Profiling over Adaptive Ranges

Shashidhar Mysore, Banit Agrawal, Timothy Sherwood
Nisheeth Shrivastava, Subhash Suri

Department of Computer Science
University of California, Santa Barbara

4th Annual ACM/IEEE International Symposium on Code Generation and Optimization (CGO), 27th March 2006, Manhattan, NY

Presented by: SHASHI MYSORE
shashimc@cs.ucsb.edu

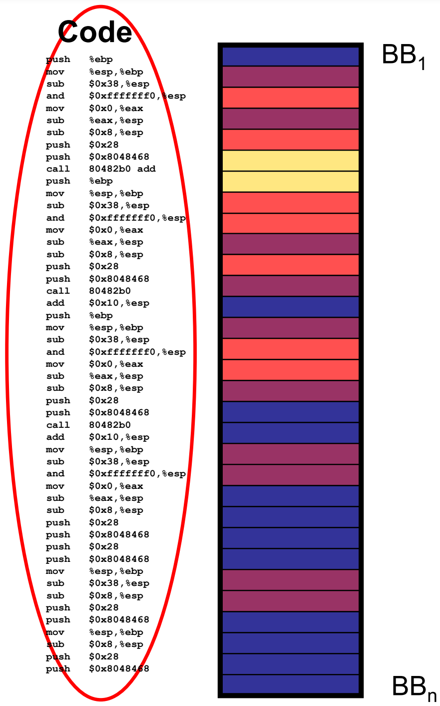
Motivation

Program Profiling: Understand system-workload interactions - gather data, *quantify, analyze*, and optimize

- At the core: We need to count events
- ***Basic blocks, load value distribution, load instructions, load addresses, zero-value loads, narrow-width operands*, etc.**
- Challenge:
  - Huge complex programs
  - Limited storage - tiny streaming profilers
  - Runtime analysis - feasible hardware solutions

Let us consider an example of code profiling …

An example: Code Profiling



Each basic block executes some number of times

Some are hot

Some are not

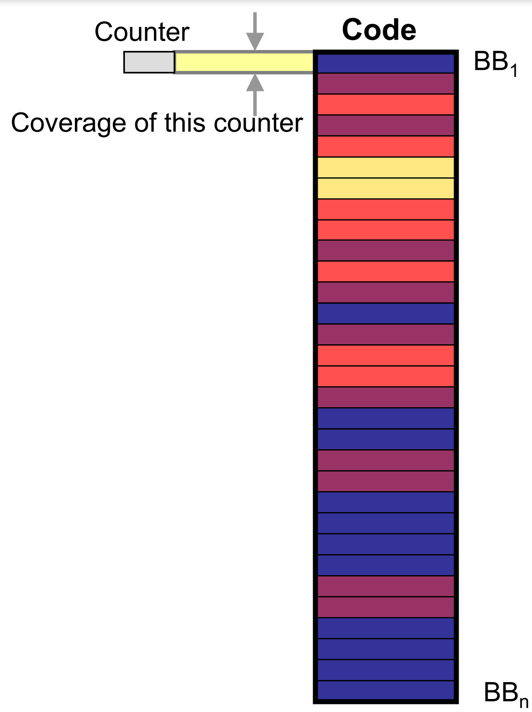
Where are the hot regions?

How hot are they?

… And can we *discover* this knowledge at run time?

Use counters …

Naïve Approach: Unlimited Counters



Naïve Approach: Unlimited Counters

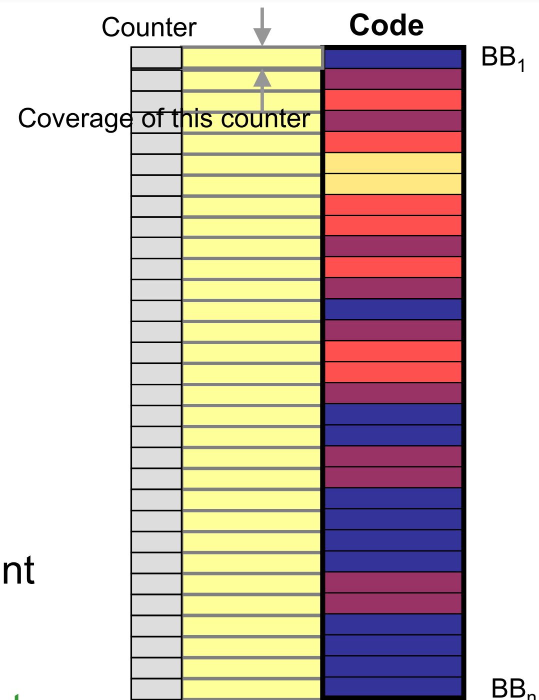
N basic blocks – *N* counters
Each counter covers one basic block

We get:
- High Coverage
- High Precision

Problem:
- Many programs have 800000 basic blocks or more!

but.. not all of them are important to be quantified

So let's limit the number of counters …



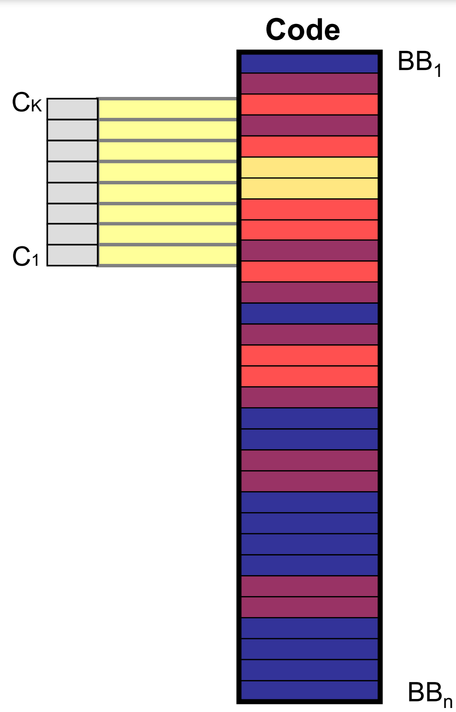
Naïve Approach: Limited Counters

N basic blocks – *K* counters

pick *K* basic blocks and let the *K* counters cover them

We get:
- High Precision - For the hot spots
- Low Coverage - At the right spots

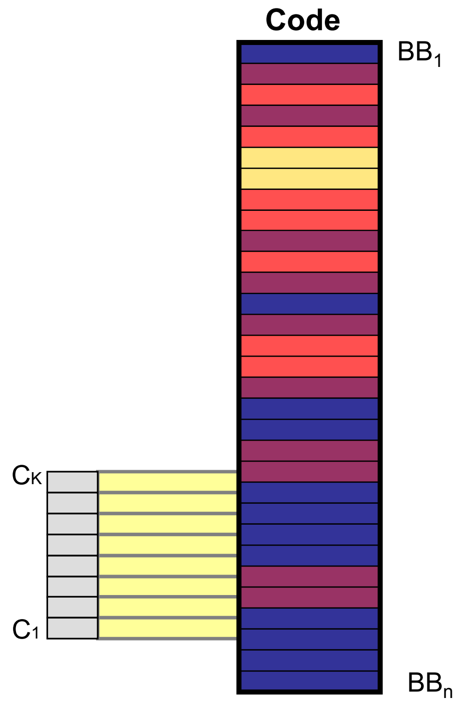
but what if did …



Naïve Approach: Limited Counters

N basic blocks – *K* counters

pick *K* basic blocks and let the *K* counters cover them



Naïve Approach: Limited Counters

N basic blocks – *K* counters

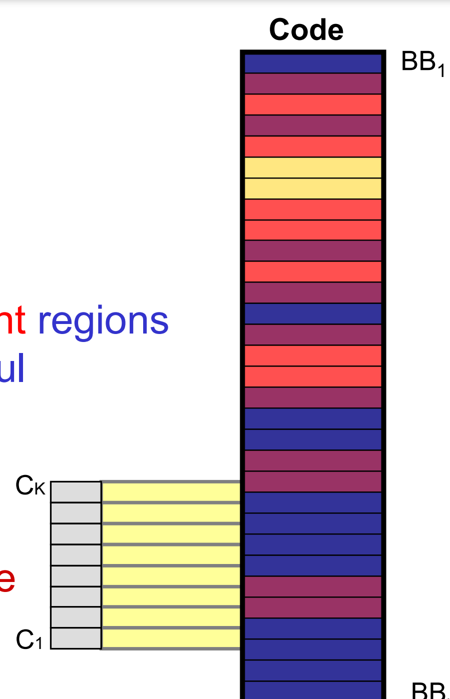
pick *K* basic blocks and let the *K* counters cover them

We get:

- Low Coverage – and at unimportant regions
- High Precision – but is not as useful

Problem:

- We have zero information about hot regions
- How do we know which region of the code to cover with the K counters?



Distribute the basic blocks among the counters …

Naïve Approach: Uniform Ranges

Each counter counts a *range* of basic blocks

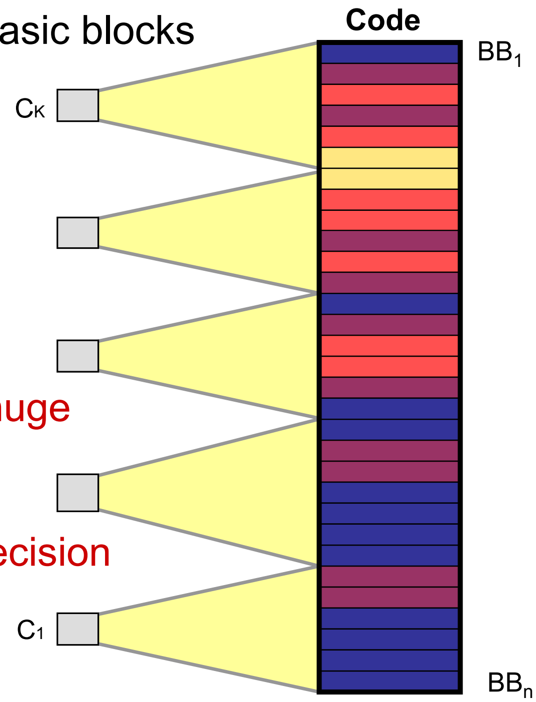
K counters to cover the entire program

We get:

- High Coverage with *K* counters
- Low Precision

Problem:

- One counter associated with a huge set of basic blocks
- Only average behavior – low precision
- Precision important – especially for hot regions



Related Work

- Profile Gathering and analysis schemes
  - [Anderson, et. Al., ’97], [Arnold, et. Al., ’01], [Heil and Smith, ’00], [Sastry, et. Al., ’01], [Ball and Larus, ’96], [Calder, et. Al., 97], [Hirzel and Chilimbi, ’01]
- Hardware assisted profiling and optimizations
  - [Brooks, et. Al., ’99], [Conte, et, al., ’94, ’96] [Dean, et, al., ’97], [Narayanasamy, et., al.,’03], [Zhou, et. Al., ’04], [Zilles and Sohi, ’01], [Nagpurkar et. Al., ’05], [Mousa, et. Al, ’05]

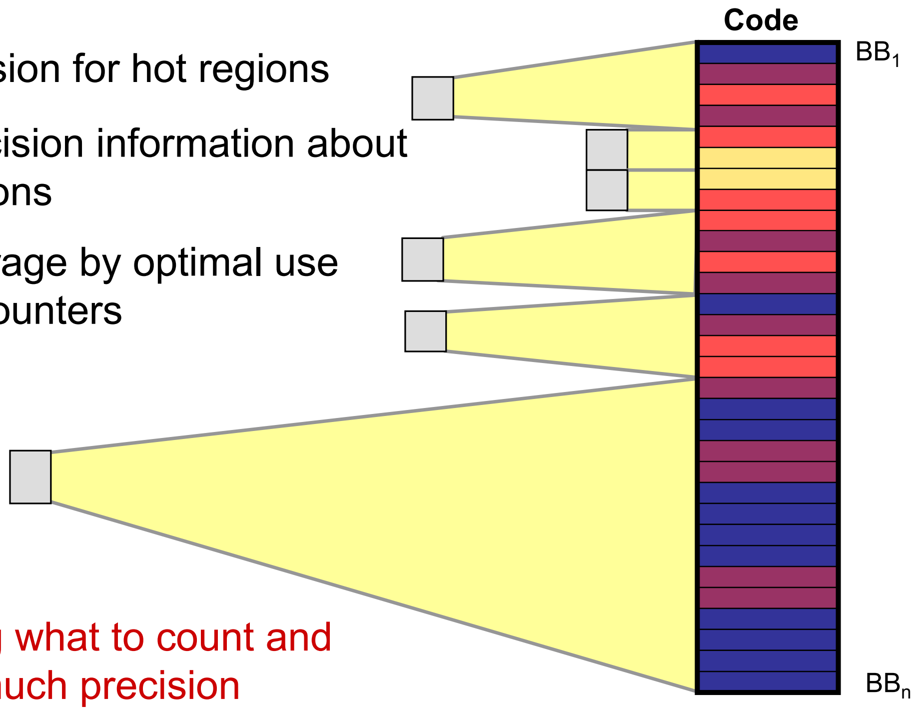
▪High Coverage
▪High precision
▪Limited number of counters
▪Covers any stream of profile data

•Low precision information on cold regions
•Divide profile data hierarchically

Ideally: Best Ranges

We want:

- High Precision for hot regions
- Lower precision information about colder regions
- High coverage by optimal use of a few counters



Challenge:
Discovering what to count and
With how much precision

Challenges

Ideal Profiler: Selects the best possible ranges; decides the precision

Real Problem: Identifying the best possible ranges to count before we already start counting

Or..

Challenges

Ideal Profiler: Selects the best possible ranges; decides the precision

Real Problem: Identifying the best possible ranges to count before we already start counting

Start Counting

Identify ranges

What comes first?

Challenges

Ideal Profiler: Selects the best possible ranges; decides the precision

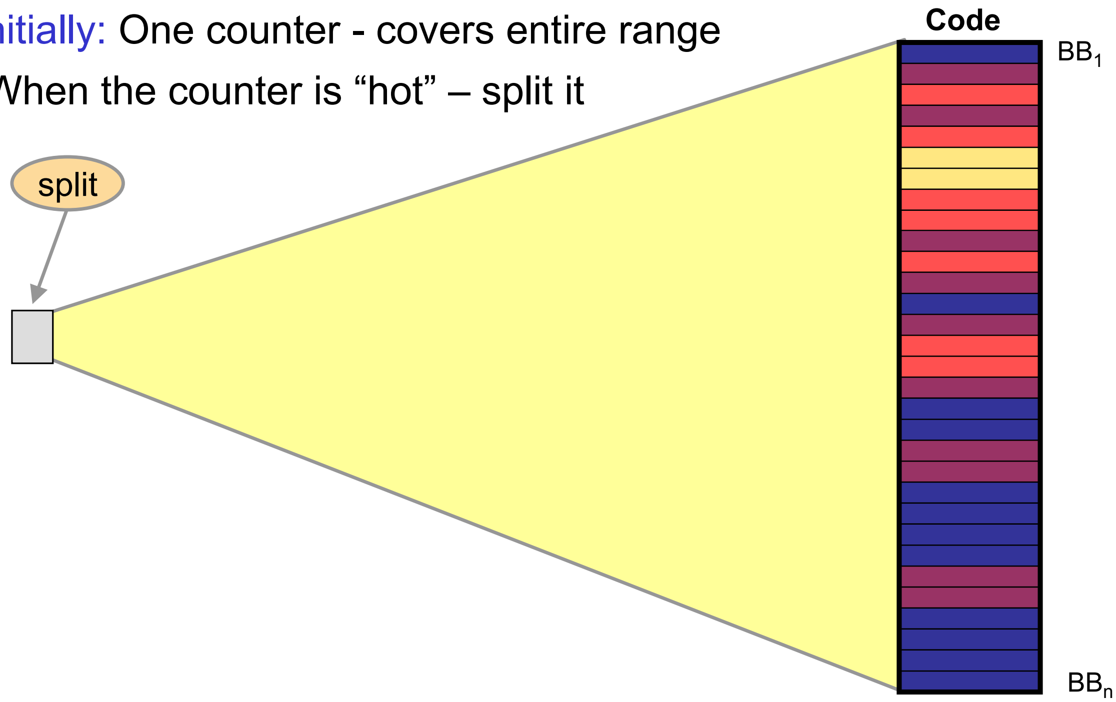
Real Problem: Identifying the best possible ranges to count before we already start counting

Range Adaptive Profiler solves exactly this problem by dynamically identifying ranges as we count

Our Approach: Adaptive Profiling

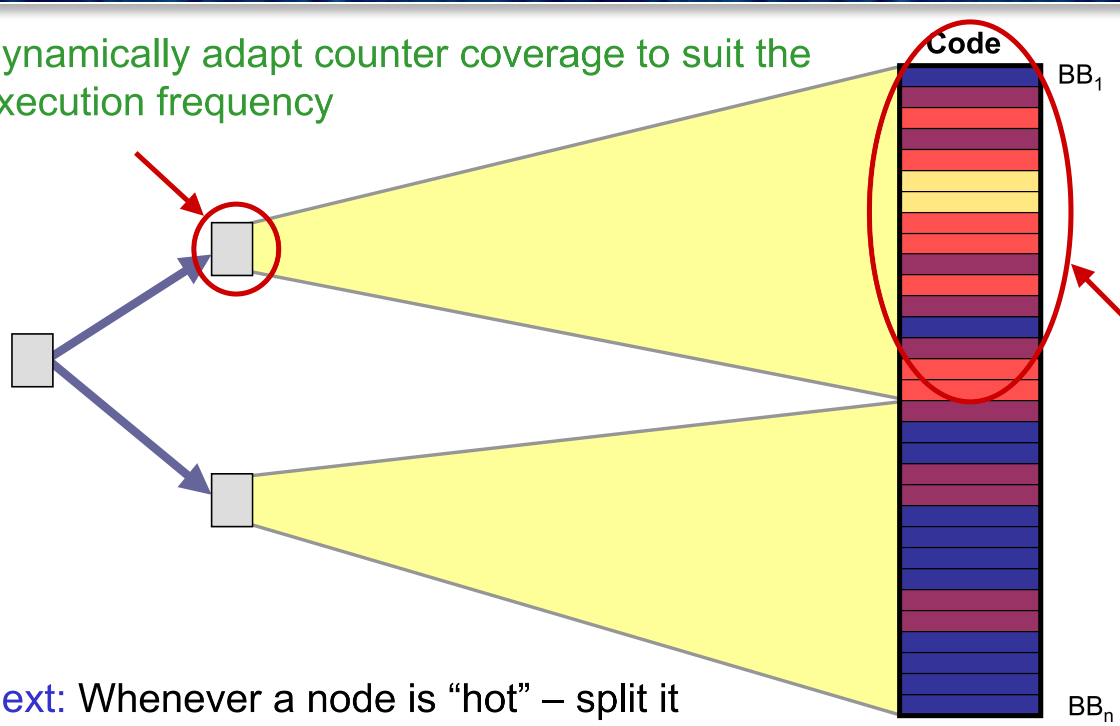
Initially: One counter - covers entire range

When the counter is "hot" – split it



Our Approach: Adaptive Profiling

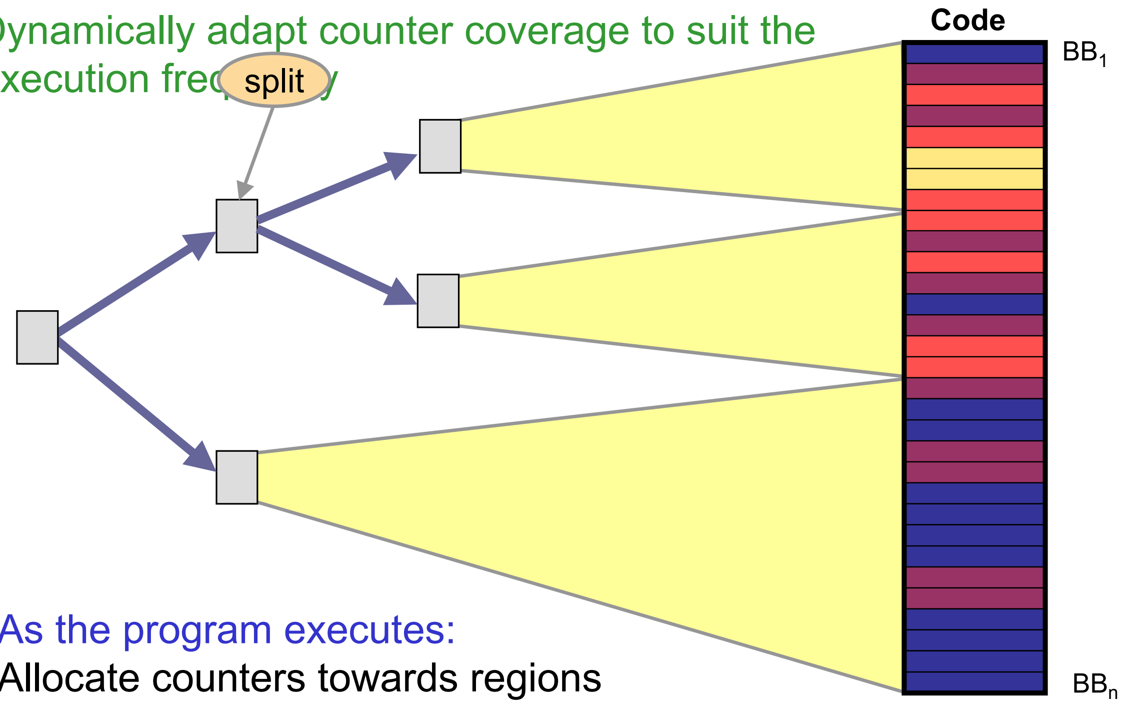
Dynamically adapt counter coverage to suit the execution frequency



Next: Whenever a node is "hot" – split it

Our Approach: Adaptive Profiling

Dynamically adapt counter coverage to suit the execution frequency

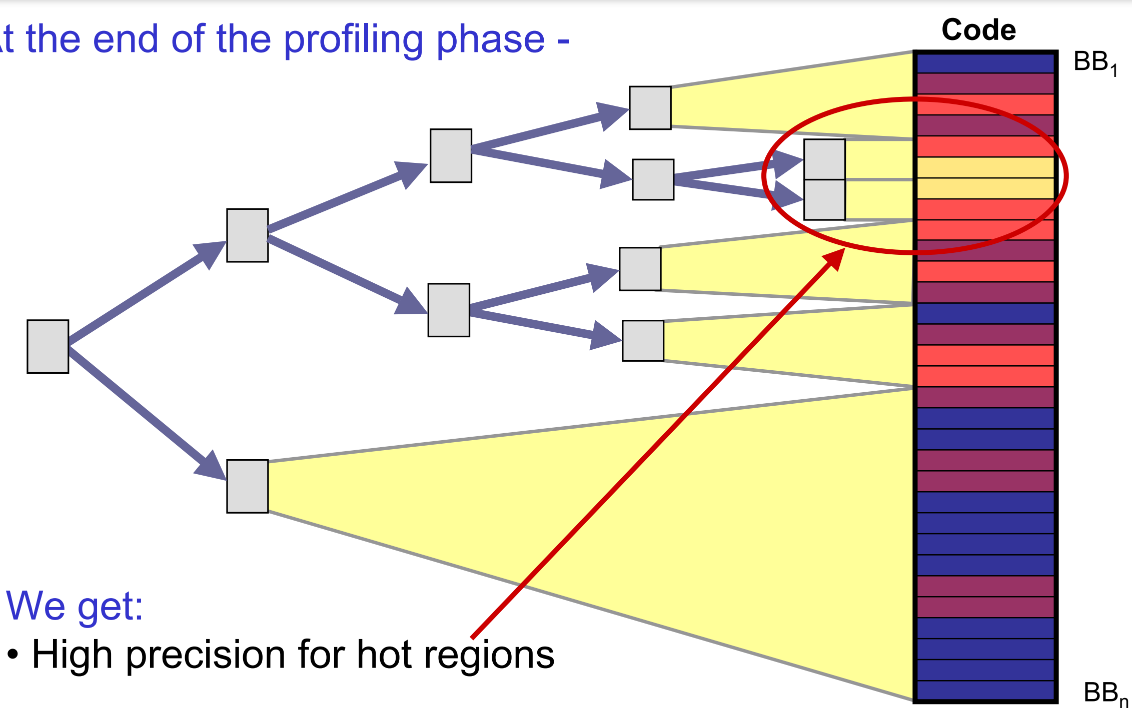


As the program executes:

Allocate counters towards regions that are hotter

Our Approach: Adaptive Profiling

At the end of the profiling phase -

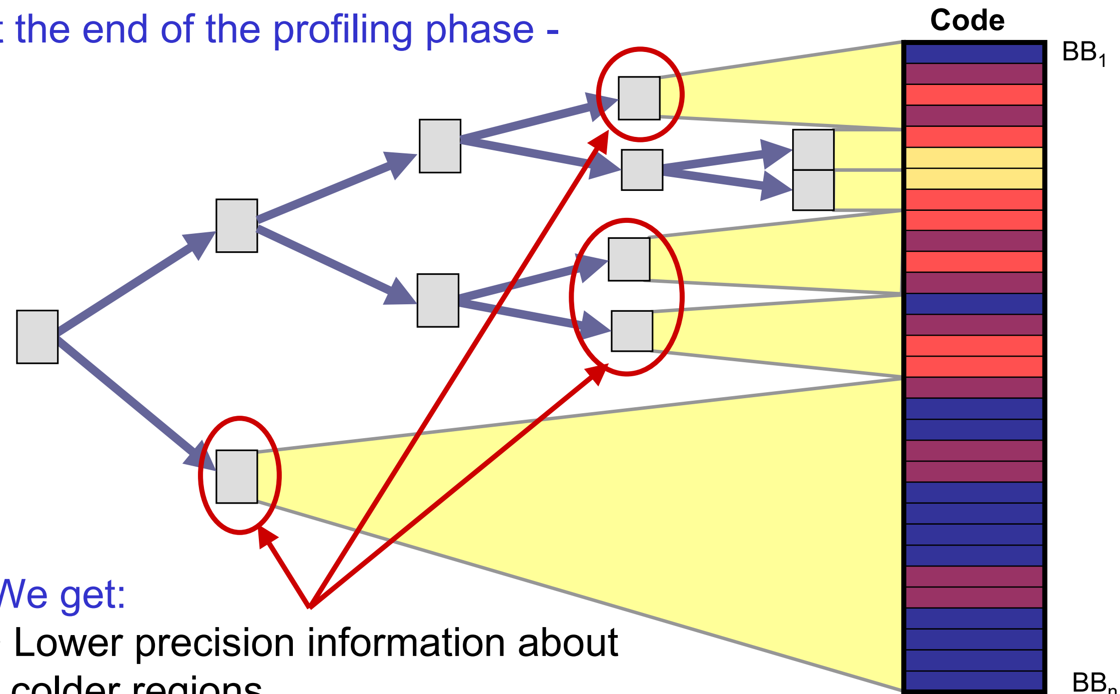


We get:

- High precision for hot regions

Our Approach: Adaptive Profiling

At the end of the profiling phase -

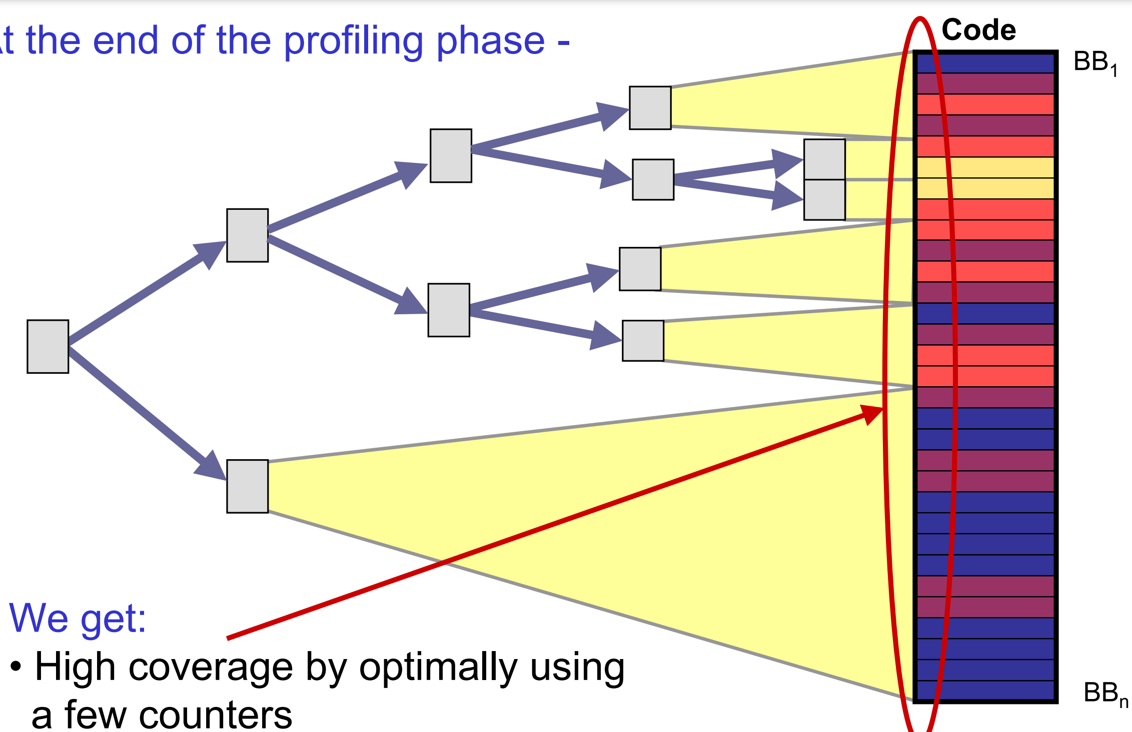


We get:

- Lower precision information about colder regions

Our Approach: Adaptive Profiling

At the end of the profiling phase -



We get:

- High coverage by optimally using a few counters

Range Adaptive Profiling

Advantages:

- A streaming (one-pass) technique to hierarchically classify events
- Fixed number of counters – *O(log(R) * 1/E)*
- Precision adaptive to hot regions
- Guaranteed error bounds

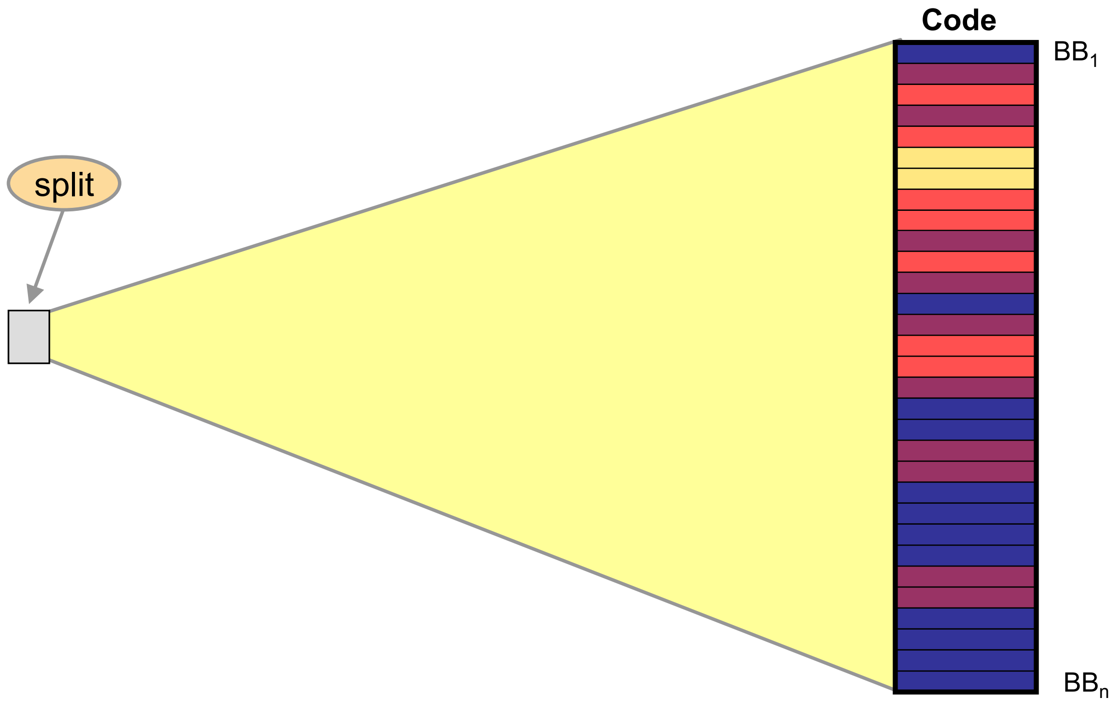
Any stream of profile data that can be divided hierarchically:

- Code profiling
- Values profiling
- Load address profiling
- Zero-value load profiling
- Narrow-width operand profiling

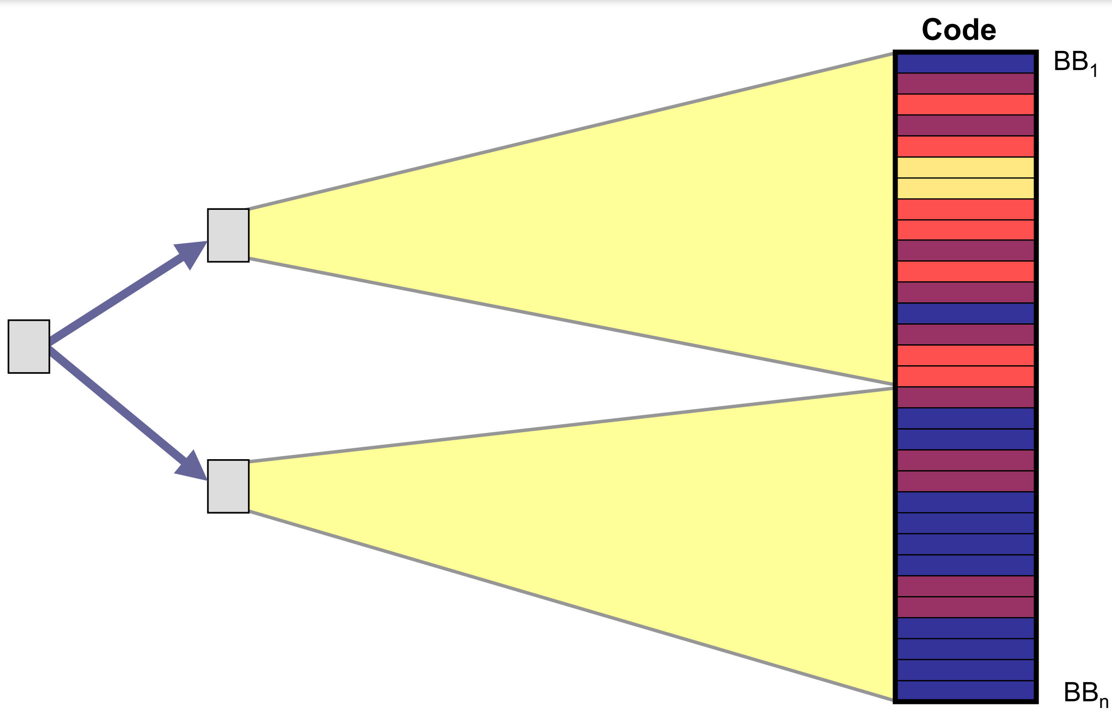
Outline

- Program Profiling
  - An example: Code profiles
  - Related work
- Range Adaptive Profiling
  - Advantages and Applications
  - Splits
  - Merges
- Making it efficient
  - Batching merges
  - Branching Factor
- RAP implementation
  - Results – Quantify error and memory
  - Hardware and Software
- Conclusions

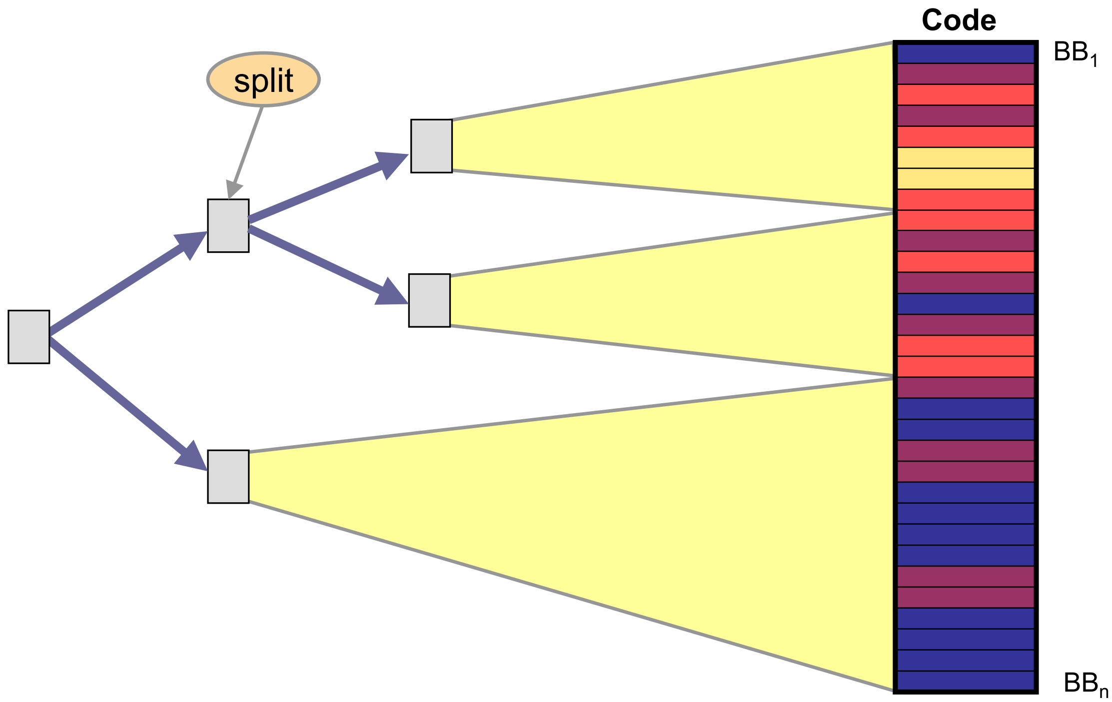
Adaptive Profiling - Splits



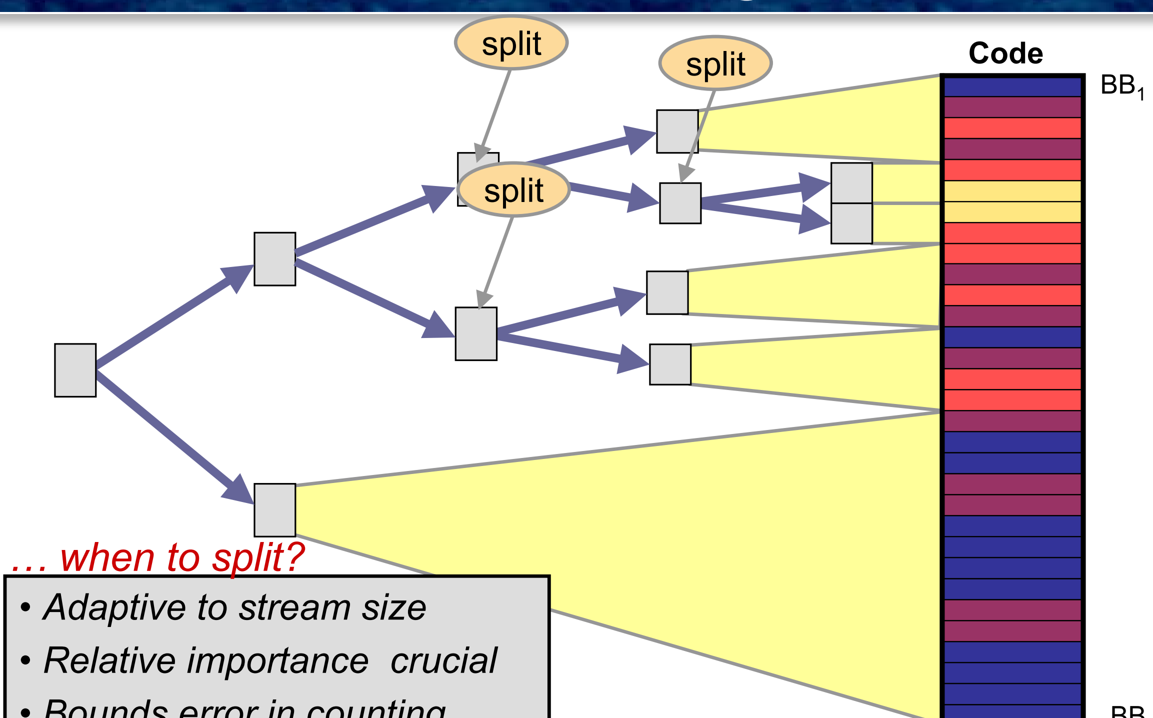
Adaptive Profiling - Splits



Adaptive Profiling - Splits



Adaptive Profiling - Splits

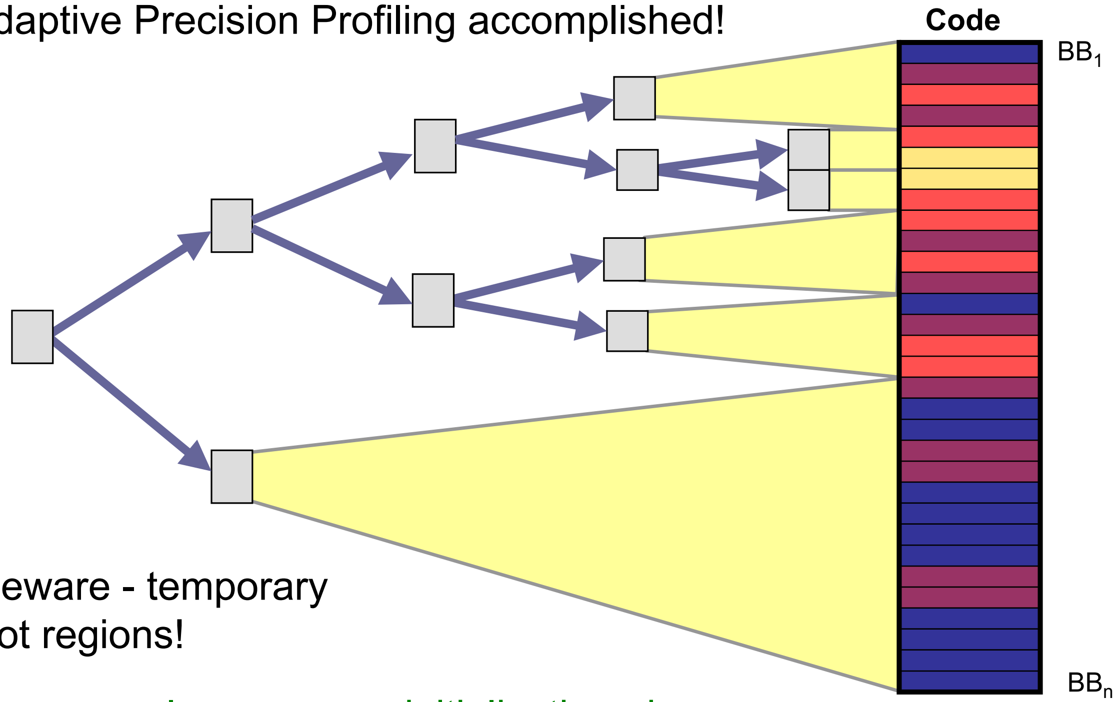


… when to split?

- Adaptive to stream size
- Relative importance crucial
- Bounds error in counting
- SplitThreshold = E.N/(log(R))

Adaptive Profiling - Merges

Adaptive Precision Profiling accomplished!

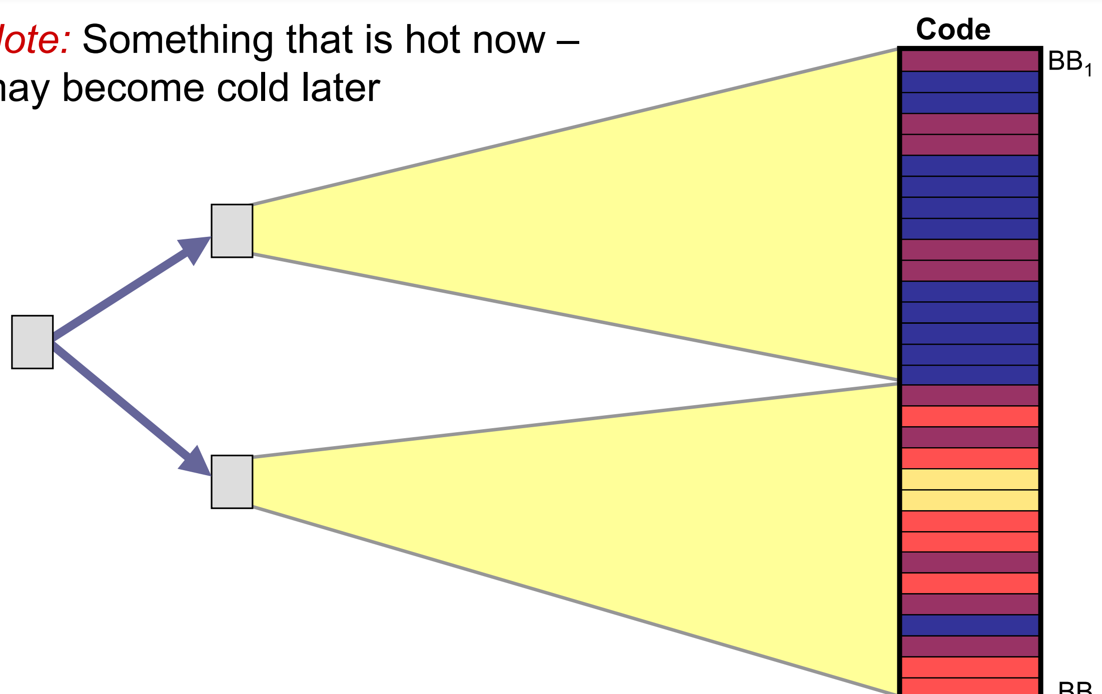


Beware - temporary hot regions!

For example – program initialization phase…

Adaptive Profiling - Splits

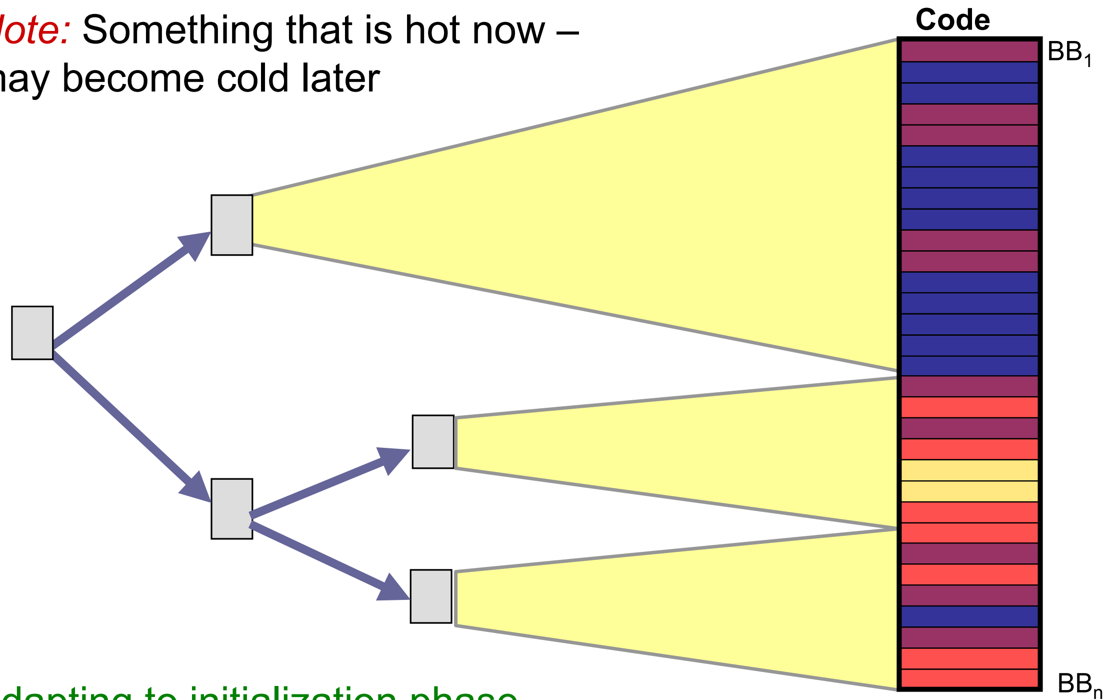
Note: Something that is hot now – may become cold later



Adapting to initialization phase …

Adaptive Profiling - Splits

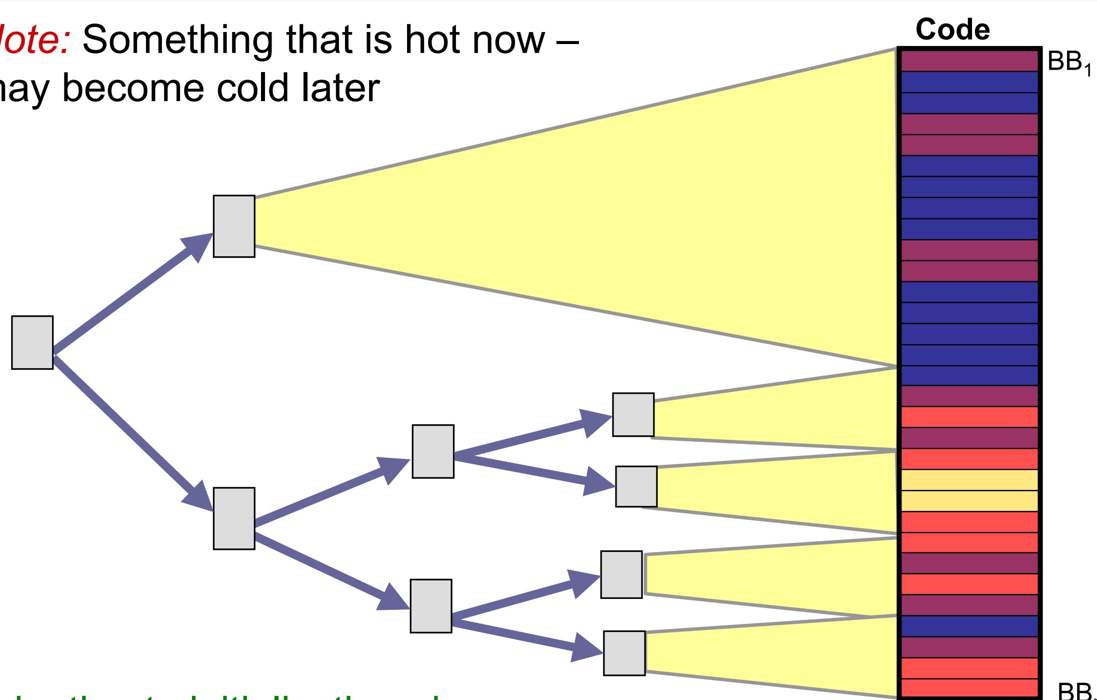
Note: Something that is hot now – may become cold later



Adapting to initialization phase …

Adaptive Profiling - Splits

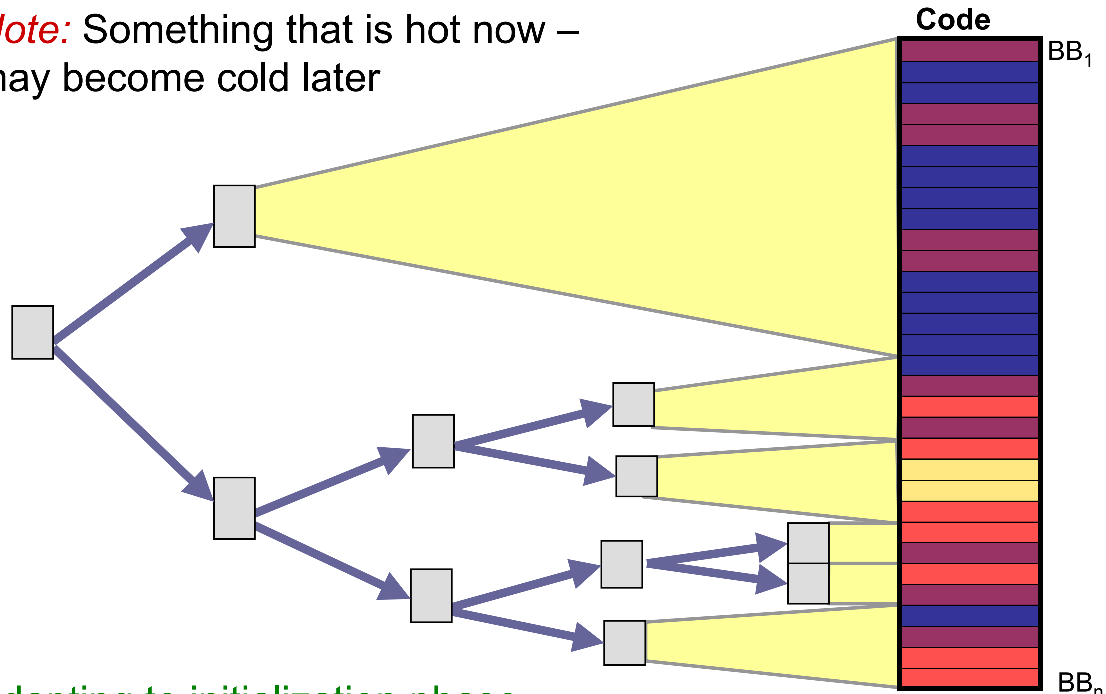
Note: Something that is hot now – may become cold later



Adapting to initialization phase …

Adaptive Profiling - Splits

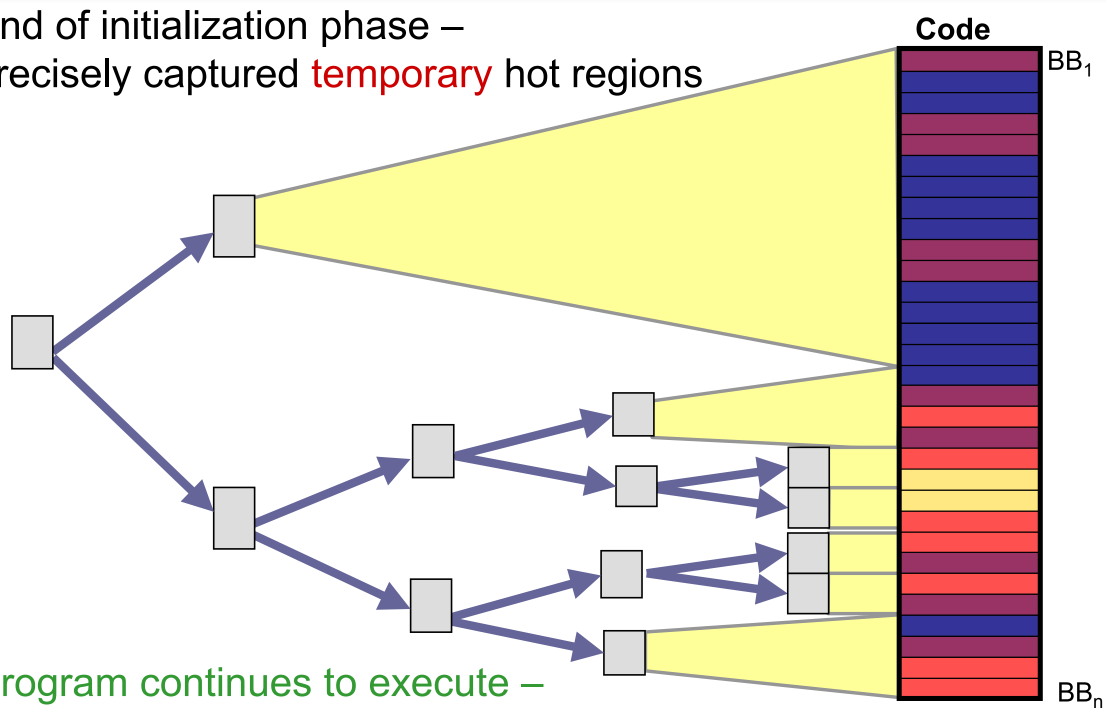
Note: Something that is hot now – may become cold later



Adapting to initialization phase …

Adaptive Profiling - Splits

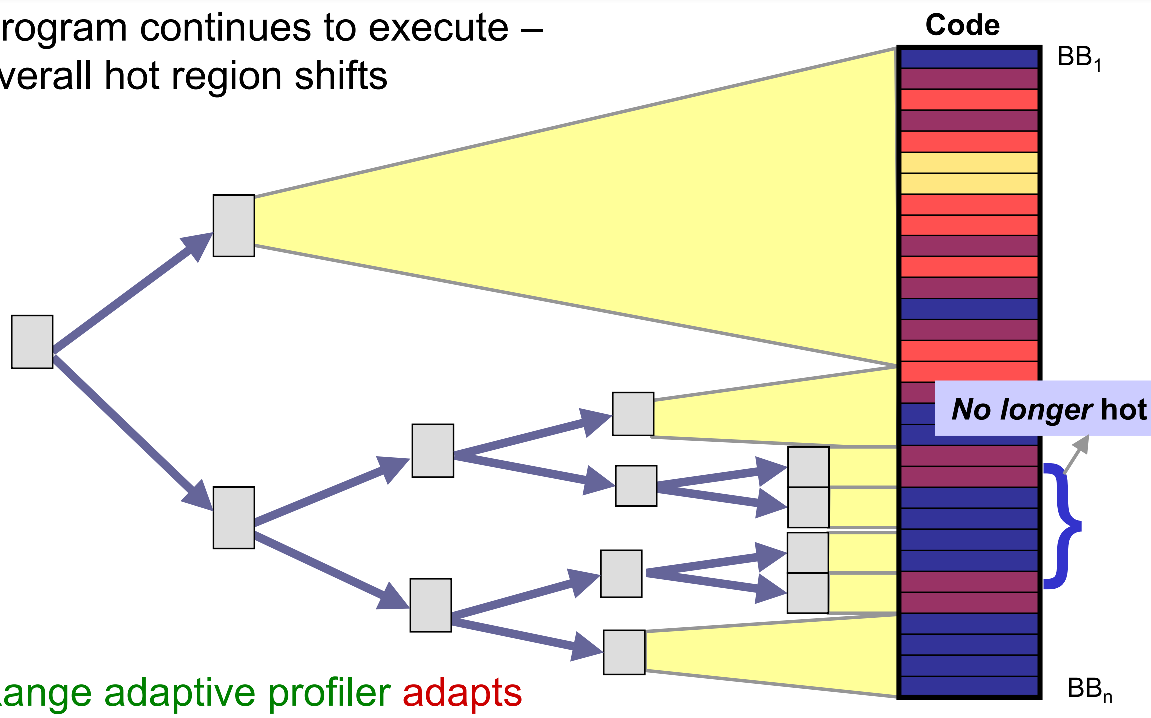
End of initialization phase – precisely captured temporary hot regions



Program continues to execute – overall hot region shifts …

Adaptive Profiling - Splits

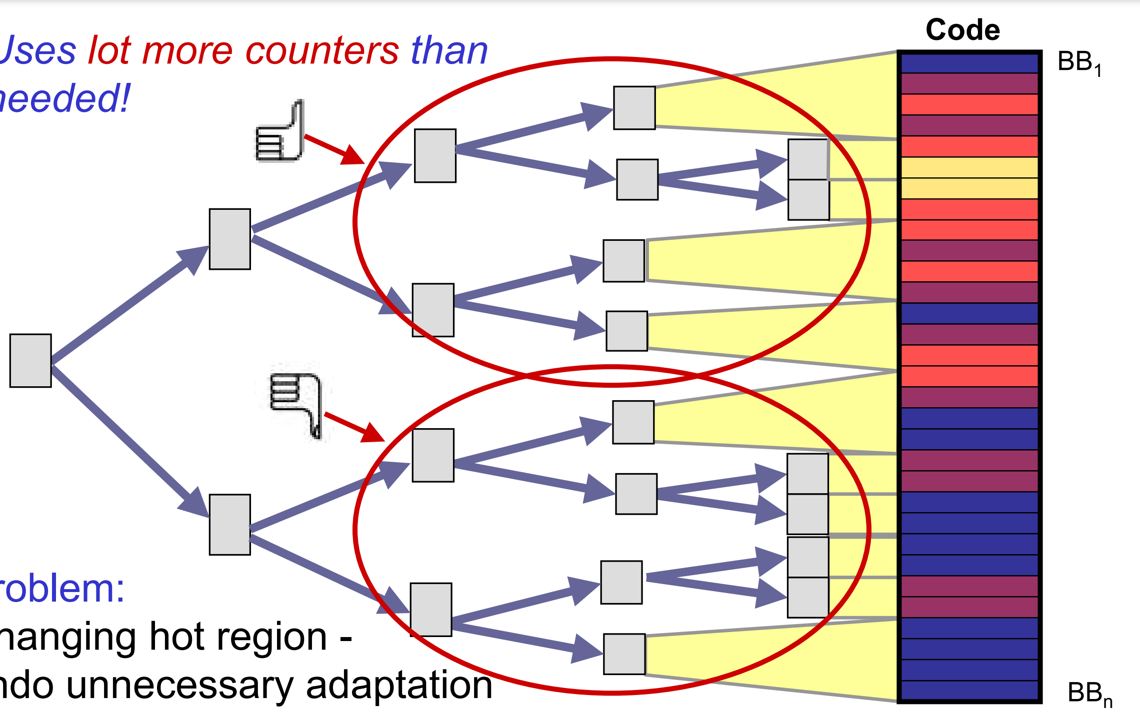
Program continues to execute – overall hot region shifts



Range adaptive profiler adapts to the new hot region

Adaptive Profiling - Merges

Uses lot more counters than needed!



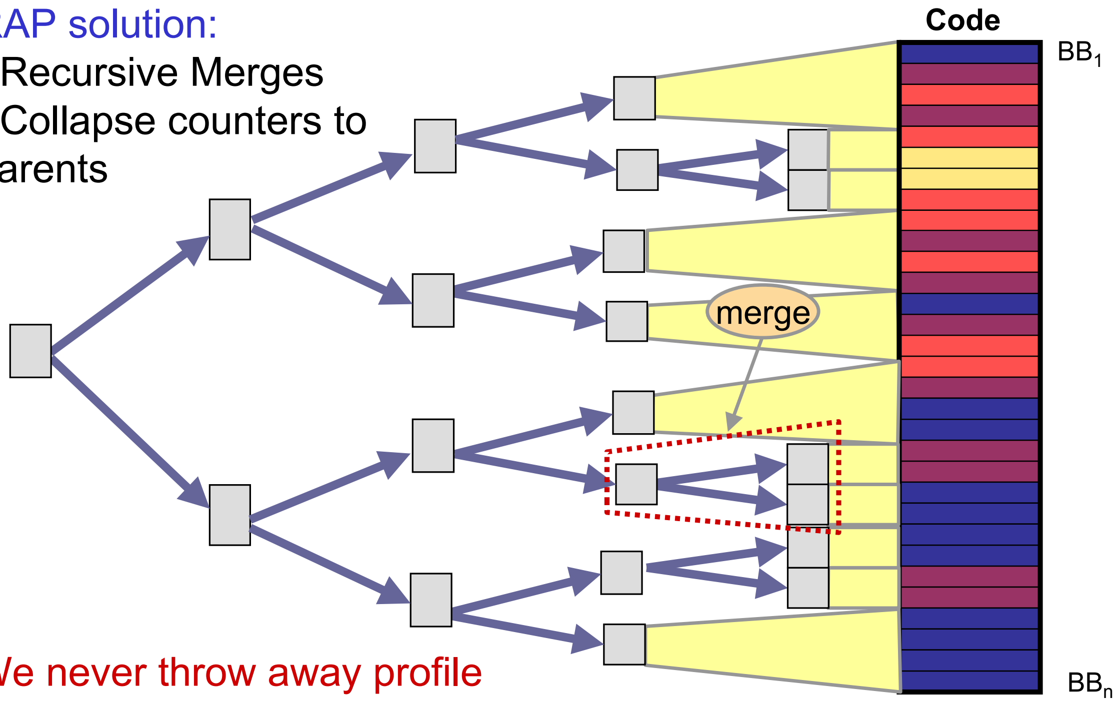
Problem:
Changing hot region - undo unnecessary adaptation

Merge '*non-hot* counters' …

Adaptive Profiling - Merges

RAP solution:

- Recursive Merges
- Collapse counters to parents

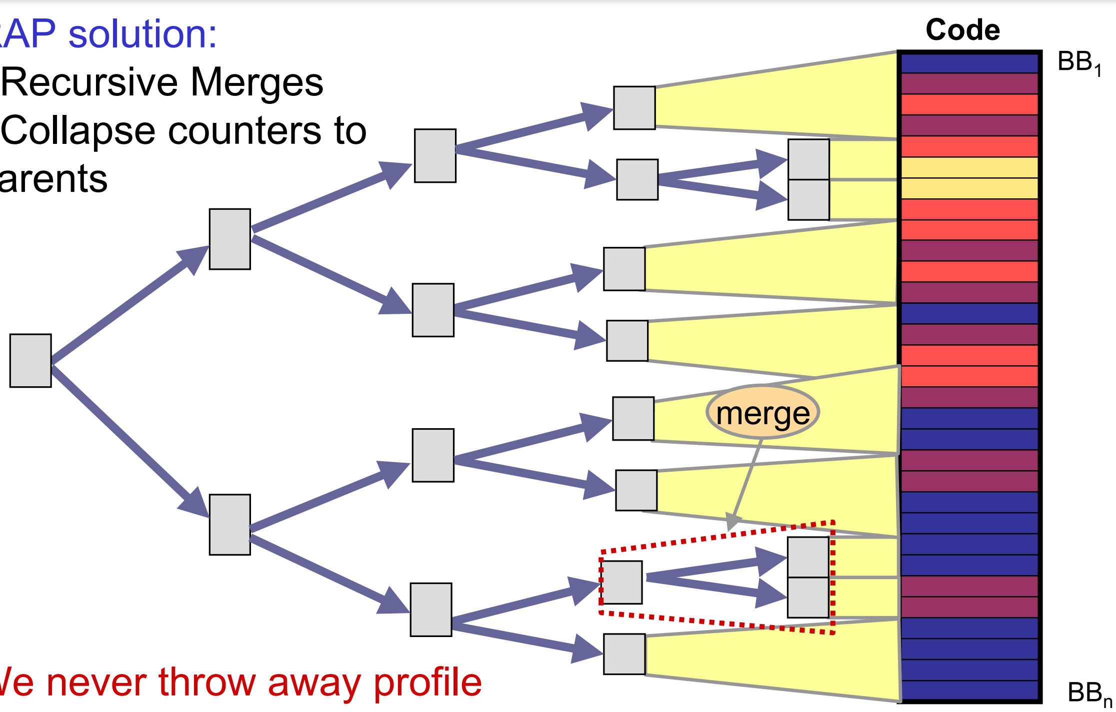


We never throw away profile information, we only merge

Adaptive Profiling - Merges

RAP solution:

- Recursive Merges
- Collapse counters to parents

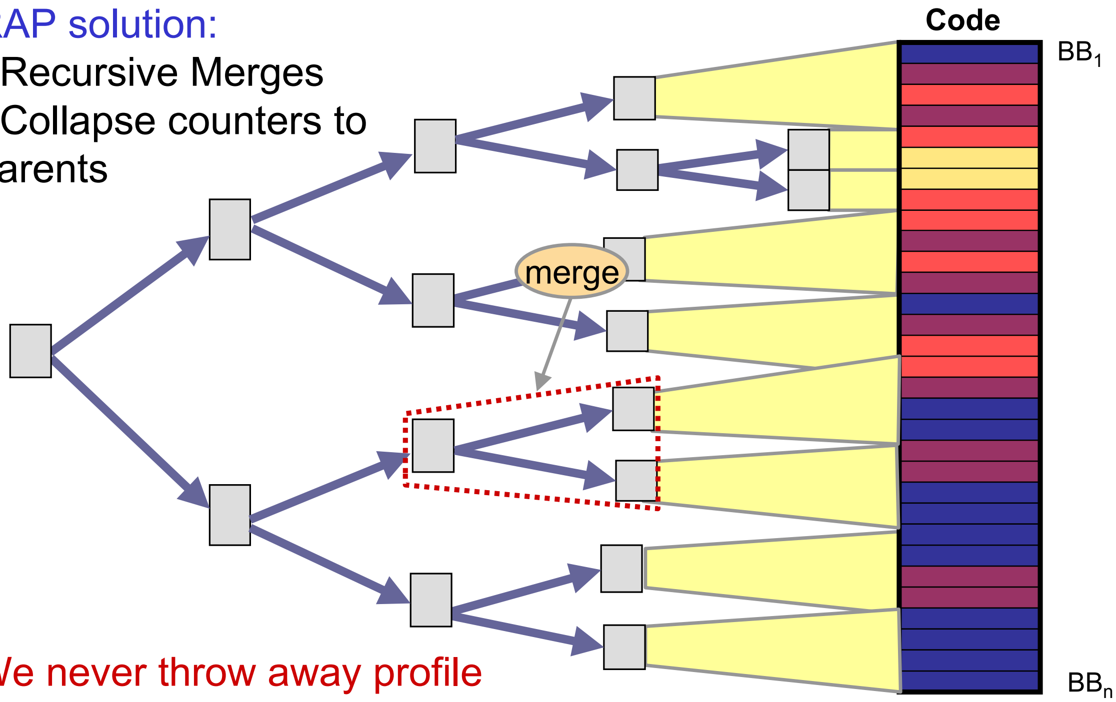


We never throw away profile information, we only merge

Adaptive Profiling - Merges

RAP solution:

- Recursive Merges
- Collapse counters to parents

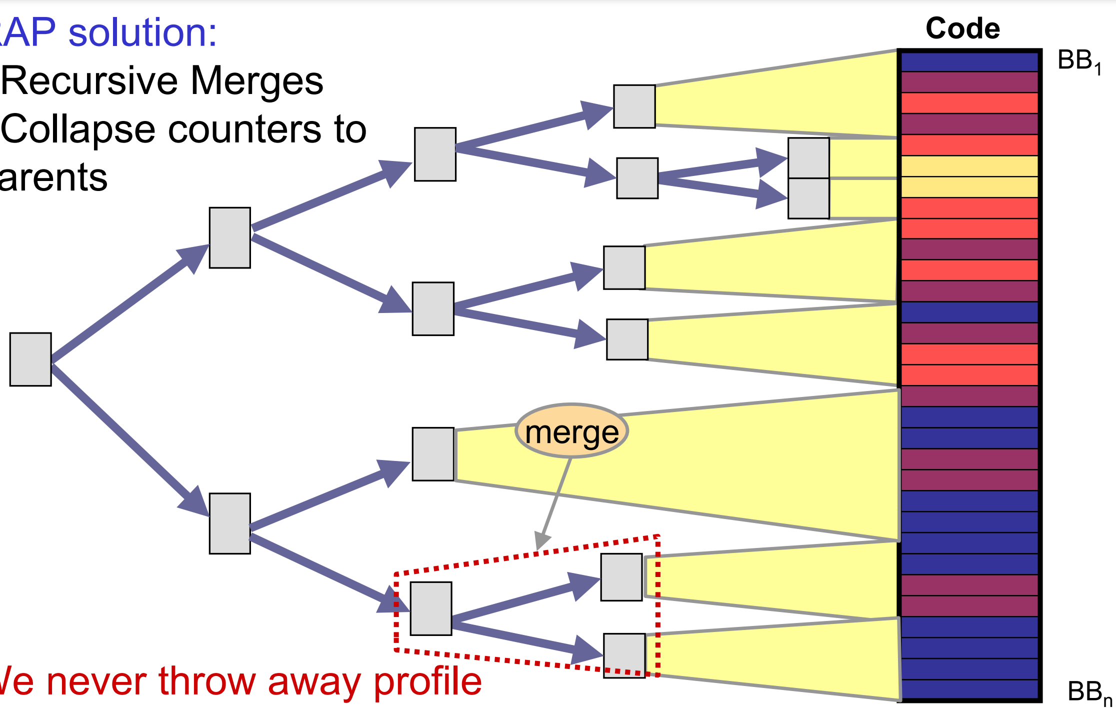


We never throw away profile information, we only merge

Adaptive Profiling - Merges

RAP solution:

- Recursive Merges
- Collapse counters to parents

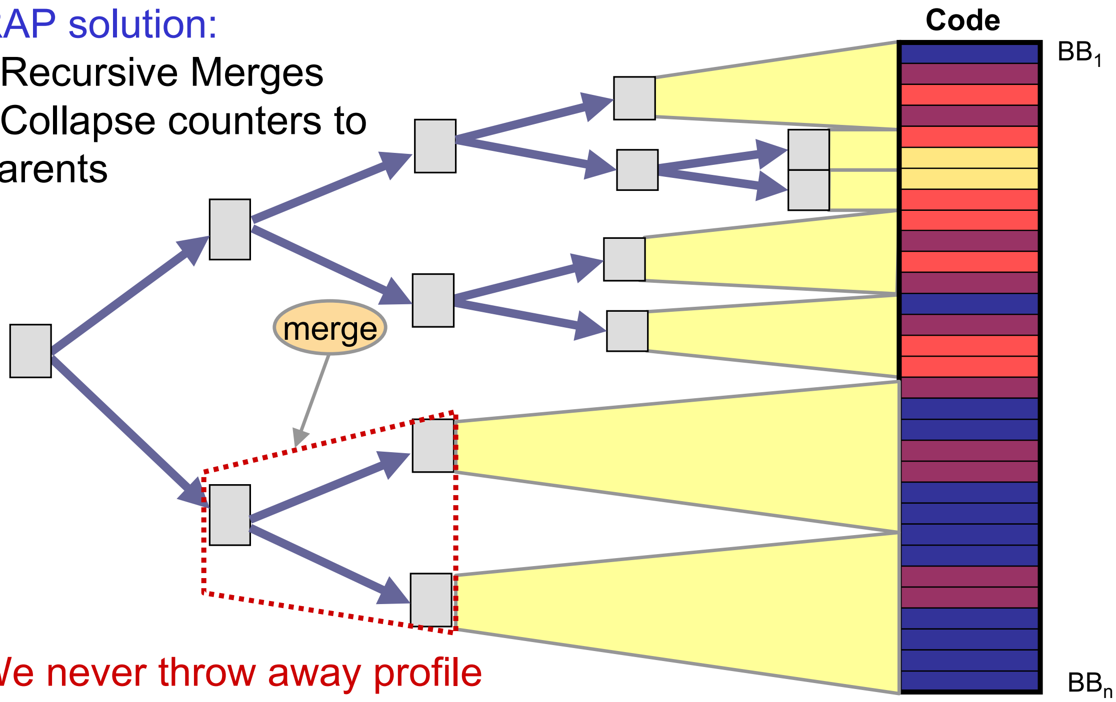


We never throw away profile information, we only merge

Adaptive Profiling - Merges

RAP solution:

- Recursive Merges
- Collapse counters to parents

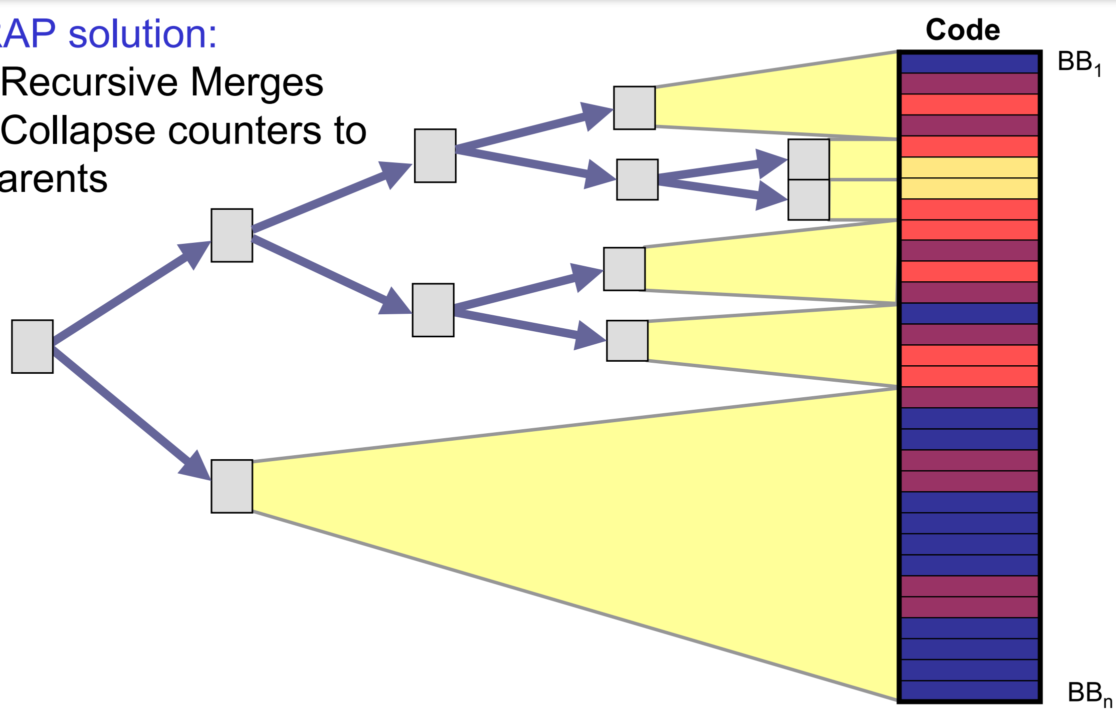


We never throw away profile information, we only merge

Adaptive Profiling - Merges

RAP solution:

- Recursive Merges
- Collapse counters to parents



Range Adaptive Profiling

Advantages

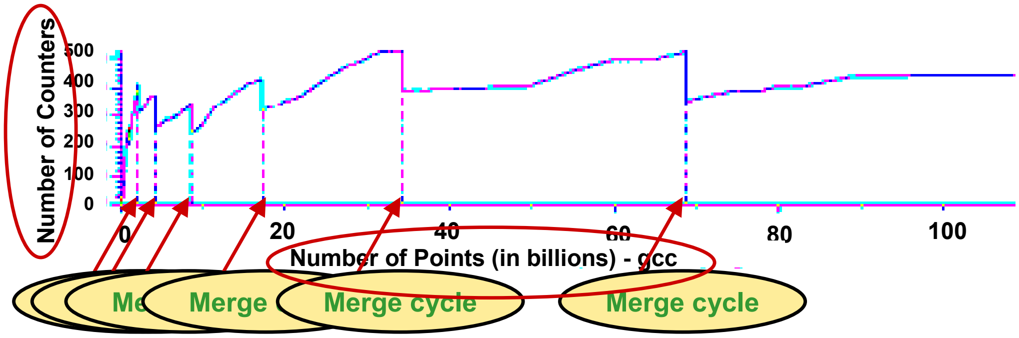
- Precision dynamically adaptive to hot regions
- Guaranteed error bounds
- Optimal usage of a few counters

- Plus -
  - Independent of the stream size
  - Independent of the stream order

Outline

- Program Profiling
  - An example: Code profiles
  - Related work
- Range Adaptive Profiling
  - Advantages and Applications
  - Splits
  - Merges
- Making it efficient
  - Batching merges
  - Branching Factor
- RAP implementation
  - Results – Quantify error and memory
  - Hardware and Software
- Conclusions

Batched Merges

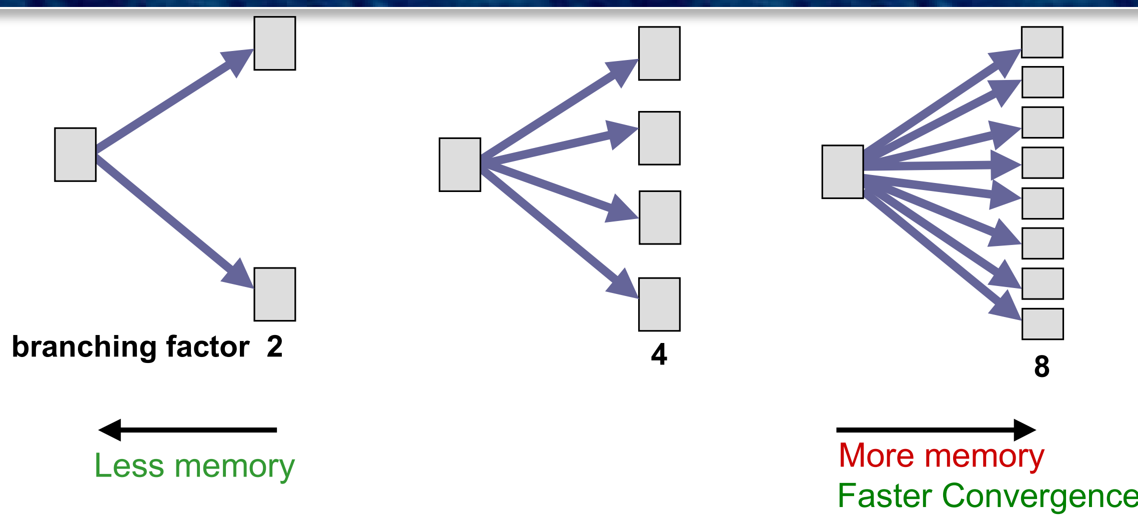


RAP tree does not grow faster than *logarithmic* rate

When do we initiate a merge cycle:

- Periodic merging
- Exponentially increasing periods

Branching Factor



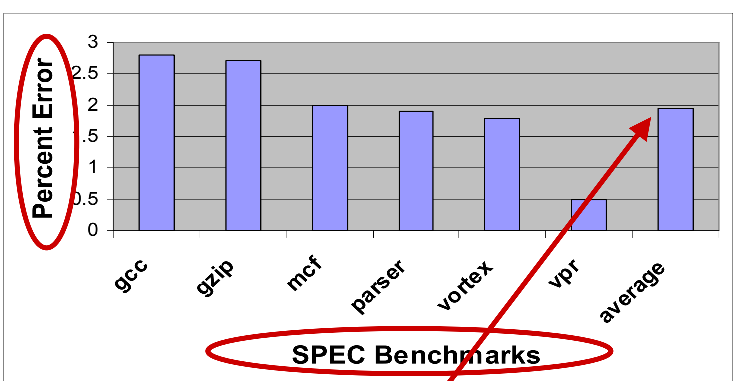
- We show that optimal branching factor is four

Outline

- Program Profiling
  - An example: Code profiles
  - Related work
- Range Adaptive Profiling
  - Advantages and Applications
  - Splits
  - Merges
- Making it efficient
  - Batching merges
  - Branching Factor
- RAP implementation
  - Results – Quantify error and memory
  - Hardware and Software
- Conclusions

Results

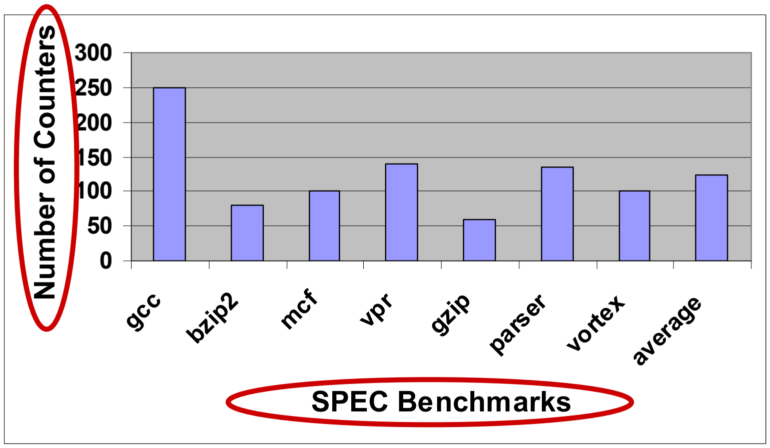
- Range adaptive profiler
  - Online technique
  - Does not have ideal knowledge – counts everything
  - Error introduced by not splitting early enough



Average percent error less than 2%

Results

High accuracy – but at what cost?



- On an average - 150 counters provides 99% accurate information on code profiles

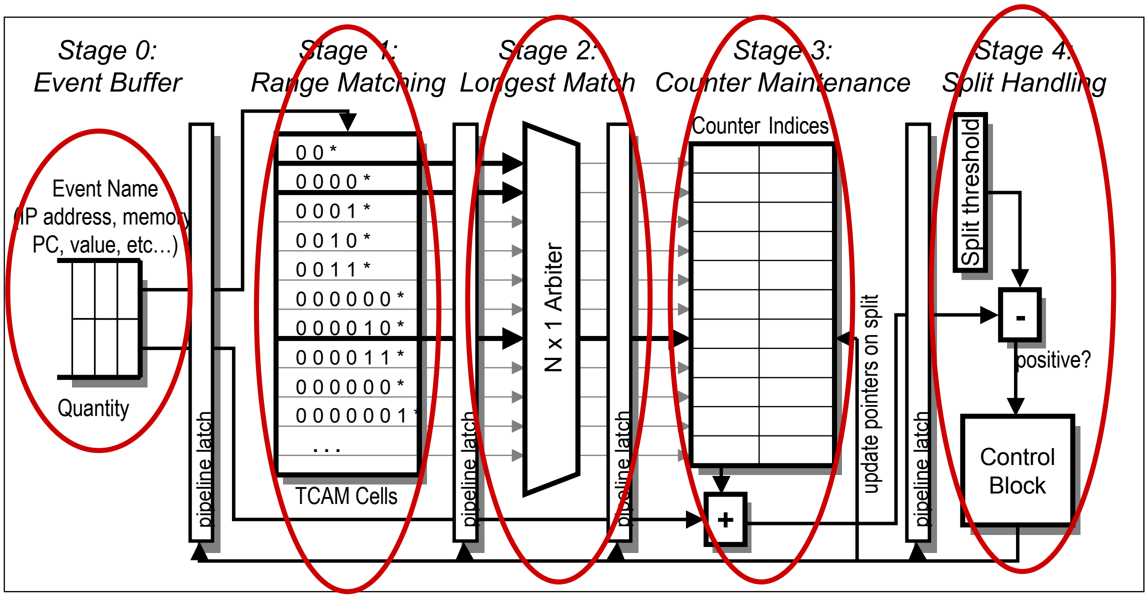
We show more results in the paper

Software implementation

- Simple set of APIs
  - Offline and online profiling
  - *rap_init*
  - *rap_add_points* – builds the RAP tree, takes care of splits and merges too.
  - *rap_finalize*
- Webpage
  - www.cs.ucsb.edu/~arch/rap

Extremely high throughput profile data analysis …

Hardware Profiling Engine



Conclusion

- Range Adaptive Profiling
  – Summarizes high bandwidth profile data
  – Fully streaming scheme
  – Bounded memory and error
  – General purpose – high applicability

  – Multi-dimensional Profiling

Future Work: Multidimensional Profiling

Code-mem

Edges

Code-val

Val-mem

Thank You

Profiling over Adaptive Ranges -
http://www.cs.ucsb.edu/~arch/rap
http://www.cs.ucsb.edu/~shashimc