TECHNISCHE UNIVERSITÄT DRESDEN

Department of Computer Science Institute for Systems Architecture, Systems Engineering Group

Minimizing Latency in Fault-Tolerant Distributed Stream Processing Systems

Andrey Brito[1], Christof Fetzer[1], Pascal Felber[2]

[1] Technische Universität Dresden, Germany
[2] Université de Neuchâtel, Switzerland

ICDCS'09, June 23rd, 2009

Goal

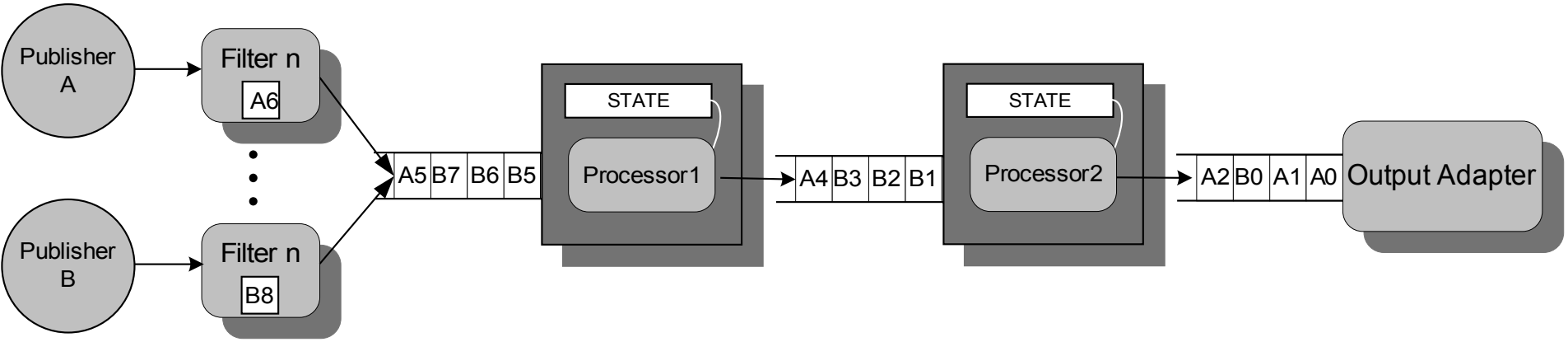
Minimize the cost of logging/checkpointing in event stream processing systems

Contribution: Usage of an speculation framework based on transactional memory to overlap logging and processing

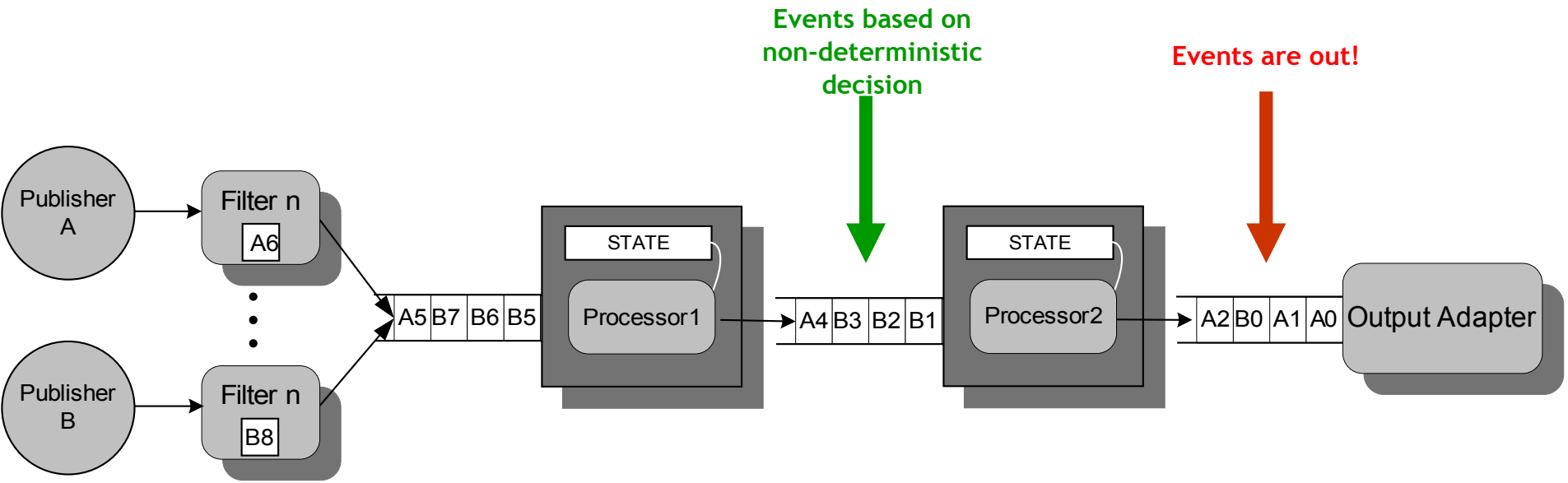
Motivation (1)

- Event stream applications
  - Directed acyclic graph of operators
  - Some operators don't keep state
    - Trivially parallelizable
  - Some do keep state
    - Not trivially parallelizable
  - Sometimes they are order sensitive
    - Need to process events sequentially, maybe even waiting for the order to be restored

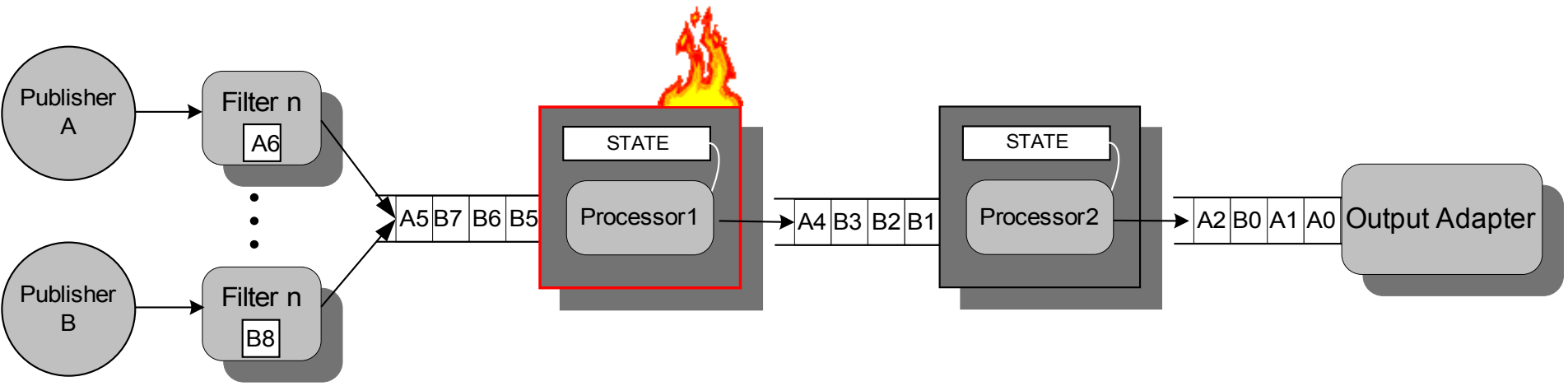
Application example



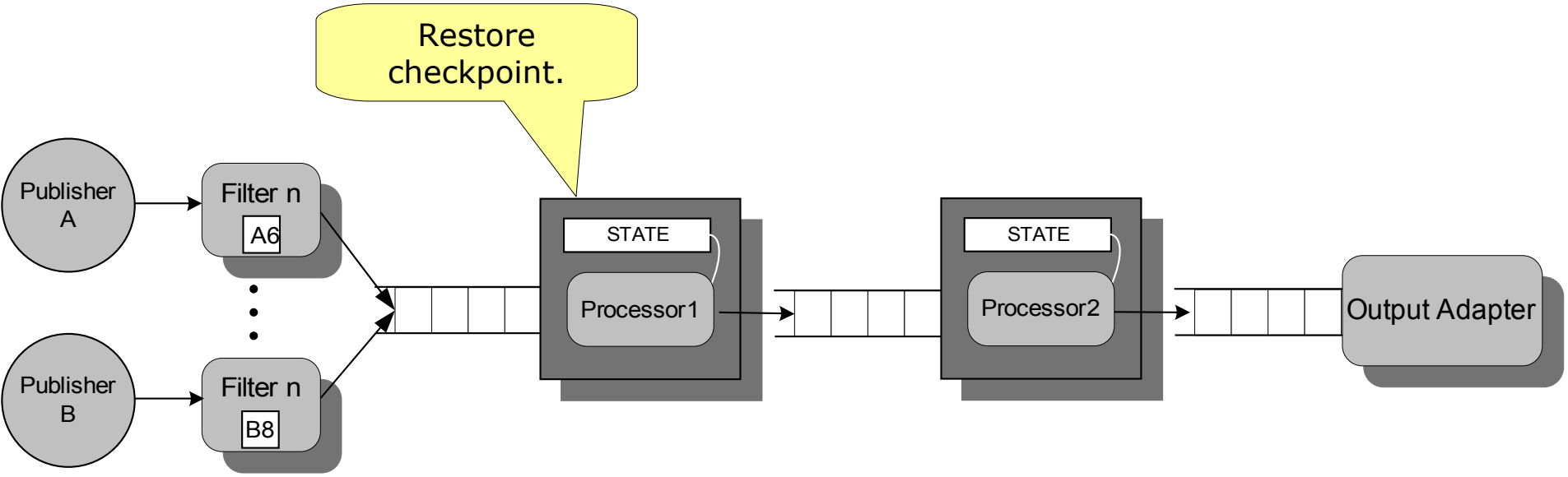
Application example



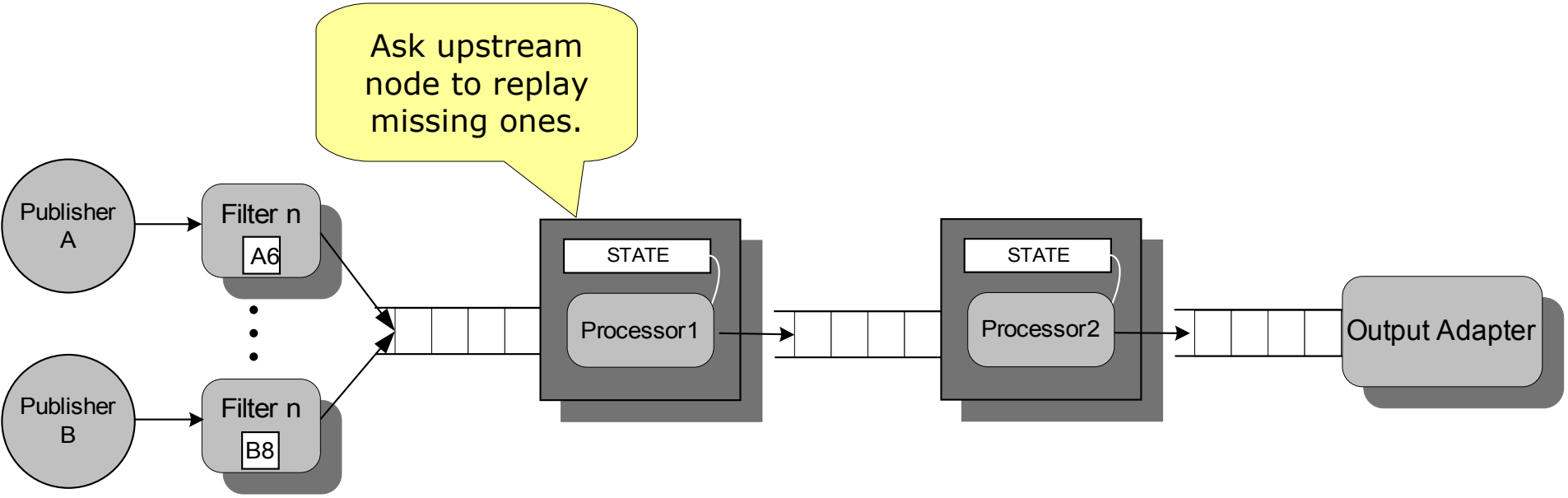
Application example



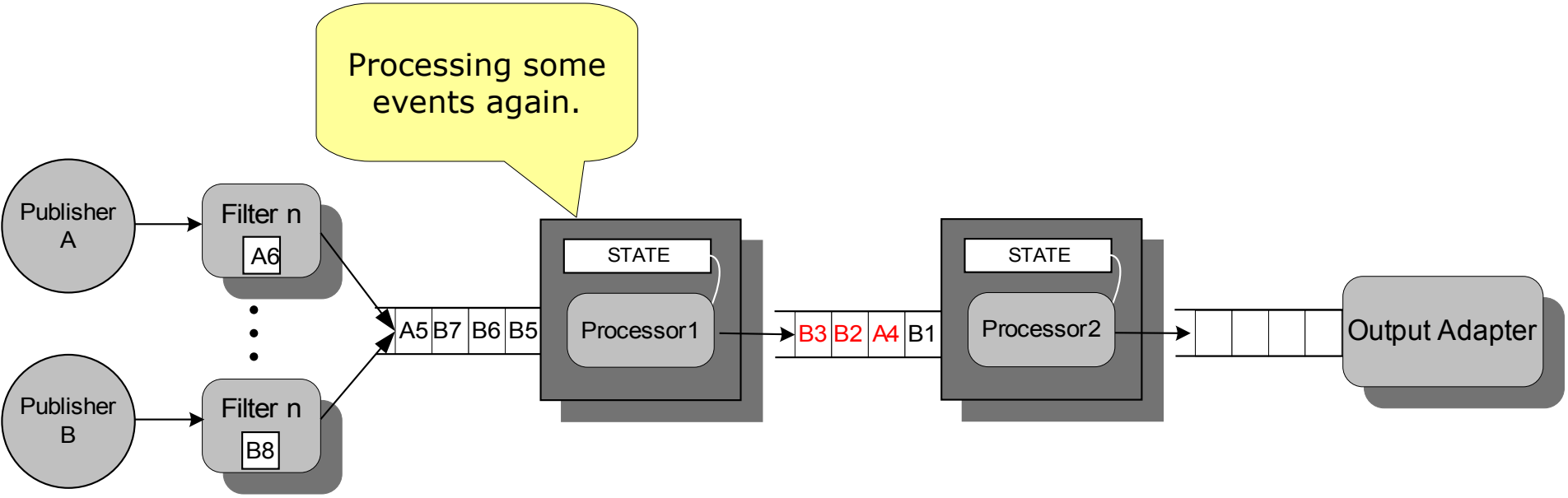
Application example



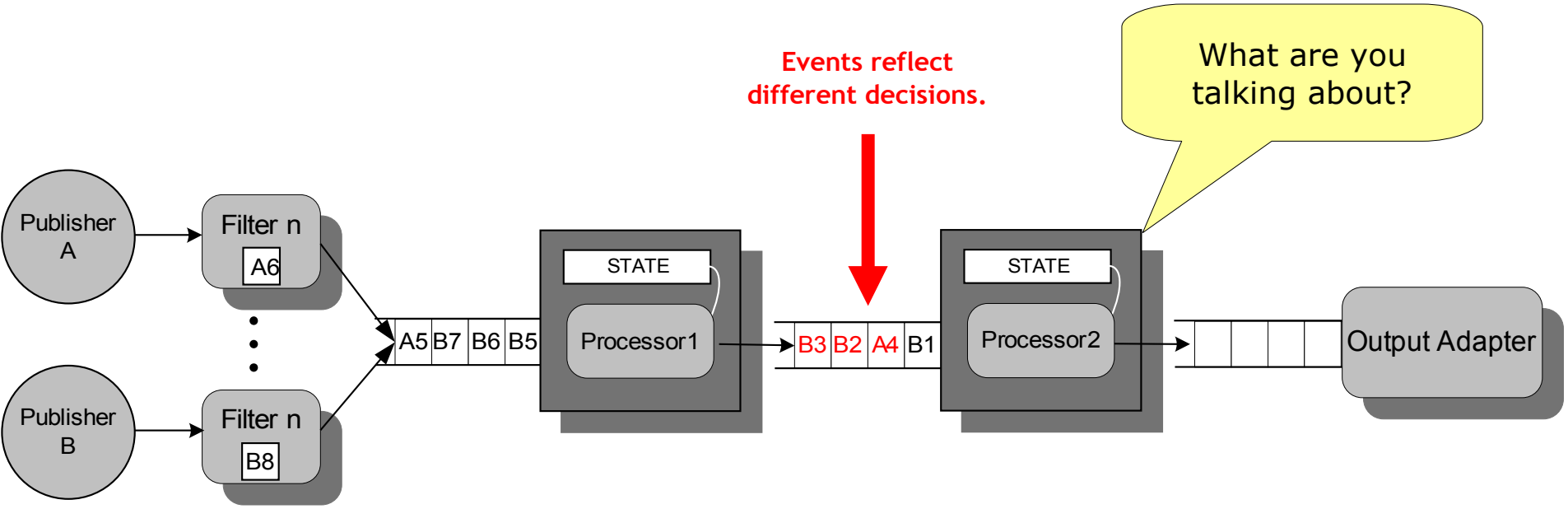
Application example



Application example



Application example

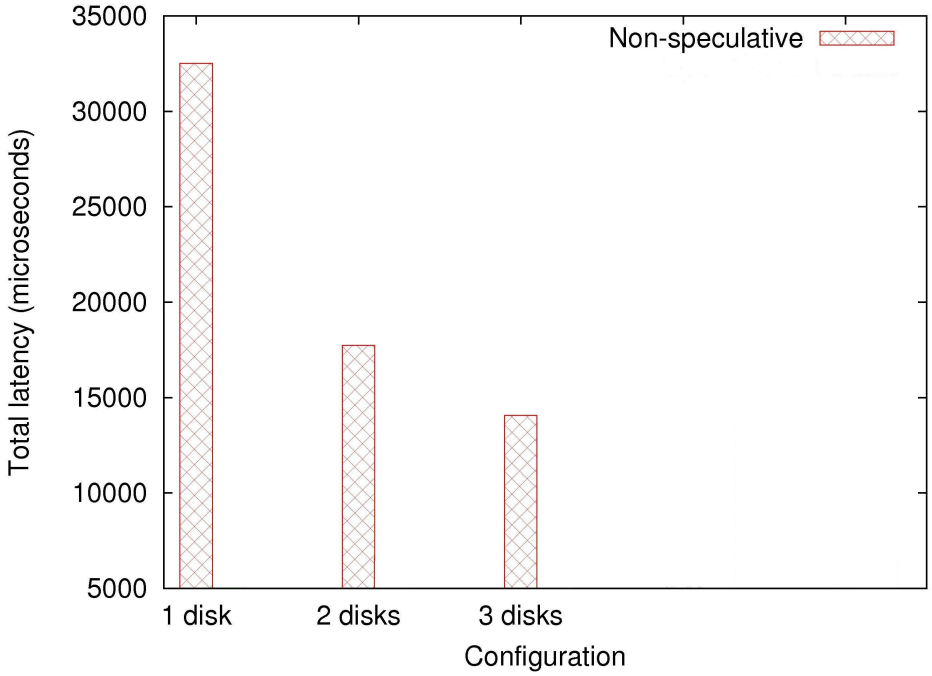


Incomplete log of non-deterministic decisions → no repeatability

Motivation (2)

- Fault-tolerant event stream applications
  - Precise recovery
  - Even if order does not matter, repeatability does
  - Non-determinism
    - Input order from different streams
    - Non-determinism in processing (multi-threading, time, random numbers)
  - Log or checkpoint before each output

Logging is expensive



My solution

- Speculate...
- … to parallelize stateful components
- … to not have to wait for events
- … to not have to wait for logging

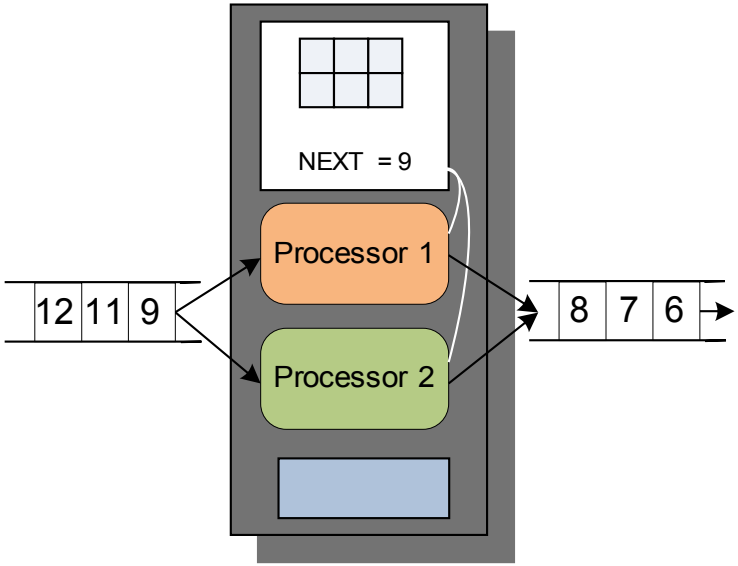
Outline

- How the speculation works
- Logging algorithm
- Experiments
- Final remarks

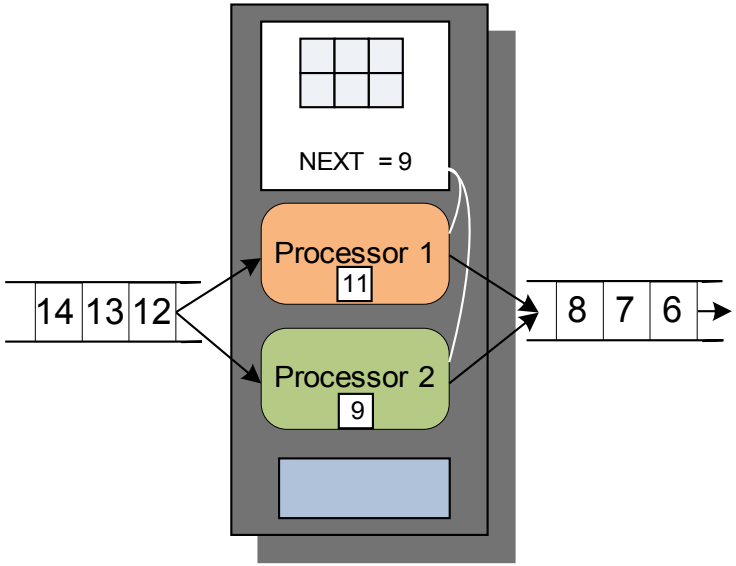
How the speculation works

- Base: TinySTM
  - Some extra features added
  - But same basic rule: “it appears to be atomic”
- Goal: track accesses to shared memory
  - Instrumentation
    - Reads and writes are intercepted
    - Hold back writes, validate reads until all dependencies satisfied

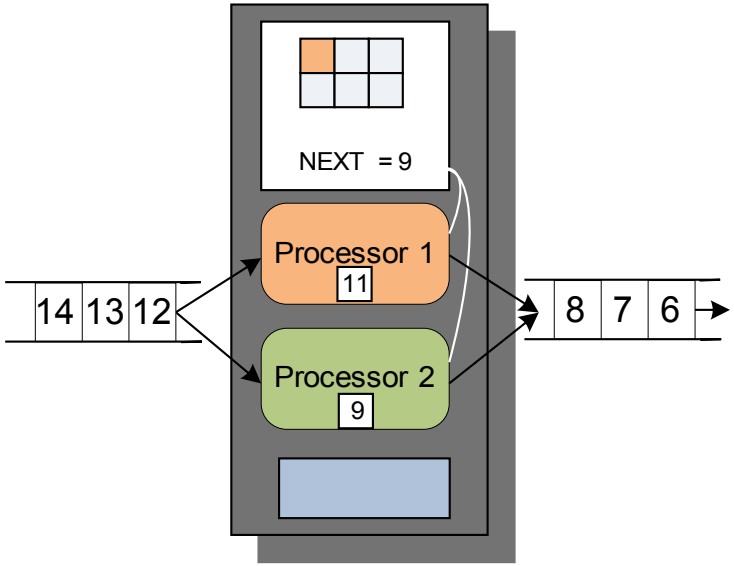
Speculative execution: parallelization



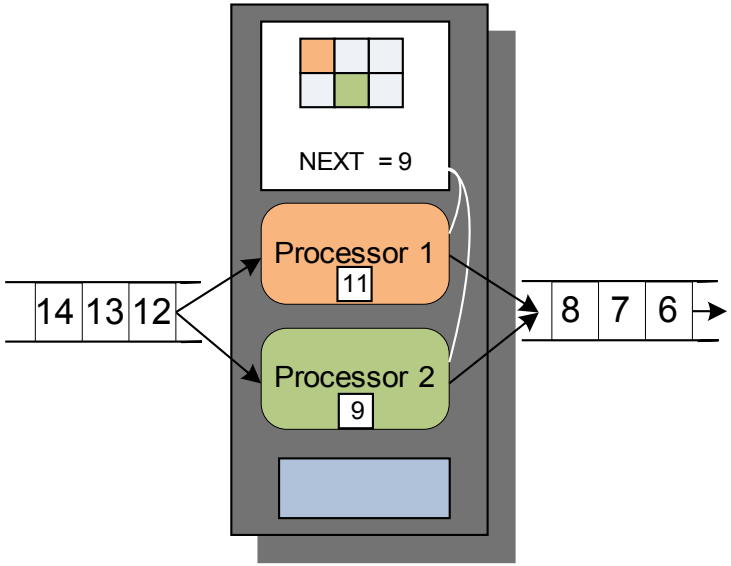
Speculative execution: parallelization



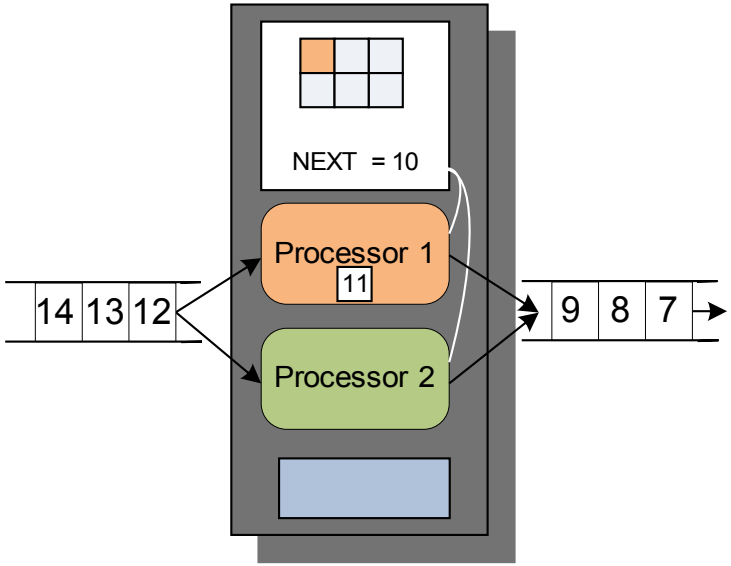
Speculative execution: parallelization



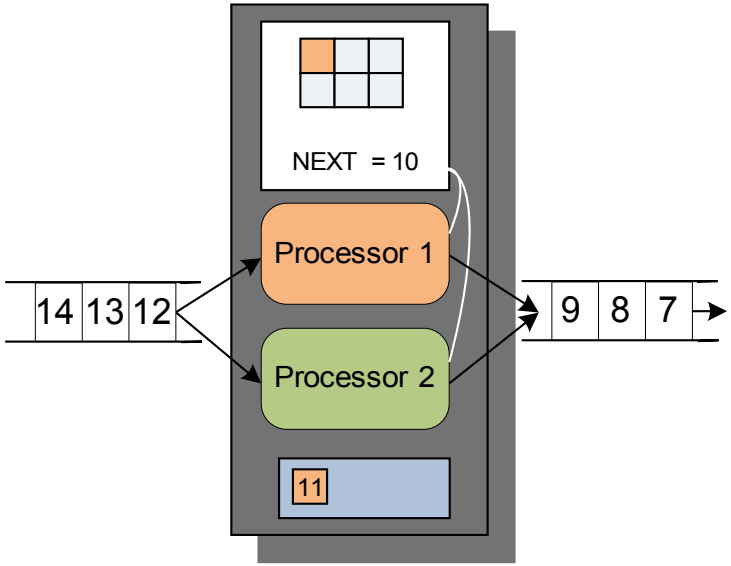
Speculative execution: parallelization



Speculative execution: parallelization



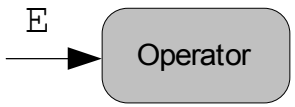
Speculative execution: parallelization

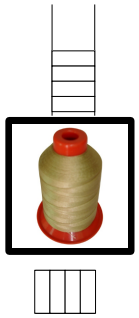


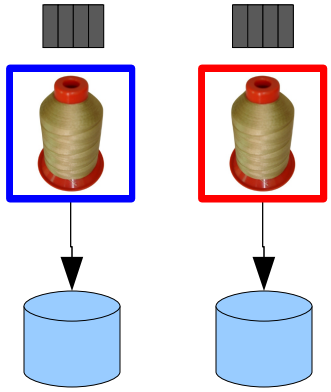
Logging algorithm

- Operator enqueues all events & decisions
- N+1 threads for N disks
  - One groups requests in a buffers
  - The others write their buffers to disk

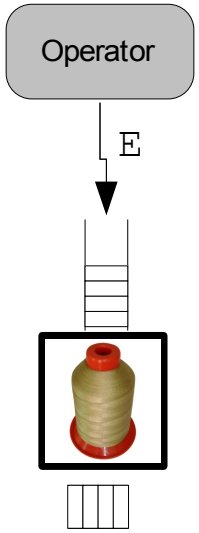
Logging algorithm

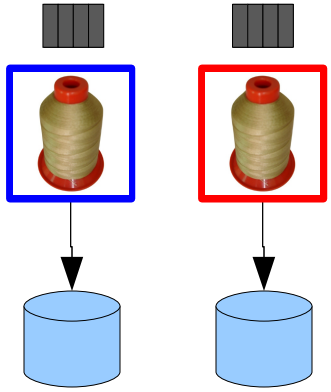






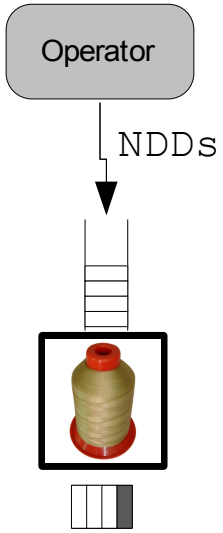
Logging algorithm

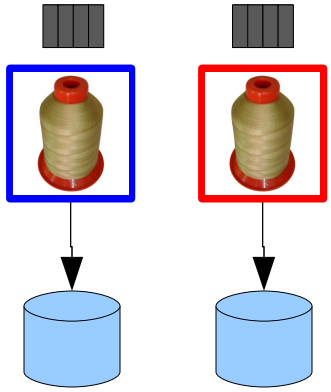




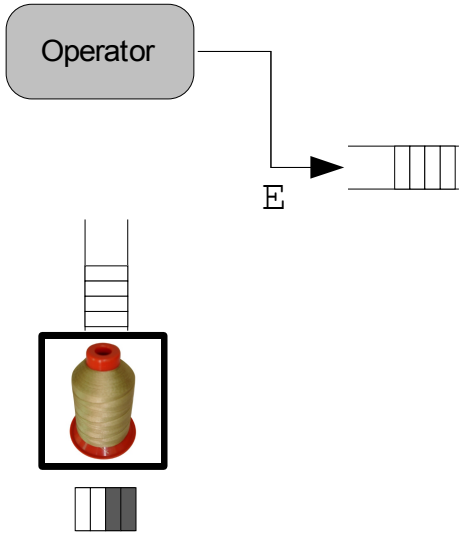
Logging algorithm

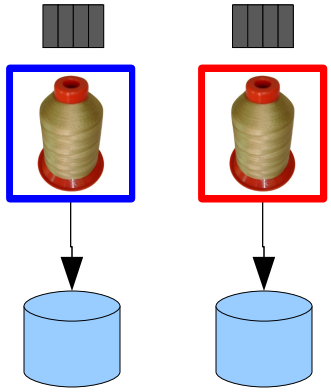
Logging algorithm



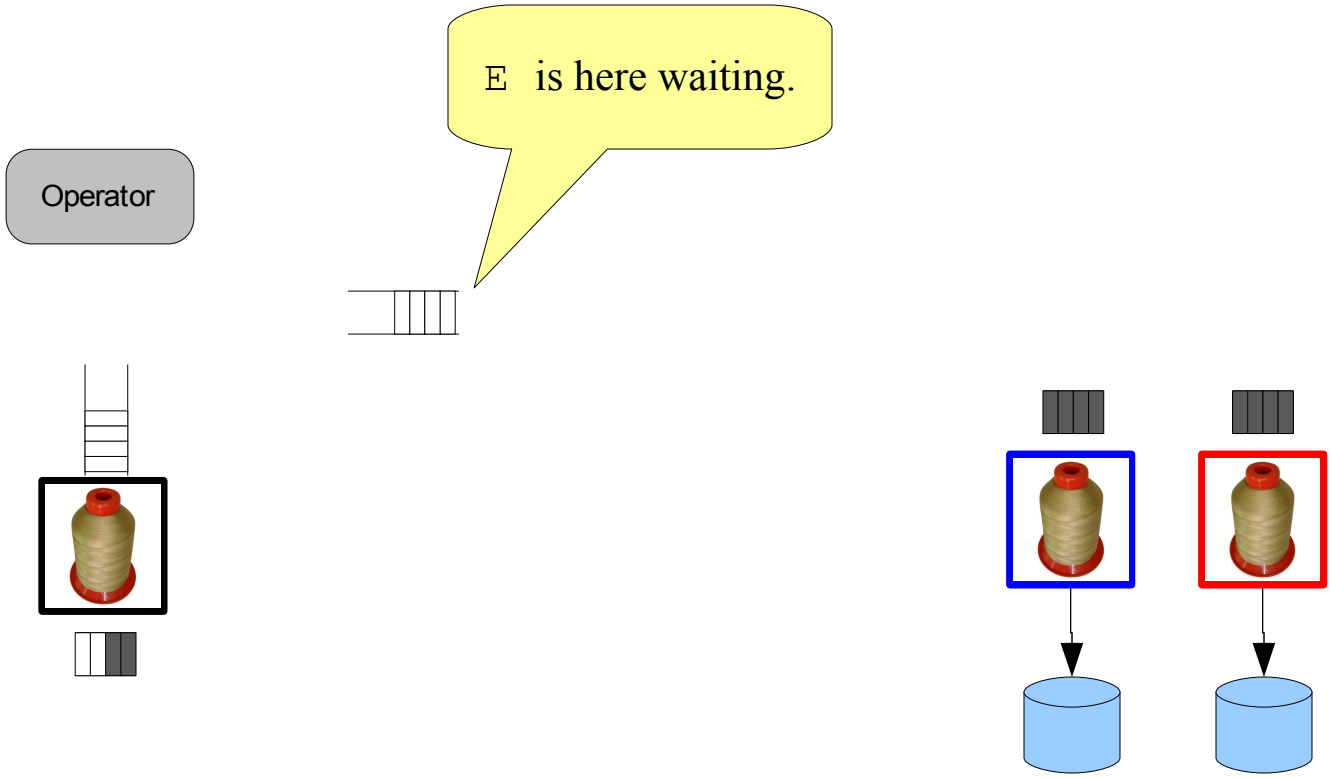


Logging algorithm

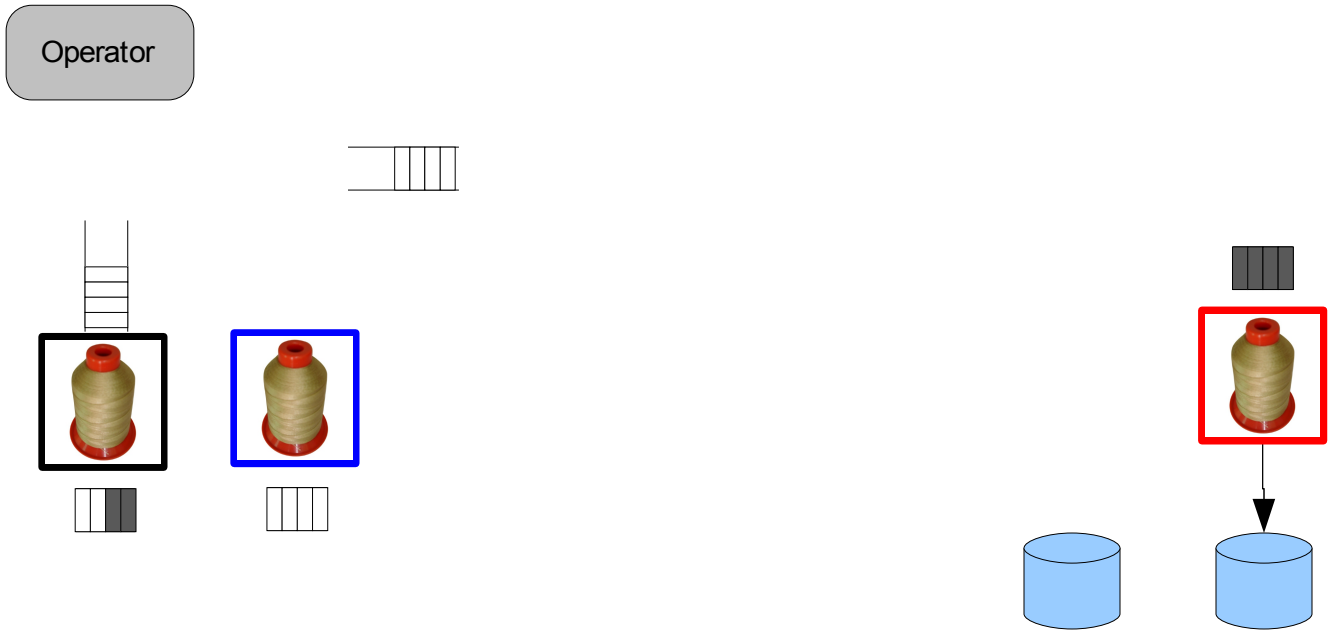




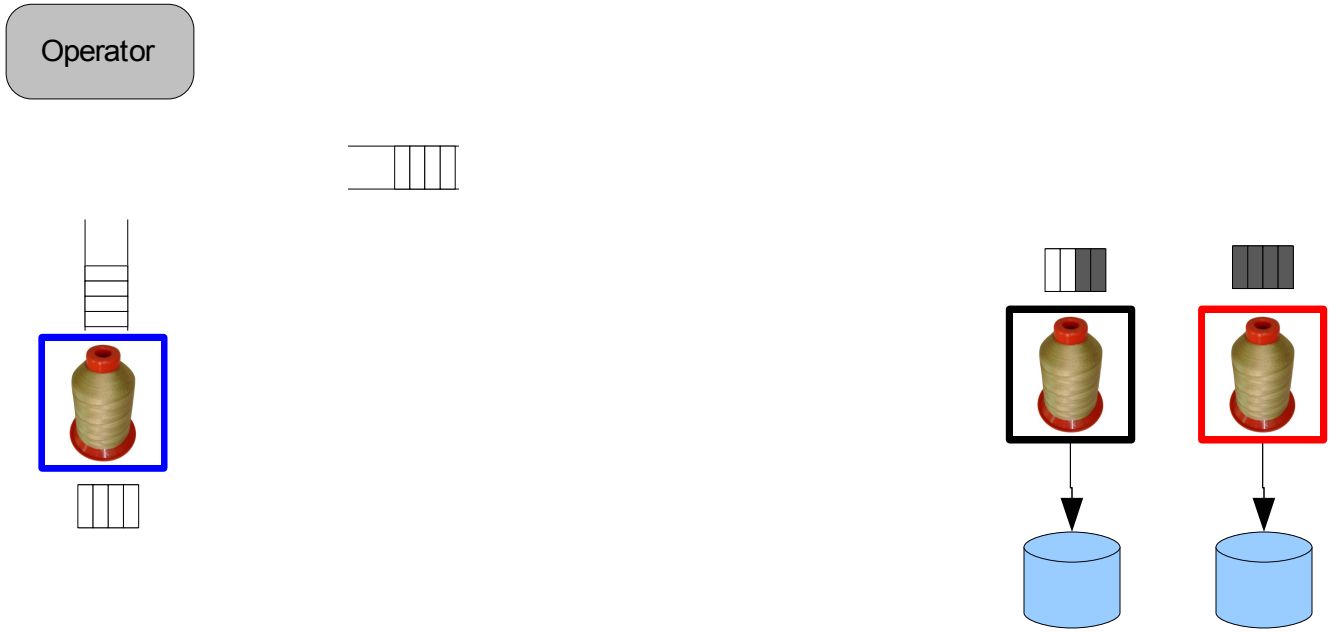
Logging algorithm



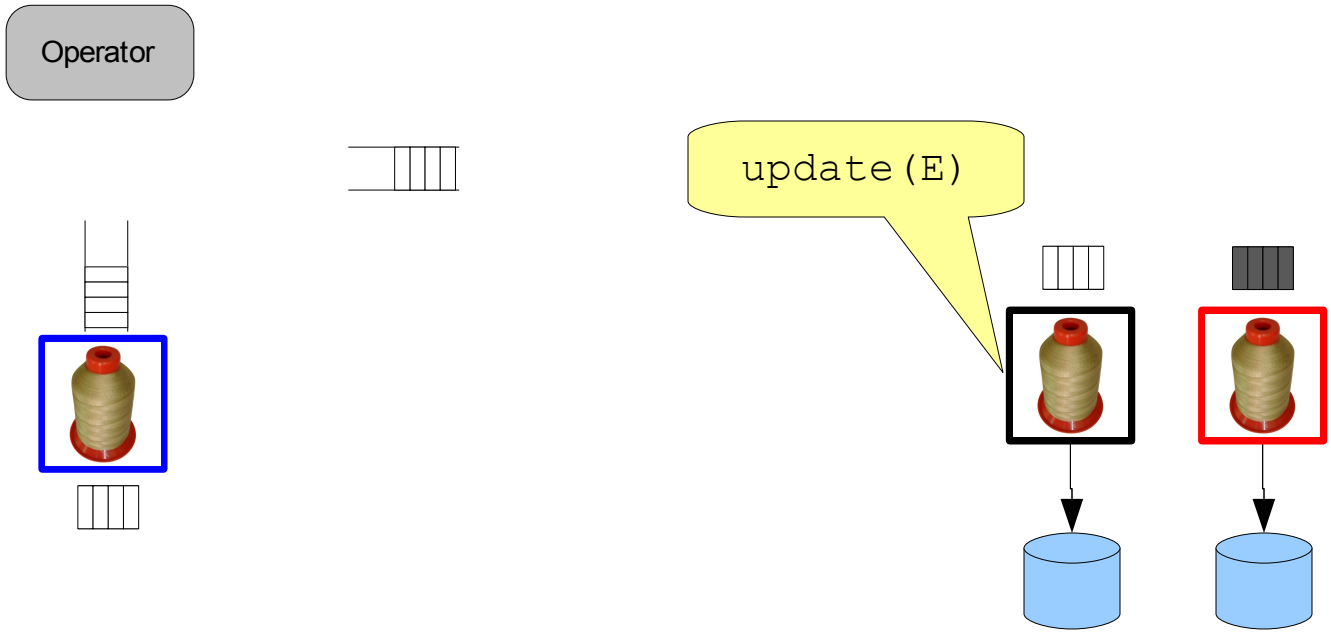
Logging algorithm



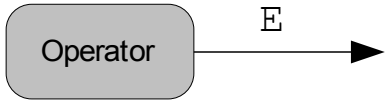
Logging algorithm

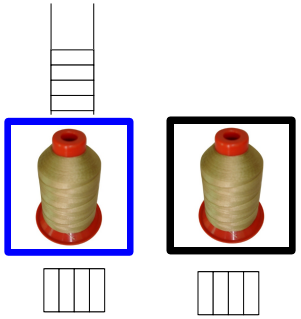


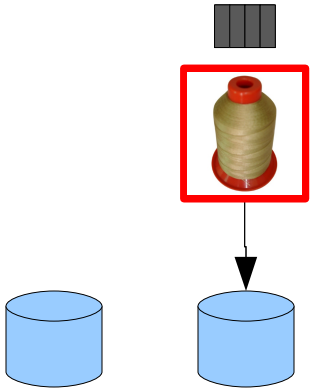
Logging algorithm



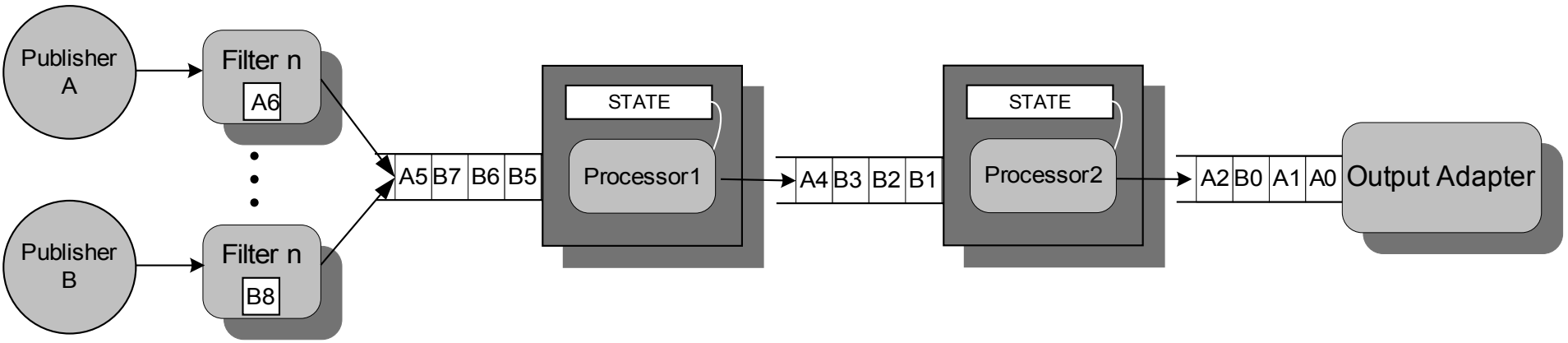
Logging algorithm



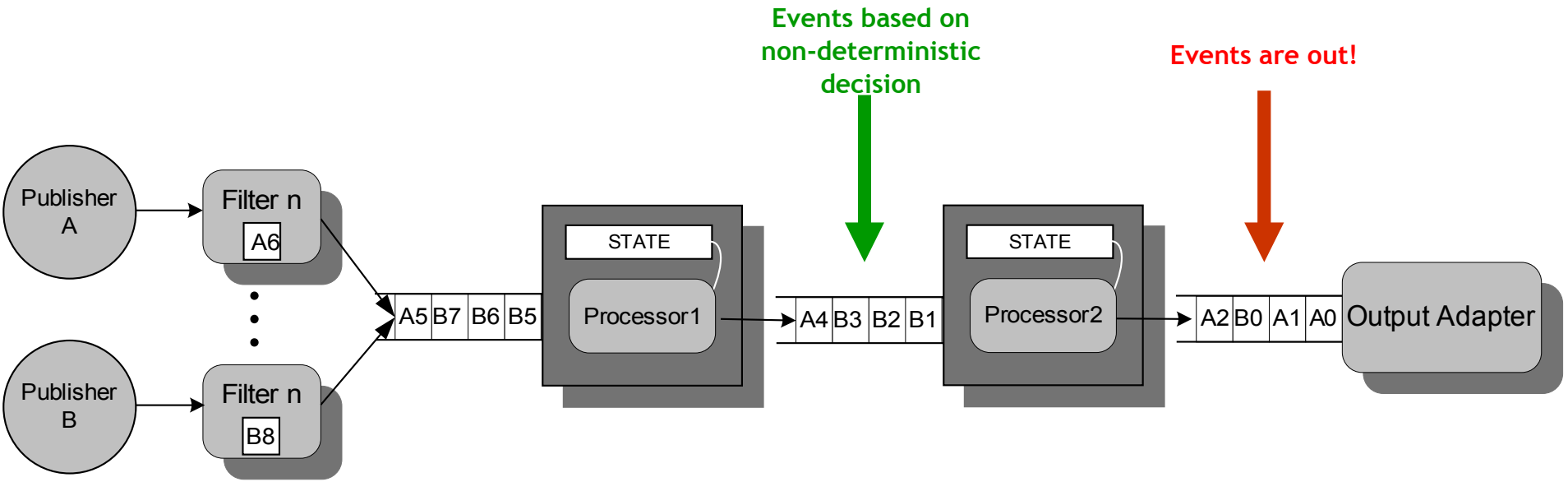




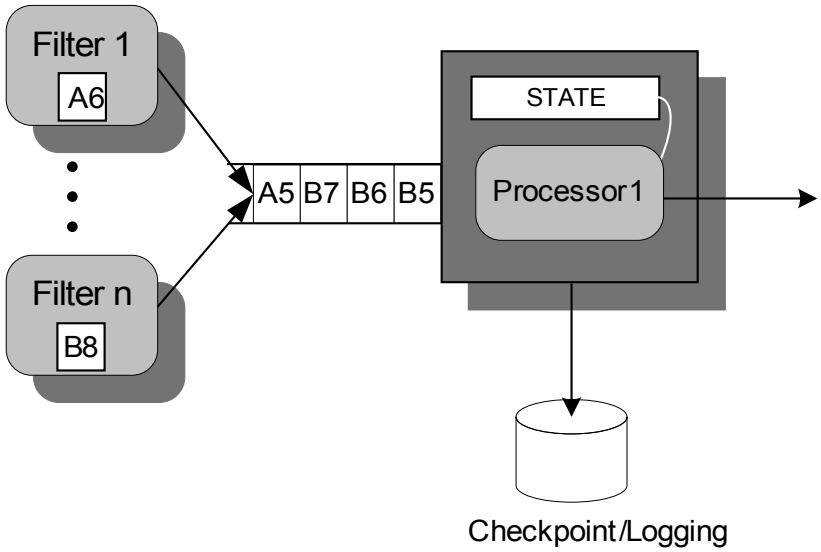
Logging algorithm



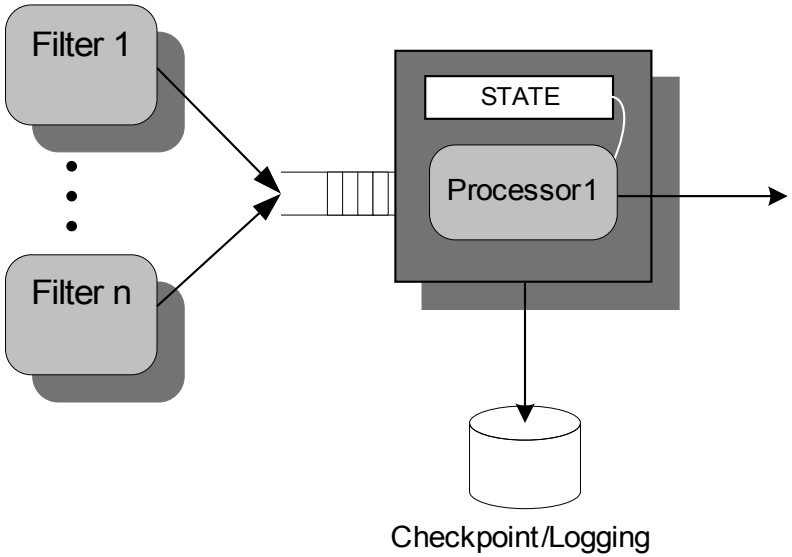
Logging algorithm



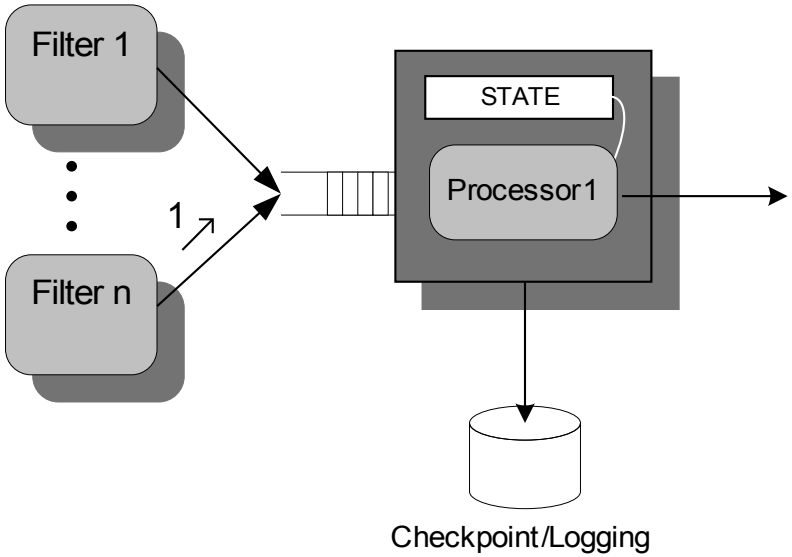
Logging algorithm



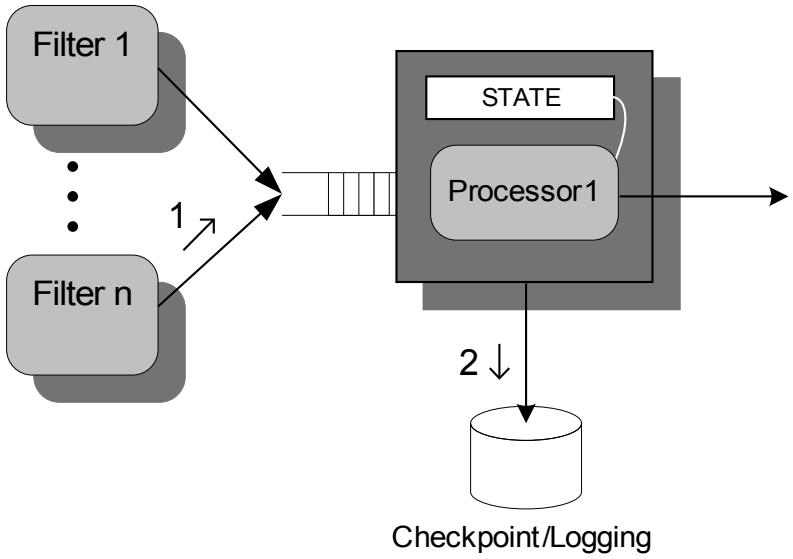
Logging algorithm



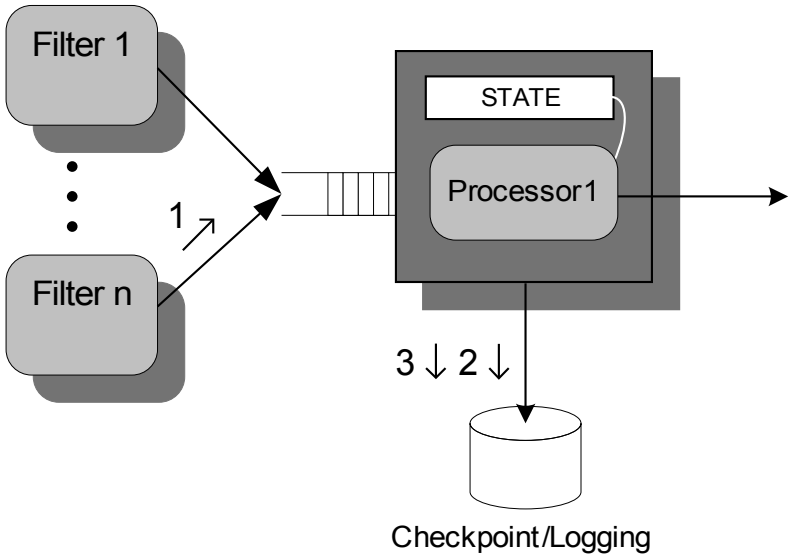
Logging algorithm



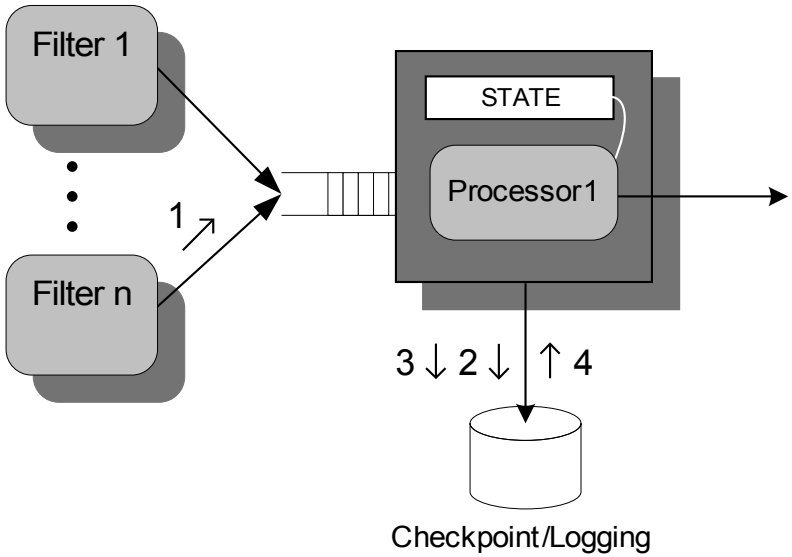
Logging algorithm



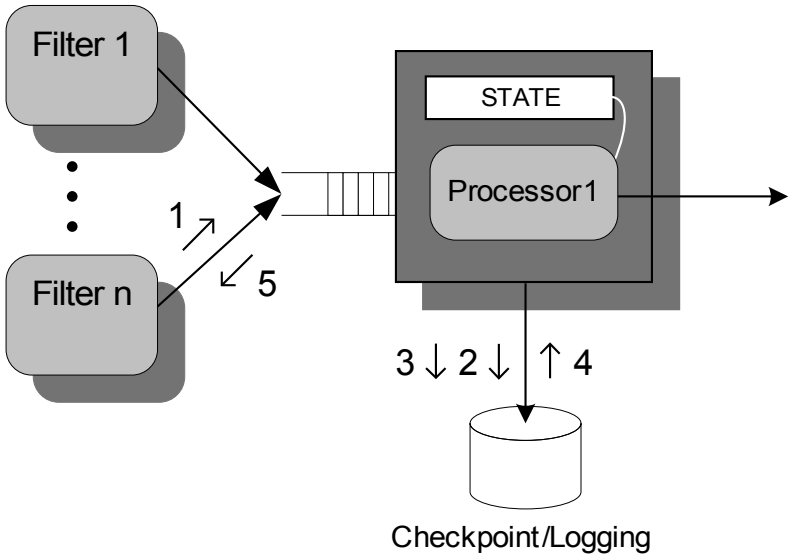
Logging algorithm



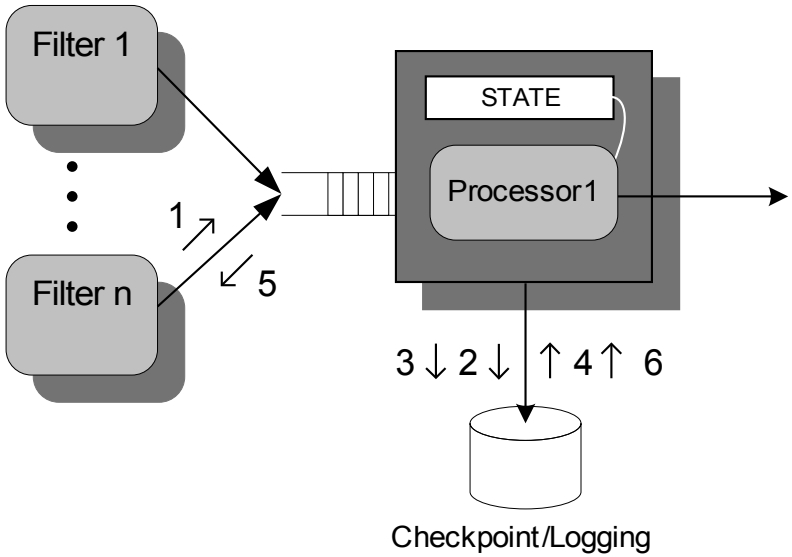
Logging algorithm



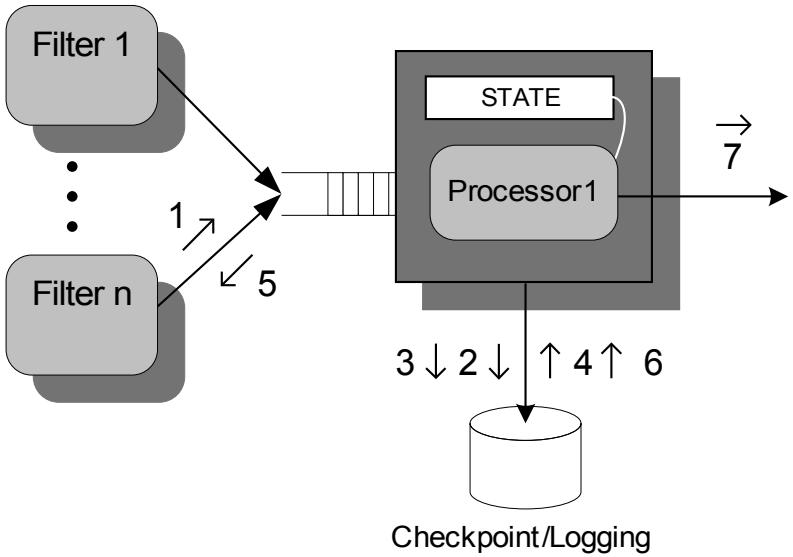
Logging algorithm



Logging algorithm



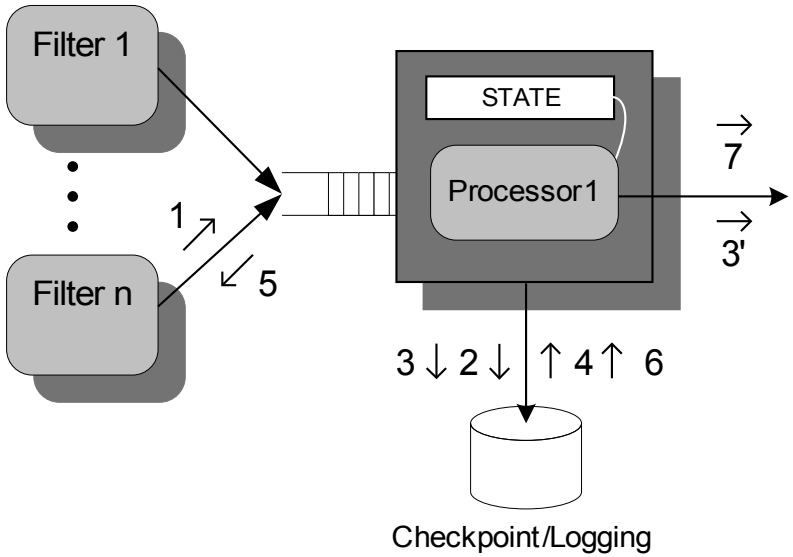
Logging algorithm



Speculative processing + Logging

- From the original node's viewpoint
  - Emit outputs as speculative
  - When logging requests are acknowledged, emit final
- The next downstream node
  - If speculative event modifies some state, keep track
    - Outputs that consider that part of the state are speculative
    - Speculative status is contagious

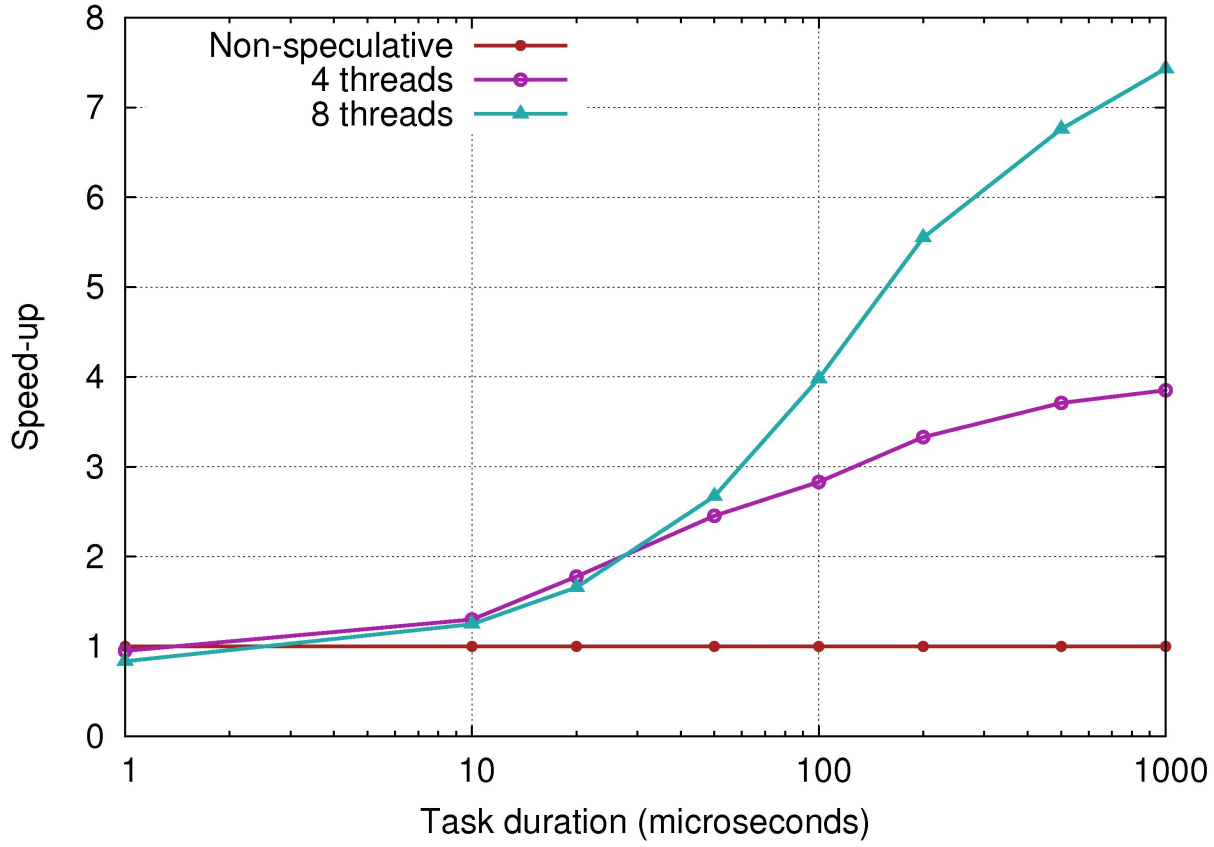
Speculation + Logging



Experiments

- Parallelization: benefits & STM's overheads
- Optimism control
- Overlapping processing and logging

Speculation costs & speed-ups



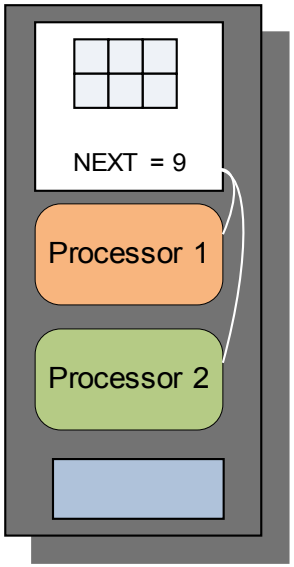
Hardware: Sun T1000

Speculation creation-commit-disposal overheads.

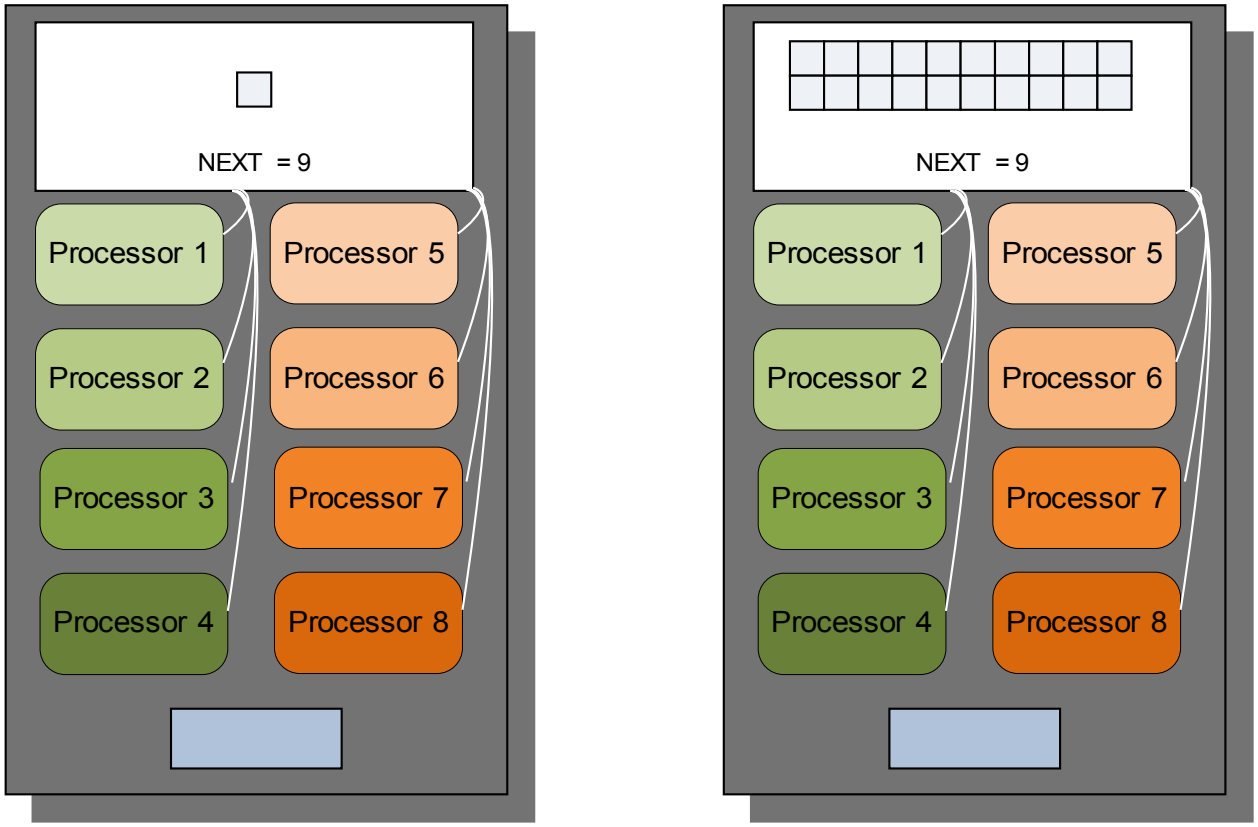
Few shared-memory accesses.

Amdahl's law influence.

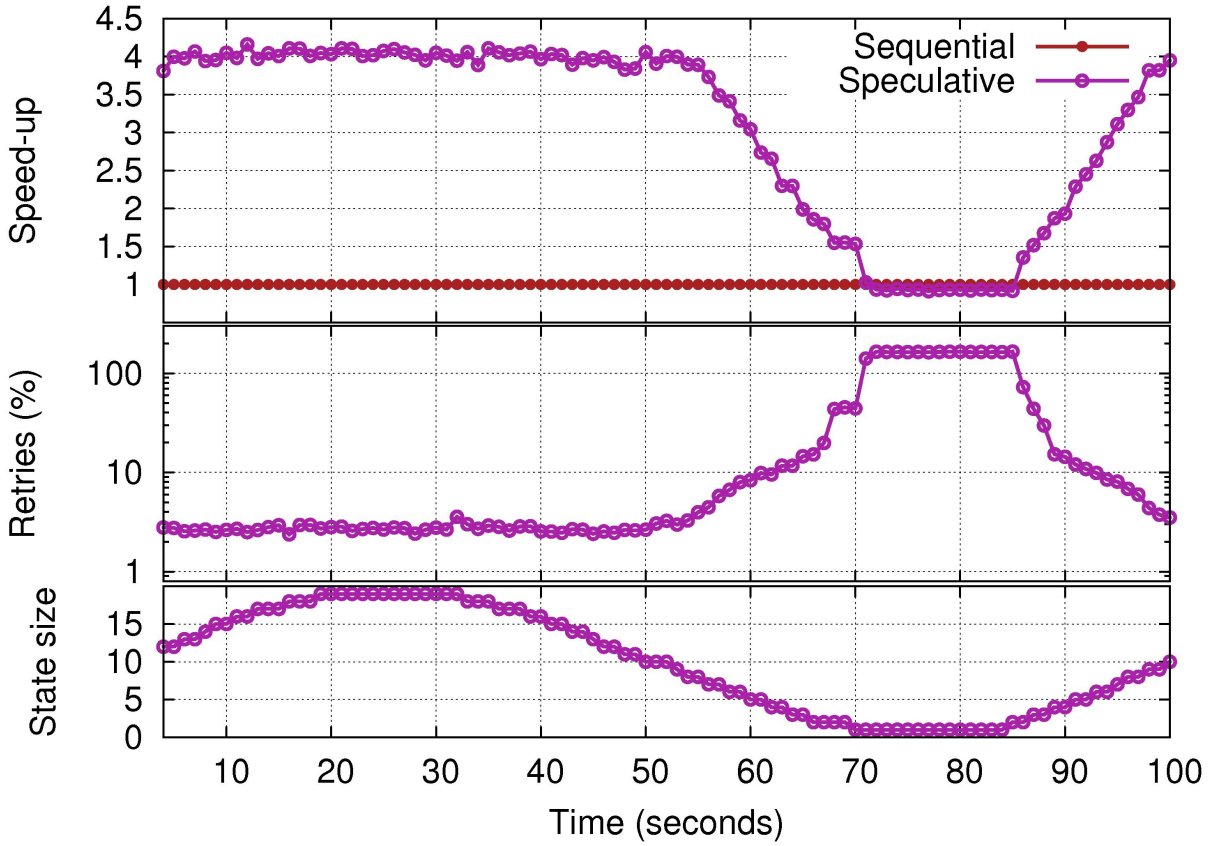
Controlling optimism



Controlling optimism



Controlling optimism



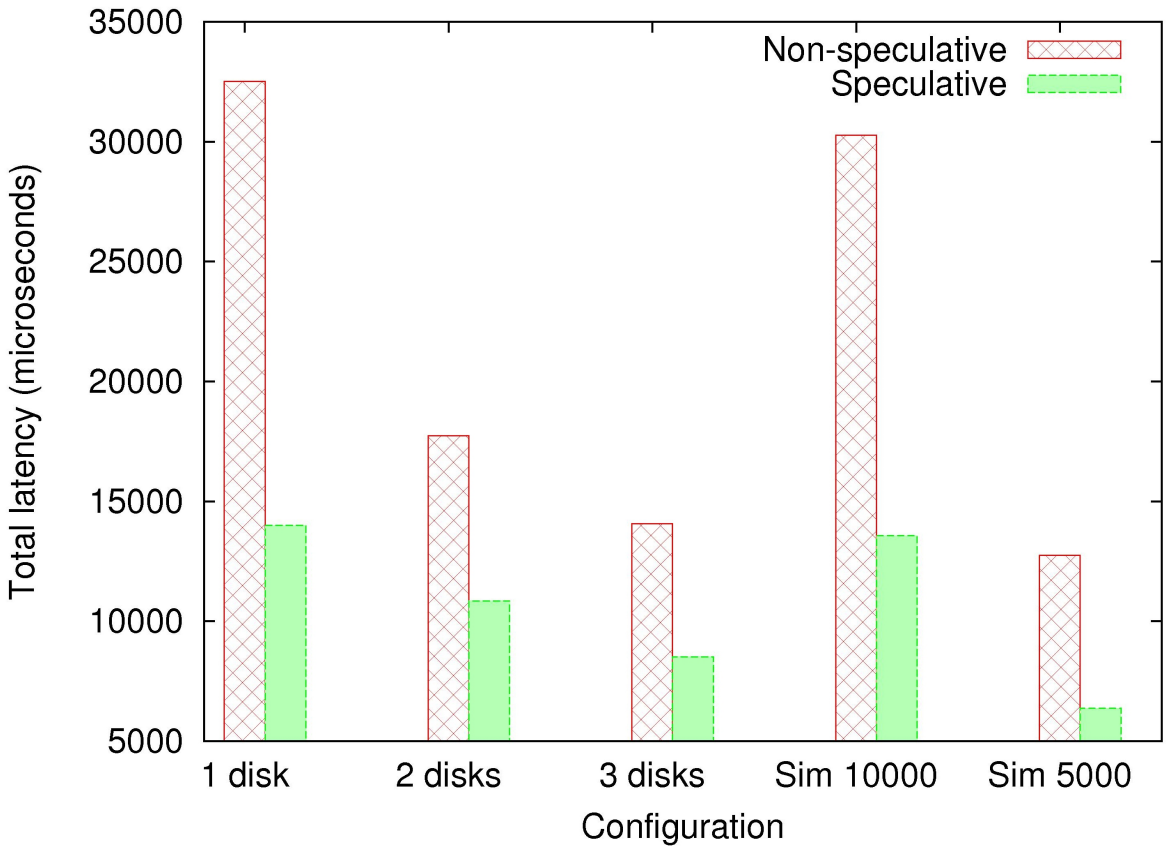
Hardware: SUN T1000

State size varies between 1 and 20.

Size=1: concurrent executions will conflict.

Size=20: considerable parallelism.

Motivation for distributed speculation: logging costs

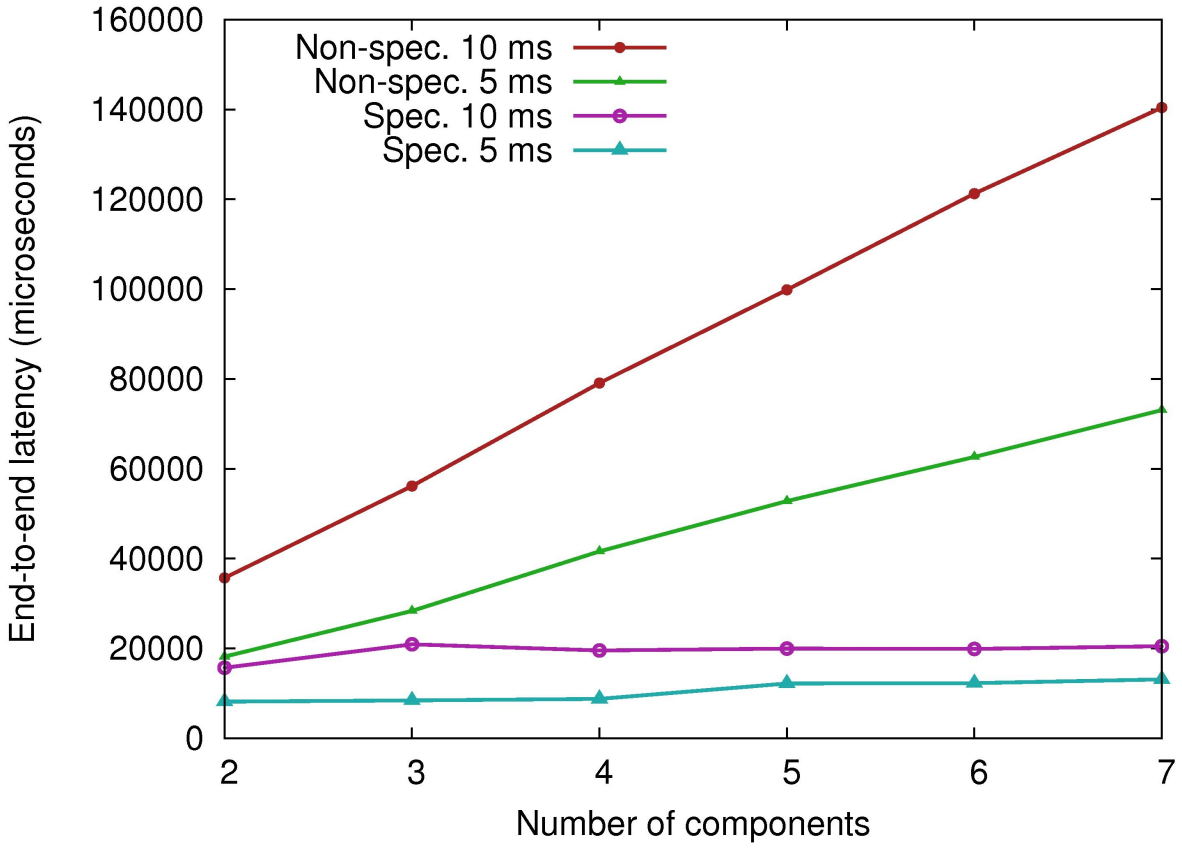


2 components do logging.

Non-speculative: only stable events are sent.

Speculative: send events before logging is finished.

Accumulated gains



X axis: number of components logging.

Even in a SAN/WAN, the shapes would look similar.

For deterministic components: add "fixed" latency.

Final remarks - Parallelization

- Parallelization through speculation
  - Easier, less bugs
    - Programmer does not need to fight with locks
    - Keeps sequential semantics
  - Waste of resources reduced with optimism control
- Overhead can be much lower with hardware support for TM (e.g., ASF)

Final remarks - Logging

- Overlap logging with processing
  - Independent of available parallelism
  - Distributed speculation possible due to less aborts
  - But do not let speculative results get out of the system
- In combination with speculative parallelization may even reduce logging

Thank you!

http://streammine.inf.tu-dresden.de
http://wwwse.inf.tu-dresden.de