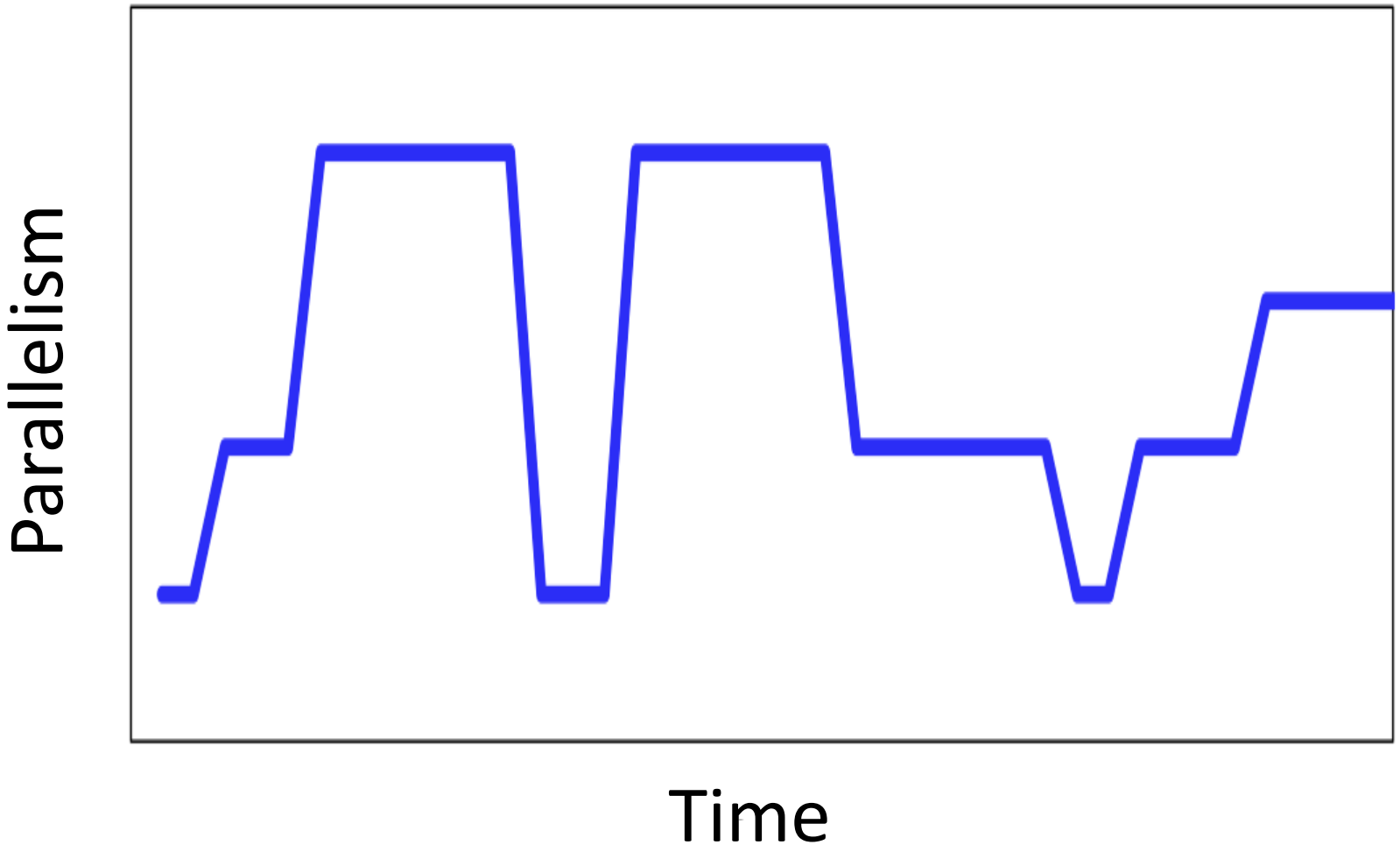
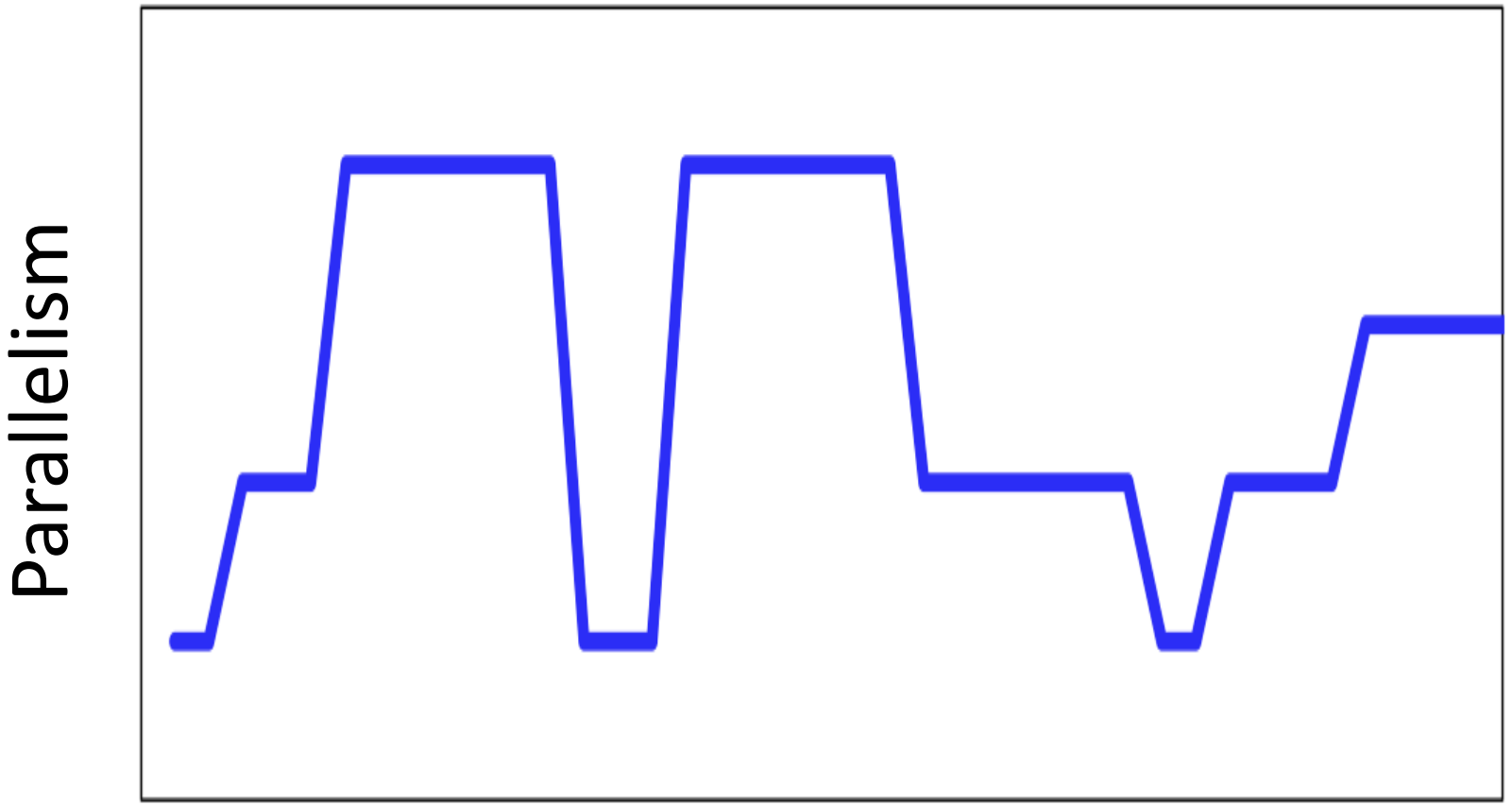
Harmony: Collection and Analysis of Parallel Block Vectors

Melanie Kambadur

Kui Tang, Martha Kim

Columbia University

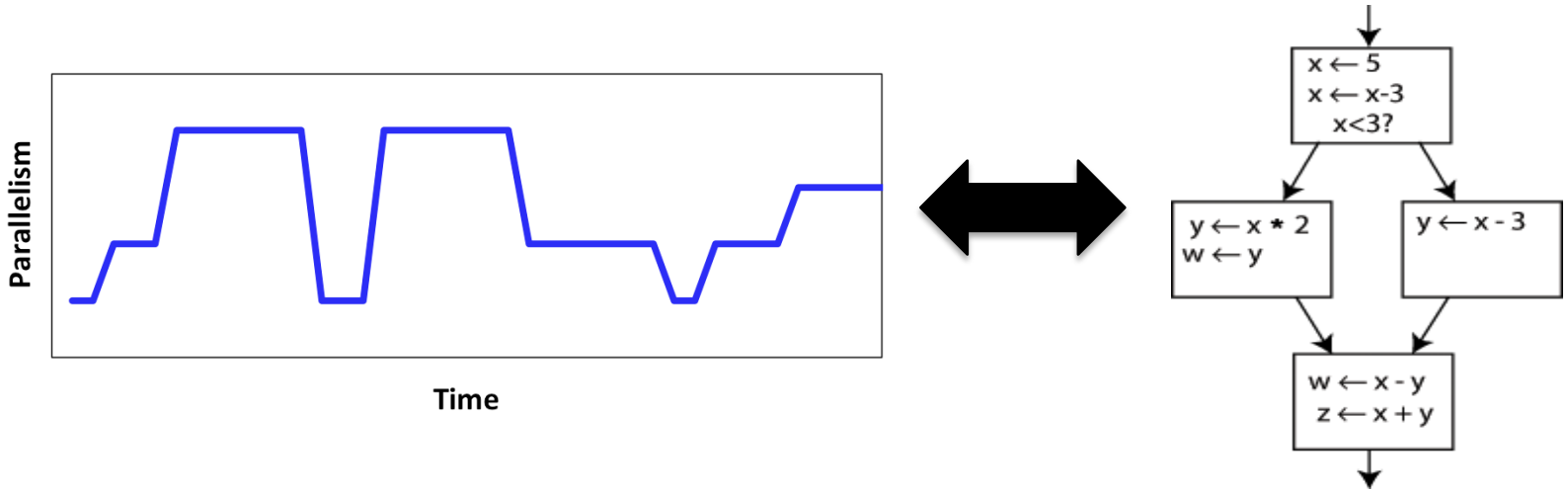

Parallelism
Time



~~Time~~

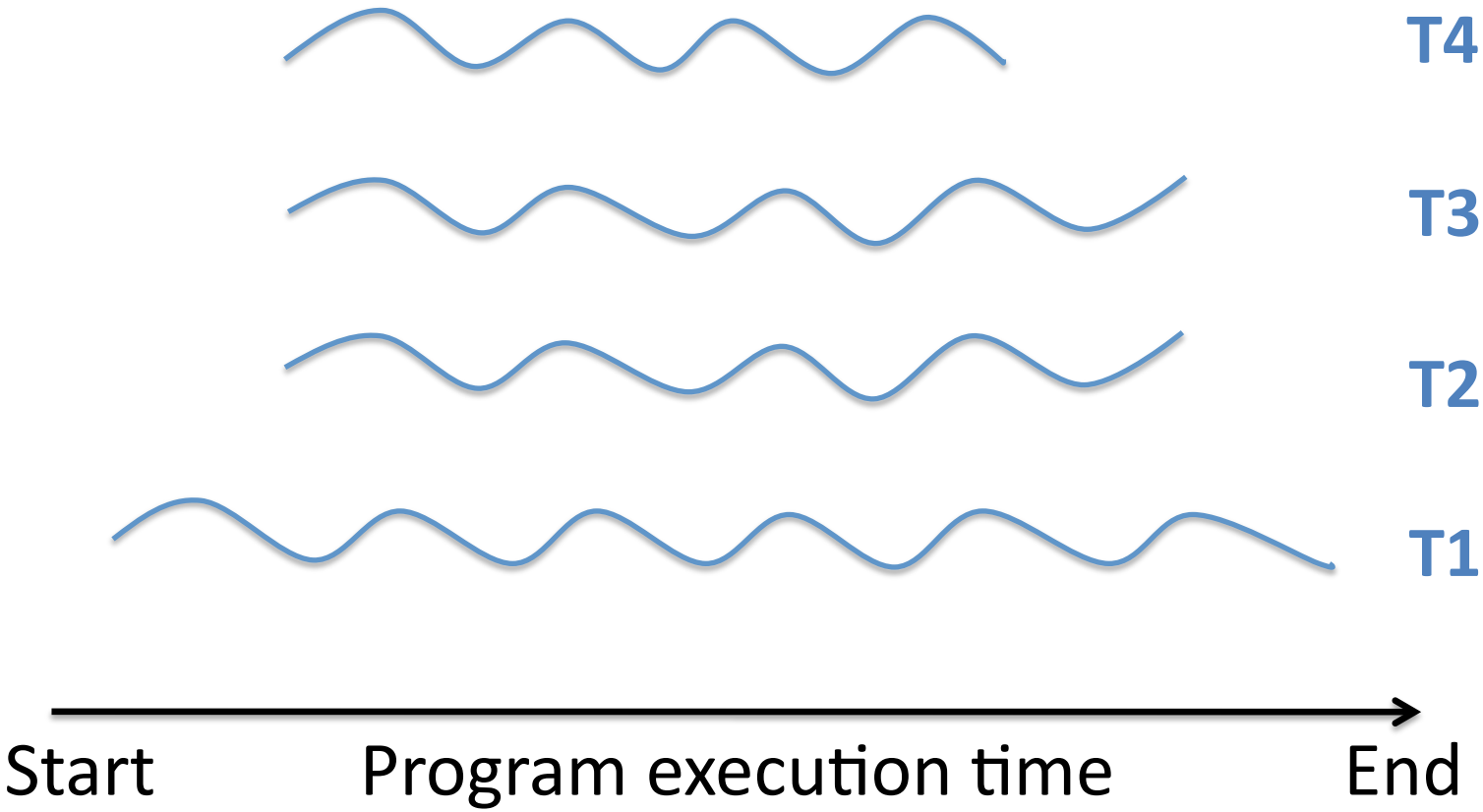
Code currently executing

Parallel Block Vector Profiles (PBVs)

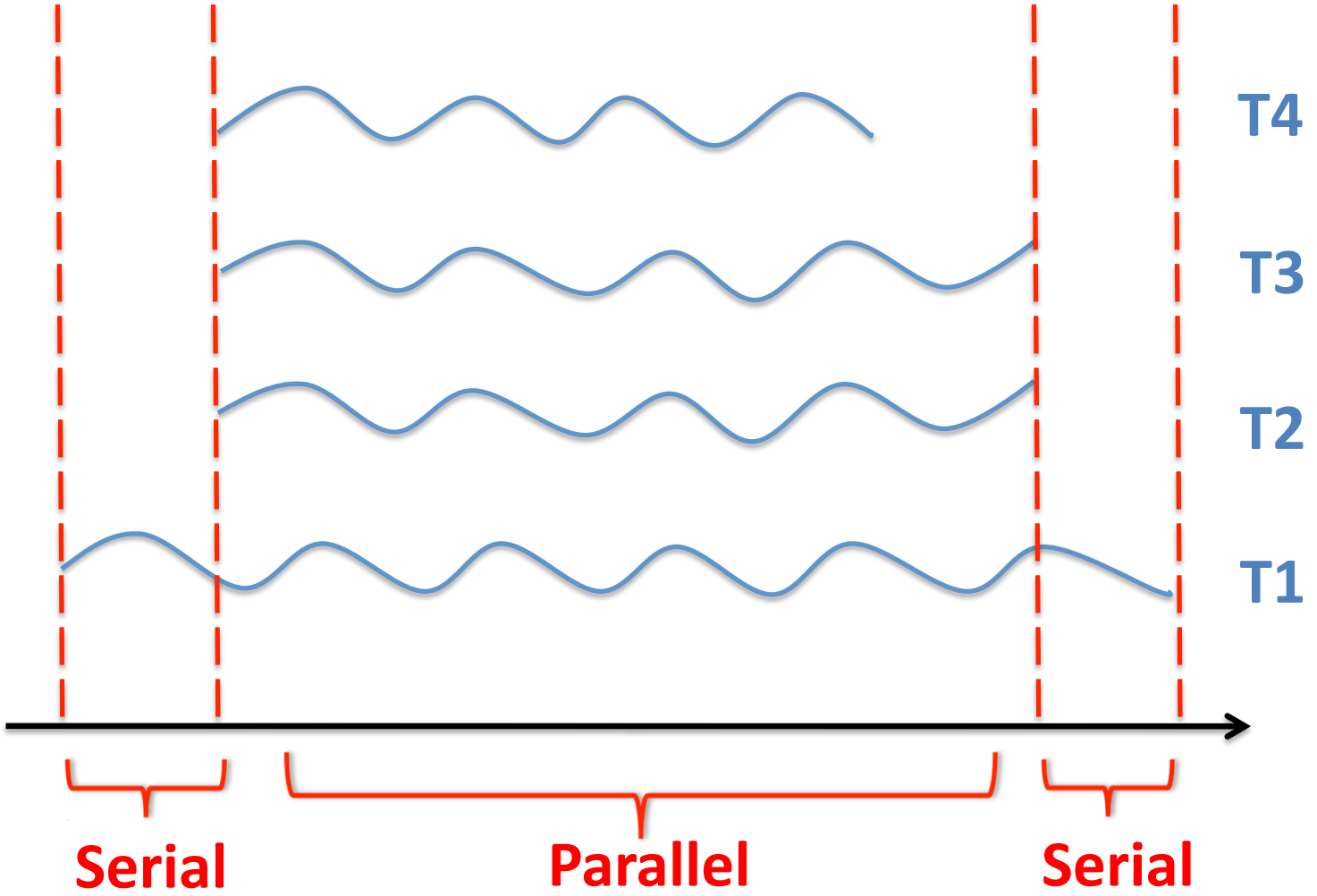


- Maps between dynamic parallelism and program code
- Harmony: low overhead collection of PBVs

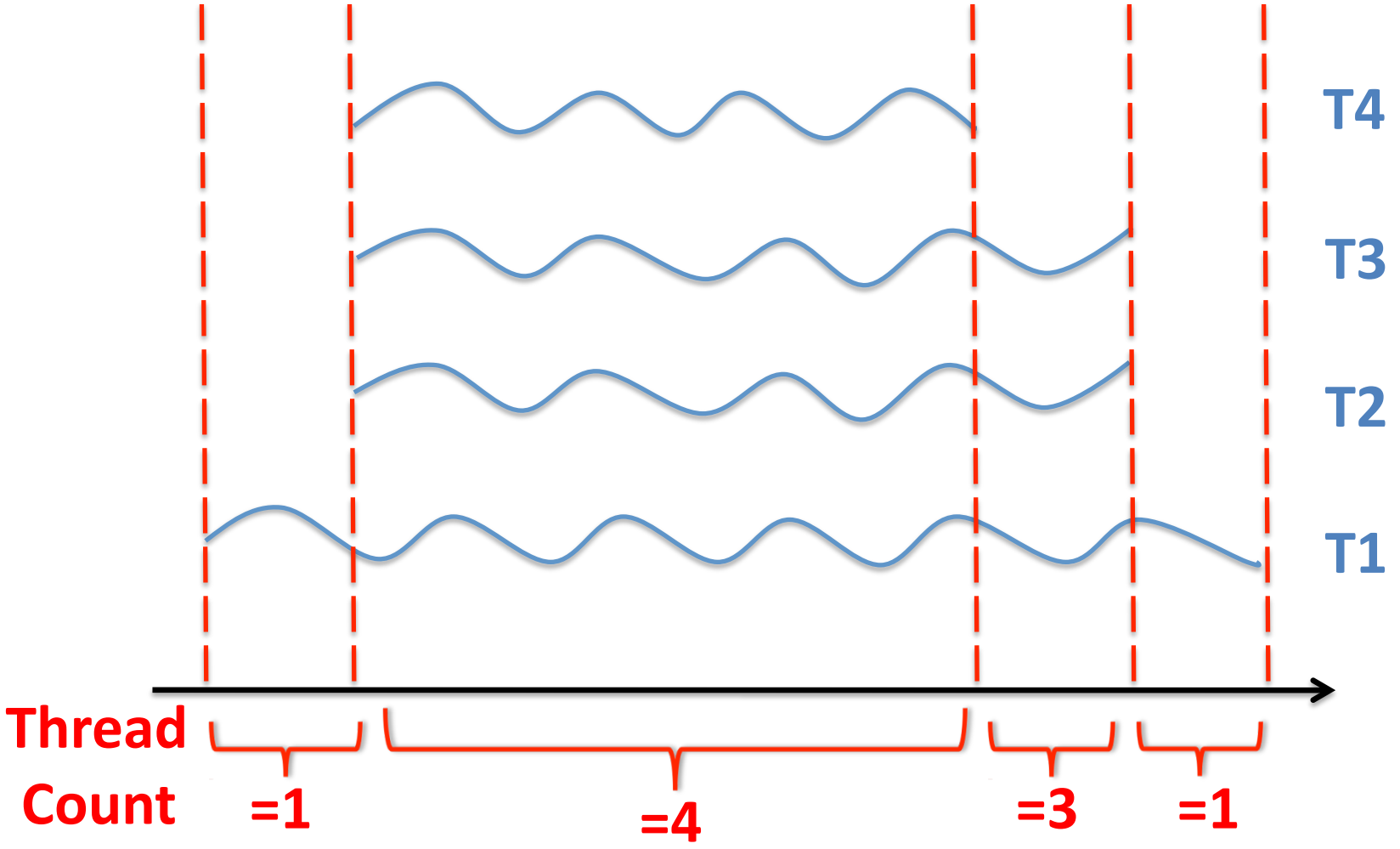
In a multithreaded program execution…



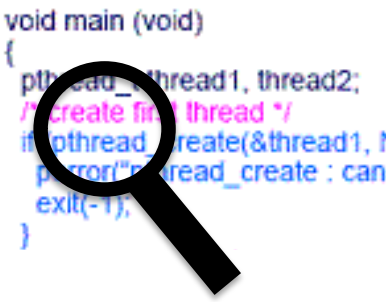
…there are parallel and serial phases



To see how parallel, count threads at each change:



What can already be measured?

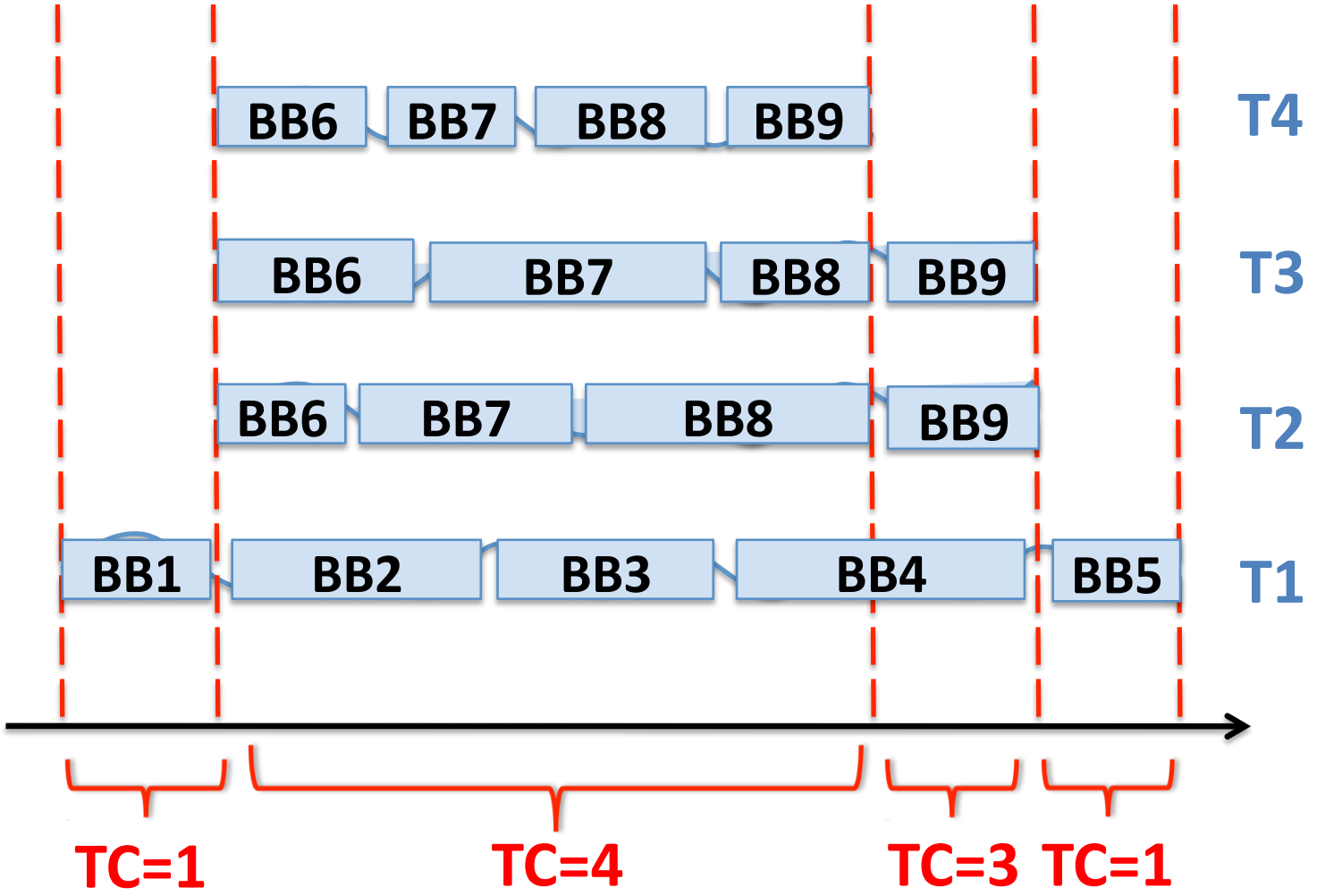


Static analysis can track thread creation and destruction calls

Tools like Vtune, Gprof + Quartz give the dynamic length of each phase

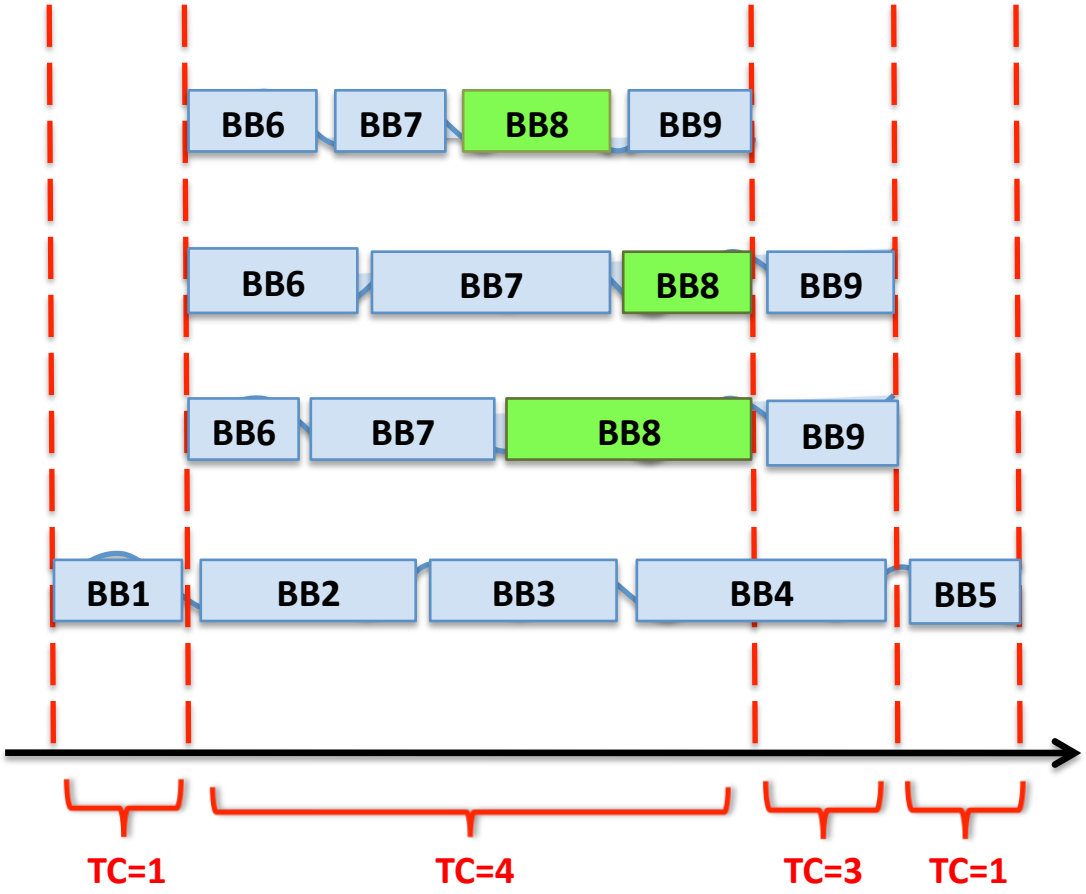
No way to link dynamic parallelism phases and code

What parts of the program execute in different parallelism phases?



PBVs count block execs per parallelism phase

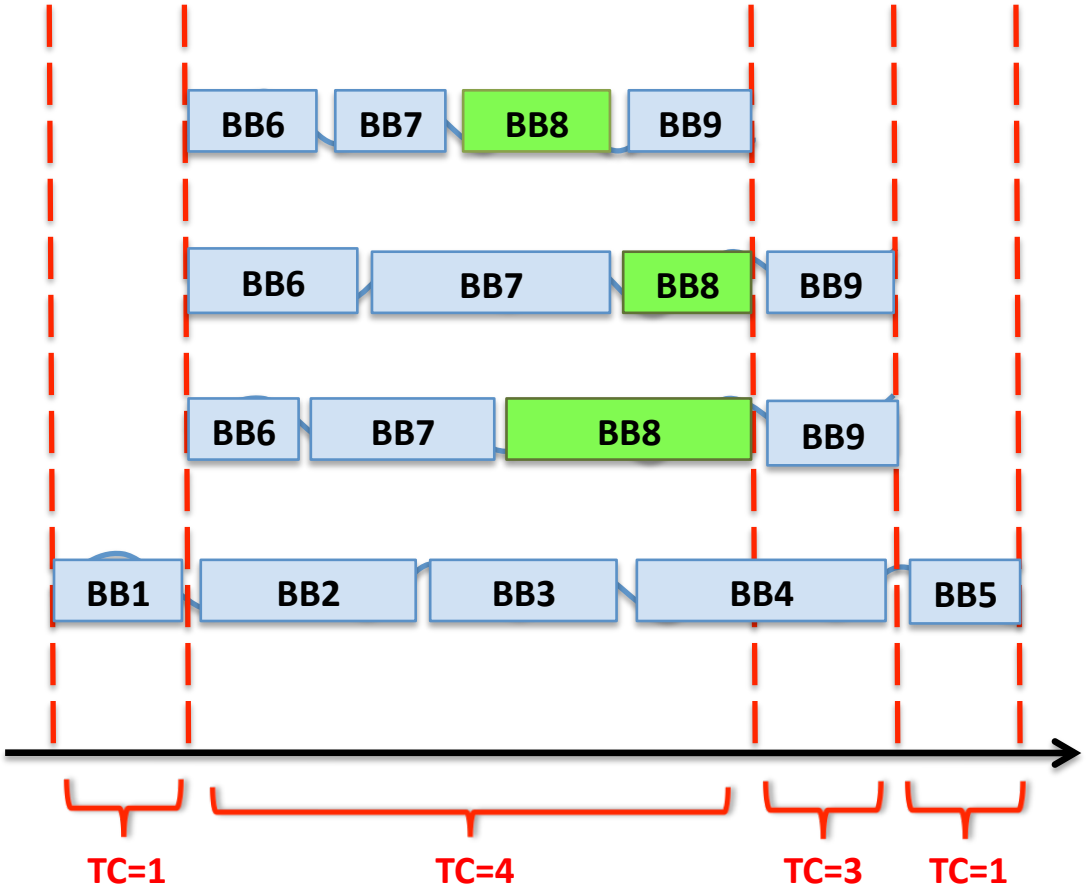
Dynamic Execution → **Parallel Block Vectors**



TC=	1	2	3	4
BB1				
BB2				
BB3				
BB4				
BB5				
BB6				
BB7				
BB8	0	0	0	3
BB9				

PBVs count block execs per parallelism phase

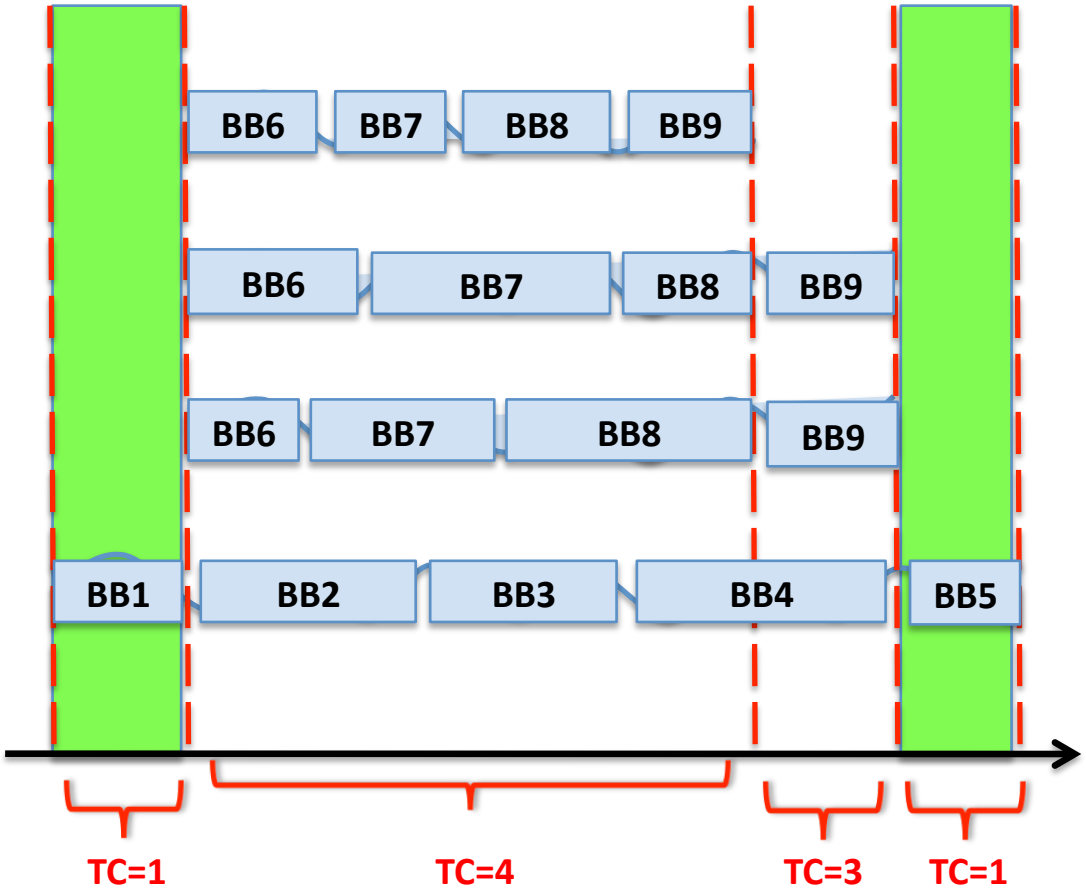
Dynamic Execution → **Parallel Block Vectors**



TC=	1	2	3	4
BB1	1	0	0	0
BB2	0	0	0	1
BB3	0	0	0	1
BB4	0	0	0	1
BB5	1	0	0	0
BB6	0	0	0	3
BB7	0	0	0	3
BB8	0	0	0	3
BB9	0	0	1	2

PBVs also show which blocks execute in specific parallelism phases

Dynamic Execution → **Parallel Block Vectors**



TC=	1	2	3	4
BB1	1	0	0	0
BB2	0	0	0	1
BB3	0	0	0	1
BB4	0	0	0	1
BB5	1	0	0	0
BB6	0	0	0	3
BB7	0	0	0	3
BB8	0	0	0	3
BB9	0	0	1	2

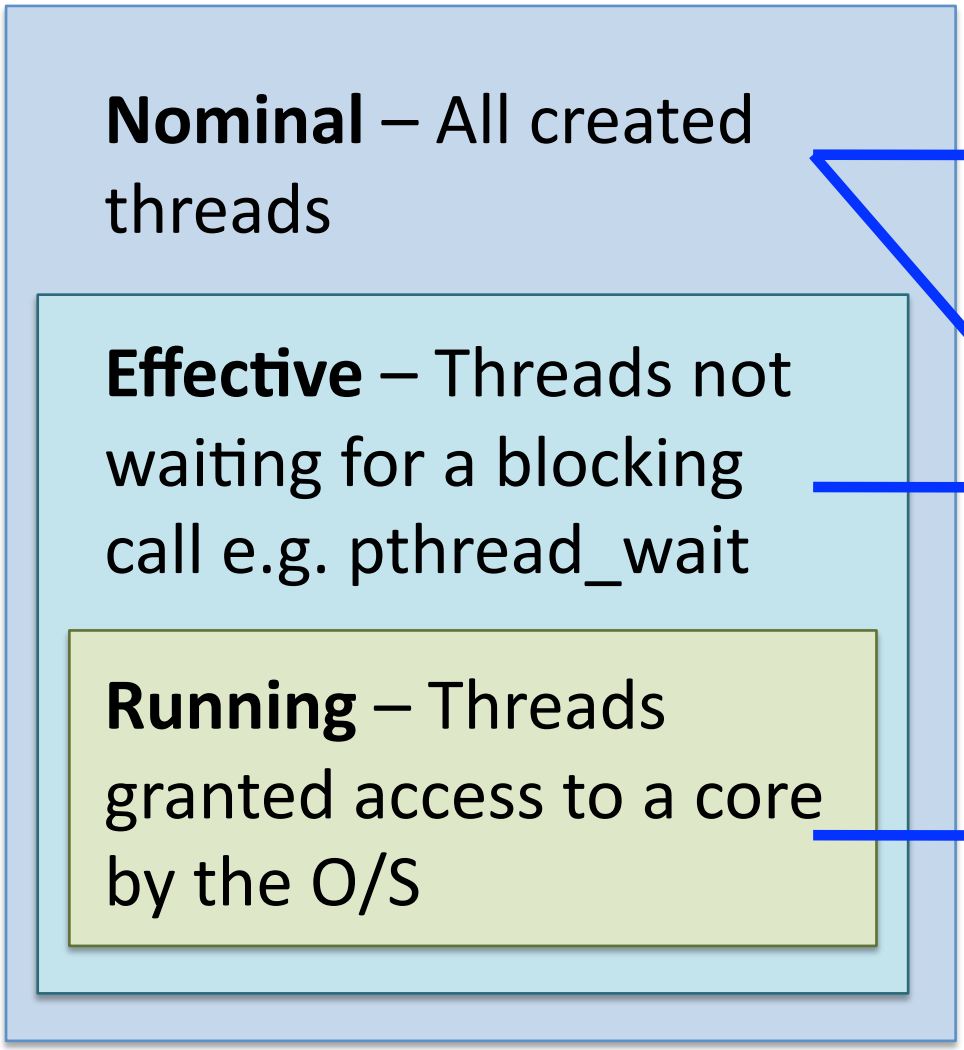
How exactly should we count threads?

Nominal – All created threads

Effective – Threads not waiting for a blocking call e.g. pthread_wait

Running – Threads granted access to a core by the O/S

How exactly should we count threads?



See how programmer intent matches dynamic run

Diff. with nominal to see blocking call slowdown

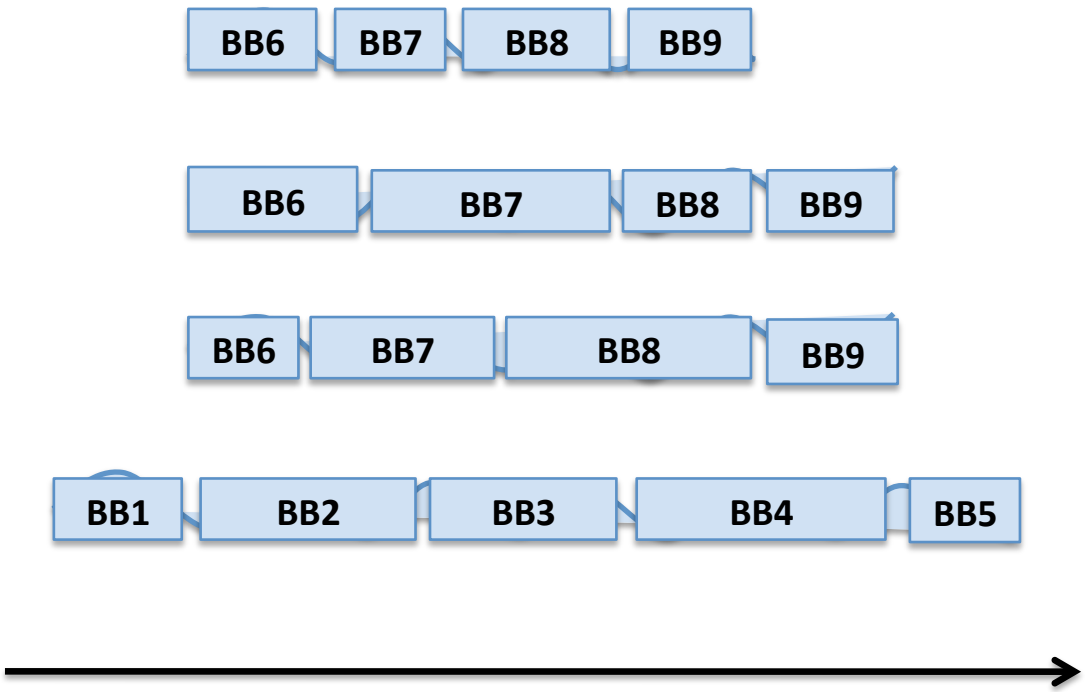
More specific count if there are more threads than cores

Harmony: A tool for PBV Collection

- **Downloadable tool:**

  arcade.cs.columbia.edu/harmony
- **LLVM Pass –** easy for people to use, just run compile program then run normally
- **Precise, not sampled –** otherwise might not see outliers
- **Optimized for low overheads –** don't want to perturb parallel programs!

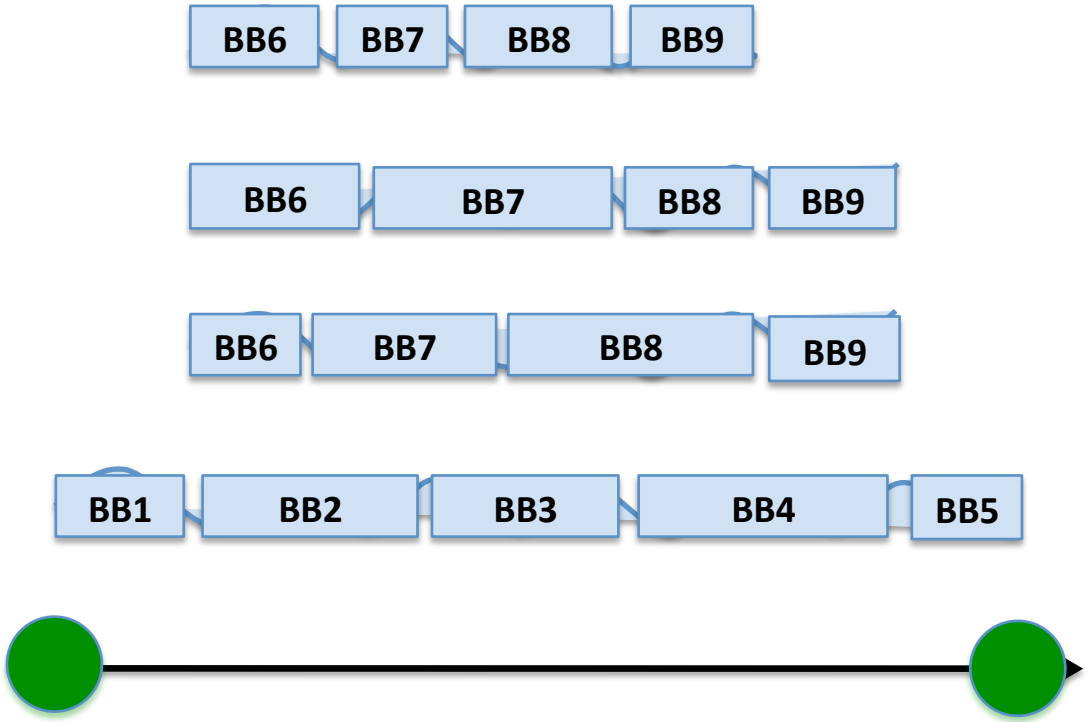
Collecting PBV Profiles



Main Idea:

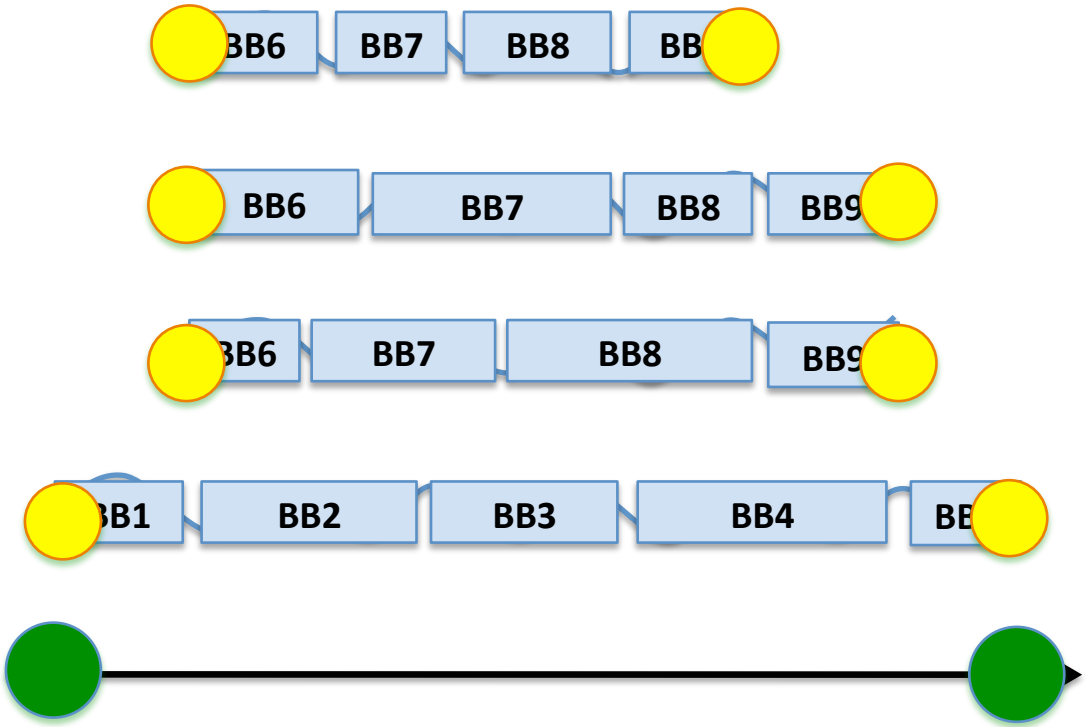
Low overhead collection via statically inserted instrumentation points to collect data dynamically upon execution.

Collecting PBV Profiles



- At program start and end: *create data structures*

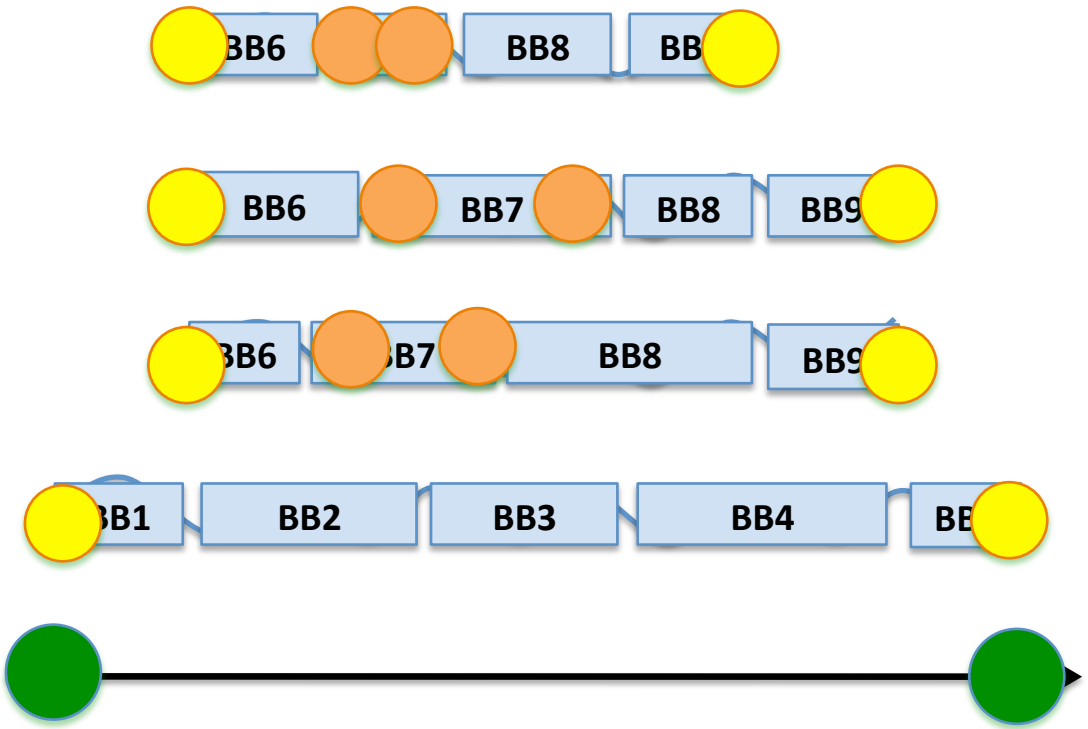
Collecting PBV Profiles



At program start and end

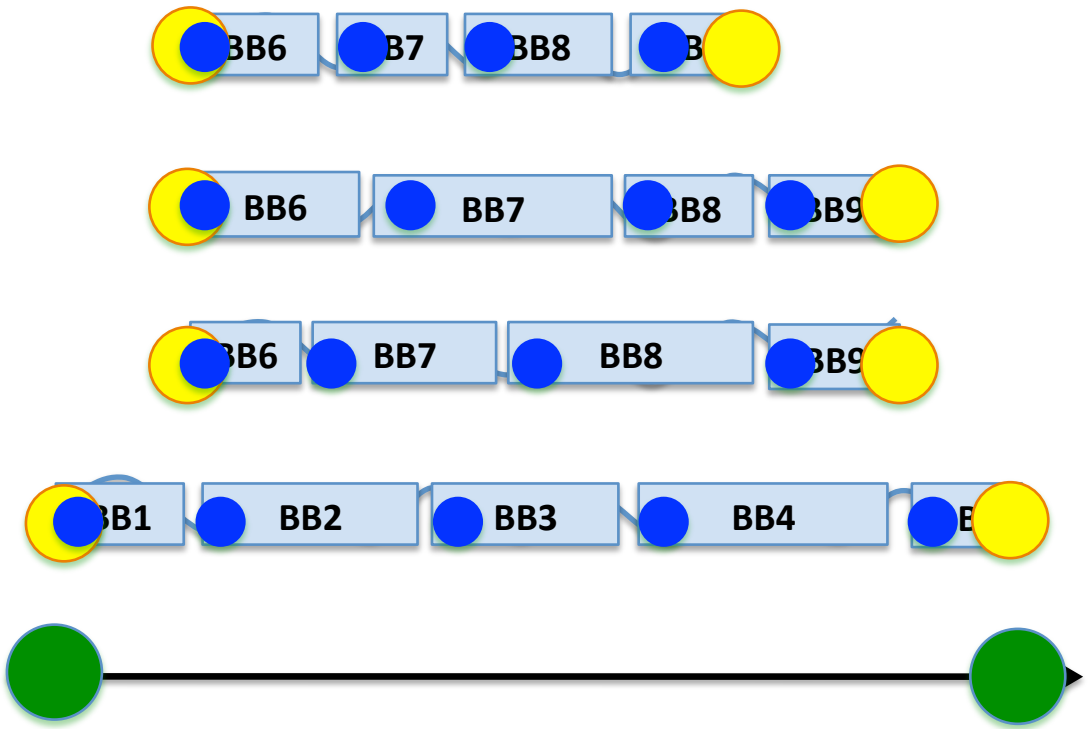
At thread create and exit: *adjust global thread counts*

Collecting PBV Profiles



- At program start and end
- At thread create and exit
- At blocking calls: *adjust global thread counts*

Collecting PBV Profiles



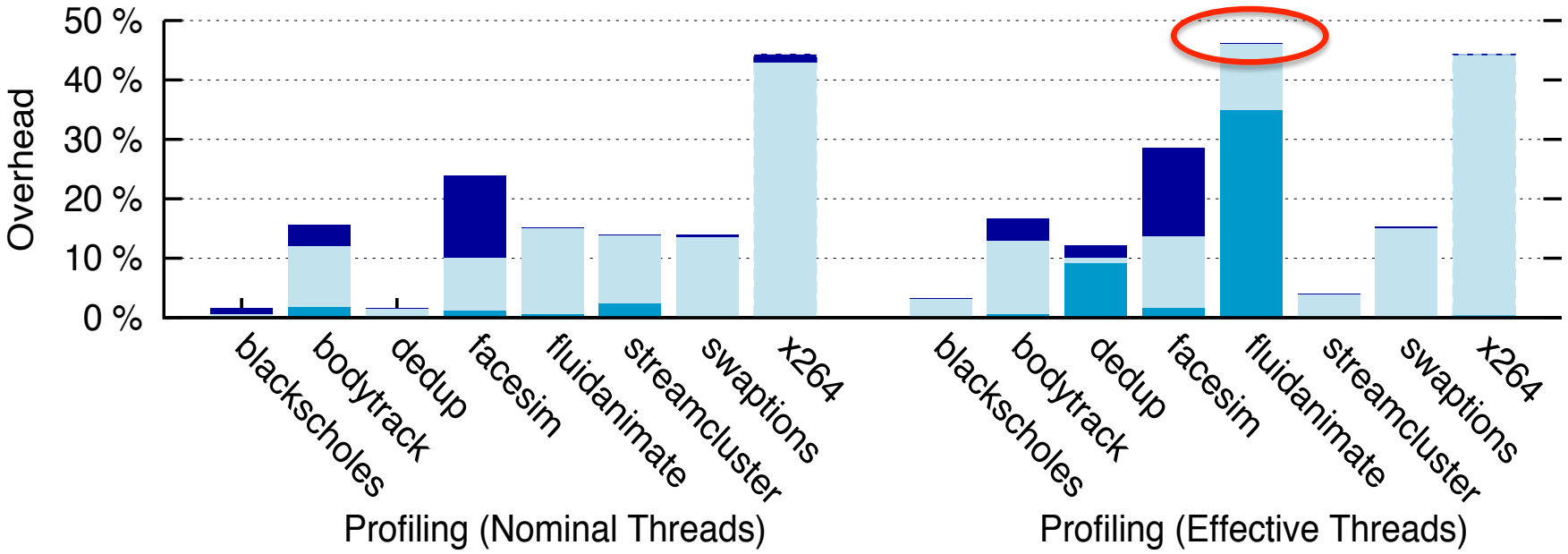
- At program start and end
- At thread create and exit
- Per basic block: *Increment counter*

Strategies for Keeping Overheads Low

Strategy	**Effect**
Static instrumentation	Overhead in compile, not runtime
Push instrumentation outside of basic blocks	Min. overhead (single incr.) per block
Thread specific structures & counters	No excessive locking
Organize data for cache locality	L1-I and L1-D miss rates increased <=0.06, 0.2%

Harmony Overheads on Parsec

Max overhead = 44%

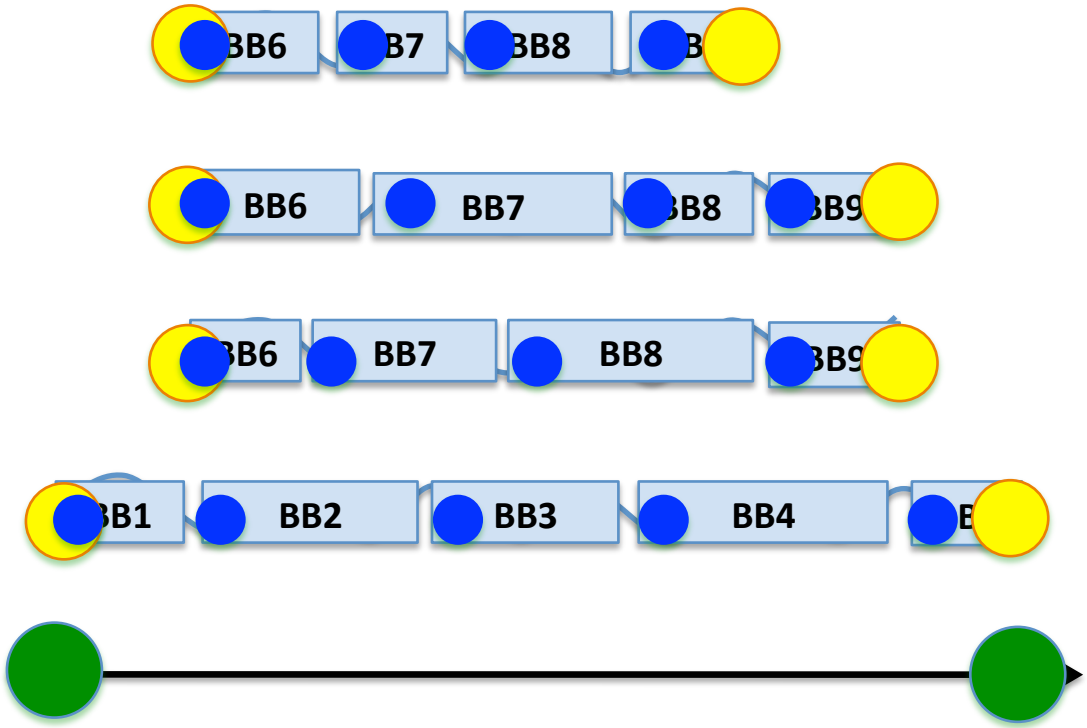


Avg. overhead = 16%

Avg. overhead = 21%

Harmony Overheads on Parsec

Avg. overhead = 16.2%

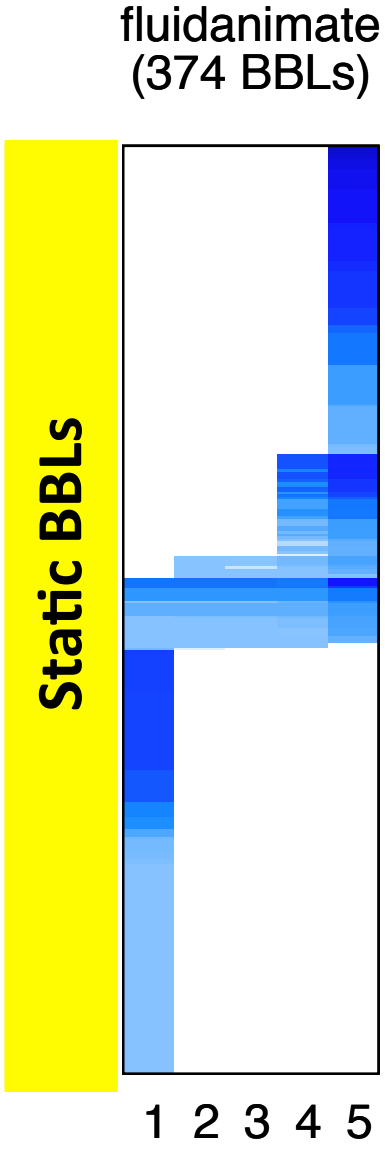


2.5%

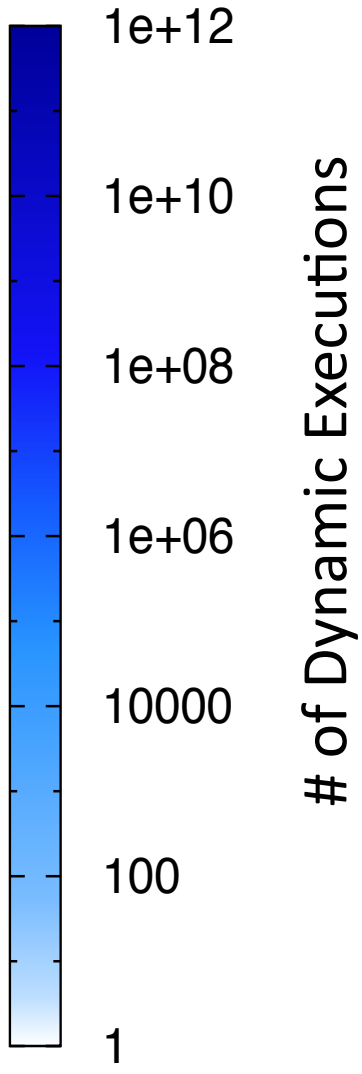
0.8%

13.0%

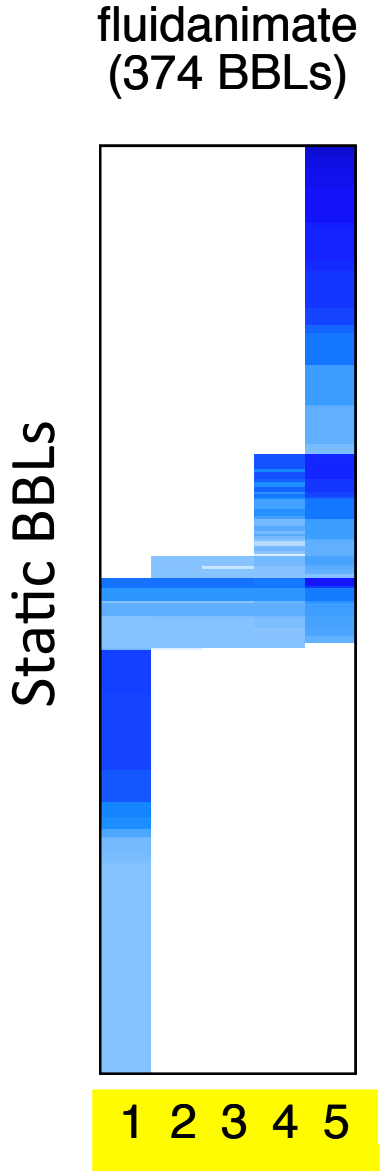
PBV Profiles of Parsec Applications



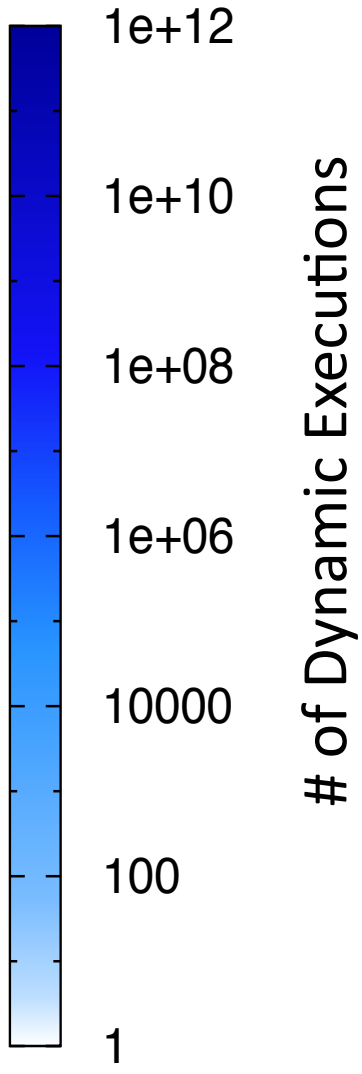
TC=	1	2	3	4
BB1	1	0	0	0
BB2	0	0	0	1
BB3	0	0	0	1
BB4	0	0	0	1
BB5	1	0	0	0
BB6	0	0	0	3
BB7	0	0	0	3
BB8	0	0	0	3
BB9	0	0	1	2



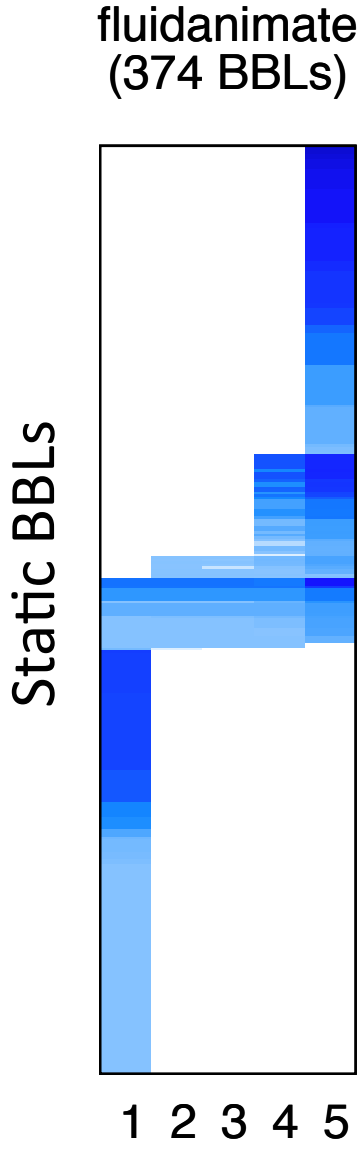
PBV Profiles of Parsec Applications



TC=	1	2	3	4
BB1	1	0	0	0
BB2	0	0	0	1
BB3	0	0	0	1
BB4	0	0	0	1
BB5	1	0	0	0
BB6	0	0	0	3
BB7	0	0	0	3
BB8	0	0	0	3
BB9	0	0	1	2

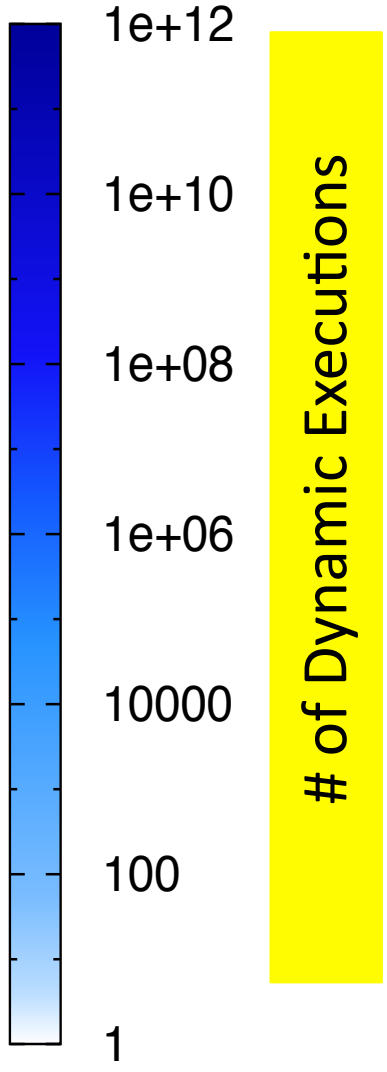


PBV Profiles of Parsec Applications

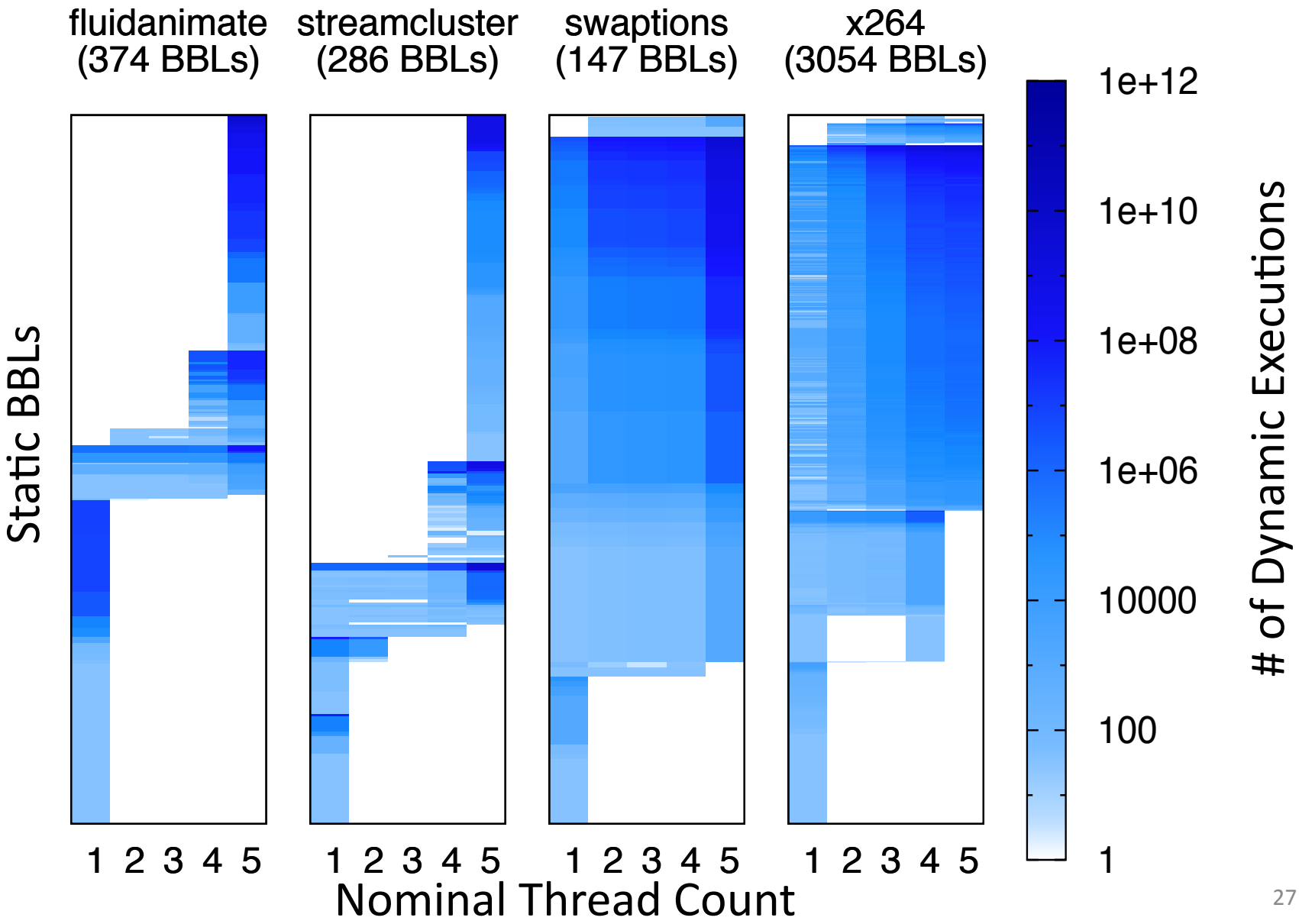


TC=	1	2	3	4
BB1	1	0	0	0
BB2	0	0	0	1
BB3	0	0	0	1
BB4	0	0	0	1
BB5	1	0	0	0
BB6	0	0	0	3
BB7	0	0	0	3
BB8	0	0	0	3
BB9	0	0	1	2

Nominal Thread Count



PBV Profiles of Parsec Applications



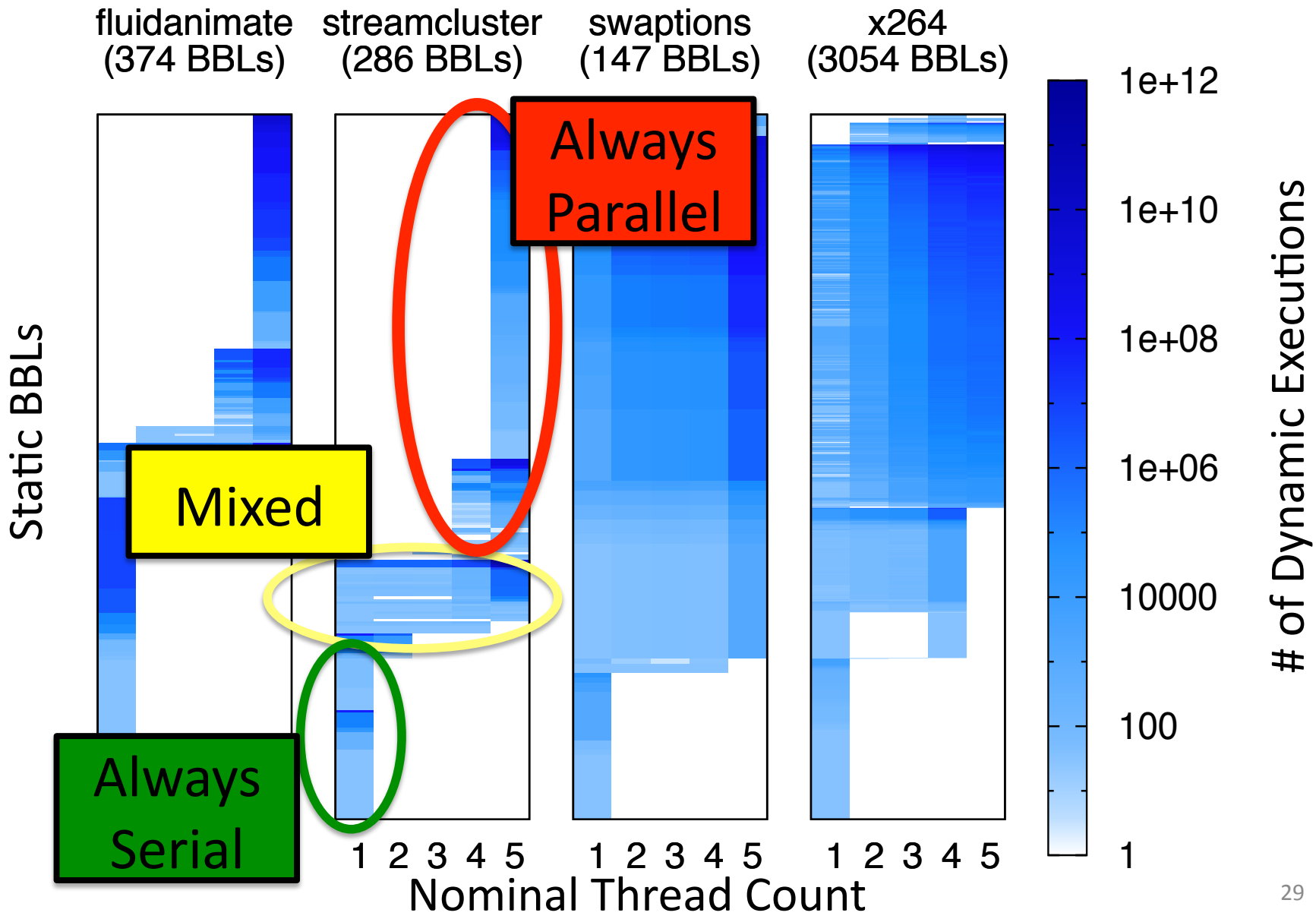
Using PBVs to Uncover Program Features

1. Can Amdahl's Law for estimating scalability be mapped to source code?

$$Speedup_{parallel}(f,n) = \frac{1}{(1-f) + \frac{f}{n}}$$

What code runs in these fractions?

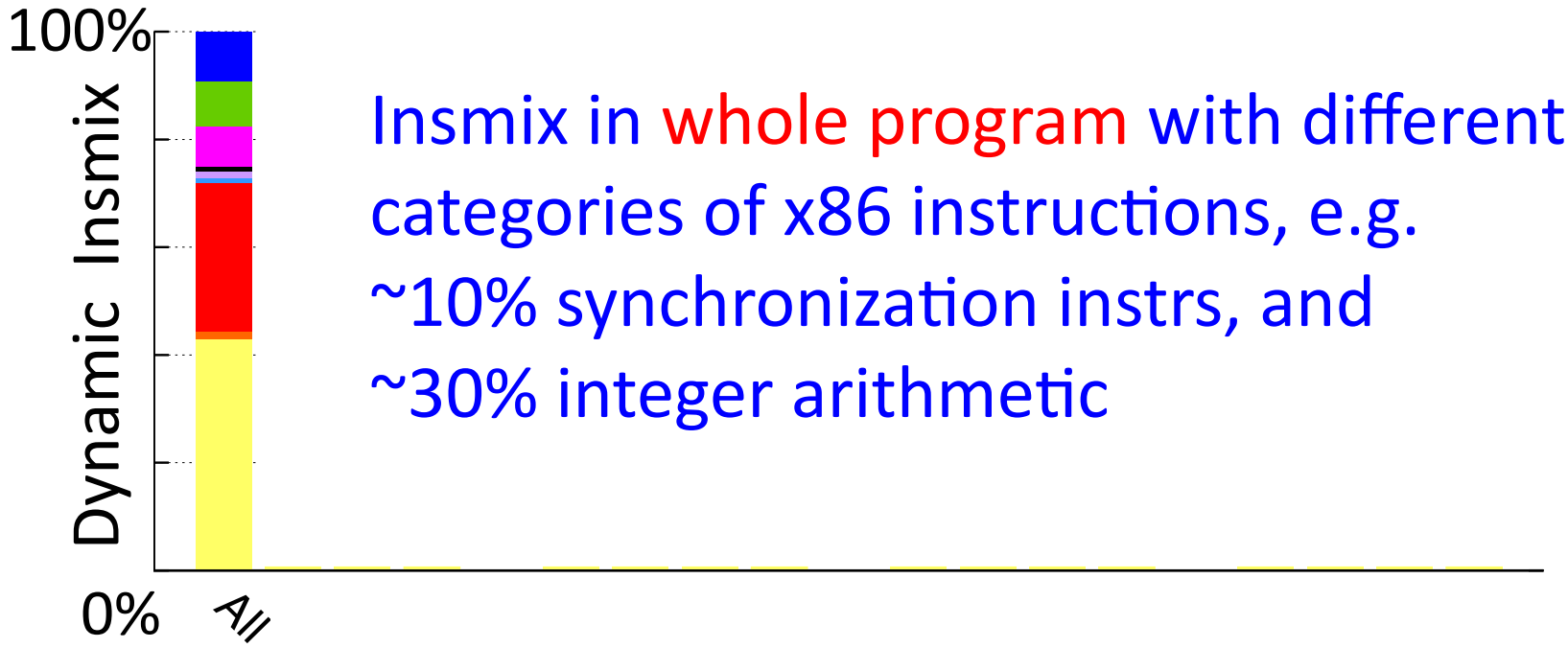
Parallel Phases in the Sample Profiles



Using PBVs to Uncover Program Features

1. ~~Can Amdahl's Law for estimating scalability be mapped to source code?~~
2. Do we need to design accelerators differently for different parallelism phases?

Instruction Mixes by Parallelism Class

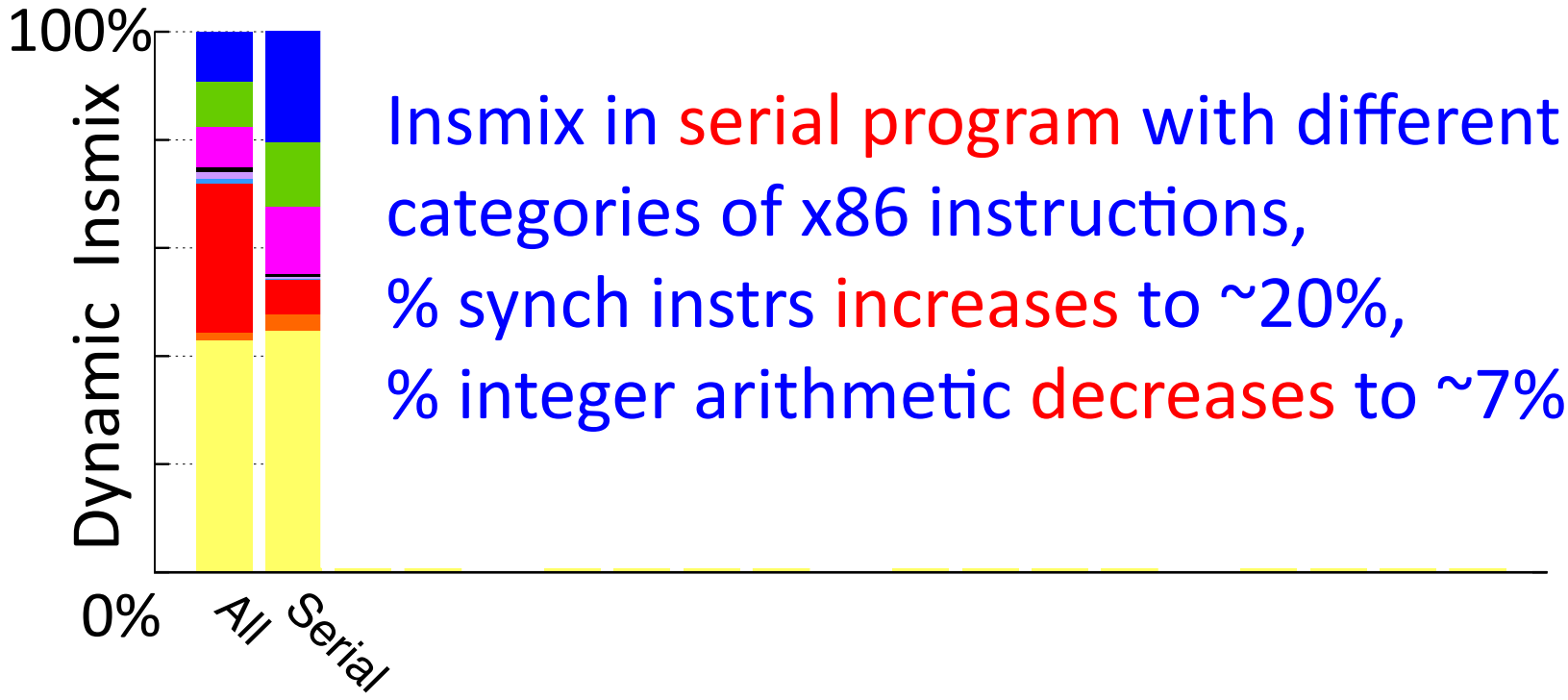


Insmix in whole program with different categories of x86 instructions, e.g. ~10% synchronization instrs, and ~30% integer arithmetic

facesim

ld/st
lea
arith (int)
arith (fp)
stack
cmp
cond CT
uncond CT
sync

Instruction Mixes by Parallelism Class

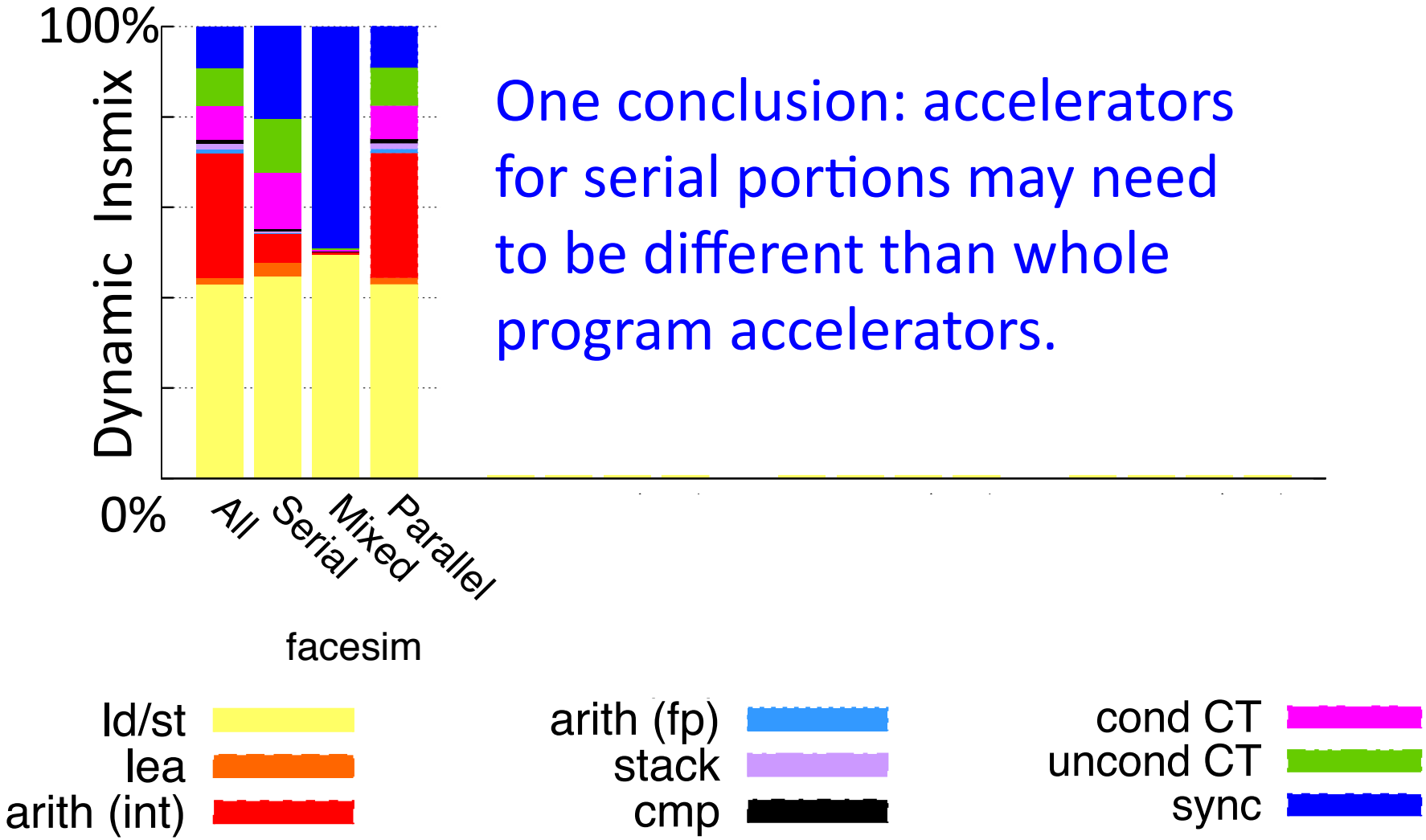


Insmix in serial program with different categories of x86 instructions,
% synch instrs increases to ~20%,
% integer arithmetic decreases to ~7%

facesim

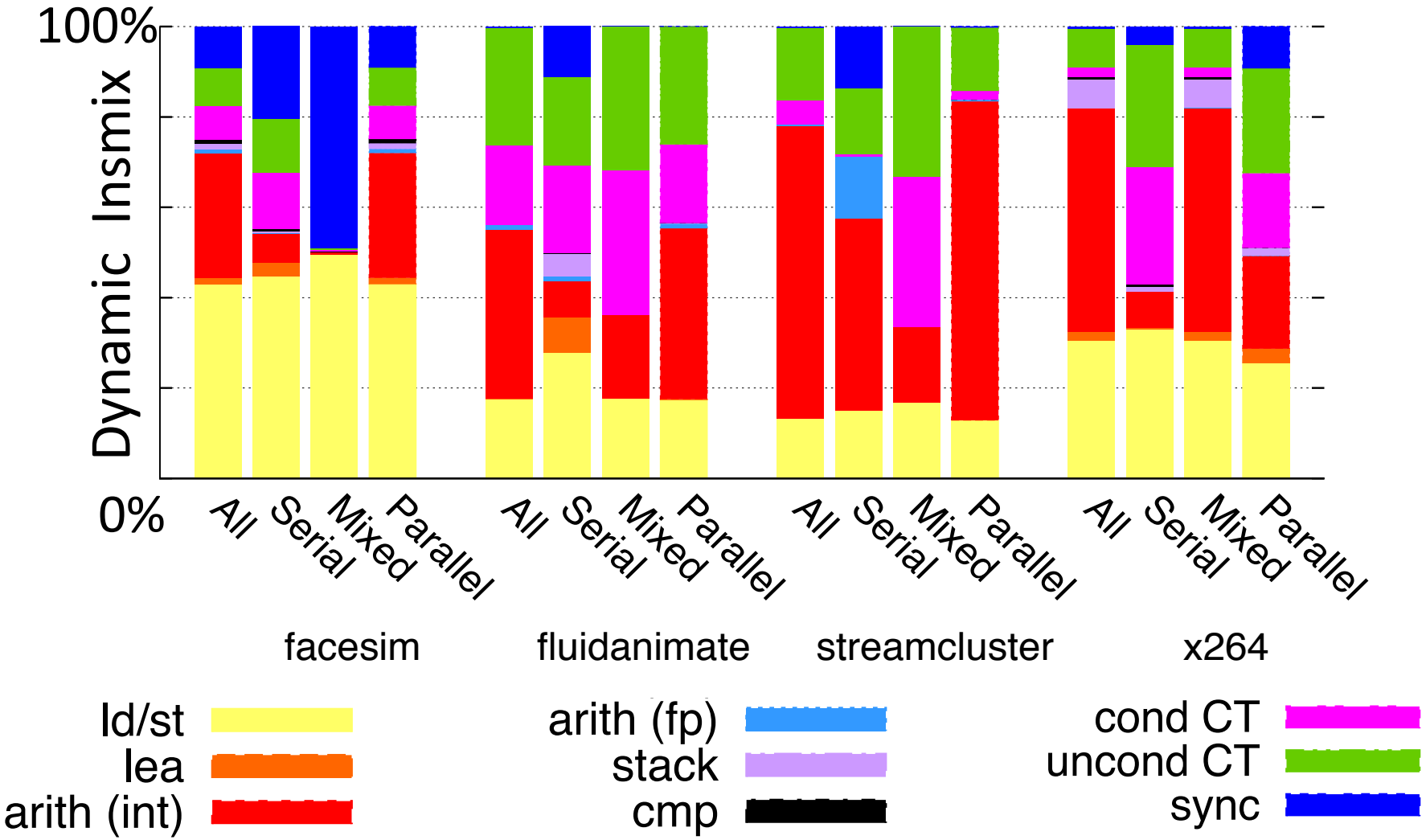
ld/st
lea
arith (int)
arith (fp)
stack
cmp
cond CT
uncond CT
sync

Instruction Mixes by Parallelism Class



One conclusion: accelerators for serial portions may need to be different than whole program accelerators.

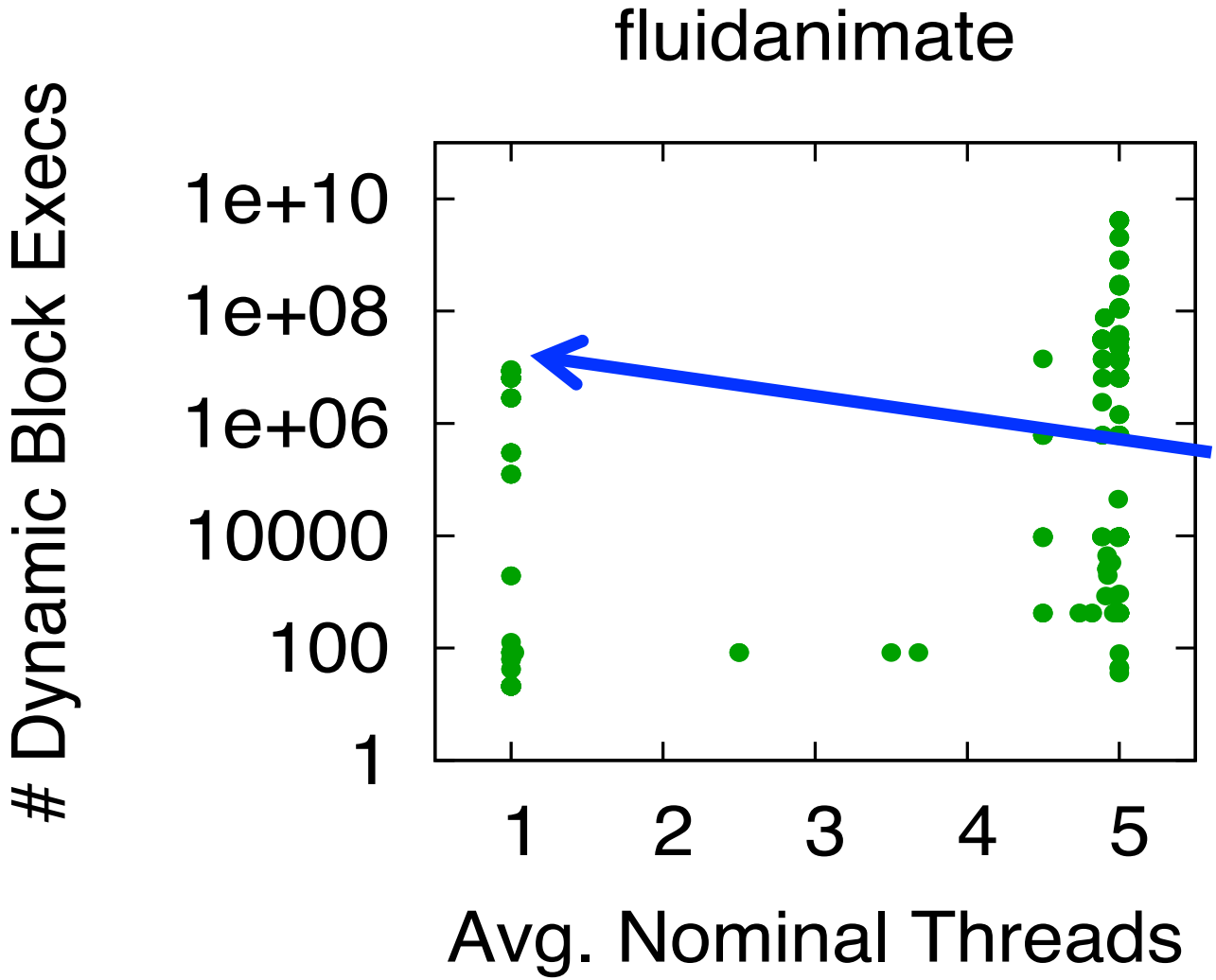
Instruction Mixes by Parallelism Class



Using PBVs to Uncover Program Features

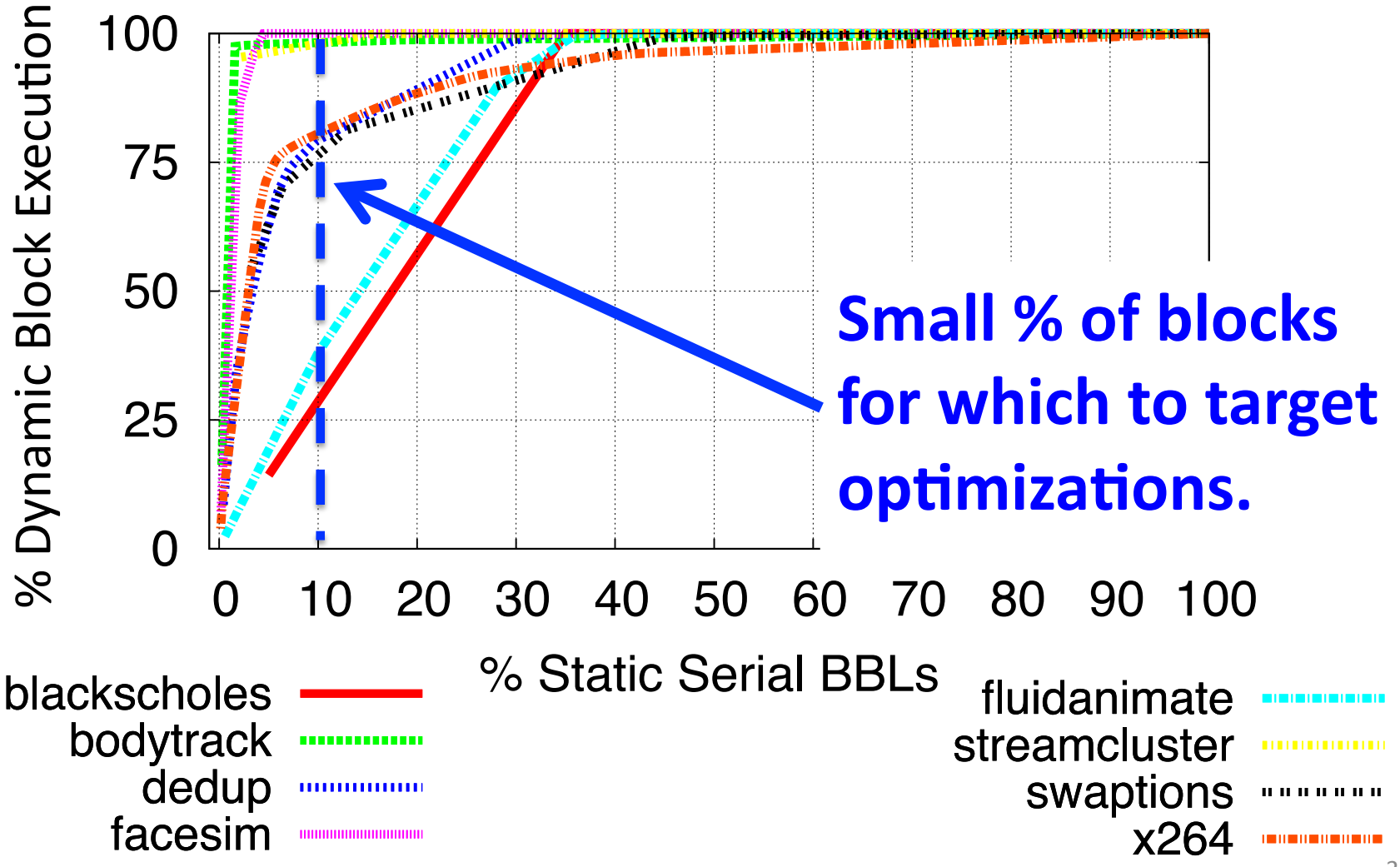
1. ~~Can Amdahl's Law for estimating scalability be mapped to source code?~~
2. ~~Do we need to design accelerators differently for different parallelism phases?~~
3. What can we learn about hotspots using PBVs?

Hottest blocks are not always the most parallel blocks



Serial blocks are hot (worth optimizing)

Few blocks represent large portion of serial block executions



Future directions for PBVs?

- Up to you! (download our code)

- A few ideas…

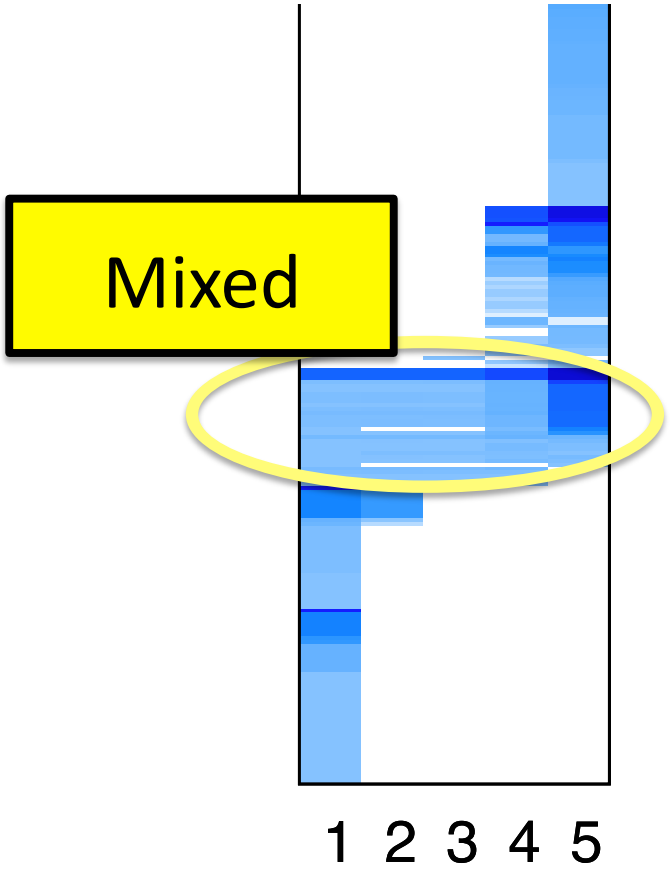
Using PBVs in Software Engineering

Parallelism Assertions to check if applications run as developers expect.

Identify Irregularities if a block runs serially 99% of the time and in parallel 1% of the time, might be a concurrency bug.

TC =	1	2	3	4
BB:24	99	0	0	1

Using PBVs in Systems Scheduling

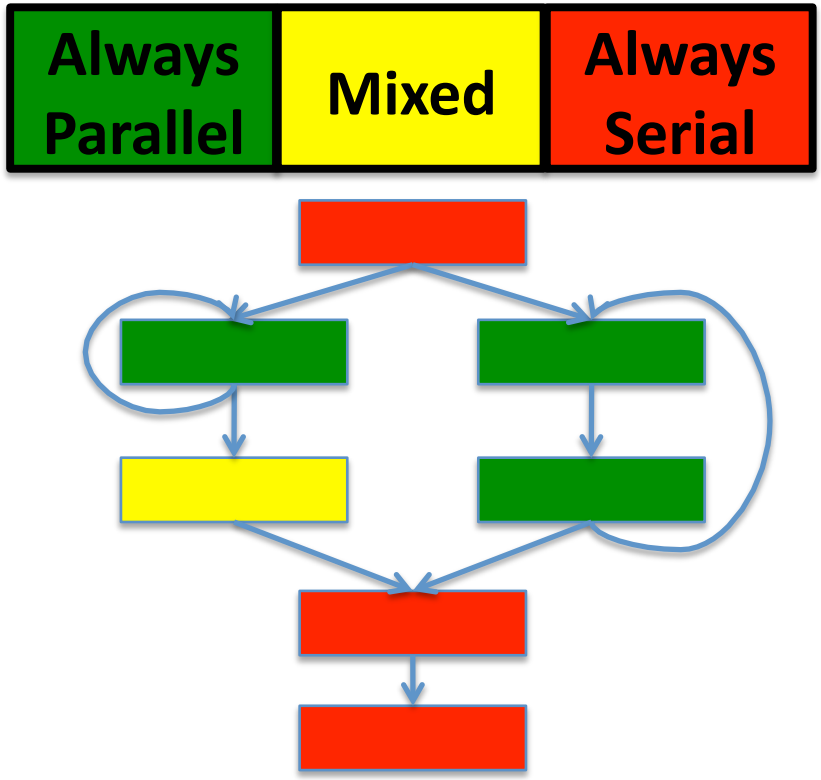


Examine mixed blocks: are some indicative of poor scheduling choices?

Predicting or classifying scalability at granularities like function or algorithm level instead of whole program.

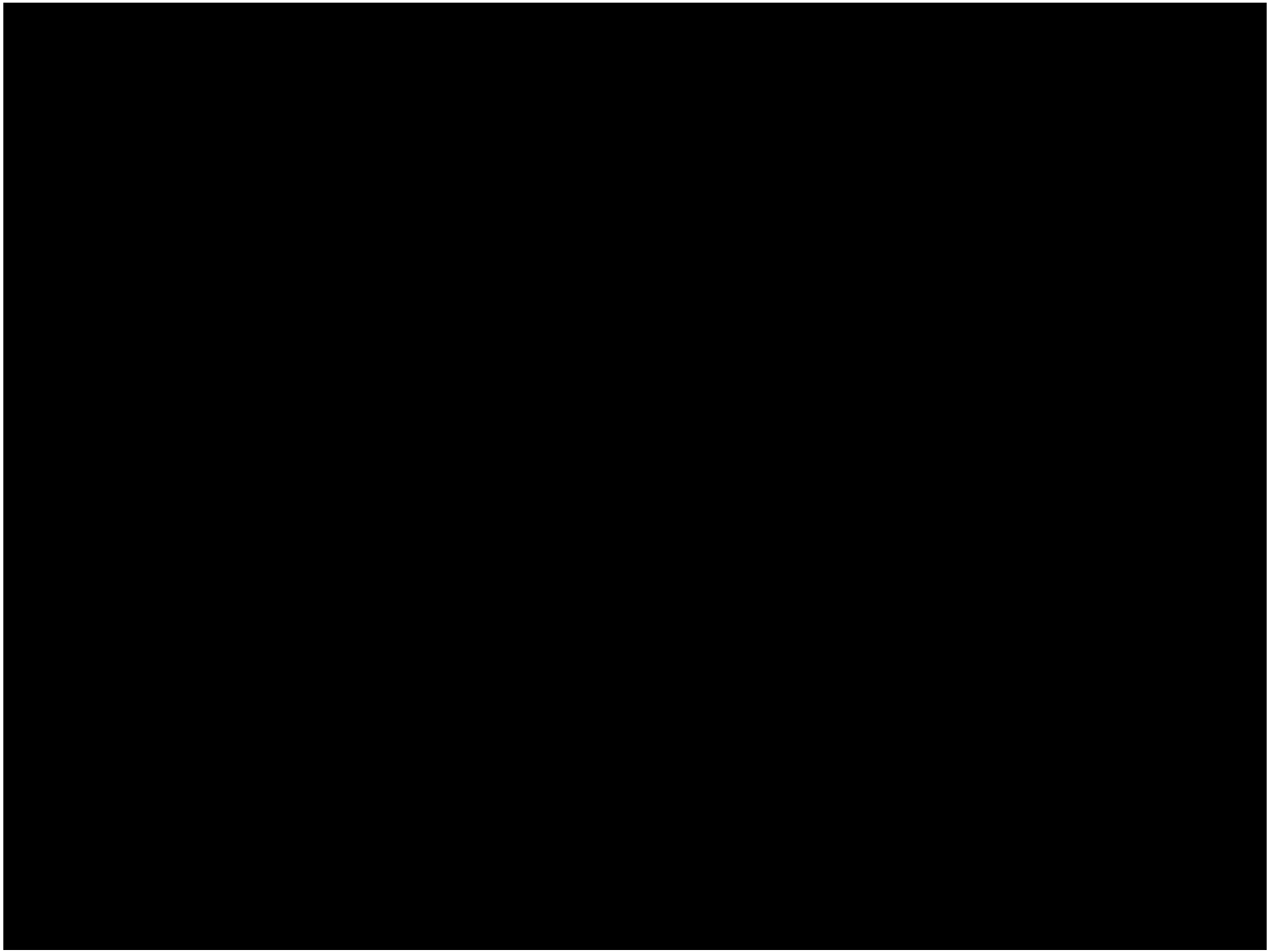
Using PBVs in Compilers

Annotate blocks by parallelism then use annotations to choose optimizations, better target code to processors.



Takeaways

- Parallel block vector profiles map dynamic parallelism to source code (at fine granularity) for the first time.

- New features identifiable with PBVs could lead to advances in architectural design, O/S, S/W engineering, compilers

http://arcade.cs.columbia.edu/harmony