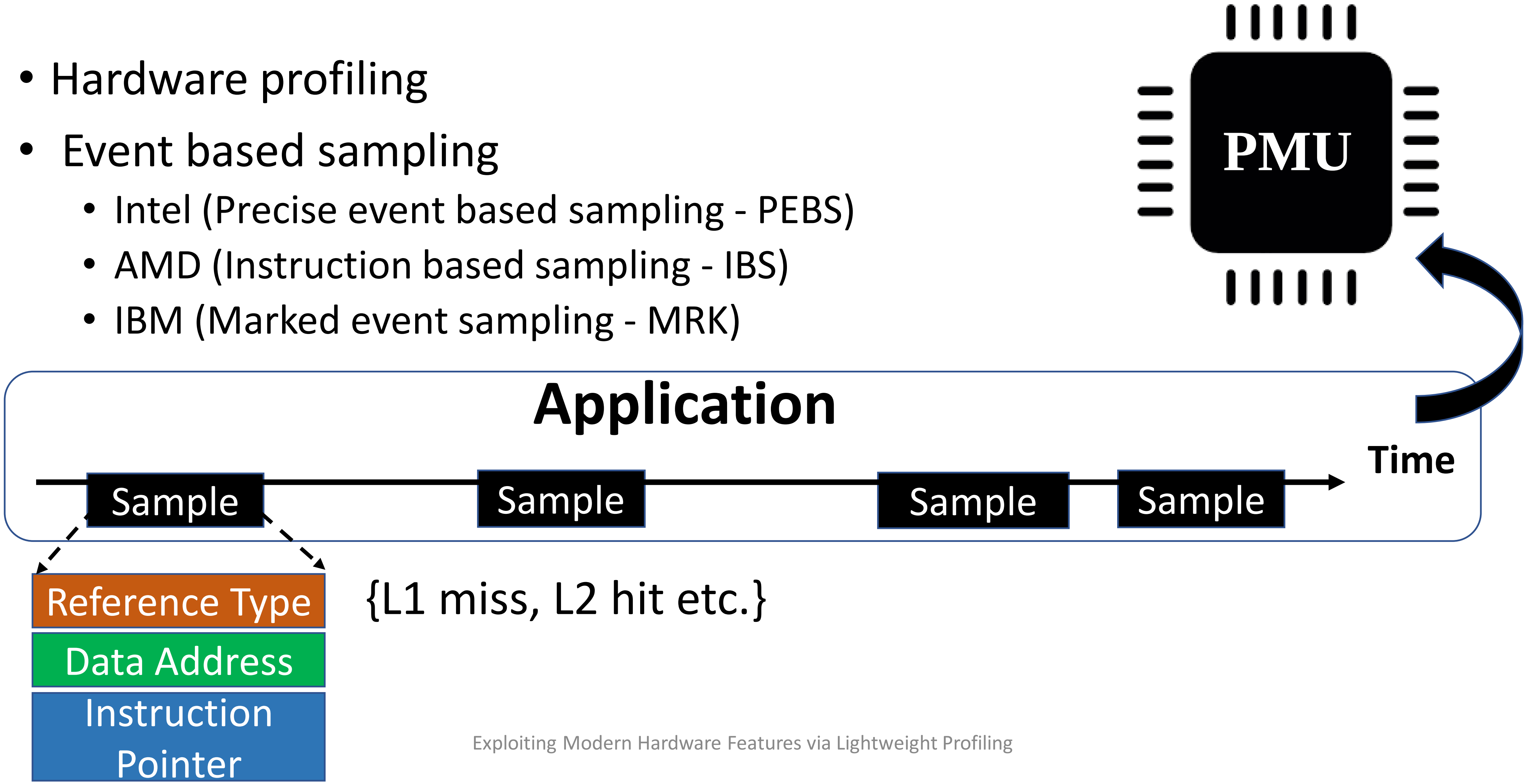
# Lightweight memory profiling

- Hardware profiling
- Event based sampling
    - Intel (Precise event based sampling - PEBS)
    - AMD (Instruction based sampling - IBS)
    - IBM (Marked event sampling - MRK)



# Lightweight memory profiling

- Hardware profiling
- Event based sampling
  - Intel (Precise event based sampling - PEBS)
  - AMD (Instruction based sampling - IBS)
  - IBM (Marked event sampling - MRK)



# Outline

- ✓Lightweight profiling
- ✓*SMT-aware optimization*
- Detection of cache conflicts
- Guiding data-structure layout transformation

# SMT-Aware Instantaneous Footprint Optimization

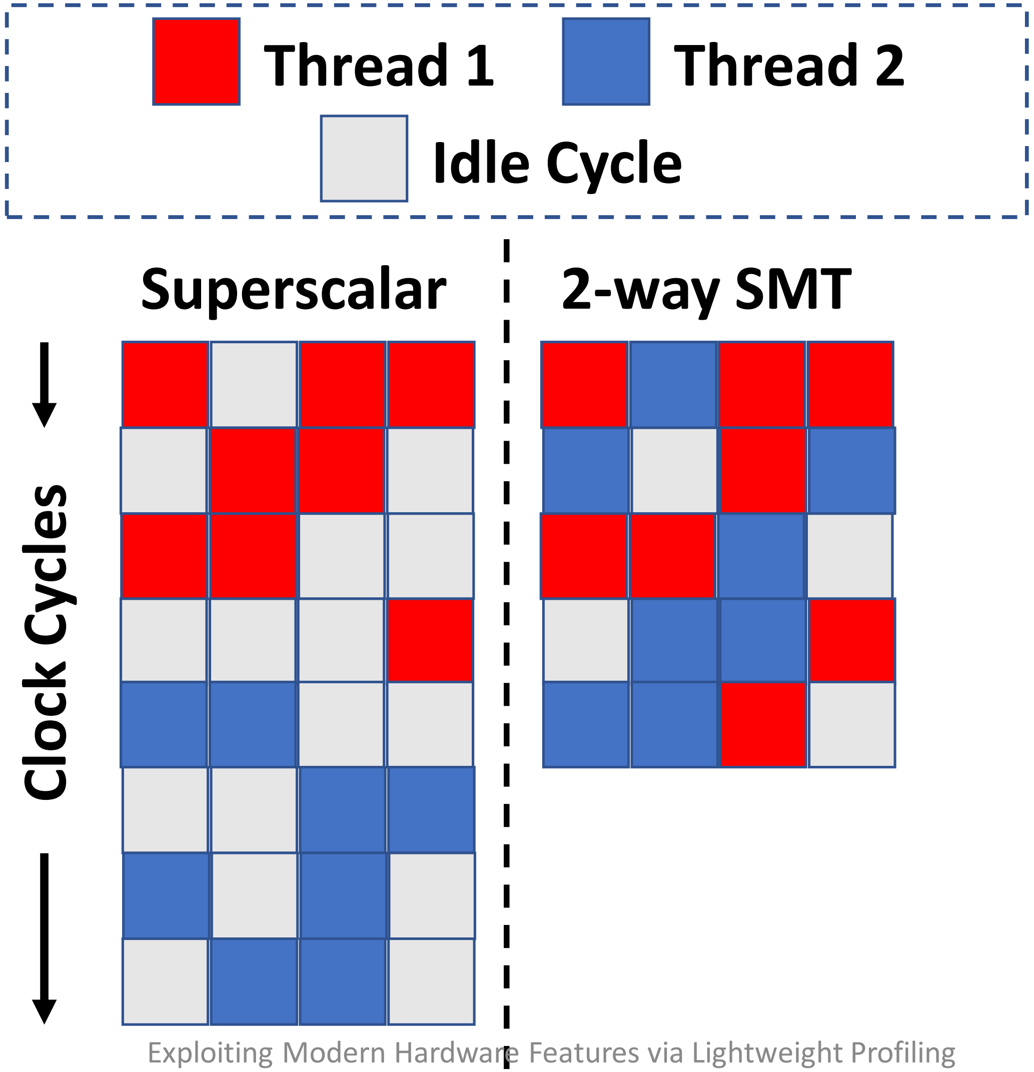
[HPDC – 2016]

**Probir Roy**, Shuaiwen Leon Song, *Xu Liu*

WILLIAM & MARY
CHARTERED 1693

Pacific Northwest
NATIONAL LABORATORY

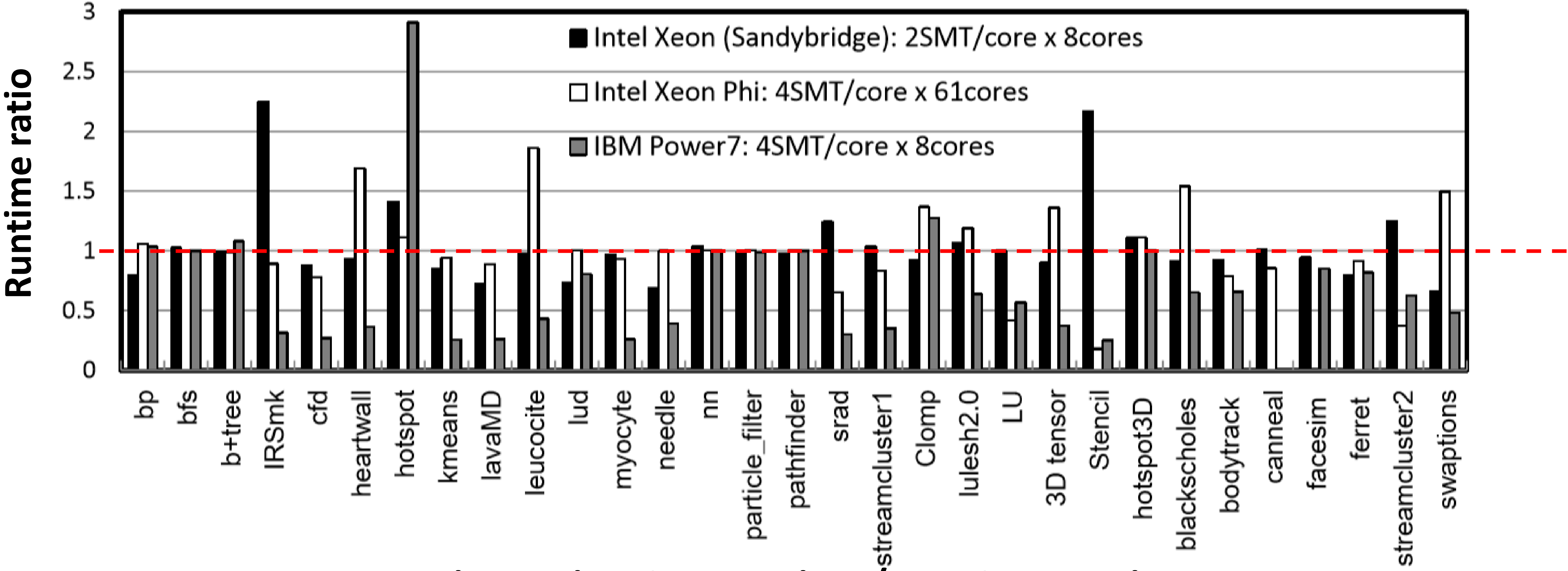
# SMT (Simultaneous Multi-Threading)



# SMT scalability

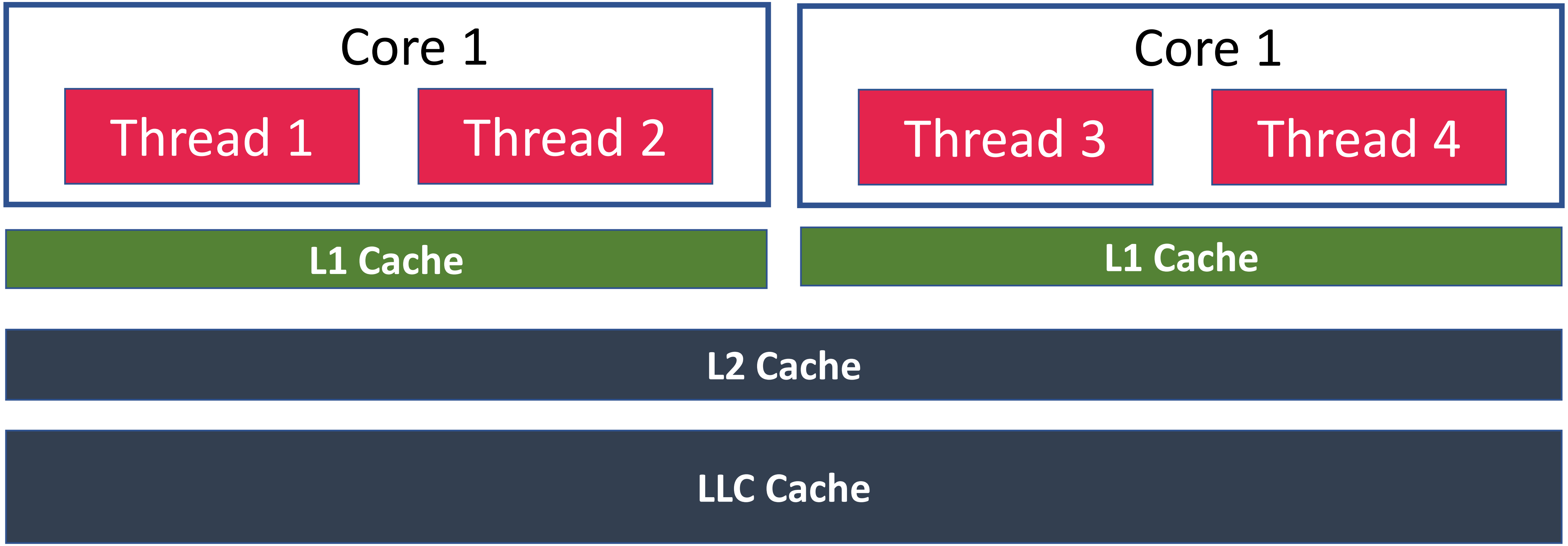
Shared memory SPMD application

Lower is better



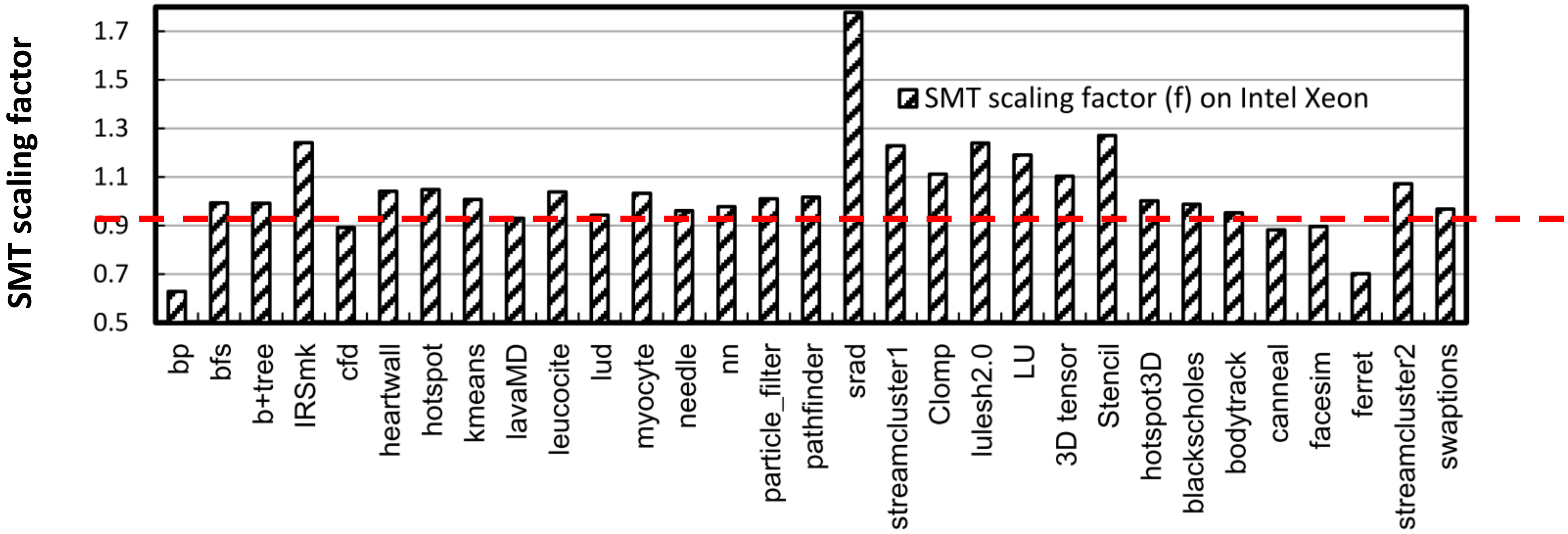
Runtime ratio = SMT runtime / non-SMT runtime

# SMT architecture: shared cache



# SMT: Memory scalability

Lower is better



**SMT scaling factor (F) = access Latency of SMT/ access Latency of non-SMT**

# Characterization based on sensitivity

**L = Memory Access Latency; F = scaling factor**

| (L,F) | Benchmarks | Characterization |
|---|---|---|
| (high, high) | srad, streamcluster1, Lulesh2.0, IRSmk, LU, 3D tensor, Stencil, streamcluster2, hotspot, Clomp | potentially sensitive to mem-centric SMT optimizations |
| (high, low) | lud, needle, bfs, nn, bp, canneal, Ferret | not clear if they can further benefit from SMT optimizations |
| (low, high) | leucocite, heartwall, pathfinder, myocyte | little benefit from mem-centric SMT optimization |
| (low, low) | b+tree, cfd, kmeans, lavaMD, particle filter, hotspot3D, blackscholes, bodytrack, facesim, Swaptions | good memory performance with SMT enabled |

# Characterization based on sensitivity

L = Memory Access Latency; F = scaling factor

| (L,F) | Benchmarks | Characterization |
|---|---|---|
| (high, high) | srad, streamcluster1, Lulesh2.0, IRSmk, LU, 3D tensor, Stencil, streamcluster2, hotspot, Clomp | potentially sensitive to mem-centric SMT optimizations |
| (high, low) | lud, needle, bfs, nn, bp, canneal, Ferret | not clear if they can further benefit from SMT optimizations |
| (low, high) | leucocite, heartwall, pathfinder, myocyte | little benefit from mem-centric SMT optimization |
| (low, low) | b+tree, cfd, kmeans, lavaMD, particle filter, hotspot3D, blackscholes, bodytrack, facesim, Swaptions | good memory performance with SMT enabled |

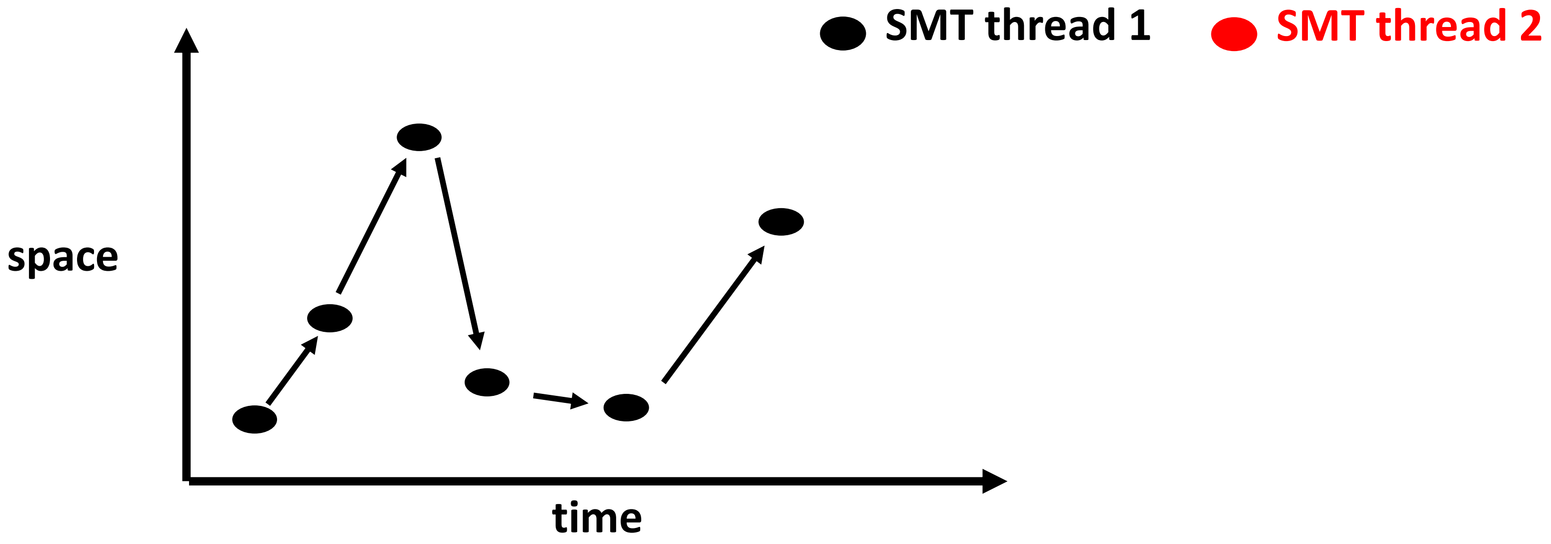
# Source of memory contention

Little/no locality

# Source of memory contention

Little/no locality

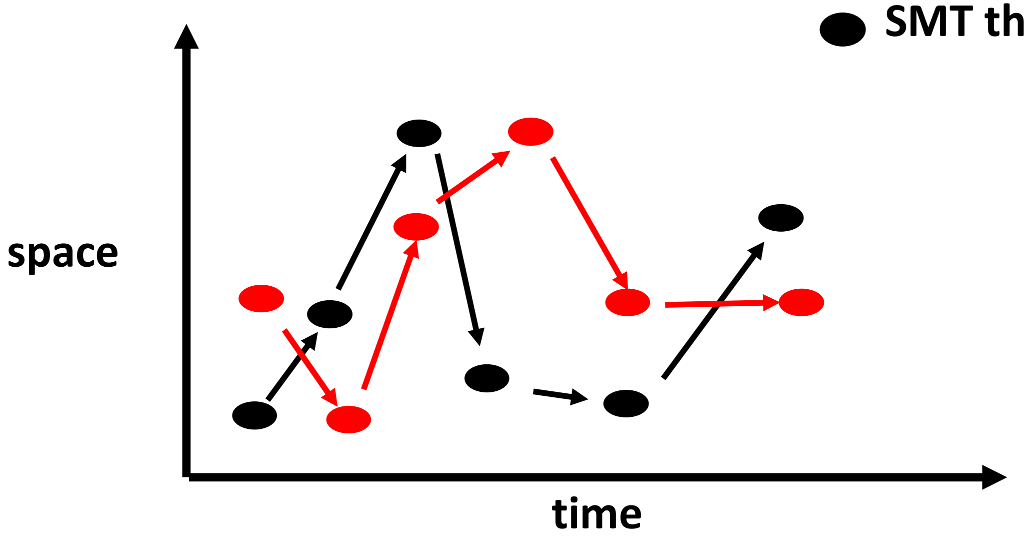
Intra-thread



# Source of memory contention

Little/no locality

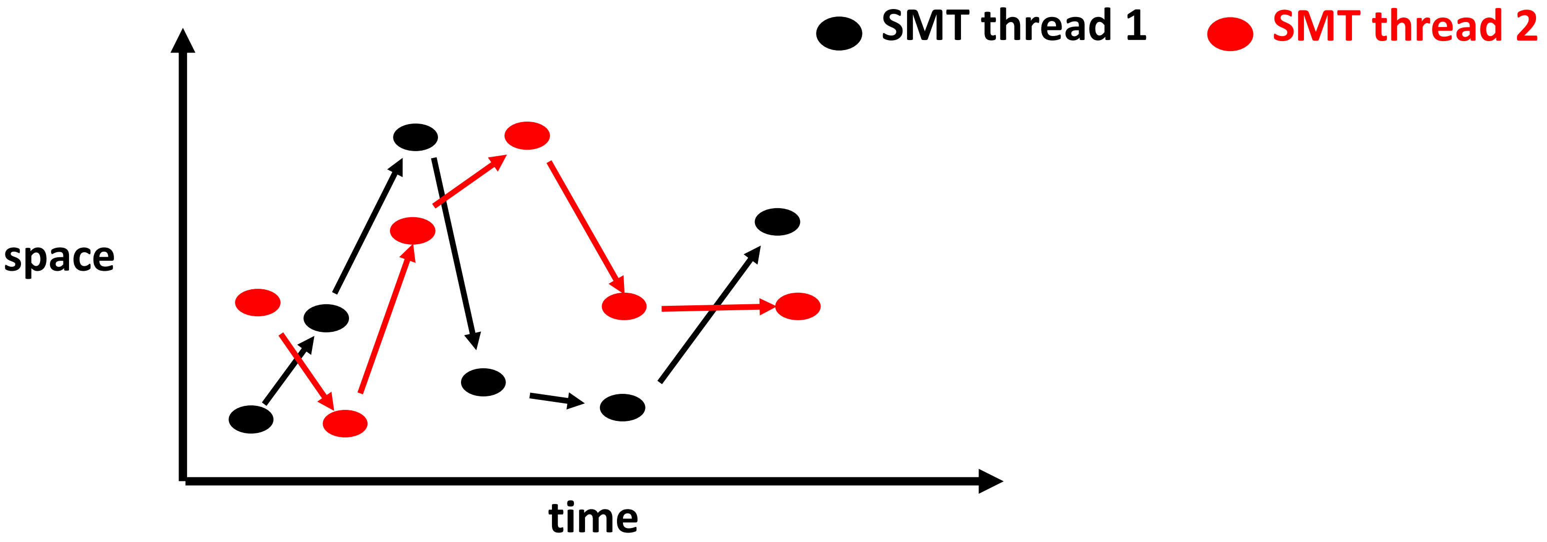
Intra-thread



# Source of memory contention

Little/no locality

Intra-thread



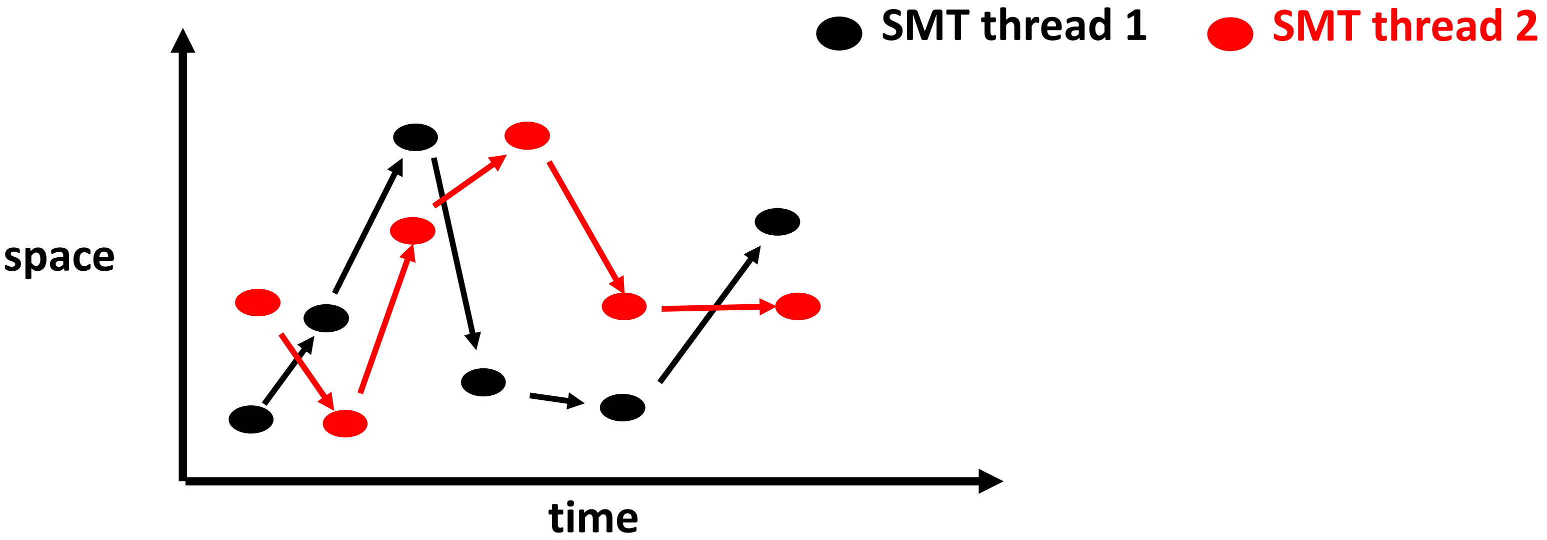
Optimization: Improve cache line utilization

# Source of memory contention

Little/no locality

Intra-thread

Inter-thread



Optimization: Improve cache line utilization

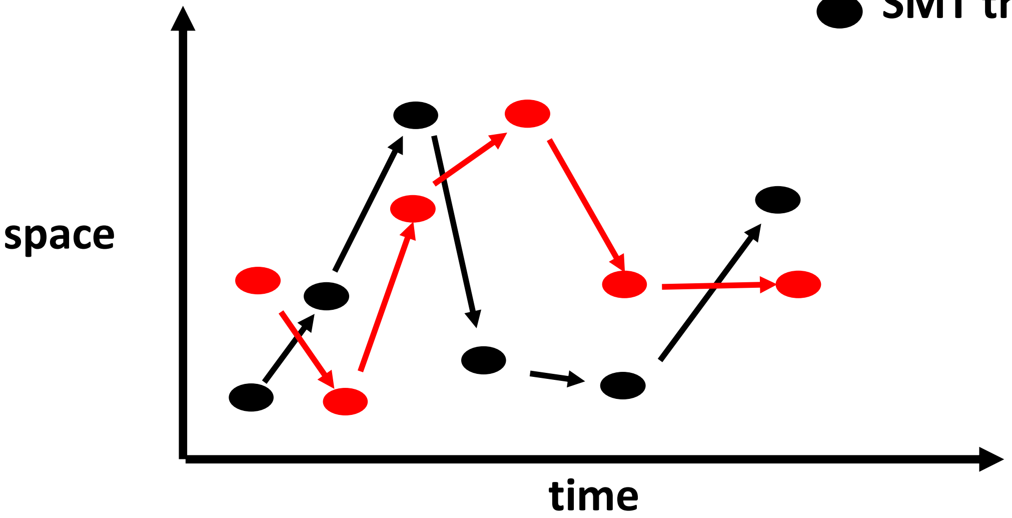
# Source of memory contention

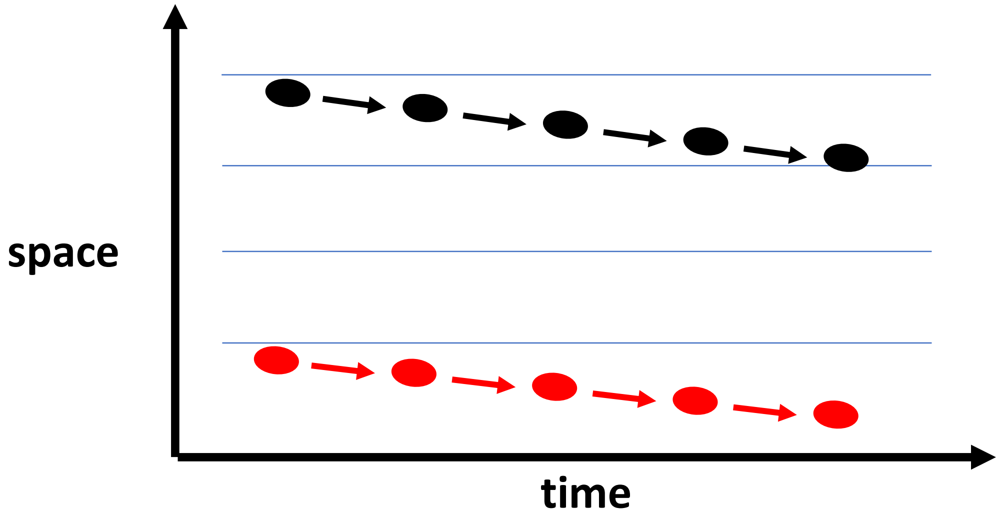
Little/no locality

Intra-thread

Inter-thread

● SMT thread 1 ● SMT thread 2





Optimization: Improve cache line utilization

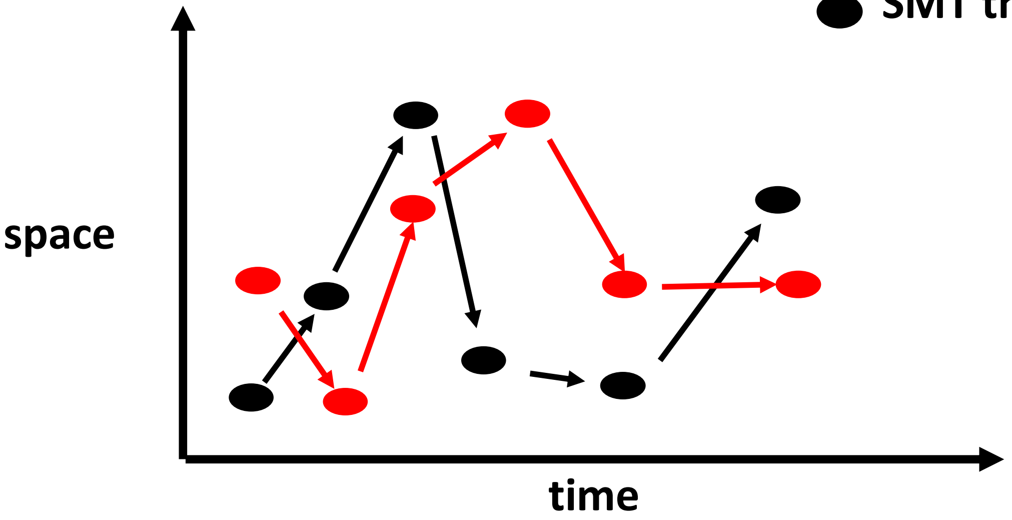
# Source of memory contention

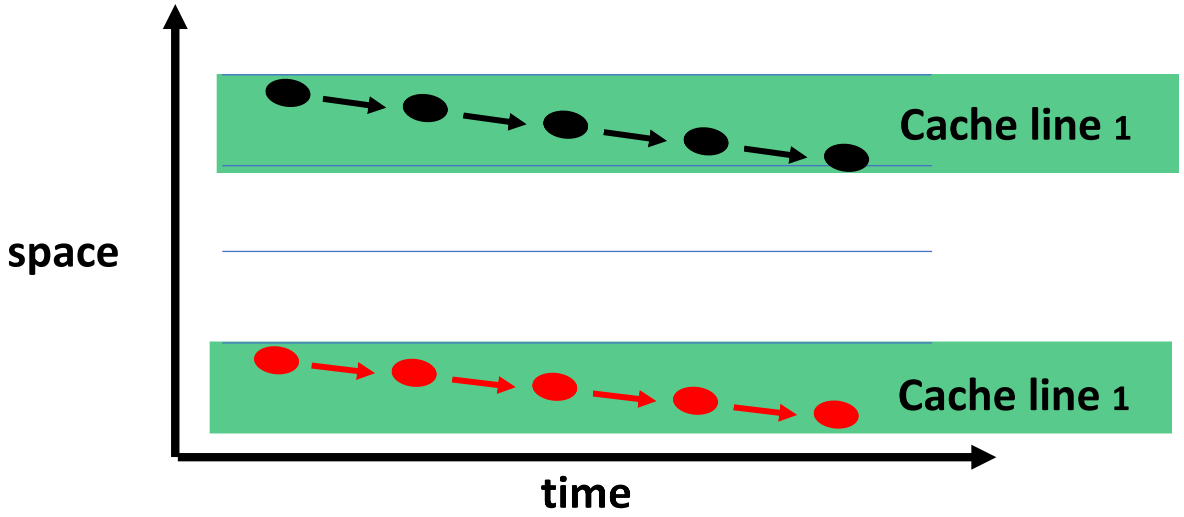
Little/no locality

Intra-thread

Inter-thread

● SMT thread 1 ● SMT thread 2





Optimization: Improve cache line utilization

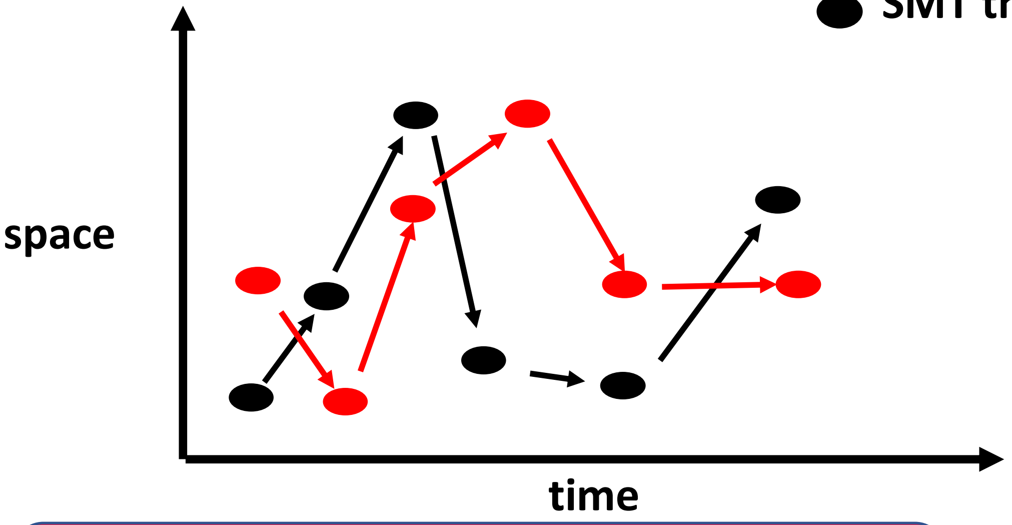
# Source of memory contention

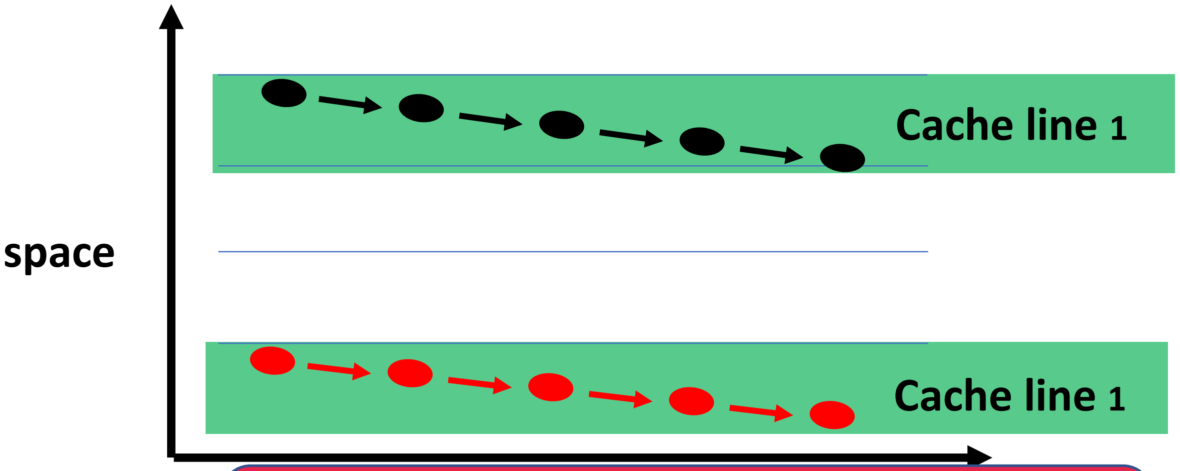
Little/no locality

Intra-thread

Inter-thread

● SMT thread 1 ● SMT thread 2





Optimization: Improve cache line utilization

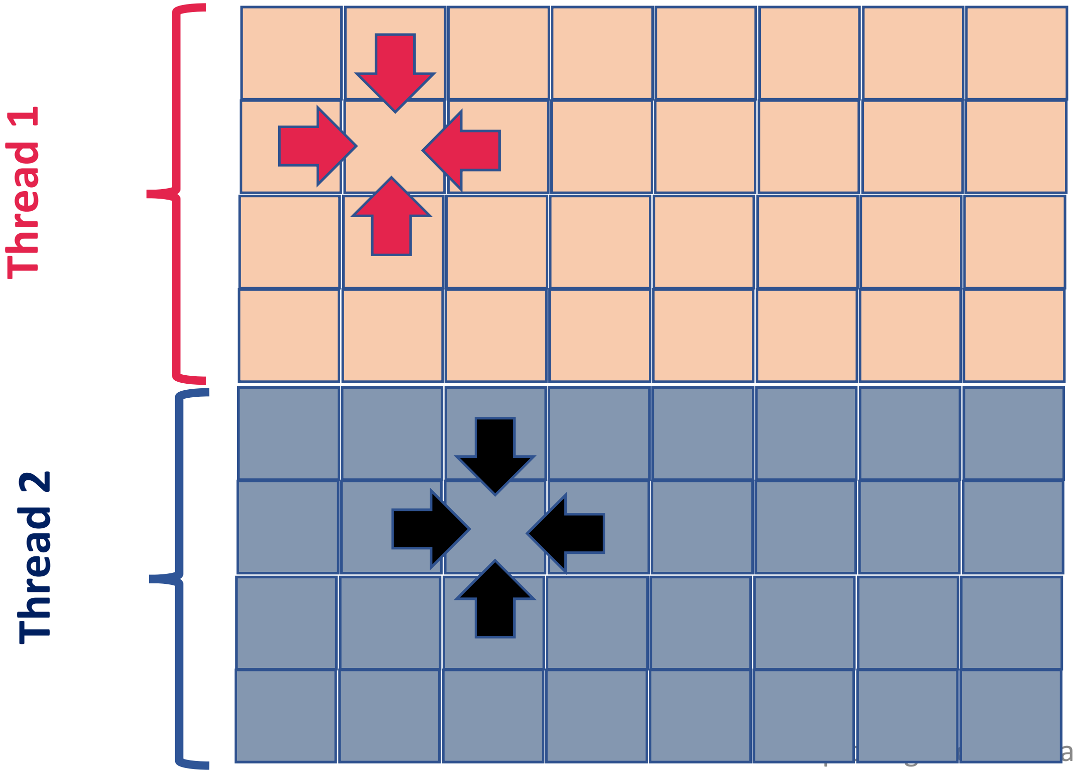
Optimization: Collaboration

# SMT locality (Stencil code)

```
#pragma omp parallel for
for (int i=T; i<N-T; i++)
  for (int j=T; j<N-T; j++)
     for (int k=0; k<T; k++)
       R[i][j] = matrix[i][j]
               + matrix[i-k][j]+matrix[i][j-k]
               + matrix[i+k][j]+matrix[i][j+k];
```

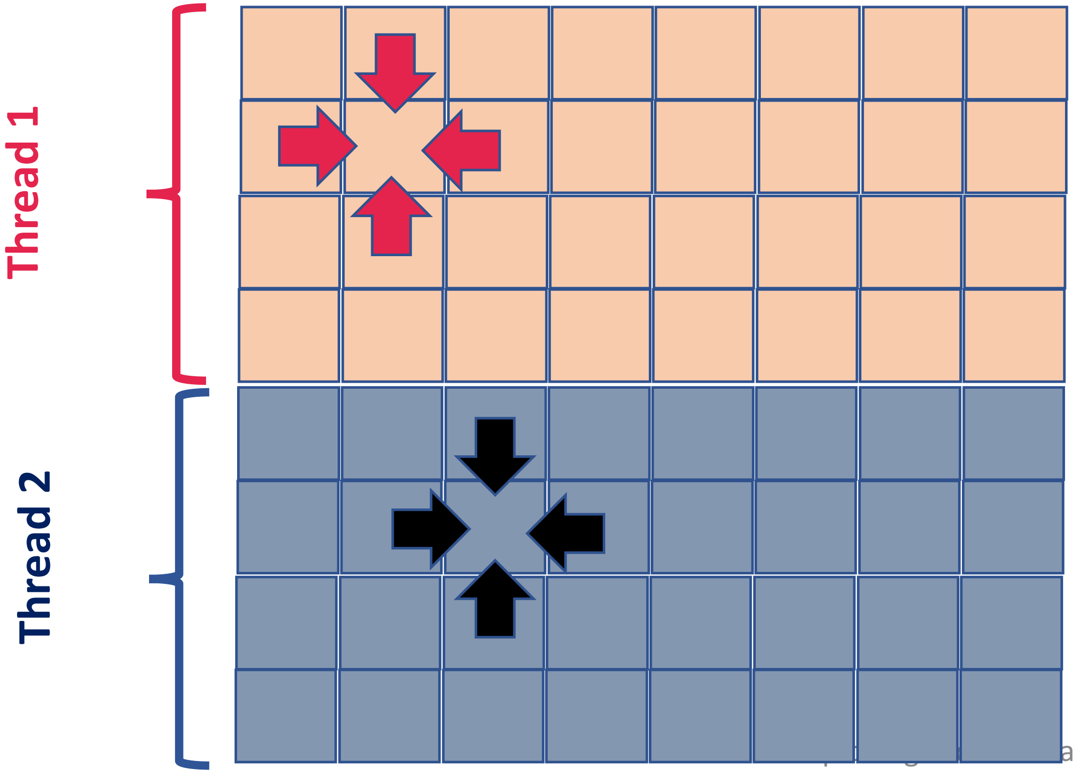
# SMT locality (Stencil code)

```
#pragma omp parallel for
for (int i=T; i<N-T; i++)
  for (int j=T; j<N-T; j++)
     for (int k=0; k<T; k++)
       R[i][j] = matrix[i][j]
               + matrix[i-k][j]+matrix[i][j-k]
               + matrix[i+k][j]+matrix[i][j+k];
```



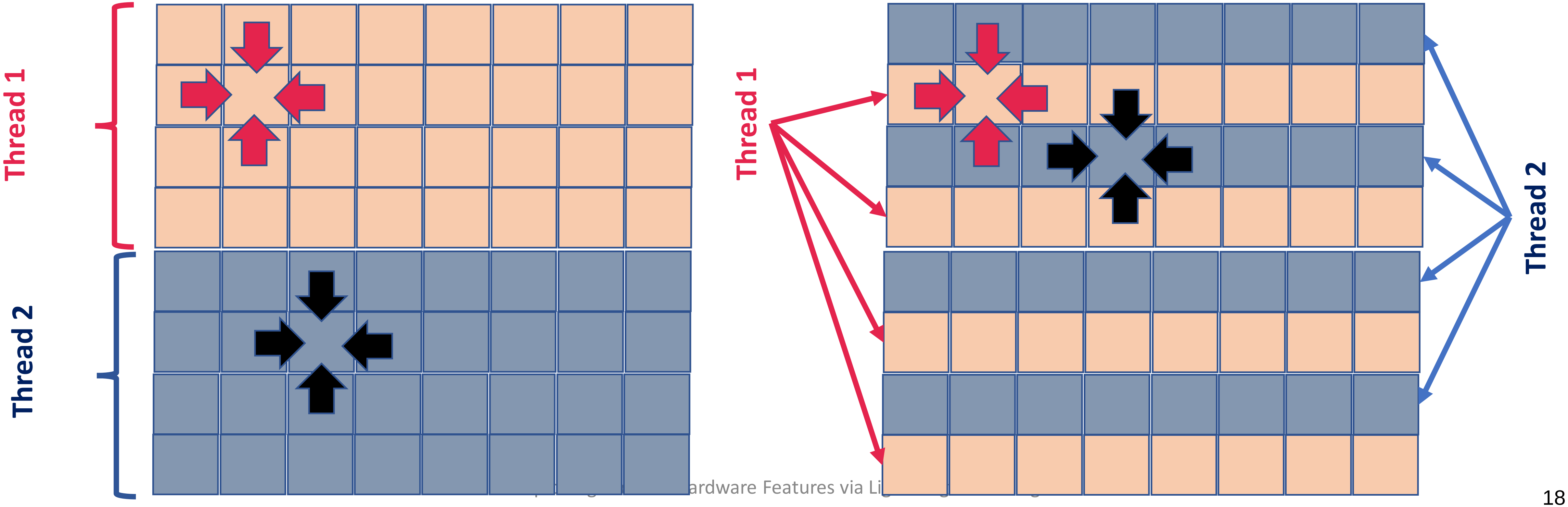
# SMT locality (Stencil code)

```
#pragma omp parallel for schedule(static,1)
for (int i=T; i<N-T; i++)
  for (int j=T; j<N-T; j++)
     for (int k=0; k<T; k++)
        R[i][j] = matrix[i][j]
                 + matrix[i-k][j]+matrix[i][j-k]
                 + matrix[i+k][j]+matrix[i][j+k];
```



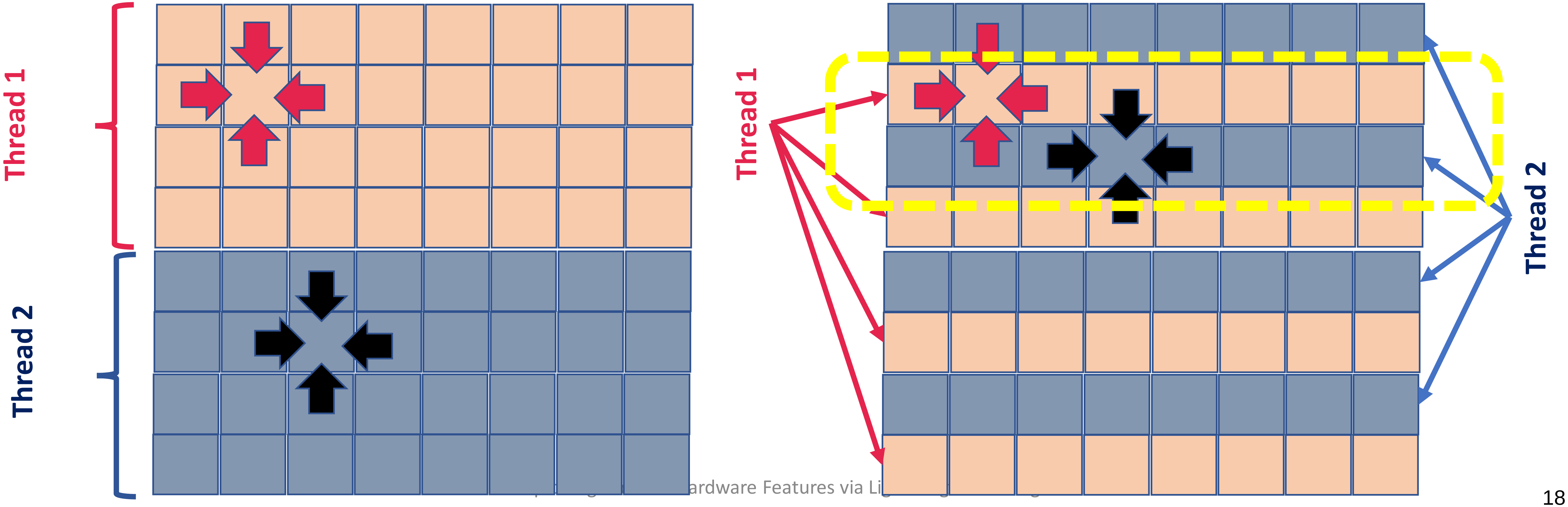
# SMT locality (Stencil code)

```
#pragma omp parallel for schedule(static,1)
for (int i=T; i<N-T; i++)
   for (int j=T; j<N-T; j++)
      for (int k=0; k<T; k++)
         R[i][j] = matrix[i][j]
                 + matrix[i-k][j]+matrix[i][j-k]
                 + matrix[i+k][j]+matrix[i][j+k];
```

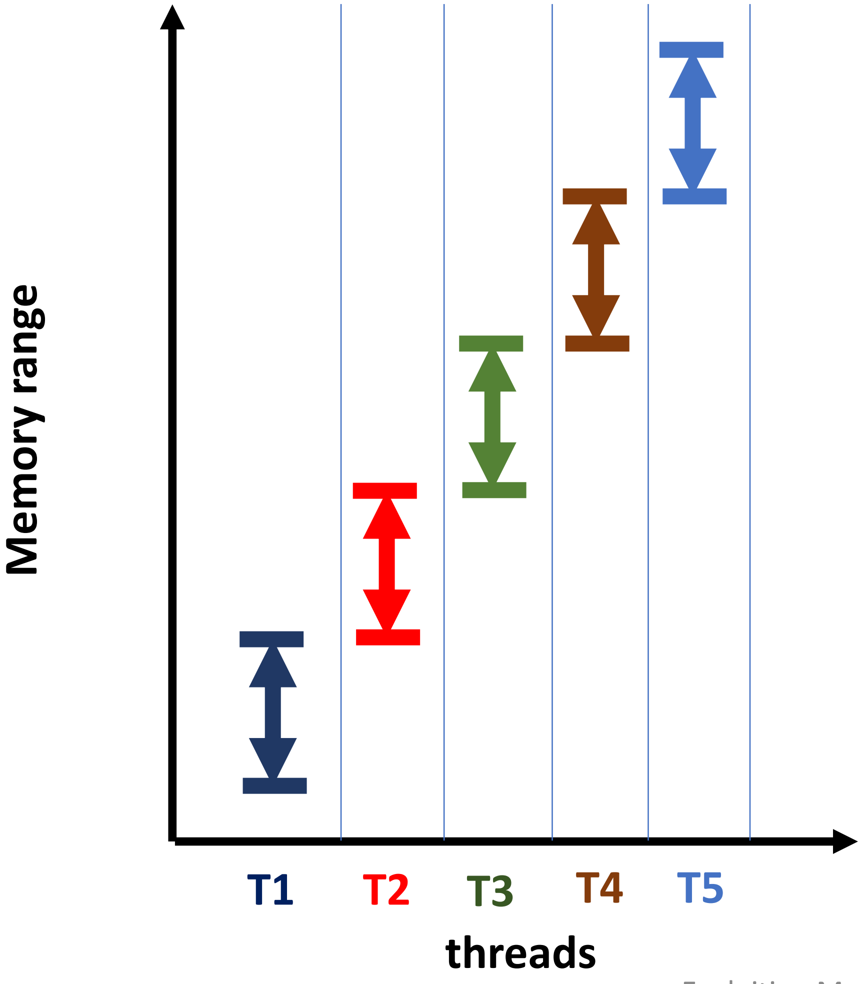


# SMT locality (Stencil code)

```
#pragma omp parallel for schedule(static,1)
for (int i=T; i<N-T; i++)
  for (int j=T; j<N-T; j++)
     for (int k=0; k<T; k++)
        R[i][j] = matrix[i][j]
               + matrix[i-k][j]+matrix[i][j-k]
               + matrix[i+k][j]+matrix[i][j+k];
```

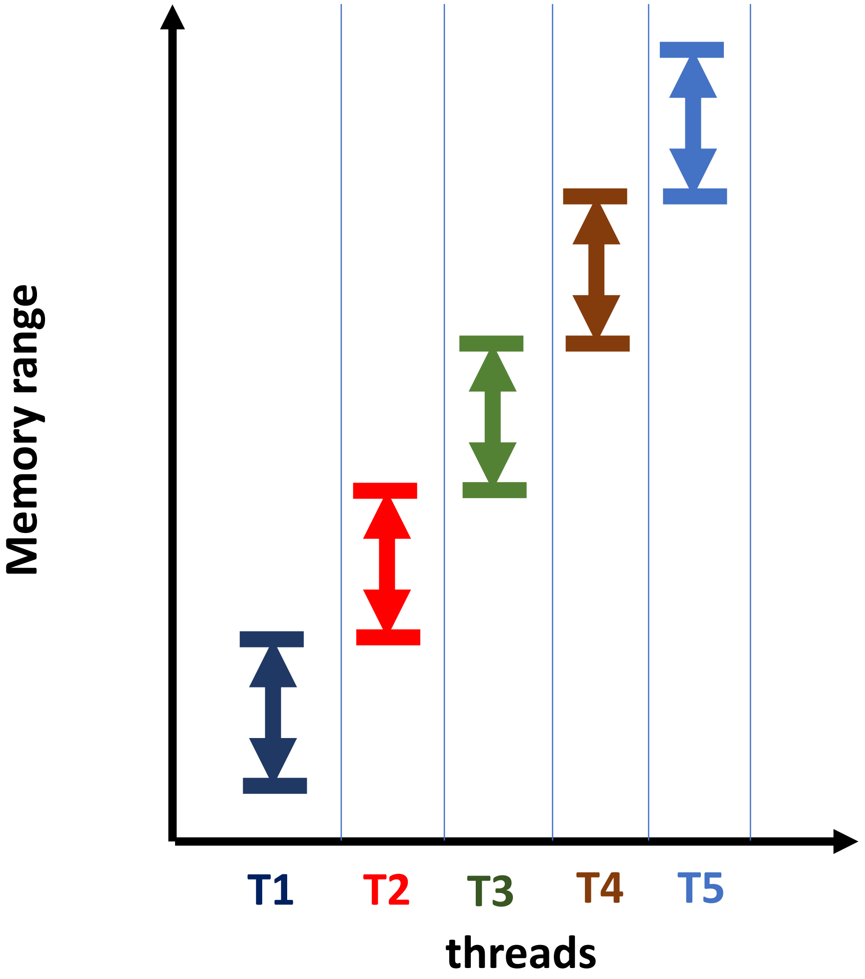


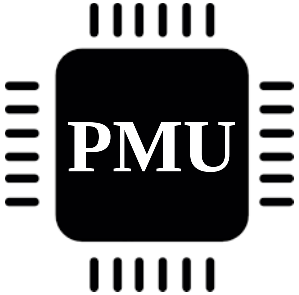
# SMT-Analyzer: Analyzing memory access pattern



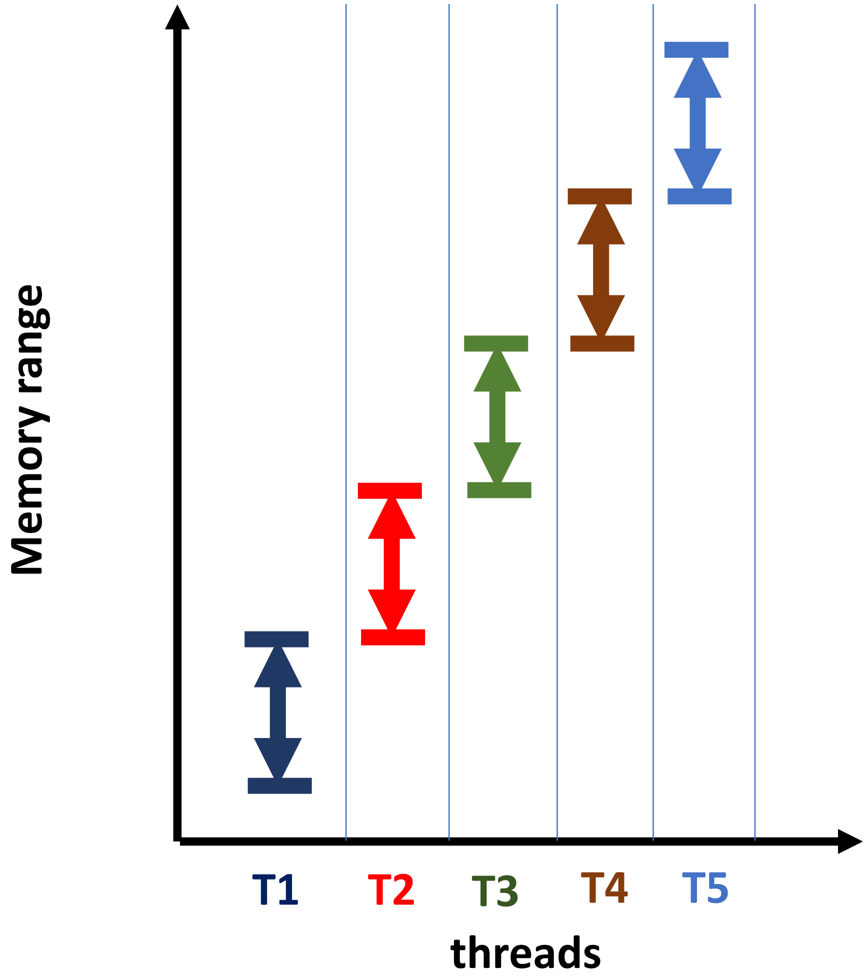
Loop

# SMT-Analyzer: Analyzing memory access pattern

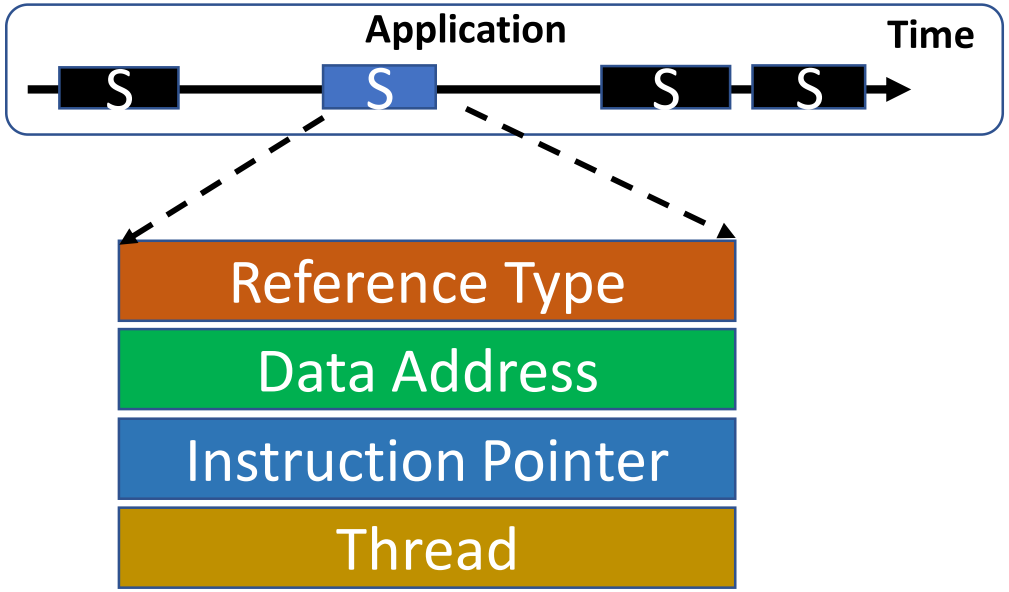




# SMT-Analyzer: Analyzing memory access pattern



PMU



Reference Type

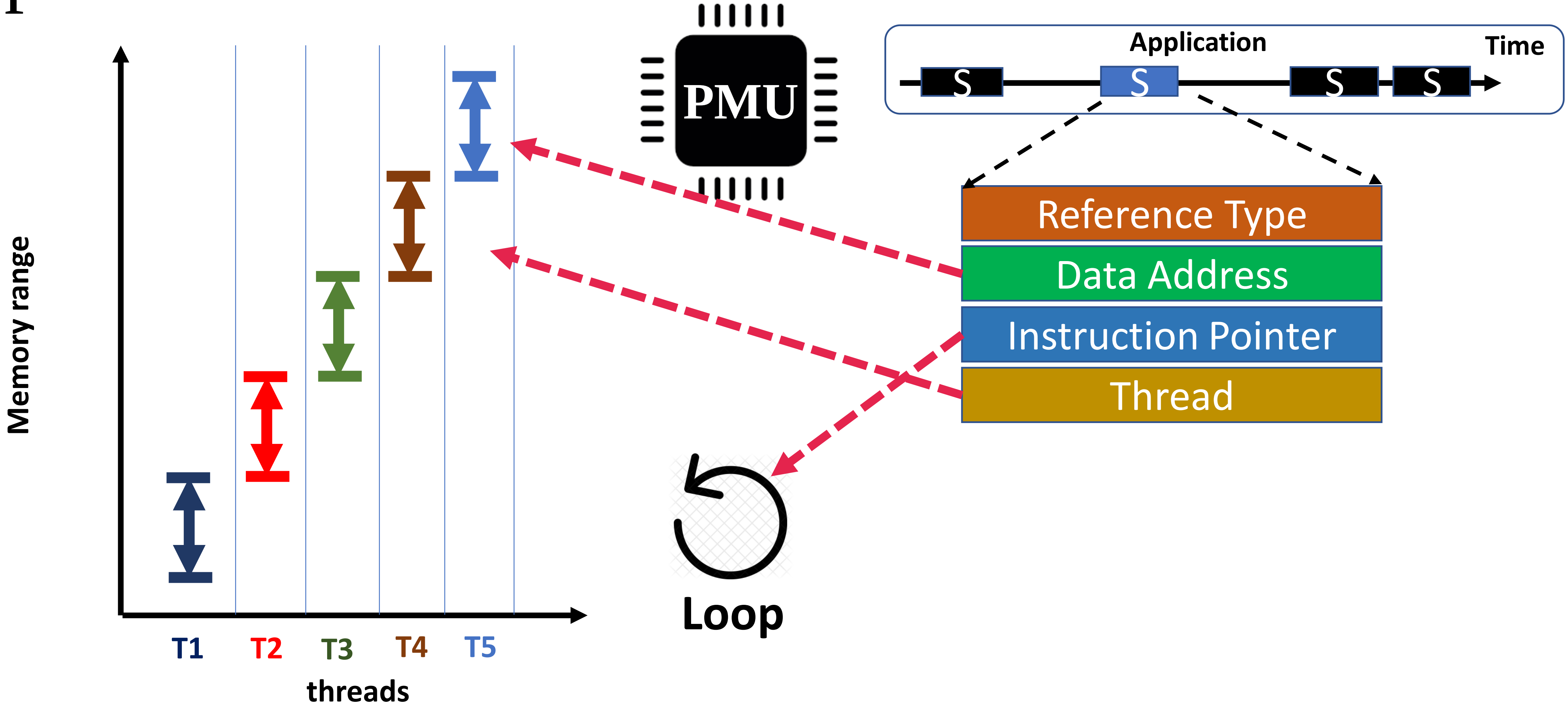
Data Address

Instruction Pointer

Thread

Loop

# SMT-Analyzer: Analyzing memory access pattern



Reference Type

Data Address

Instruction Pointer

Thread

# Benchmarks

| Benchmarks | bottleneck region | % of total latency | overhead | OPT method | Speedups |
|---|---|---|---|---|---|
| lulesh2.0 | lulesh.cc: 604-609 | 3.6% | +3.1% | inter-thread | 1.43× |
| IRSmk | rmatmult3.c: 86-103 | 78.6% | +3.2% | intra-thread | 4.86× |
| needle | needle.cpp:185-187 | 20% | +2.99% | inter-thread | 2.37× |
| srad | srad.cpp:136-167 | 80.1% | +2.47% | intra-thread | 1.74× |
| LU | rhs.f:318-328 | 8.4% | +10.6% | inter-thread | 1.36× |
| Stencil | stencil.c:16-21 | 95.7% | +1.55% | inter-thread | 10.9× |
| 3D tensor | mt.c: 22-22 | 69.4% | +2.4% | inter-thread | 1.44× |
| streamcluster2 | streamcluster.cpp:653 | 14.1% | +15.2% | inter-thread | 6.72× |

# Benchmarks

| Benchmarks | bottleneck region | % of total latency | overhead | OPT method | Speedups |
|---|---|---|---|---|---|
| lulesh2.0 | lulesh.cc: 604-609 | 3.6% | +3.1% | inter-thread | 1.43× |
| IRSmk | rmatmult3.c: 86-103 | 78.6% | +3.2% | intra-thread | 4.86× |
| needle | needle.cpp:185-187 | 20% | +2.99% | inter-thread | 2.37× |
| srad | srad.cpp:136-167 | 80.1% | +2.47% | intra-thread | 1.74× |
| LU | rhs.f:318-328 | 8.4% | +10.6% | inter-thread | 1.36× |
| Stencil | stencil.c:16-21 | 95.7% | +1.55% | inter-thread | 10.9× |
| 3D tensor | mt.c: 22-22 | 69.4% | +2.4% | inter-thread | 1.44× |
| streamcluster2 | streamcluster.cpp:653 | 14.1% | +15.2% | inter-thread | 6.72× |

# Benchmarks

| Benchmarks | bottleneck region | % of total latency | overhead | OPT method | Speedups |
|---|---|---|---|---|---|
| lulesh2.0 | lulesh.cc: 604-609 | 3.6% | +3.1% | inter-thread | 1.43× |
| IRSmk | rmatmult3.c: 86-103 | 78.6% | +3.2% | intra-thread | 4.86× |
| needle | needle.cpp:185-187 | 20% | +2.99% | inter-thread | 2.37× |
| srad | srad.cpp:136-167 | 80.1% | +2.47% | intra-thread | 1.74× |
| LU | rhs.f:318-328 | 8.4% | +10.6% | inter-thread | 1.36× |
| Stencil | stencil.c:16-21 | 95.7% | +1.55% | inter-thread | 10.9× |
| 3D tensor | mt.c: 22-22 | 69.4% | +2.4% | inter-thread | 1.44× |
| streamcluster2 | streamcluster.cpp:653 | 14.1% | +15.2% | inter-thread | 6.72× |

# Benchmarks

| Benchmarks | bottleneck region | % of total latency | overhead | OPT method | Speedups |
|---|---|---|---|---|---|
| lulesh2.0 | lulesh.cc: 604-609 | 3.6% | +3.1% | inter-thread | 1.43× |
| IRSmk | rmatmult3.c: 86-103 | 78.6% | +3.2% | intra-thread | 4.86× |
| needle | needle.cpp:185-187 | 20% | +2.99% | inter-thread | 2.37× |
| srad | srad.cpp:136-167 | 80.1% | +2.47% | intra-thread | 1.74× |
| LU | rhs.f:318-328 | 8.4% | +10.6% | inter-thread | 1.36× |
| Stencil | stencil.c:16-21 | 95.7% | +1.55% | inter-thread | 10.9× |
| 3D tensor | mt.c: 22-22 | 69.4% | +2.4% | inter-thread | 1.44× |
| streamcluster2 | streamcluster.cpp:653 | 14.1% | +15.2% | inter-thread | 6.72× |

# Benchmarks

| Benchmarks | bottleneck region | % of total latency | overhead | OPT method | Speedups |
|---|---|---|---|---|---|
| lulesh2.0 | lulesh.cc: 604-609 | 3.6% | +3.1% | inter-thread | 1.43× |
| IRSmk | rmatmult3.c: 86-103 | 78.6% | +3.2% | intra-thread | 4.86× |
| needle | needle.cpp:185-187 | 20% | +2.99% | inter-thread | 2.37× |
| srad | srad.cpp:136-167 | 80.1% | +2.47% | intra-thread | 1.74× |
| LU | rhs.f:318-328 | 8.4% | +10.6% | inter-thread | 1.36× |
| Stencil | stencil.c:16-21 | 95.7% | +1.55% | inter-thread | 10.9× |
| 3D tensor | mt.c: 22-22 | 69.4% | +2.4% | inter-thread | 1.44× |
| streamcluster2 | streamcluster.cpp:653 | 14.1% | +15.2% | inter-thread | 6.72× |

Related work: MACPO (selective instrumentation): 2x - 5x

# Outline

- ✓Lightweight profiling
- ✓SMT-aware optimization
- *Detection of cache conflicts*
- Guiding data-structure layout transformation

# Lightweight Detection of Cache Conflicts

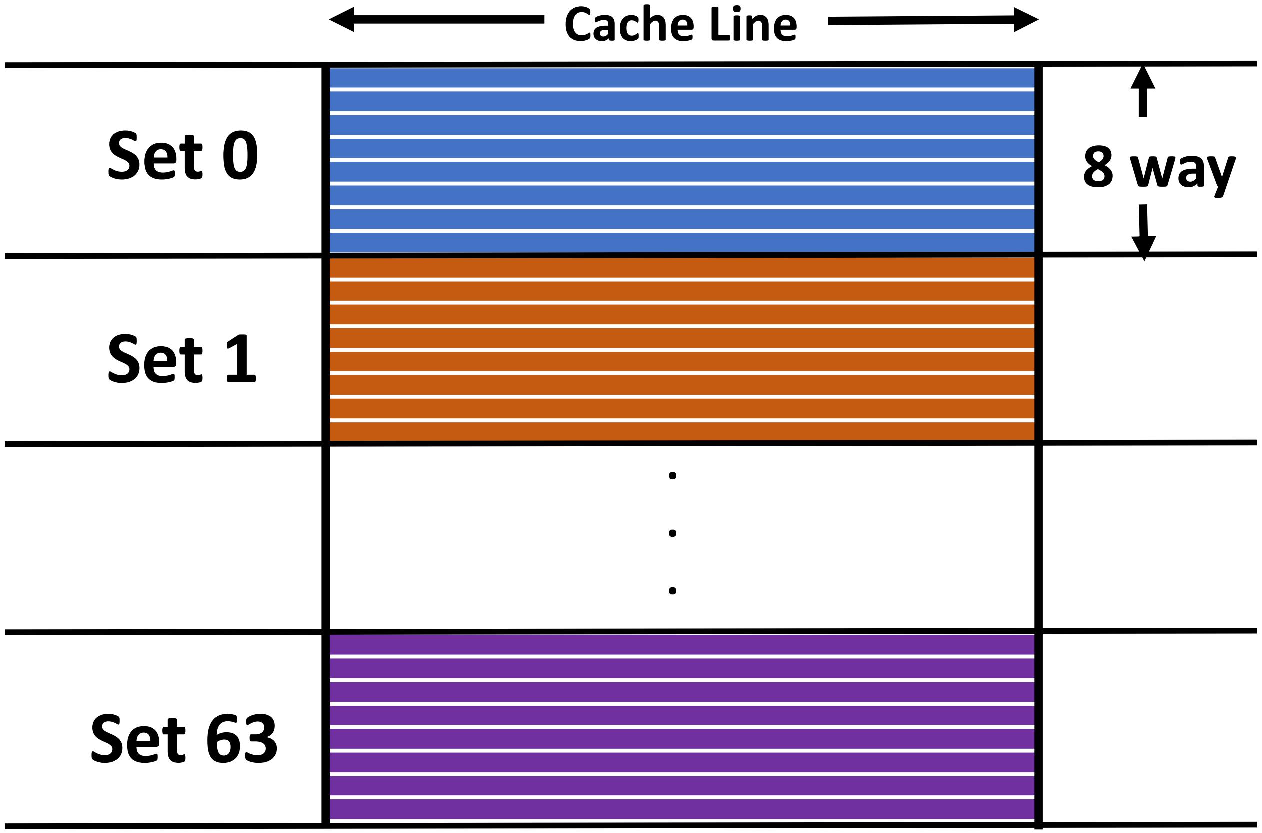
[CGO – 2018]

**Probir Roy**, Shuaiwen Leon Song, Sriram Krishnamoorthy, *Xu Liu*

WILLIAM & MARY
CHARTERED 1693

Pacific Northwest
NATIONAL LABORATORY

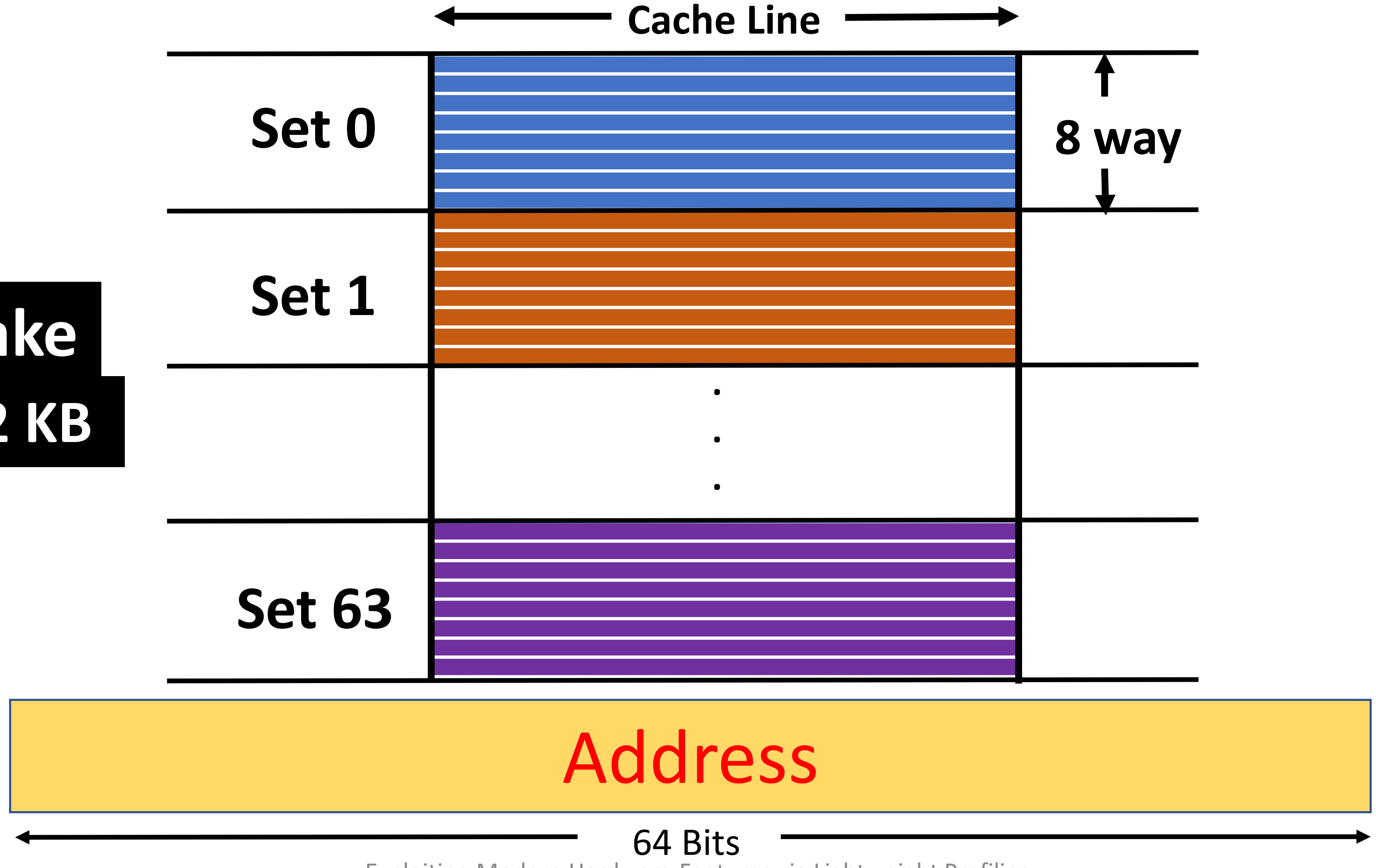
# Set-associative cache



**Intel Skylake**

**L1 cache: 32 KB**

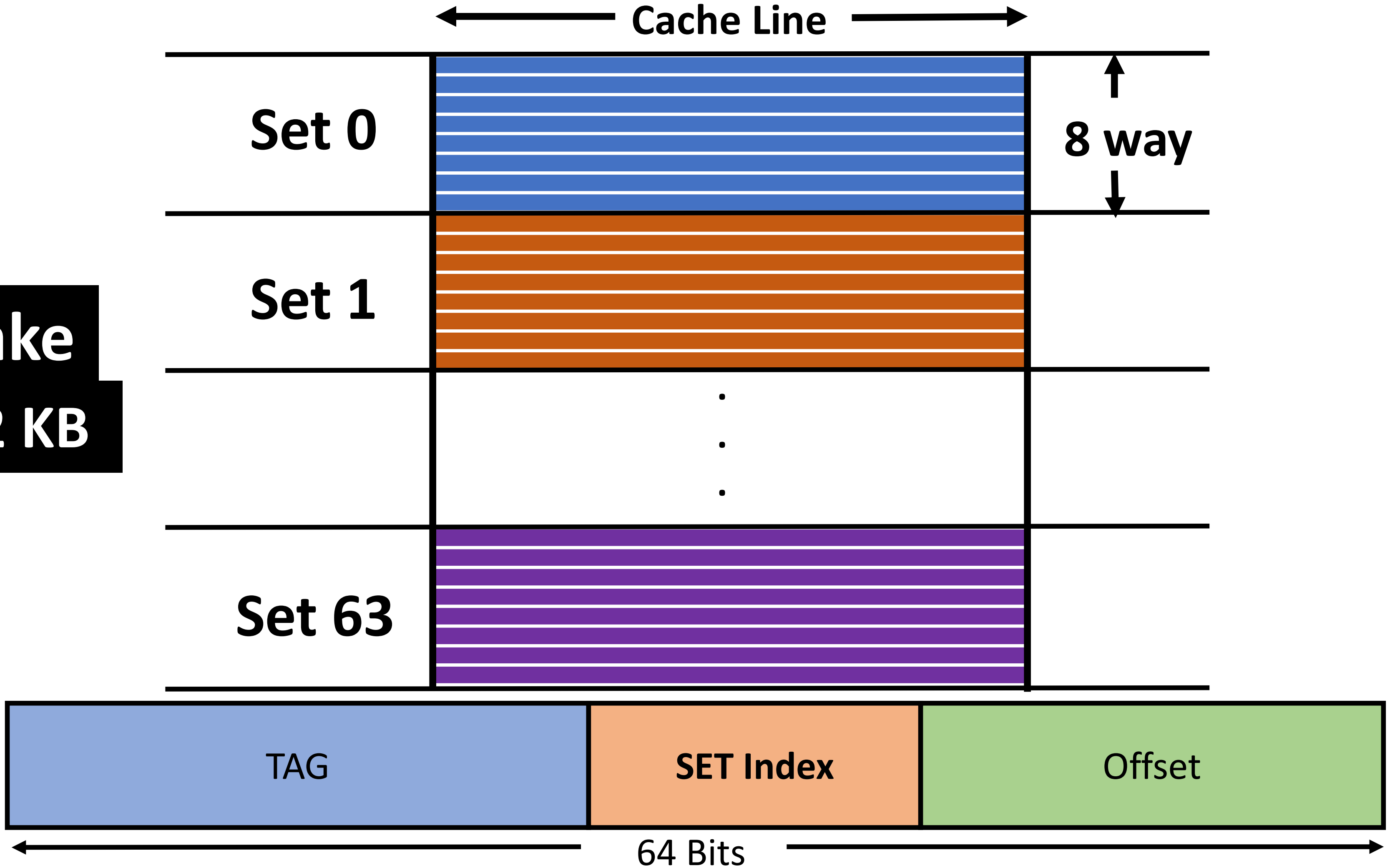
# Set-associative cache



**Intel Skylake**

**L1 cache: 32 KB**

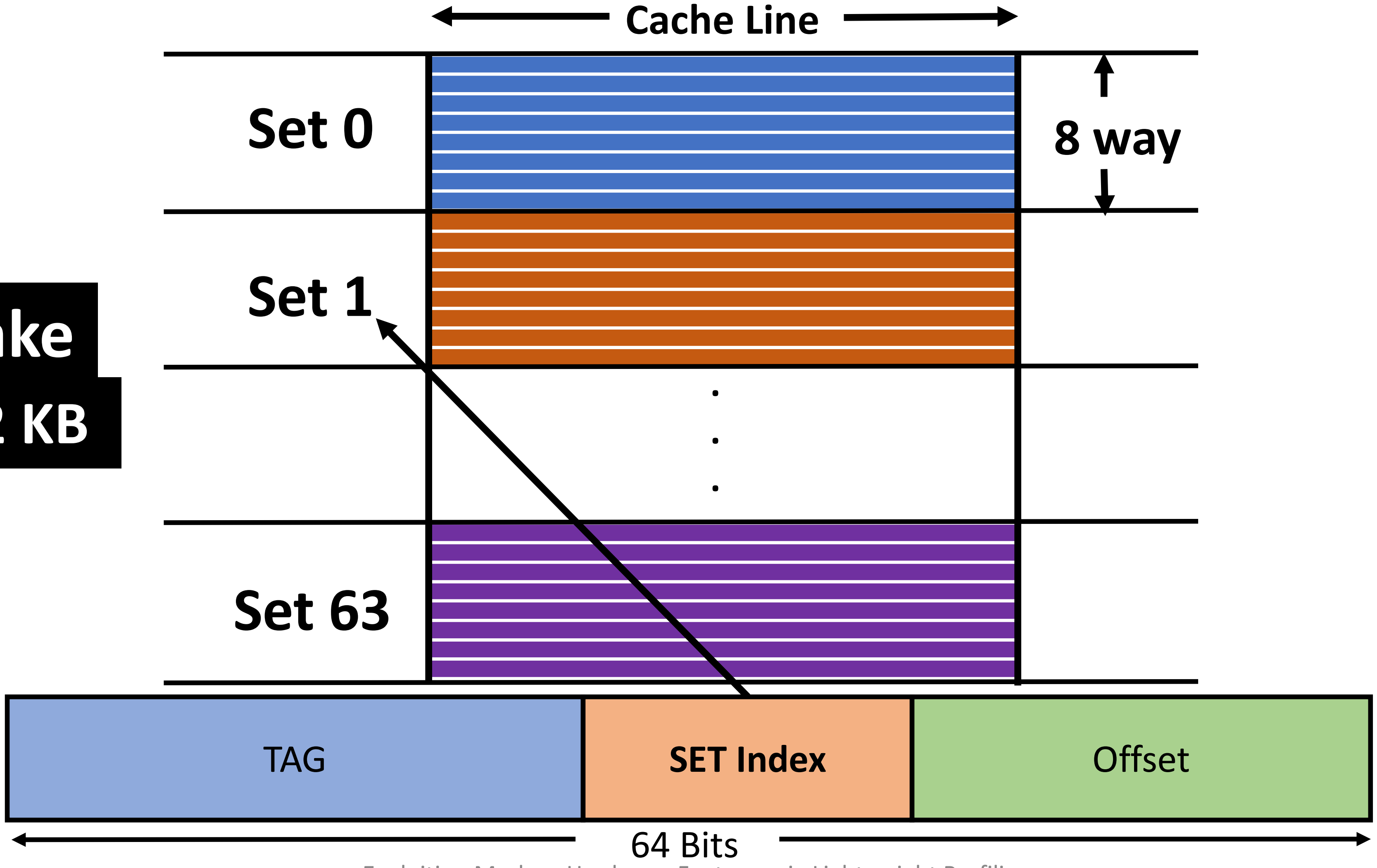
# Set-associative cache



**Intel Skylake**

**L1 cache: 32 KB**

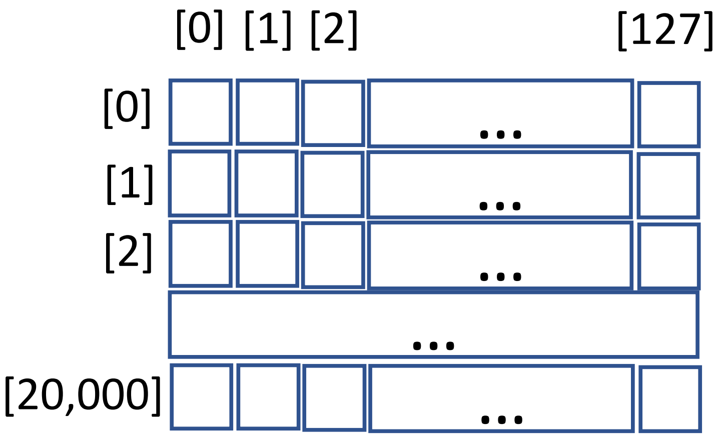
# Set-associative cache



**Intel Skylake**

**L1 cache: 32 KB**

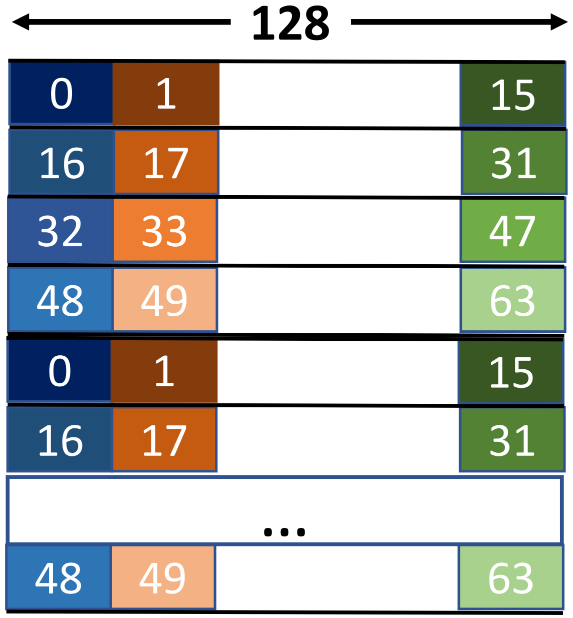
# Set conflict



```
double Array [20,000][128];
```

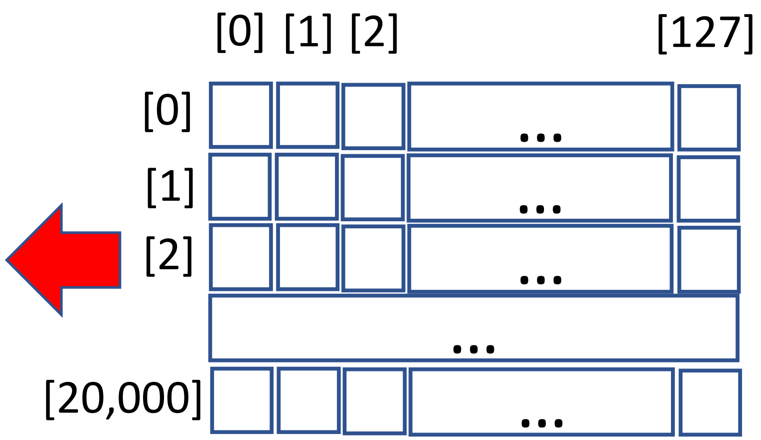
# Set conflict

**Set mapping**



**128**

| | | | |
|---|---|---|---|
| 0 | 1 | | 15 |
| 16 | 17 | | 31 |
| 32 | 33 | | 47 |
| 48 | 49 | | 63 |
| 0 | 1 | | 15 |
| 16 | 17 | | 31 |
| ... | | | |
| 48 | 49 | | 63 |

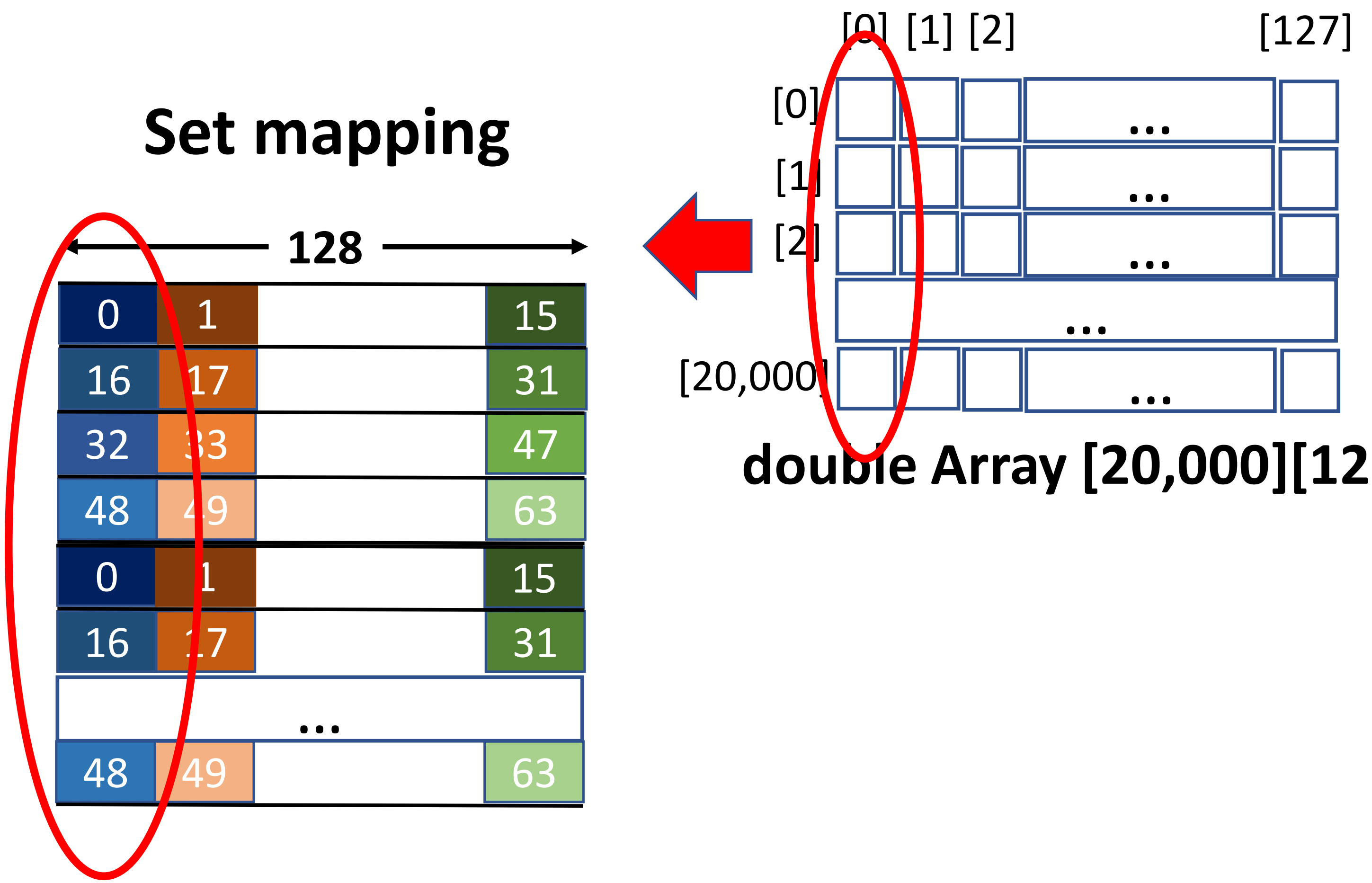


| | [0] | [1] | [2] | | [127] |
|---|---|---|---|---|---|
| [0] | | | | ... | |
| [1] | | | | ... | |
| [2] | | | | ... | |
| | ... | | | | |
| [20,000] | | | | ... | |

**double Array [20,000][128];**

# Set conflict

**Set mapping**

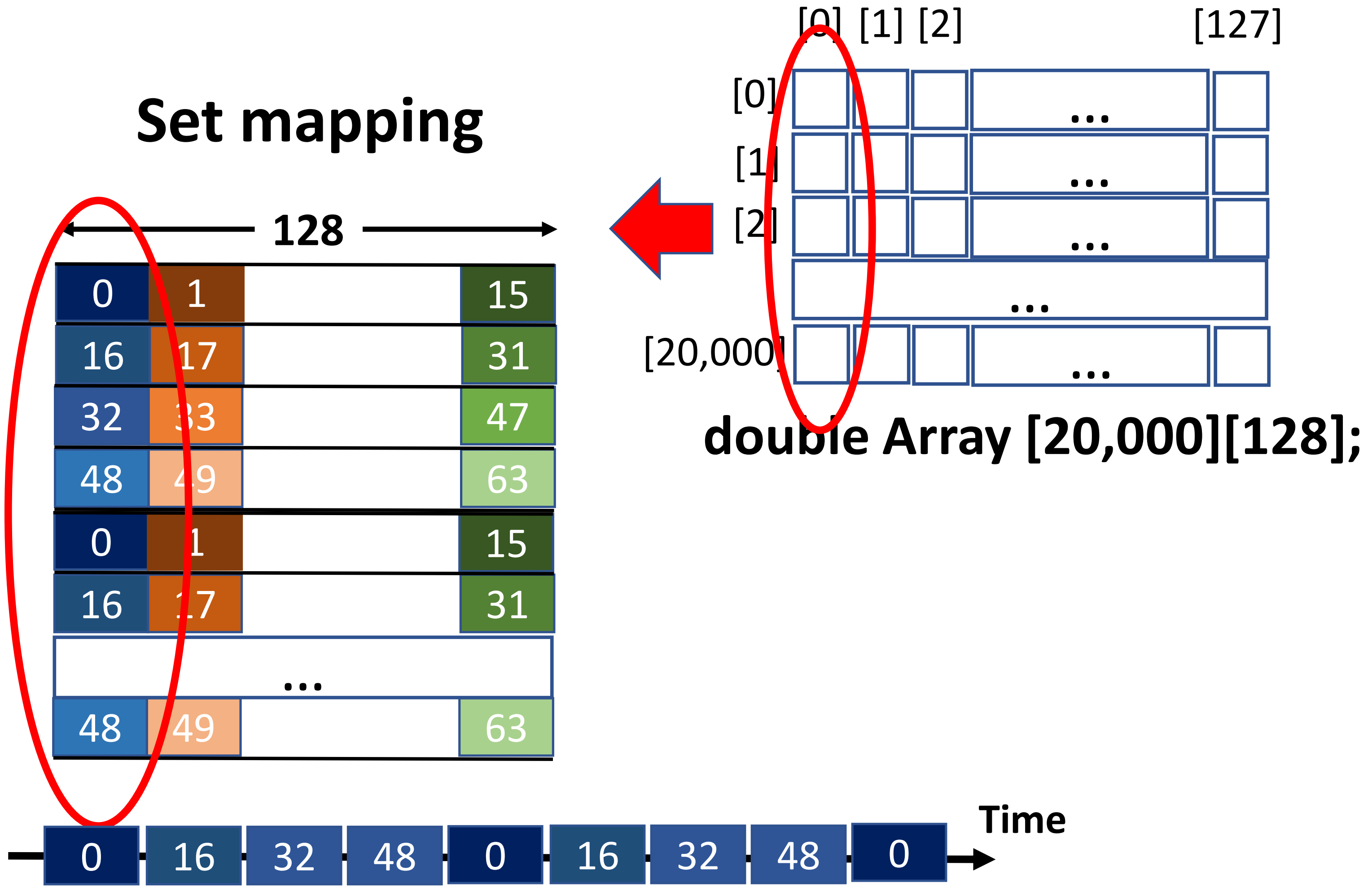


| 128 | | |
|---|---|---|
| 0 | 1 | 15 |
| 16 | 17 | 31 |
| 32 | 33 | 47 |
| 48 | 49 | 63 |
| 0 | 1 | 15 |
| 16 | 17 | 31 |
| | ... | |
| 48 | 49 | 63 |

| | [0] | [1] | [2] | | [127] |
|---|---|---|---|---|---|
| [0] | | | | ... | |
| [1] | | | | ... | |
| [2] | | | | ... | |
| | | | ... | | |
| [20,000] | | | | ... | |

**double Array [20,000][128];**

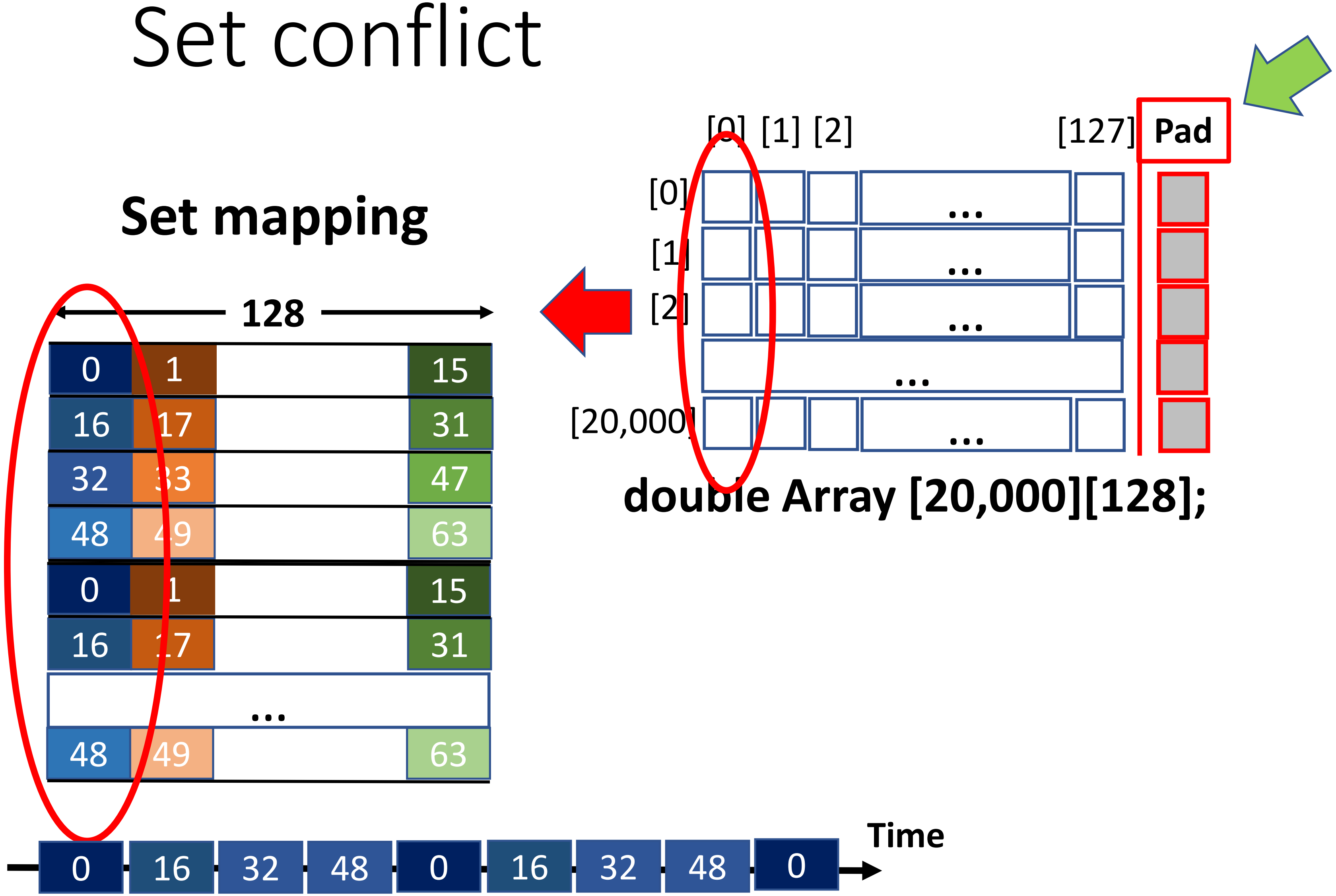
# Set conflict



double Array [20,000][128];

# Set conflict

## Set mapping

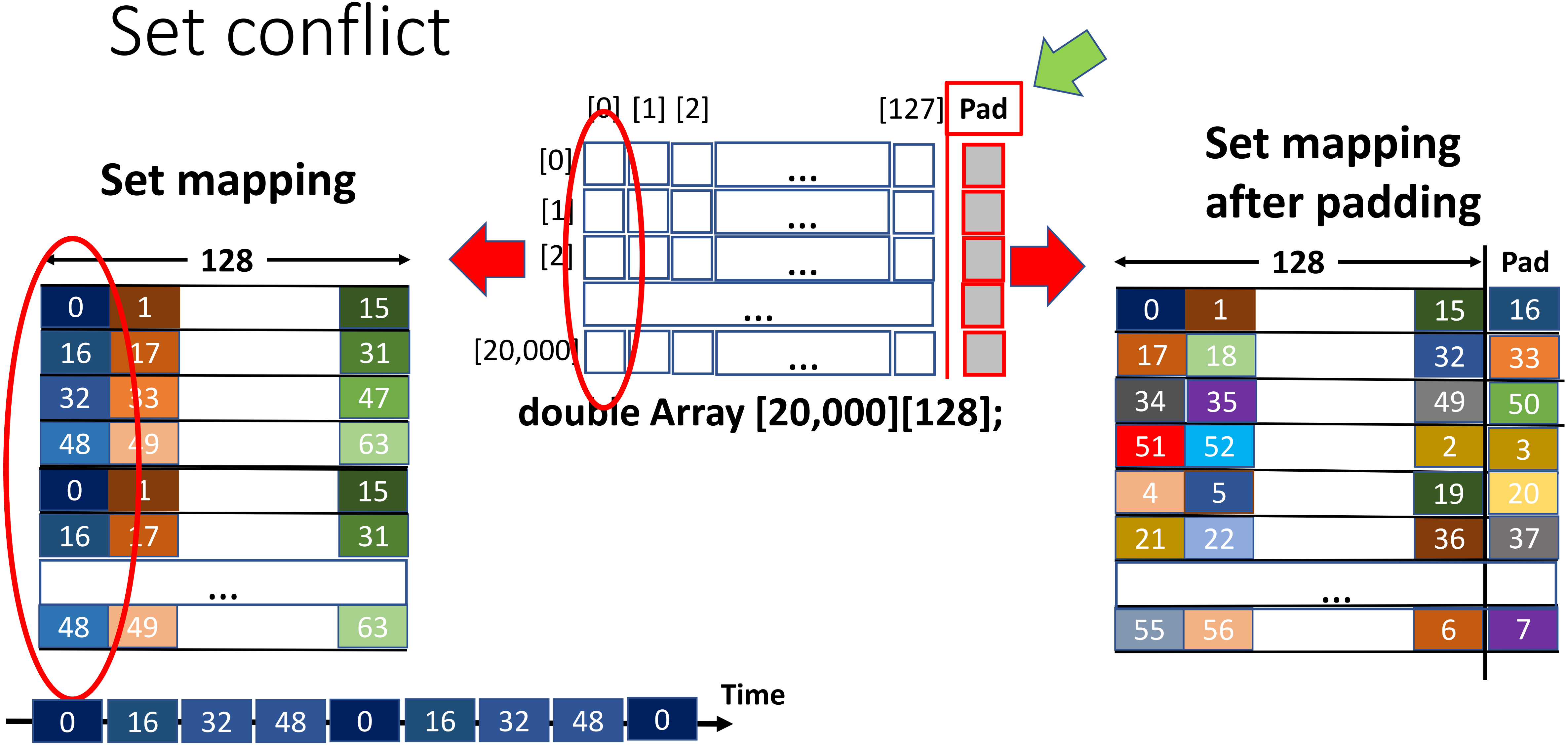


| | | | |
|---|---|---|---|
| 0 | 1 | | 15 |
| 16 | 17 | | 31 |
| 32 | 33 | | 47 |
| 48 | 49 | | 63 |
| 0 | 1 | | 15 |
| 16 | 17 | | 31 |
| ... | | | |
| 48 | 49 | | 63 |

0 16 32 48 0 16 32 48 0

**double Array [20,000][128];**

# Set conflict



**double Array [20,000][128];**

**Set mapping**

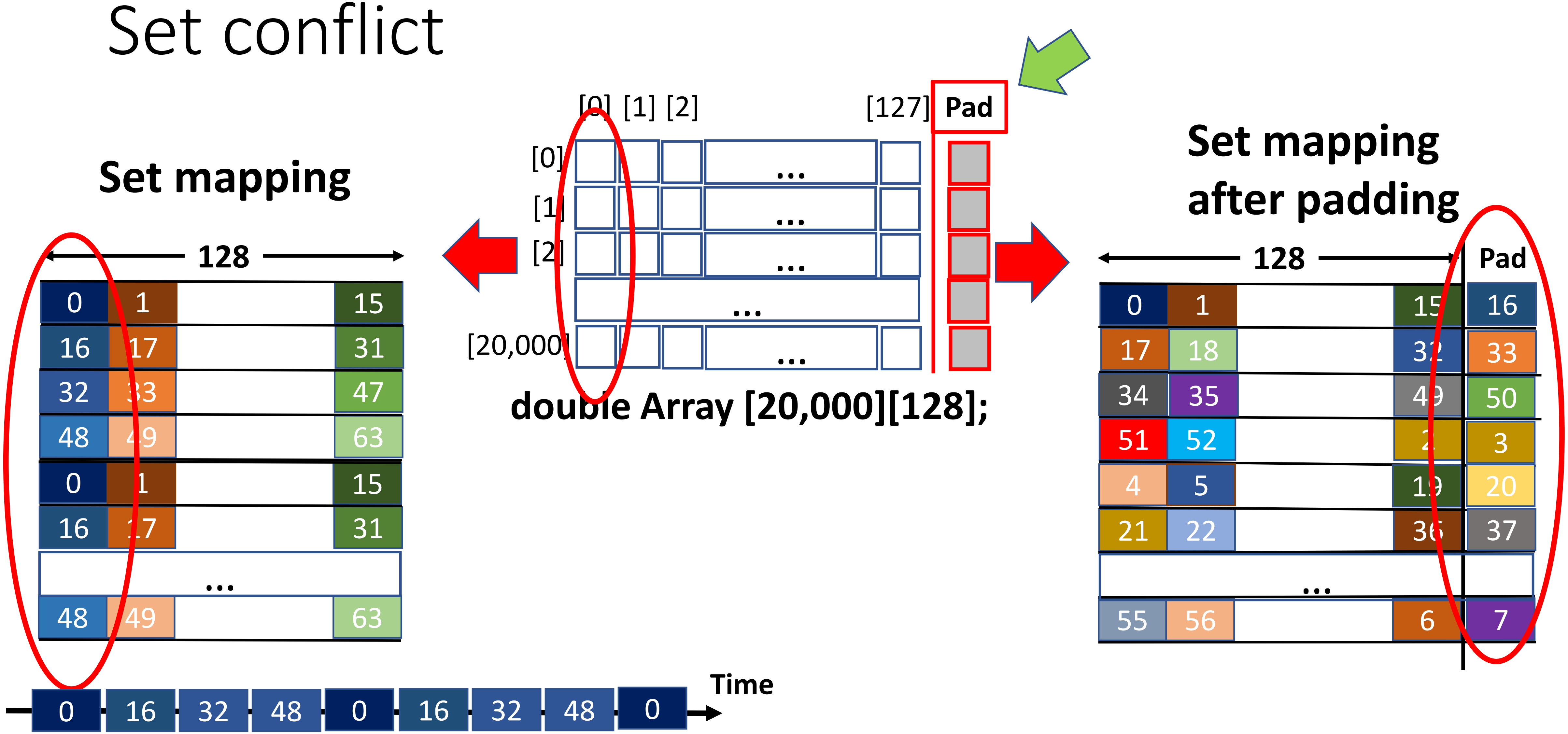
| | | | |
|---|---|---|---|
| 0 | 1 | | 15 |
| 16 | 17 | | 31 |
| 32 | 33 | | 47 |
| 48 | 49 | | 63 |
| 0 | 1 | | 15 |
| 16 | 17 | | 31 |
| | | ... | |
| 48 | 49 | | 63 |

**Set mapping after padding**

| | | | | Pad |
|---|---|---|---|---|
| 0 | 1 | | 15 | 16 |
| 17 | 18 | | 32 | 33 |
| 34 | 35 | | 49 | 50 |
| 51 | 52 | | 2 | 3 |
| 4 | 5 | | 19 | 20 |
| 21 | 22 | | 36 | 37 |
| | | ... | | |
| 55 | 56 | | 6 | 7 |

0 16 32 48 0 16 32 48 0

# Set conflict



double Array [20,000][128];

Set mapping

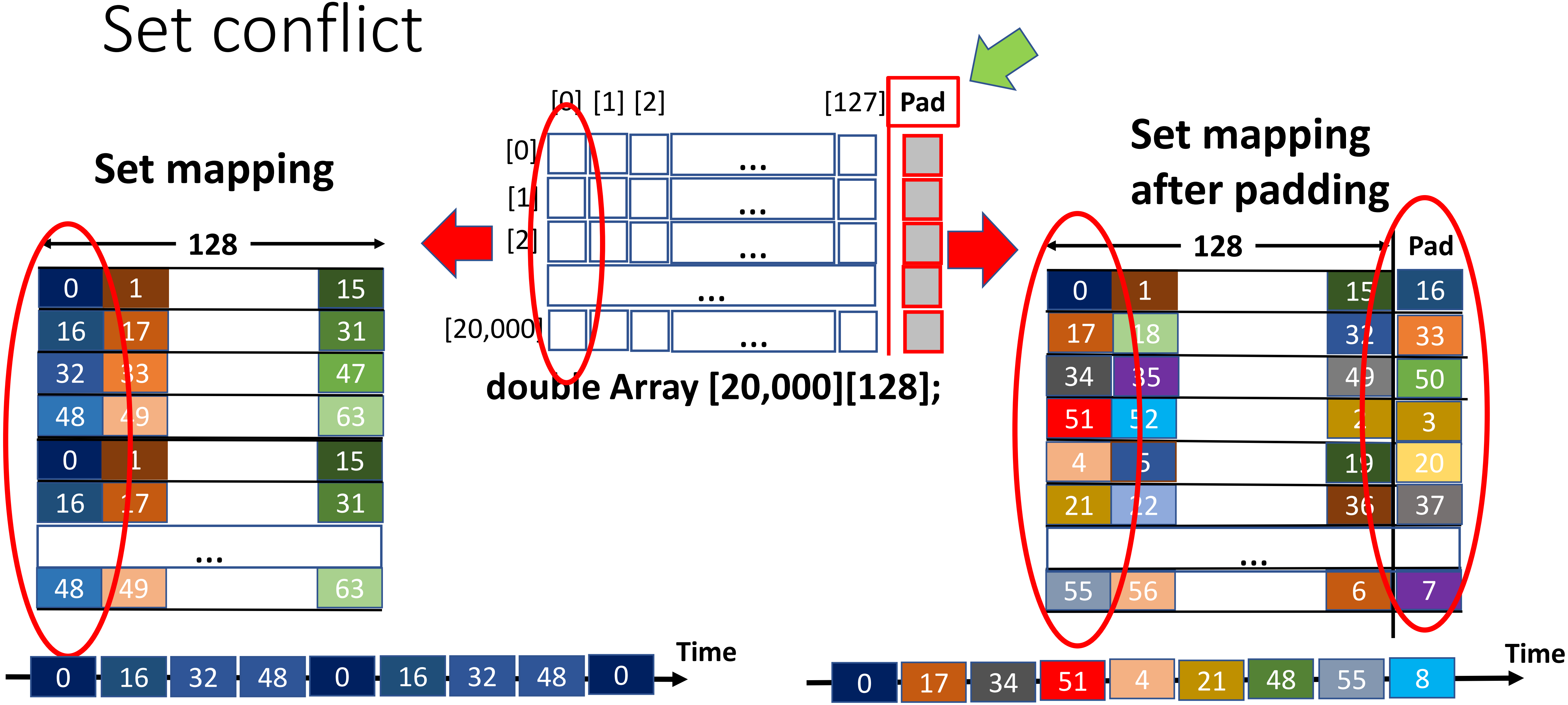
| | | | |
|---|---|---|---|
| 0 | 1 | | 15 |
| 16 | 17 | | 31 |
| 32 | 33 | | 47 |
| 48 | 49 | | 63 |
| 0 | 1 | | 15 |
| 16 | 17 | | 31 |
| | ... | | |
| 48 | 49 | | 63 |

0 16 32 48 0 16 32 48 0 Time

Set mapping after padding

| | | | | Pad |
|---|---|---|---|---|
| 0 | 1 | | 15 | 16 |
| 17 | 18 | | 32 | 33 |
| 34 | 35 | | 49 | 50 |
| 51 | 52 | | 2 | 3 |
| 4 | 5 | | 19 | 20 |
| 21 | 22 | | 36 | 37 |
| | ... | | | |
| 55 | 56 | | 6 | 7 |

# Set conflict

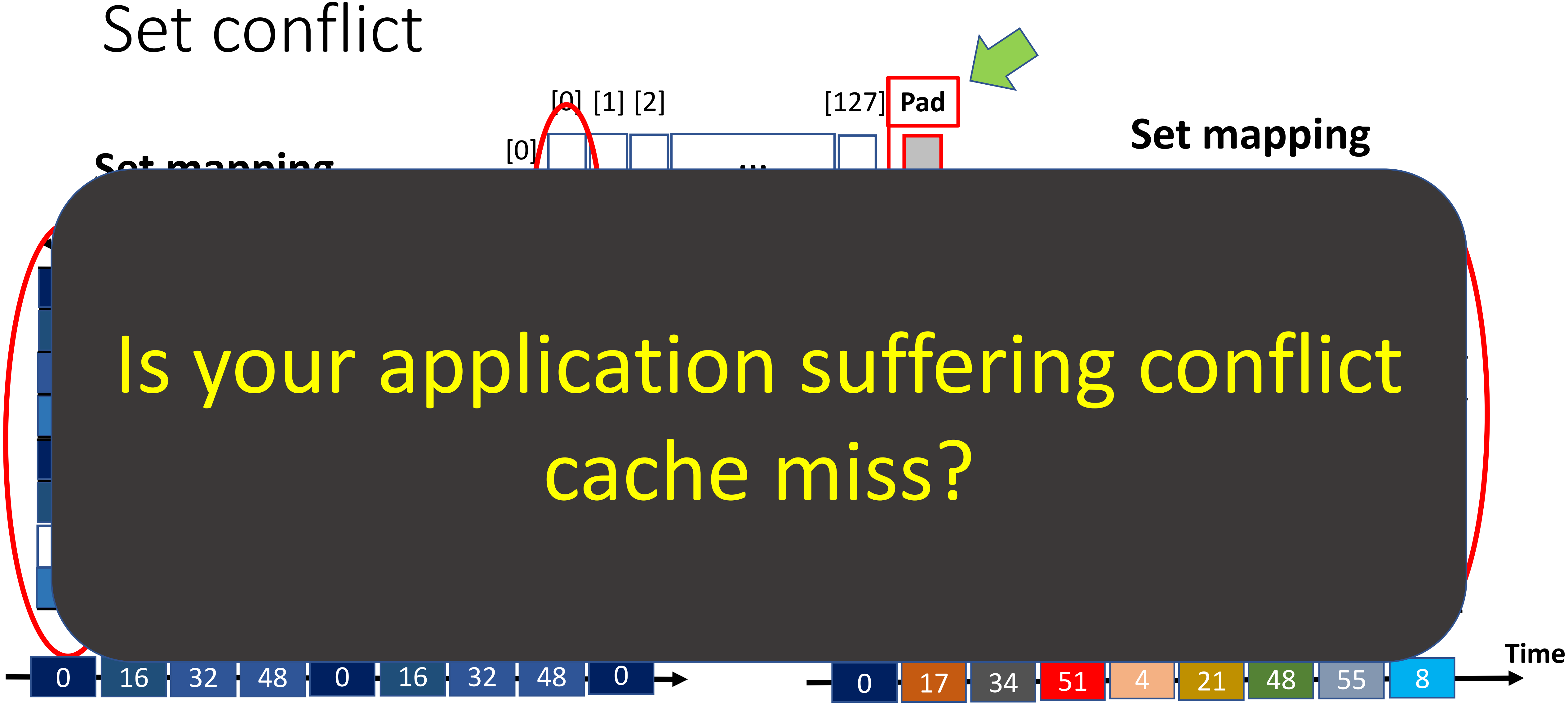


Set mapping

double Array [20,000][128];

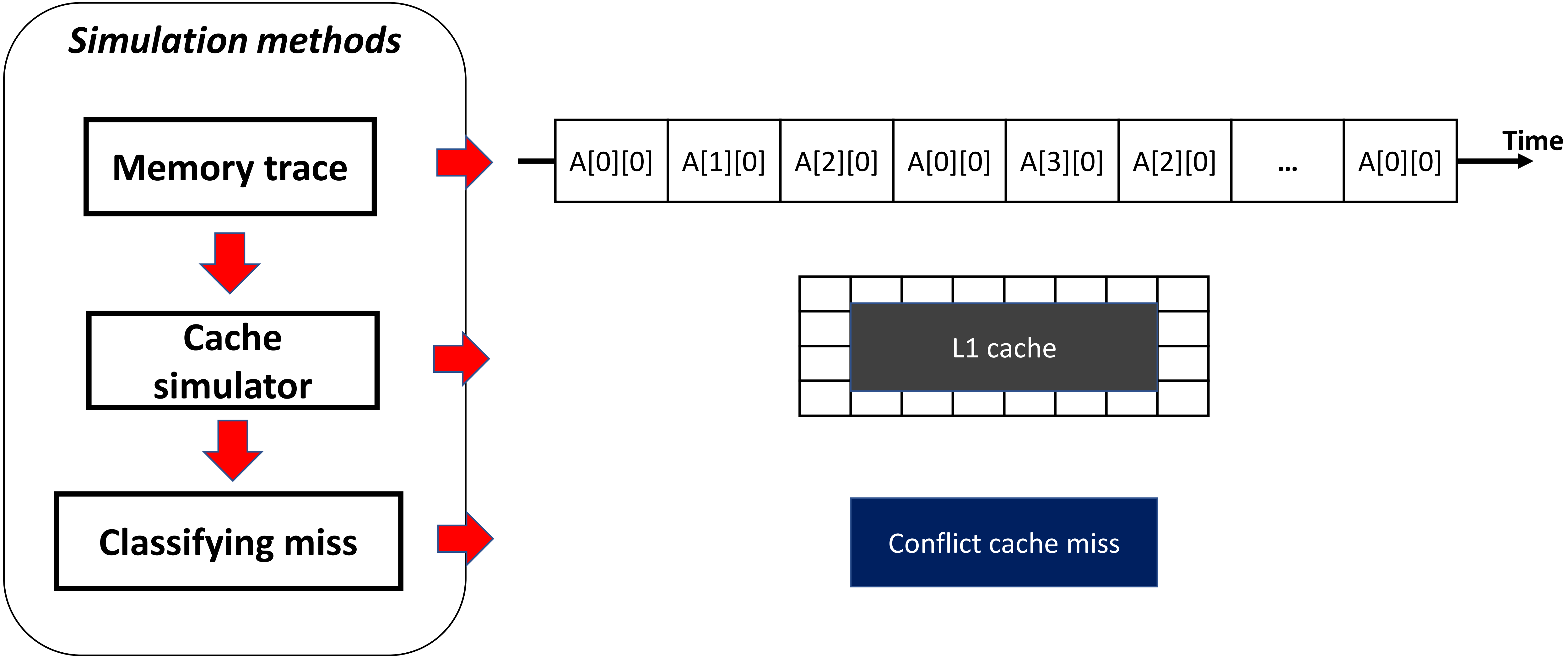
Set mapping after padding

# Set conflict



Is your application suffering conflict cache miss?

# Trace driven cache simulation



*Simulation methods*

**Memory trace**

**Cache simulator**

**Classifying miss**

| A[0][0] | A[1][0] | A[2][0] | A[0][0] | A[3][0] | A[2][0] | … | A[0][0] |
|---|---|---|---|---|---|---|---|

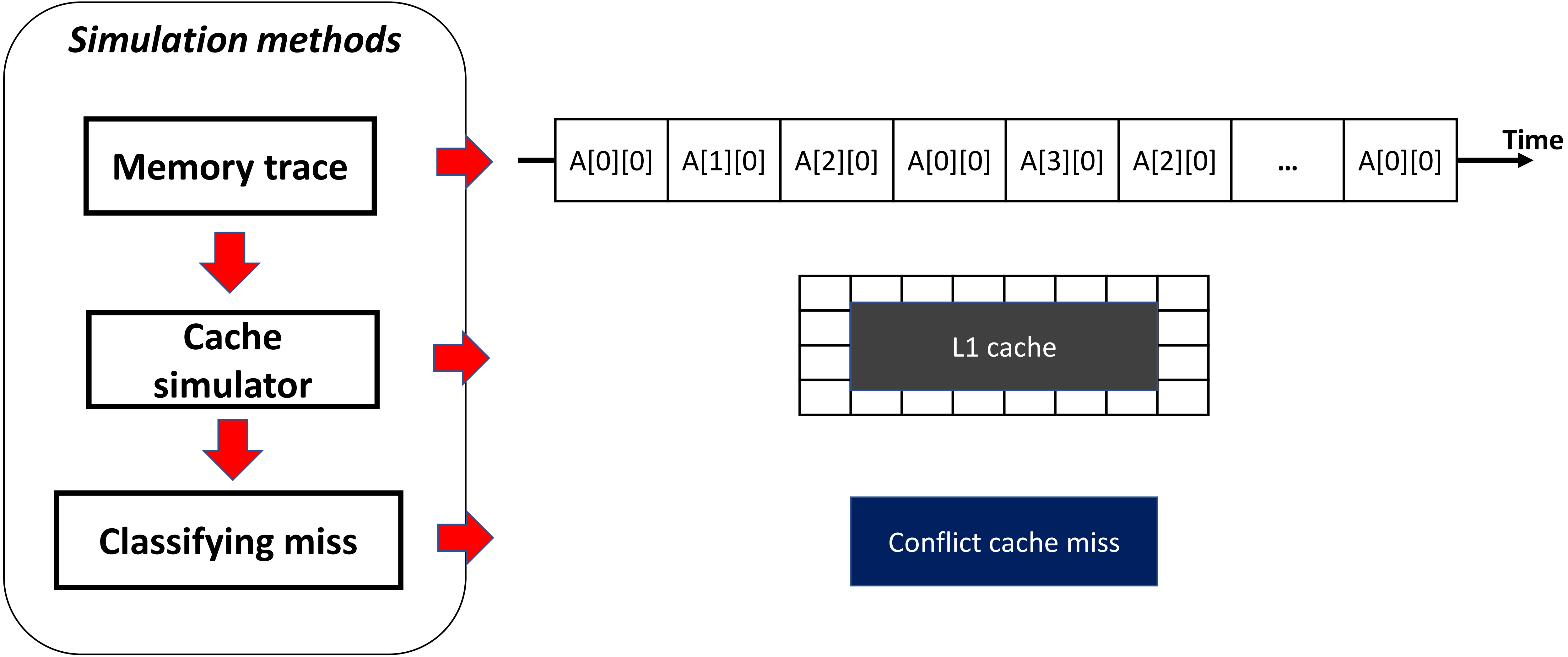
**Time**

L1 cache

Conflict cache miss

# Trace driven cache simulation

Overhead: average 38 times



***Simulation methods***

**Memory trace**

**Cache simulator**

**Classifying miss**

| A[0][0] | A[1][0] | A[2][0] | A[0][0] | A[3][0] | A[2][0] | … | A[0][0] |
|---|---|---|---|---|---|---|---|

**Time**

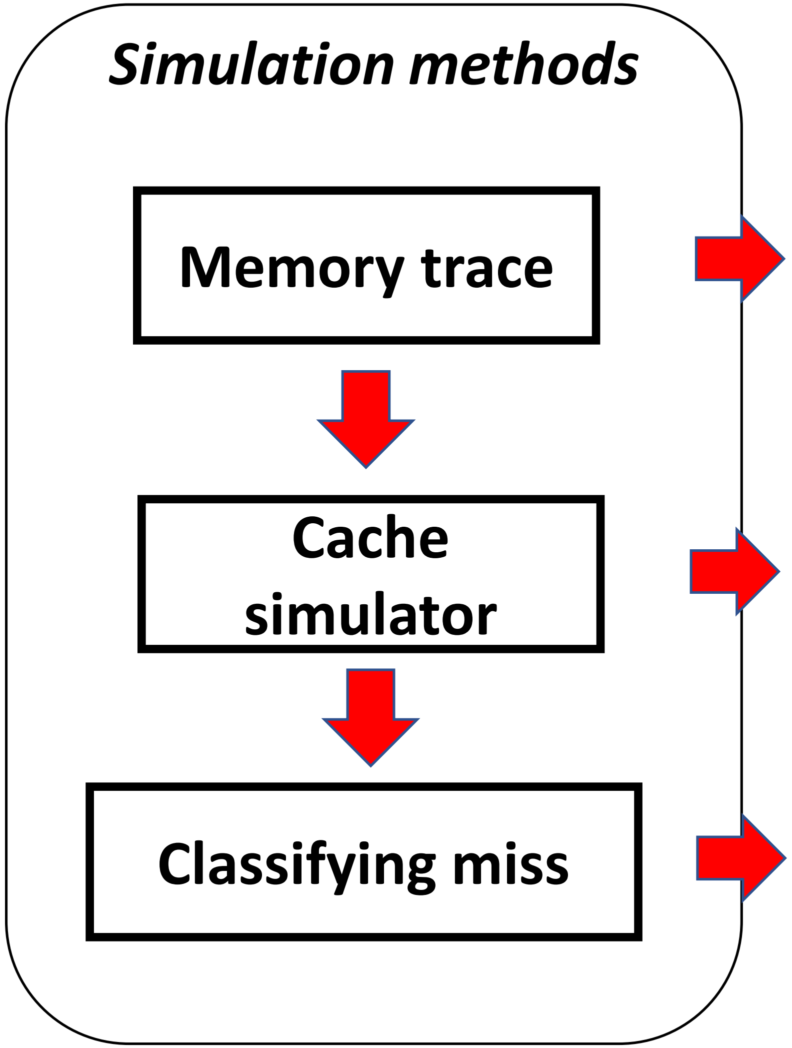
L1 cache

Conflict cache miss

# Trace driven cache simulation

Overhead: average 38 times

Xiang, Xiaoya, Chen Ding, Hao Luo, and Bin Bao. "HOTL: a higher order theory of locality." *ACM SIGPLAN Notices* 48, no. 4 (2013): 343-356.

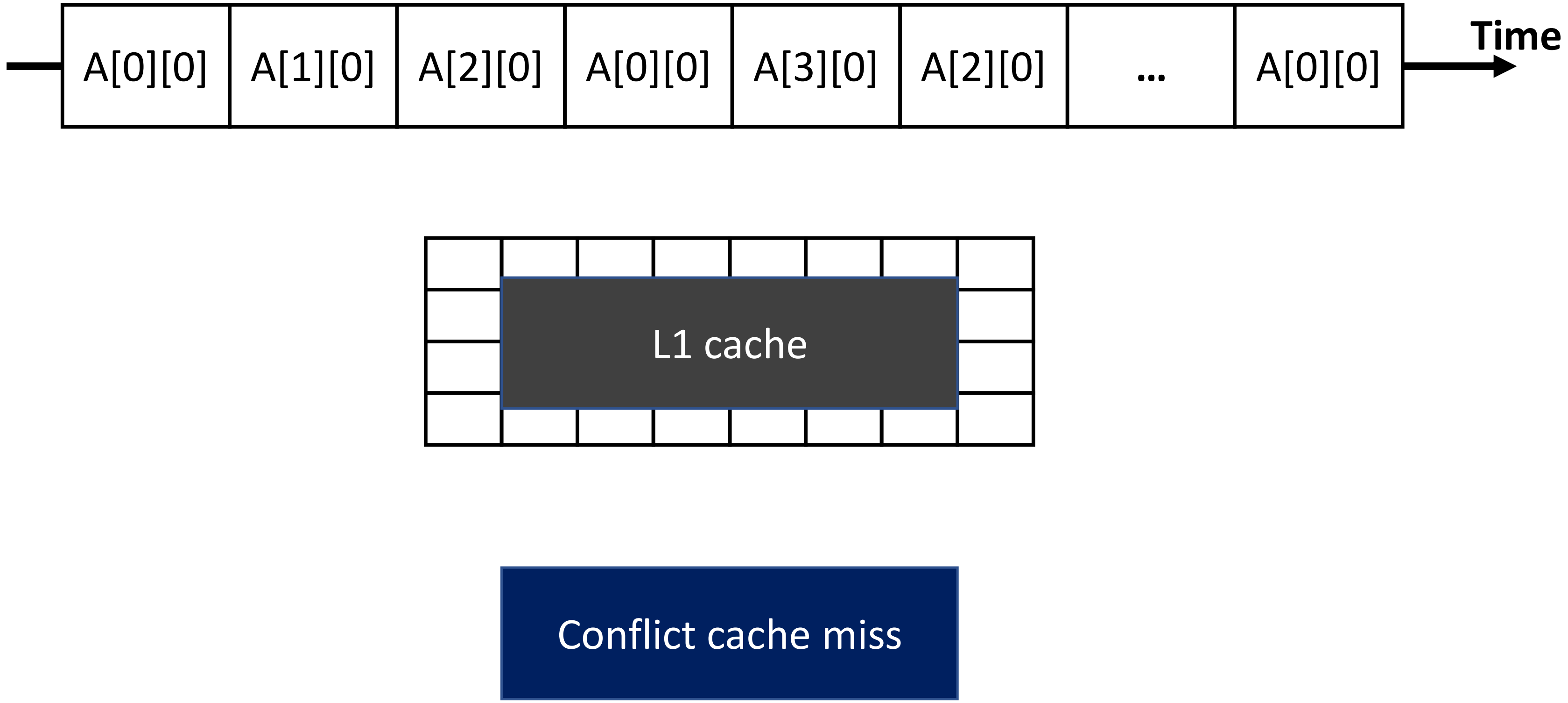


***Simulation methods***

**Memory trace**

**Cache simulator**

**Classifying miss**



A[0][0] | A[1][0] | A[2][0] | A[0][0] | A[3][0] | A[2][0] | … | A[0][0]

**Time**

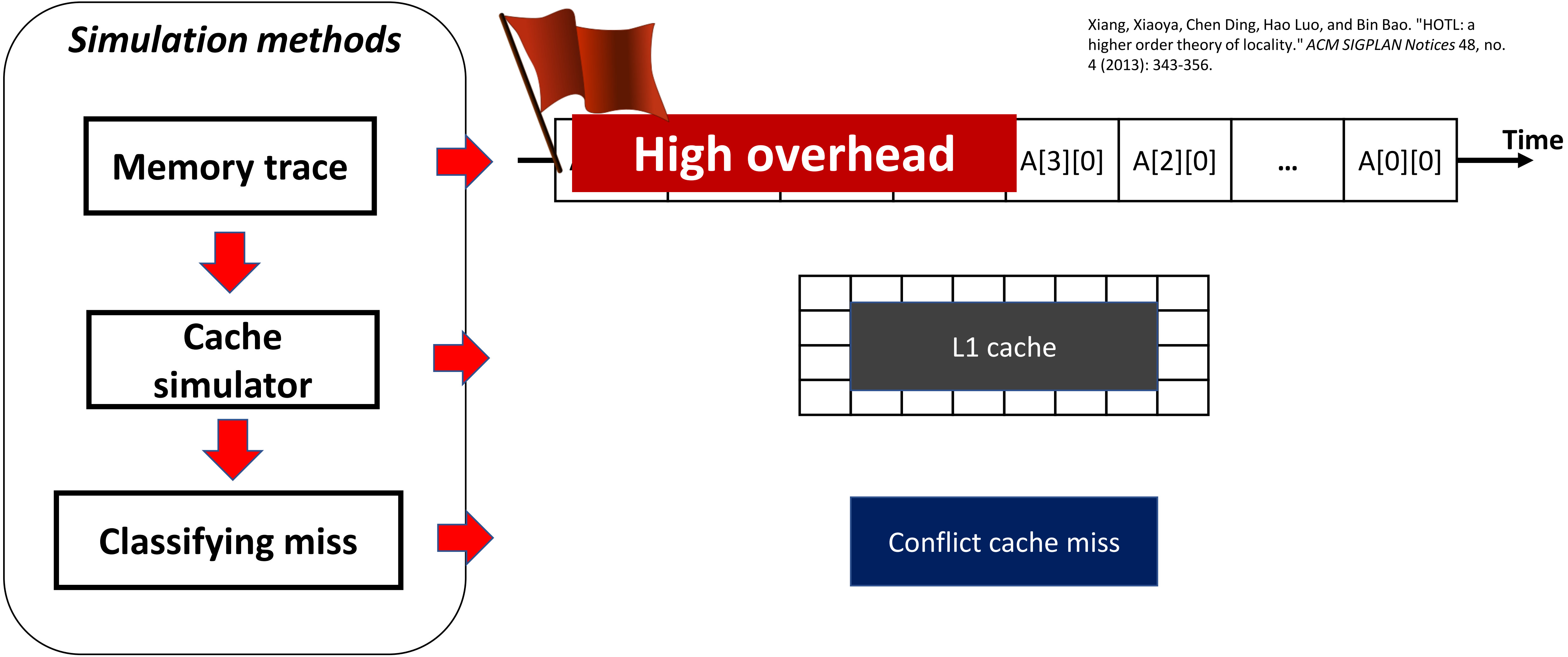
L1 cache

Conflict cache miss

# Trace driven cache simulation

Overhead: average 38 times

Xiang, Xiaoya, Chen Ding, Hao Luo, and Bin Bao. "HOTL: a higher order theory of locality." *ACM SIGPLAN Notices* 48, no. 4 (2013): 343-356.



***Simulation methods***

**Memory trace**

**Cache simulator**

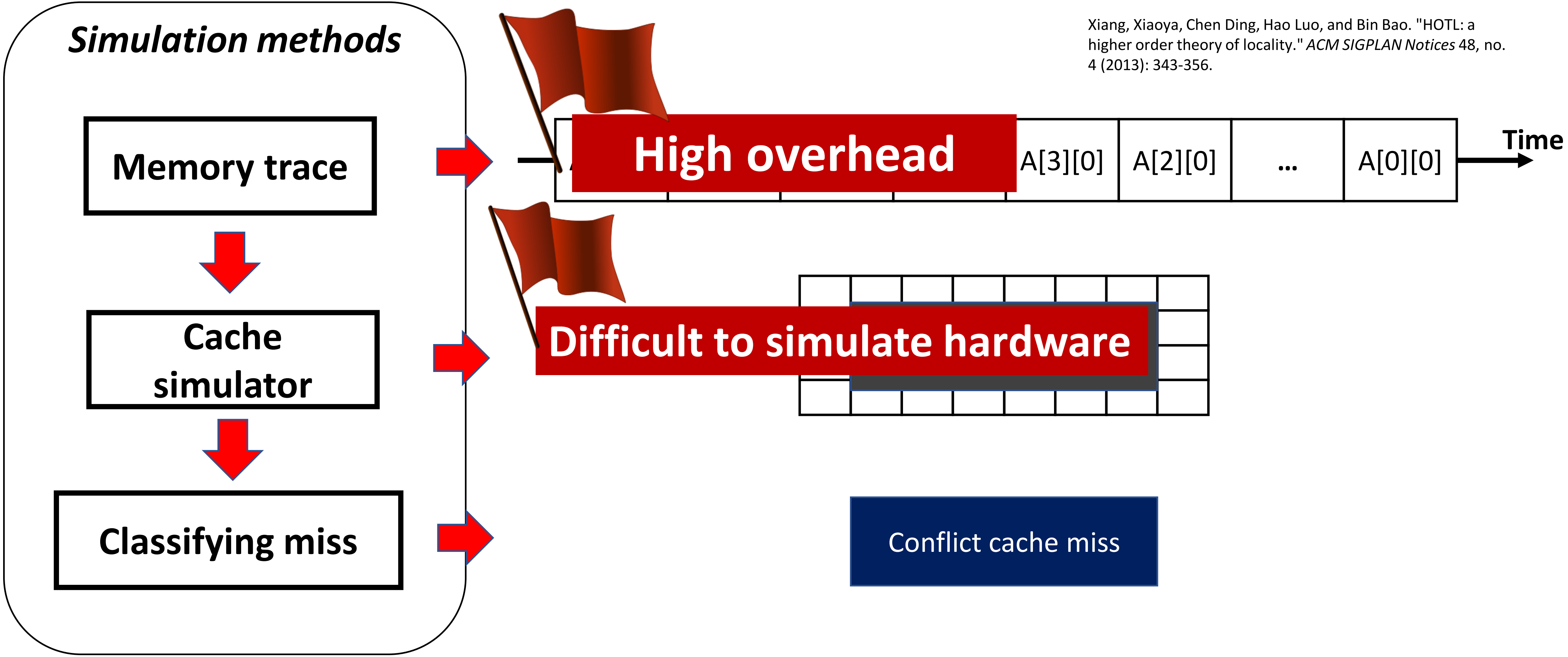
**Classifying miss**

**High overhead**

# Trace driven cache simulation

Overhead: average 38 times

Xiang, Xiaoya, Chen Ding, Hao Luo, and Bin Bao. "HOTL: a higher order theory of locality." *ACM SIGPLAN Notices* 48, no. 4 (2013): 343-356.



***Simulation methods***

Memory trace

Cache simulator

Classifying miss

High overhead

A[3][0] A[2][0] … A[0][0]

Time

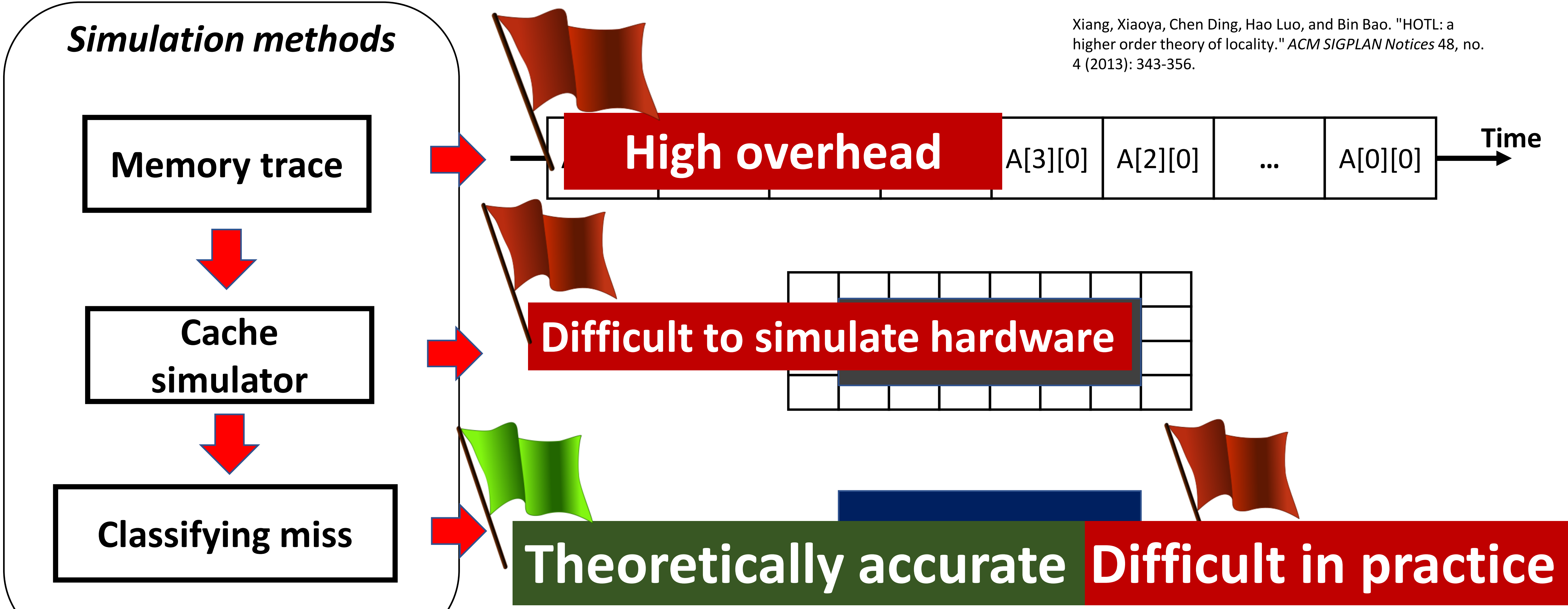
Difficult to simulate hardware

Conflict cache miss

# Trace driven cache simulation

Overhead: average 38 times

Xiang, Xiaoya, Chen Ding, Hao Luo, and Bin Bao. "HOTL: a higher order theory of locality." *ACM SIGPLAN Notices* 48, no. 4 (2013): 343-356.



***Simulation methods***

**Memory trace**

**Cache simulator**

**Classifying miss**

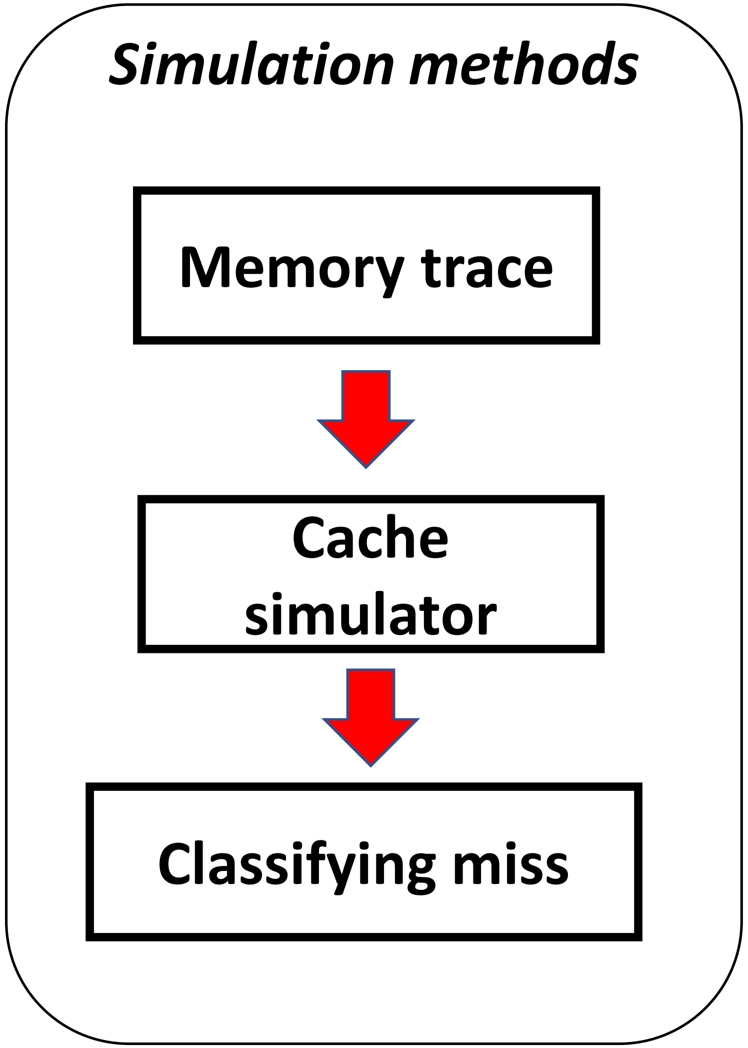
**High overhead**

**Difficult to simulate hardware**

**Theoretically accurate**

**Difficult in practice**

# A practical low overhead solution



*Simulation methods*

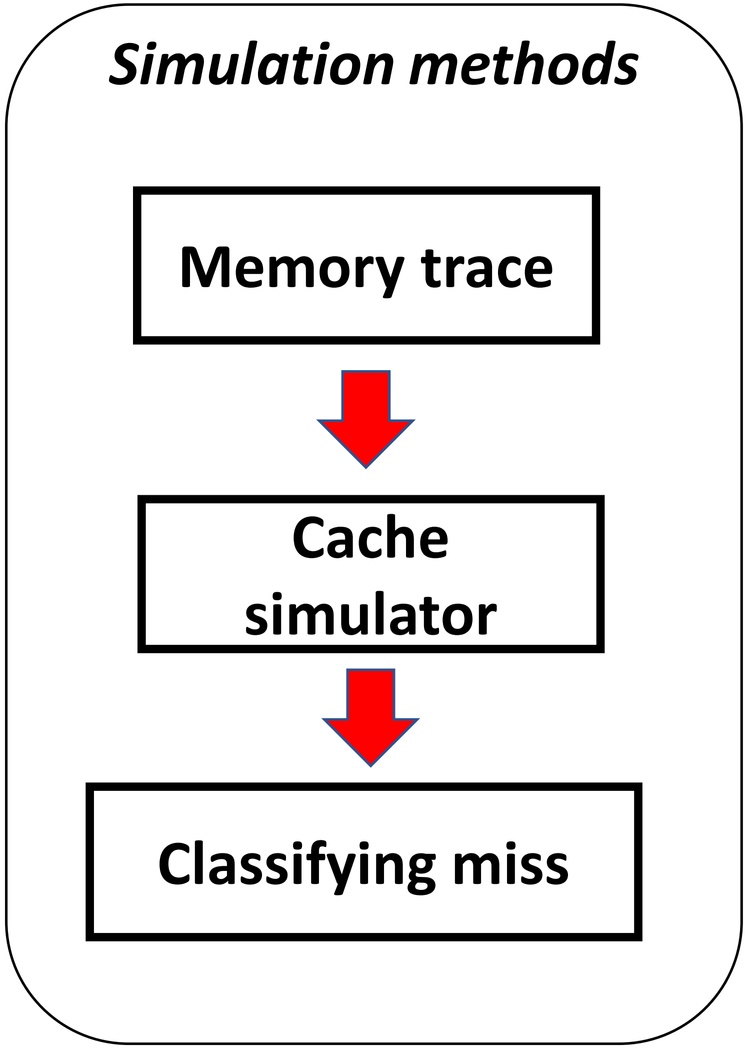
**Memory trace**

**Cache simulator**

**Classifying miss**

# A practical low overhead solution

***CCProf***



***Simulation methods***

**Memory trace**

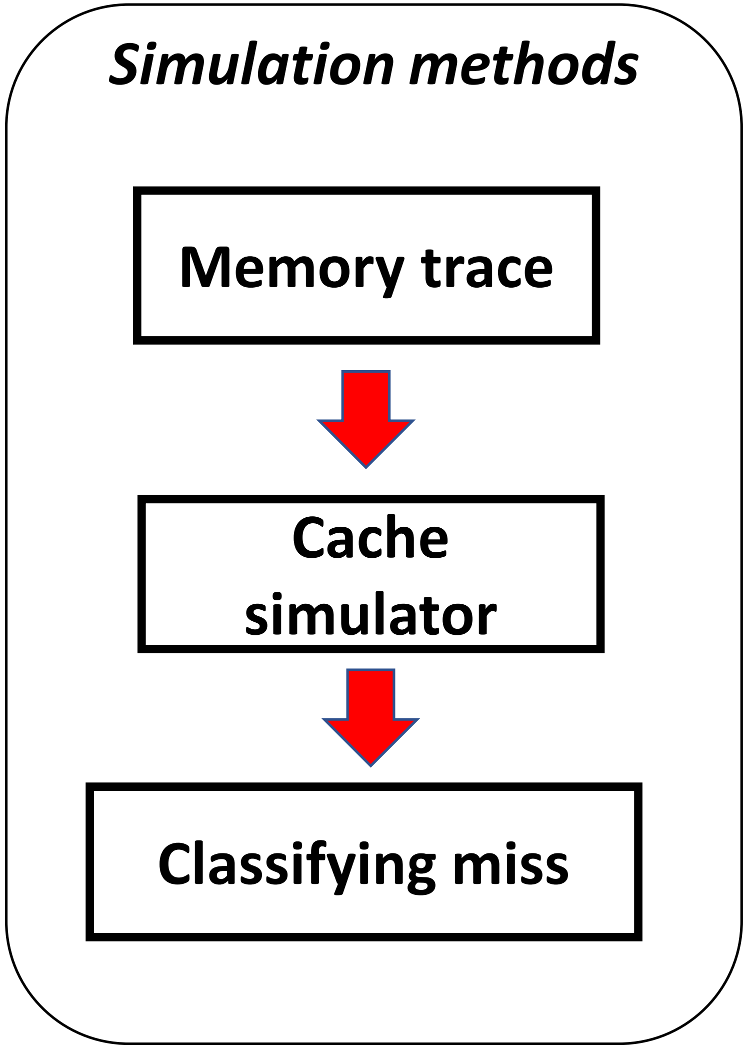
**Cache simulator**

**Classifying miss**

# A practical low overhead solution

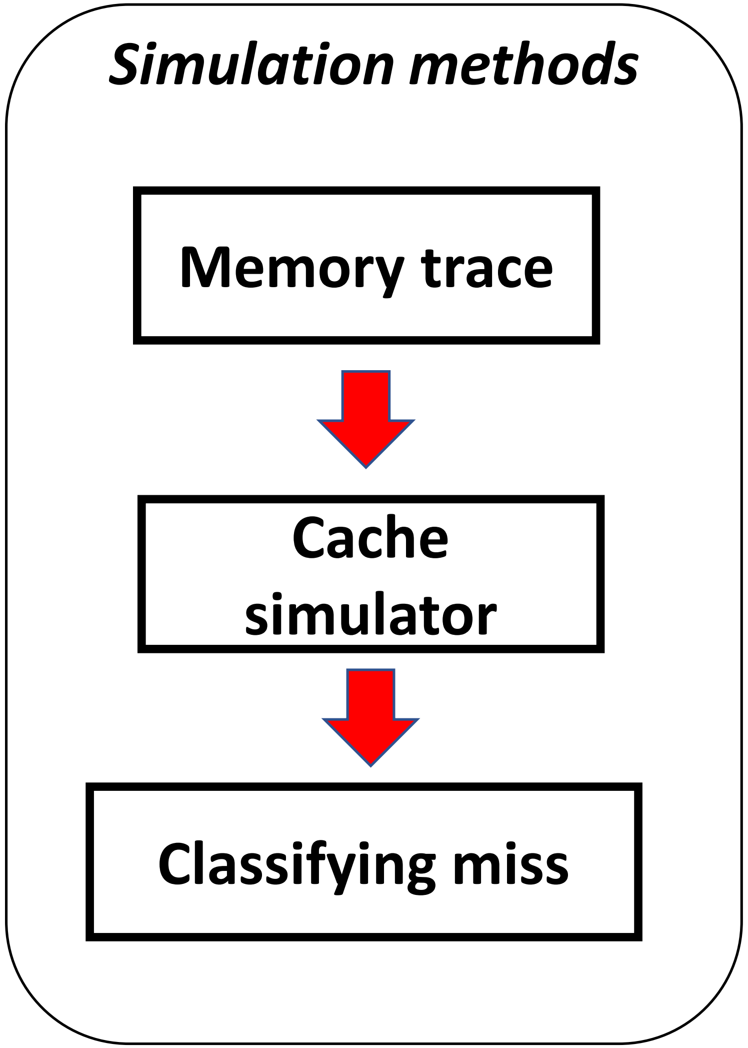
**CCProf**

*Simulation methods*



*Measurement methods*

# A practical low overhead solution

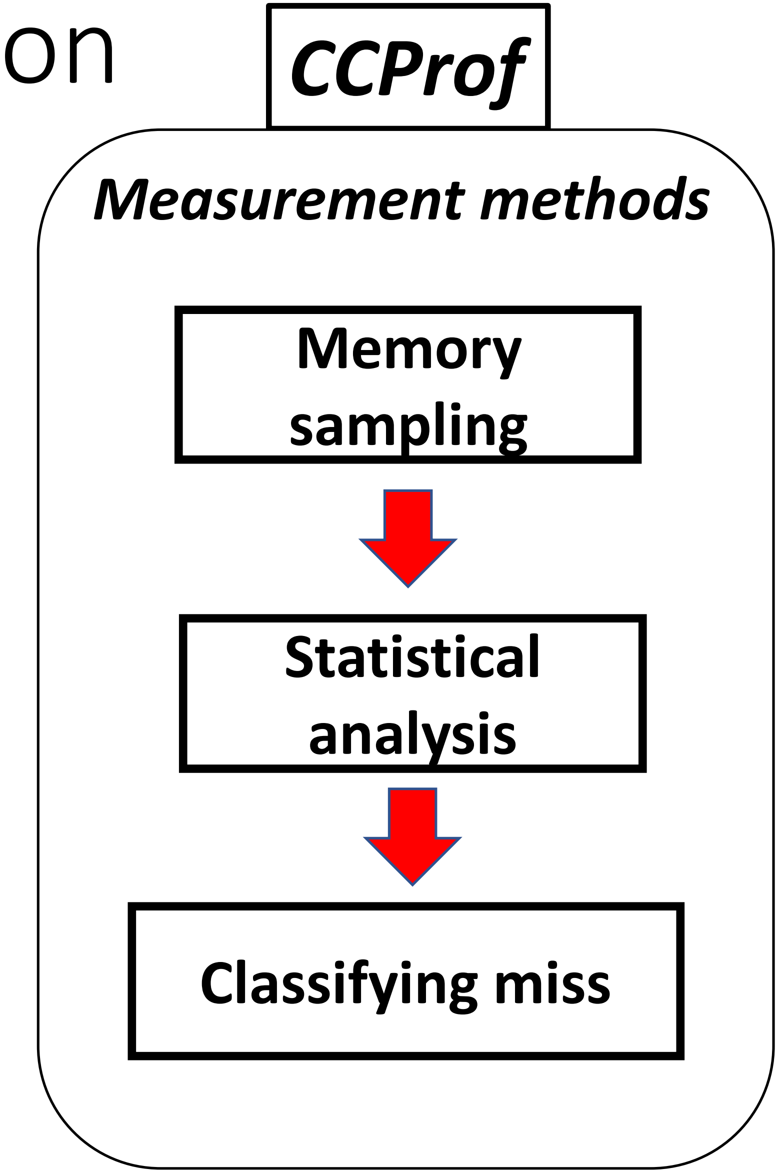


***Simulation methods***

**Memory trace**

**Cache simulator**

**Classifying miss**



***CCProf***

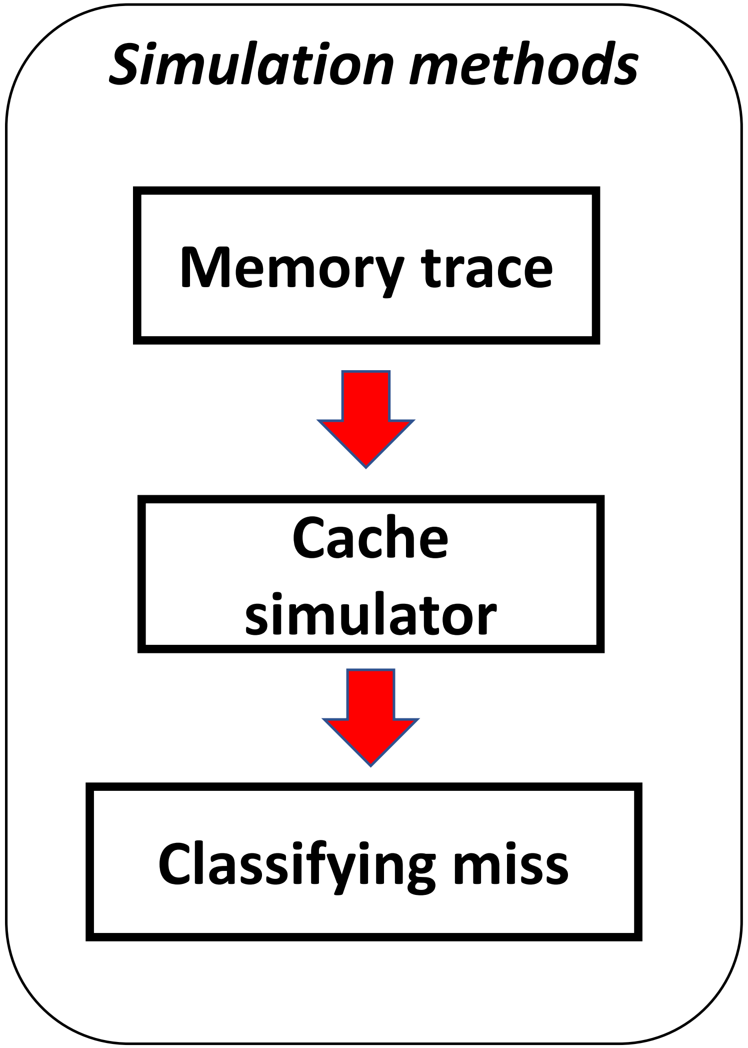
***Measurement methods***

**Memory sampling**

**Statistical analysis**

**Classifying miss**

# A practical low overhead solution



***Simulation methods***

**Memory trace**

**Cache simulator**

**Classifying miss**

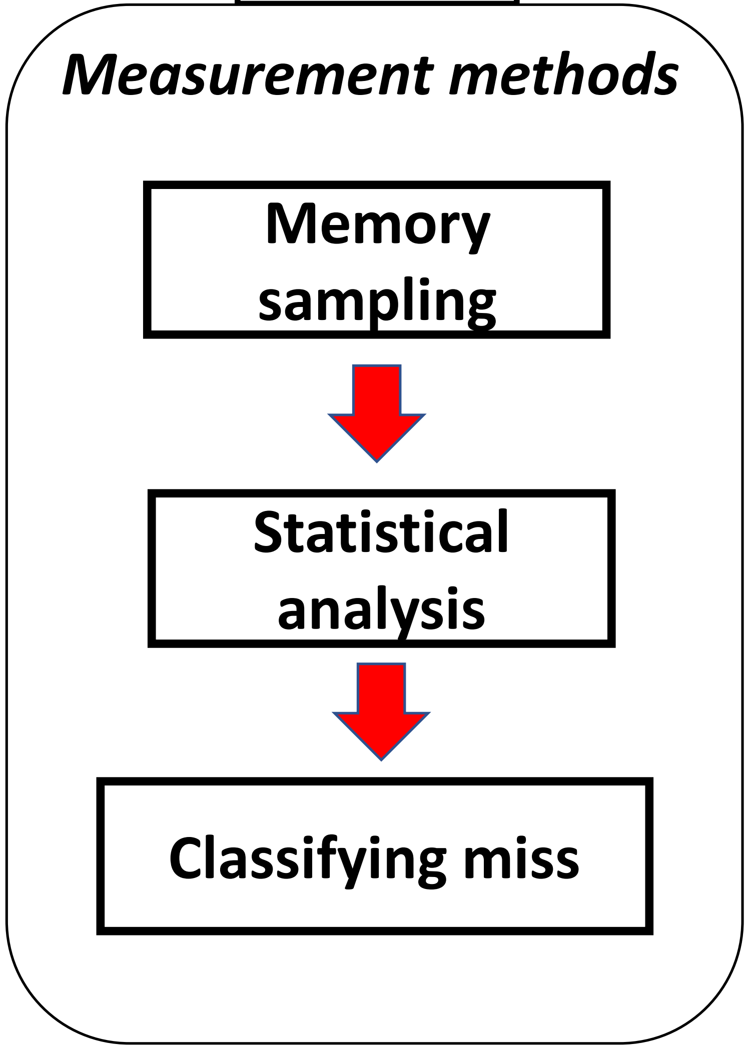
**Overhead**

**>>**

**Accuracy**

**~**

***CCProf***



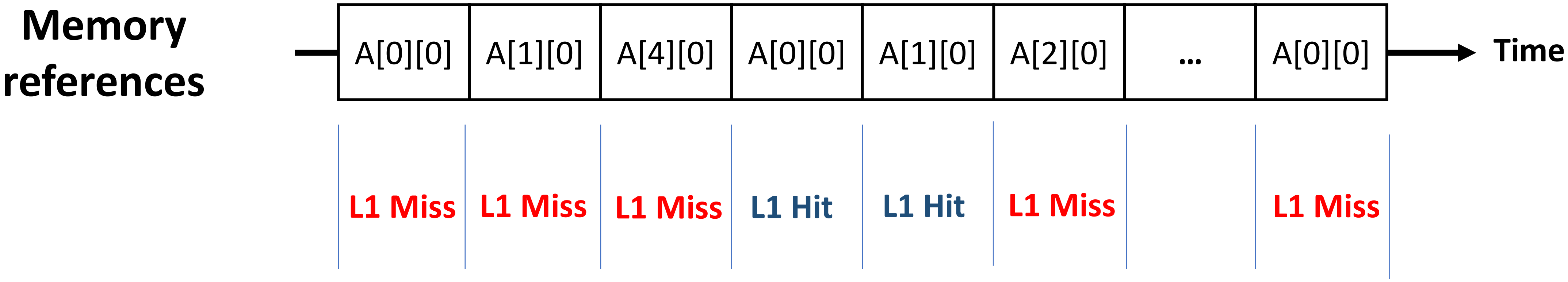
***Measurement methods***

**Memory sampling**

**Statistical analysis**

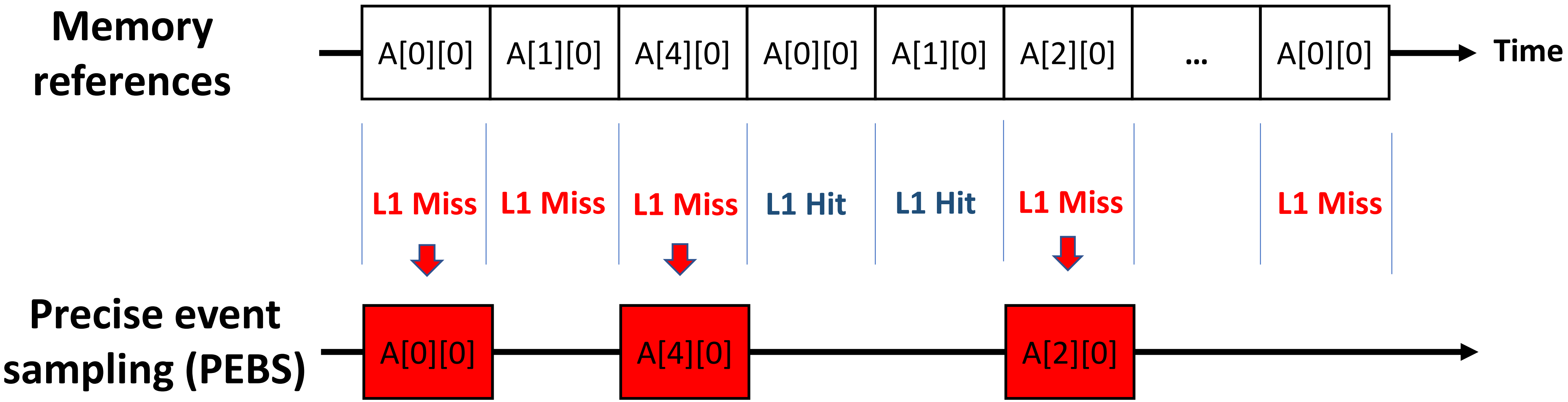
**Classifying miss**

# Hardware-based address sampling (Cont.)



| Memory references | A[0][0] | A[1][0] | A[4][0] | A[0][0] | A[1][0] | A[2][0] | … | A[0][0] | Time |
|---|---|---|---|---|---|---|---|---|---|
| | L1 Miss | L1 Miss | L1 Miss | L1 Hit | L1 Hit | L1 Miss | | L1 Miss | |

# Hardware-based address sampling (Cont.)



**Memory references**

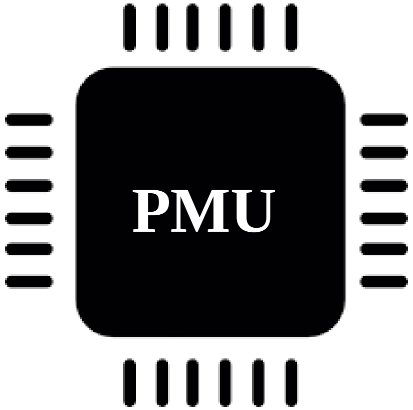
A[0][0] | A[1][0] | A[4][0] | A[0][0] | A[1][0] | A[2][0] | … | A[0][0]

**Time**

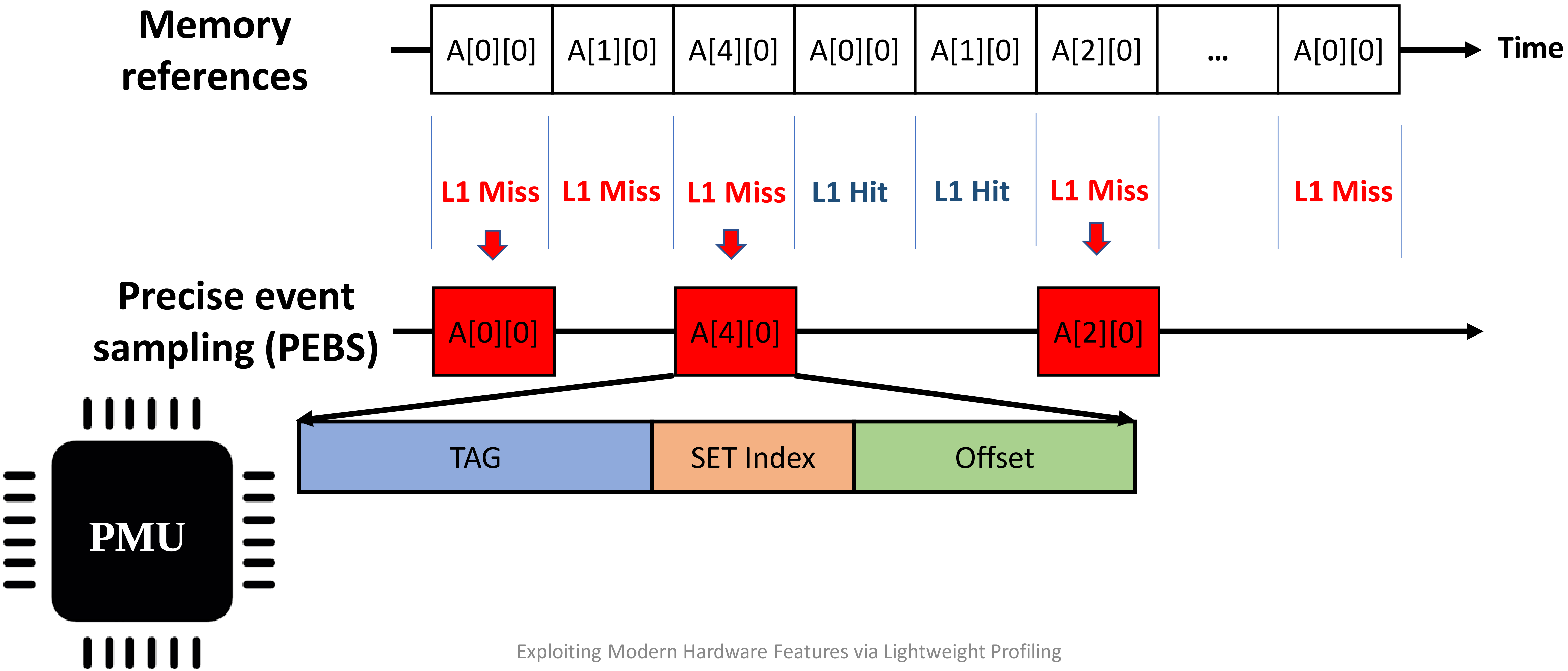
L1 Miss | L1 Miss | L1 Miss | L1 Hit | L1 Hit | L1 Miss | | L1 Miss

**Precise event sampling (PEBS)**

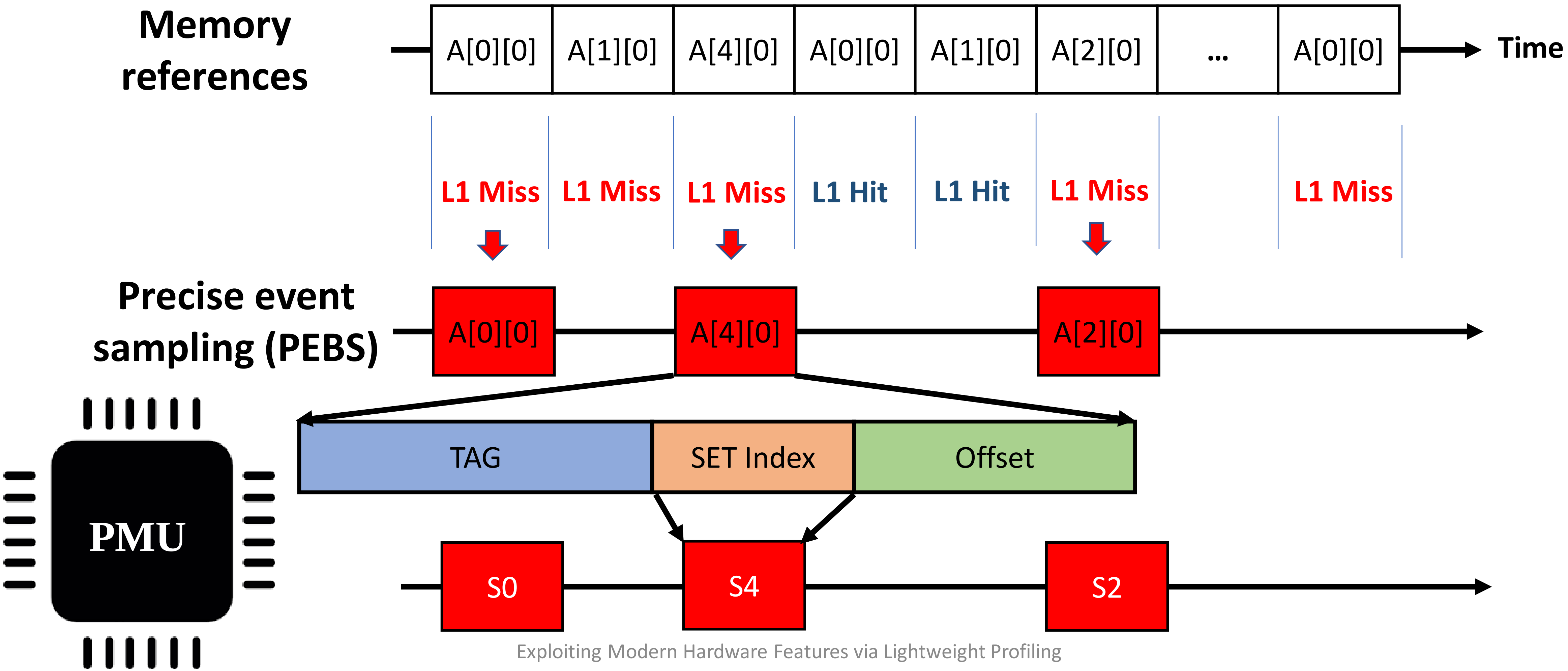
A[0][0] | A[4][0] | A[2][0]



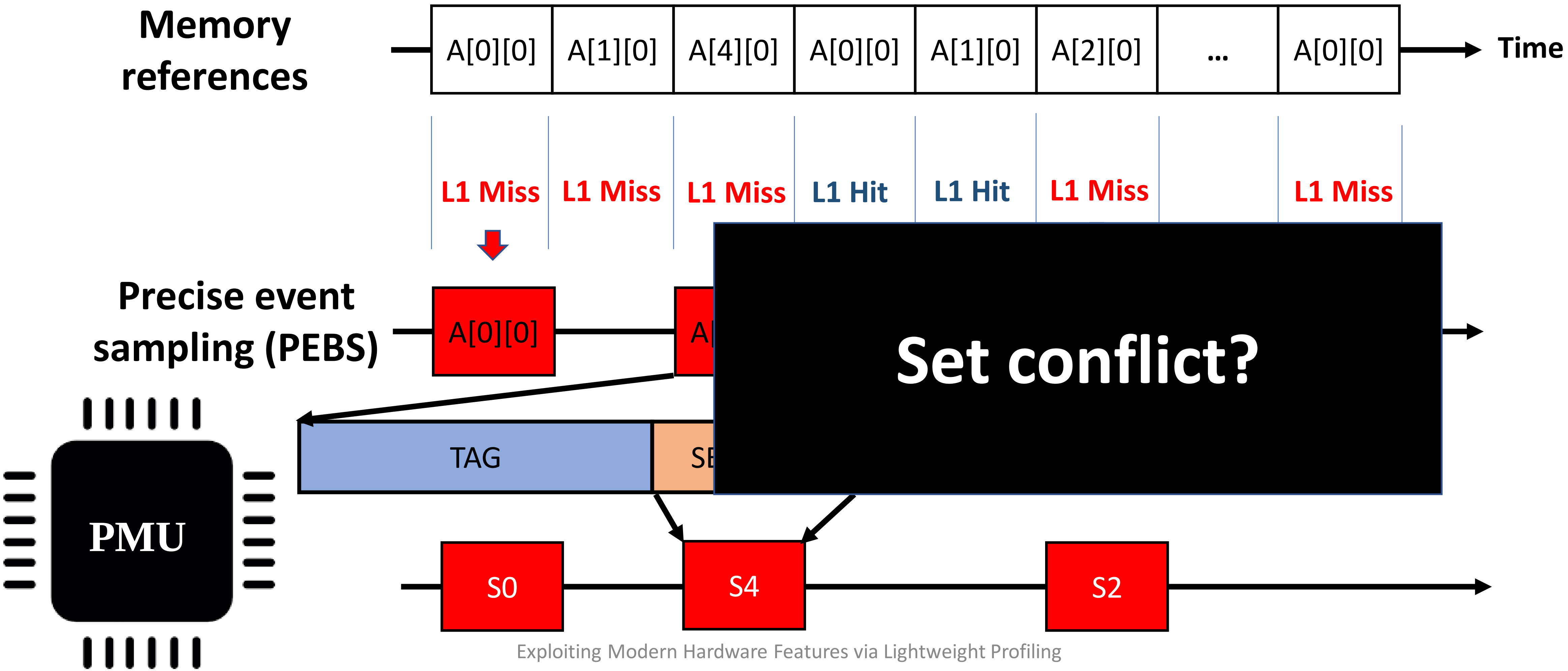
# Hardware-based address sampling (Cont.)



# Hardware-based address sampling (Cont.)

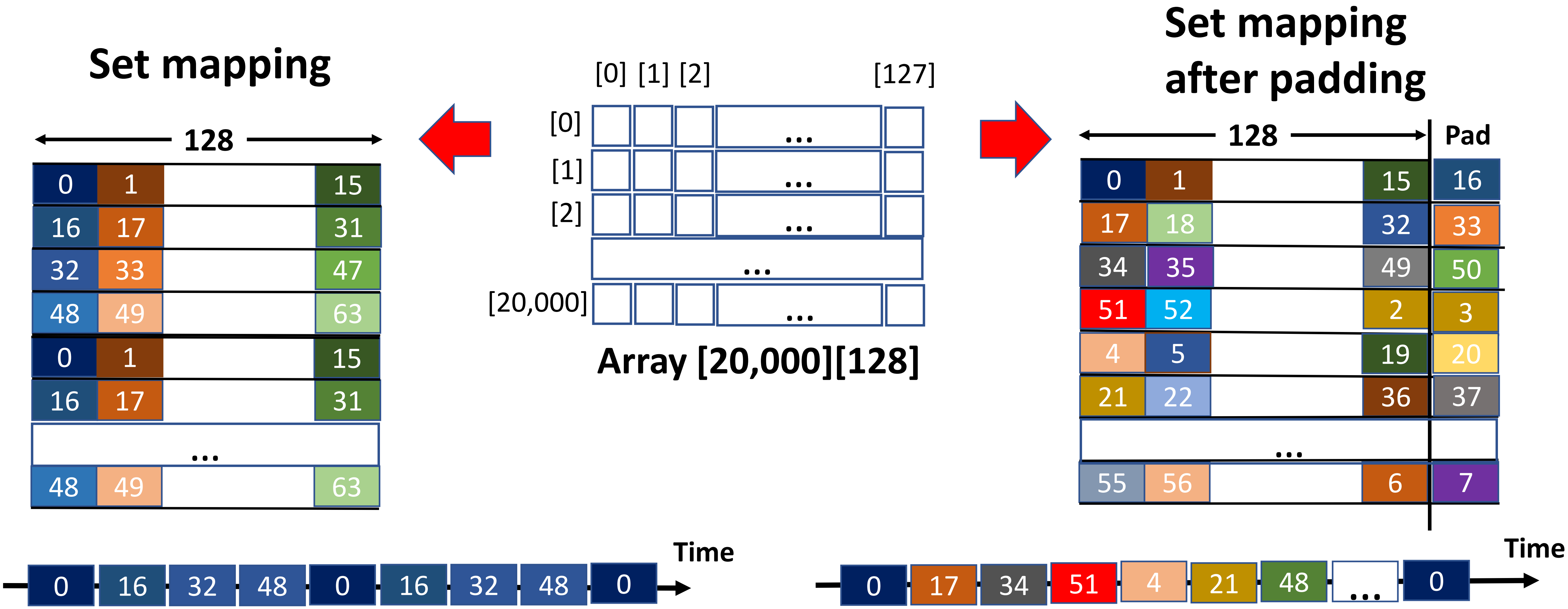


# Hardware-based address sampling (Cont.)



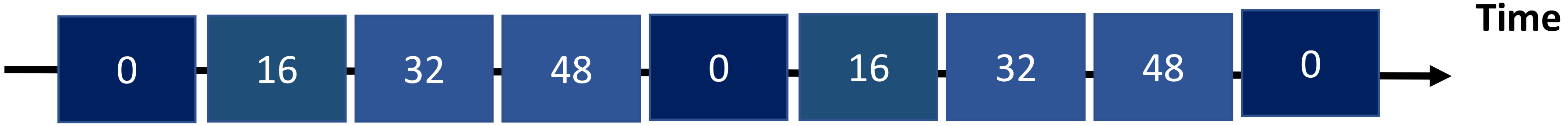
**Set conflict?**

# *Observation:* ***temporal pattern*** *of* ***conflict***

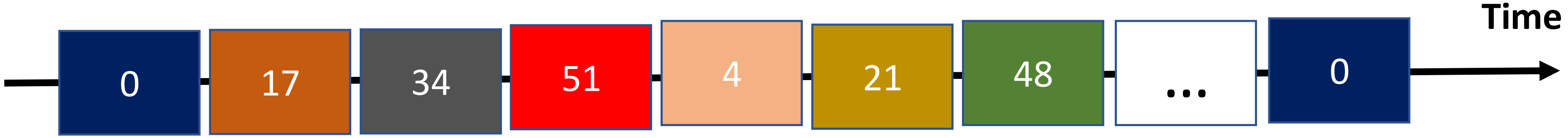


# *Observation*: ***temporal pattern*** of ***conflict*** (cont.)

**Conflict**

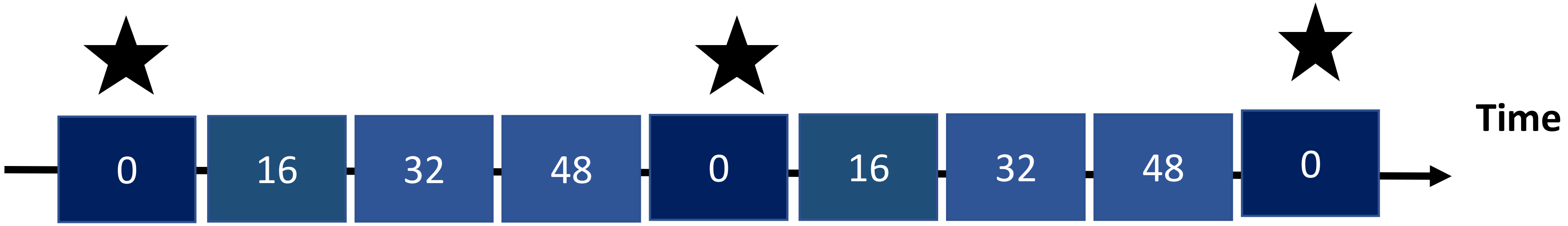


**No conflict**

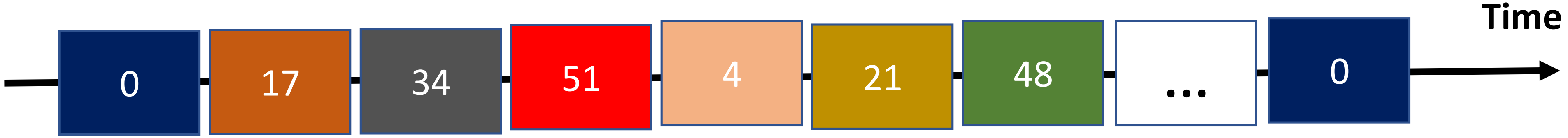


# *Observation*: ***temporal pattern*** of ***conflict*** (cont.)

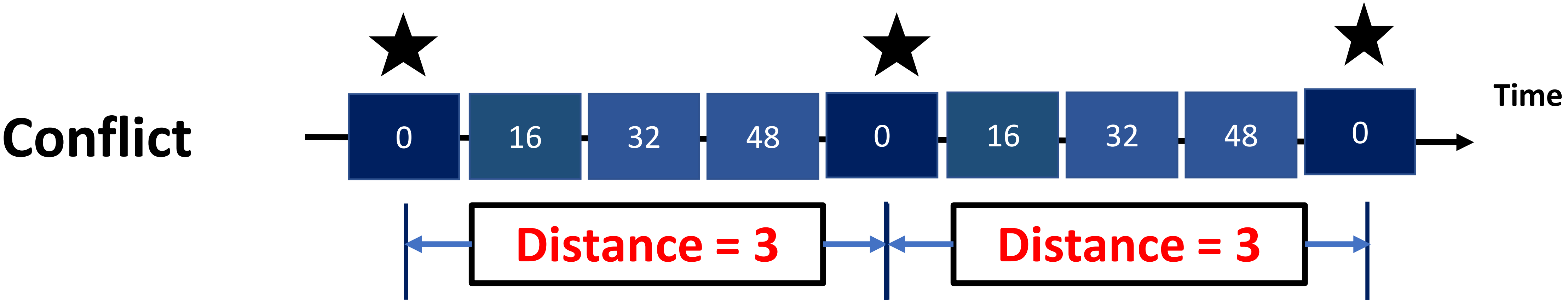
**Conflict**

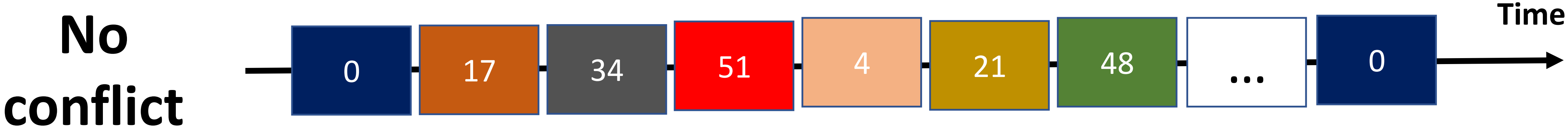


**No conflict**

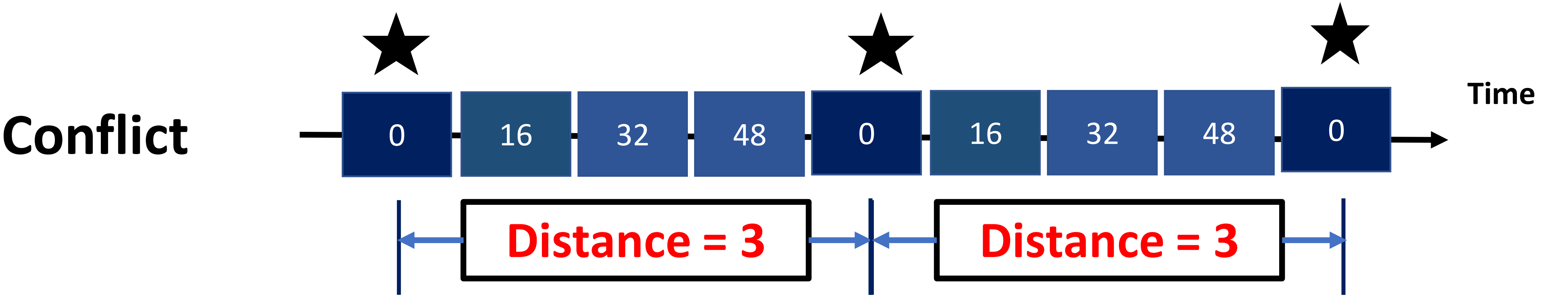


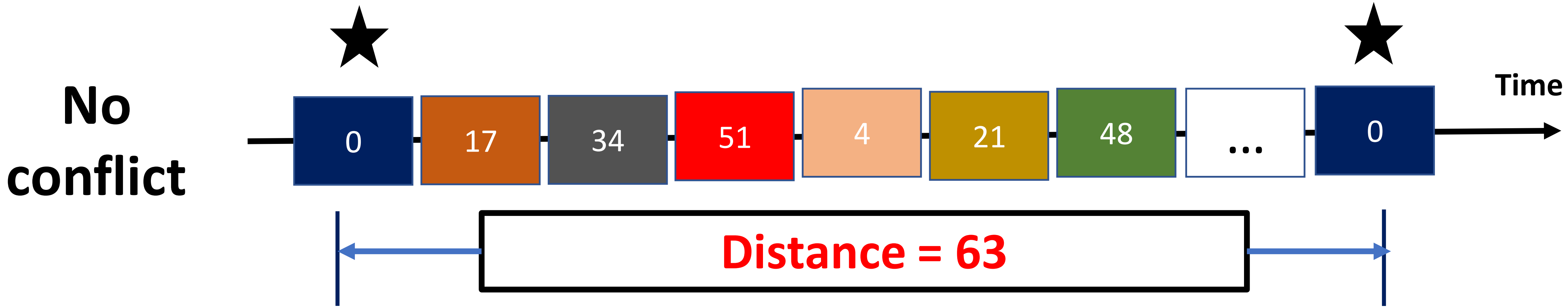
# *Observation*: ***temporal pattern*** of ***conflict*** (cont.)





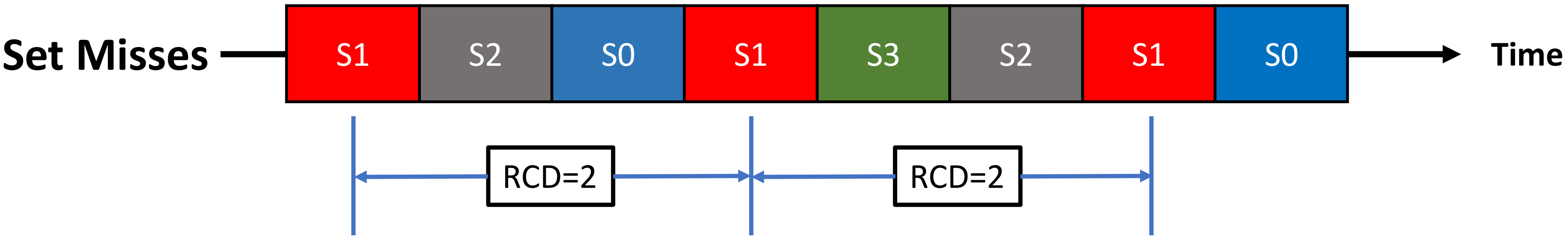
# *Observation*: ***temporal pattern* of *conflict*** (cont.)





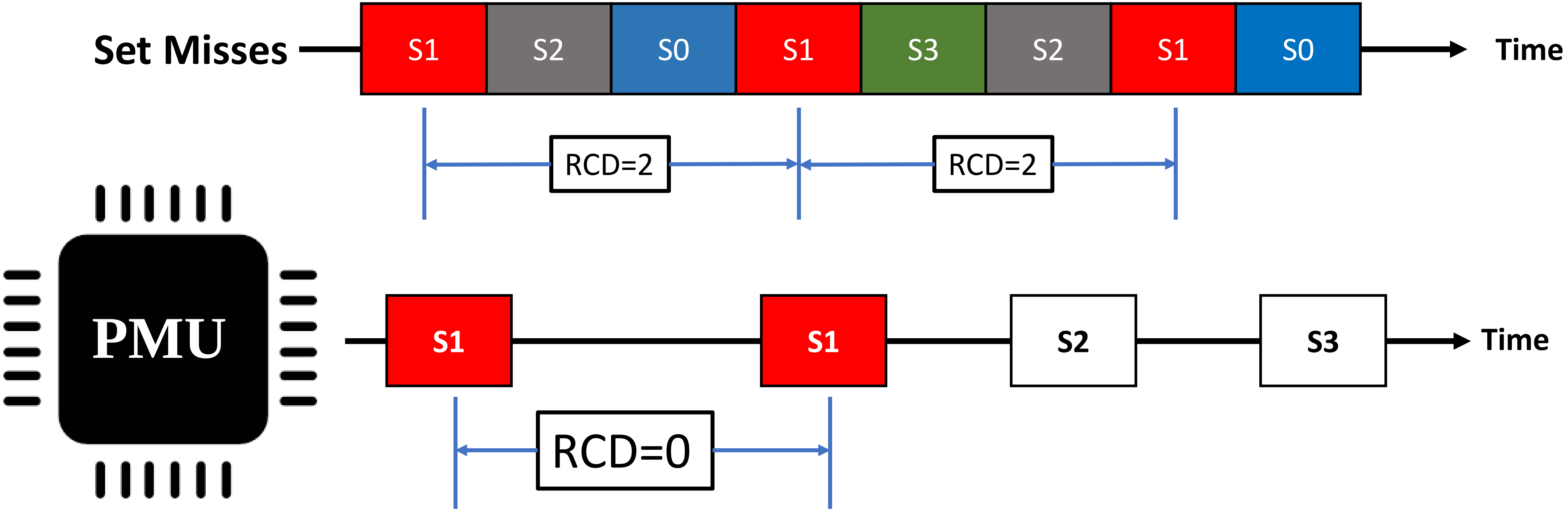
# Re-conflict Distance (RCD)

- Number of ***<u>cache misses</u>*** in other cache sets between two consecutive misses in one particular set



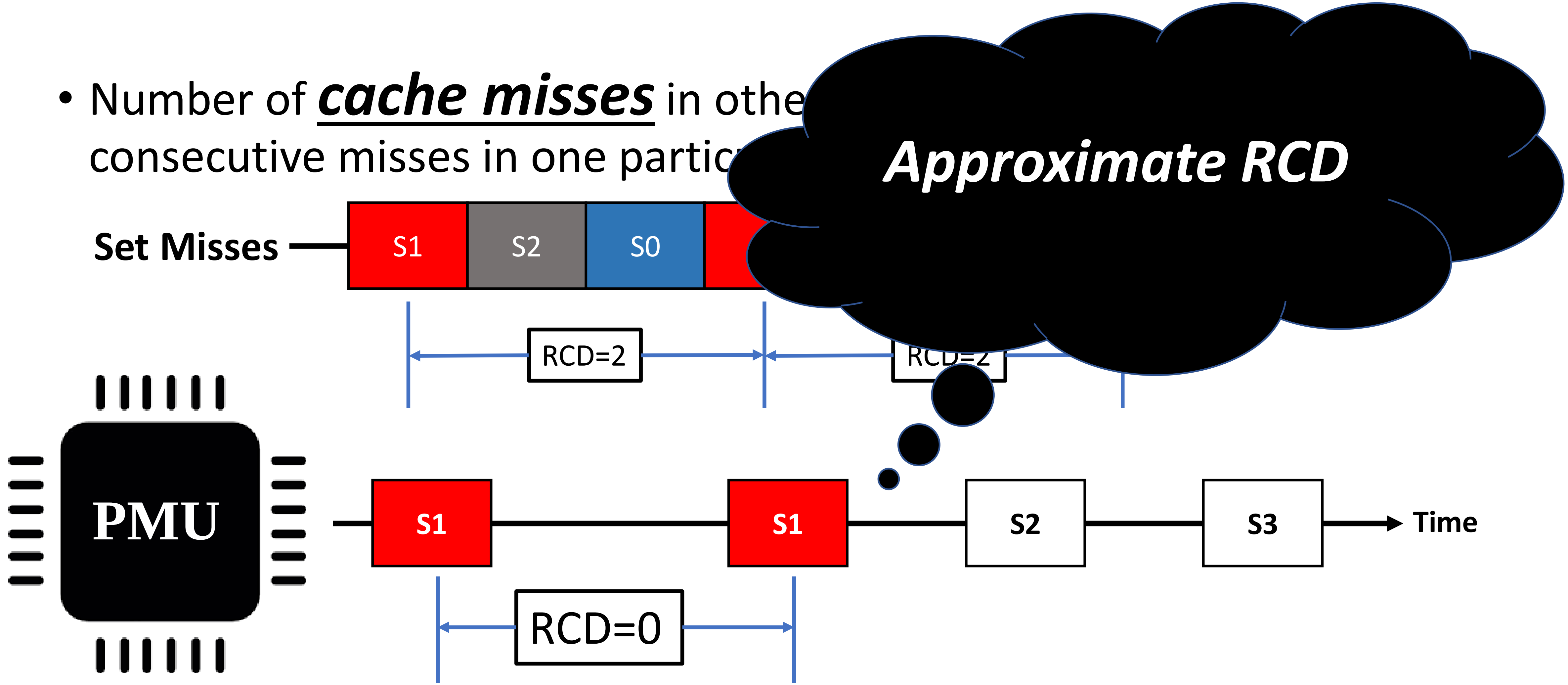
# Re-conflict Distance (RCD)

- Number of ***cache misses*** in other cache sets between two consecutive misses in one particular set



# Re-conflict Distance (RCD)

- Number of ***cache misses*** in othe consecutive misses in one partic

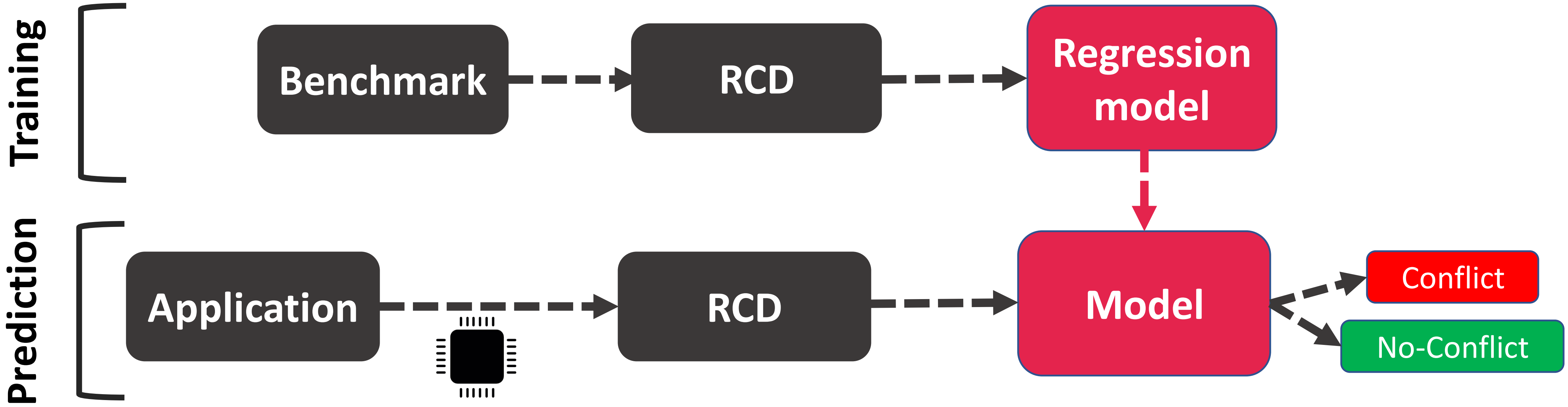


# RCD and it’s contribution

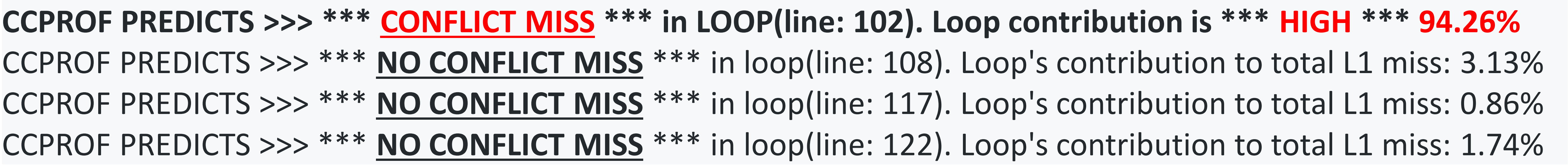
| RCD | Count | Is conflict? |
|---|---|---|
| Short | Large | Yes |
| Short | Small | No |
| Long | ~ | No |

# RCD and it's contribution

| RCD | Count | Is conflict? |
|---|---|---|
| Short | Large | Yes |
| Short | Small | No |
| Long | ~ | No |

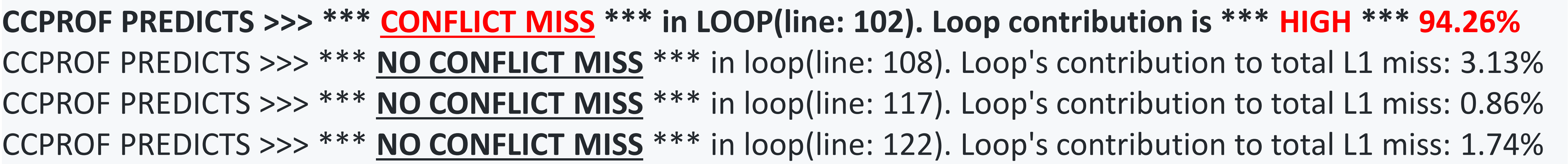


# Case Study: PolyBench/C -ADI

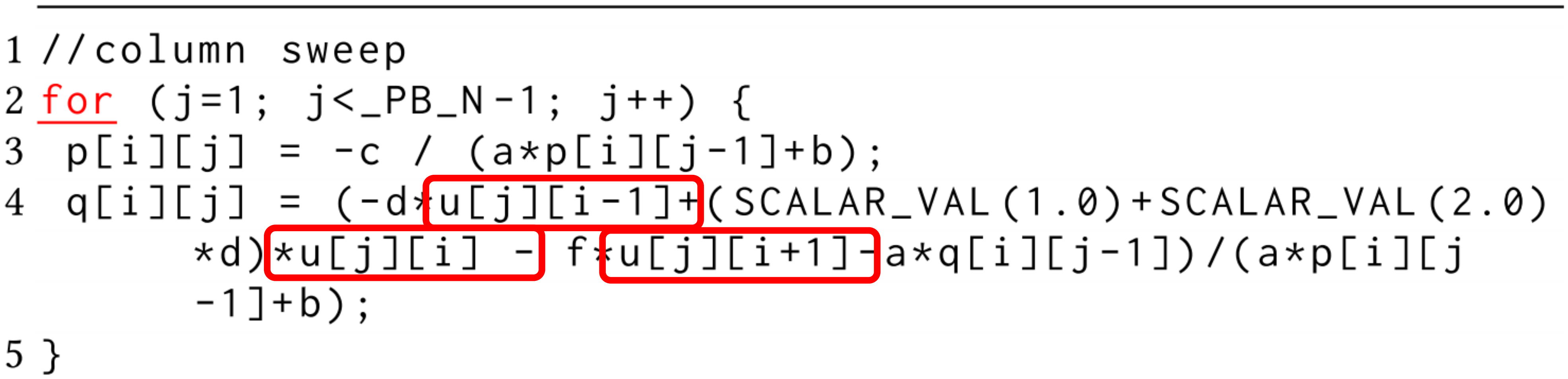


```
CCPROF PREDICTS >>> *** CONFLICT MISS *** in LOOP(line: 102). Loop contribution is *** HIGH *** 94.26%
CCPROF PREDICTS >>> *** NO CONFLICT MISS *** in loop(line: 108). Loop's contribution to total L1 miss: 3.13%
CCPROF PREDICTS >>> *** NO CONFLICT MISS *** in loop(line: 117). Loop's contribution to total L1 miss: 0.86%
CCPROF PREDICTS >>> *** NO CONFLICT MISS *** in loop(line: 122). Loop's contribution to total L1 miss: 1.74%
```

# Case Study: PolyBench/C -ADI

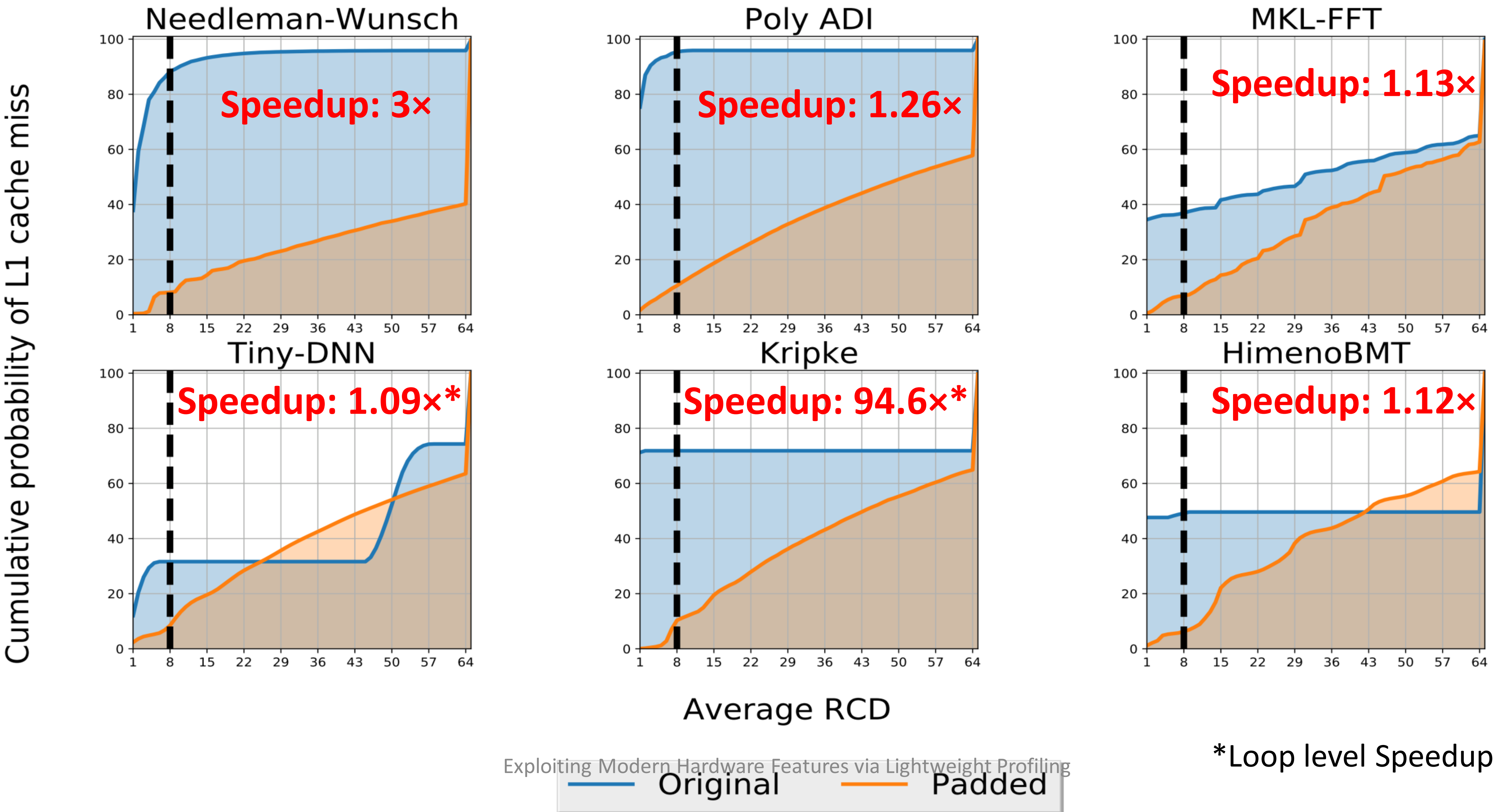


```
CCPROF PREDICTS >>> *** CONFLICT MISS *** in LOOP(line: 102). Loop contribution is *** HIGH *** 94.26%
CCPROF PREDICTS >>> *** NO CONFLICT MISS *** in loop(line: 108). Loop's contribution to total L1 miss: 3.13%
CCPROF PREDICTS >>> *** NO CONFLICT MISS *** in loop(line: 117). Loop's contribution to total L1 miss: 0.86%
CCPROF PREDICTS >>> *** NO CONFLICT MISS *** in loop(line: 122). Loop's contribution to total L1 miss: 1.74%
```



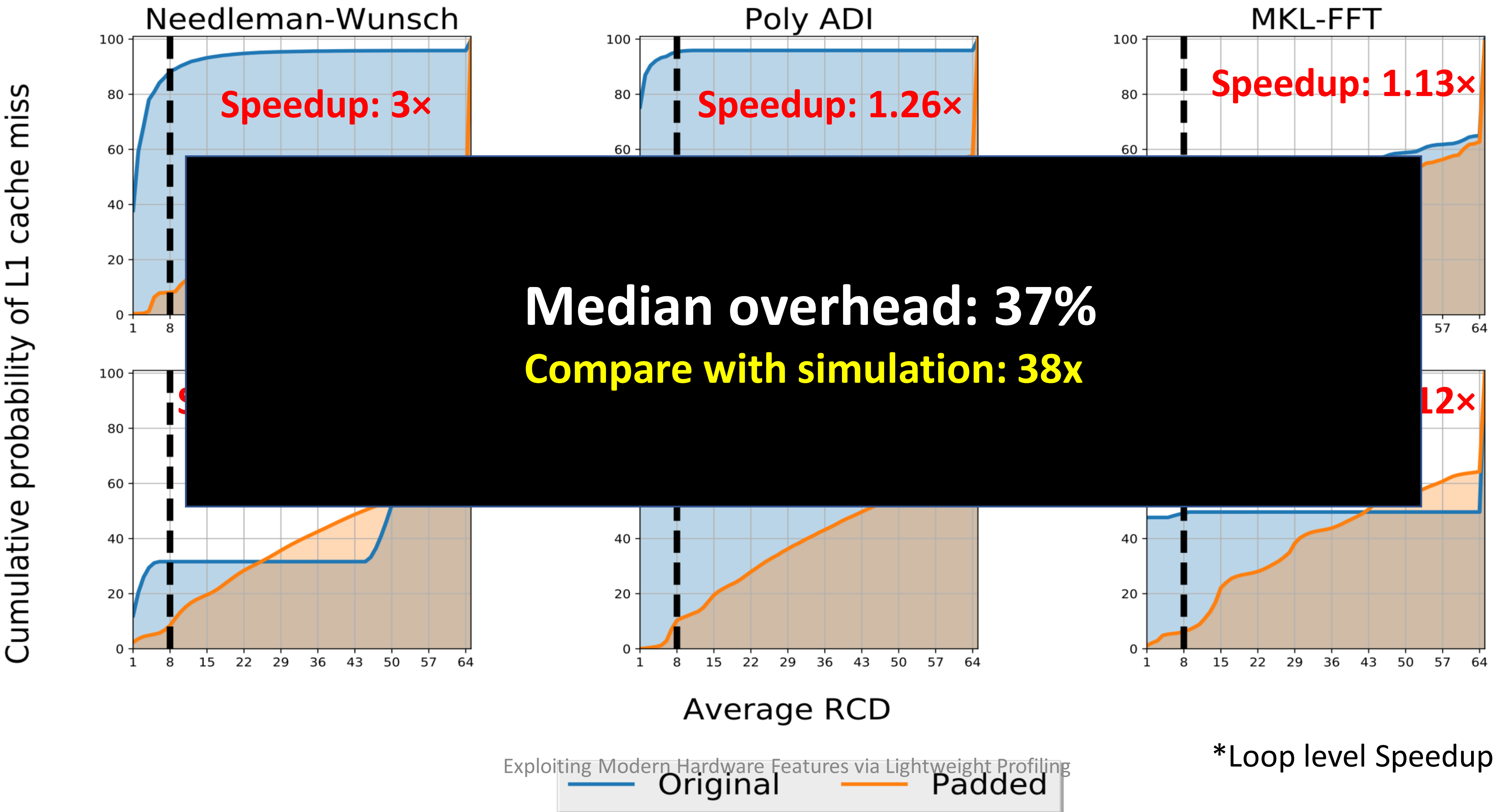
```
//column sweep
for (j=1; j<_PB_N-1; j++) {
 p[i][j] = -c / (a*p[i][j-1]+b);
 q[i][j] = (-d*u[j][i-1]+(SCALAR_VAL(1.0)+SCALAR_VAL(2.0)
       *d)*u[j][i] - f*u[j][i+1]-a*q[i][j-1])/(a*p[i][j
       -1]+b);
}
```

# RCD – *before* and *after* optimization



*Loop level Speedup

# RCD – *before* and *after* optimization



**Median overhead: 37%**

**Compare with simulation: 38x**

*Loop level Speedup

# Outline

- ✓ Lightweight profiling
- ✓ SMT-aware optimization
- ✓ Detection of cache conflicts
- *Guiding data-structure layout transformation*

# StructSlim: A lightweight profiler to guide structure splitting

[CGO – 2016]

**Probir Roy** , *Xu Liu*

# LWPTool: A Lightweight Profiler to Guide Data Layout Optimization

[TPDS – 2018]

Chao Yu, **Probir Roy**, Yuebin Bai, Hailong Yang, *Xu Liu*

# StructSlim: A lightweight profiler to guide structure splitting

[CGO – 2016]

**Probir Roy** , *Xu Liu*

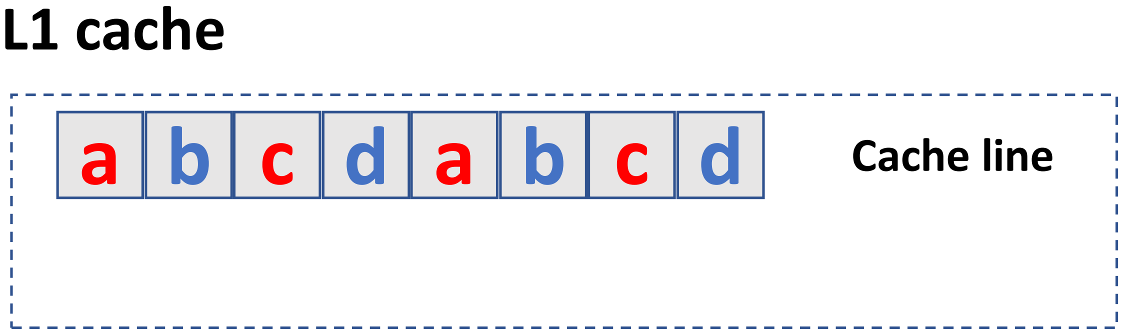
# LWPTool: A Lightweight Profiler to Guide Data Layout Optimization

[TPDS – 2018]

Chao Yu, **Probir Roy**, Yuebin Bai, Hailong Yang, *Xu Liu*

# Inefficient data-structure

```
struct type {int a; int b; int c; int d;};
struct type Arr[N];
for (i = 0; i < N; i++)
        B[i] = Arr[i].a + Arr[i].c;
```

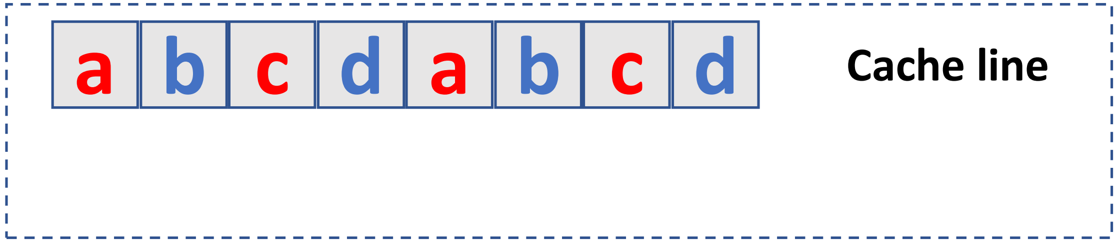


# Inefficient data-structure

```
struct type {int a; int b; int c; int d;};
struct type Arr[N];
for (i = 0; i < N; i++)
        B[i] = Arr[i].a + Arr[i].c;
```

L1 cache

Utilization = 50%

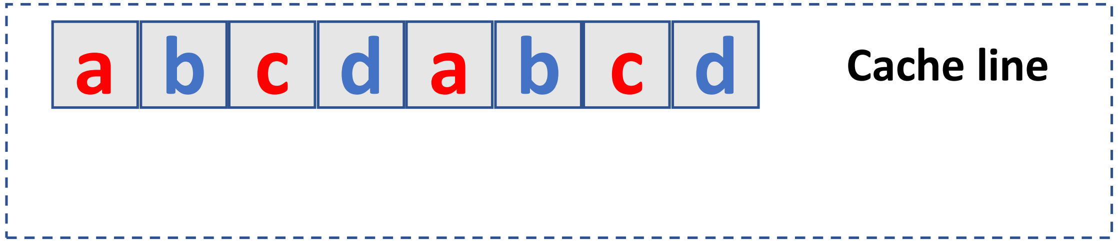


# Inefficient data-structure

```
struct type {int a; int b; int c; int d;};
struct type Arr[N];
for (i = 0; i < N; i++)
       B[i] = Arr[i].a + Arr[i].c;
```

Utilization = 50%

**L1 cache**



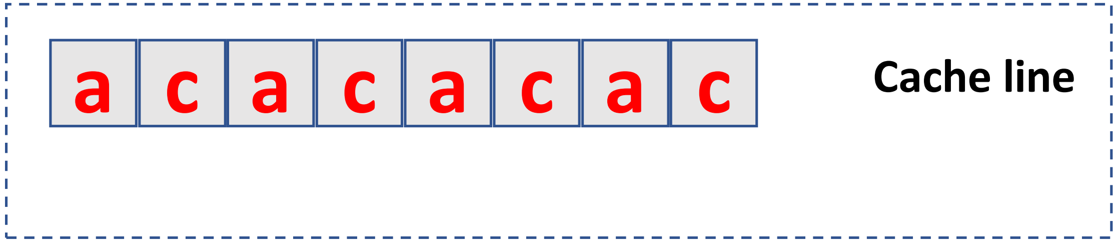
a b c d a b c d

**Cache line**

**Split structure**

```
struct type_part1 {int a; int c;};
struct type_part2 {int b; int d;};
```

**L1 cache**



a c a c a c a c

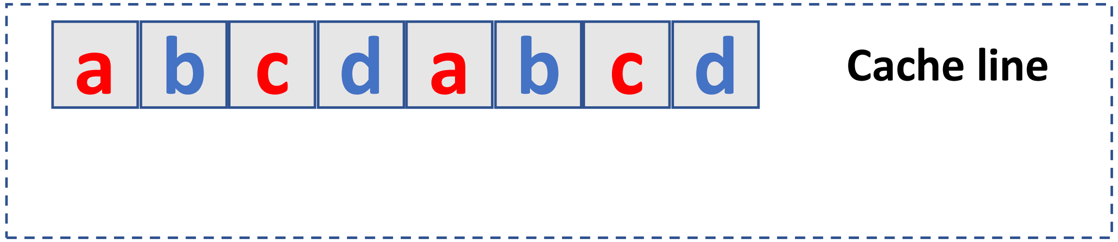
**Cache line**

# Inefficient data-structure

```
struct type {int a; int b; int c; int d;};
struct type Arr[N];
for (i = 0; i < N; i++)
       B[i] = Arr[i].a + Arr[i].c;
```

L1 cache

Utilization = 50%

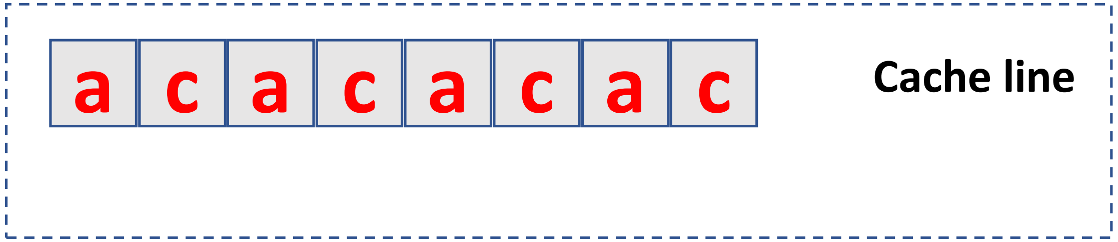


**Split structure**

```
struct type_part1 {int a; int c;};
struct type_part2 {int b; int d;};
```

L1 cache

Utilization = 100%



# Structure splitting- Questions to ask

How to split structure?

Which data structures are significant?

Which fields to keep together?

# Structure splitting- Questions to ask

How to split structure?

Which data structures are significant?

High usage

Which fields to keep together?

# Structure splitting- Questions to ask

How to split structure?

Which data structures are significant?

High usage

Which fields to keep together?

Loop level analysis

Field affinity

# Structure splitting- Questions to ask

How to split structure?

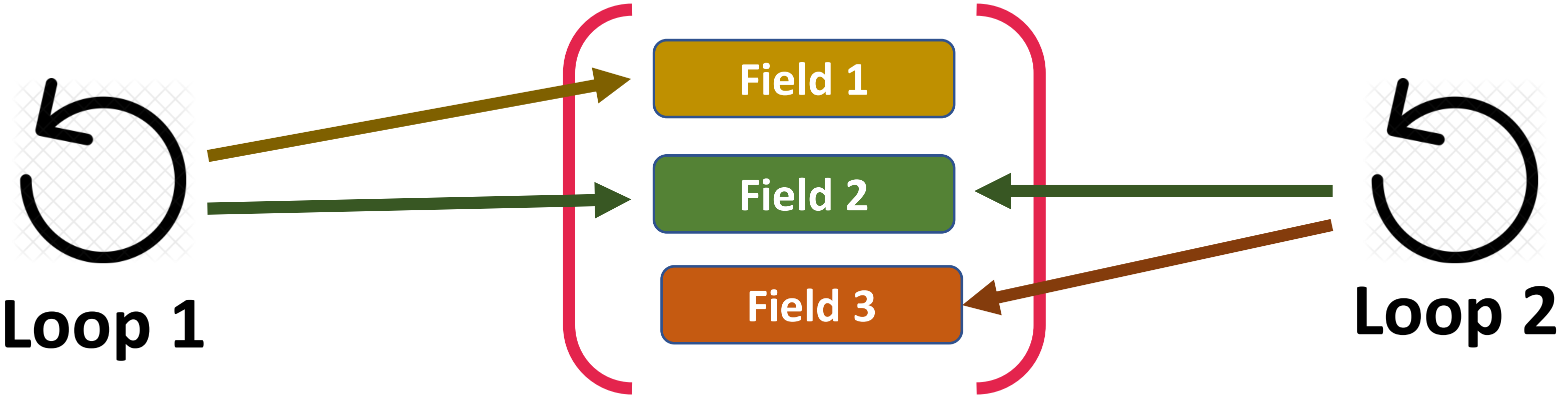
Which data structures are significant?

High usage

Which fields to keep together?

Loop level analysis

Field affinity



# Structure splitting- Questions to ask

How to split structure?

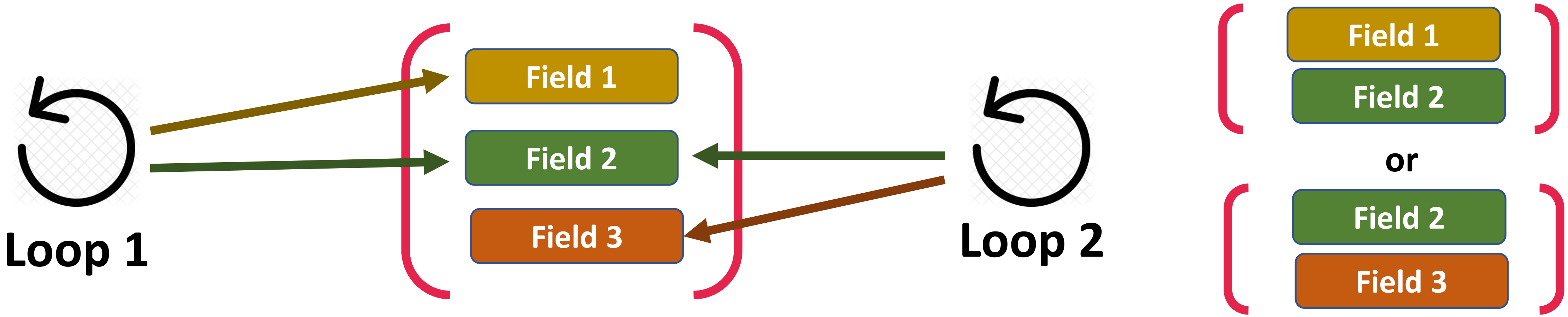
Which data structures are significant?

High usage

Which fields to keep together?

Loop level analysis

Field affinity



# Structure splitting- Questions to ask

How to split structure?

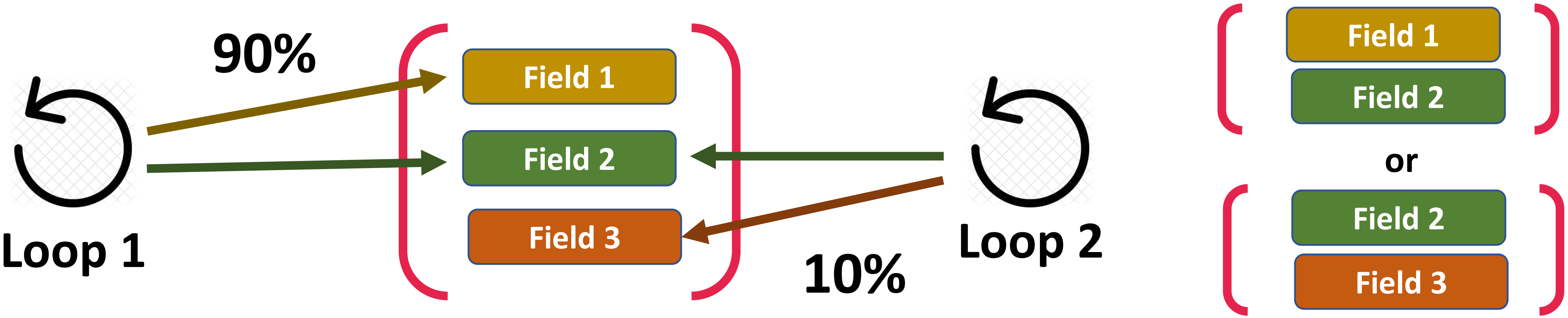
Which data structures are significant?

High usage

Which fields to keep together?

Loop level analysis

Field affinity



# Structure splitting- Questions to ask

How to split structure?

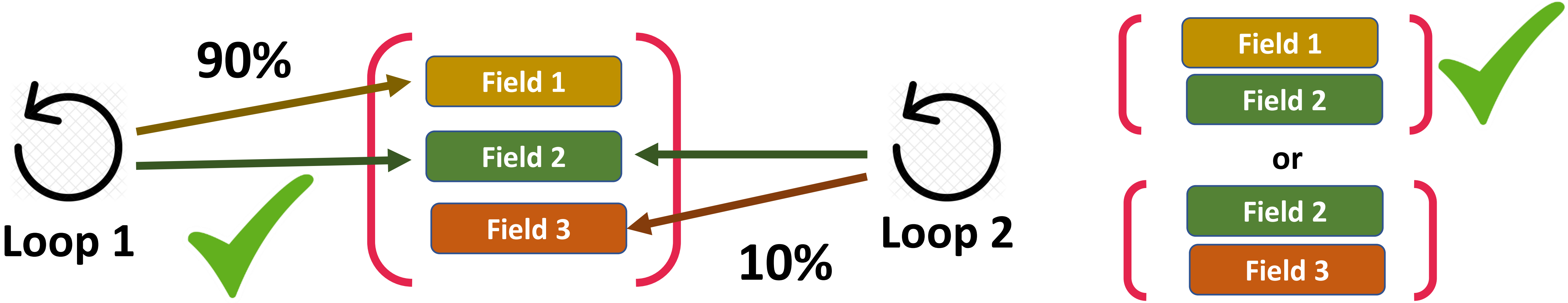
Which data structures are significant?

High usage

Which fields to keep together?

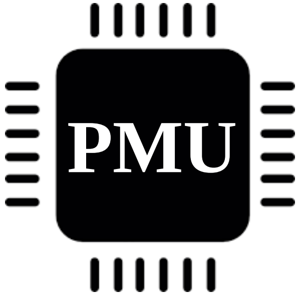
Loop level analysis

Field affinity

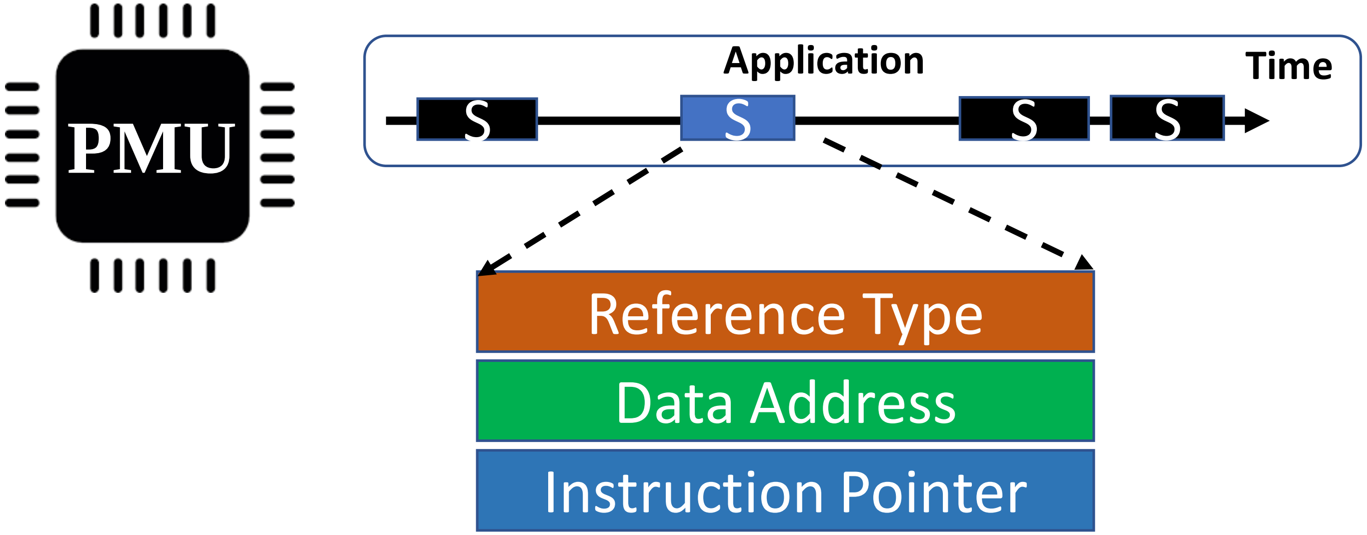


# Code-centric and data-centric attribution

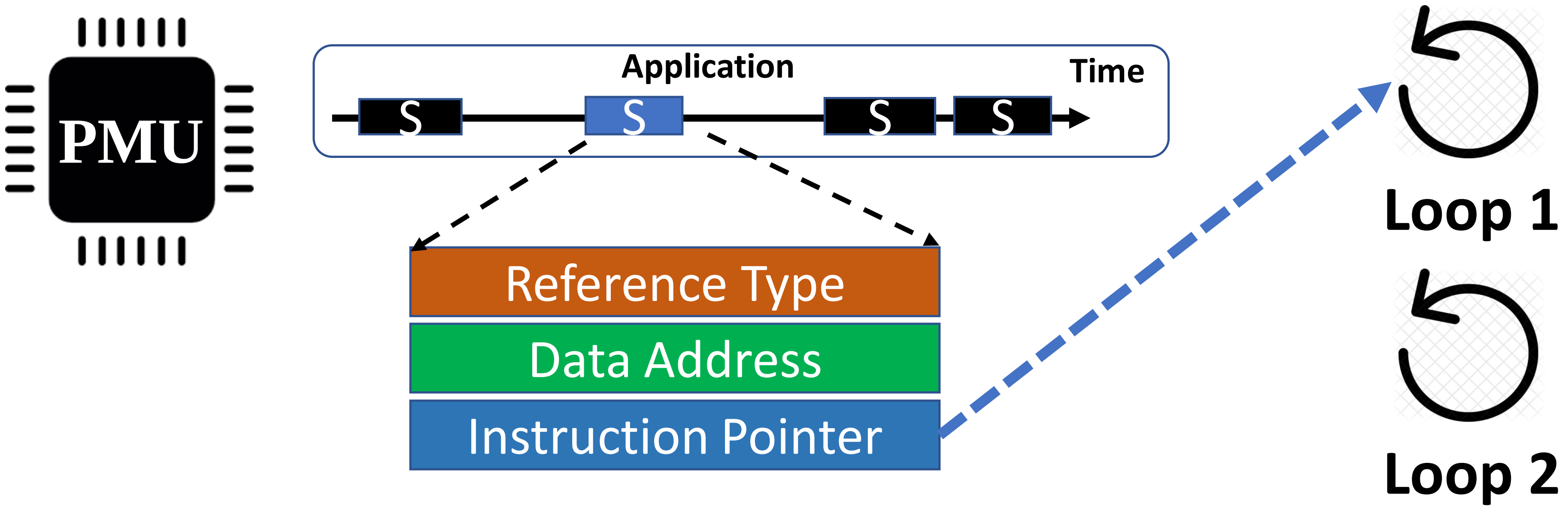
# Code-centric and data-centric attribution



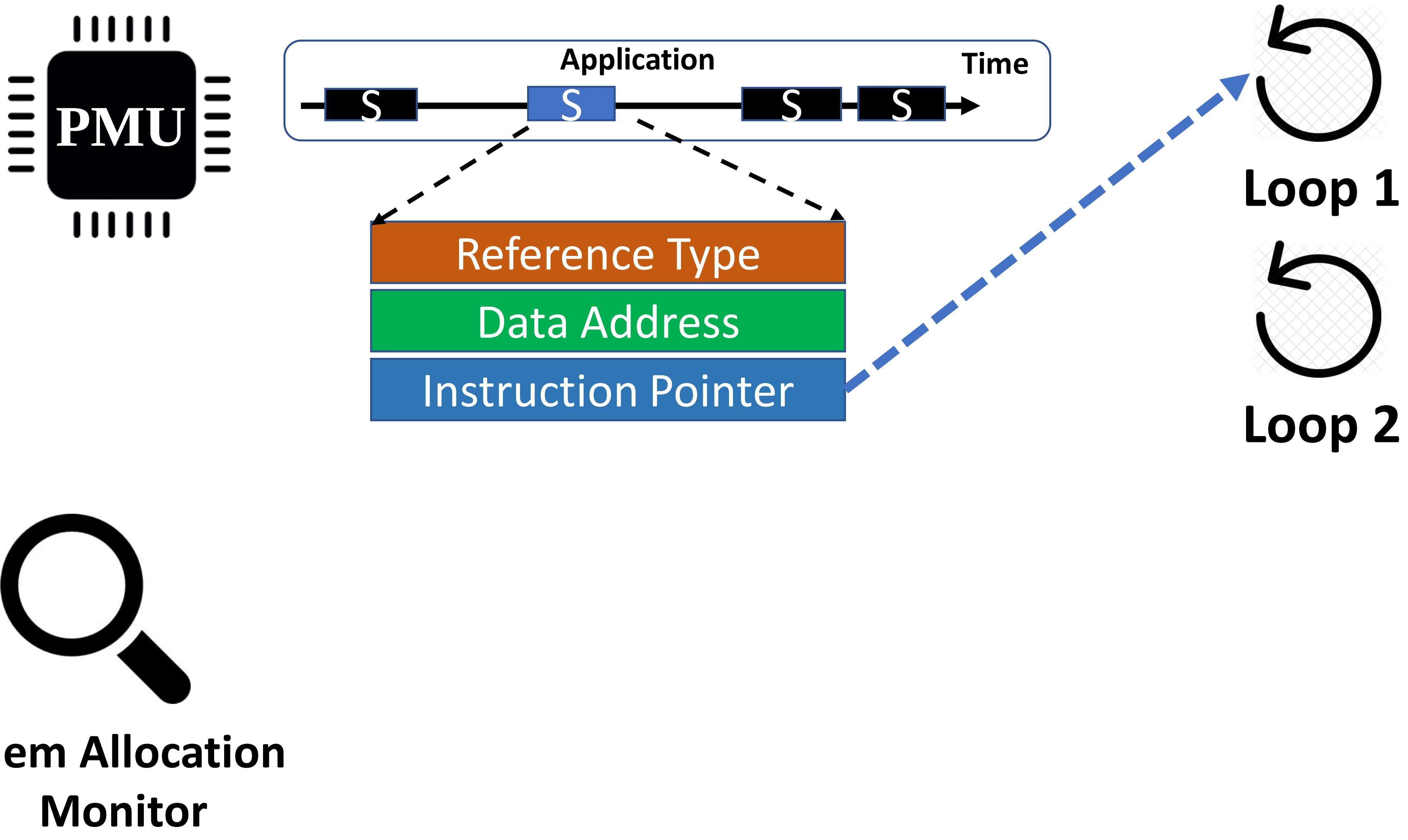
# Code-centric and data-centric attribution



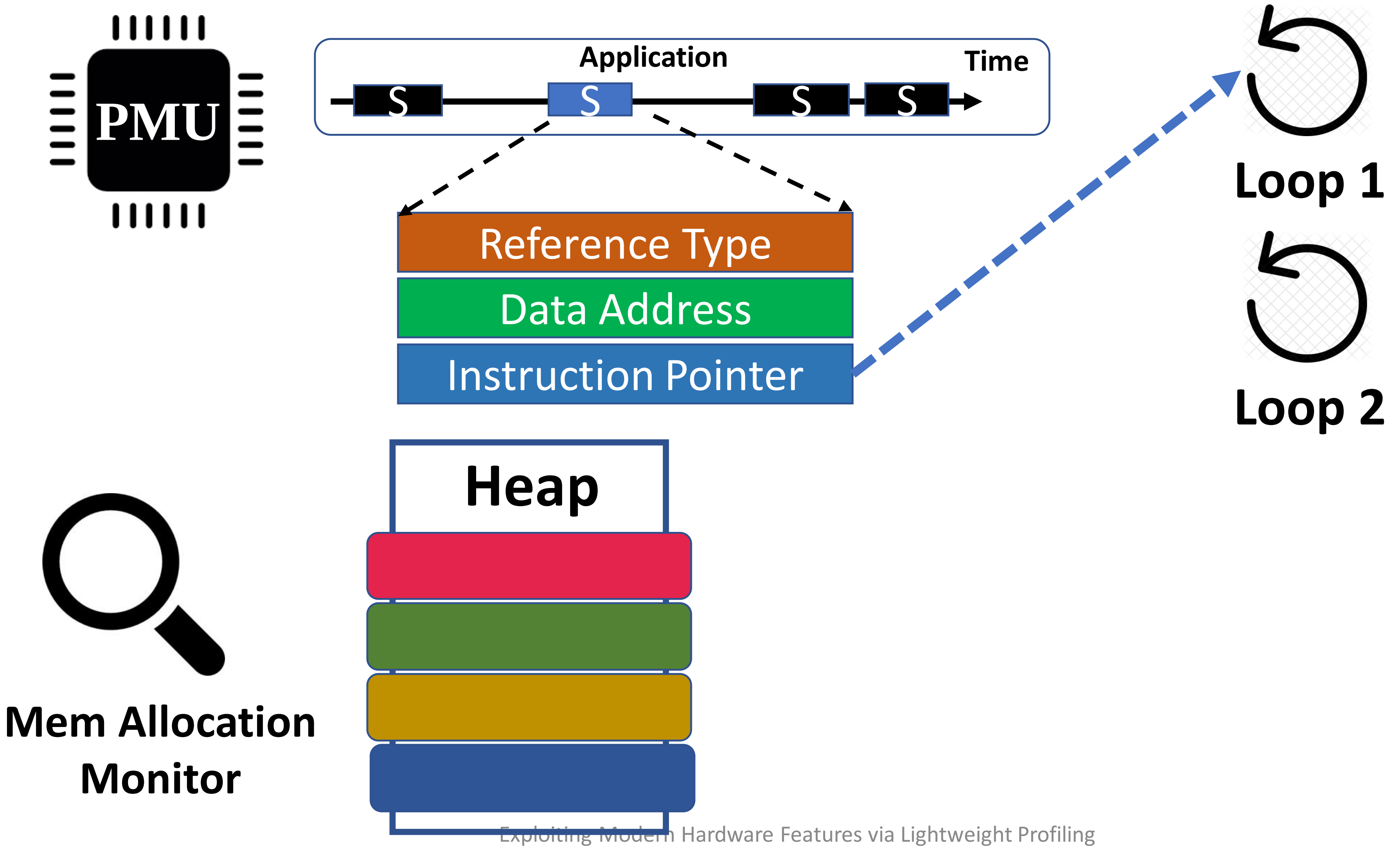
# Code-centric and data-centric attribution



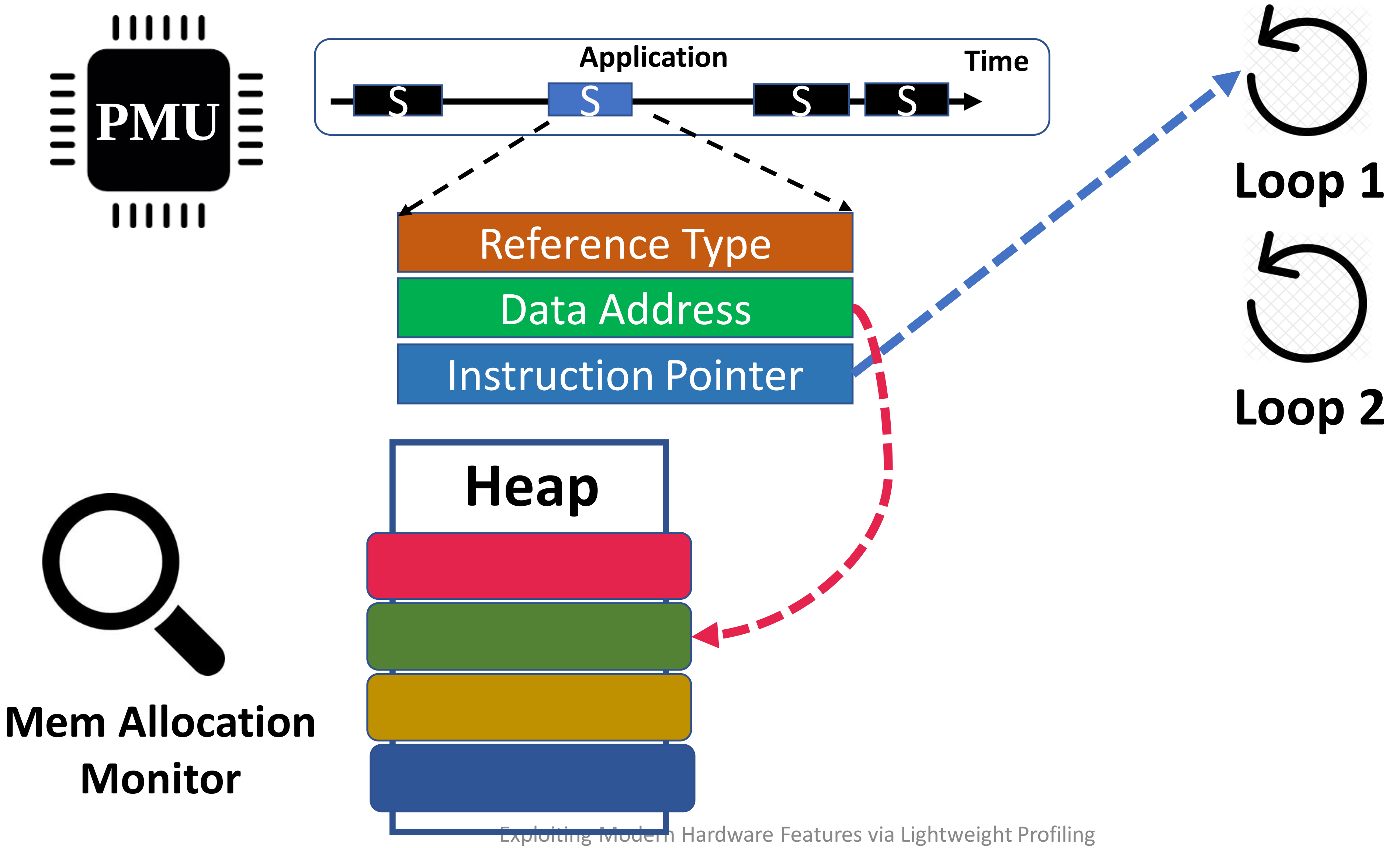
# Code-centric and data-centric attribution



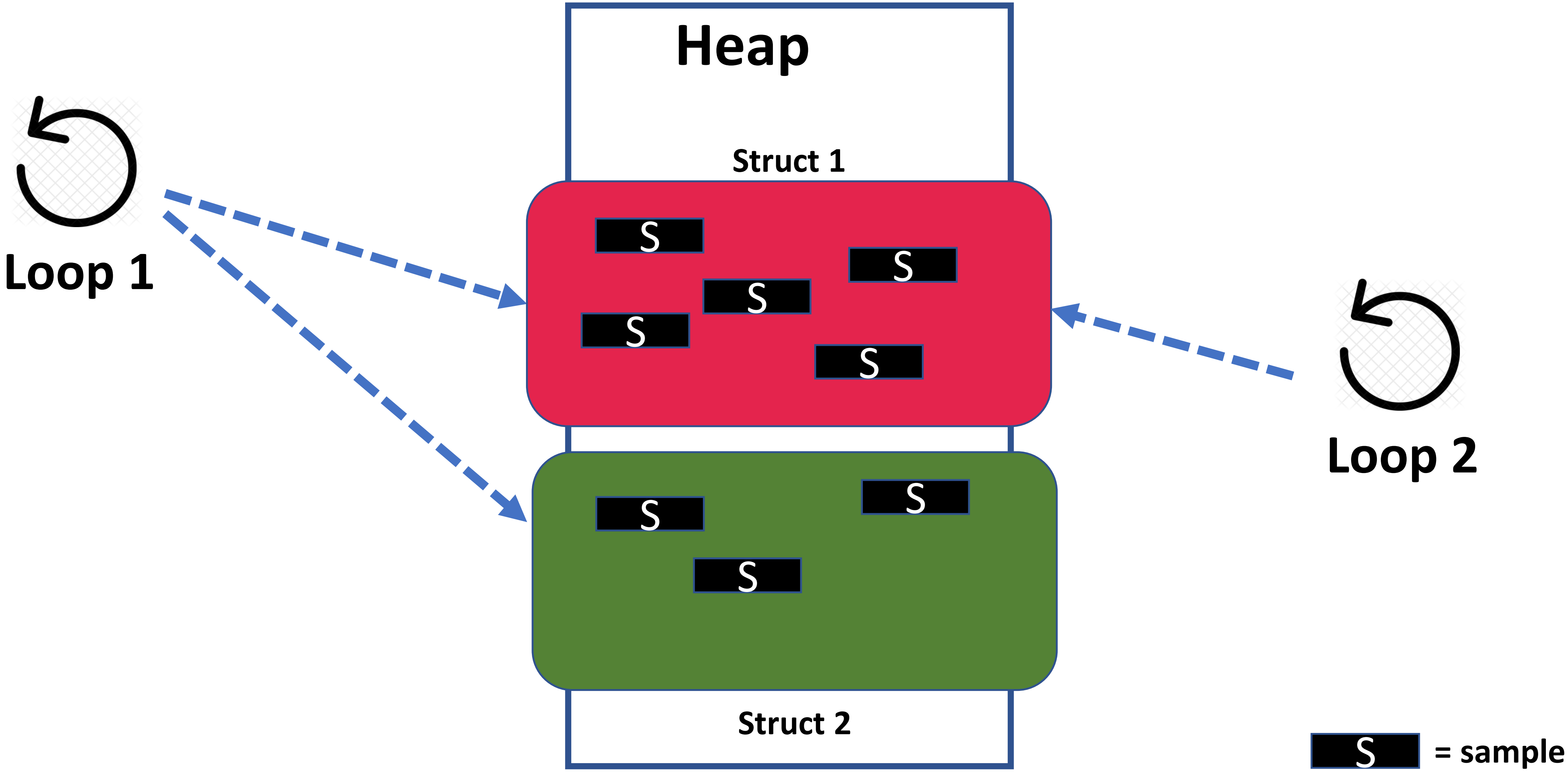
# Code-centric and data-centric attribution



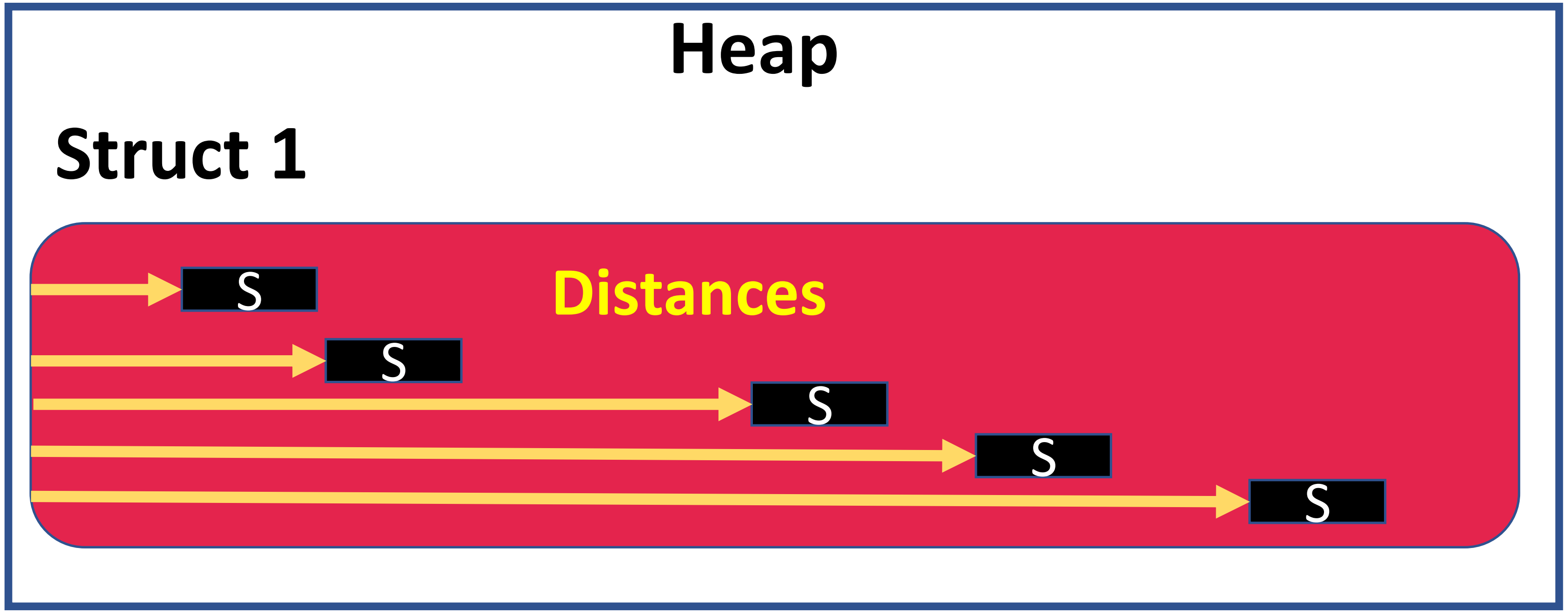
# Code-centric and data-centric attribution



# Code-centric and data-centric attribution

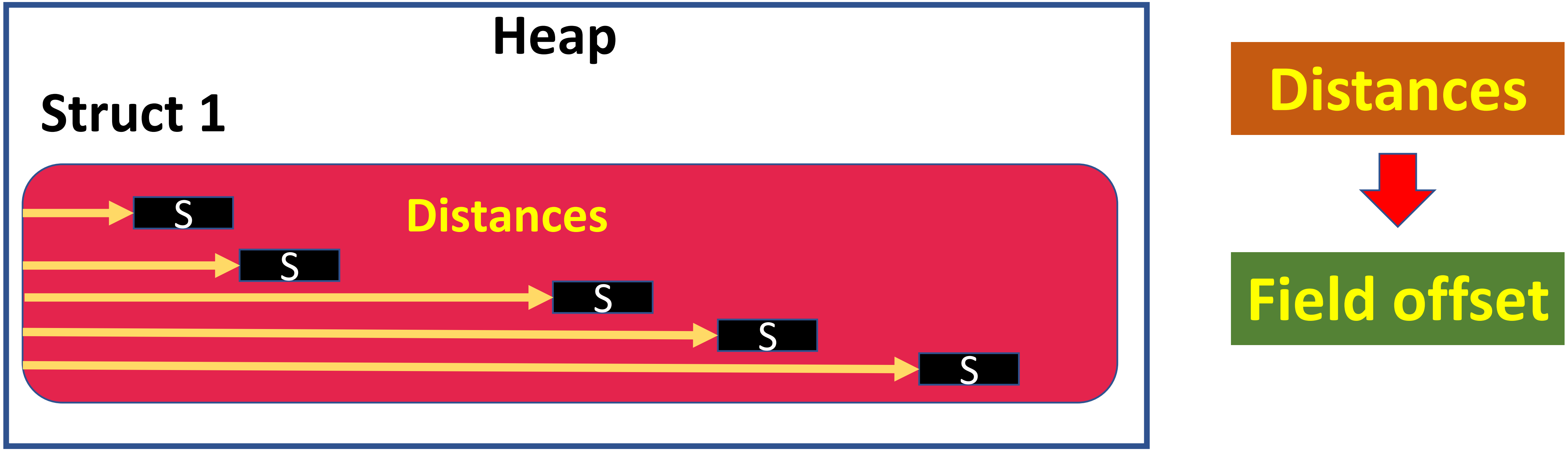


# Code-centric and data-centric attribution



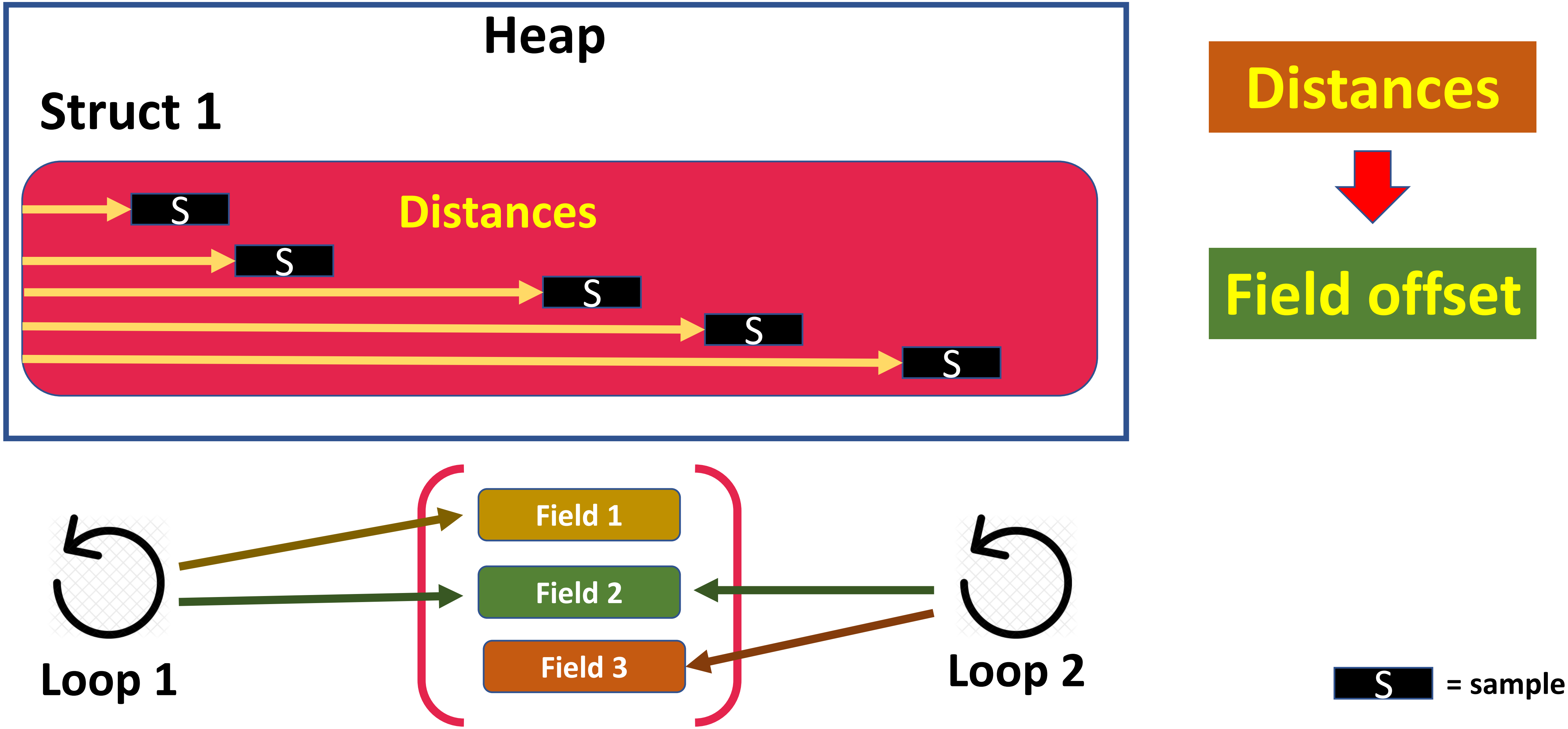
S = sample

# Code-centric and data-centric attribution



S = sample

# Code-centric and data-centric attribution



Heap

Struct 1

Distances

S

S

S

S

S

Distances

Field offset

Field 1

Field 2

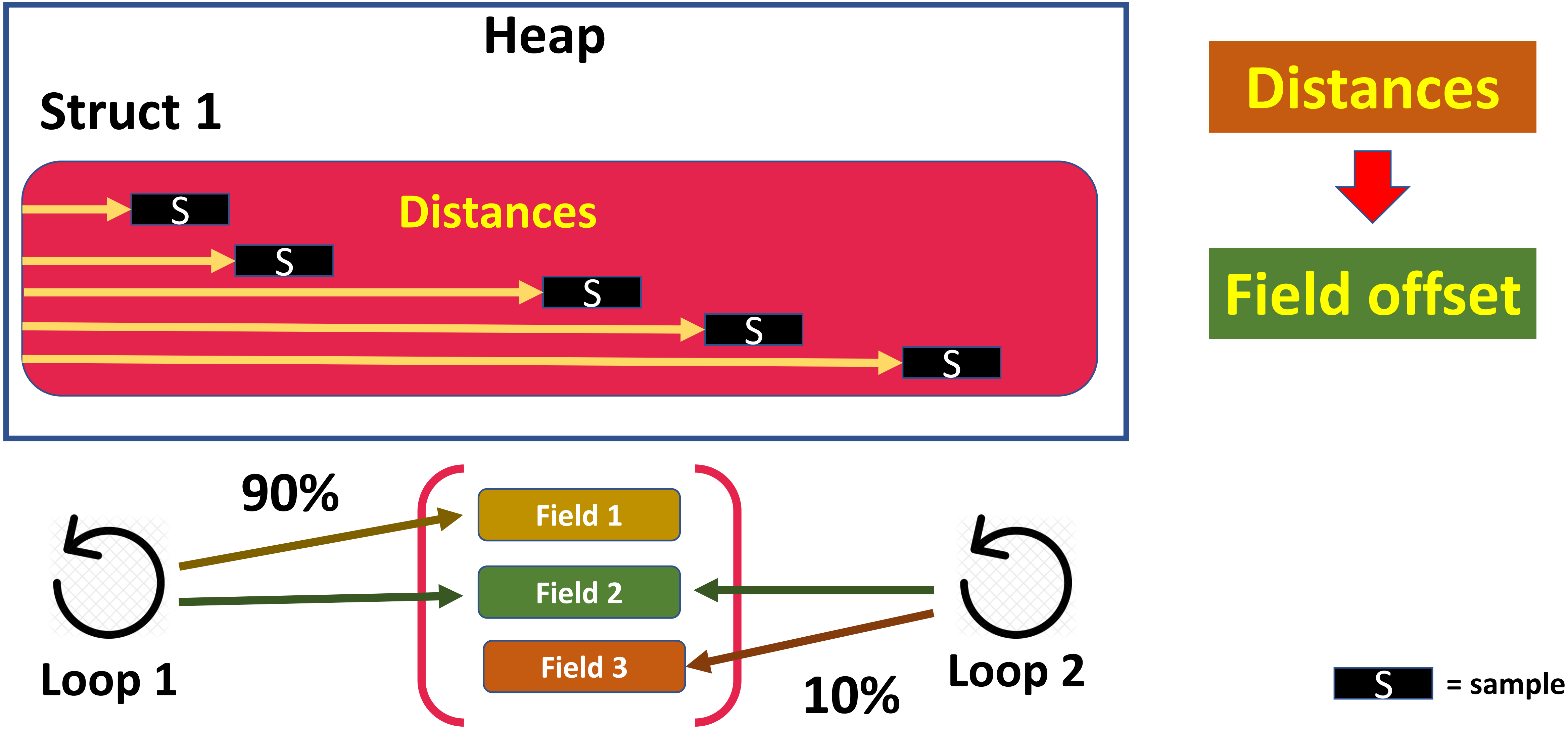
Field 3

Loop 1

Loop 2

S = sample

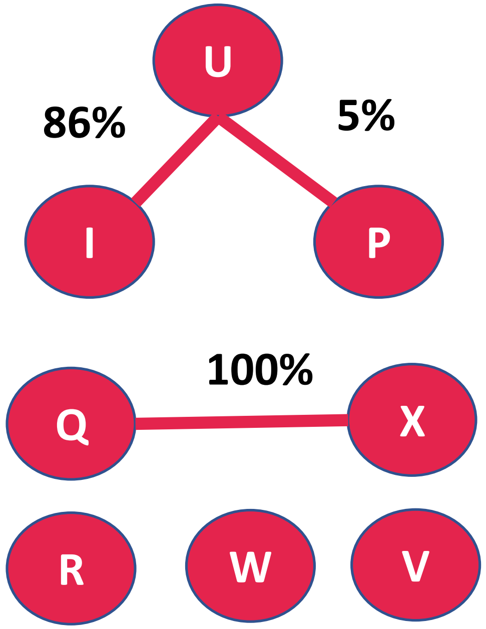
# Code-centric and data-centric attribution



# Case study: SPEC CPU 2000 ART

```
typedef struct
{
        double *I; double W; double X; double V; double U; double P; double Q; double R;
}f1_neuron
```

| Loops with line numbers | Latency percentage | Accessed fields |
|---|---|---|
| 131-138 | 1.59% | U,P |
| 559-570 | 8.42% | X,Q |
| 553-554 | 1.98% | W |
| 545-548 | 10.83% | U, I |
| 615-616 | 56.57% | P |
| 607-608 | 14.40% | P |
| 589-592 | 2.25% | U, P |
| 575-576 | 3.72% | V |
| 1015-1016 | 0.24% | I |



**Affinity graph**

# Case study: SPEC CPU 2000 ART

```
typedef struct
{
        double *I; double W; double X; double V; double U; double P; double Q; double R;
}f1_neuron
```

| Loops w<br>nu |
|---|
| 13 |
| 55 |
| 55 |
| 54 |
| 61 |
| 60 |
| 58 |
| 57 |
| 1015 |

```
typedef struct{ double *I; double U;} f1_neuron_IU;
typedef struct{ double Q; double X;} f1_neuron_QX;
typedef struct{ double P;} f1_neuron_P;
typedef struct{ double V;} f1_neuron_V;
typedef struct{ double W;} f1_neuron_W;
typedef struct{ double R;} f1_neuron_R;
```

# Benchmarks: speedup, overhead, cache miss

| Benchmarks | Speedups | Runtime overhead | L1 miss reduction | L2 miss reduction |
|---|---|---|---|---|
| 179.ART | 1.37× | 2.05% | 46.5% | 51.1% |
| 462.Libquantum | 1.09× | 2.79% | 49% | 82.6% |
| TSP | 1.09× | 2.42% | 13.3% | 19.9% |
| Mser | 1.03× | 2.95% | 8.3% | 8.4% |
| CLOMP 1.2 | 1.25× | 16.1% | 15.5% | 26.4% |
| Health | 1.12× | 18.3% | 66.7% | 90.8% |
| NN | 1.33× | 5.21% | 87.2% | 98.0% |
| **Average** | **1.18×** | **7.1%** | | |

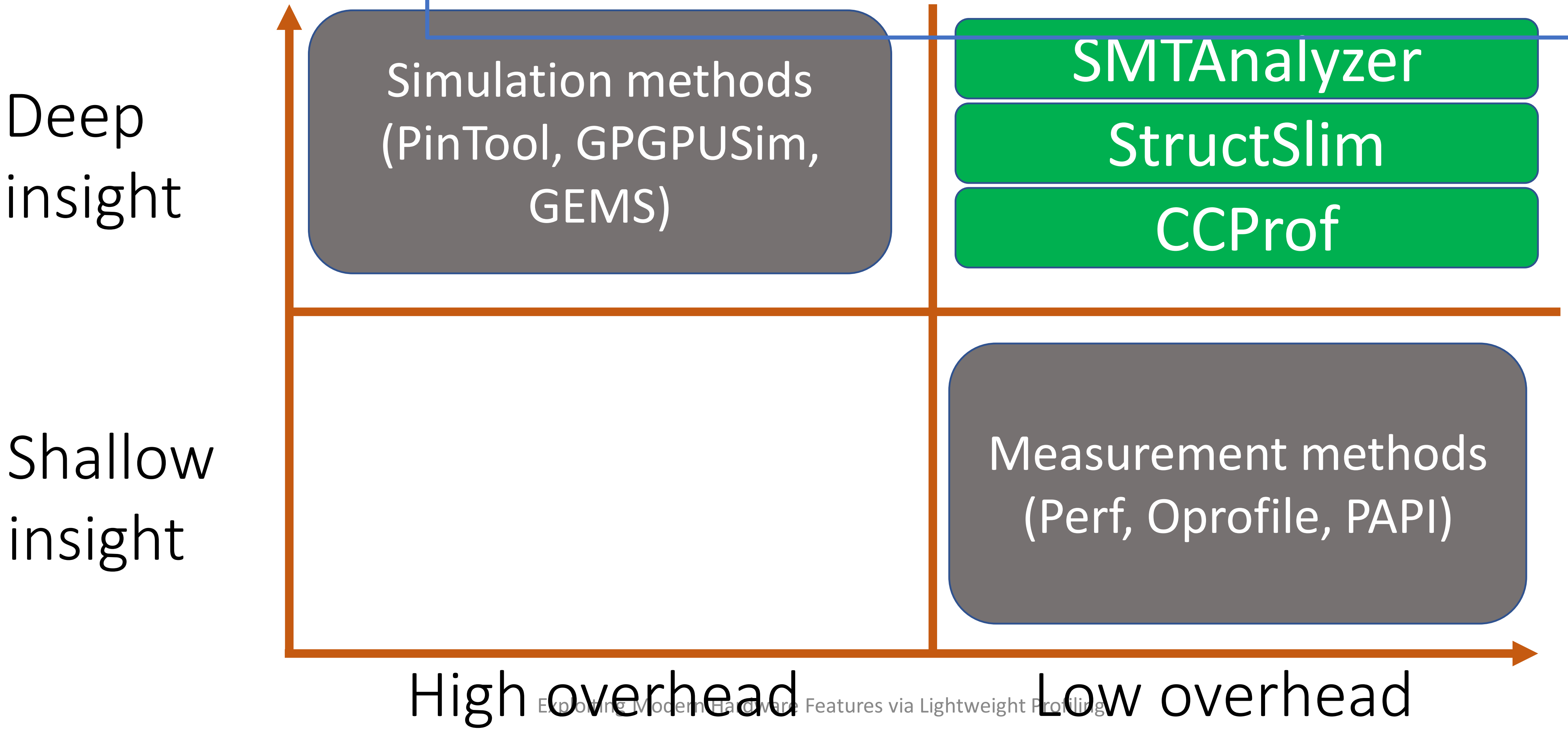
**gcc -O3**

Related work: Overhead: average 4x

Yan, Jianian, Jiangzhou He, Wenguang Chen, Pen-Chung Yew, and Weimin Zheng. "ASLOP: A field-access affinity-based structure data layout optimizer."

# Conclusions

**Lightweight profiling with PMUs can provide deep insights into performance issues cause by memory hierarchies and poor algorithm choice.**



Deep insight

Shallow insight

Simulation methods (PinTool, GPGPUSim, GEMS)

SMTAnalyzer

StructSlim

CCProf

Measurement methods (Perf, Oprofile, PAPI)

High overhead

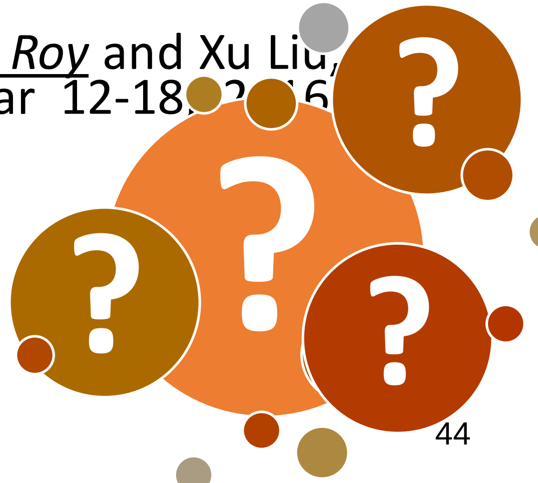
Low overhead

# Publications

- **[CGO'18] "Lightweight Detection of Cache Conflicts",** *Probir Roy***,** Shuaiwen Leon Song, Sriram Krishnamoorthy and Xu Liu, The 2018 International Symposium on Code Generation and Optimization, Feb 24 - 28th, 2018, Vienna, Austria. Acceptance ratio: 28%.
- **[TACO'18] "NUMA-Caffe: NUMA-Aware Deep Learning Neural Networks",** *Probir Roy***,** Shuaiwen Leon Song, Sriram Krishnamoorthy, Abhinav Vishnu, Dipanjan Sengupta, Xu Liu, ACM Transactions on Architecture and Code Optimization, 2018.
- **[TPDS'18] "LWPTool: A Lightweight Profiler to Guide Data Layout Optimization"**, Chao Yu, *Probir Roy*, Yuebin Bai, Hailong Yang, Xu Liu, IEEE Transactions on Parallel and Distributed Systems, 2018.
- **[HPDC'16] "SMT-Aware Instantaneous Footprint Optimization",** *Probir Roy*, Xu Liu and Shuaiwen Leon Song, The 25th ACM International Symposium on High-Performance and Distributed Computing, May 31 - Jun 4, 2016, Kyoto, Japan. Acceptance ratio: 15.5% (20/129).
- **[CGO'16] "StructSlim: A Lightweight Profiler to Guide Structure Splitting"**, *Probir Roy* and Xu Liu, The 2016 International Symposium on Code Generation and Optimization, Mar 12-18, 2016, Barcelona, Spain. Acceptance ratio: 23%.

# Publications

- **[CGO'18] "Lightweight Detection of Cache Conflicts",** *Probir Roy***,** Shuaiwen Leon Song, Sriram Krishnamoorthy and Xu Liu, The 2018 International Symposium on Code Generation and Optimization, Feb 24 - 28th, 2018, Vienna, Austria. Acceptance ratio: 28%.
- **[TACO'18] "NUMA-Caffe: NUMA-Aware Deep Learning Neural Networks",** *Probir Roy***,** Shuaiwen Leon Song, Sriram Krishnamoorthy, Abhinav Vishnu, Dipanjan Sengupta, Xu Liu, ACM Transactions on Architecture and Code Optimization, 2018.
- **[TPDS'18] "LWPTool: A Lightweight Profiler to Guide Data Layout Optimization"**, Chao Yu, *Probir Roy*, Yuebin Bai, Hailong Yang, Xu Liu, IEEE Transactions on Parallel and Distributed Systems, 2018.
- **[HPDC'16] "SMT-Aware Instantaneous Footprint Optimization",** *Probir Roy*, Xu Liu and Shuaiwen Leon Song, The 25th ACM International Symposium on High-Performance and Distributed Computing, May 31 - Jun 4, 2016, Kyoto, Japan. Acceptance ratio: 15.5% (20/129).
- **[CGO'16] "StructSlim: A Lightweight Profiler to Guide Structure Splitting"**, *Probir Roy* and Xu Liu, The 2016 International Symposium on Code Generation and Optimization, Mar 12-18, 2016, Barcelona, Spain. Acceptance ratio: 23%.



# Challenges ahead

- Program analysis for declarative programming languages
  - Domain specific languages provide high-level abstraction
    - Machine learning (PyTorch), HPC (HDF5), big-data (SQL)
- Analyzing and optimizing data center and cloud application
  - Resource utilization/scheduling in multi-tenant environment
  - Heterogenous architecture resource management
- Security analysis
  - Program analysis to identify vulnerable source code
- Analysis of emerging hardware
  - GPU, FPGA, Tensor processing units