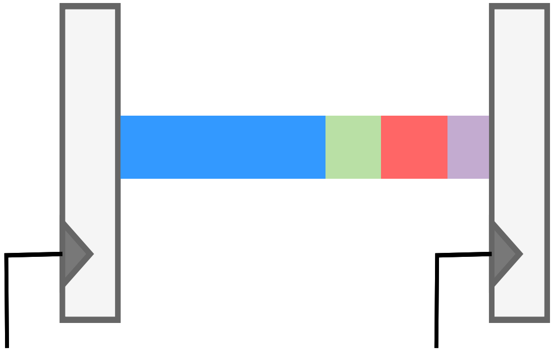
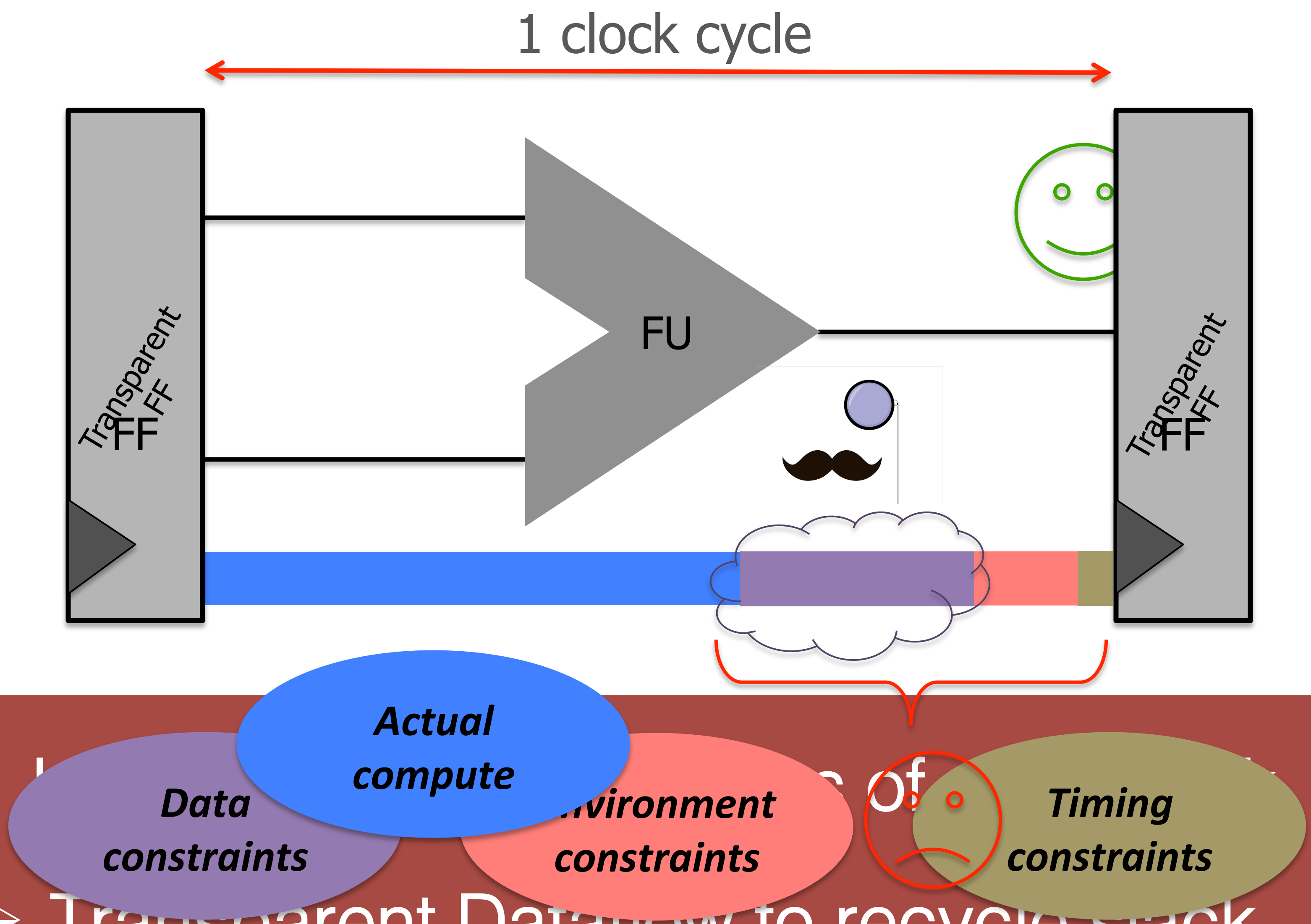


REDSOC:
Recycling Data Slack in OOO Cores

Gokul Subramanian Ravi

Prof. Mikko Lipasti

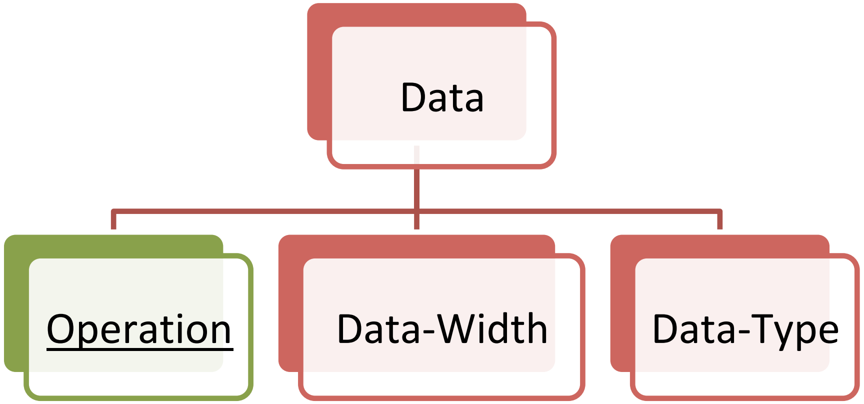
University of Wisconsin - Madison



- ➢ I [illegible] s of [illegible]
- ➢ Transparent Dataflow to recycle slack
- ➢ Optimized slack-aware OOO scheduler

DATA SLACK

BACKGROUND & CLASSIFICATION



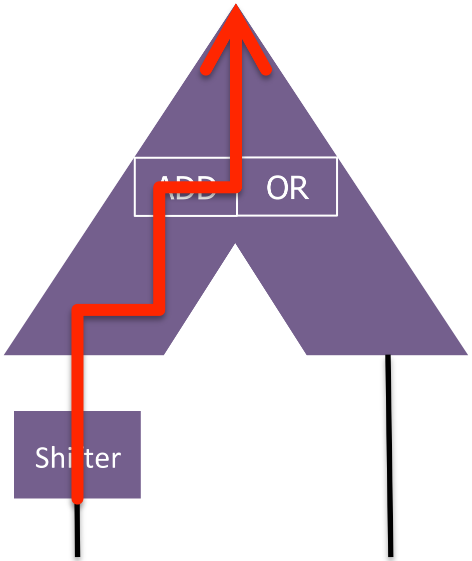
Data

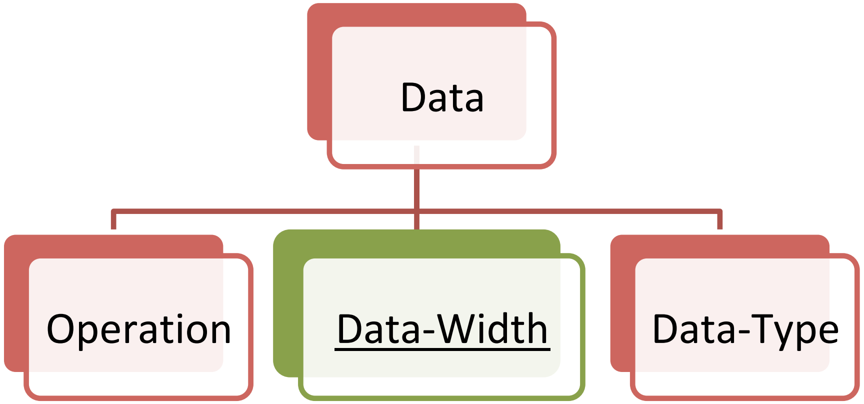
Operation

Data-Width

Data-Type

Arithmetic Logic Unit





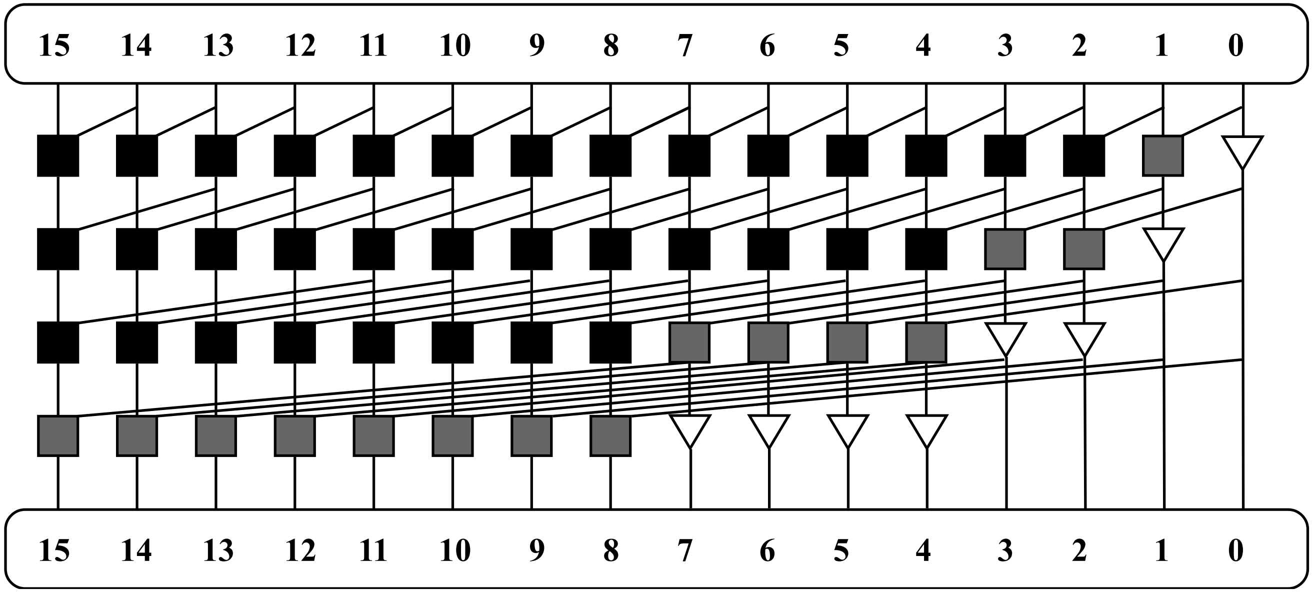
Data

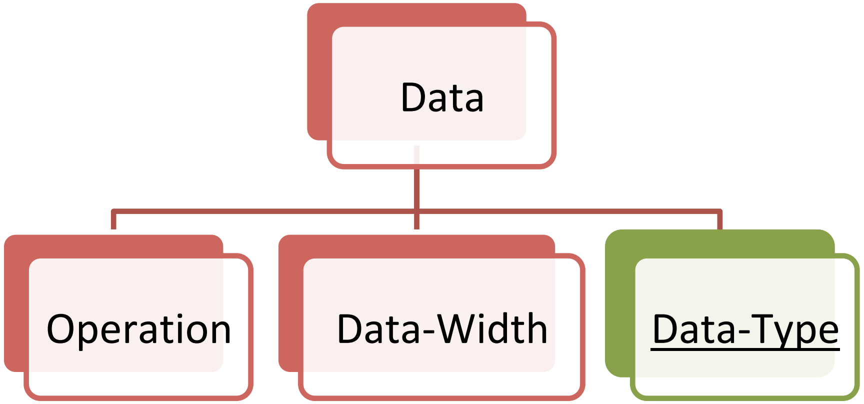
Operation

Data-Width

Data-Type

16-bit Kogge Stone Adder

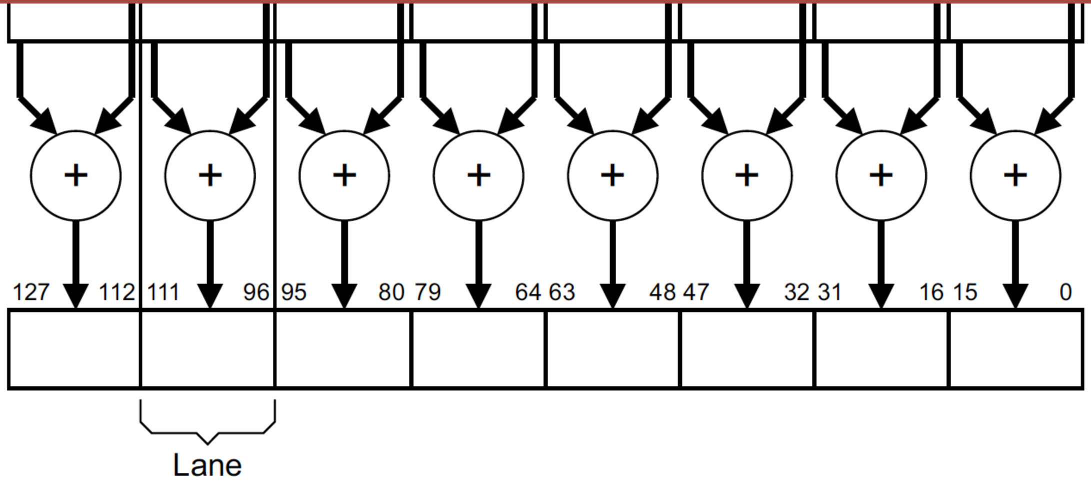




VADD.I16 Q0, Q1, Q2[1]

Details in paper!

Identifying data slack at decoder: instruction / prediction / lookup table

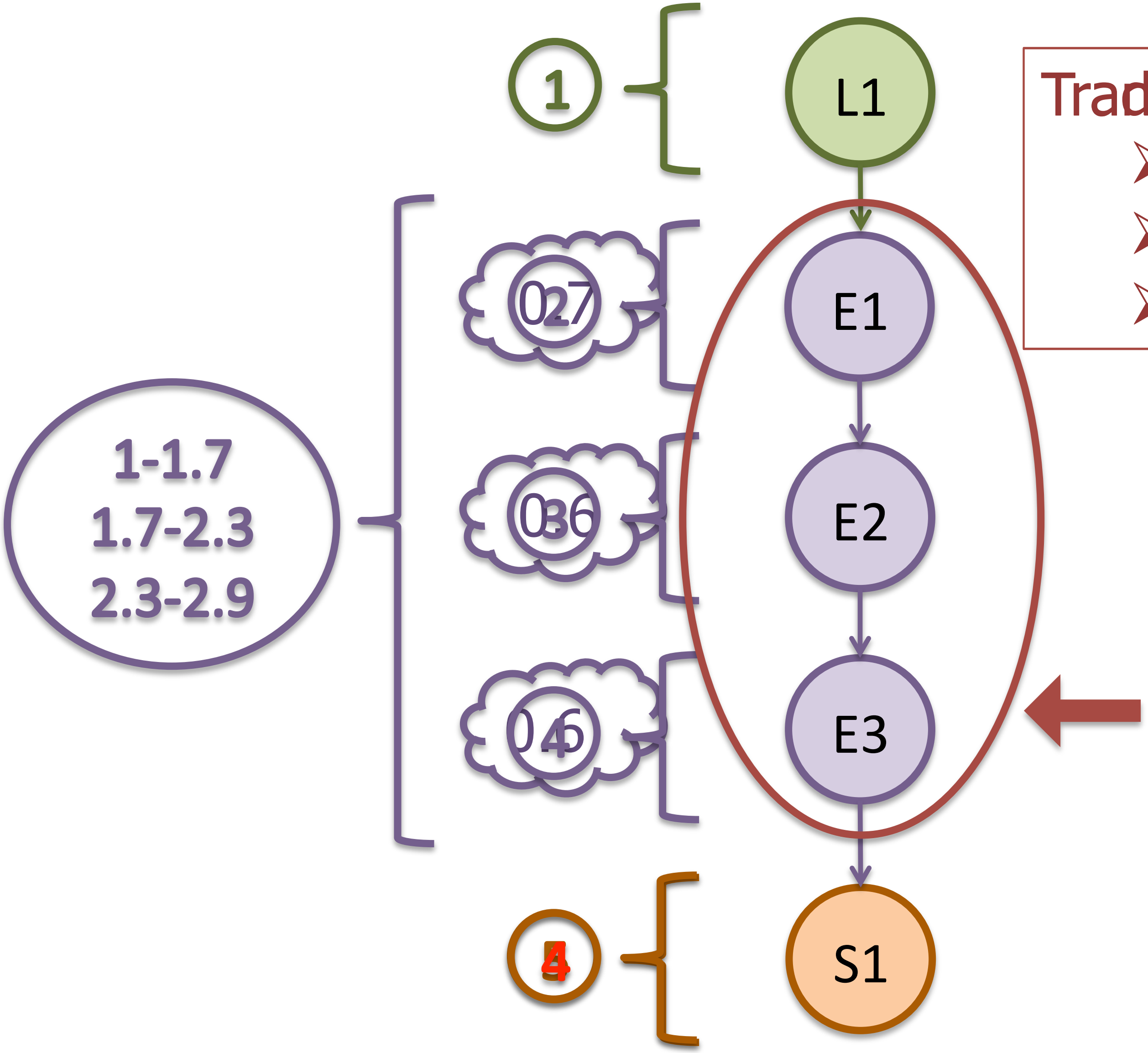


[1] ARM Ltd., 2009

EXECUTE STAGE

TRANSPARENT DATAFLOW

Transparence on a Dataflow Graph

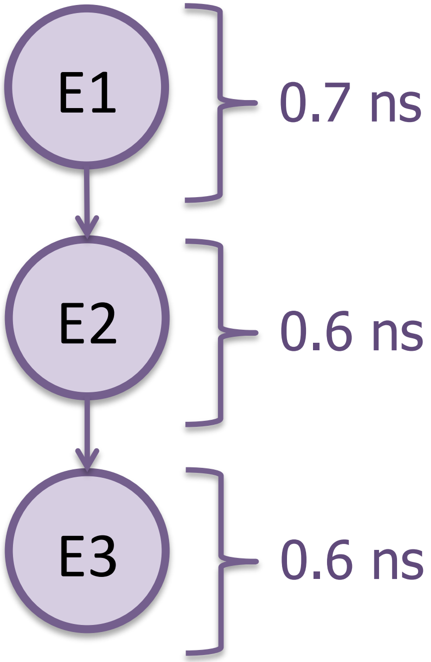


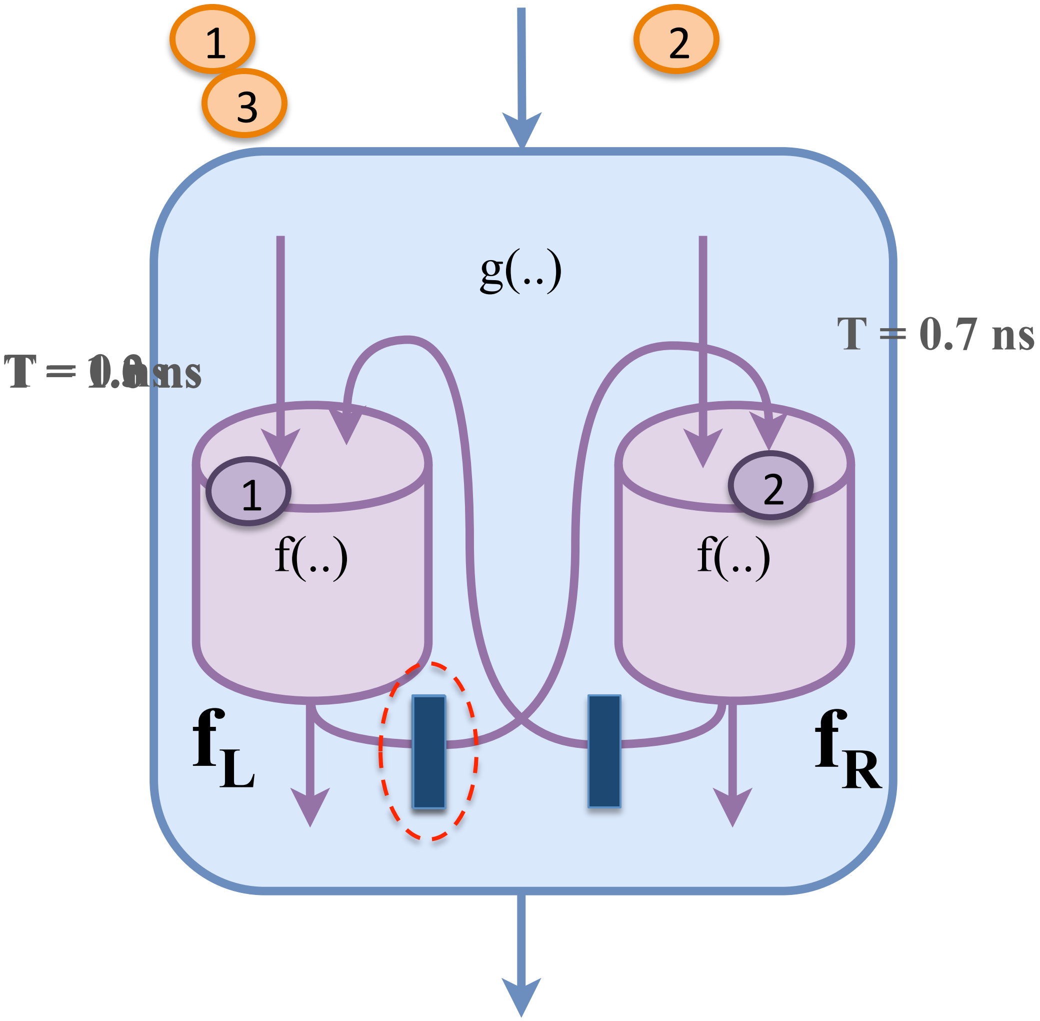
Transparence boundary

Traditional Dataflow:
Transparent Dataflow:

- ➢ Load Synchronous
- ➢ Exec Synchronous
- ➢ Exec Transparent
- ➢ Store Synchronous

Transparent Dataflow with Synchronous Control



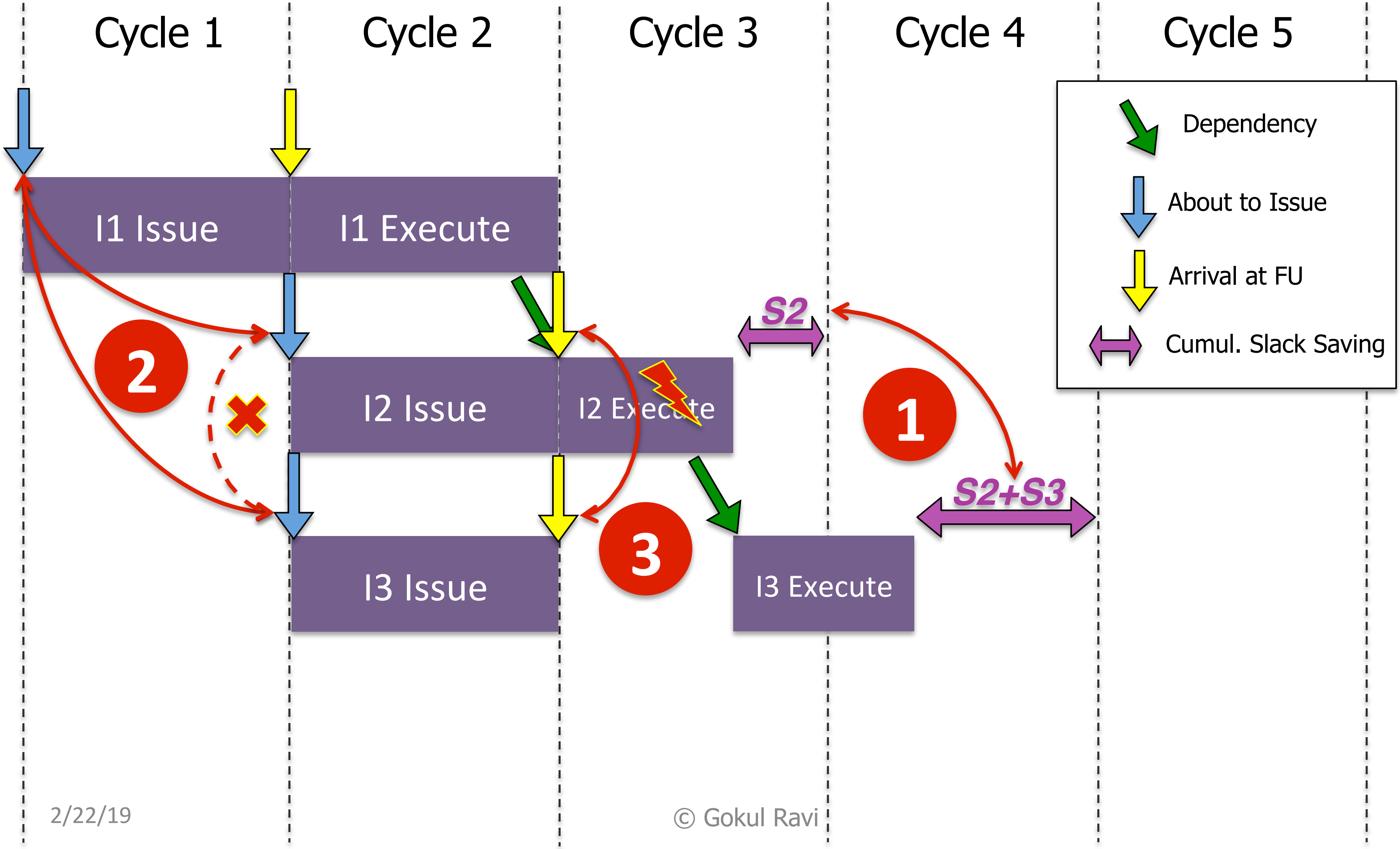


Clock Period = 1ns

INSTRUCTION SCHEDULER

SLACK INCORPORATED SCHEDULING

Slack-efficient Scheduler: Motivation



Slack-aware Scheduler: Proposal

Cycle 1 Cycle 2 Cycle 3 Cycle 4 Cycle 5

① Slack Accumulation

- ➢ Tracking and accumulating slack over dataflow graphs

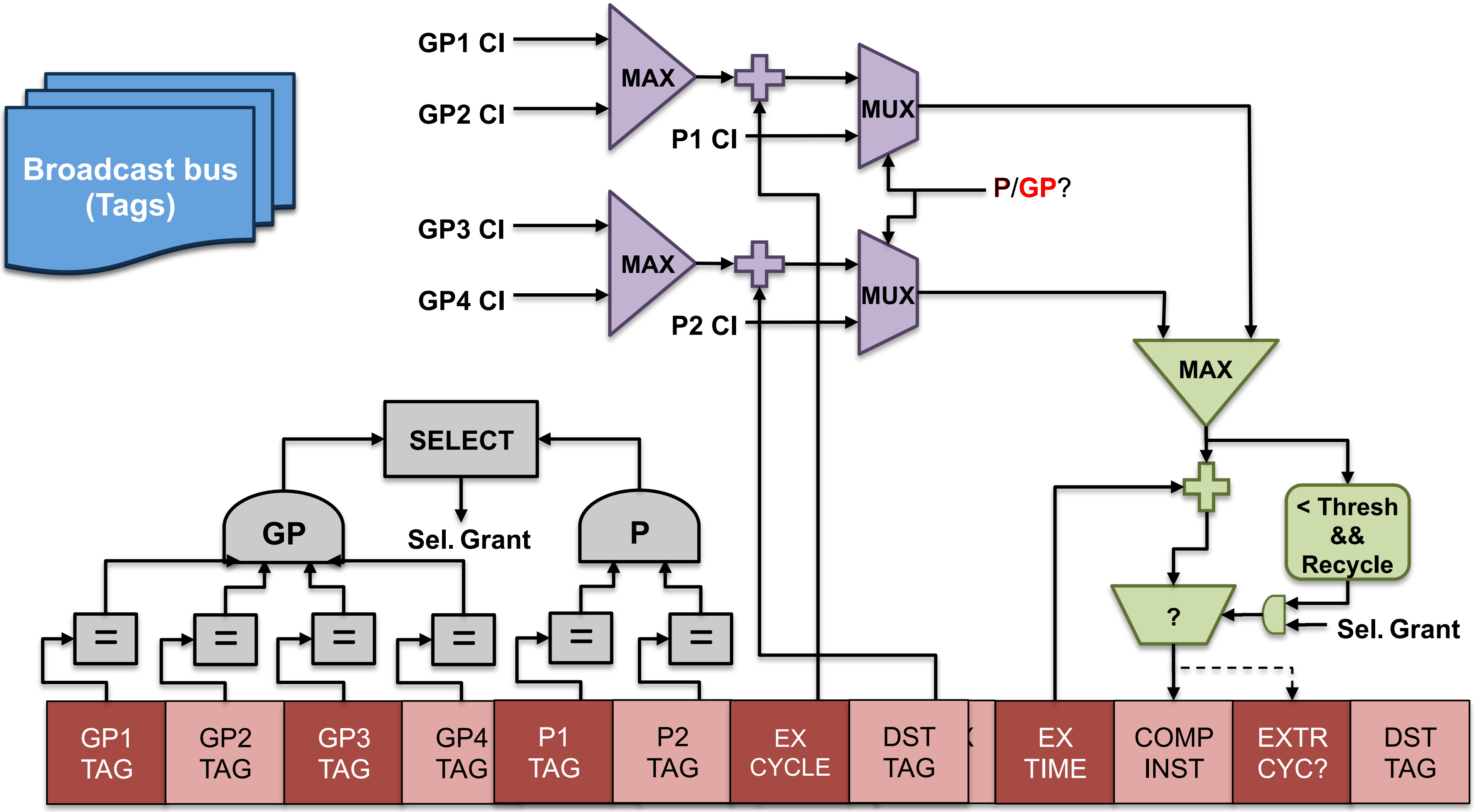
② Eager Grandparent Wakeup

- ➢ Speculative child wakeup via grand parent tags[Stark, 2000]

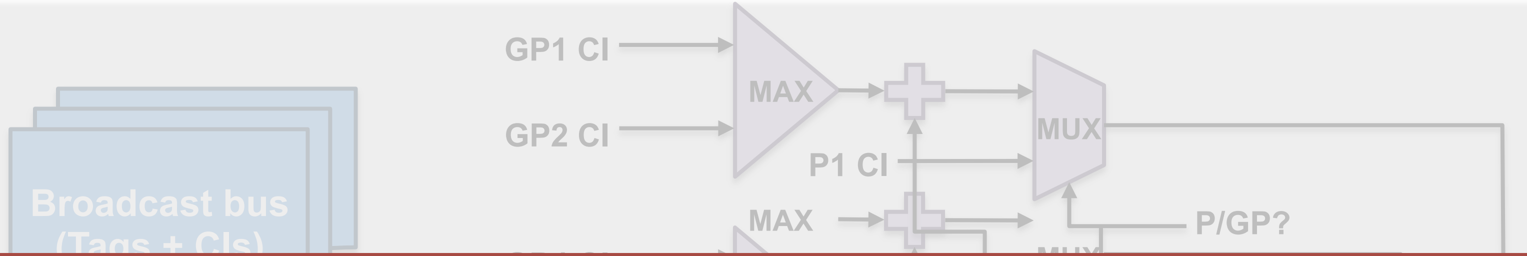
③ Skewed Selection logic

- ➢ Non-speculative parent preferred over speculative child

Scheduling Microarchitecture (illustrative)

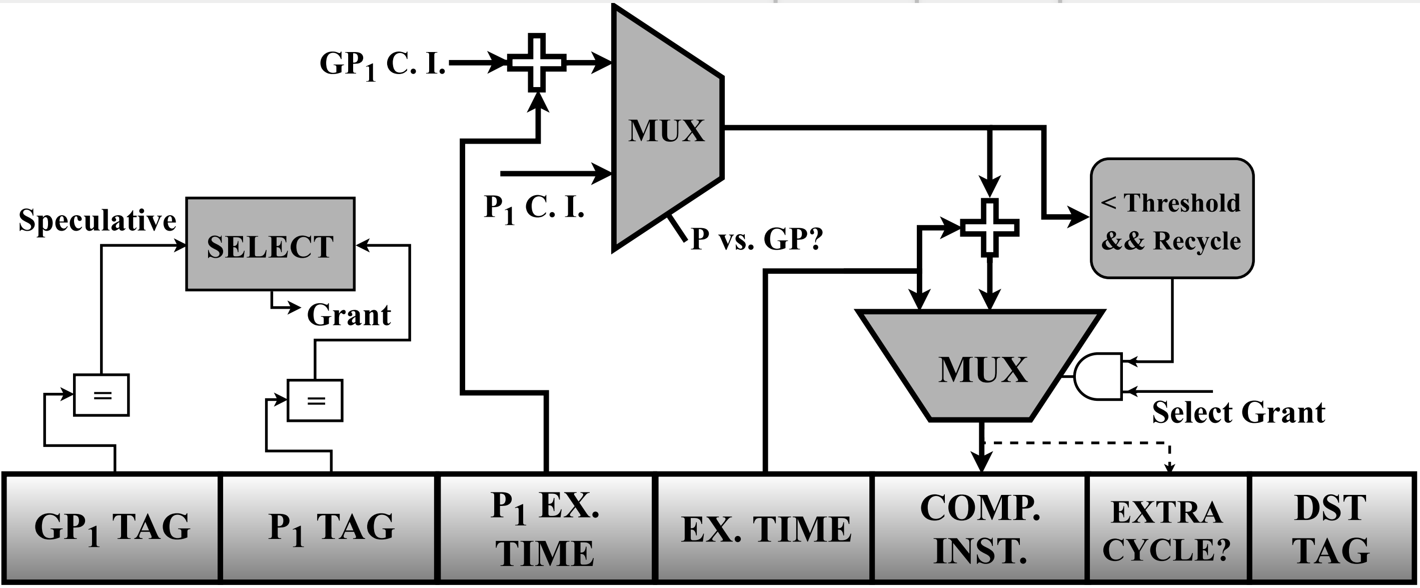


Scheduling Microarchitecture (Operational)



Details in paper!

Operational design with single predicted parent / grandparent!



EVALUATION

METHODOLOGY AND RESULTS

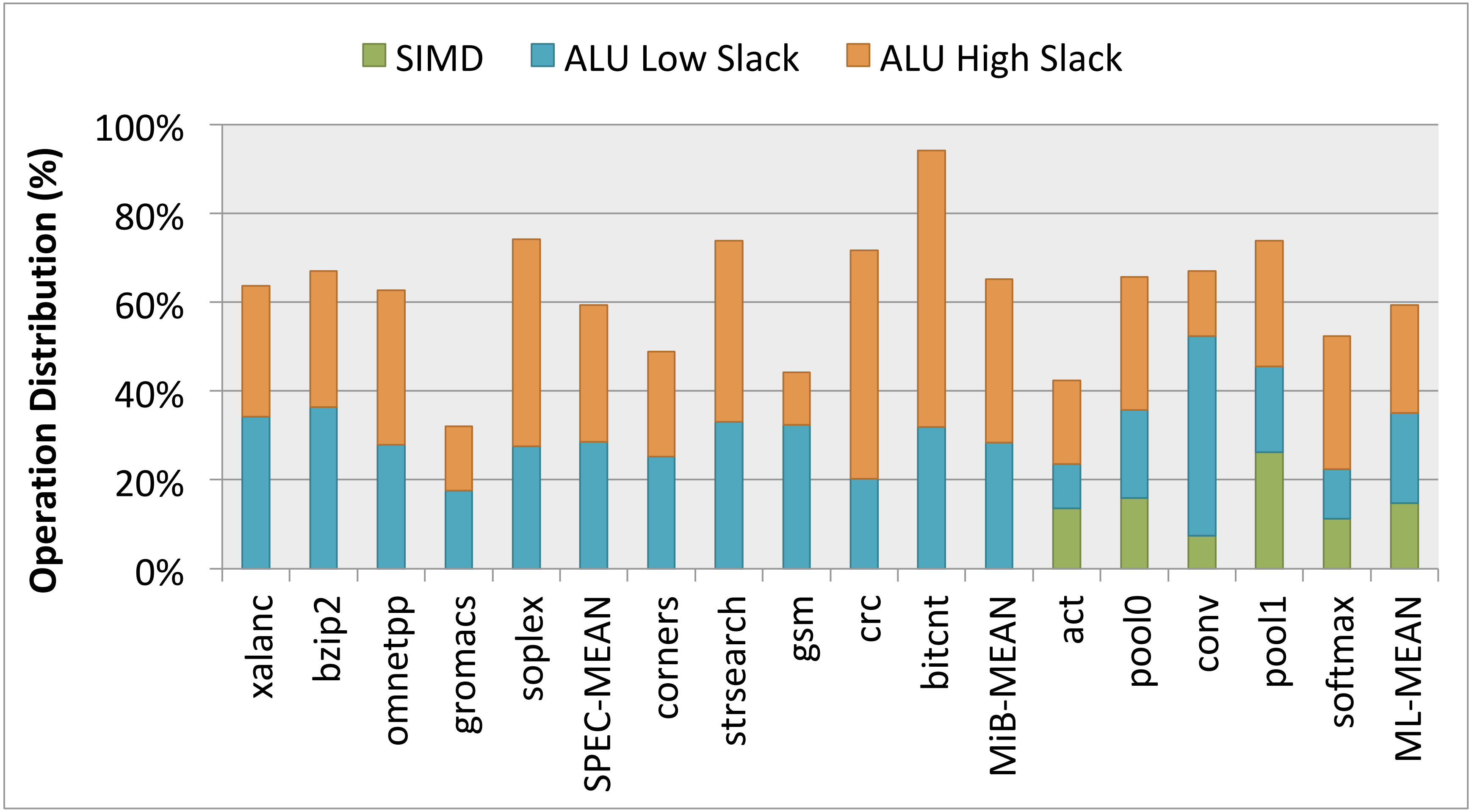
Experimental Setup

- Methodology:
  - ➢ Perf & Power: Gem5 + McPAT
  - ➢ Slack analysis: Synthesis on Synopsys Design Compiler
  - ➢ Freq & Tech: 2 GHz, 45nm

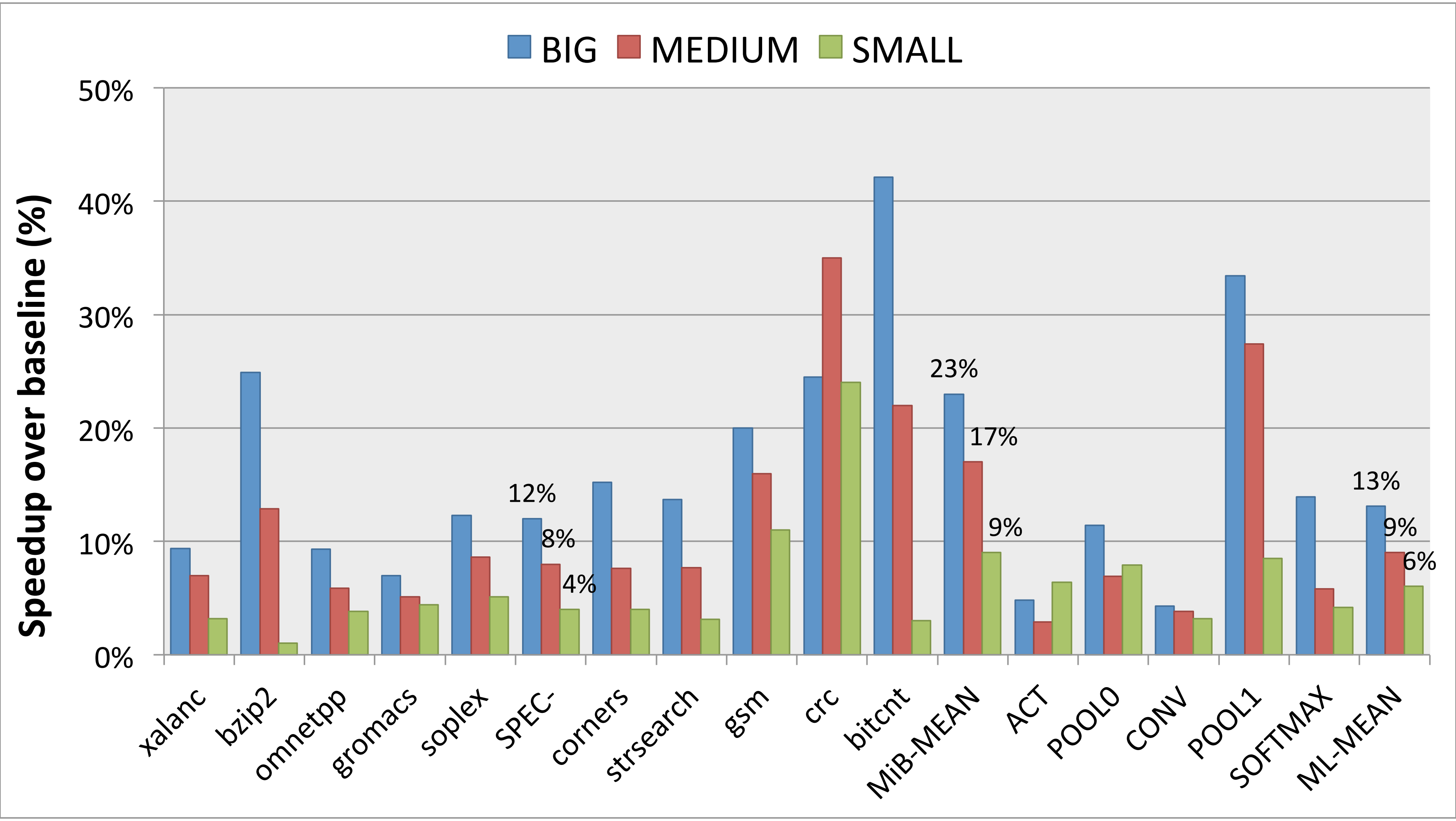
- Baseline:
  - ➢ 3 sizes of OOO cores
    - ▪ Front-end: 3/4/8
    - ▪ Execution: 3/4/6
    - ▪ 64K L1 IC/DC, 2M L2

- Benchmarks:
  - ➢ Compute intensive benchmarks from SPEC CPU2006 / MiBench / ARM Compute Library (ARM ISA)

Benchmark Operation Distribution

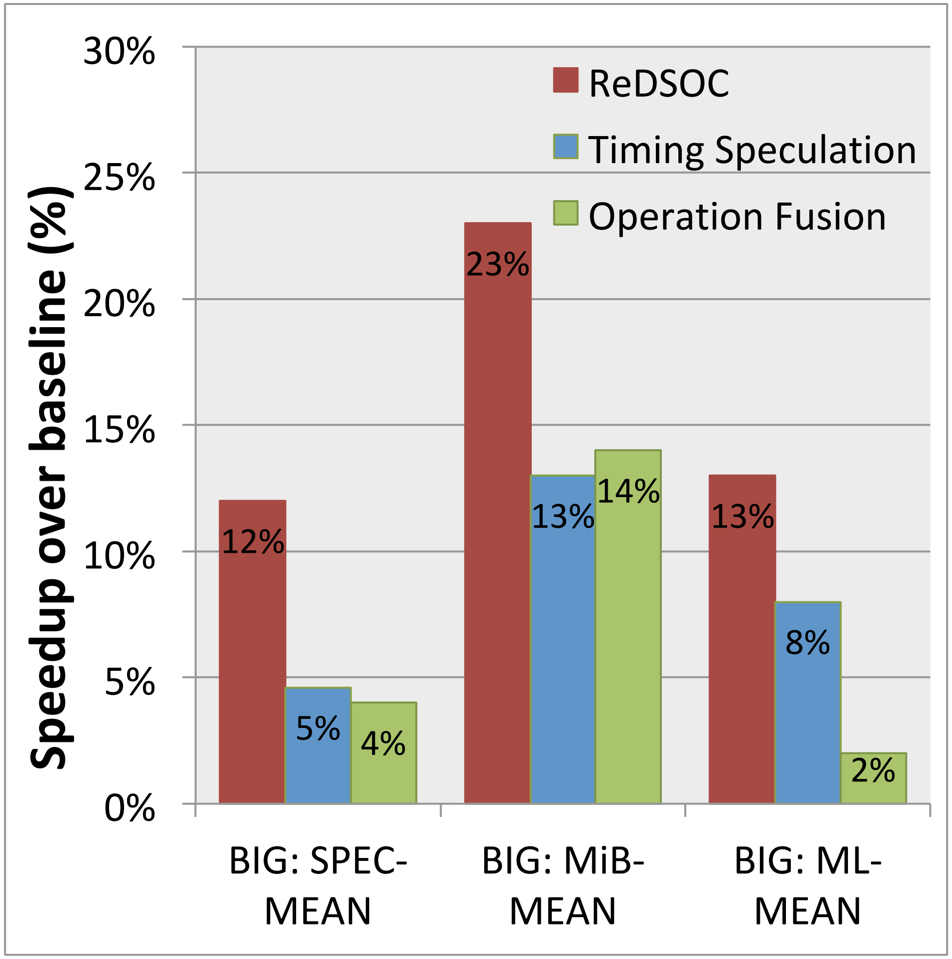


Speedup over different cores



© Gokul Ravi

Comparison with other proposals



- Timing Speculation
  - Increase frequency at fixed voltage, with timing errors
  - Error detection, coarse tuning granularity, potential error every operation
- Operation Fusion
  - Pairs of operations in single (standard) clock cycle
  - Low opportunity, costly compiler or h/w optimization

REDSOC Conclusions

- I hope I convinced you that
  - ➢ Data Slack is considerable
  - ➢ Transparent dataflow designs are attractive
  - ➢ OOO modifiable with reasonable overheads

- Advantages
  - ➢ No timing errors or detection
  - ➢ Instruction granularity control
  - ➢ Traditional cores/apps/compilers

Future work:

- ➢ Slack between FUs
- ➢ Approximate
- ➢ Other parts of core
- ➢ Other processing designs

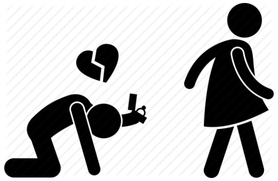
- Results
  - ➢ 5 to 25% performance improvement
  - ➢ 2x to 6x more efficient than prior proposals

Thank you!!

Questions??

Backup slides

Prior Proposals



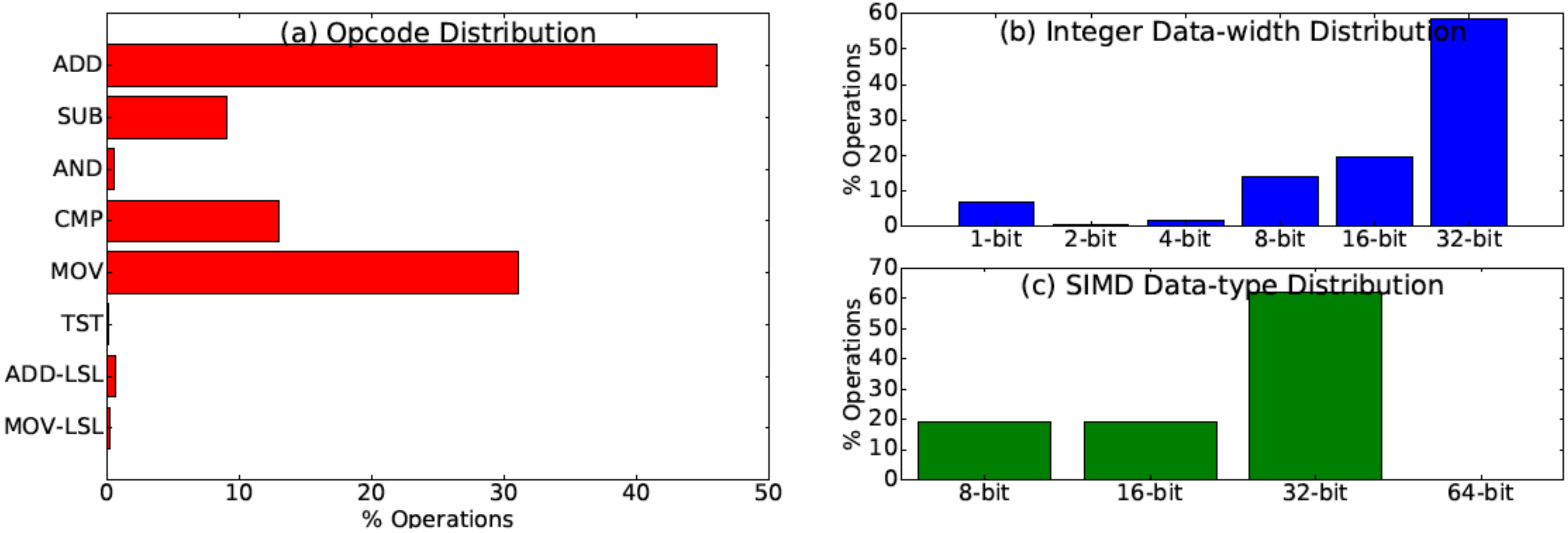
Prior Work	Description	Limitations
Elastic Pipelines [Nowick, 2011]	Asynchronous Blocks w/ handshake mechanisms	Completion detection / handshake overheads. High sync. integration costs
Specialized Data-paths [Sampson, 2011]	Single 'slow' cycle executing chained combinational ops.	Poor flexibility, low throughput or replication overheads
Operation Fusion [Park, 2009]	Sequence of operations in single (standard) clock cycle	Low opportunity, costly compiler or h/w optimization
Synchronous Timing Speculation	Increase/decrease frequency/ voltage, with timing errors	Error detection/recovery, high tuning overhead, potential error

Need for aggressive solutions w/ low (or no) risk, suited to general purpose compute!!

Processor Configurations

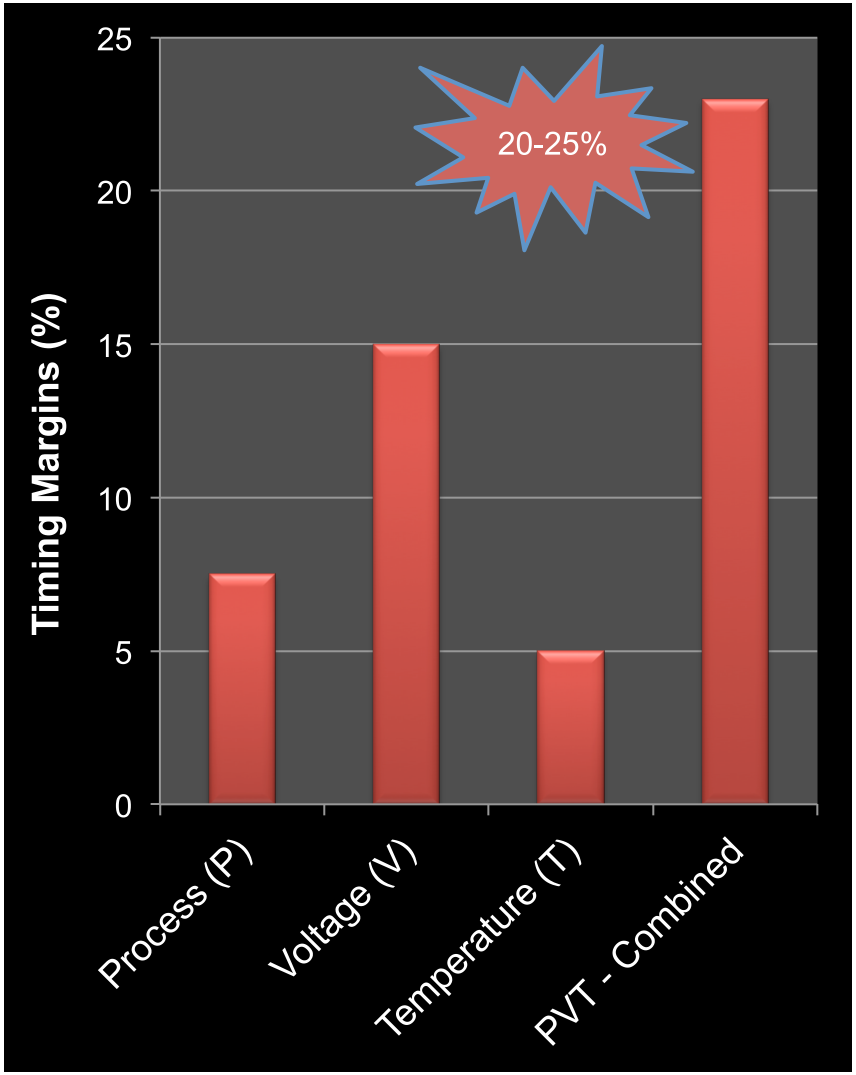
Parameter	Small	Medium	Big
Frequency	2 GHz	2 GHz	2 GHz
Front-End Width	3	4	8
ROB Size	40	80	160
LSQ Size	16	32	64
RSEs	32	64	128
ALUs	3	4	6
L1 I/D Cache	64 kB	64 kB	64 kB
L2 Cache	2 MB	2 MB	2 MB

Low-precision GEMM library



~50% of operations show 25-50% data slack!!

PVT Slack

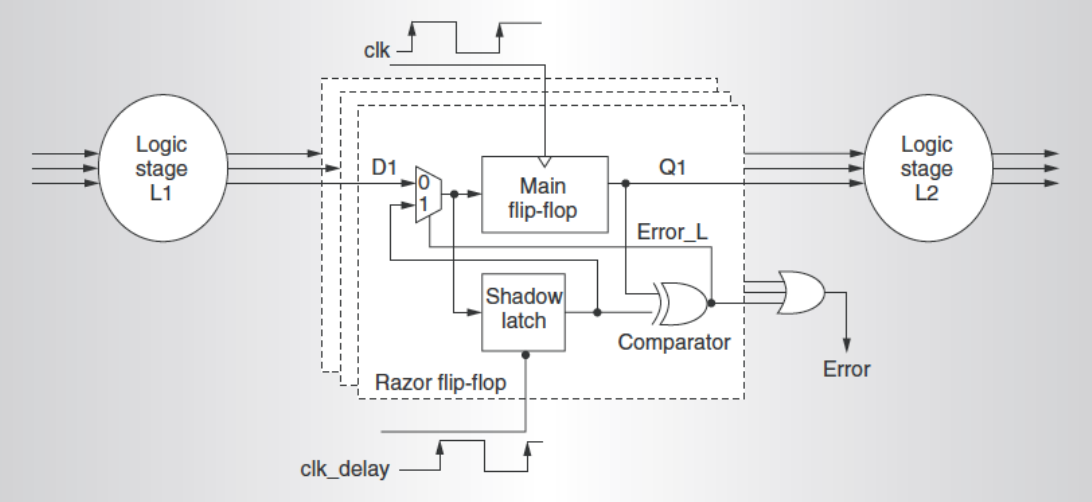


- Process:
  - Manufacturing variability
  - V_{th}, L_{gate}
- Voltage:
  - Current fluctuations
  - Workload activity
- Temperature:
  - Hotspots
  - Electron collisions

Tribeca, Gupta et al., MICRO ‘09

Prior Proposal #1: **Timing Speculation**

- Increase frequency OR reduce voltage allowing *some* timing errors to occur.



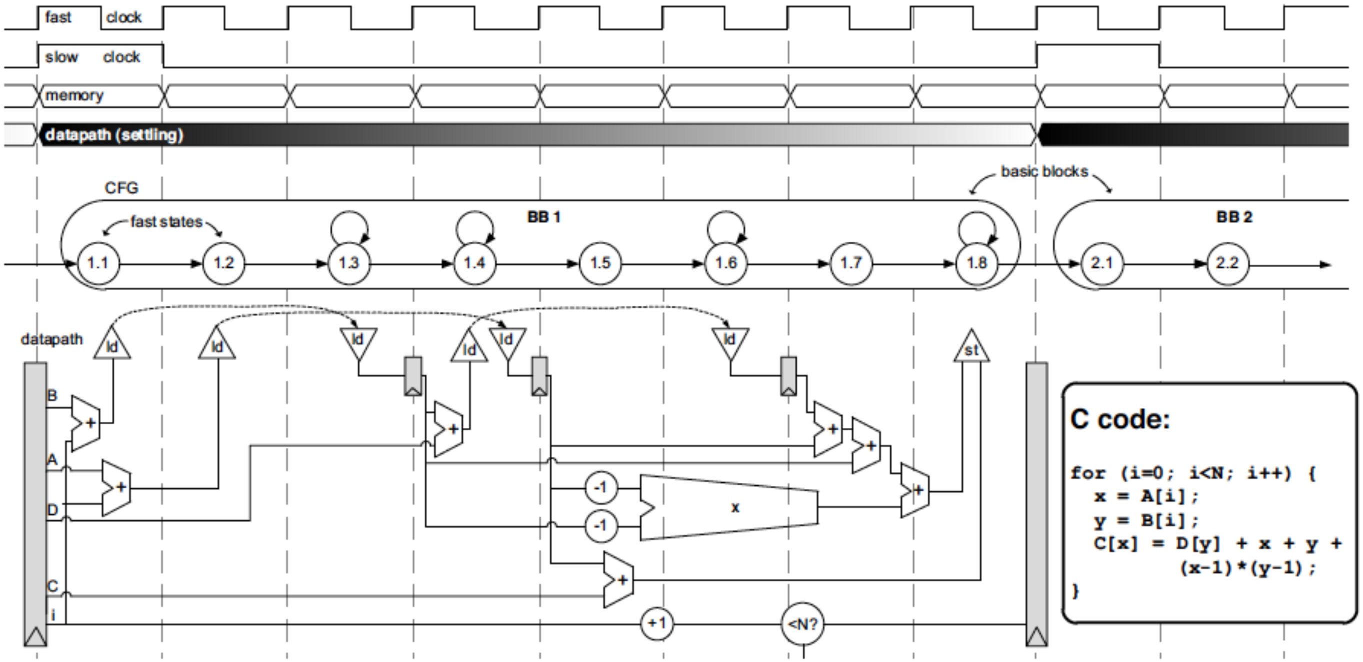
*[Ernst, 2003]

☠ Requires costly timing error detection, recovery mechanisms.

☠ Only allows coarse grained control – hence speculation is conservative (for low ER).

Prior Proposal #2: **Specialized Data Paths**

- Multi-cycle data path with sequence of combinational events executed in one "slow tick"



C code:

```
for (i=0; i<N; i++) {
  x = A[i];
  y = B[i];
  C[x] = D[y] + x + y +
         (x-1)*(y-1);
}
```

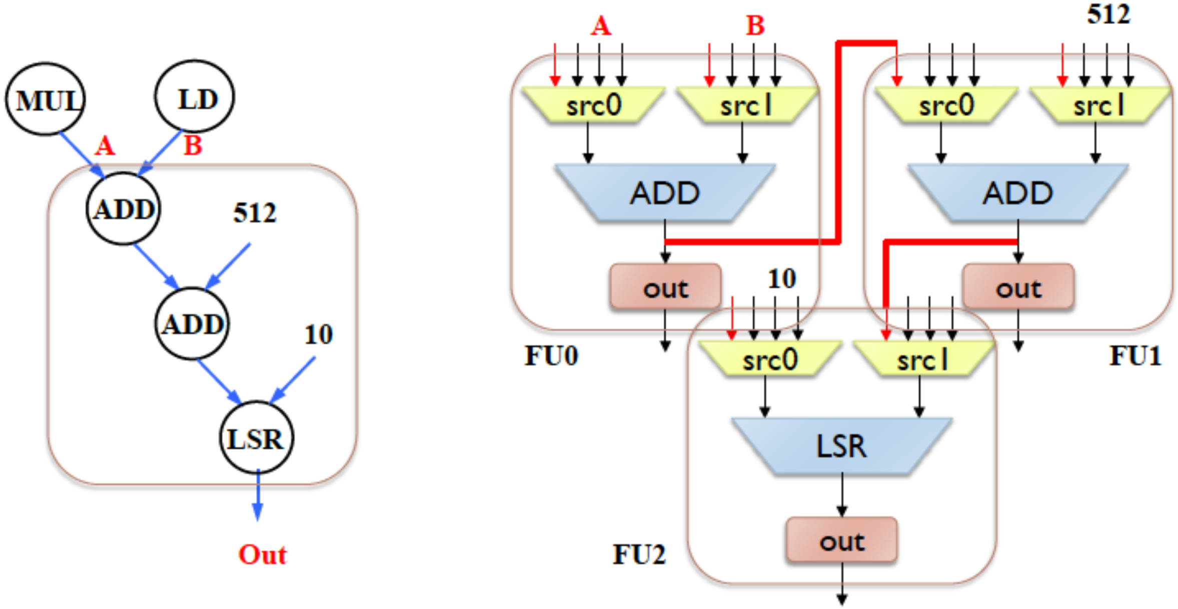
*[Sampson, 2011]

☠ Poor throughput or significant replication overheads

☠ No flexibility for general purpose processing

Prior Proposal #3: **Operation Fusion**

- Squeeze a sequence of operations into a single (standard) clock cycle

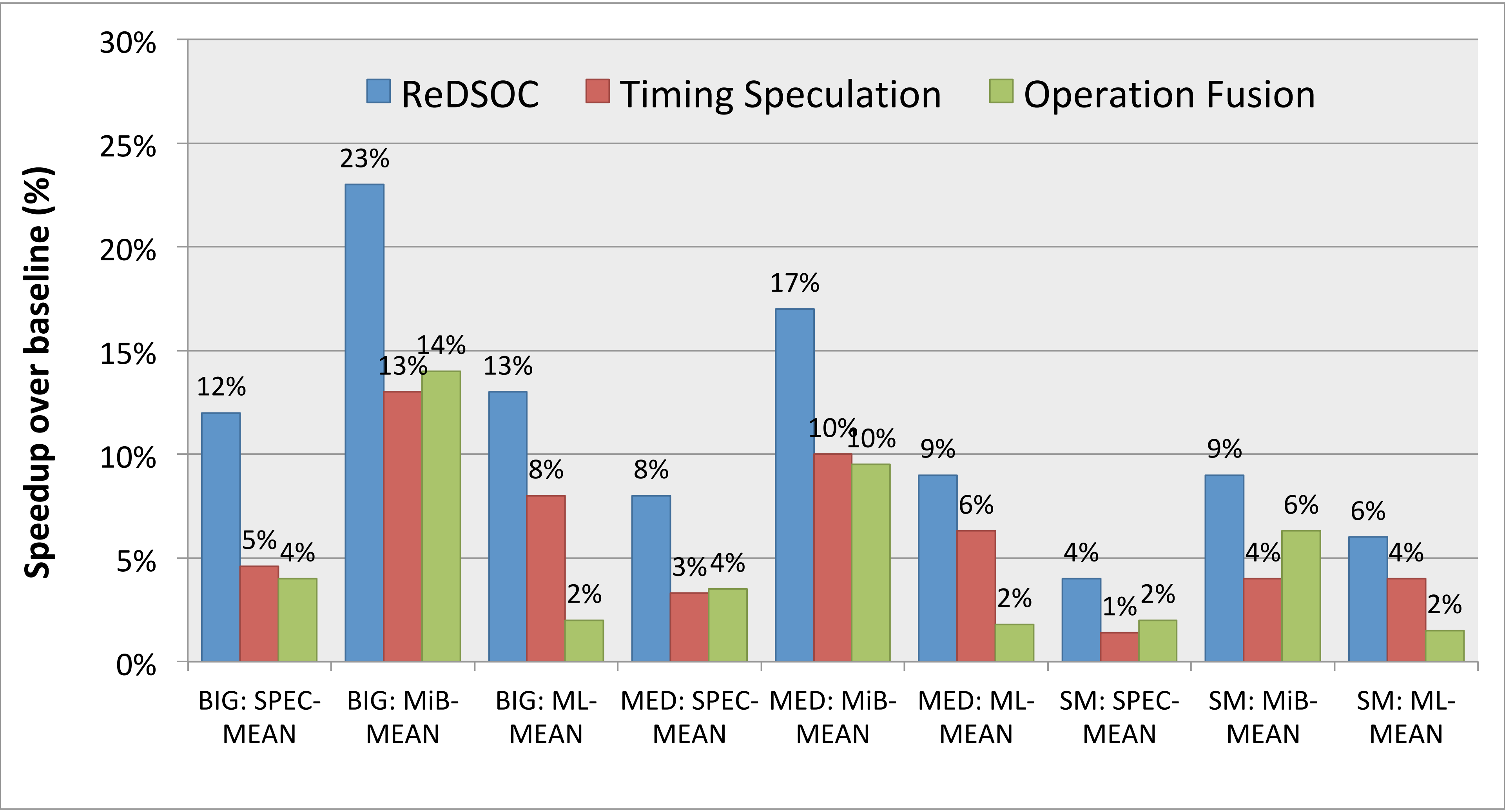


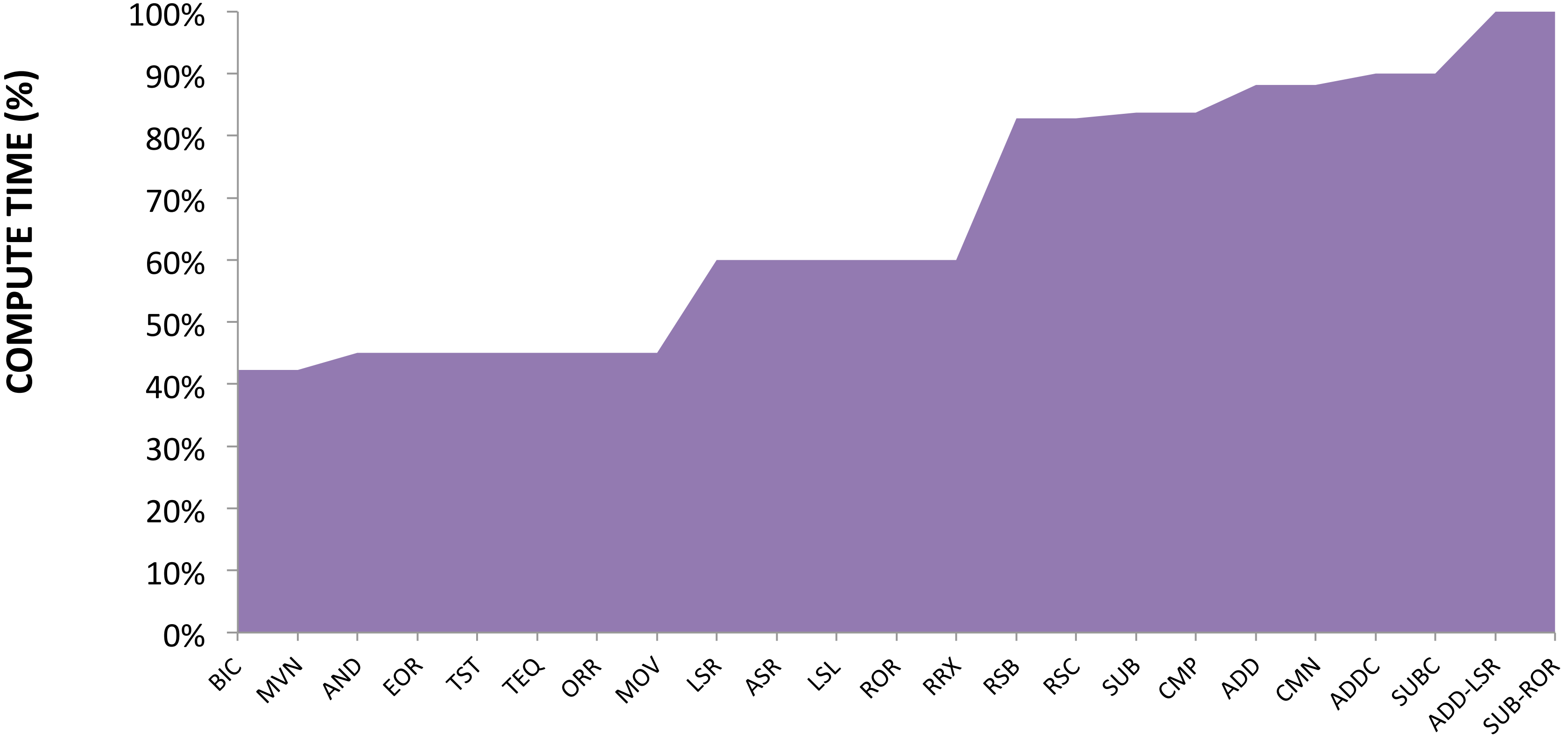
*[Park, 2009]

☠ Low opportunity in un-optimized code

☠ Costly compiler or hardware optimizations to attempt significant operations reordering

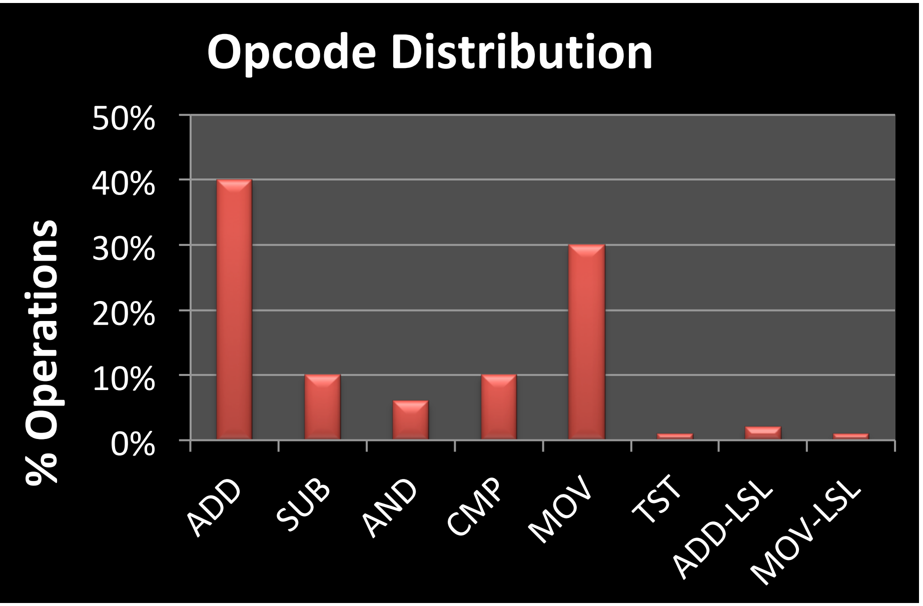
Comparison with other proposals



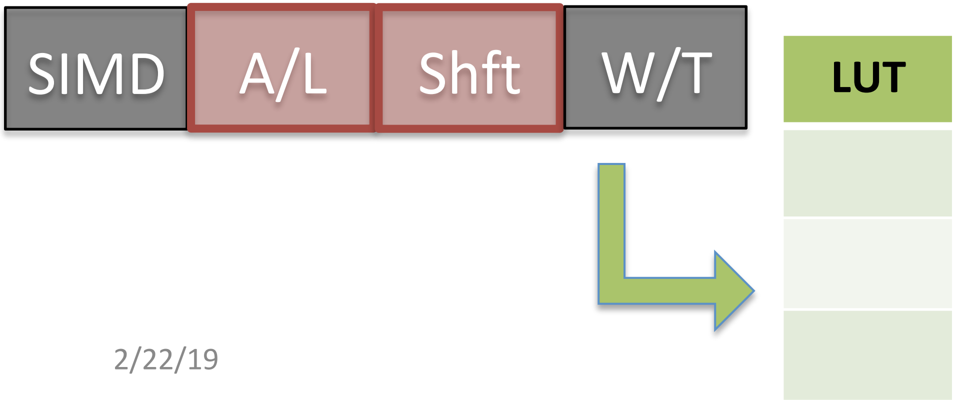

COMPUTE TIME (%)
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
BIC
MVN
AND
EOR
TST
TEQ
ORR
MOV
LSR
ASR
LSL
ROR
RRX
RSB
RSC
SUB
CMP
ADD
CMN
ADDC
SUBC
ADD-LSR
SUB-ROR

Data Slack Classification

LP-GEMM

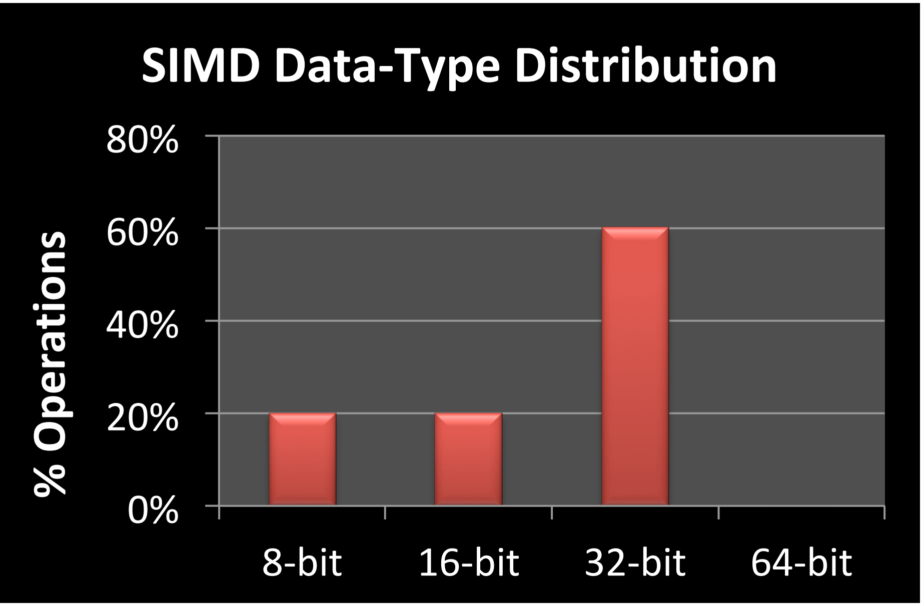


- Operation Slack:
  - ➢ Encoded within instruction
  - ➢ Obtained via decode

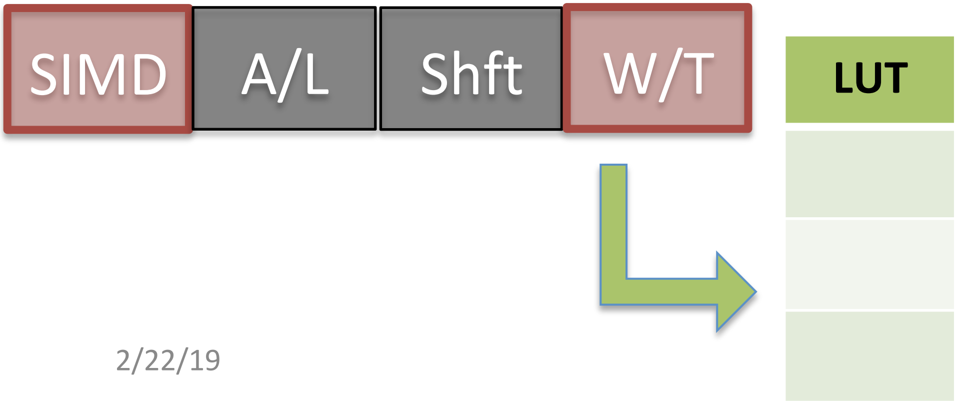


Data Slack Classification

LP-GEMM



- Operation Slack:
  - Encoded within instruction
  - Obtained via decode
- Data-Type Slack:
  - Encoded within instruction
  - Obtained via decode

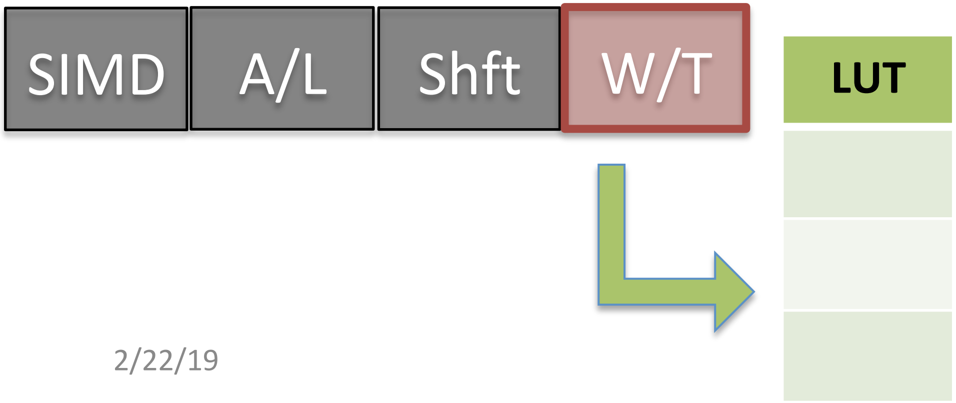


Data Slack Classification

LP-GEMM

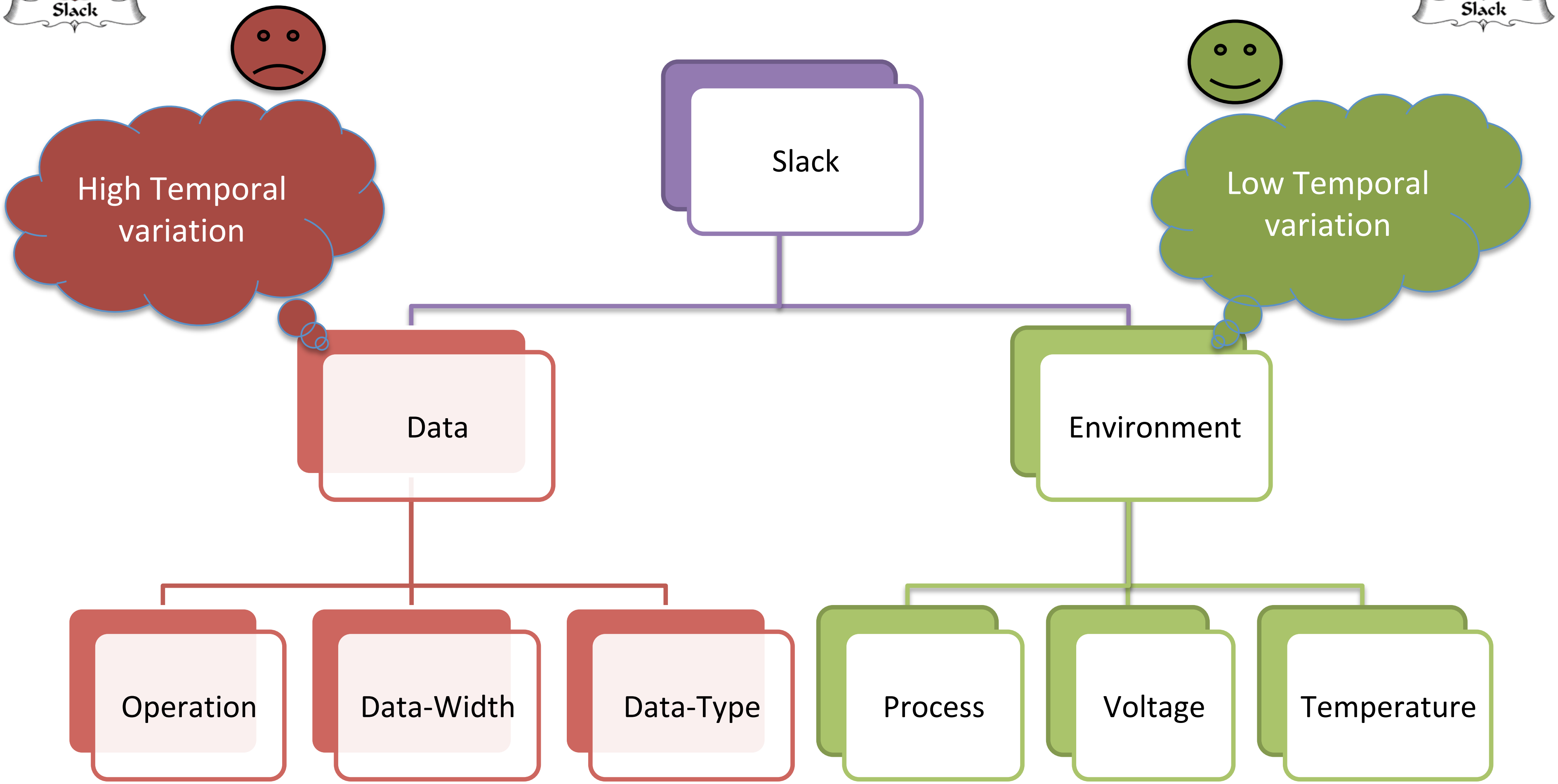
- Operation Slack:
  - Encoded within instruction
  - Obtained via decode

- Data-Type Slack:
  - Encoded within instruction
  - Obtained via decode



- Data-Width Slack:
  - From operands (too late)
  - Predict at decode[Loh, 2002]
  - Verify at execute

House of Slack



High Temporal variation

Low Temporal variation

Slack

Data

Environment

Operation

Data-Width

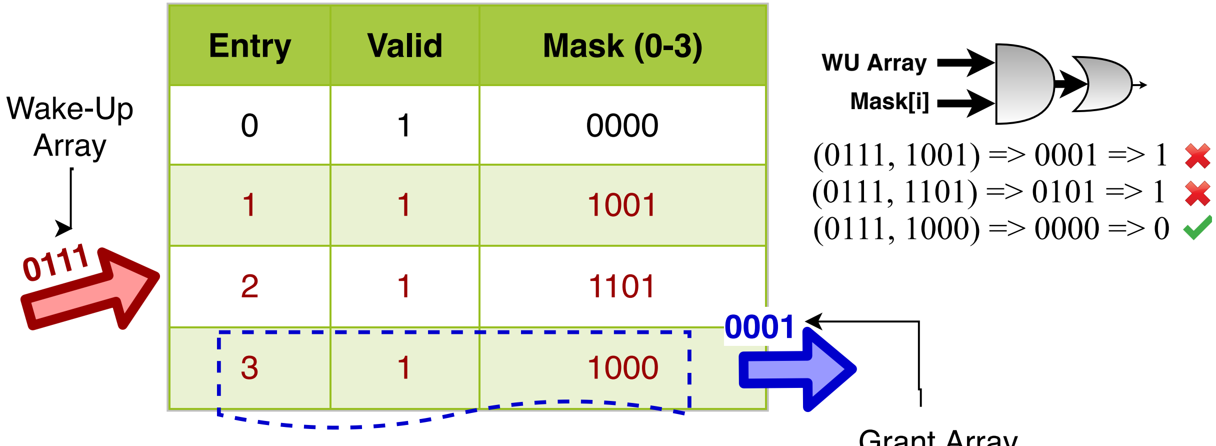
Data-Type

Process

Voltage

Temperature

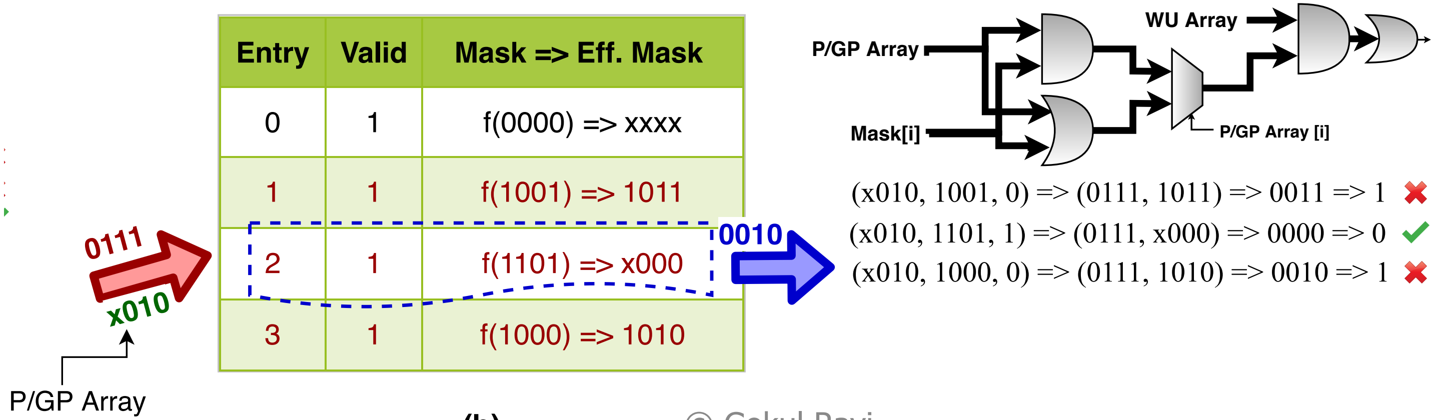
Skewed Select logic



Entry	Valid	Mask (0-3)
0	1	0000
1	1	1001
2	1	1101
3	1	1000

(0111, 1001) => 0001 => 1 ✗
(0111, 1101) => 0101 => 1 ✗
(0111, 1000) => 0000 => 0 ✓

(a)



Entry	Valid	Mask => Eff. Mask
0	1	f(0000) => xxxx
1	1	f(1001) => 1011
2	1	f(1101) => x000
3	1	f(1000) => 1010

(x010, 1001, 0) => (0111, 1011) => 0011 => 1 ✗
(x010, 1101, 1) => (0111, x000) => 0000 => 0 ✓
(x010, 1000, 0) => (0111, 1010) => 0010 => 1 ✗

(b)

© Gokul Ravi

Overheads

- Decode:
  - Width predictor uses 1.5KB of state
  - Area/Energy 0.5% of core
- Execute:
  - Negligible
- Scheduler:
  - Slack computations 3-bits wide
  - Operational design is 10 extra bits per RSE
  - Area/Energy overhead: 0.3/0.8%
  - Increase in scheduler delay is 1.5% (pessimistic)

Timing Closure in Execute

- Traditional timing paths (in a standard FF design) to analyze for timing closure would be (F1i–F1o), (F2i–F2o), (F1o–F2o) and (F2o – F1o).

- For transparent, these would be (F1i –F2o) and (F2o –F2o) when M12 is enabled for transparent dataflow. Similarly, there would be (F2i – F1o) and (F1o – F1o) when M21 is enabled for transparent dataflow.