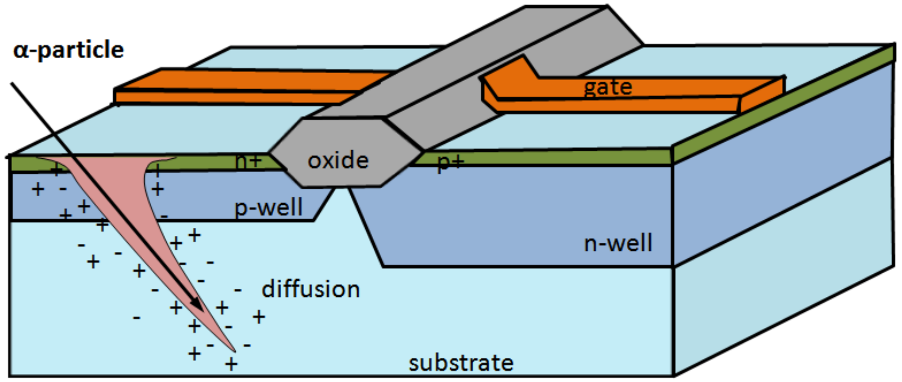
FluidCheck: A Redundant Threading based Approach for Reliable Execution in Manycore Processors

Rajshekar Kalayappan, Smruti R. Sarangi
Dept of Computer Science and Engineering
Indian Institute of Technology Delhi
New Delhi, India.

Soft Errors

- Temporary nature



[img src : aviral.lab.asu.edu]

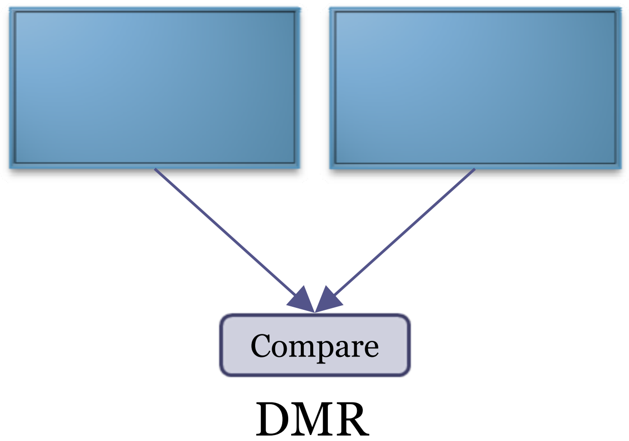
- Occurs due to particle strikes on the silicon
- Source of particles :
  - Solar ion flux
  - Explosion of distant stars
  - Impurities in the chip

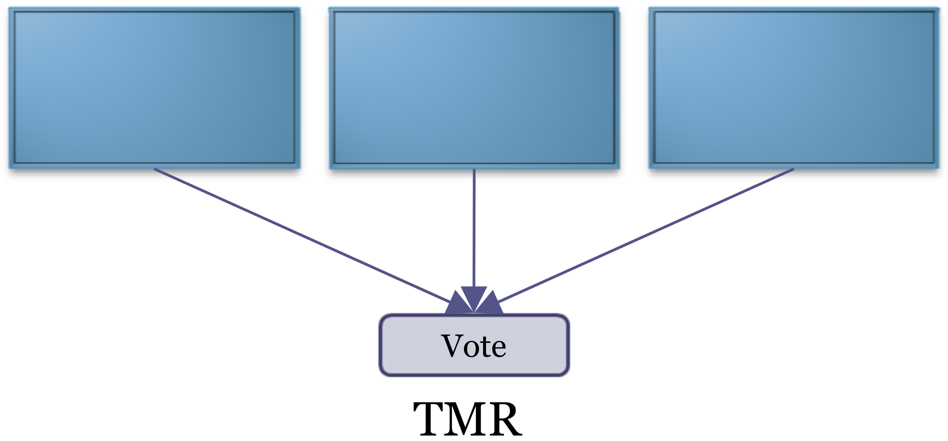
Soft Errors

- Rare event
  - Particles need to strike at the right place, at the right angle, with the right amount of energy
- Not rare enough to be ignored
  - The critical charge required to flip a bit reduces with reducing feature size and operating voltage

Soft Errors

- Solutions
  - Device level radiation hardening
    - Two to four generations behind commercial counterparts [Courtland2015]
  - System level hardening techniques required
    - Redundancy



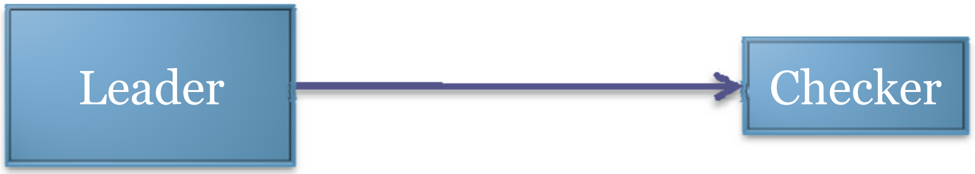


Problem Statement

- To *efficiently* execute a set of applications on a chip multi-processor (homogeneous SMT-capable cores), while ensuring *reliability* in the face of soft errors

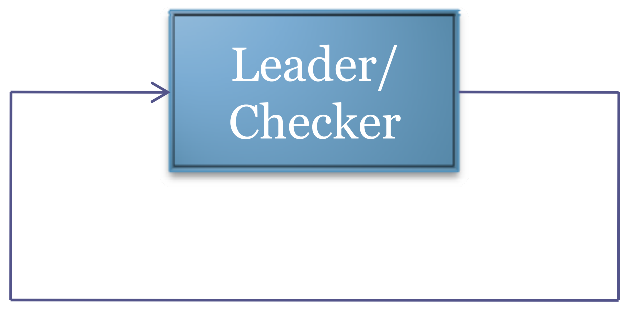
Related Work : DIVA [Austin1999]

•Meant to provide reliability.



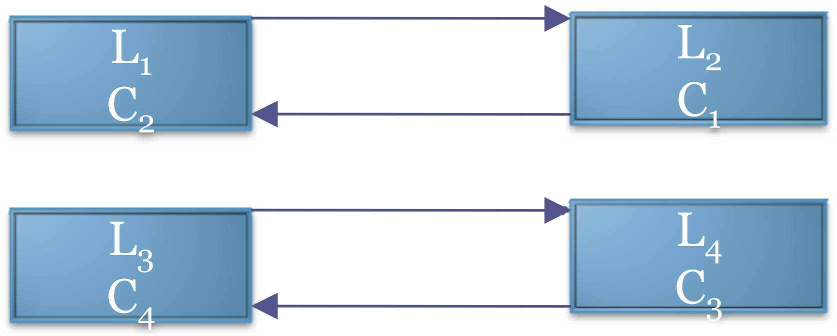
- IP
- Execution Assistance :
  - Branch Prediction Hints
  - Operand Value Hints
- Result
- Example
  <0x1234><op1=5><op2=2><res=7>
- Cache line forwarding

Related Work



SRT [Reinhardt2000], AR-SMT [Rotenberg1999]

•Saves area
•Better throughput per core

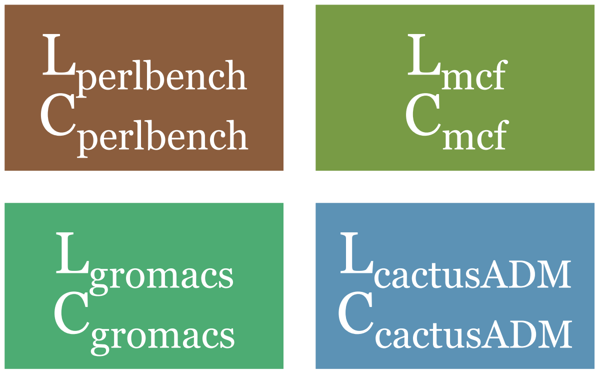


CRT [Mukherjee2002]

•Improvement over SRT
•Circumvents hazards borne out of resource requirement similarity between a leader-checker pair
•Better throughput per core

Motivational Example

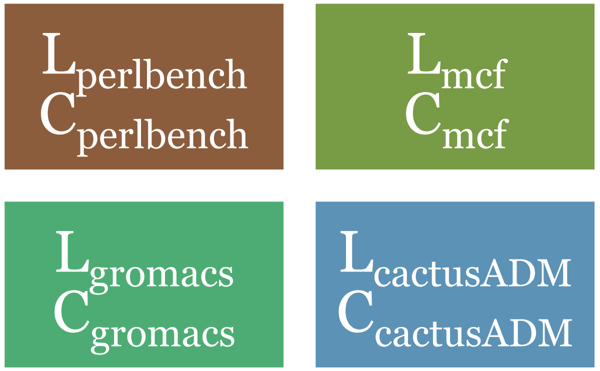
Without any checking, throughput = 4.84 instructions per cycle


$L_{perlbench}$ $C_{perlbench}$
L_{mcf} C_{mcf}
$L_{gromacs}$ $C_{gromacs}$
$L_{cactusADM}$ $C_{cactusADM}$

SRT

Motivational Example

Without any checking, throughput = 4.84 instructions per cycle

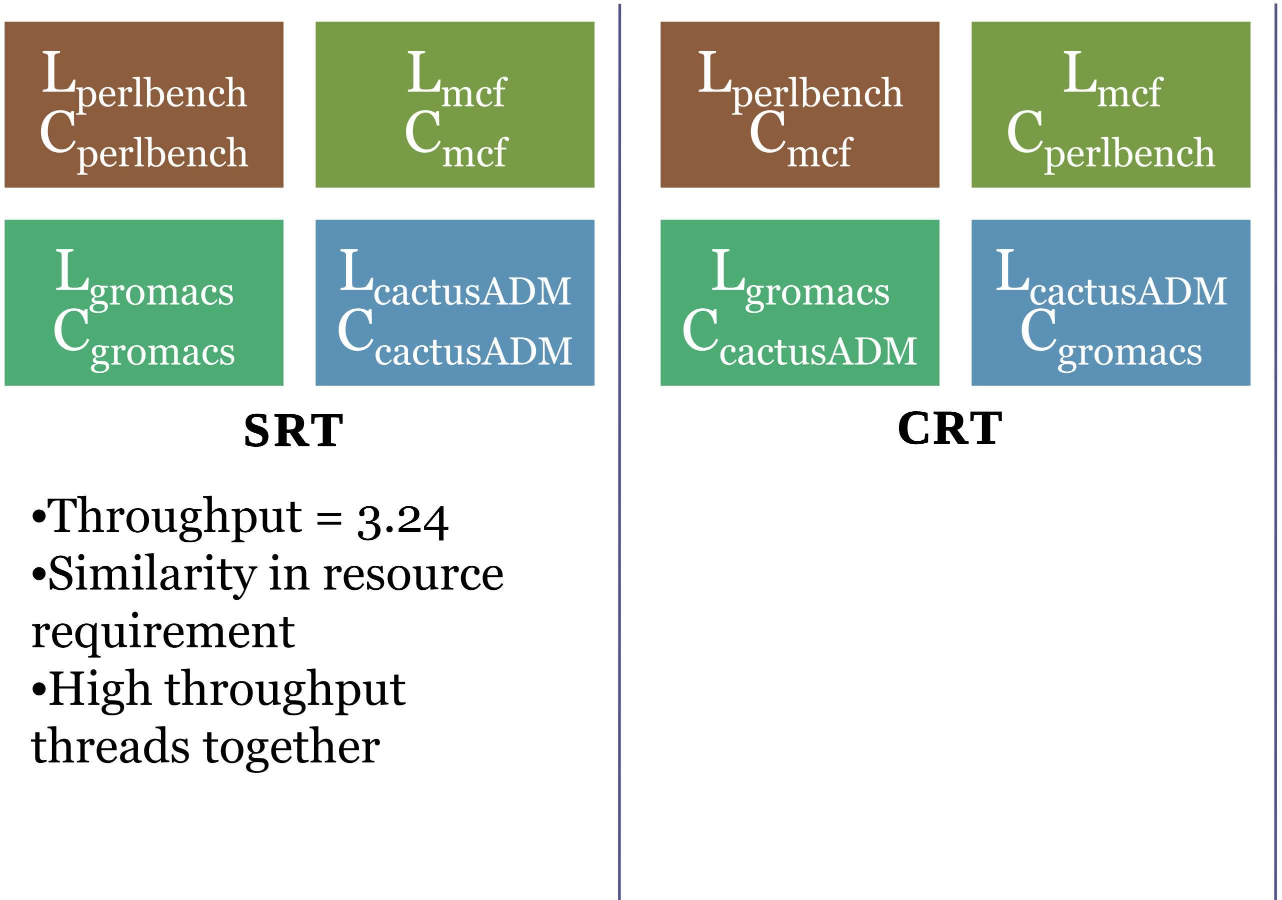


SRT

- Throughput = 3.24
- Similarity in resource requirement
- High throughput threads together

Motivational Example

Without any checking, throughput = 4.84 instructions per cycle


$L_{perlbench}$ $C_{perlbench}$
L_{mcf} C_{mcf}
$L_{gromacs}$ $C_{gromacs}$
$L_{cactusADM}$ $C_{cactusADM}$
$L_{perlbench}$ C_{mcf}
L_{mcf} $C_{perlbench}$
$L_{gromacs}$ $C_{cactusADM}$
$L_{cactusADM}$ $C_{gromacs}$

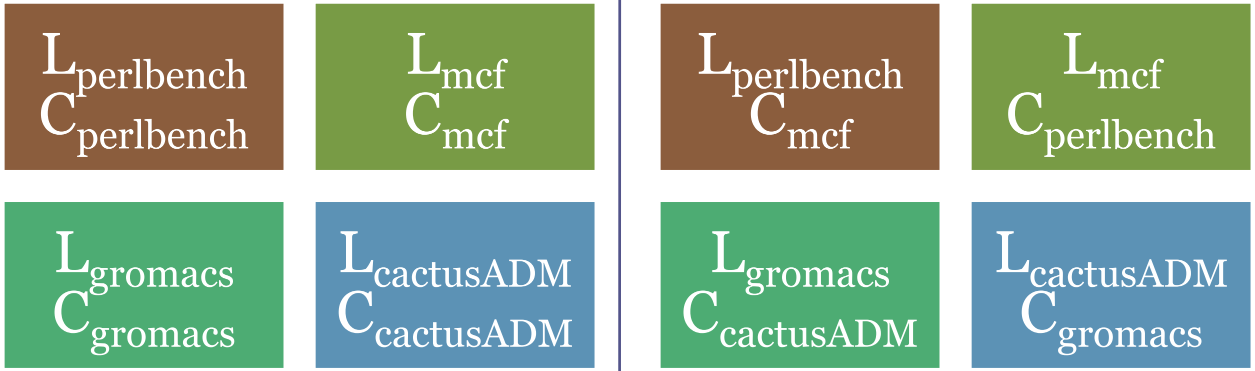
SRT

CRT

- Throughput = 3.24
- Similarity in resource requirement
- High throughput threads together

Motivational Example

Without any checking, throughput = 4.84 instructions per cycle


$L_{perlbench}$ $C_{perlbench}$
L_{mcf} C_{mcf}
$L_{gromacs}$ $C_{gromacs}$
$L_{cactusADM}$ $C_{cactusADM}$
$L_{perlbench}$ C_{mcf}
L_{mcf} $C_{perlbench}$
$L_{gromacs}$ $C_{cactusADM}$
$L_{cactusADM}$ $C_{gromacs}$

SRT

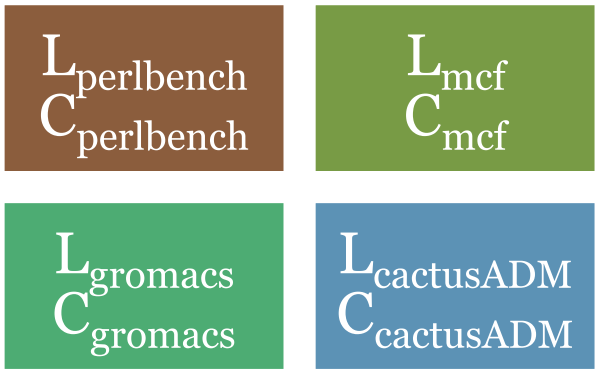
- Throughput = 3.24
- Similarity in resource requirement
- High throughput threads together

CRT

- Throughput = 3.55
- Similarity is broken
- Can we do better?

Motivational Example

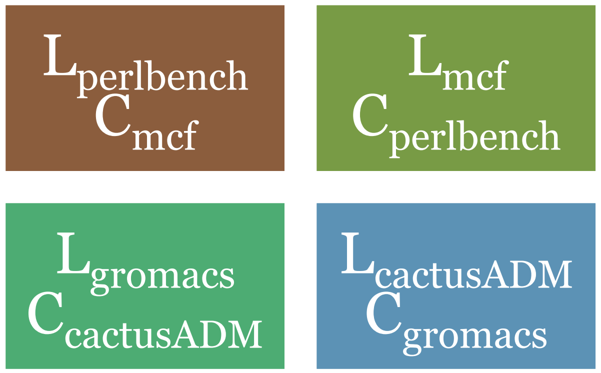
Without any checking, throughput = 4.84 instructions per cycle



$L_{perlbench}$ $C_{perlbench}$
L_{mcf} C_{mcf}
$L_{gromacs}$ $C_{gromacs}$
$L_{cactusADM}$ $C_{cactusADM}$

SRT

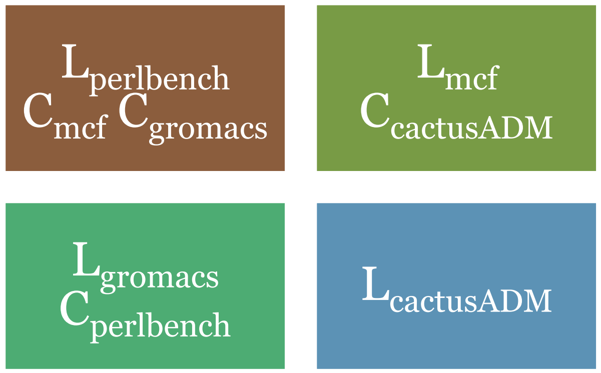
- Throughput = 3.24
- Similarity in resource requirement
- High throughput threads together



$L_{perlbench}$ C_{mcf}
L_{mcf} $C_{perlbench}$
$L_{gromacs}$ $C_{cactusADM}$
$L_{cactusADM}$ $C_{gromacs}$

CRT

- Throughput = 3.55
- Similarity is broken
- Can we do better?

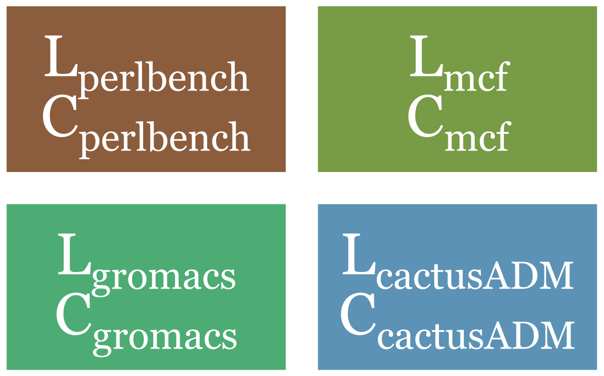


$L_{perlbench}$ C_{mcf} $C_{gromacs}$
L_{mcf} $C_{cactusADM}$
$L_{gromacs}$ $C_{perlbench}$
$L_{cactusADM}$

- Throughput = 3.76

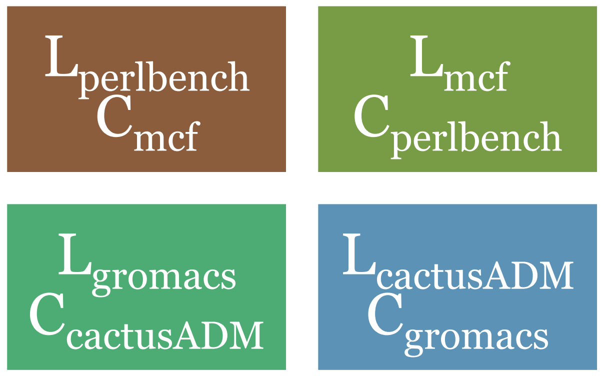
Motivational Example

Without any checking, throughput = 4.84 instructions per cycle


$L_{perlbench}$ $C_{perlbench}$
L_{mcf} C_{mcf}
$L_{gromacs}$ $C_{gromacs}$
$L_{cactusADM}$ $C_{cactusADM}$

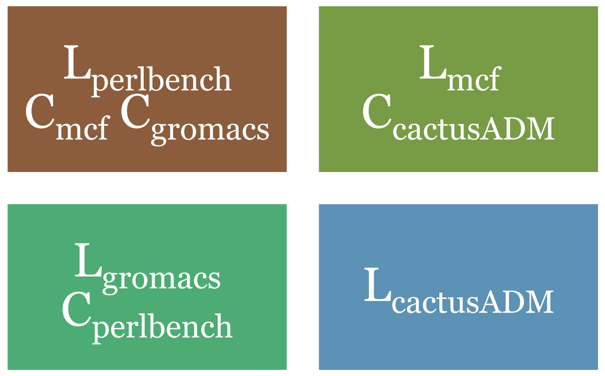
SRT

- Throughput = 3.24
- Similarity in resource requirement
- High throughput threads together


$L_{perlbench}$ C_{mcf}
L_{mcf} $C_{perlbench}$
$L_{gromacs}$ $C_{cactusADM}$
$L_{cactusADM}$ $C_{gromacs}$

CRT

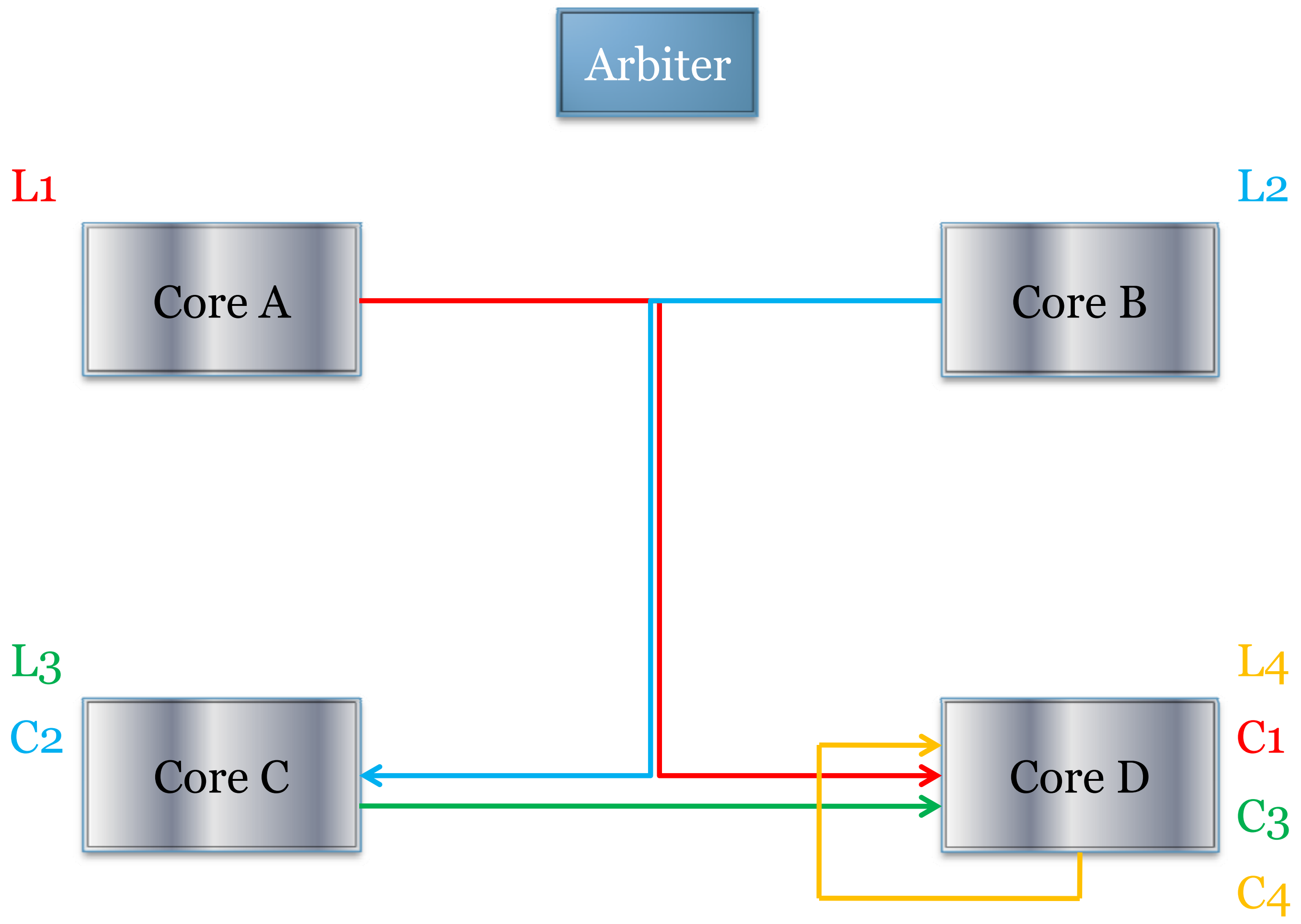
- Throughput = 3.55
- Similarity is broken
- Can we do better?


$L_{perlbench}$ C_{mcf} $C_{gromacs}$
L_{mcf} $C_{cactusADM}$
$L_{gromacs}$ $C_{perlbench}$
$L_{cactusADM}$

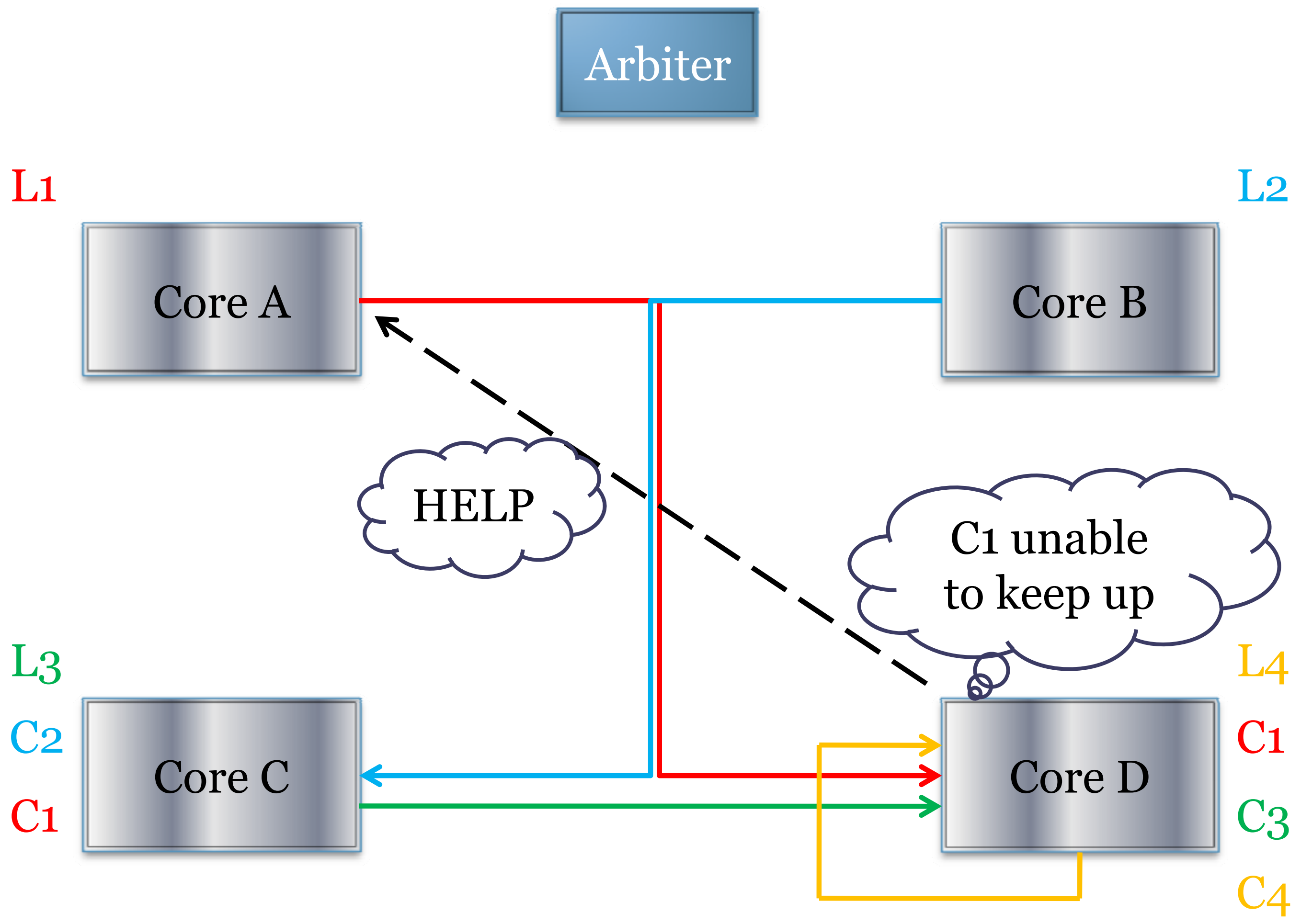
FluidCheck

- Throughput = 3.76
- Schedules based on the applications' behavior
- FluidCheck is a superset of schedules; SRT, CRT are instances within FluidCheck

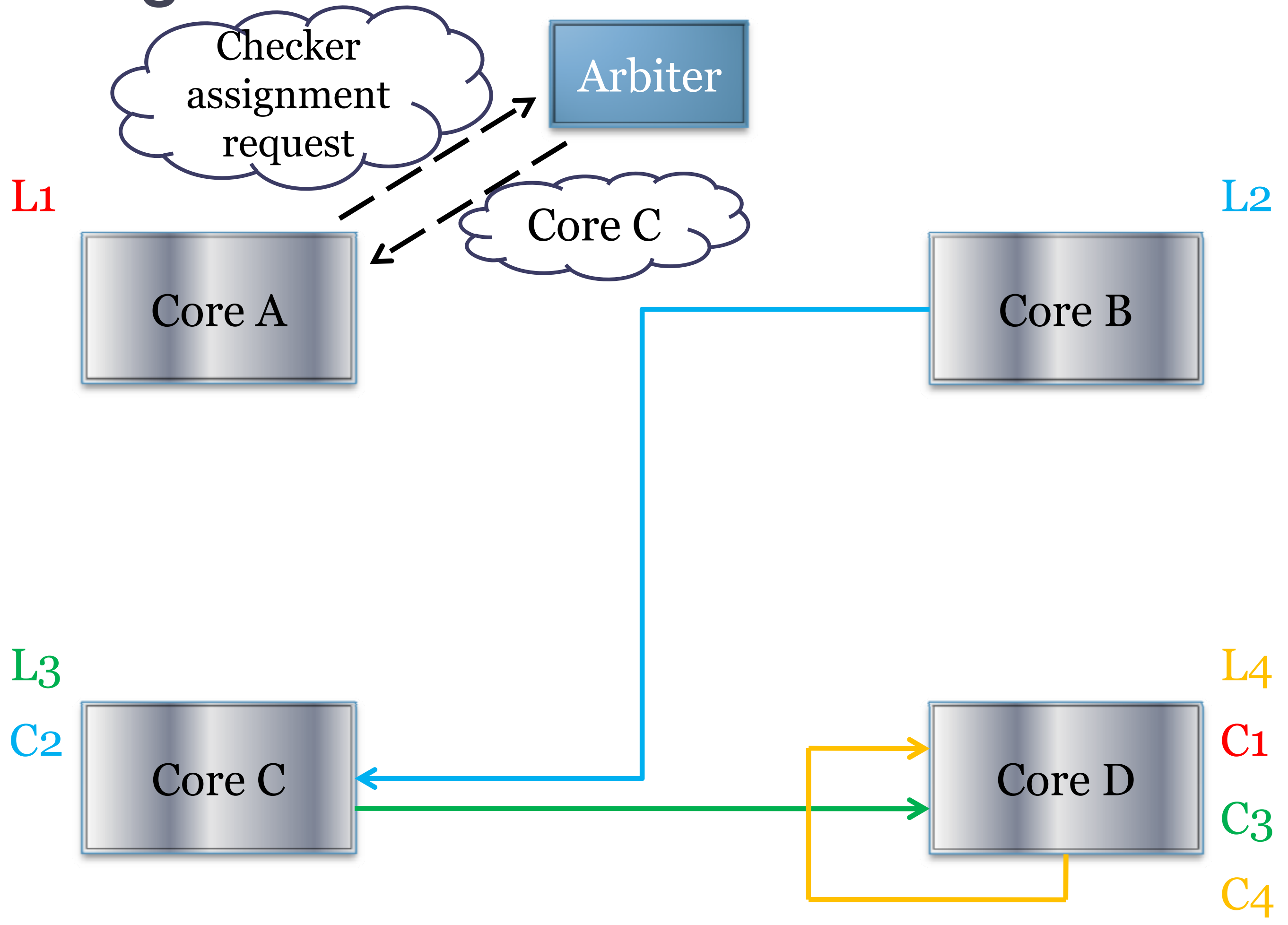
Simplified Illustration of FluidCheck’s Working



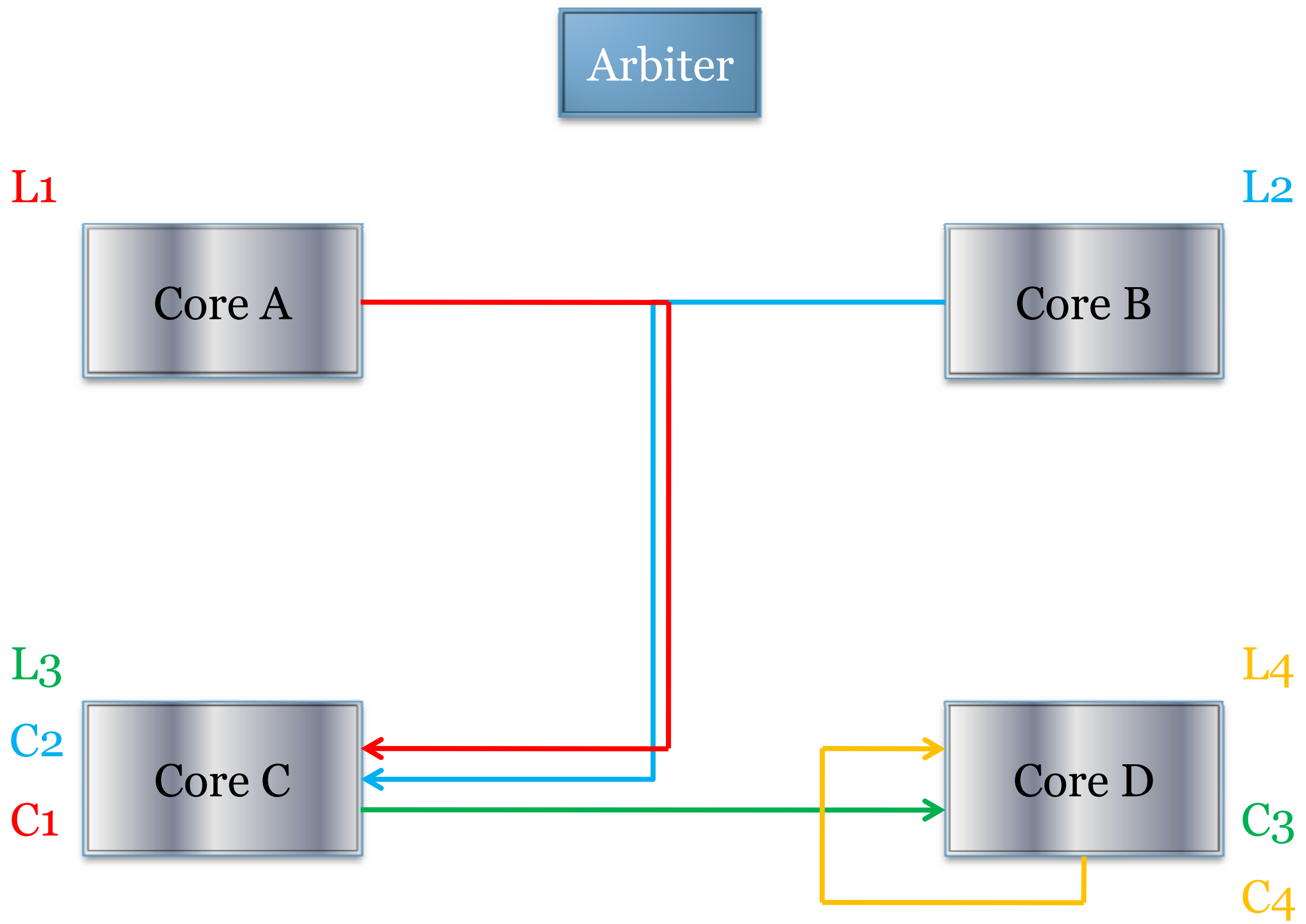
Simplified Illustration of FluidCheck's Working



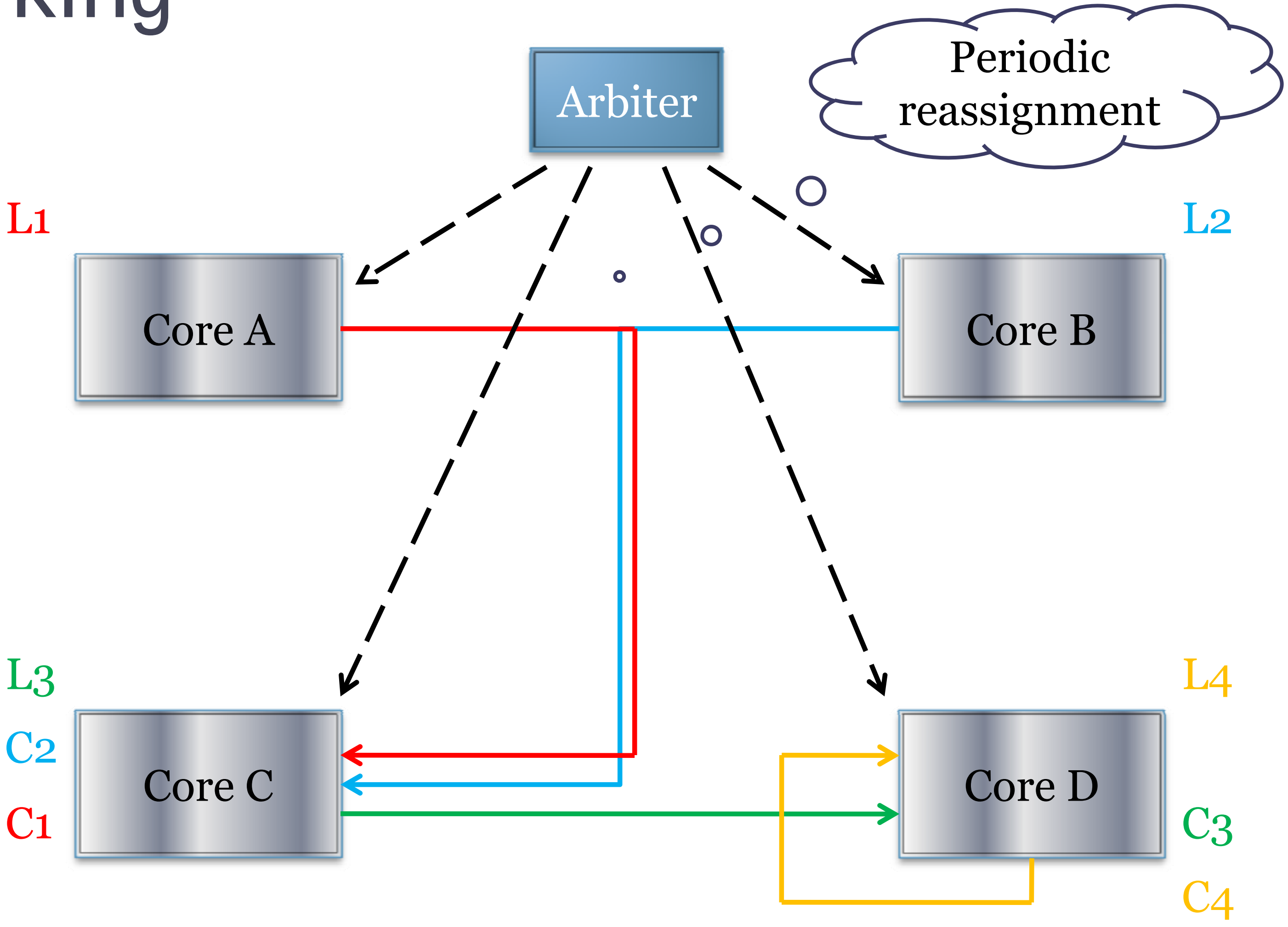
Simplified Illustration of FluidCheck's Working



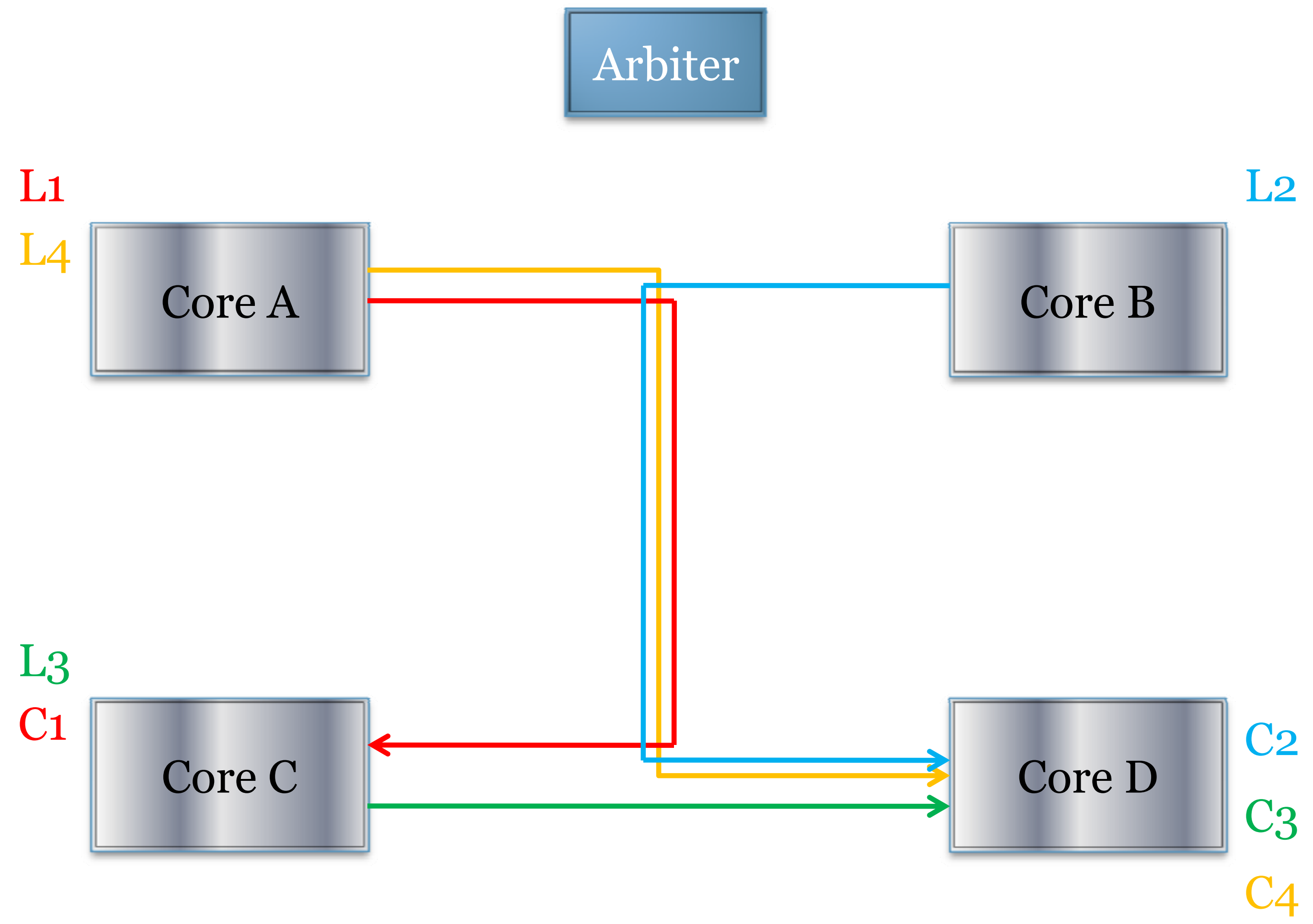
Simplified Illustration of FluidCheck’s Working



Simplified Illustration of FluidCheck's Working



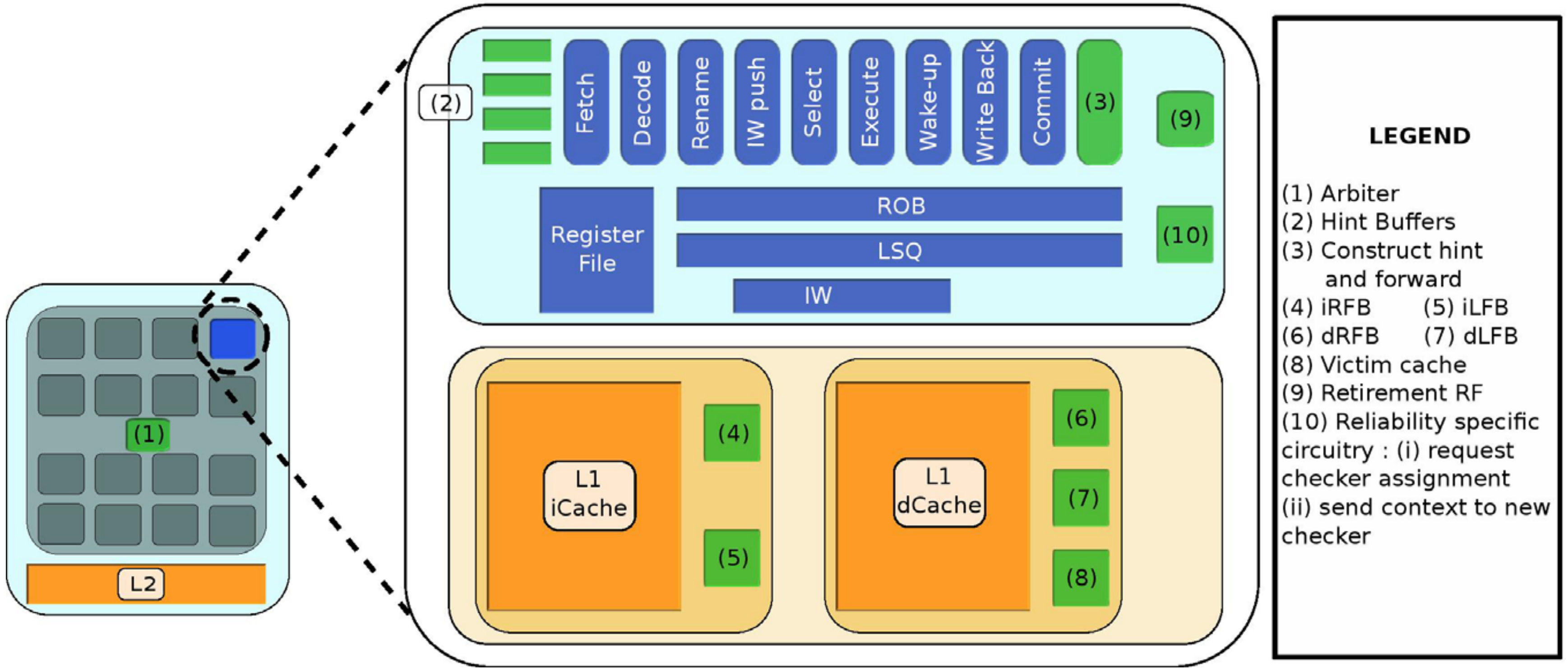
Simplified Illustration of FluidCheck's Working



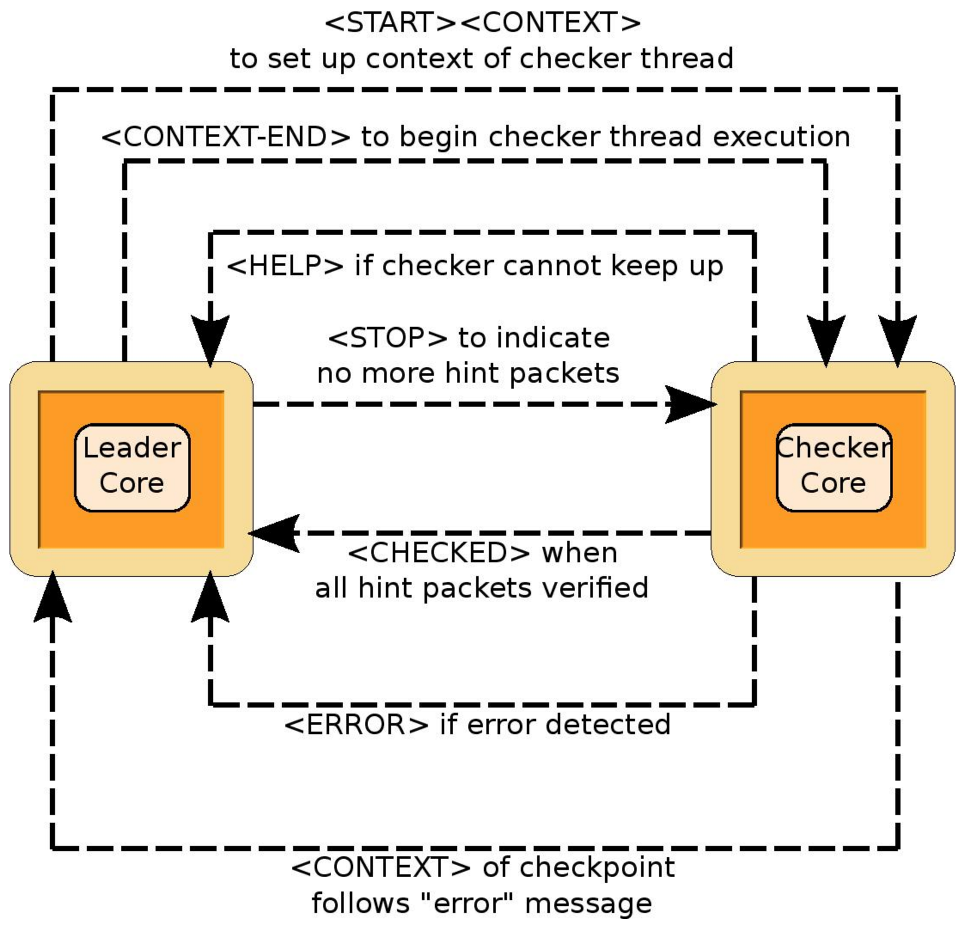
Challenges to achieving FluidCheck

- Reactive phase-based scheduler
- Efficient transfer of hints
- Efficient forwarding of cache lines from the leader to the checker
- Circumventing subtle livelock scenarios

Hardware Architecture

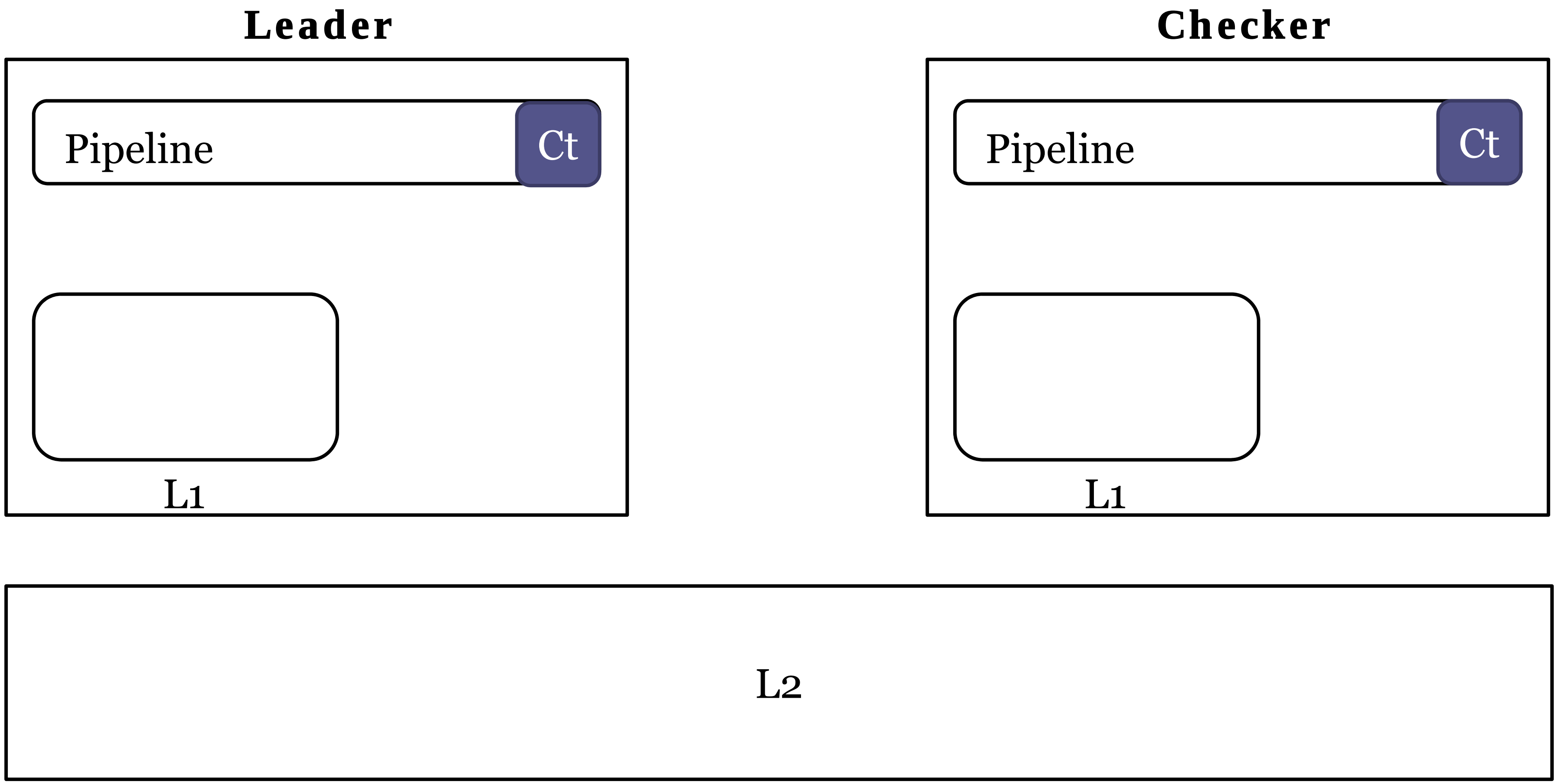


Overview of Redundant Execution

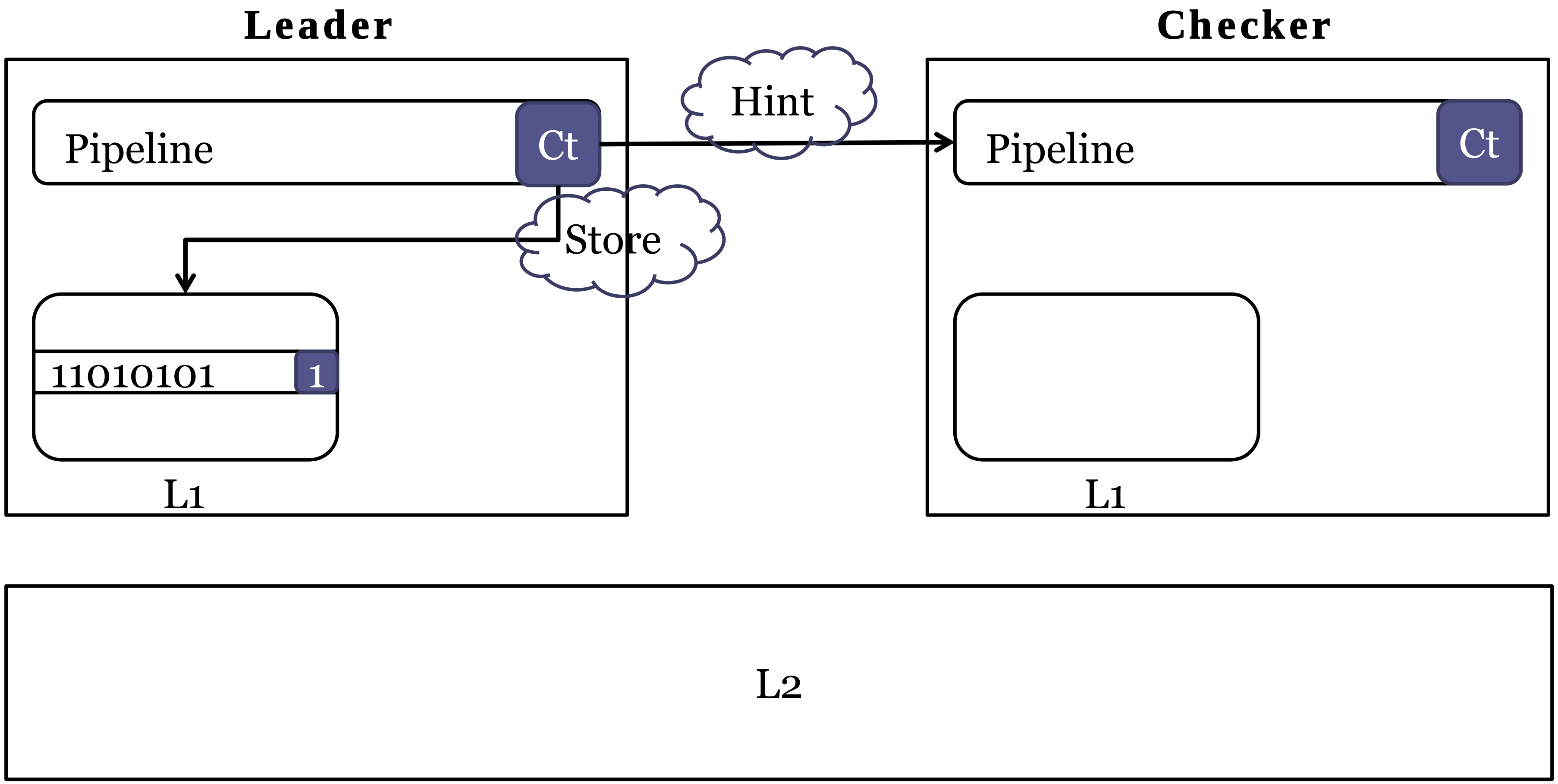


<START><CONTEXT>
to set up context of checker thread
<CONTEXT-END> to begin checker thread execution
<HELP> if checker cannot keep up
<STOP> to indicate
no more hint packets
Leader
Core
Checker
Core
<CHECKED> when
all hint packets verified
<ERROR> if error detected
<CONTEXT> of checkpoint
follows "error" message

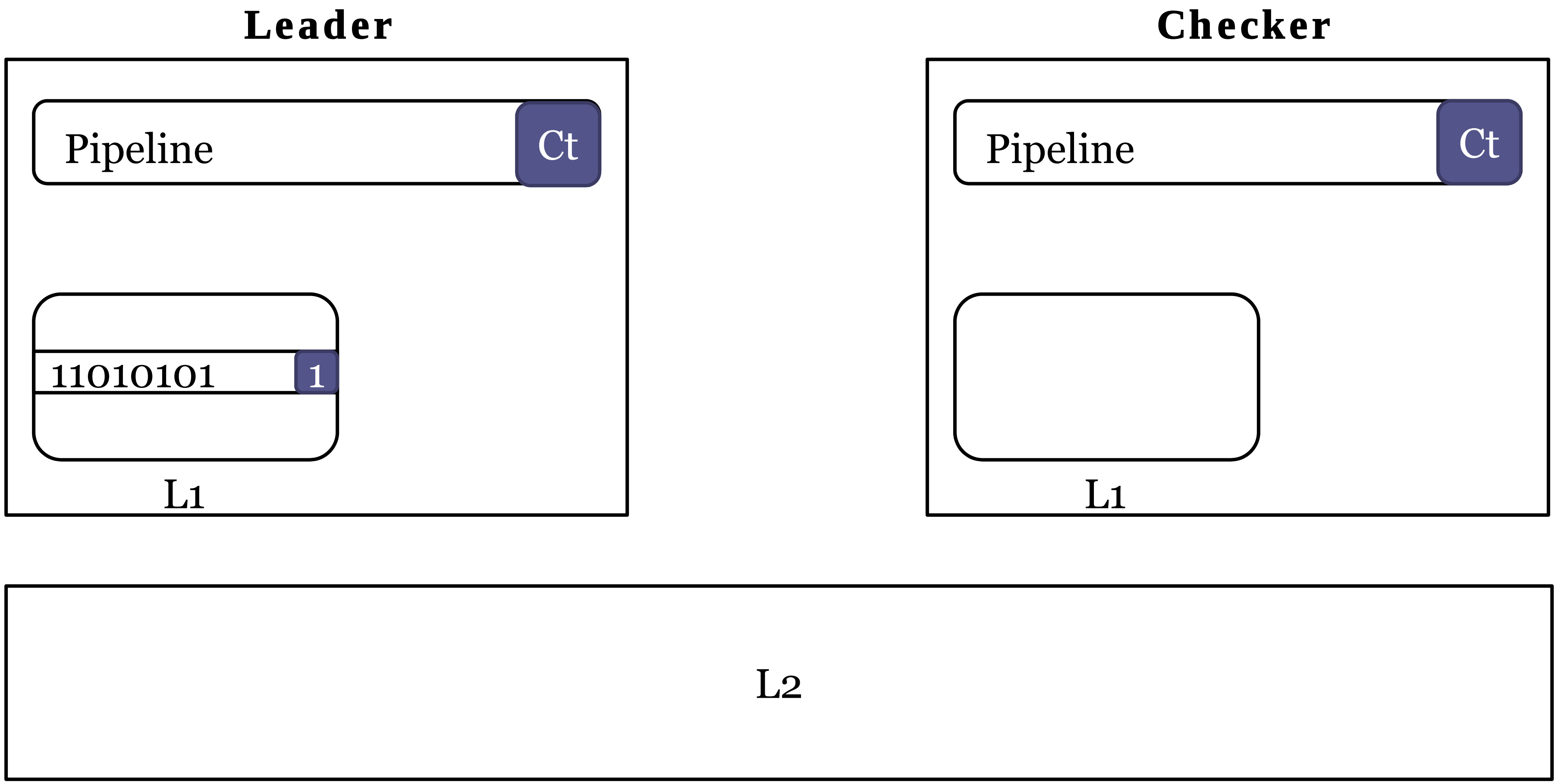
Memory Checkpointing



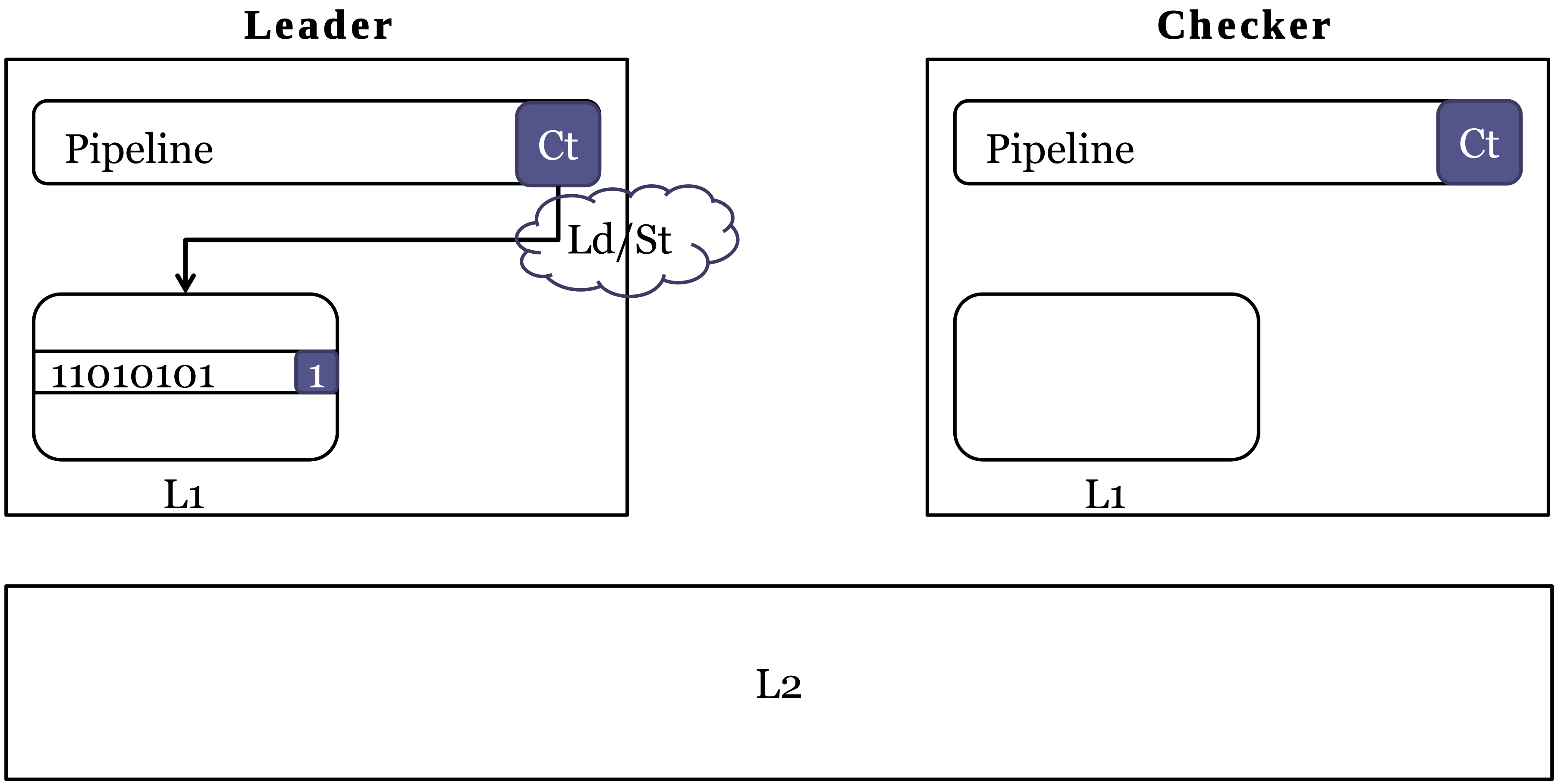
Memory Checkpointing



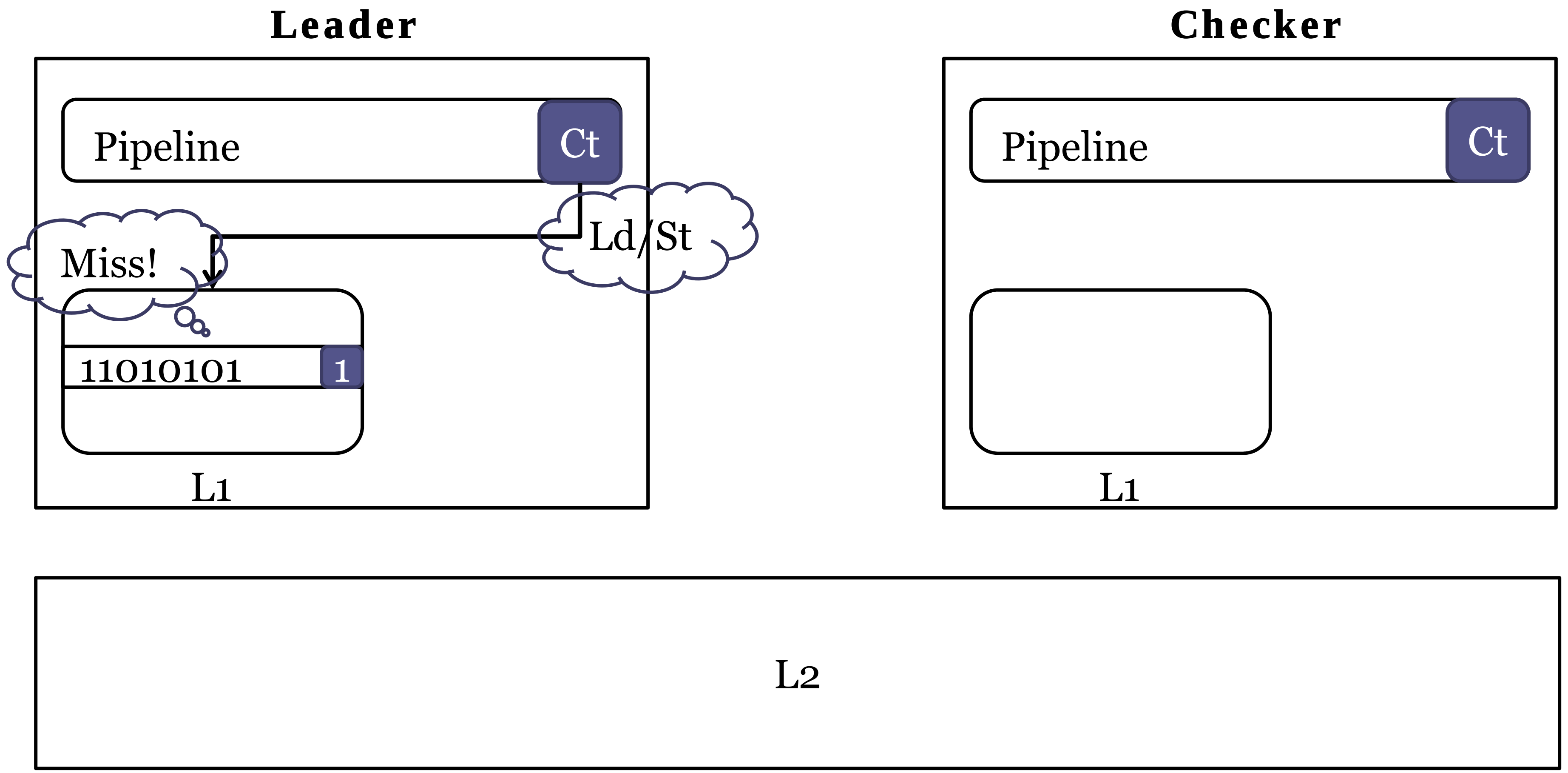
Memory Checkpointing



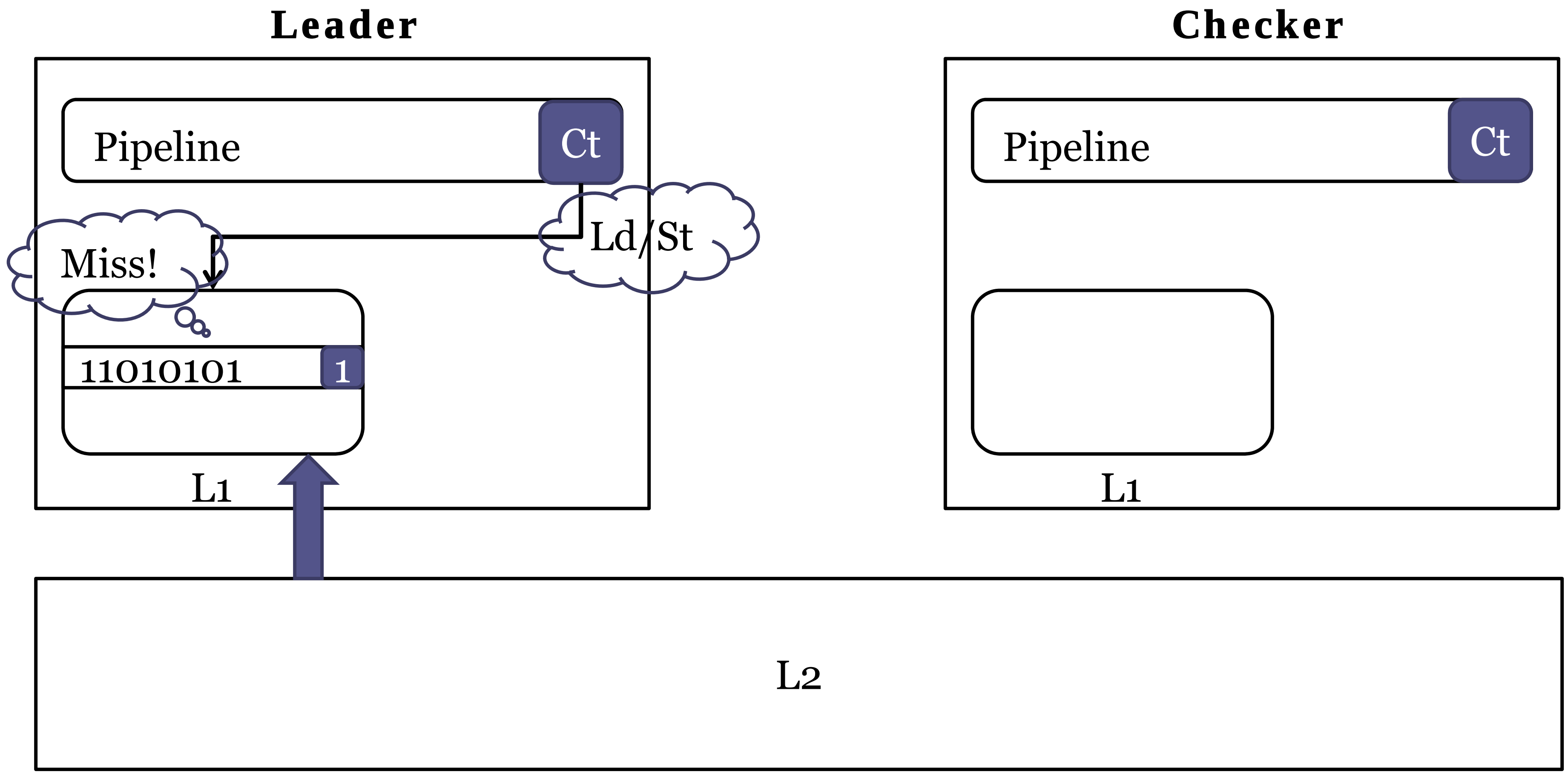
Memory Checkpointing



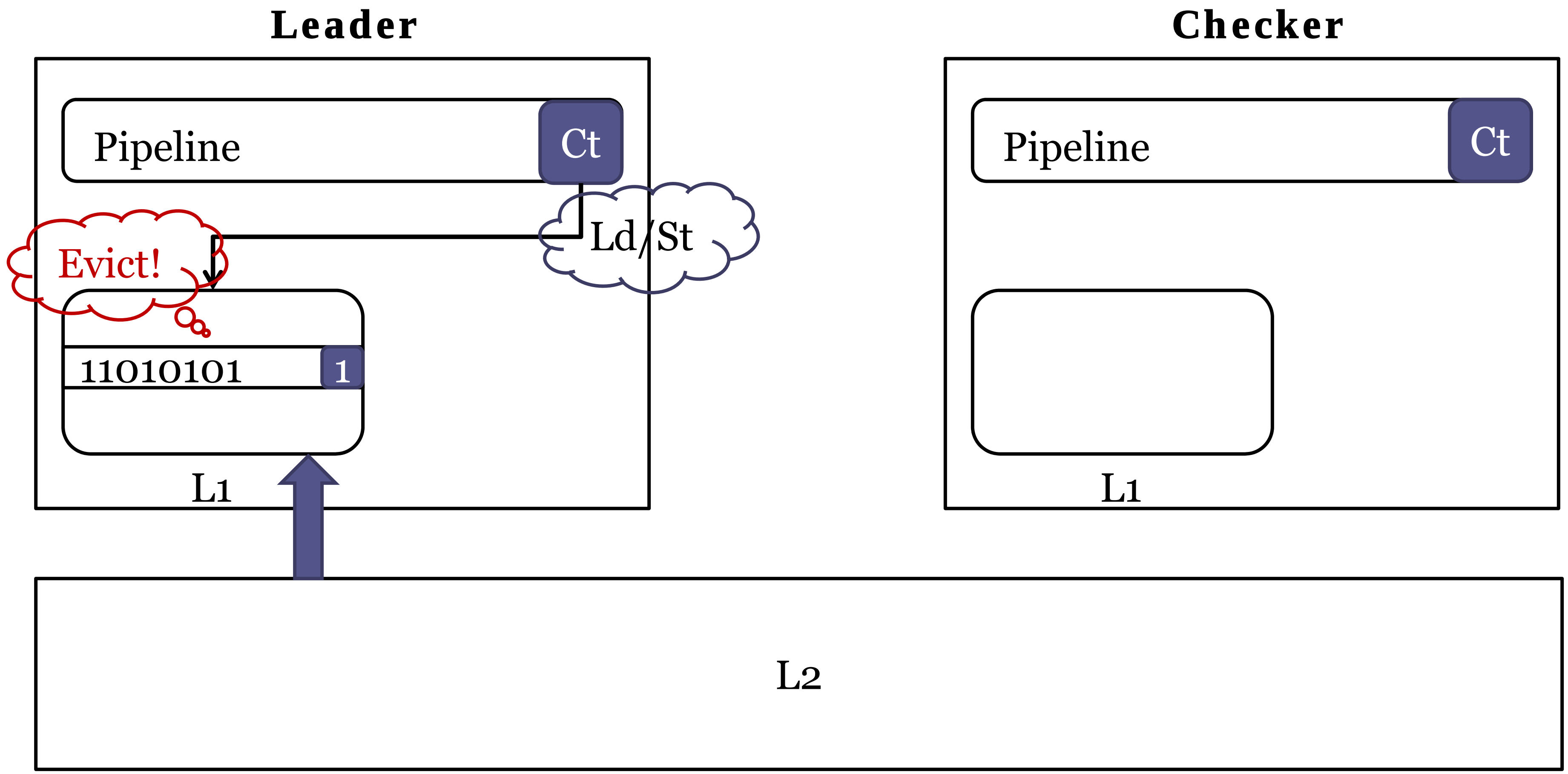
Memory Checkpointing



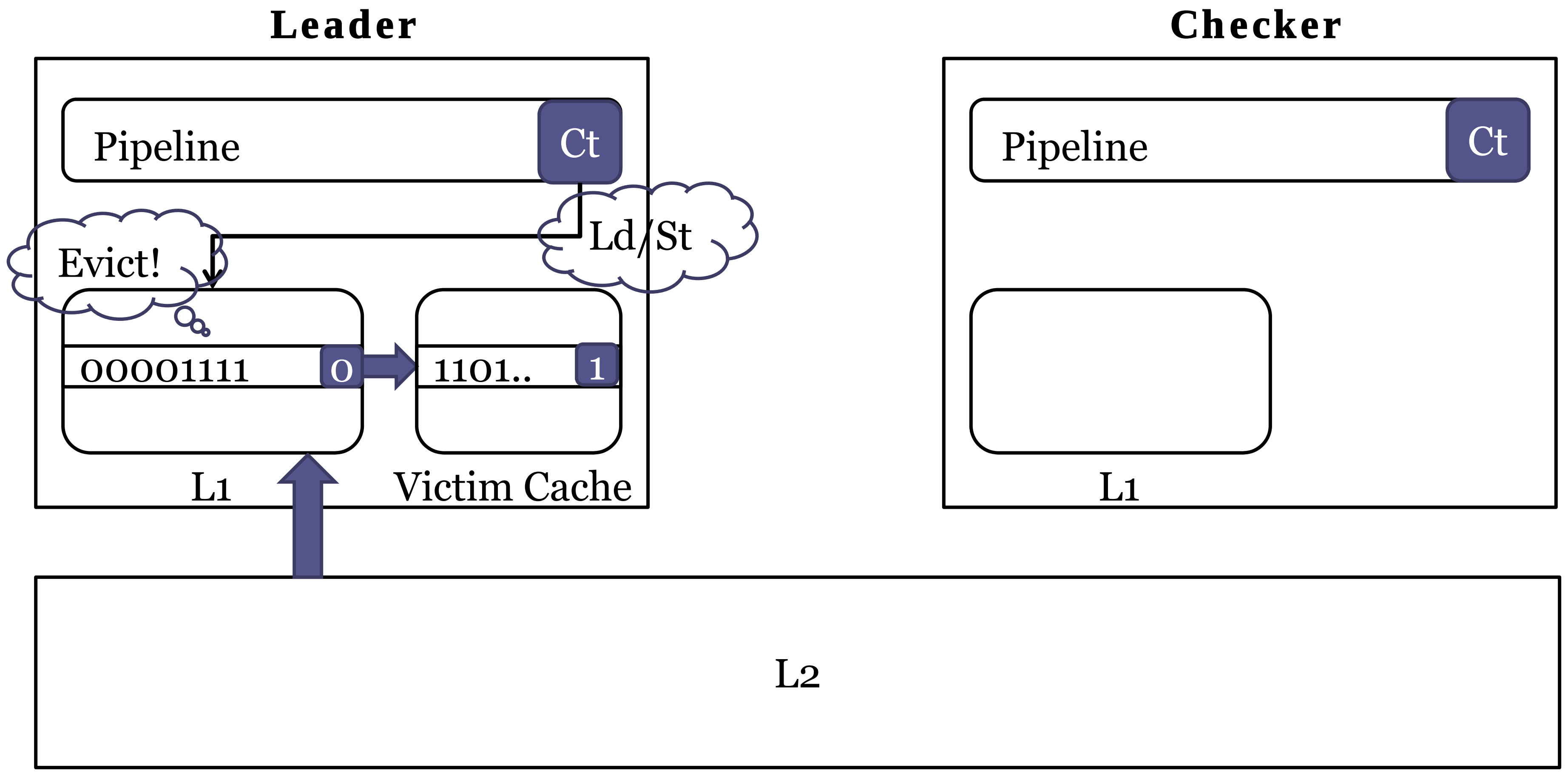
Memory Checkpointing



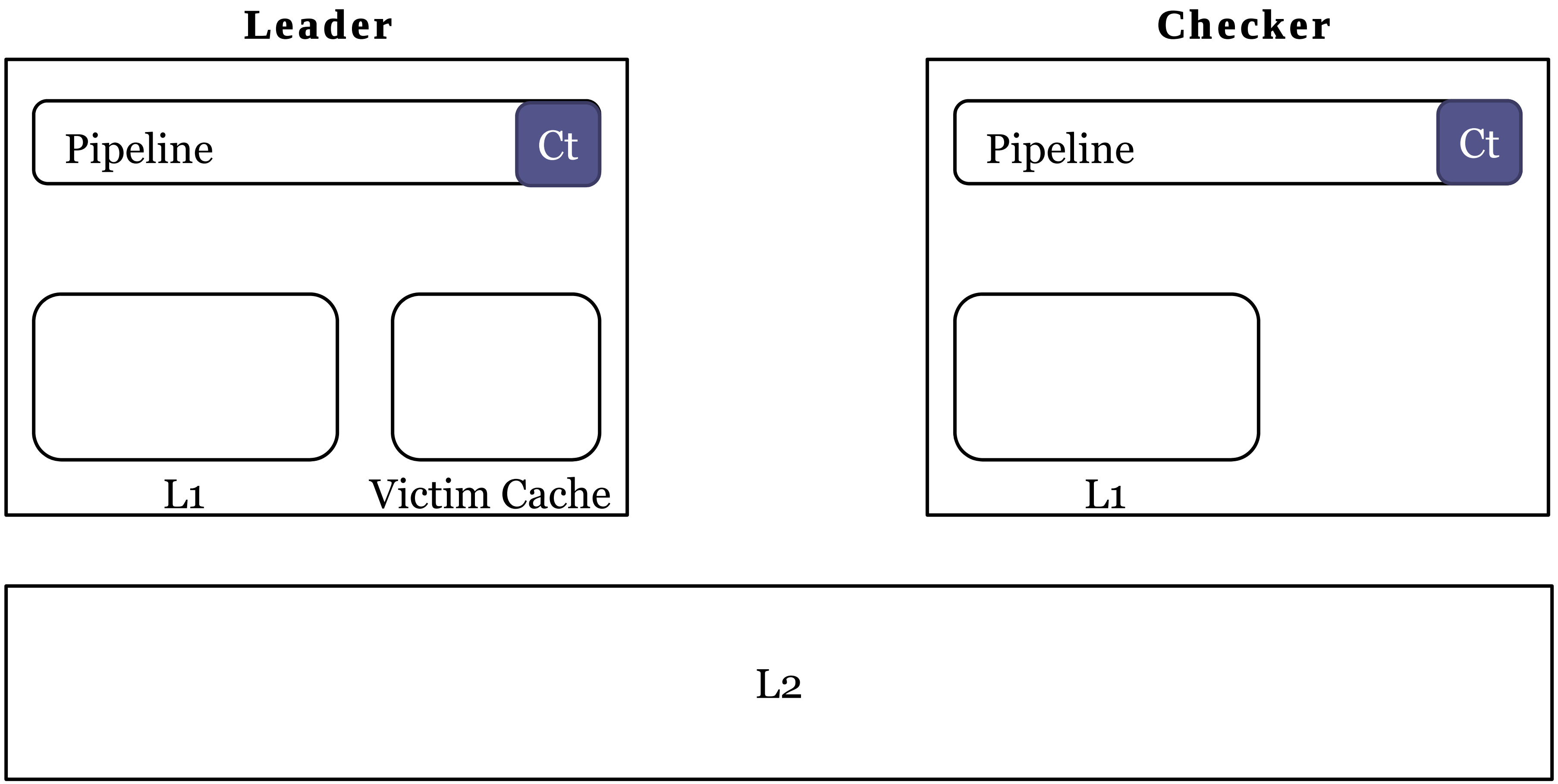
Memory Checkpointing



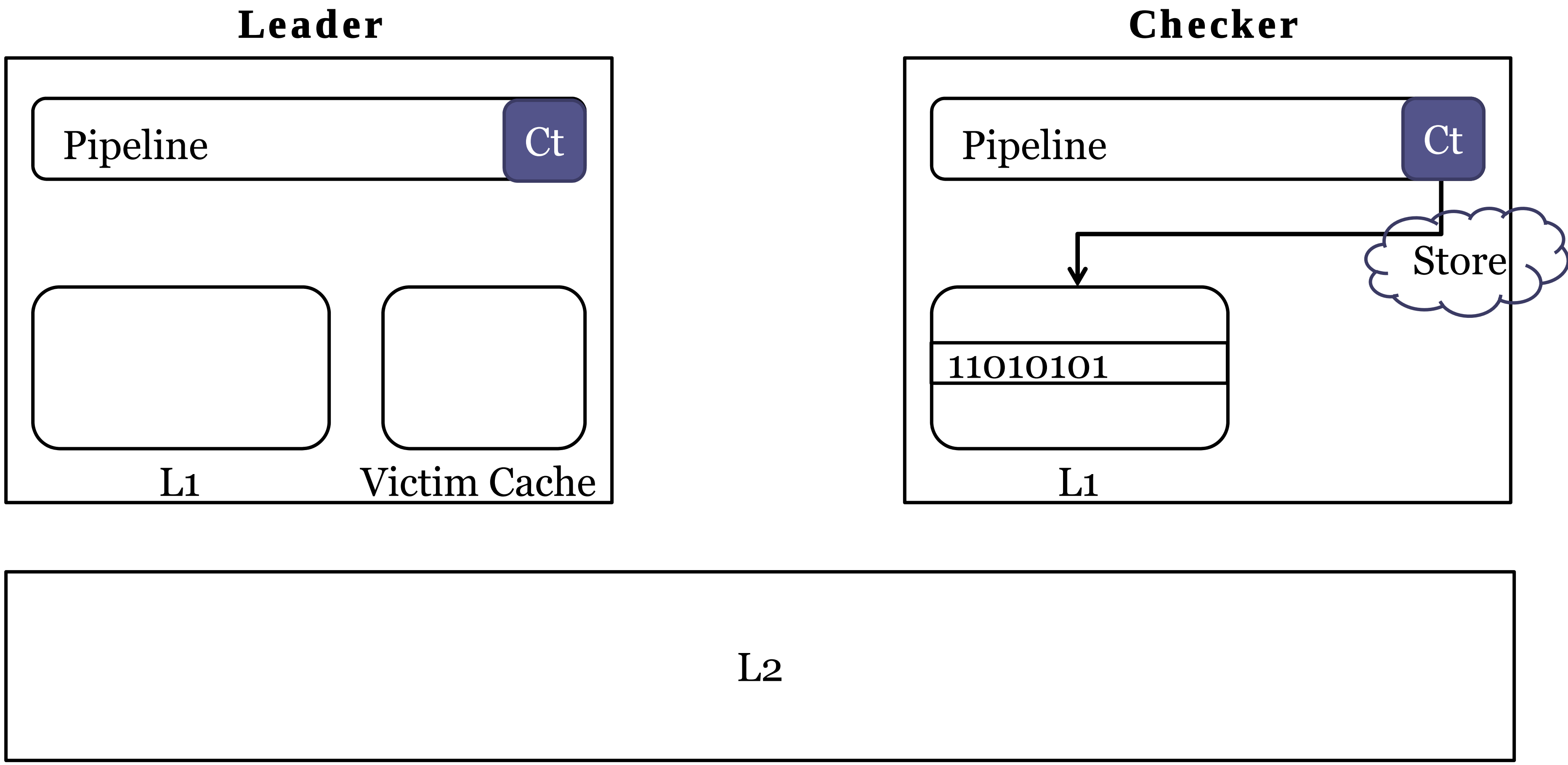
Memory Checkpointing



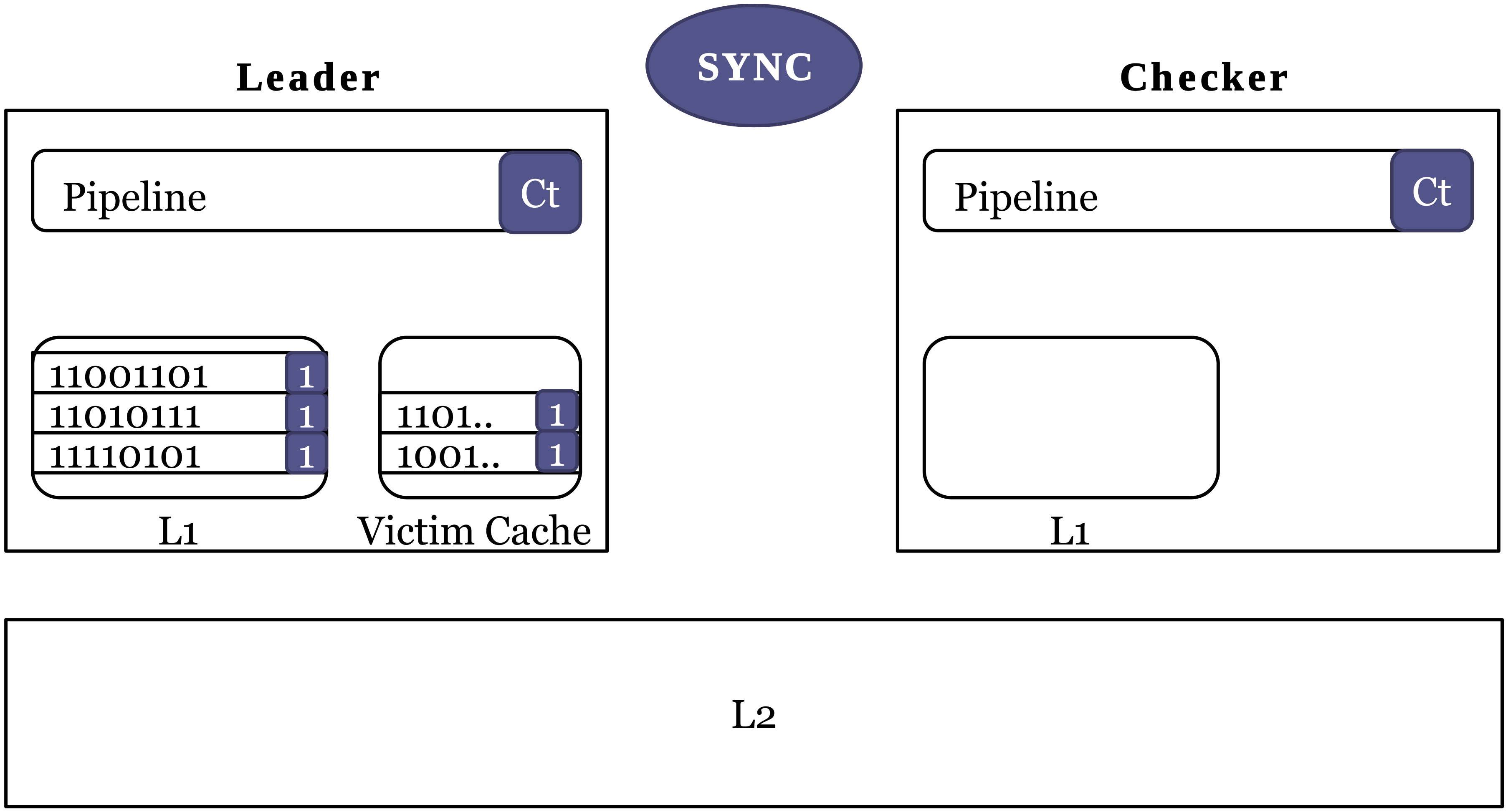
Memory Checkpointing



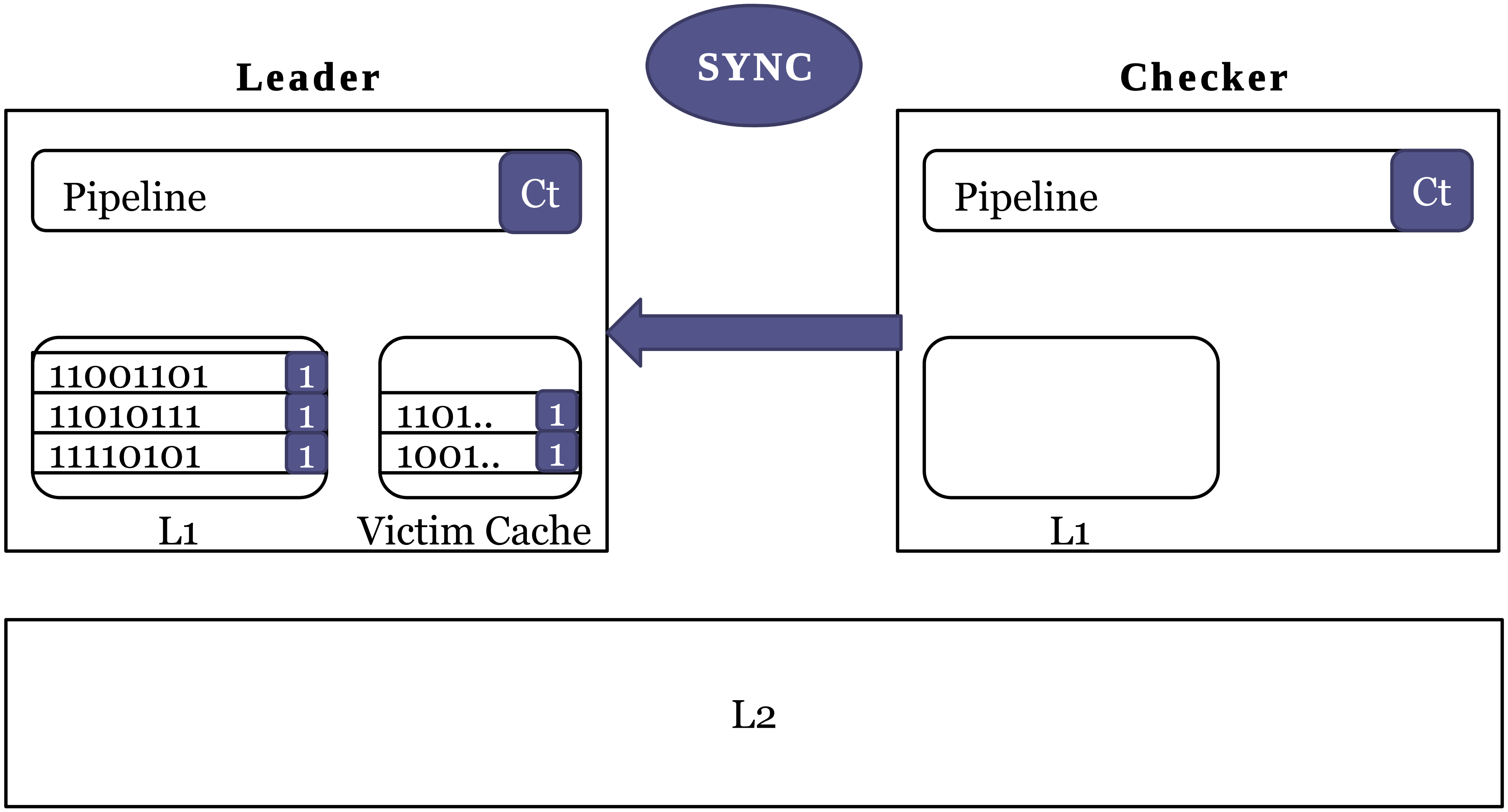
Memory Checkpointing



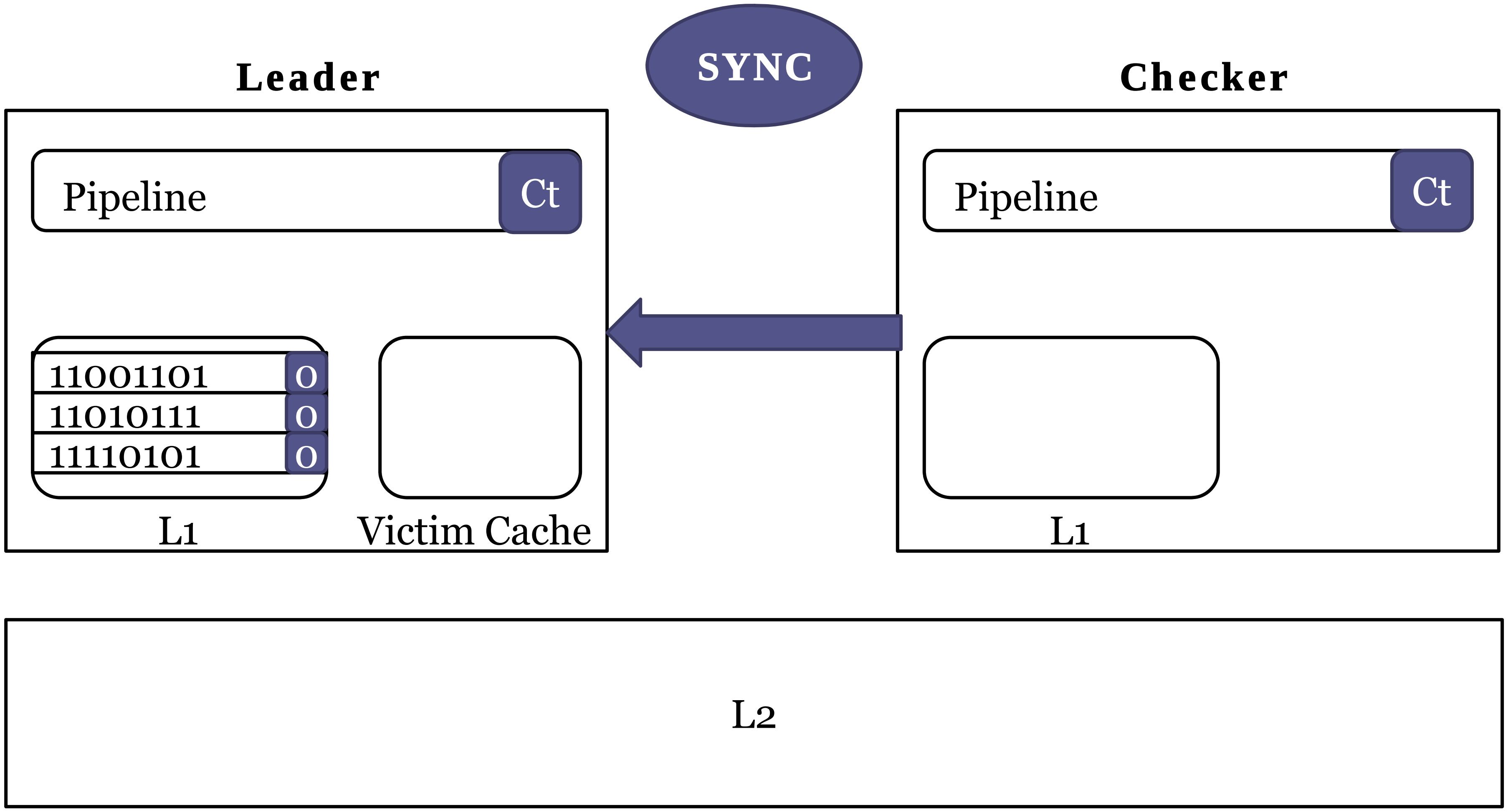
Memory Checkpointing



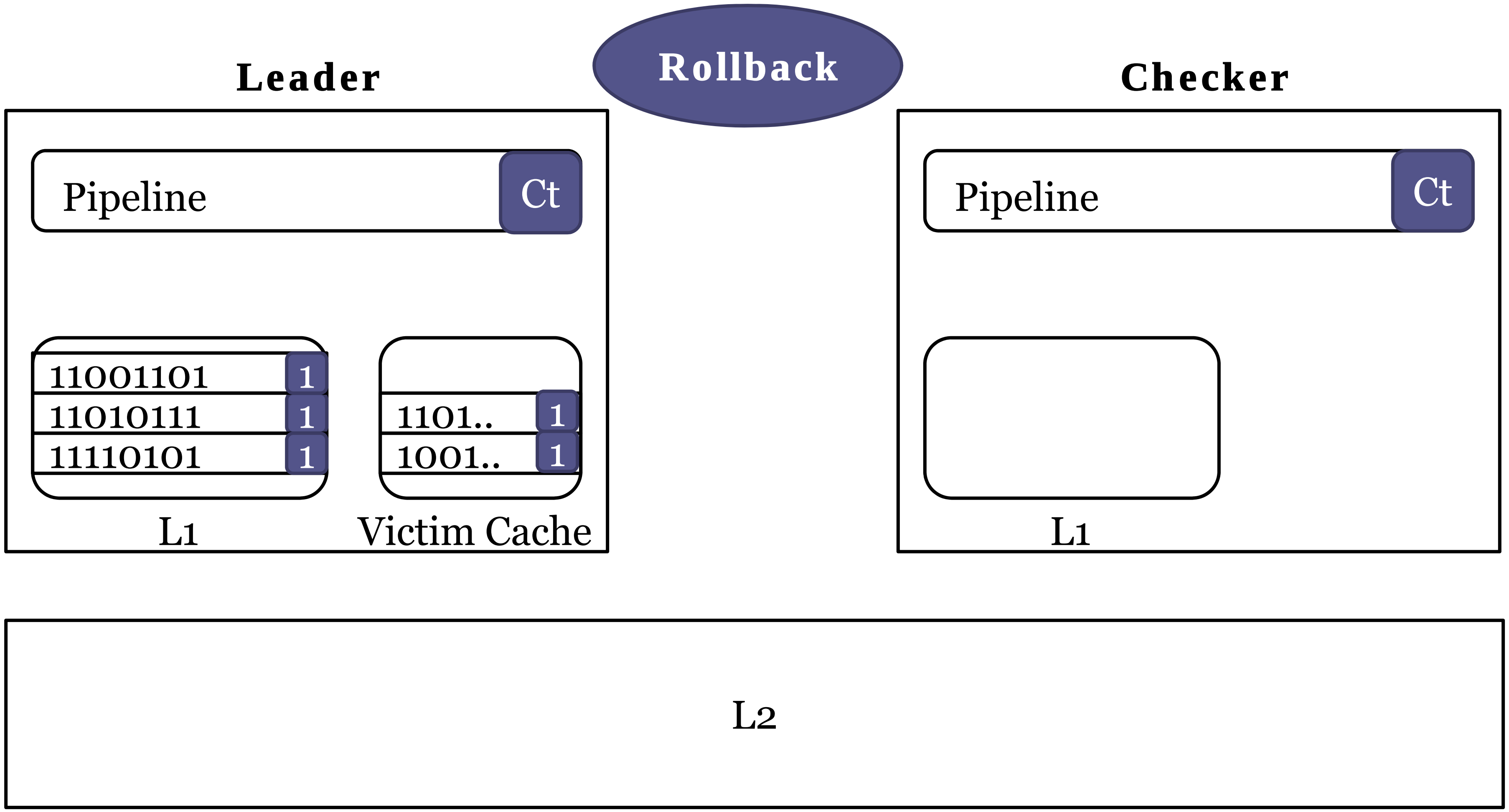
Memory Checkpointing



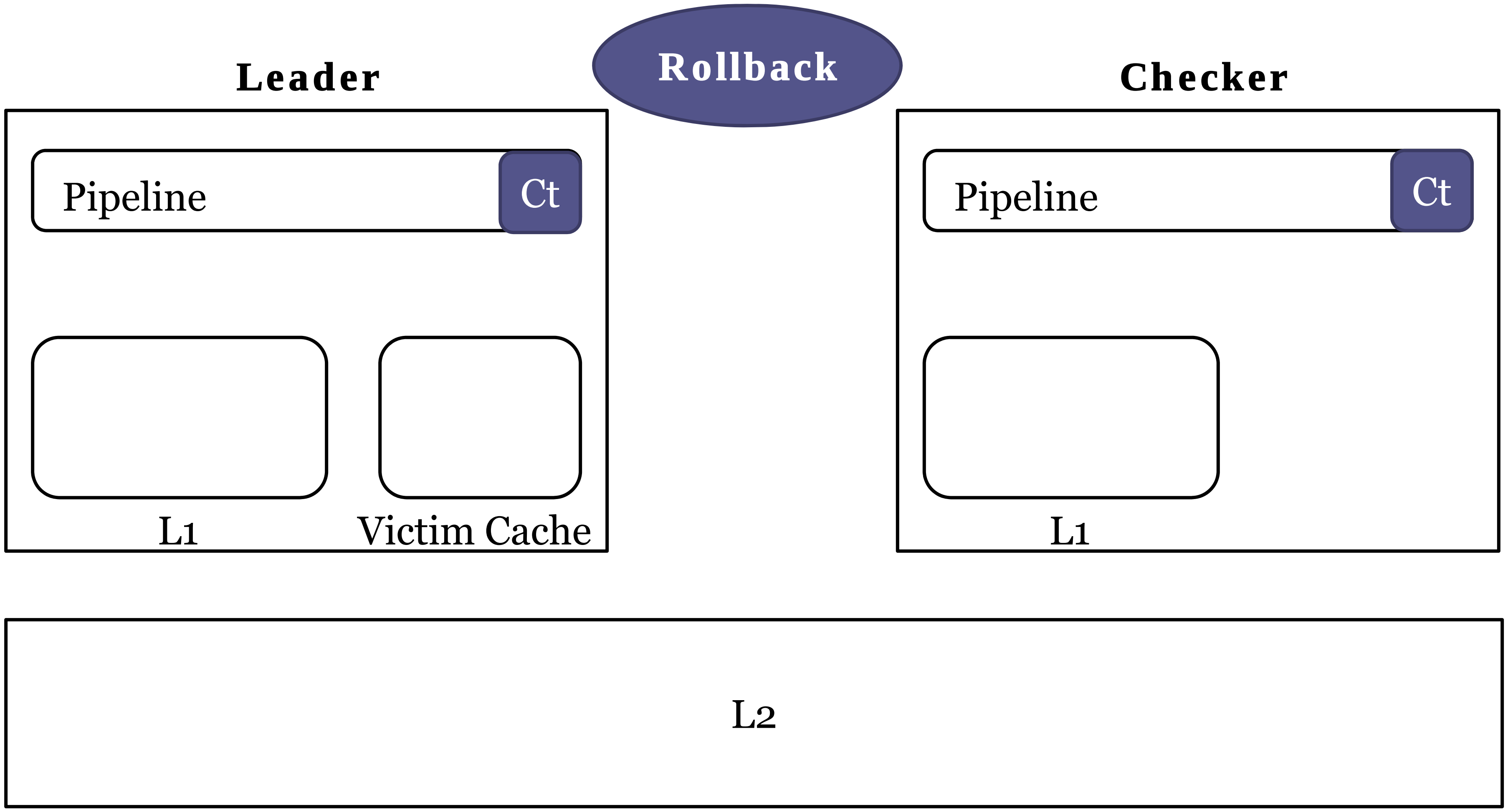
Memory Checkpointing



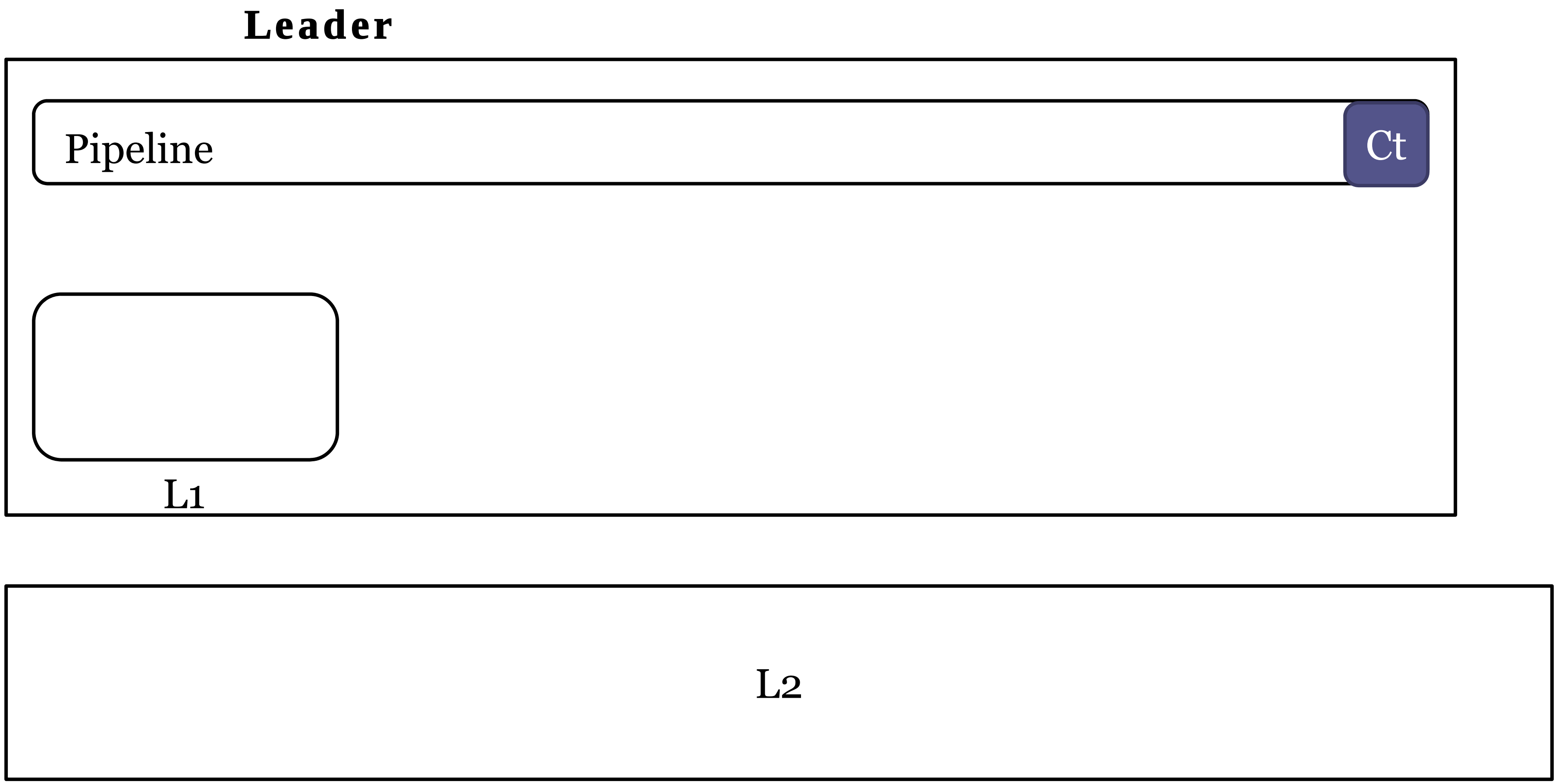
Memory Checkpointing



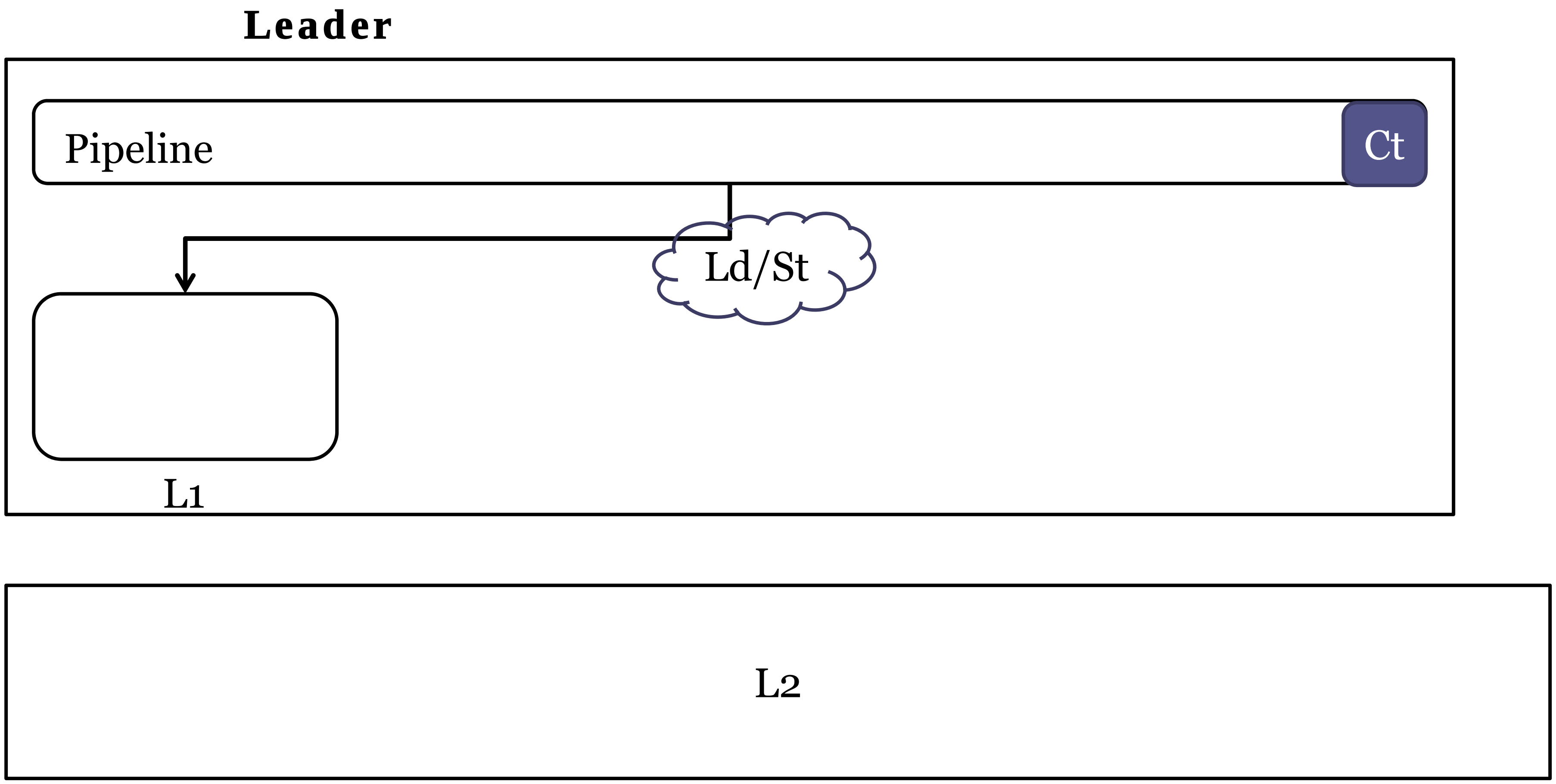
Memory Checkpointing



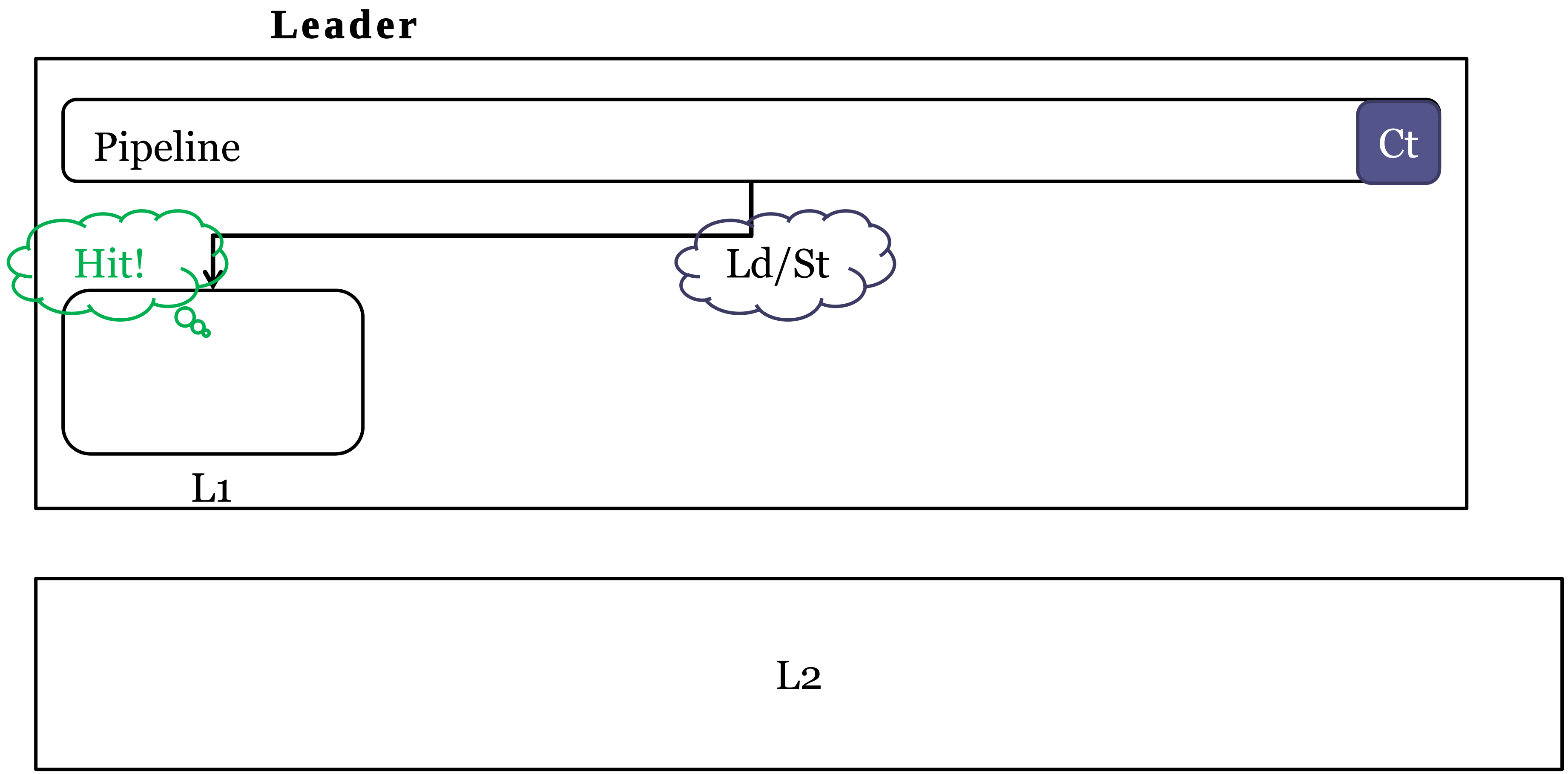
Forwarding Filters



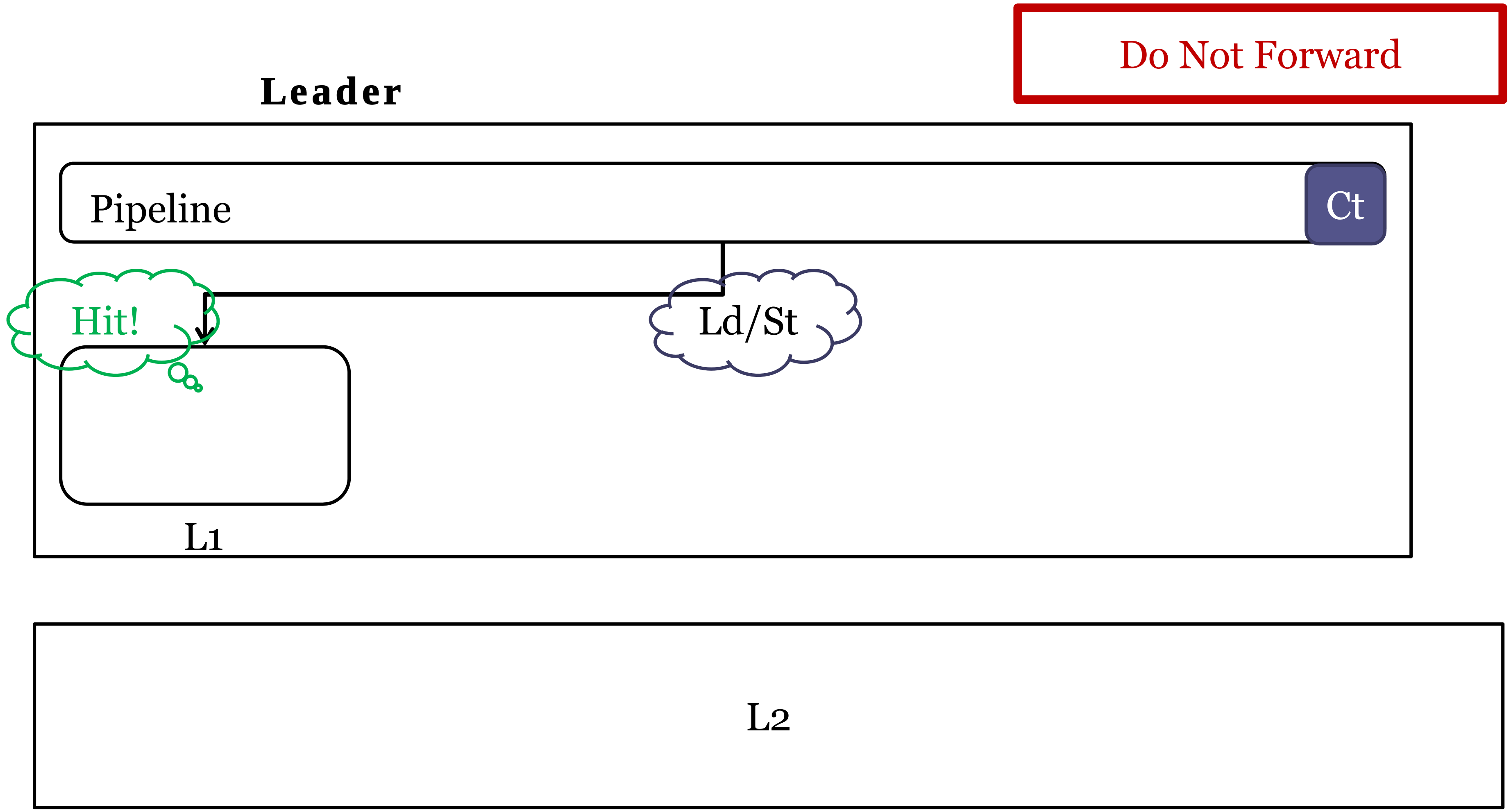
Forwarding Filters



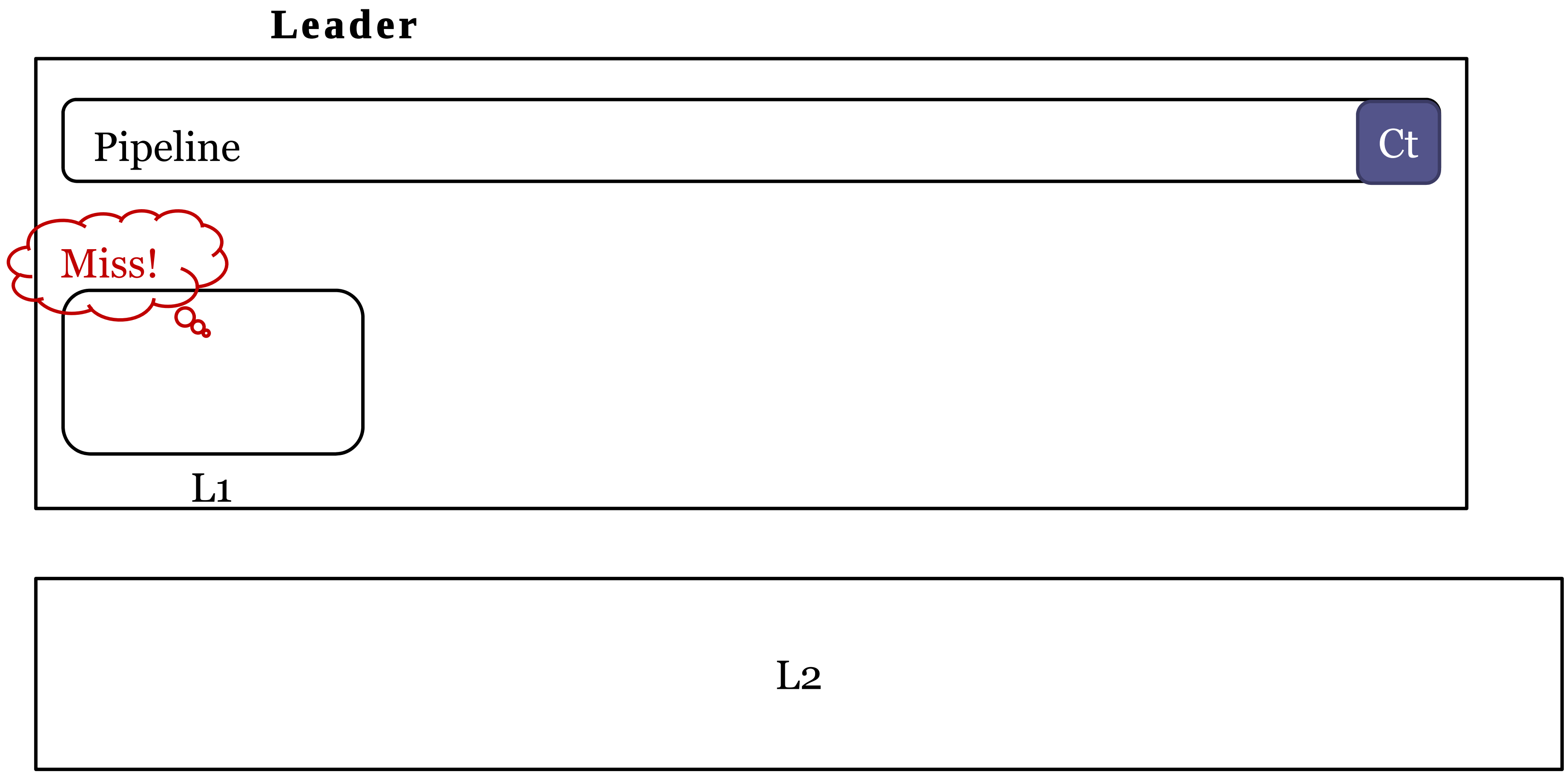
Forwarding Filters



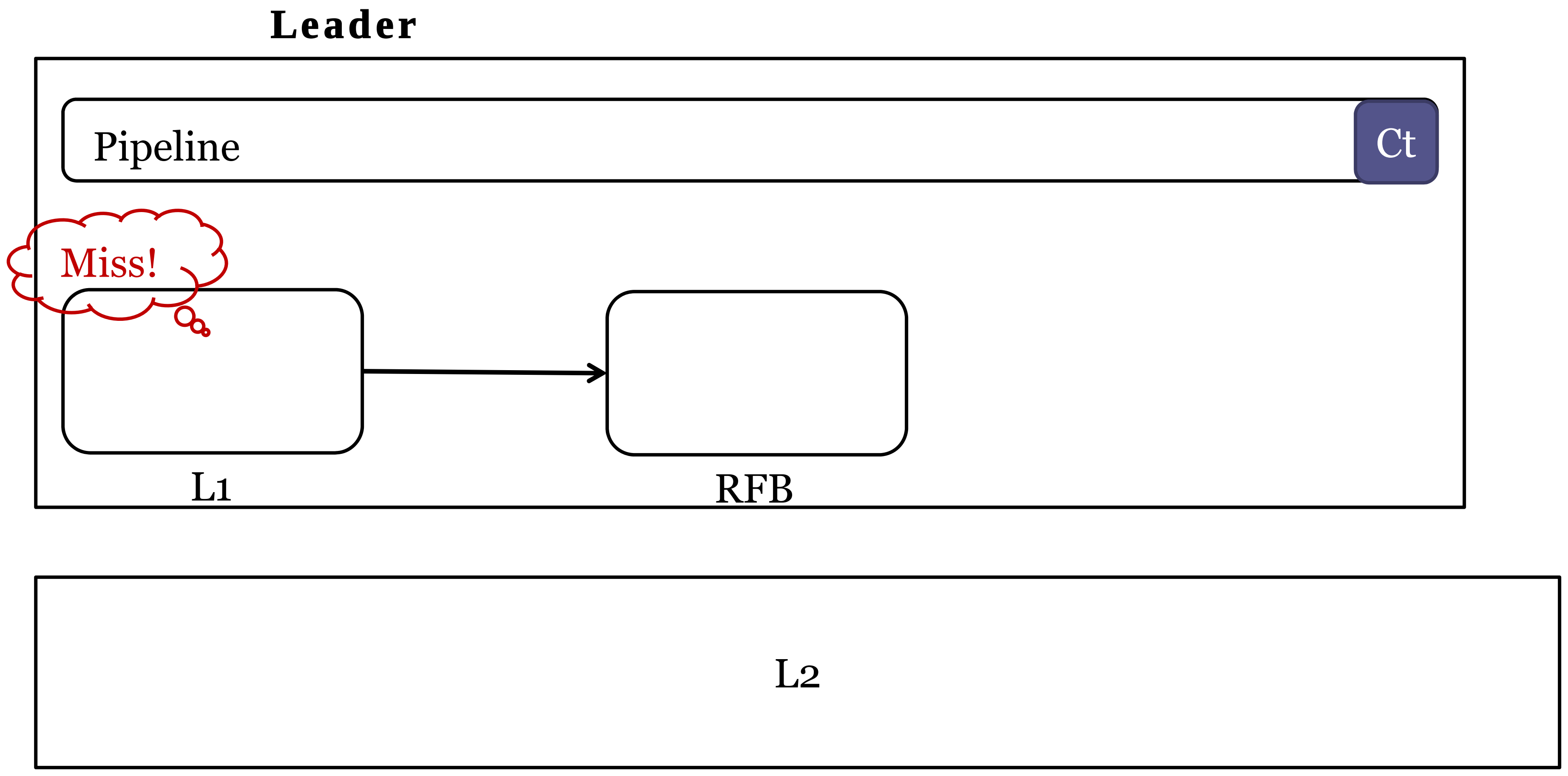
Forwarding Filters



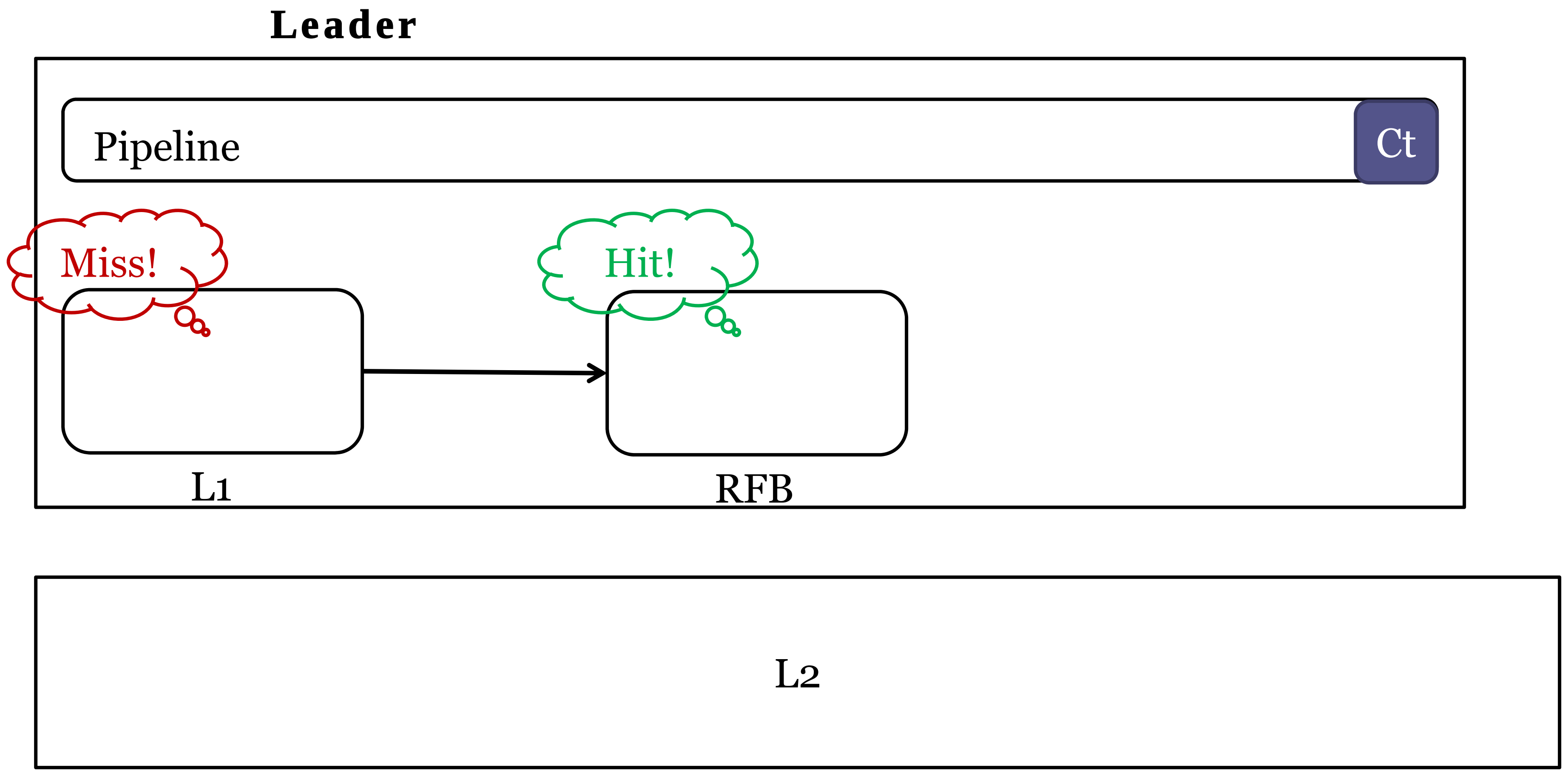
Forwarding Filters



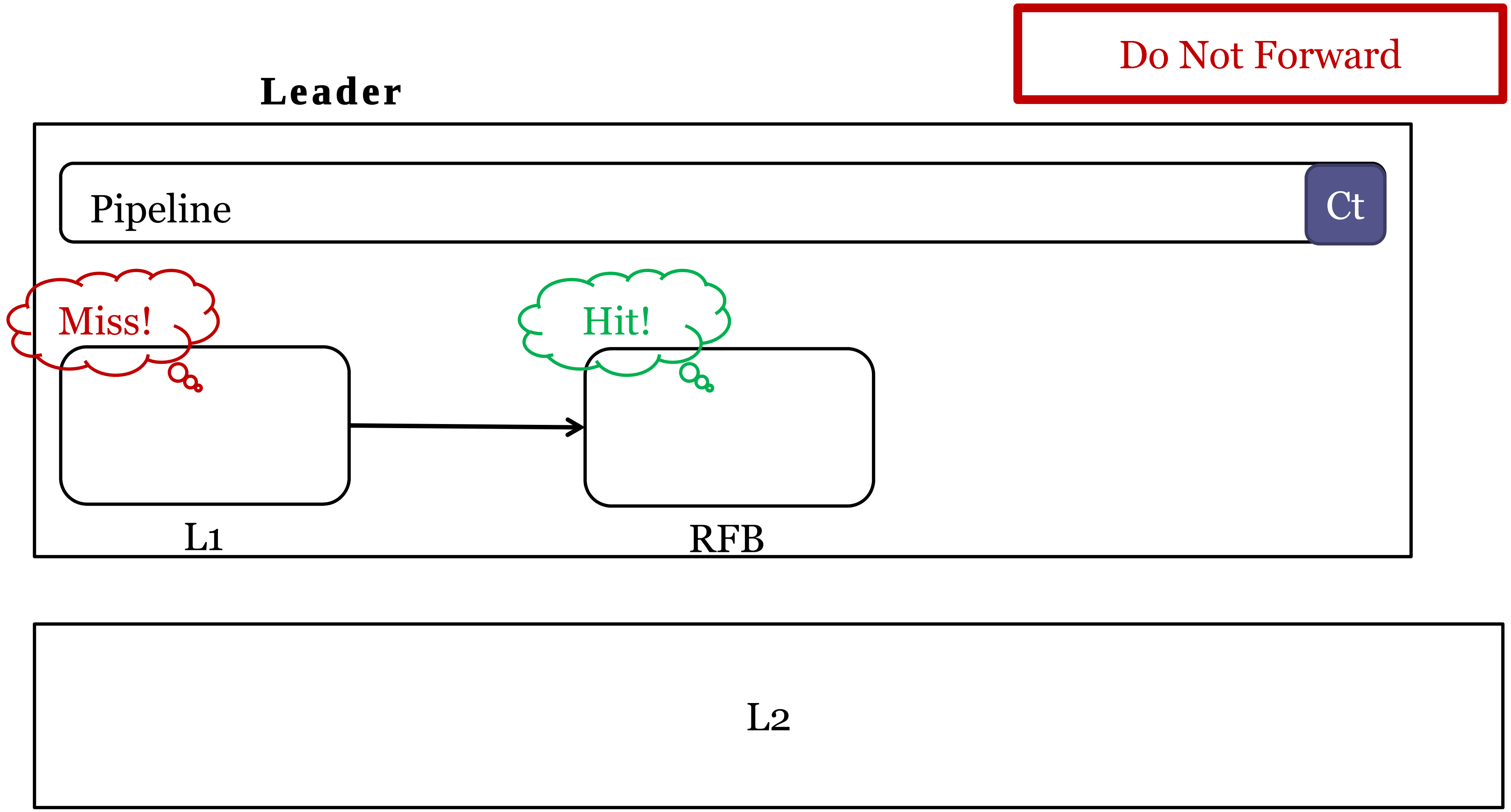
Forwarding Filters



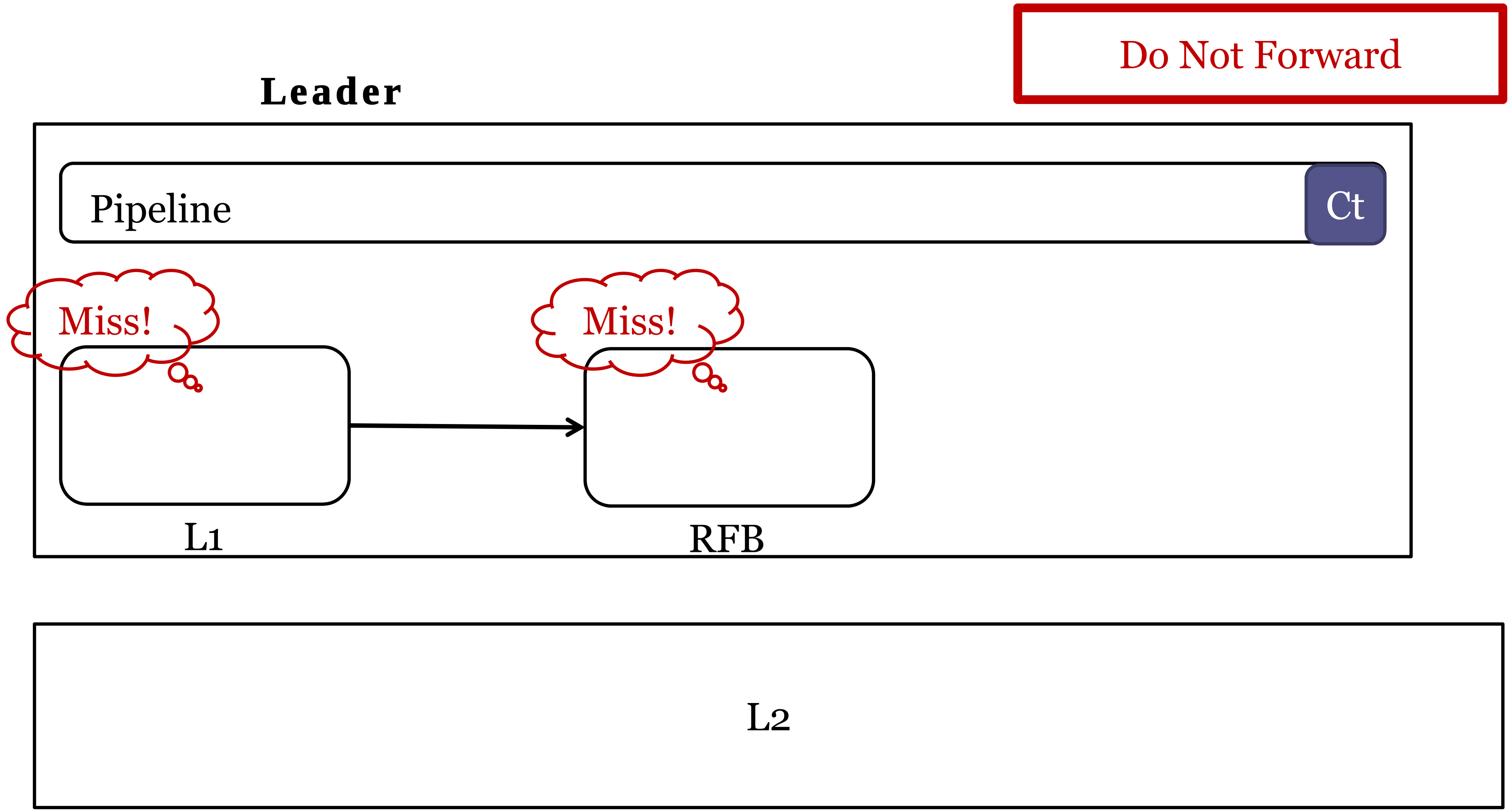
Forwarding Filters



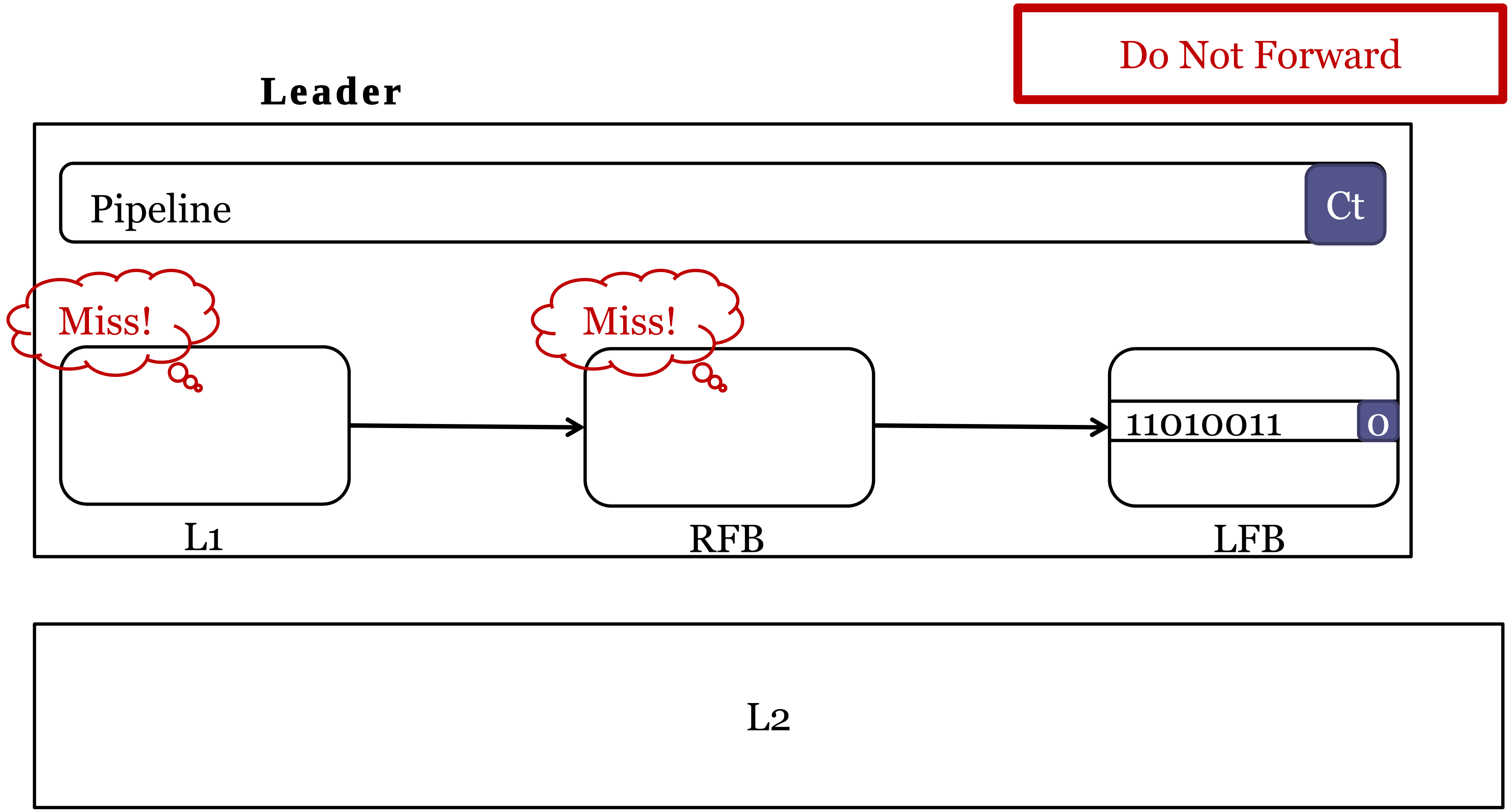
Forwarding Filters



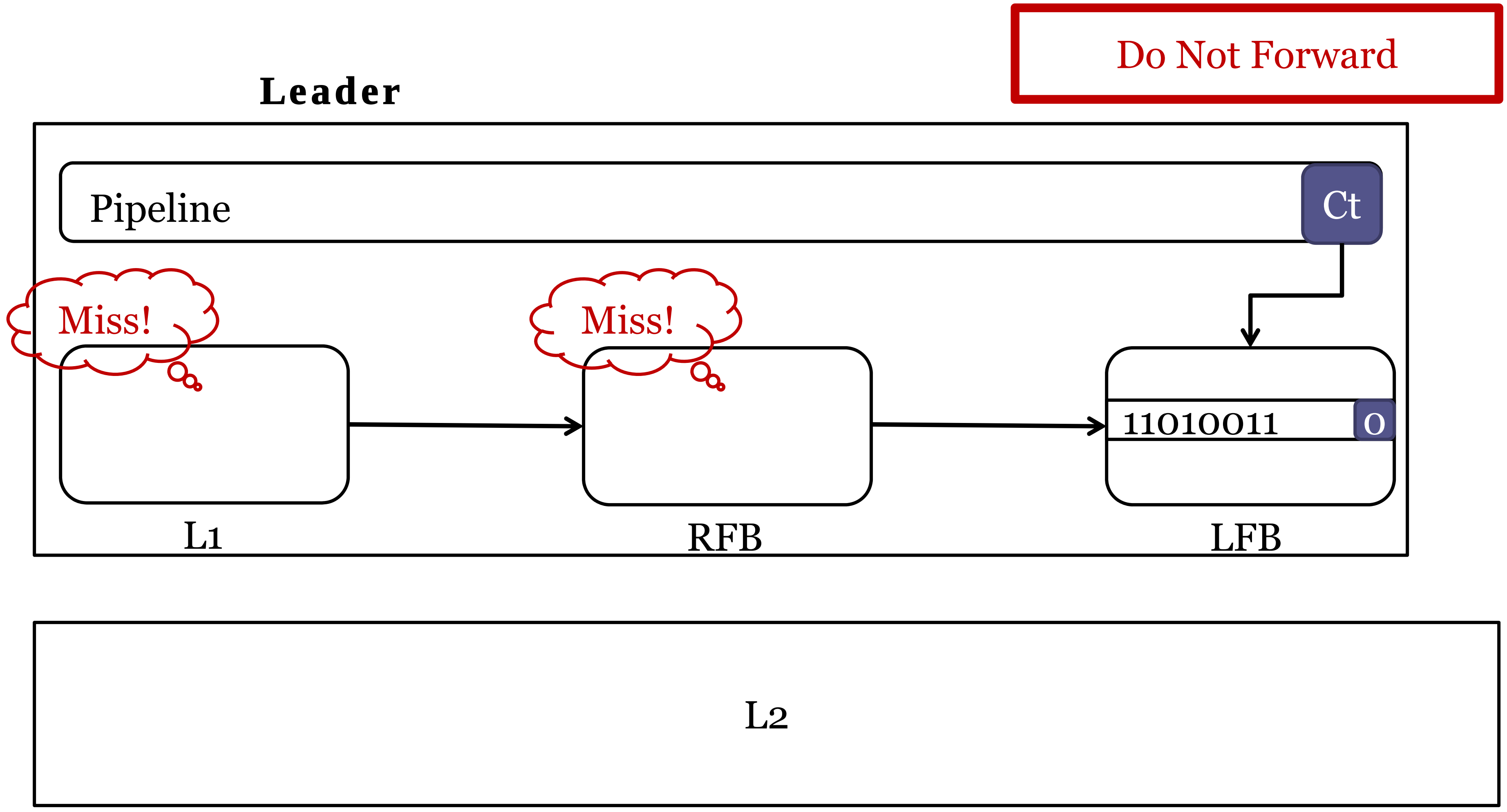
Forwarding Filters



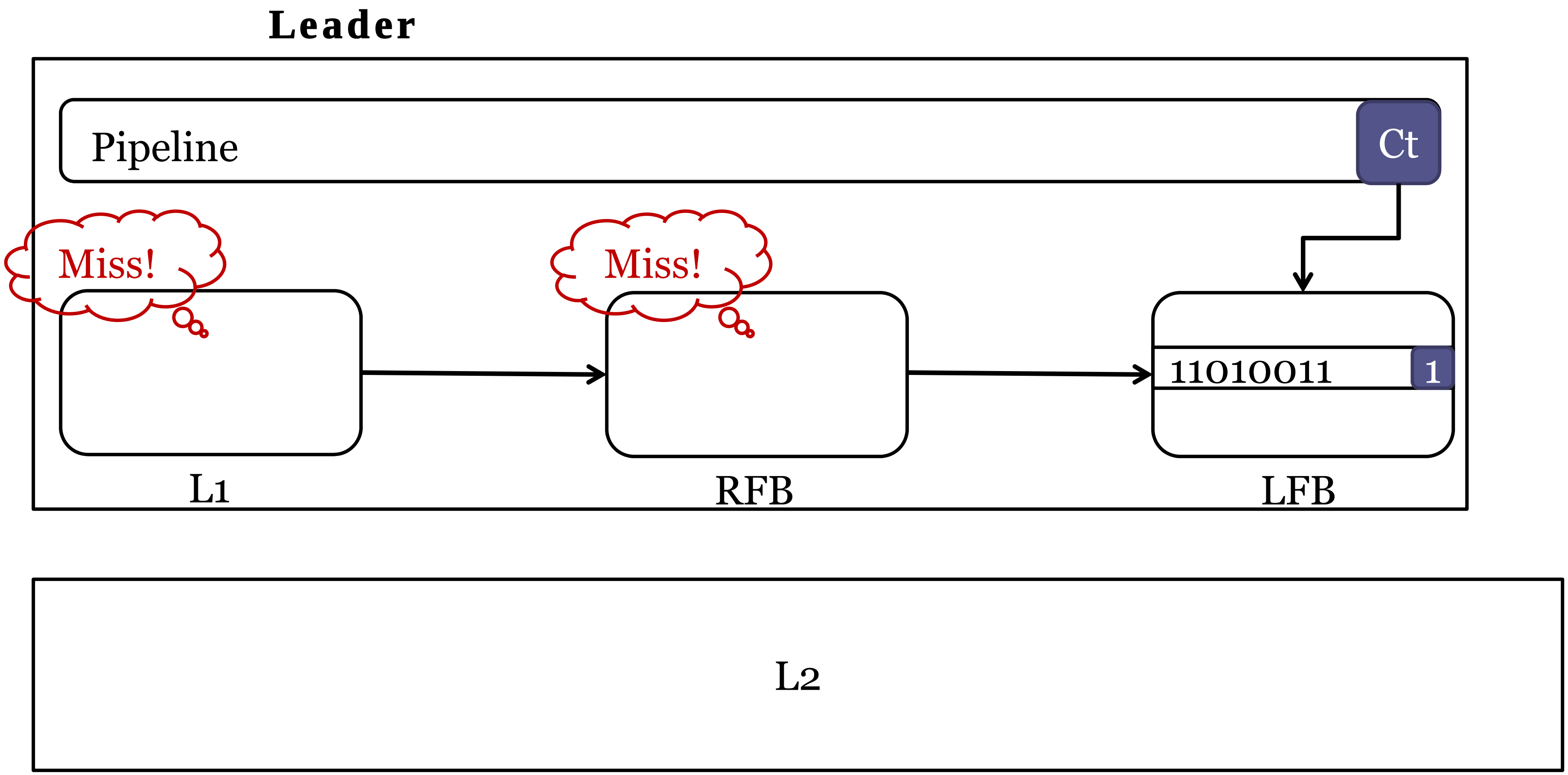
Forwarding Filters



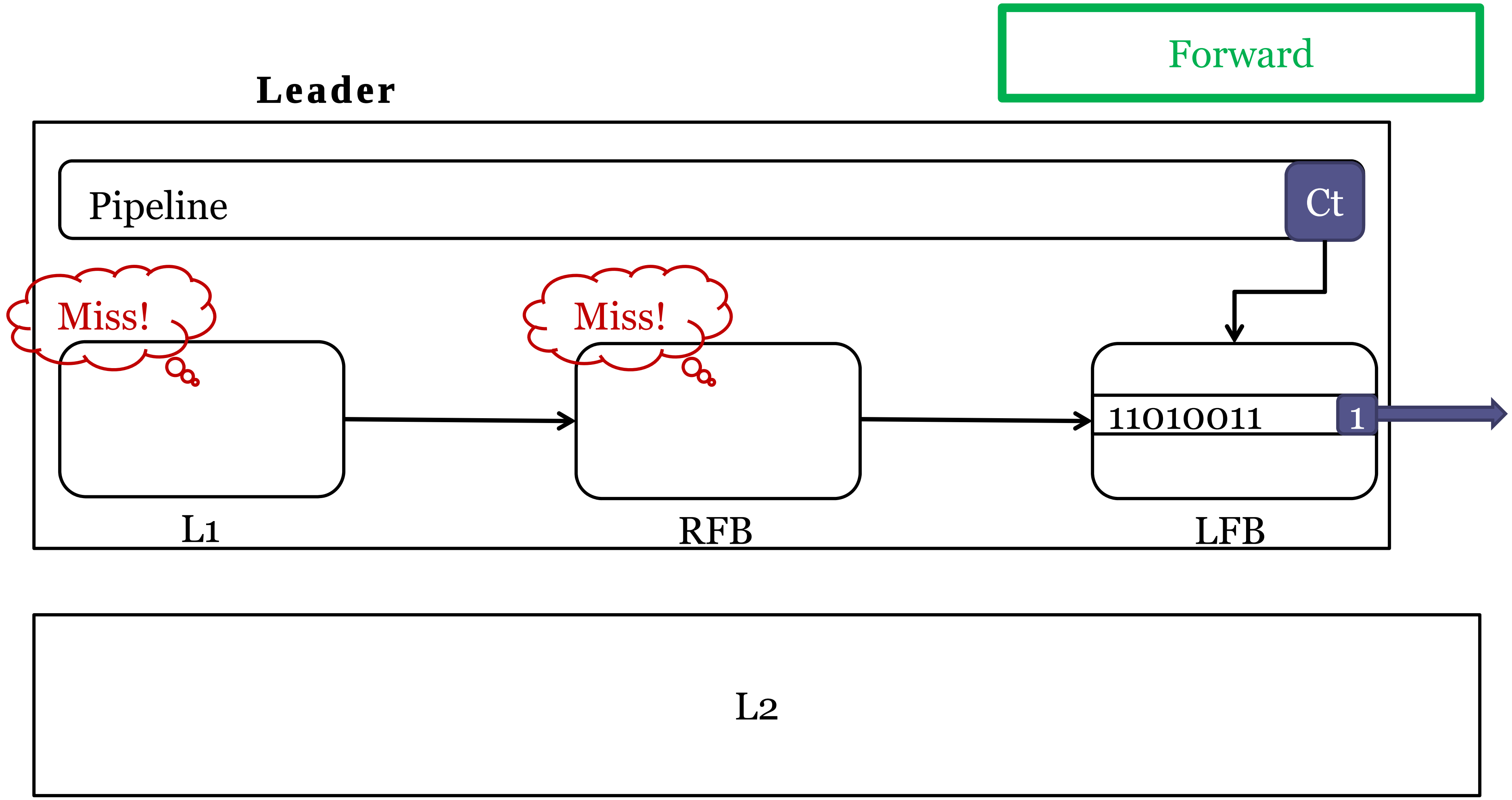
Forwarding Filters



Forwarding Filters



Forwarding Filters



Arbiter Logic: I

- Activity
  - IPC
  - WIPC(x)
- Mapping a Single Thread
  - Select the core with minimum *activity* that has free SMT slots
  - If activity is IPC, scheme is termed *minIPC*
  - If activity is WIPC(x), scheme is termed *minWIPC_x*

Arbiter Logic: II

- Mapping a Set of Threads
  - Scheduling Policies:
    - Pinned Leaders (SP-PL)
    - Unpinned Leaders (SP-UL)
    - Unpinned Leaders All Leaders First (SP-UALF)
- SMT Fetch Policy
  - Full Simultaneous Issue [Tullsen1995]
  - If *n* threads on a core have activities $A_1, A_2 .. A_n$, then the *i*th thread gets fetch cycles (cycle block of size *B* considered) $\frac{A_i}{\sum_{k=1}^{n} A_k} \times B$

Evaluation: Simulation Parameters

- 16-core processor, 4-way SMT
- Core configuration based on Intel Sandybridge and IBM Power7

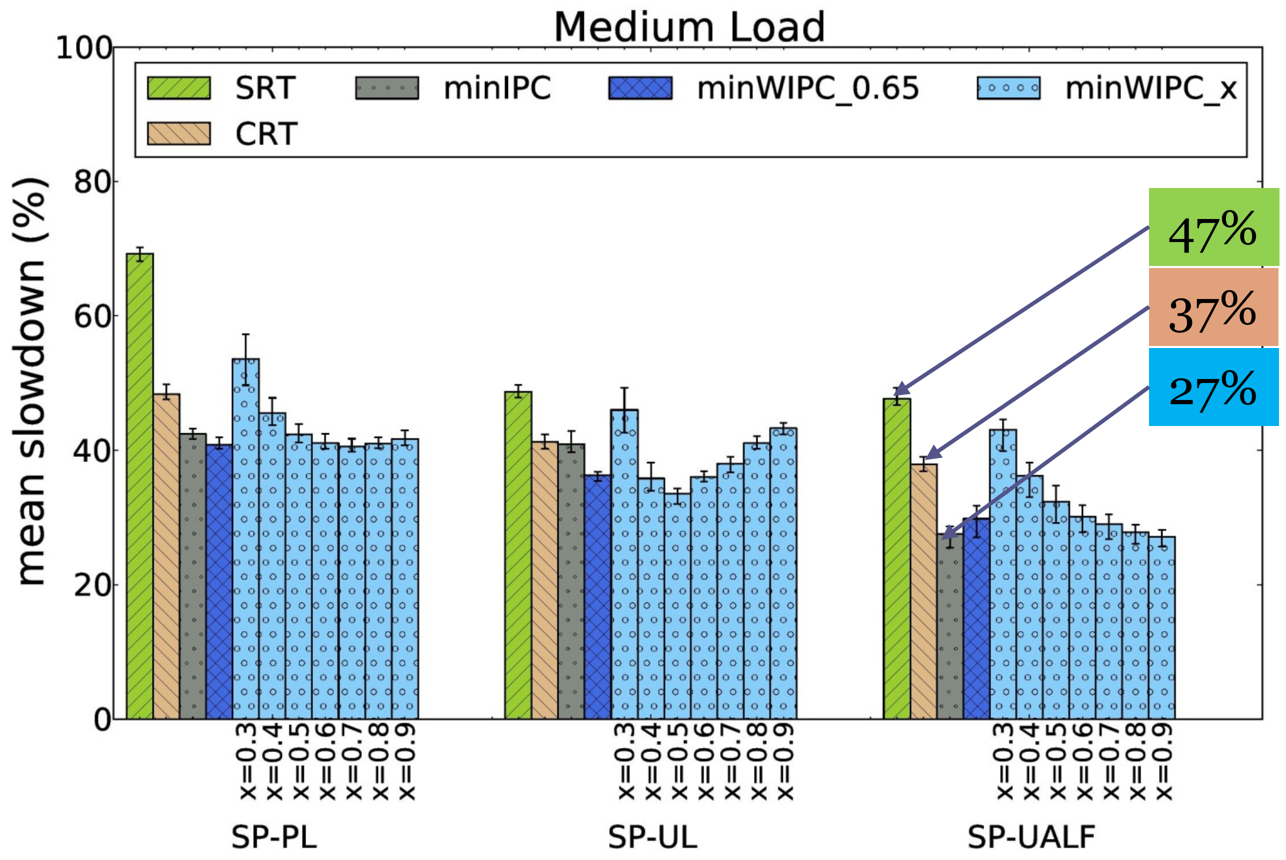
Parameter	Value
Pipeline width	4
i-cache and d-cache	32 kB
Shared L2 cache	12 MB
NOC topology	2D torus
Hint buffer	512 entry
Victim Cache	32 entry
RFB and LFB	64 entries each

Evalation Methodology

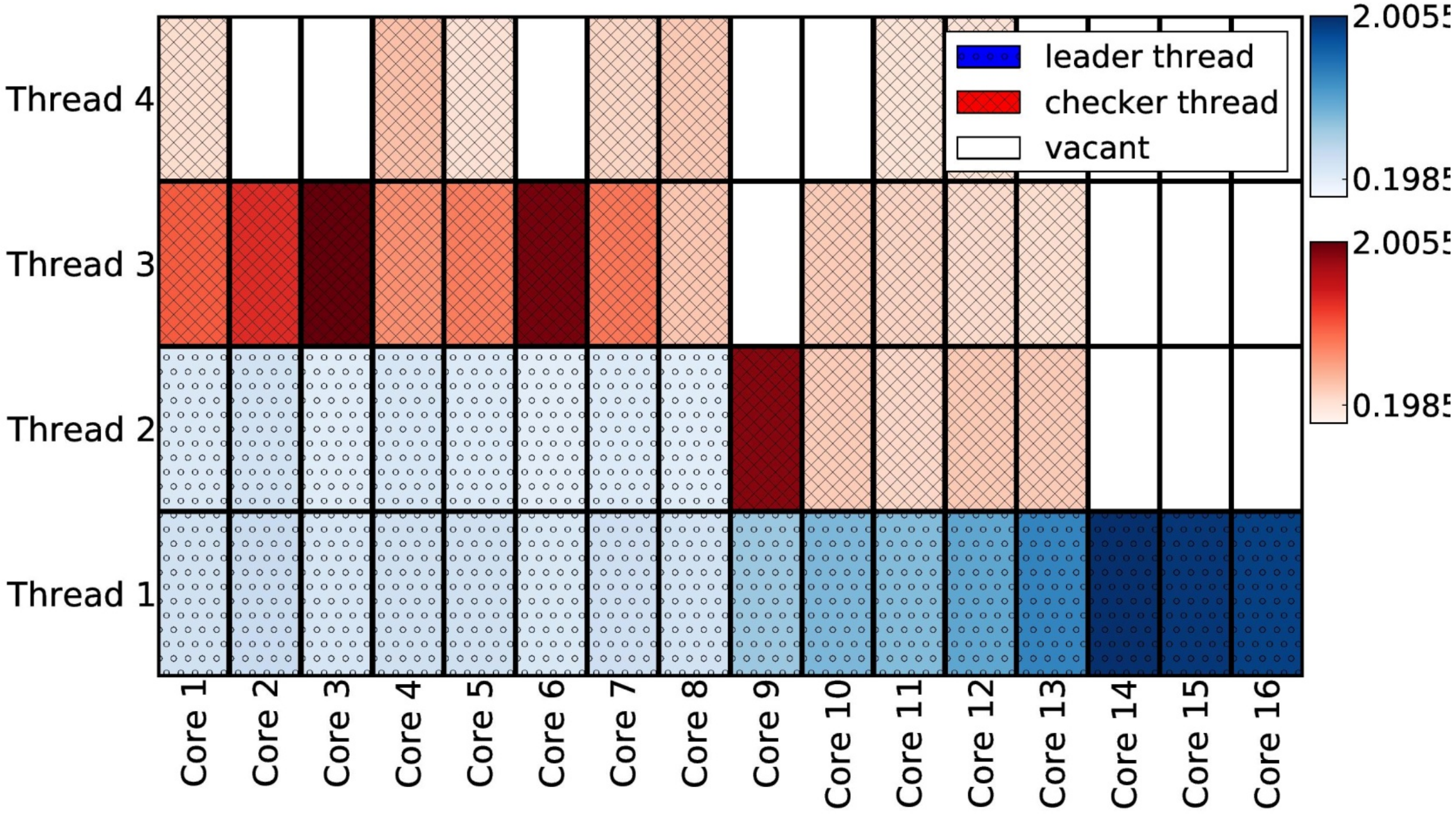
- Tools
  - Tejas Architectural Simulator
  - McPAT and Orion2 models
- Workloads
  - “low”: 16 applications (16 + 16 threads)
  - **“medium”: 24 applications (24 + 24 threads)**
  - “high”: 32 applications (32 + 32 threads)
  - In each case 100 random combinations of SPEC CPU2006 benchmarks were considered
- Comparison Metric

$$\sqrt[|W|]{\prod_{b \in W} \frac{\text{cycles taken to reliably execute } b}{\text{cycles taken to unreliably execute } b}} - 1$$

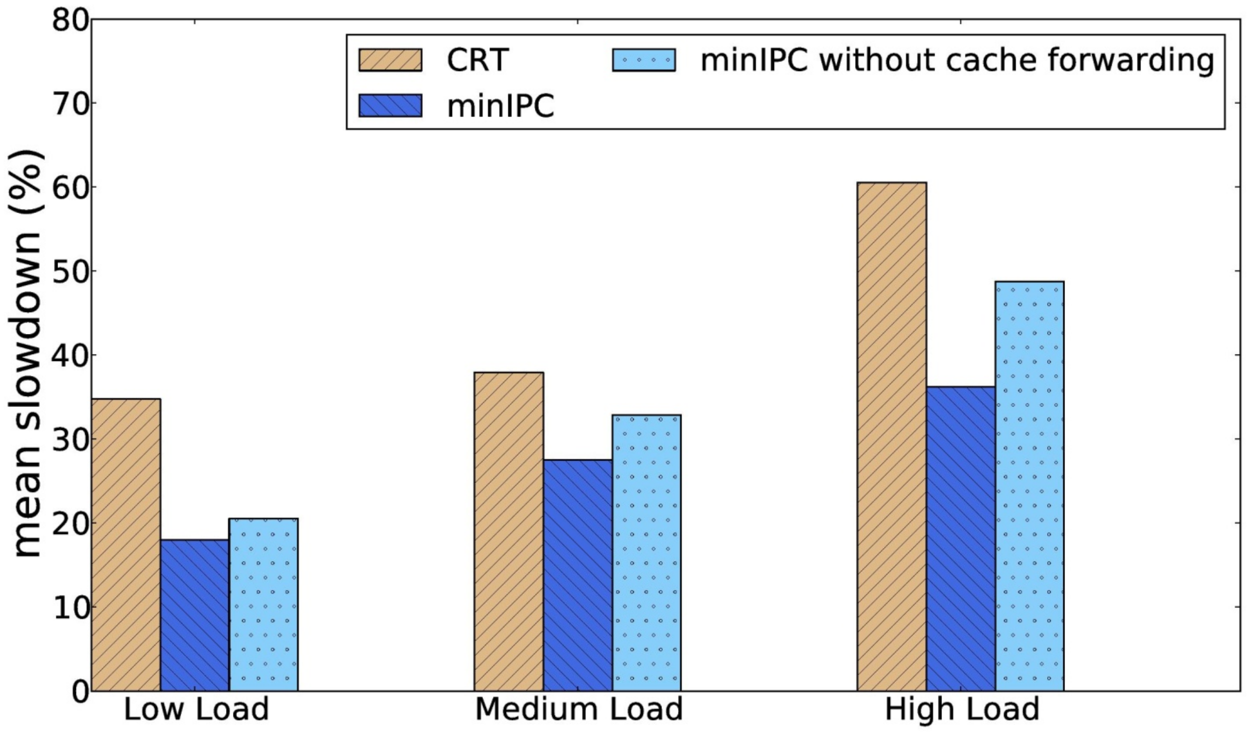
Evaluation: Results



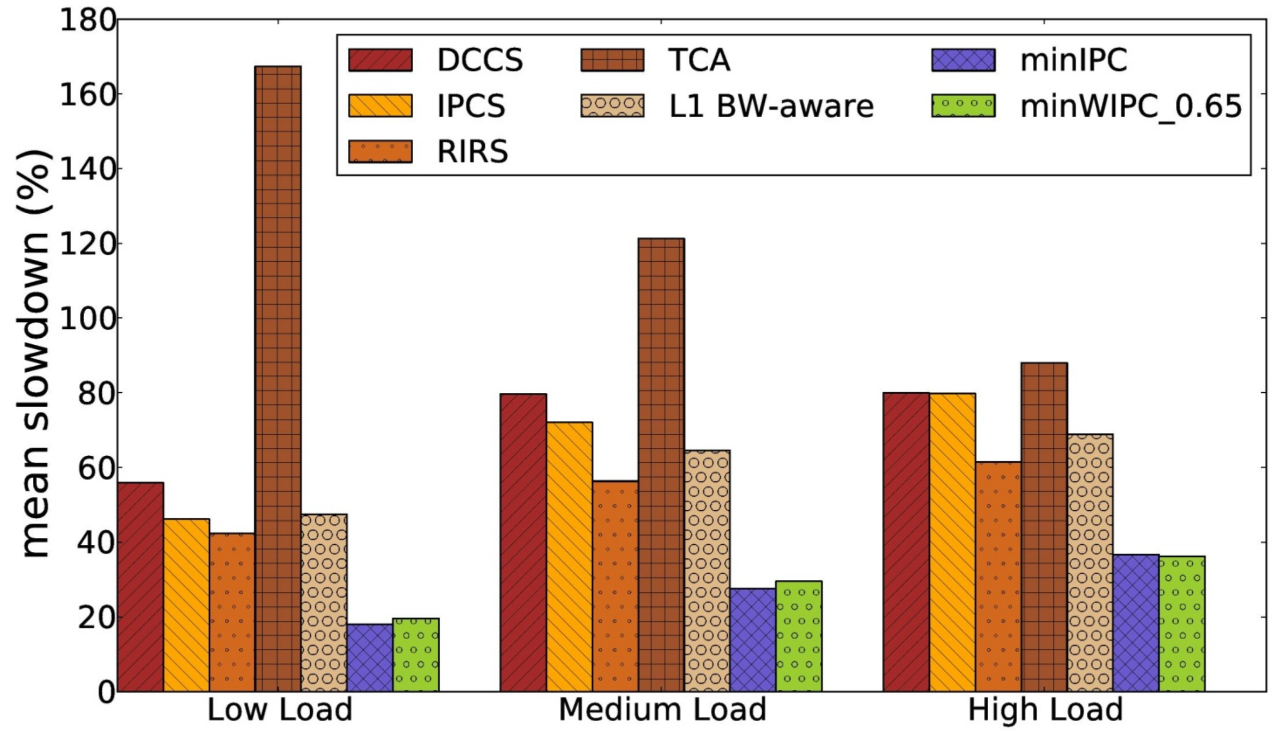
FluidCheck’s Mapping Ability



Performance of Forwarding Filters



Comparison with Generic Scheduling Schemes



- DCCS [Settle2004]
- TCA [Acosta2009]
- IPCS [Parekh 2000]
- L1 BW-aware [Feliu2013]
- RIRS [ElMoursy2006]

Conclusions

- Efficient system-level solutions to handle soft errors are critically sought
- The protection of modern multi-core, multithreading capable processors presents interesting challenges
- Our solution FluidCheck achieves reliability with a mere 27% reduction in performance on average, while seminal works such as SRT (47%) and CRT(37%) present much higher slowdowns

Extra slides

DIVA : Checker Operation

Fetch
- Check IP
- Fetch From IP
- <0x1234>

Decode
- <R1=R2+R3>

Execute
- Using the operand value hints
- <5+2>

Writeback
- Check communication
  - R2 == 5 ?
  - R3 == 2 ?
- Check computation
  - 7 == res ?
- Write 7 to R1

Commit
- Complete store

DIVA : Execution Assistance

- The DIVA checker
  - Faces no data hazards
    - Operand value hints are passed from leader
  - Faces no control hazards
    - The stream of packets from the leader are in correct dynamic order (if no soft error struck the prediction or branching logic)
    - If a soft error occurred (rare event), it is detected when the branch condition is evaluated at the checker

DIVA : Consequence of Execution Assistance

- What gains can be achieved through execution assistance?
  - Checker can be made simpler
  - Checker can be made slower
  - Checker can be made to do more work

Resolving Livelock Issues

- Suppose a checker thread faces a decode stall since the ROB was full
- Suppose some other leader thread on the same core is occupying the head of the ROB and is facing a long latency miss
- The checker thread is forced to migrate
- Possibility of multiple forced migrations in quick succession – detrimental to performance
- Solution – Reservation. If a resource is greater than 95% full, it will not accept any more leader entries