Using Trace Scratchpads to Reduce Execution Times in Predictable Real-Time Architectures

Jack Whitham and Neil Audsley

April 24th 2008

My goal

Given an arbitrary program, specify a CPU extension that

- reduces *worst-case execution time* (WCET),
- without increasing *pessimism* in WCET analysis,
- while providing a *guaranteed bound*.

ACET

CPU designers usually concentrate on reducing *average-case execution time* (ACET) using:

- caches,
- superscalar out-of-order execution,
- branch prediction,
- memory speculation,
- generally, *clever but unpredictable techniques*.

ACET optimisations

Not much help for WCET reduction, because the dynamic optimisations increase:

- the range of possible behaviors,
- dependence on data/execution history,
- generally, *the complexity of static WCET analysis models*.

And some behaviors are particularly problematic!

Reducing pessimism

- Observation:

  Pessimism is the result of dynamic CPU behavior.

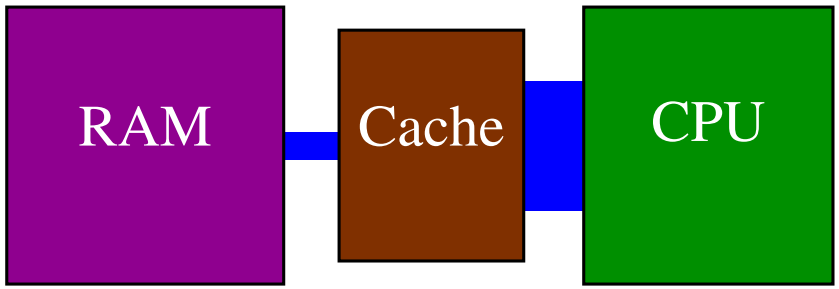
- Solution:

  Remove/constrain dynamic behavior.

Related work

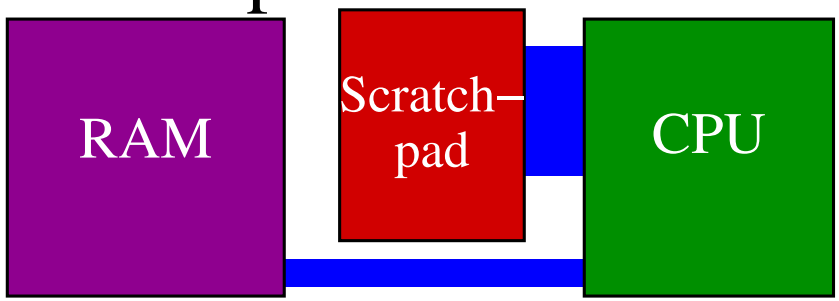
Scratchpads have been proposed as predictable replacements for caches.

Cache



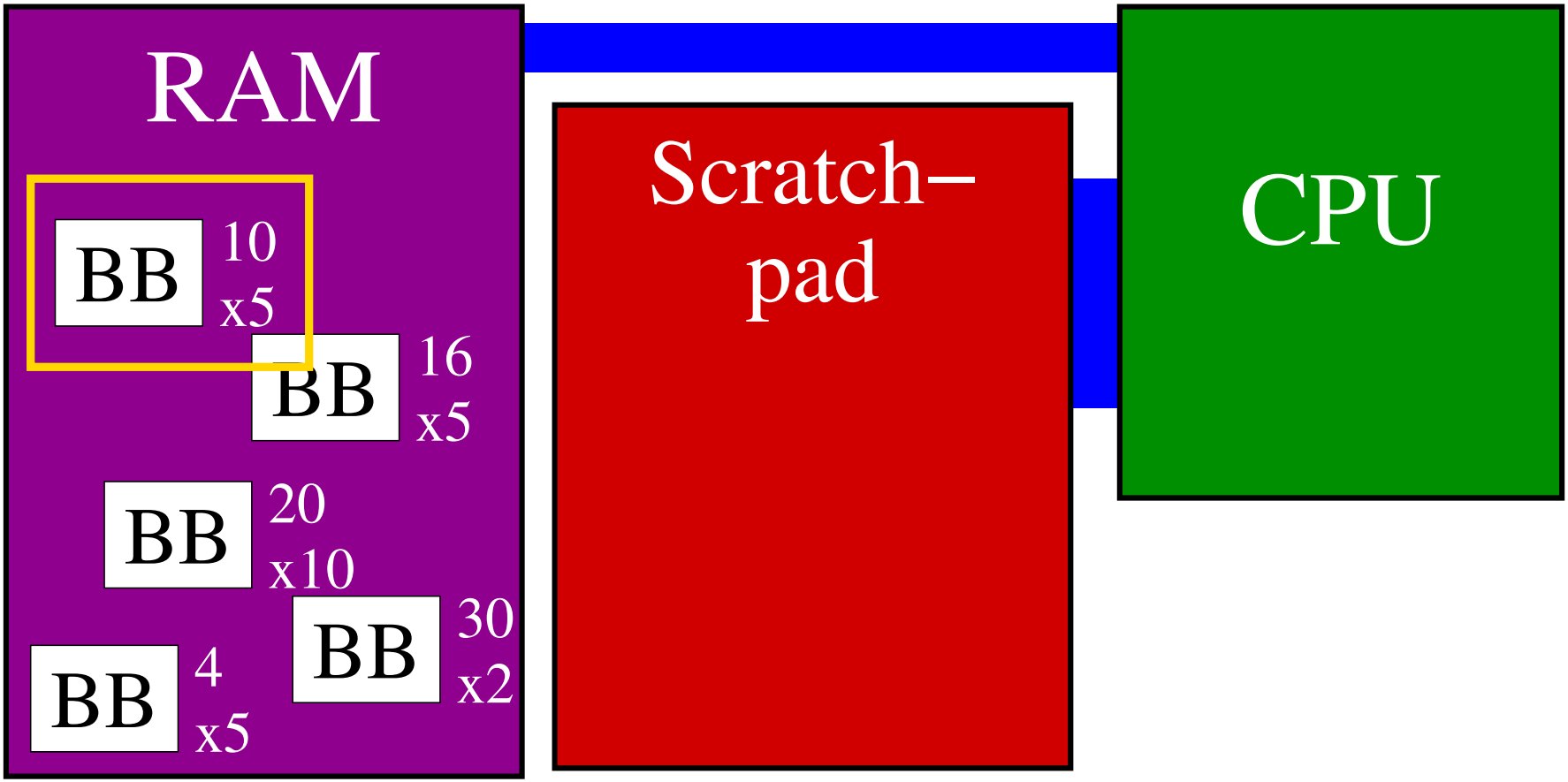
Hit 1 cycle
Miss 10 cycles
Automatically updated

Scratchpad

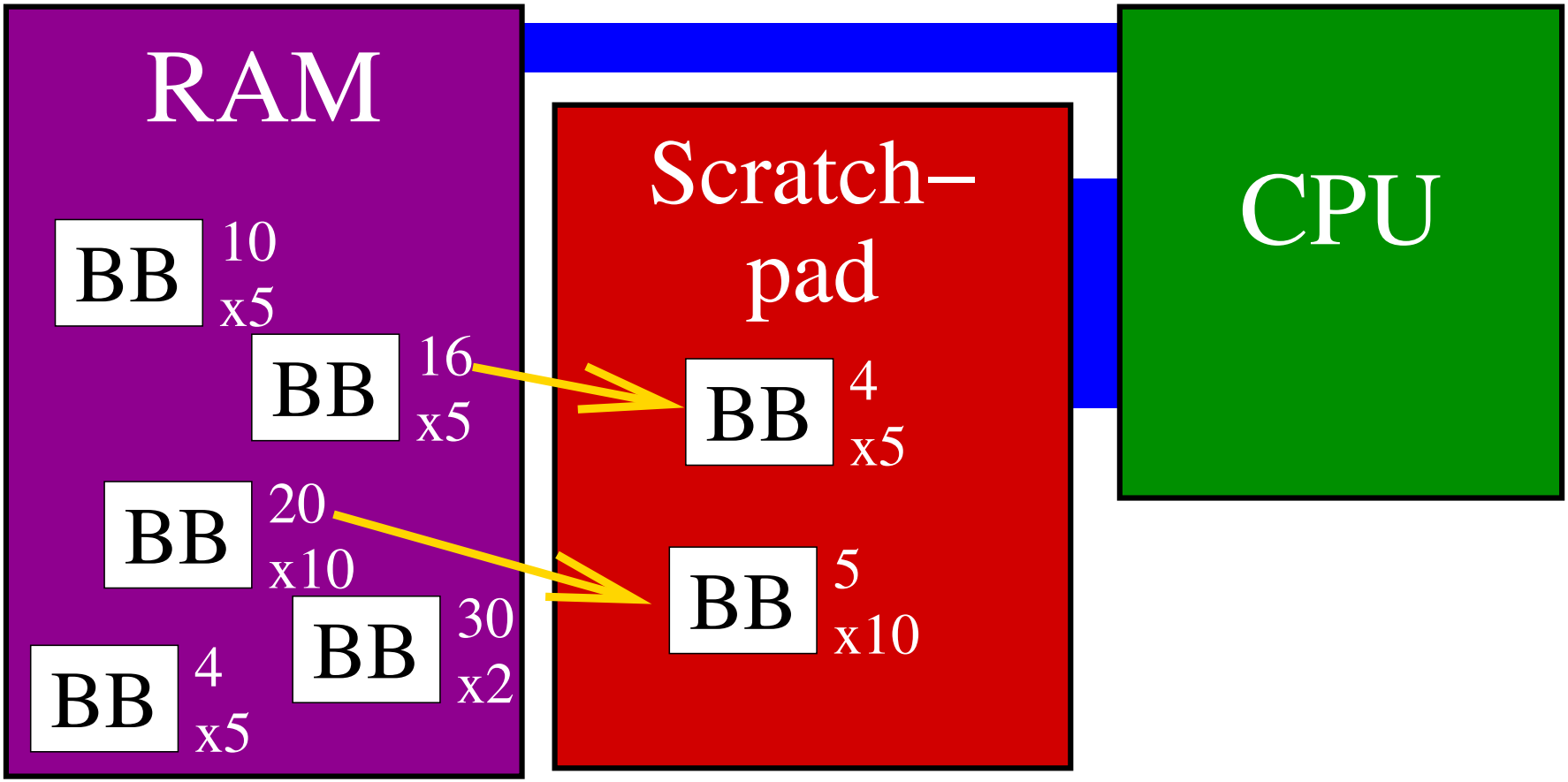


Hit 1 cycle
Miss 10 cycles
Updated explicitly by programs

Allocating scratchpad space - 1



Allocating scratchpad space - 2



Suhendra's scratchpad allocation algorithm

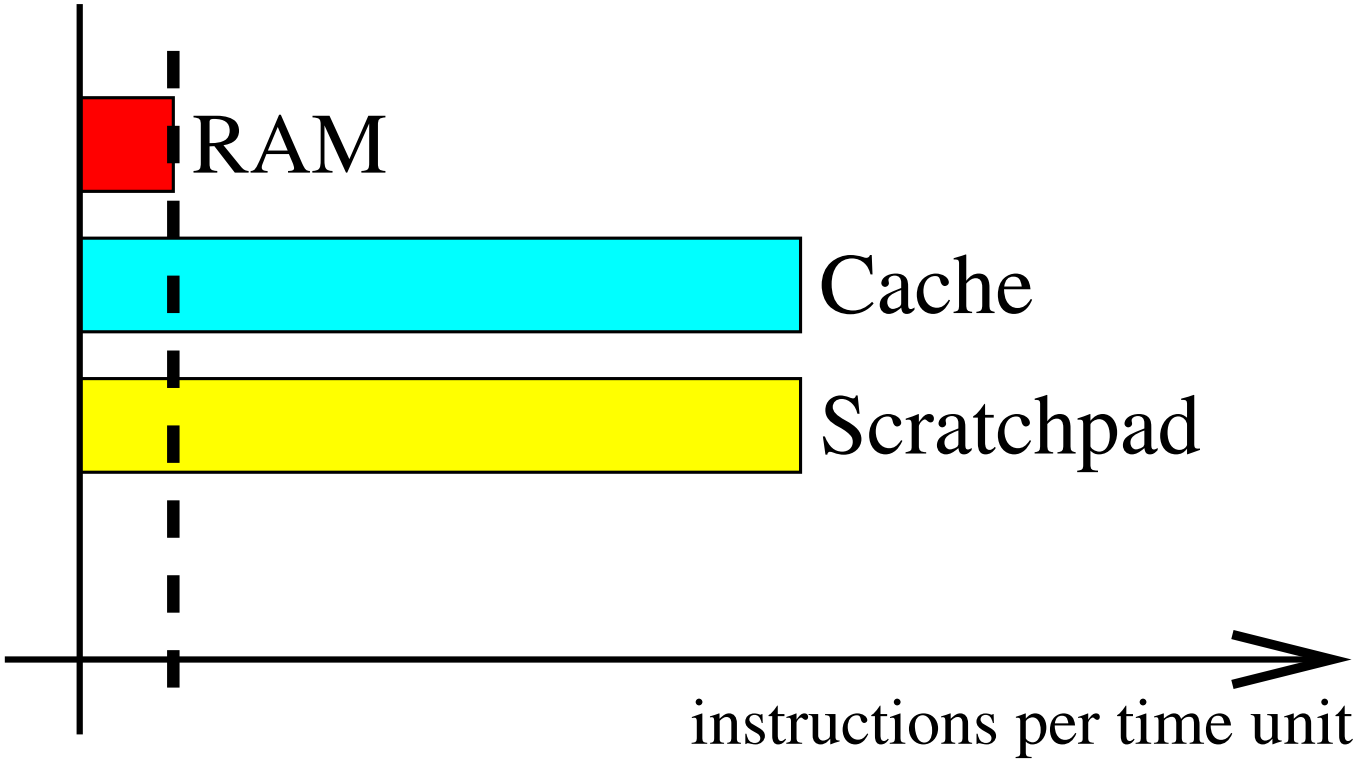
While the scratchpad is not full, do the following:

- Compute WCET and WCEP.
- Find BB with greatest contribution to WCET:

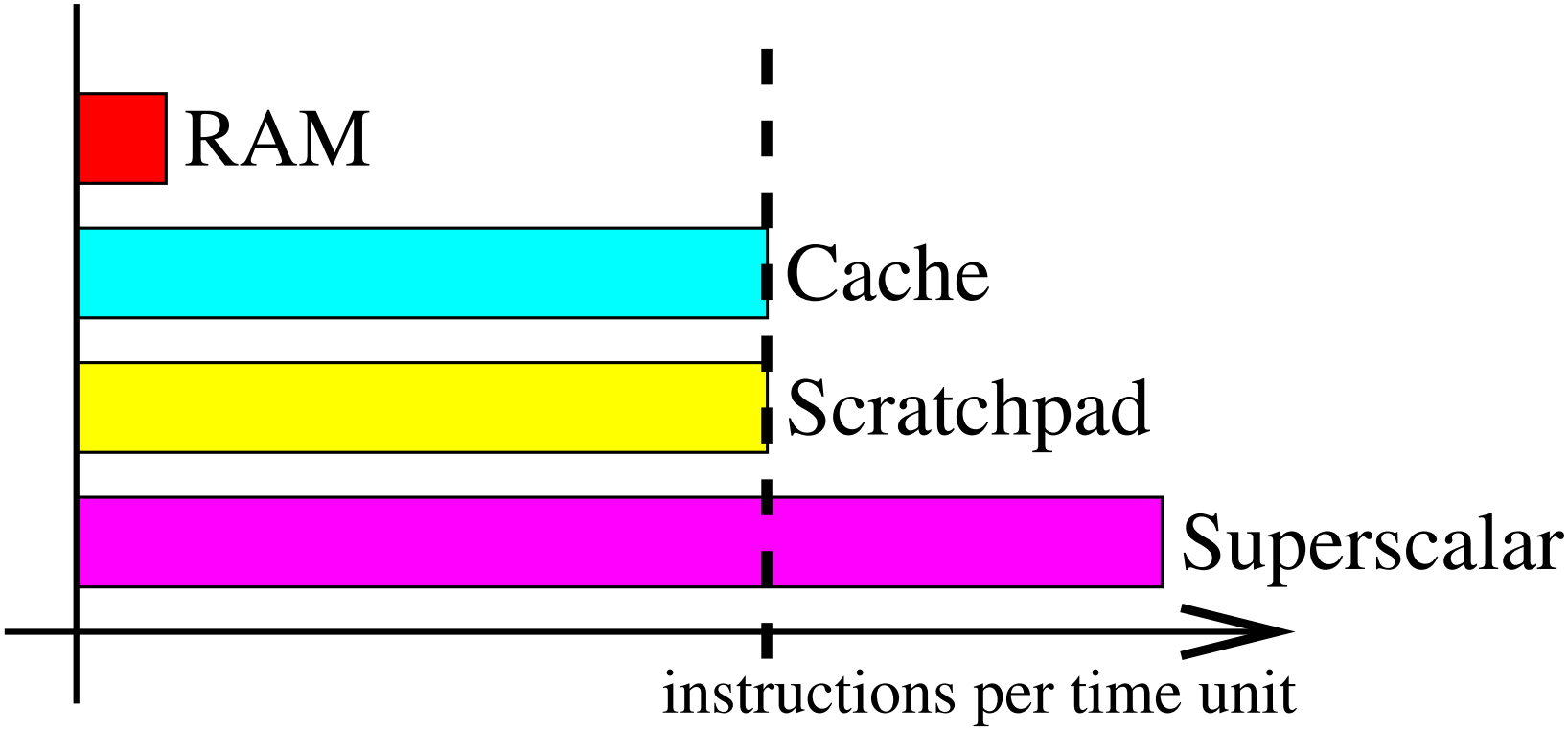
  $$\textit{number of executions} \times \textit{execution cost}$$

- Migrate BB to scratchpad.

Memory bottleneck



Instruction rate bottleneck



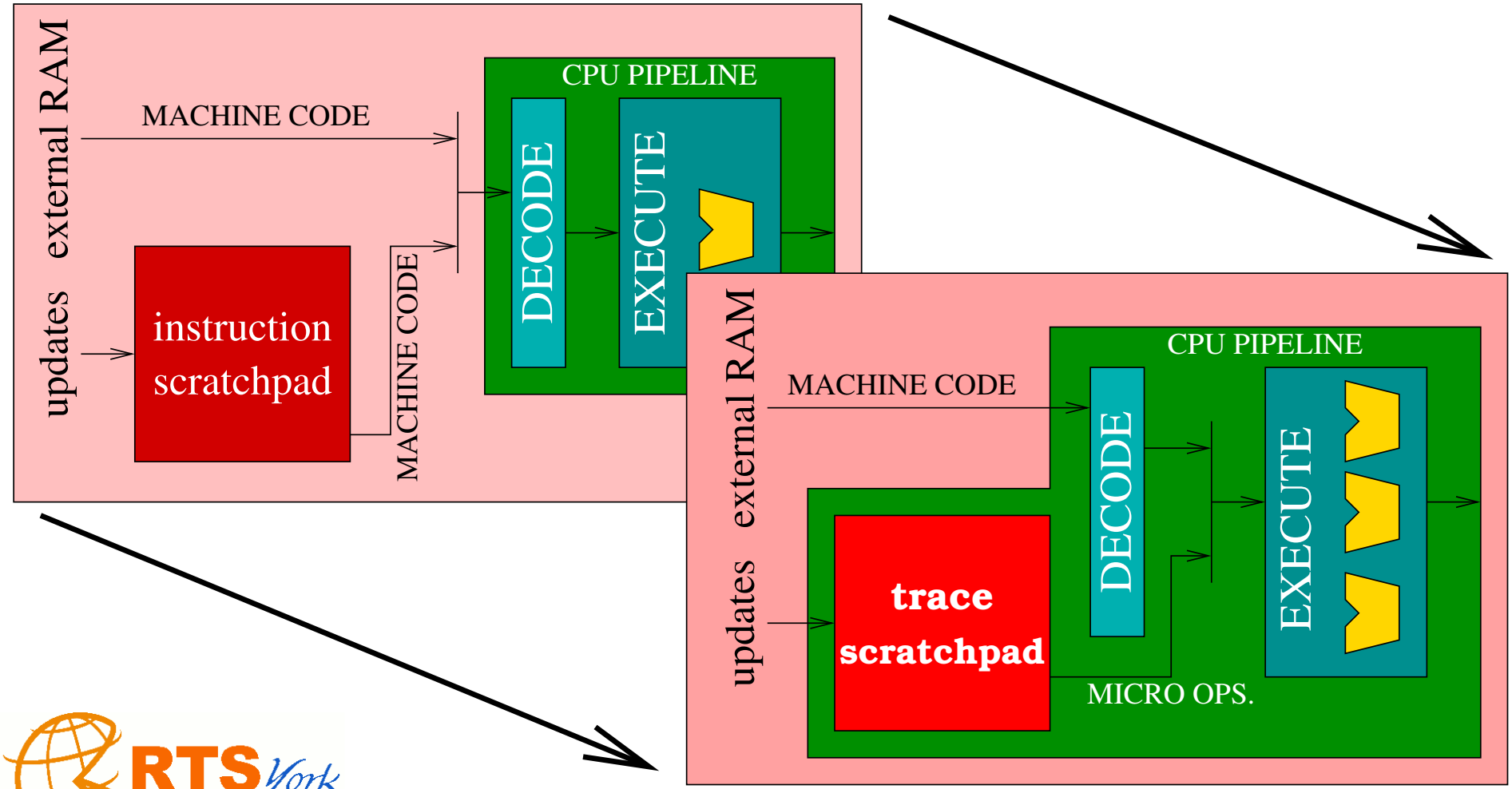
Where next?

- Need to use *instruction level parallelism* (ILP) to reduce WCET further.

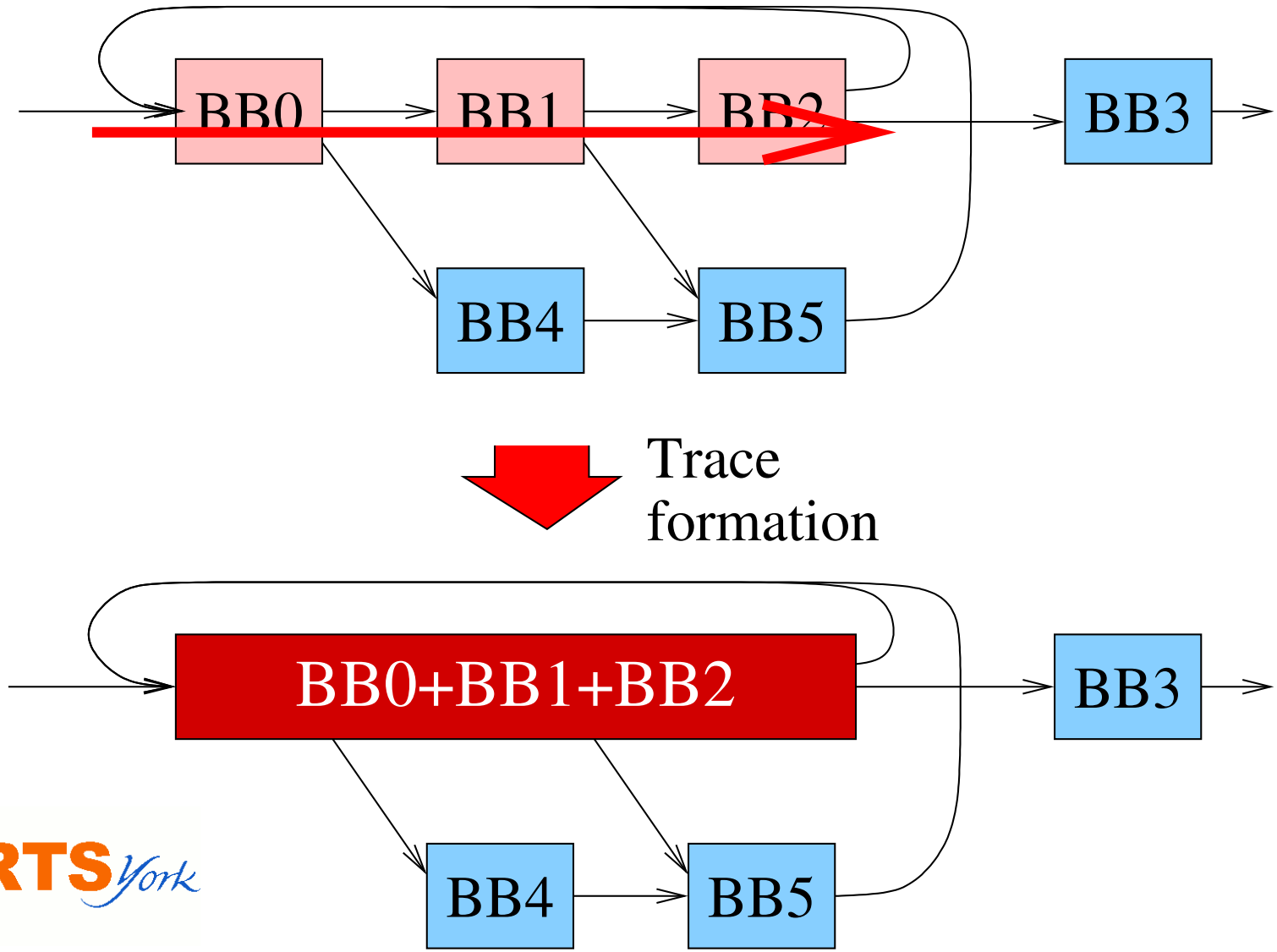
Where next?

- Need to use *instruction level parallelism* (ILP) to reduce WCET further.
- But, if we introduce dynamic behavior, we increase pessimism.

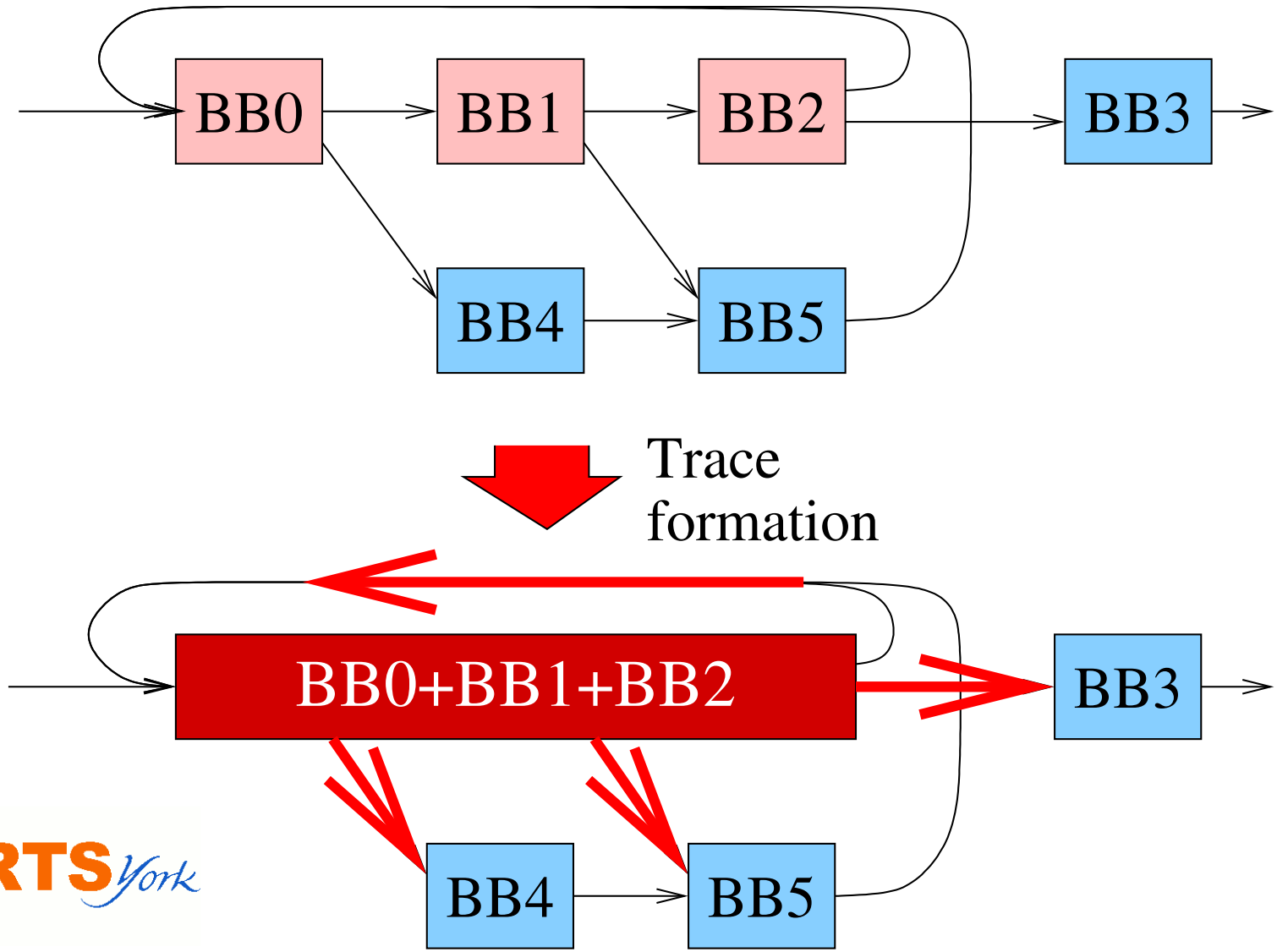
Proposed Solution



How it works


BB0
BB1
BB2
BB3
BB4
BB5
Trace formation
BB0+BB1+BB2
BB3
BB4
BB5

How it works


BB0
BB1
BB2
BB3
BB4
BB5
Trace formation
BB0+BB1+BB2
BB3
BB4
BB5

How traces are built

Incoming Machine code	
BB0	add r3,r4,r9
	add r6,r3,4
	load r5,r3
	load r4,r6
	cmp F,r5,r4
	add r7,r7,1
	bles F,BB2
BB1	store r3,r4
	add r8,r0,1
	store r6,r5
BB2	cmp F,r7,99
	sll r4,r7,2
	bles F,BB0

How traces are built

Incoming Machine code	
BB0	add r3,r4,r9
	add r6,r3,4
	load r5,r3
	load r4,r6
	cmp F,r5,r4
	add r7,r7,1
	bles F,BB2
BB1	store r3,r4
	add r8,r0,1
	store r6,r5
BB2	cmp F,r7,99
	sll r4,r7,2
	bles F,BB0

Register Renaming	
BB0	add r3,r4,r9
	add r6,r3,4
	load r5,r3
	load rA,r6
	cmp rD,r5,rA
	add rB,r7,1
	bles rD,BB2
BB1	store r3,rA
	add r8,r0,1
	store r6,r5
BB2	cmp rE,rB,99
	sll rC,rB,2
	bles rE,BB0

How traces are built

Incoming Machine code	
BB0	add r3,r4,r9
	add r6,r3,4
	load r5,r3
	load r4,r6
	cmp F,r5,r4
	add r7,r7,1
	bles F,BB2
BB1	store r3,r4
	add r8,r0,1
	store r6,r5
BB2	cmp F,r7,99
	sll r4,r7,2
	bles F,BB0

Register Renaming	
BB0	add r3,r4,r9
	add r6,r3,4
	load r5,r3
	load rA,r6
	cmp rD,r5,rA
	add rB,r7,1
	bles rD,BB2
BB1	store r3,rA
	add r8,r0,1
	store r6,r5
BB2	cmp rE,rB,99
	sll rC,rB,2
	bles rE,BB0

Dependence Order
add r8,r0,1
add r3,r4,r9
add r6,r3,4
add rB,r7,1
load r5,r3
cmp rE,rB,99
sll rC,rB,2
load rA,r6
cmp rD,r5,rA
bles rD,BB2
store r3,rA
store r6,r5
bles rE,BB0

How traces are built

Incoming Machine code	
BB0	add r3,r4,r9
	add r6,r3,4
	load r5,r3
	load r4,r6
	cmp F,r5,r4
	add r7,r7,1
	bles F,BB2
BB1	store r3,r4
	add r8,r0,1
	store r6,r5
BB2	cmp F,r7,99
	sll r4,r7,2
	bles F,BB0

Register Renaming	
BB0	add r3,r4,r9
	add r6,r3,4
	load r5,r3
	load rA,r6
	cmp rD,r5,rA
	add rB,r7,1
	bles rD,BB2
BB1	store r3,rA
	add r8,r0,1
	store r6,r5
BB2	cmp rE,rB,99
	sll rC,rB,2
	bles rE,BB0

Dependence Order
add r8,r0,1
add r3,r4,r9
add r6,r3,4
add rB,r7,1
load r5,r3
cmp rE,rB,99
sll rC,rB,2
load rA,r6
cmp rD,r5,rA
bles rD,BB2
store r3,rA
store r6,r5
bles rE,BB0

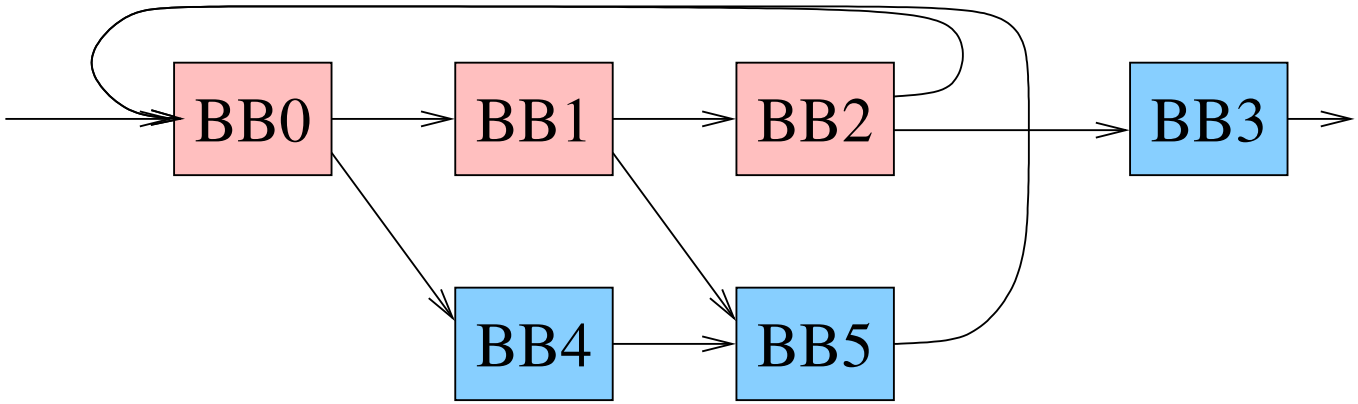
Trace Schedule	
Execution Unit 1	Execution Unit 2
add r8,r0,1	add r3,r4,r9
add r6,r3,4	add rB,r7,1
load r5,r3	cmp rE,rB,99
sll rC,rB,2	load rA,r6
cmp rD,r5,rA	
bles rD,BB2	store r3,rA
store r6,r5	
bles rE,BB0	

An optimization problem

- How do we allocate the limited trace scratchpad space such that WCET is minimized?

An optimization problem

- How do we allocate the limited trace scratchpad space such that WCET is minimized?
- A trace is characterized by *starting point* e, and by *length* l: $T_{e,l}$



- Θ = set of $T_{e,l}$ such that WCET is minimized.

$\Rightarrow$ *Our algorithm tries to find* Θ.

Search space

- There is a vast search space!
- We cut it down using heuristics:
  1. Find up to H possible traces.
  2. Test up to W possible start points.
  3. Limit trace length to L branches.

Trace scratchpad allocation

The search problem is solved in two basic steps:

1. *Where* should traces be created?
2. *How long* should each be?

Trace scratchpad allocation

The search problem is solved in two basic steps:

1. *Where* should traces be created?
   - Find H starting points $e_h \in \{e_1, ..., e_H\}$, forming traces at each.
2. *How long* should each be?

Trace scratchpad allocation

The search problem is solved in two basic steps:

1. *Where* should traces be created?
   - Find H starting points $e_h \in \{e_1, ..., e_H\}$, forming traces at each.
2. *How long* should each be?
   - Select the best length $l \in [1..L]$ for each trace, balancing the WCET reduction benefit $B_{e,l}$ against the scratchpad space usage $C_{e,l}$

"Where" - Find_Candidates part 1

Repeat for all $h \in [1, H]$:

1. Compute WCET and WCEP.
2. Find W BB sequences with greatest contribution to WCET:

   $$\textit{number of executions} \times \textit{estimated execution cost of path}$$

3. Generate trace at each, of maximum length.
4. Compute new WCET with each trace: choose the trace that minimized the WCET as e_h.

"How long" - Find_Candidates part 2

For each e_h:

1. Compute WCET with $T_{e_h,l}$ for each length $l \in [1, L]$.

   Reduction in WCET is $B_{e_h,l}$: the benefit.

2. Compute space cost of each trace.

   This is $C_{e_h,l}$: the cost.

"How long" - Solve_Knapsack

Select the best length $l \in [1..L]$ for each starting point e_h:

Start Point	Trace Scratchpad Space				WCET Reduction Benefit			
	L=1	L=2	L=3	L=4	L=1	L=2	L=3	L=4
BB29	32	51	70	89	12200	24450	28540	30580
BB15	44	71	98	125	13300	18550	20300	21182
BB6	52	259	340		928	4128	4248	
BB34	54	86	118	150	1582	2513	2836	2988
BB21	36	58	80	102	1194	2019	2294	2426
BB24	36	57	78	99	1044	1944	2244	2388
BB36	68	112	156	200	944	1469	1637	1721
BB30		38	66			145	510	
BB16	31	53	90		50	393	463	
BB4			273	354			6	112
	(microinstructions)				(clock cycles)			

How is the WCET calculated?

- We use the *implicit path enumeration technique* (IPET).

  Li and Malik - *Performance analysis of embedded software using implicit path enumeration, Proc. DAC, 1995*

  Puschner and Schedl - *Computing Maximum Execution Times - a Graph-Based Approach, RTS 13(1):67-91, 1997*

- Benefits:
  - Can incorporate *any* linear constraint on program behavior.
  - Computes the *exact* WCET (given all constraints).

How is the WCET calculated?

- We use the *implicit path enumeration technique* (IPET).

  Li and Malik - *Performance analysis of embedded software using implicit path enumeration, Proc. DAC, 1995*

  Puschner and Schedl - *Computing Maximum Execution Times - a Graph-Based Approach, RTS 13(1):67-91, 1997*

- Benefits:
  - Can incorporate *any* linear constraint on program behavior.
  - Computes the *exact* WCET (given all constraints).
- But: IPET requires constant basic block execution times.

How is the WCET calculated?

- We use the *implicit path enumeration technique* (IPET).

  Li and Malik - *Performance analysis of embedded software using implicit path enumeration, Proc. DAC, 1995*

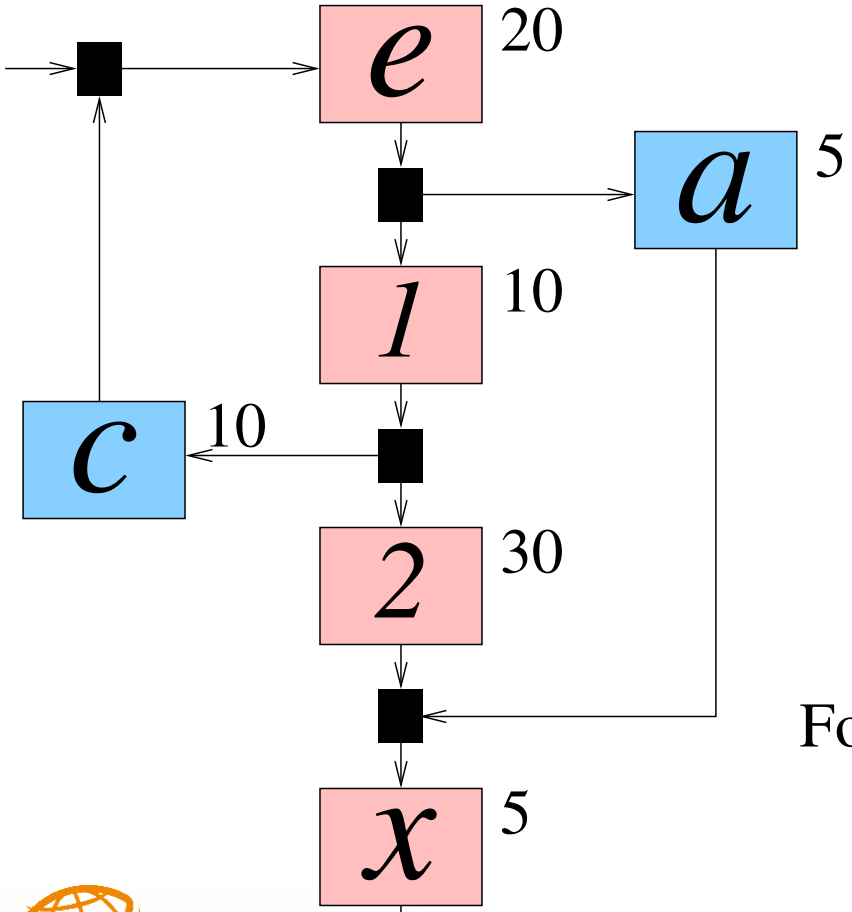
  Puschner and Schedl - *Computing Maximum Execution Times - a Graph-Based Approach, RTS 13(1):67-91, 1997*

- Benefits:
  - Can incorporate *any* linear constraint on program behavior.
  - Computes the *exact* WCET (given all constraints).
- But: IPET requires constant basic block execution times.
- No problem!

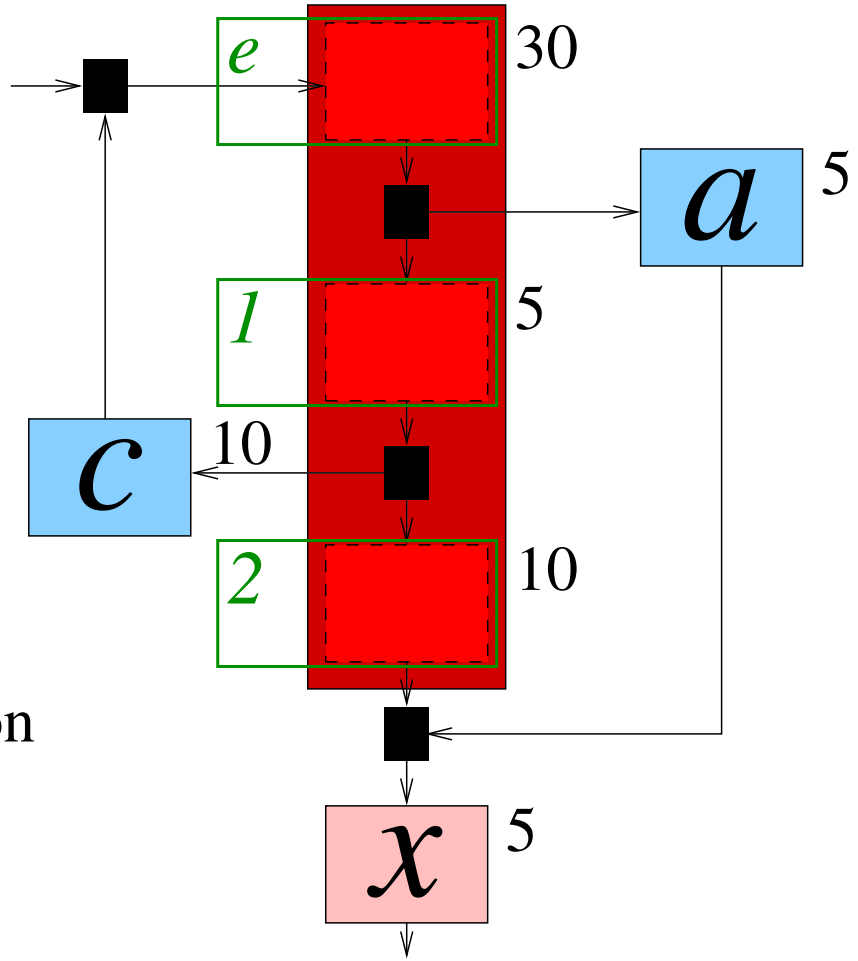
  Although traces allow speculative execution, they are composed of basic blocks in microcode.

  $\Rightarrow$ *IPET is applicable.*

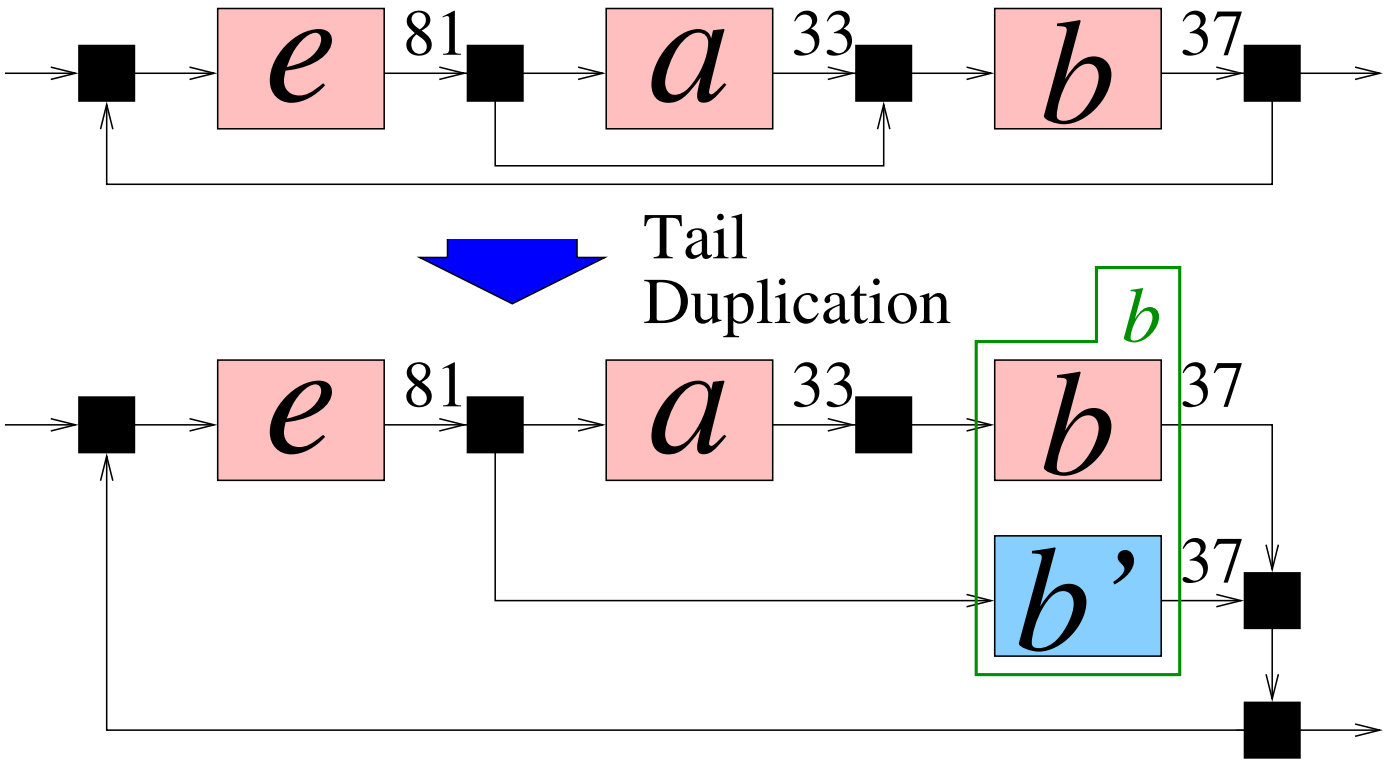
Example 1



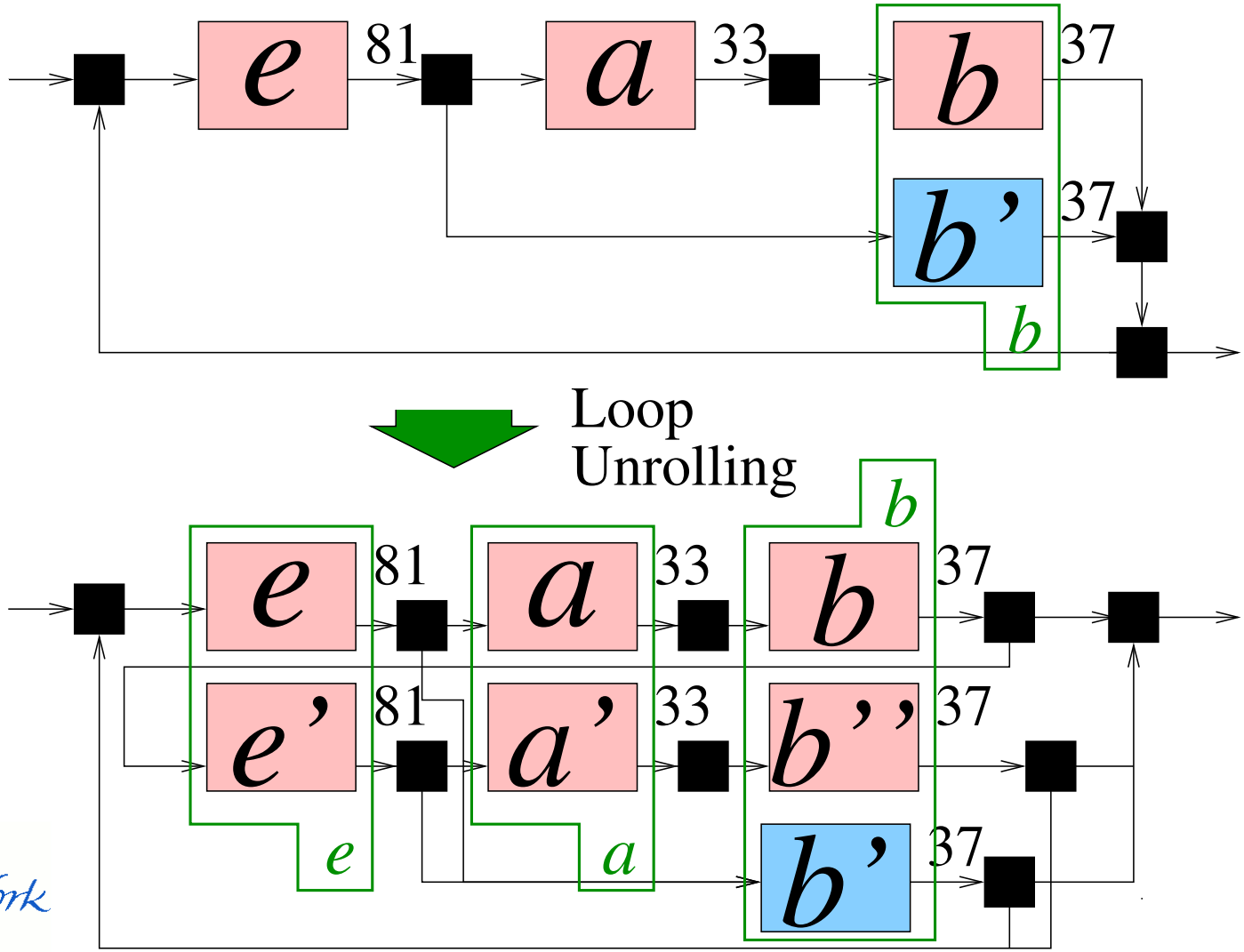
Trace Formation



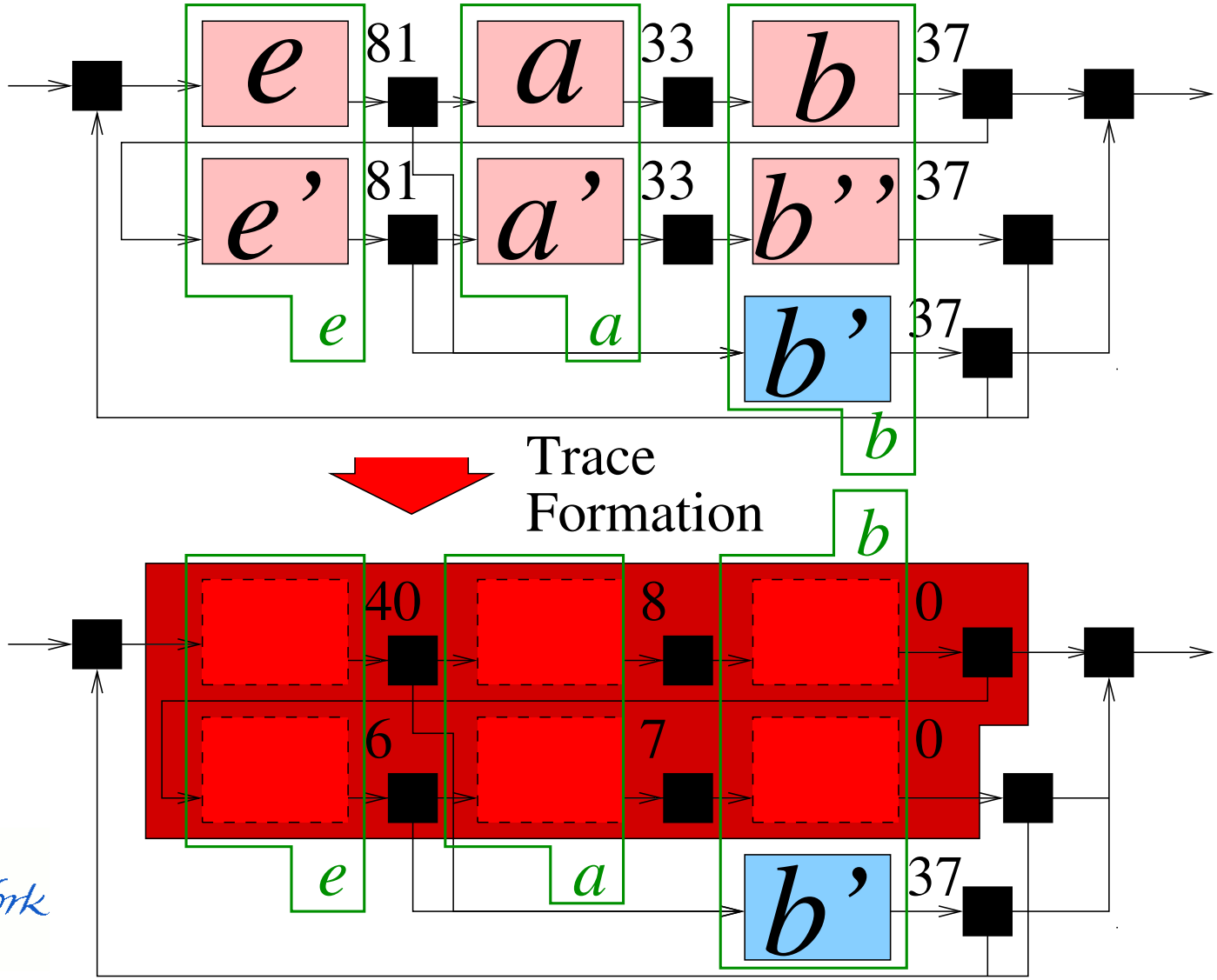
Example 2(a)



Example 2(b)


e
81
a
33
b
37
b'
37
b
Loop
Unrolling
b
e
81
a
33
b
37
e'
81
a'
33
b''
37
b'
37
e
a

Example 2(c)


e
81
a
33
b
37
e'
81
a'
33
b''
37
e
a
b'
37
b
Trace
Formation
b
40
8
0
6
7
0
e
a
b'
37

Experiment

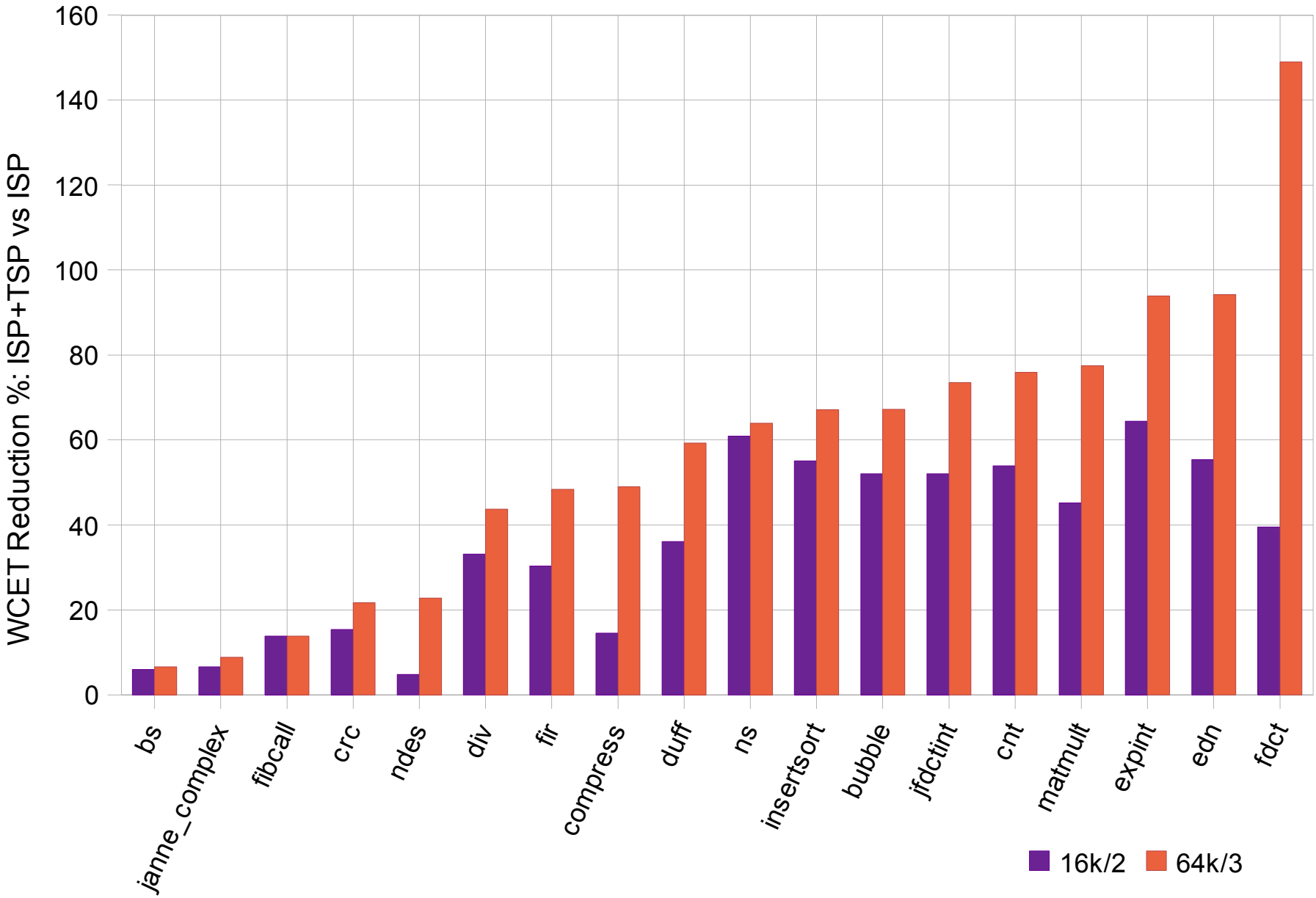
Using 18 programs from the Mälardalen WCET benchmark corpus, we compared:

- WCET after Instruction Scratchpad (ISP) allocation.
- WCET after ISP allocation *and* Trace Scratchpad (TSP) allocation.

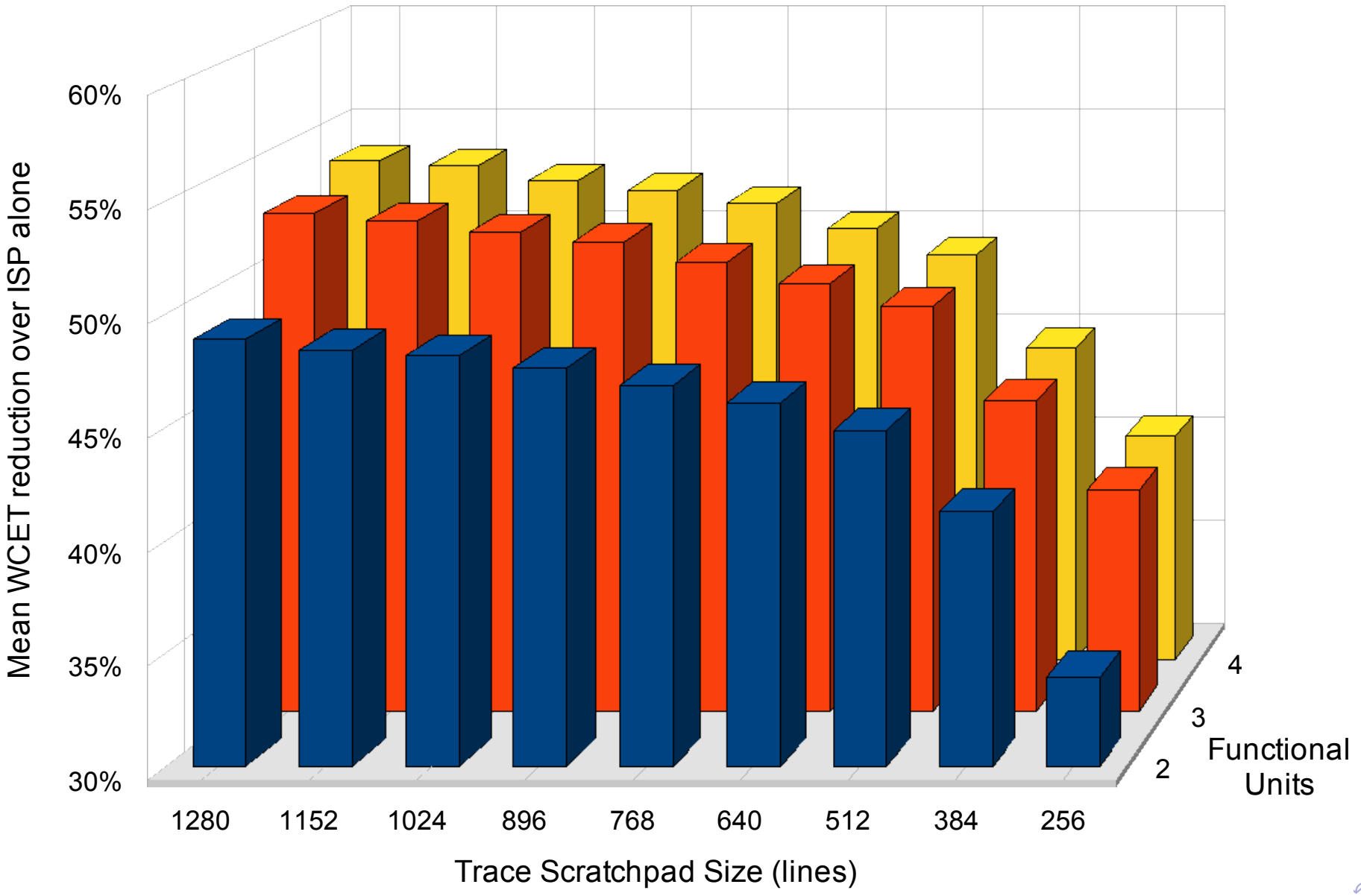
We used various CPU configurations and scratchpad sizes.

- Experiments use our MCGREP-2 CPU.
- Assumption: all data (variables) are stored in a scratchpad.
- Results computed by WCET analysis, then checked by measurement in a simulator.

Results 1


WCET Reduction %: ISP+TSP vs ISP
160
140
120
100
80
60
40
20
0
bs
janne_complex
fibcall
crc
ndes
div
fir
compress
duff
ns
insertsort
bubble
jfdctint
cnt
matmult
expint
edn
fdct
16k/2
64k/3

Results 2


Mean WCET reduction over ISP alone
60%
55%
50%
45%
40%
35%
30%
1280
1152
1024
896
768
640
512
384
256
Trace Scratchpad Size (lines)
2
3
4
Functional Units

Evaluation

- Trace scratchpads allow a program's WCET to be reduced by exploiting ILP.

Evaluation

- Trace scratchpads allow a program's WCET to be reduced by exploiting ILP.
- WCET analysis is possible using IPET.

Evaluation

- Trace scratchpads allow a program's WCET to be reduced by exploiting ILP.
- WCET analysis is possible using IPET.
- But WCET reductions are limited by:
  - the scratchpad size,
  - the degree of temporal locality in the program,
  - the degree of ILP available in the program,
  - the CPU configuration.

Conclusion

- Trace scratchpads succeed in reducing the WCET of programs.

Conclusion

- Trace scratchpads succeed in reducing the WCET of programs.
- ILP can be exploited without complicating a WCET model.

Conclusion

- Trace scratchpads succeed in reducing the WCET of programs.
- ILP can be exploited without complicating a WCET model.
- Future work:
  - We need to improve the *implementation* of traces in order to make our results more widely applicable.
  - We also need good ways to handle data accesses predictably.

Conclusion

- Trace scratchpads succeed in reducing the WCET of programs.
- ILP can be exploited without complicating a WCET model.
- Future work:
  - We need to improve the *implementation* of traces in order to make our results more widely applicable.
  - We also need good ways to handle data accesses predictably.
- The end. Questions?