Chronos: Efficient Speculative Parallelism for Accelerators

MALEEN ABEYDEERA, DANIEL SANCHEZ

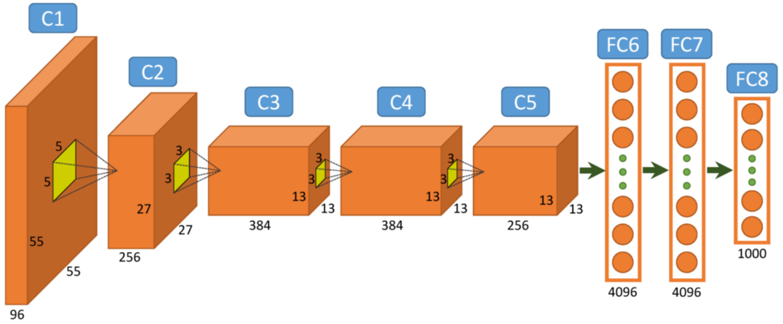
ASPLOS 2020

Current hardware accelerators are limited to easy parallelism

Current Accelerators

Target easy parallelism

Tasks and dependences known in advance

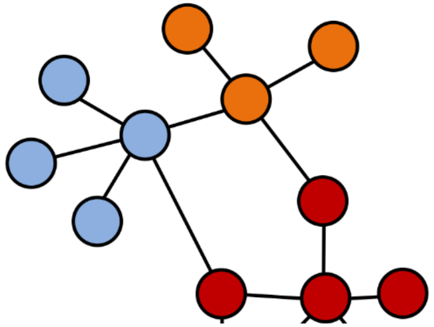


e.g.: Deep learning, Genomics

Chronos

Targets hard parallelism

Require speculative execution



e.g.: Graph analytics, simulation, transactional databases

Problem and Insight

Problem

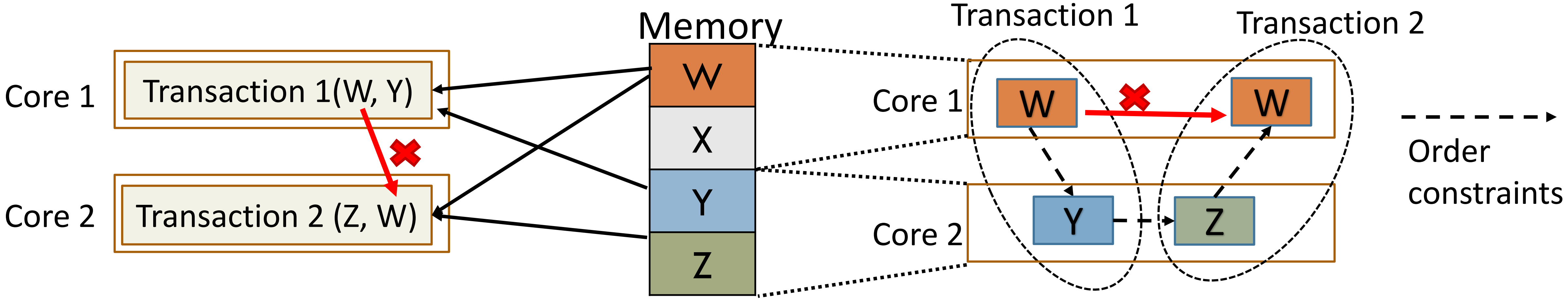
Prior speculation mechanisms (Transactional Memory, Thread Level Speculation) require global conflict detection

Insight

Limit the data that each core can access

Divide work into tiny tasks and send them to data

Coordinate tasks through order constraints



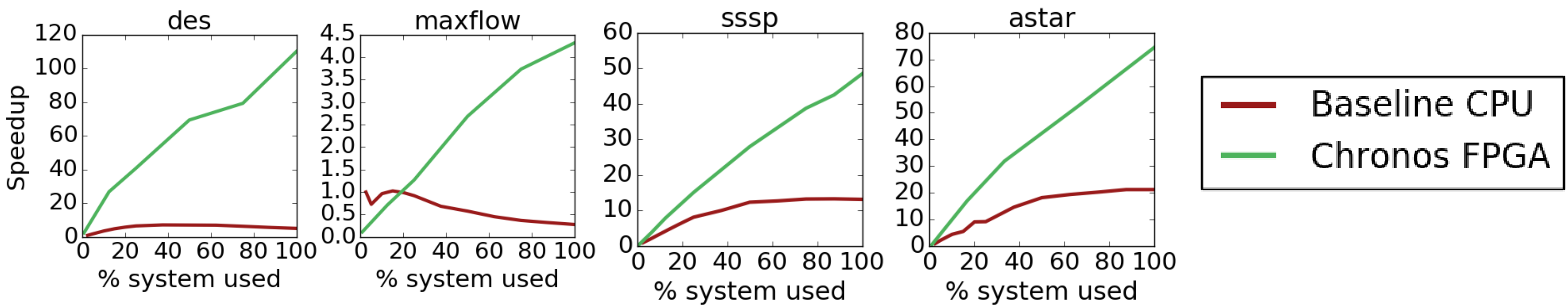
Shared memory system → coherence protocol
Coherence poorly suited for accelerators

Local conflict detection → No coherence needed

Contributions

SLOT (Spatially Located Ordered Tasks): A new execution model that does not require coherence, but relies on task ordering and spatial task mapping to detect conflicts

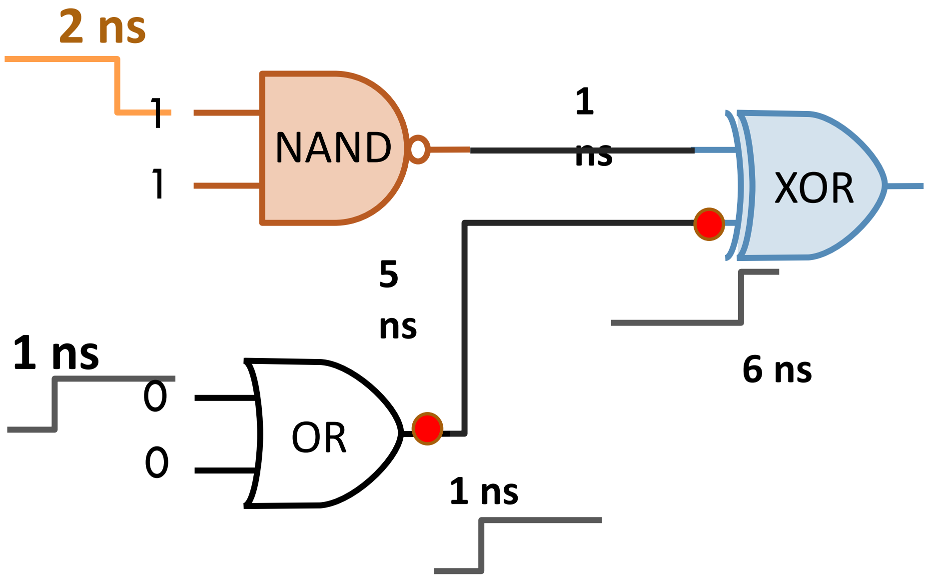
Chronos: An implementation of SLOT that provides a common framework for acceleration of applications with speculative parallelism

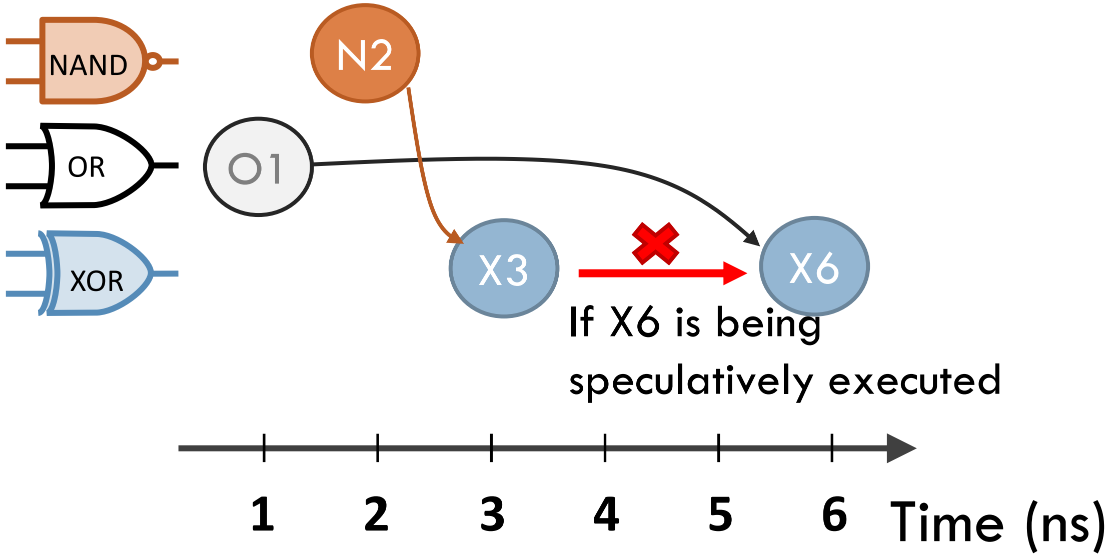


https://chronos-arch.csail.mit.edu/

Speculative parallelism with single-object tasks

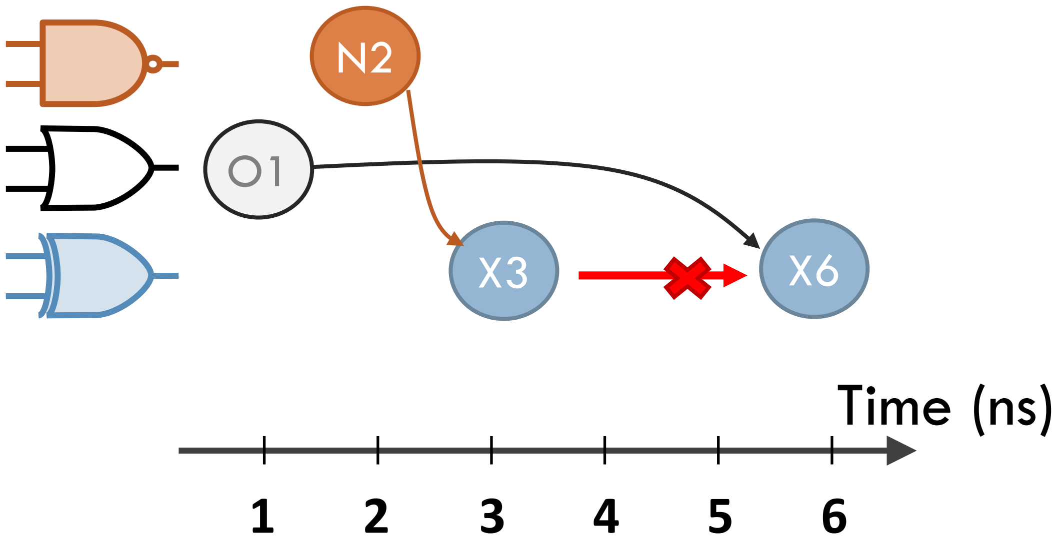
Discrete Event Simulation (DES) for Digital Circuits

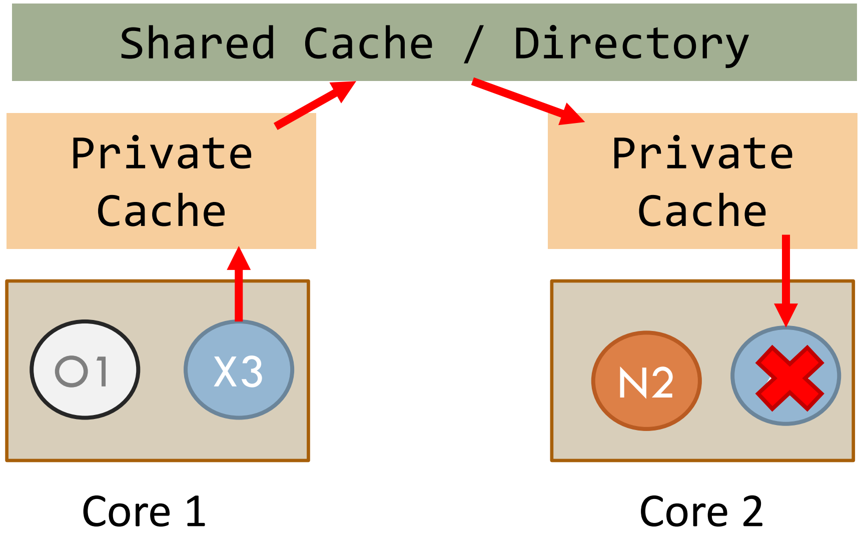




Prior techniques rely on global conflict detection

Why? No restriction on where a task can run

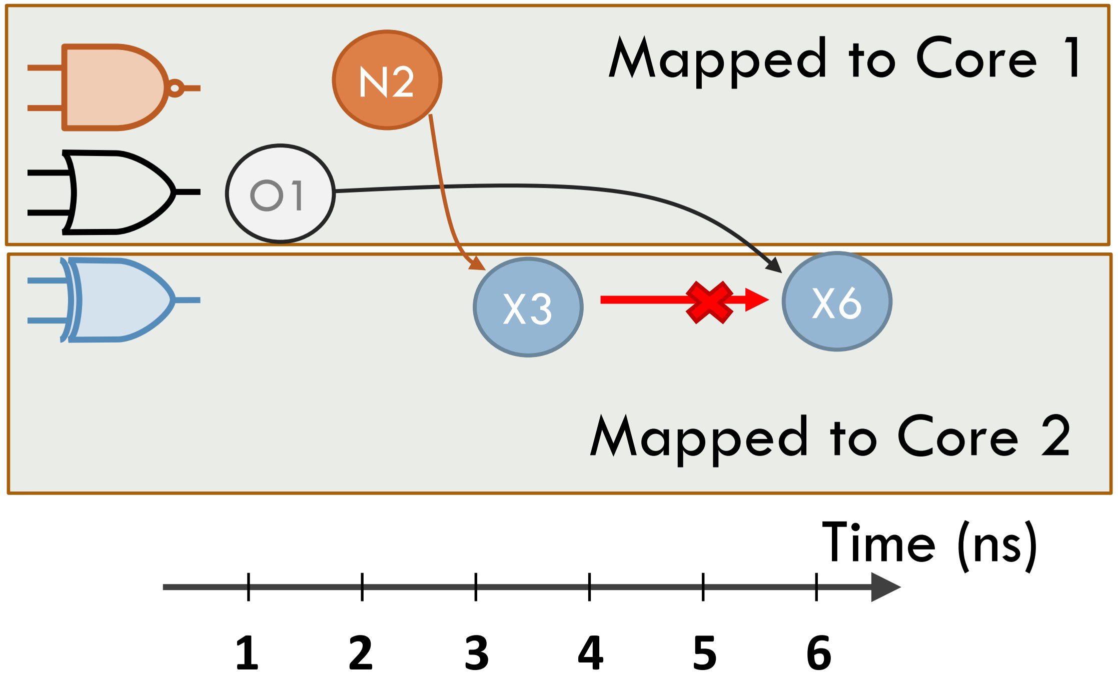


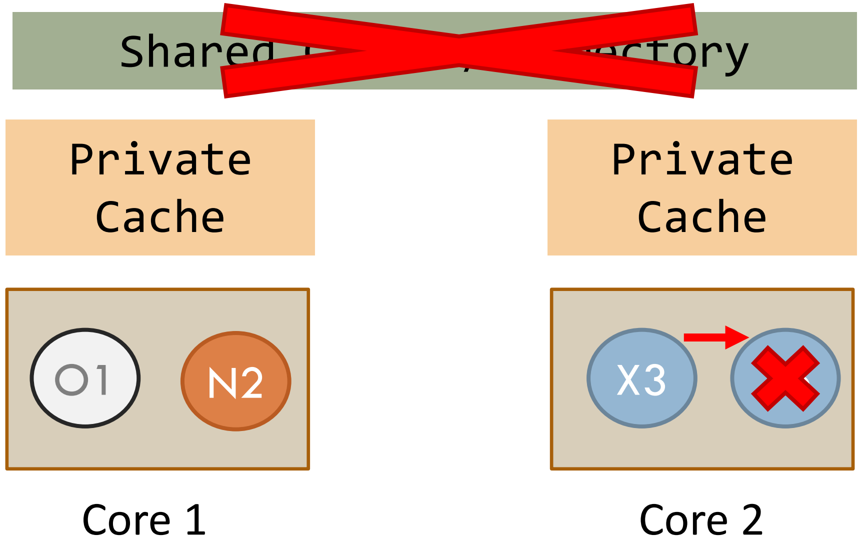


Relies on coherence protocol to find conflicts

Insight 1: Leveraging spatial task mapping for local conflict detection

Impose restrictions on where a task can run



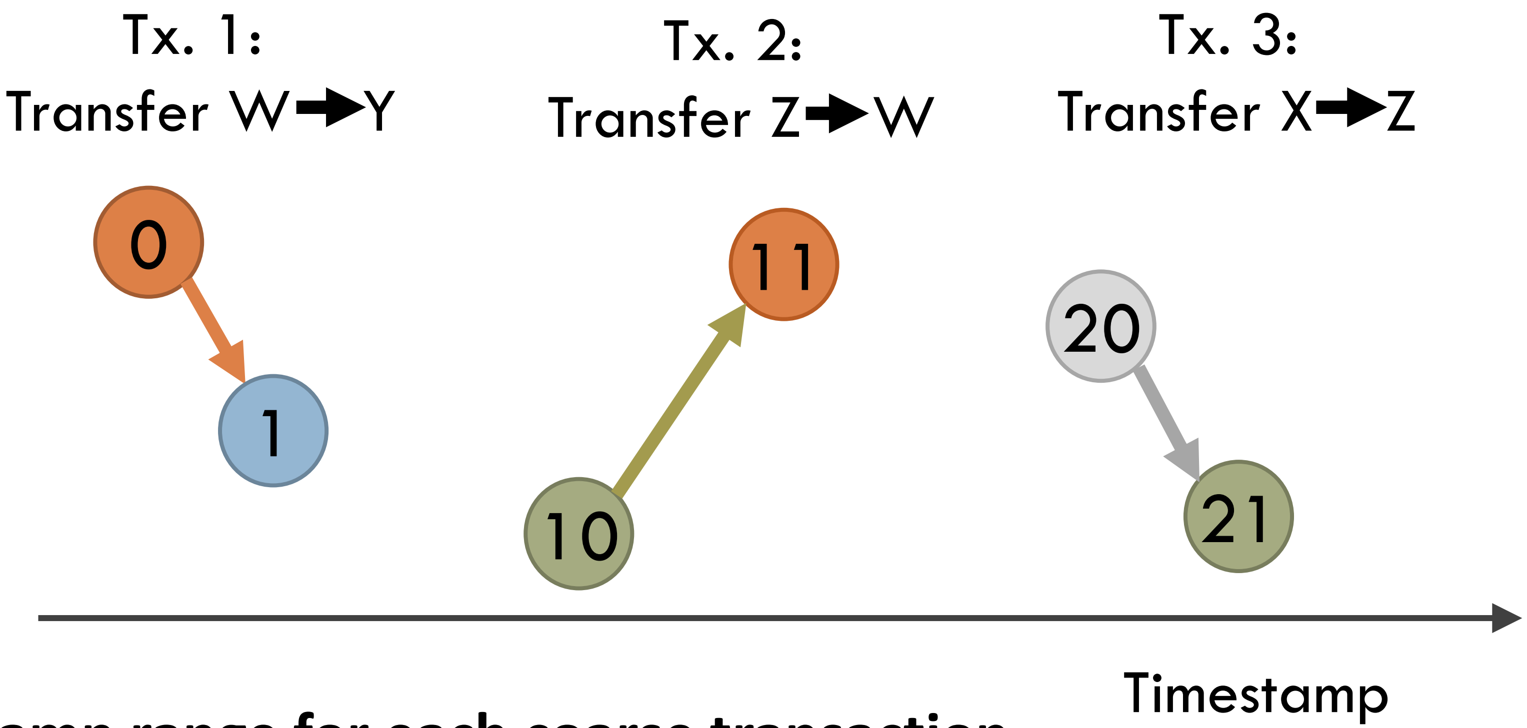


Conflict detection is local to a core

Insight 2: Leveraging order to ensure atomicity

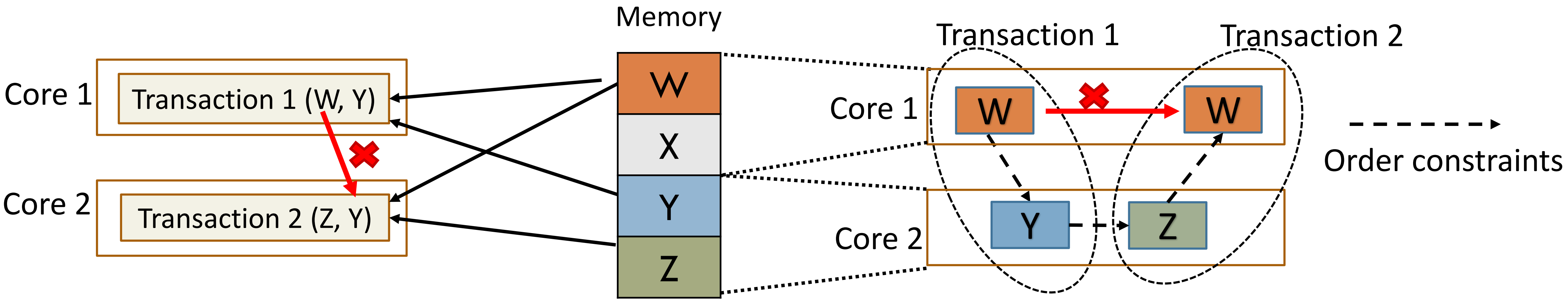
Banking application:
Each transaction decrements the balance of one account and increments another

Account (object)	Balance
W	$100
X	$1500
Y	$200
Z	$400



Assign a disjoint timestamp range for each coarse transaction

Benefits of fine-grained tasks



Brings data to compute

Sends compute to data

- ✓ Increased data locality
- ✓ Reduced network traffic
- ✓ Increased parallelism
- ✓ Low probability and impact of aborts
- ✓ Asynchronous communication

SLOT (Spatially Located Ordered Tasks)

SLOT programs consist of tasks

Tasks can create children tasks through a simple API:

slot::enqueue(*fn_ptr*, *timestamp*, *object-id*, *arguments*…);

Timestamp : Specifies order. Tasks appear to execute in timestamp order

Object-id : Specifies dependences. Tasks with same object-id are treated as data-dependent

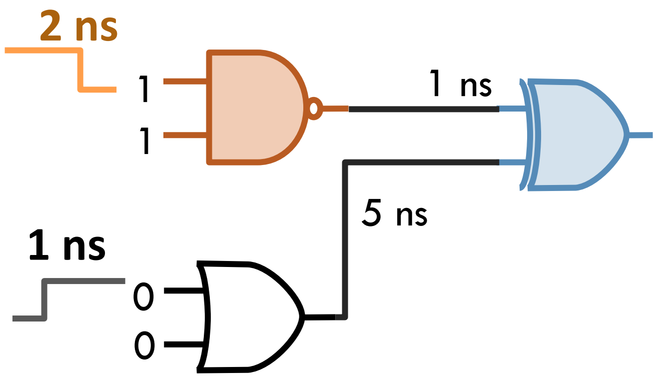
Tasks with different object-ids can only communicate through *arguments*

SLOT programming example (in software)

```
// Simulates an event arriving at a gate
void simToggle(Time time, GateInput input) {
    gate = input.gate;
    toggledOutput = updateState(gate, input);
    if (toggledOutput) {
       // create events for connected gates
       for (GateInput i : gate.connectedInputs()) {
         Time nextTime = time + gate.delay(input, i);
         eventQueue.enqueue(nextTime, i);
       }
    }
}

PriorityQueue<Time, GateInput> eventQueue;
enqueueInitialEvents()
// event loop. Sequentially execute in ts order
while (!eventQueue.empty()){
    (time, input) = eventQueue.dequeue();
    simToggle(time, input);
}
```

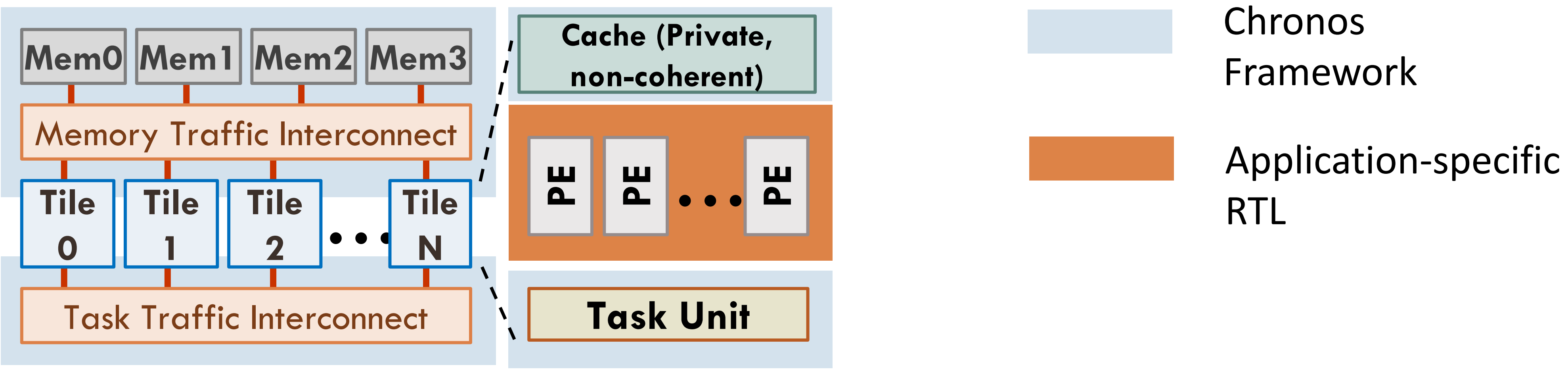
```
// Simulates an event arriving at a gate
void simToggle(Time time, GateInput input) {
    gate = input.gate;
    toggledOutput = updateState(gate, input);
    if (toggledOutput) {
       // create events for connected gates
       for (GateInput i : gate.connectedInputs()) {
          Time nextTime = time + gate.delay(input, i);
          slot::enqueue(
                simToggle, nextTime, i.gateID, i);
       }
    }
}
enqueueInitialTasks()
slot::run()
```



Chronos:
An implementation of SLOT

Chronos overview

Chronos provides a framework to build accelerators for applications with speculative parallelism

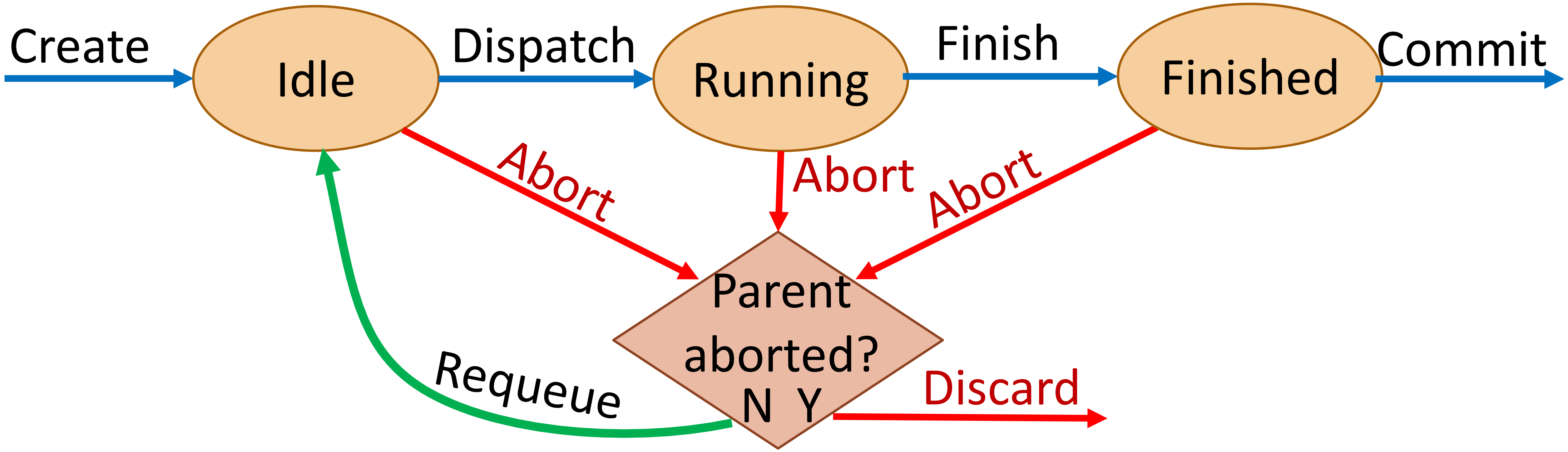


The developer specifies the tasks and how they are implemented

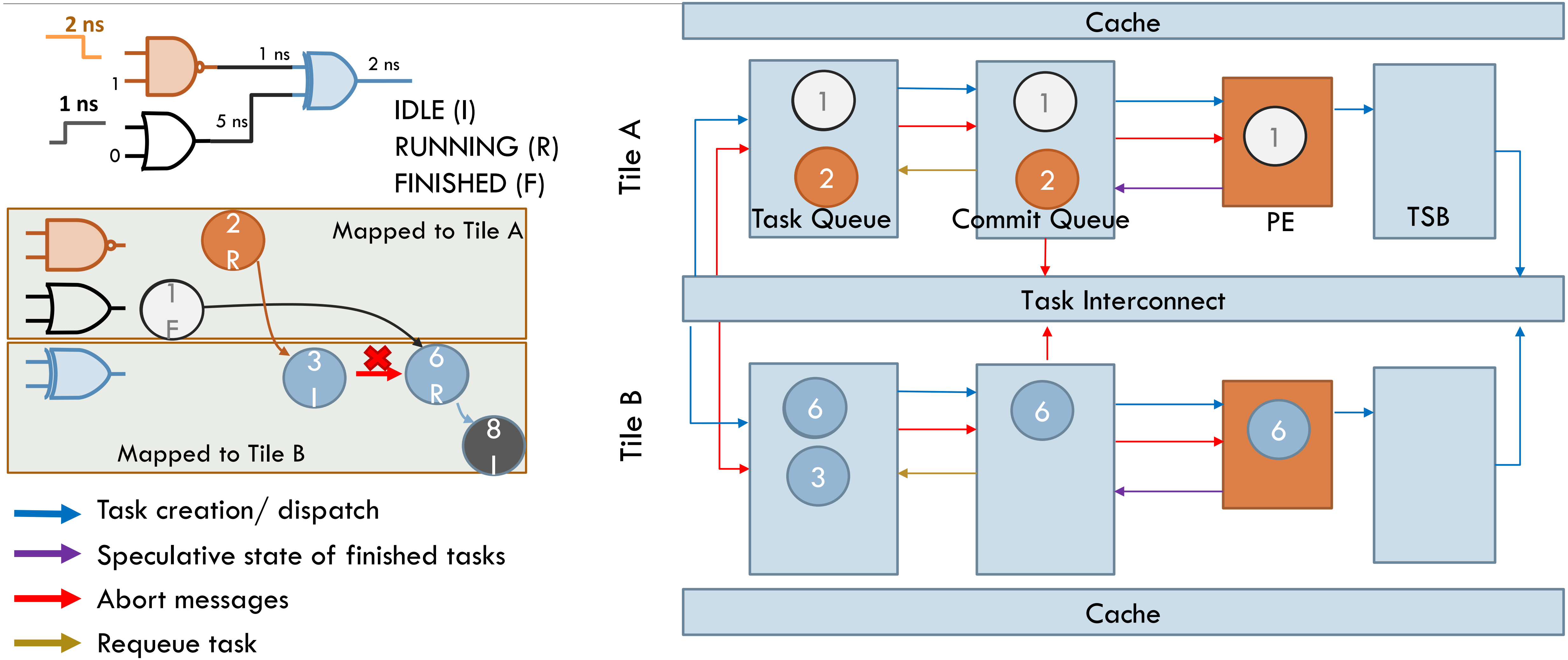
- Either software routines on soft cores, or specialized Processing Elements (PE)

Framework takes care of task management and speculative execution

Task life cycle



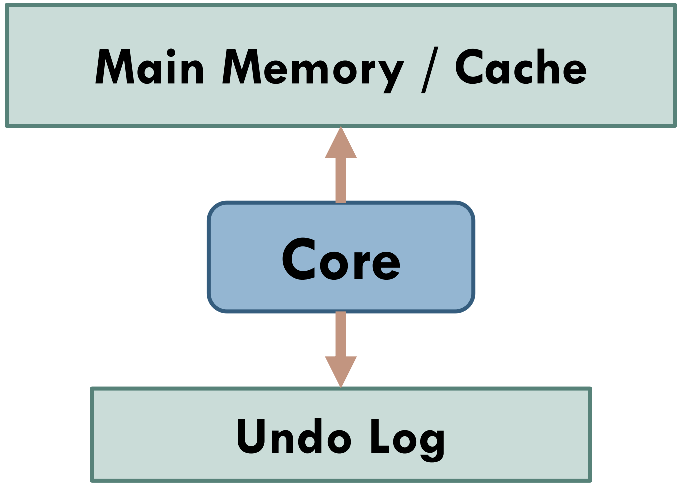
Chronos internal dataflow



Versioning and commit protocol

Eager versioning

Updates speculative values in place



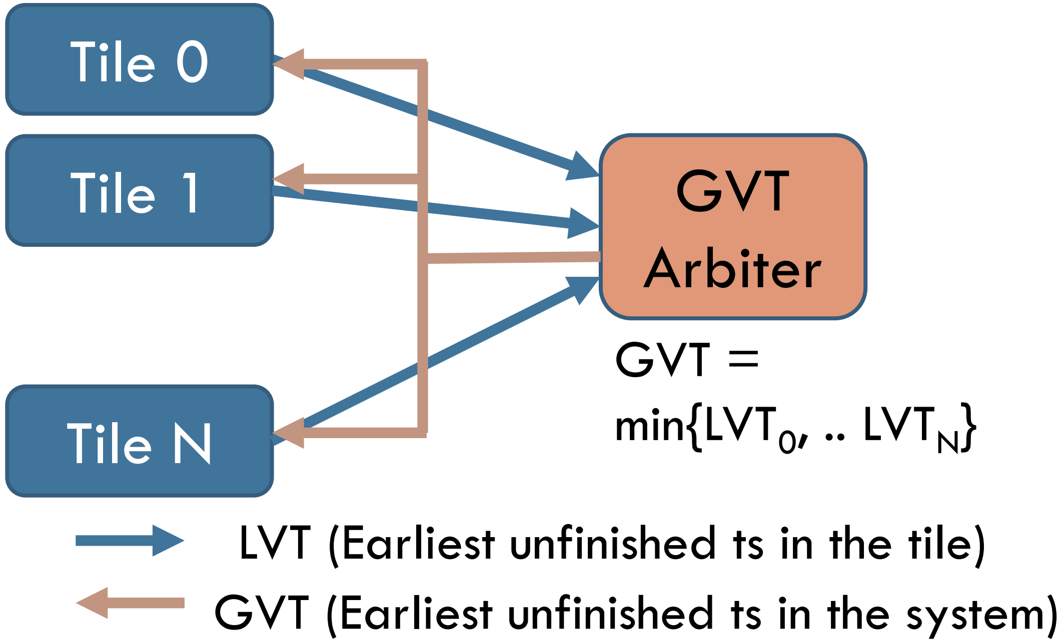
Store old values in an undo log

Key benefits

Makes the common case (commits) fast

Makes speculative data available before commit

Commit Protocol (GVT – Global Virtual Time)



Key benefits

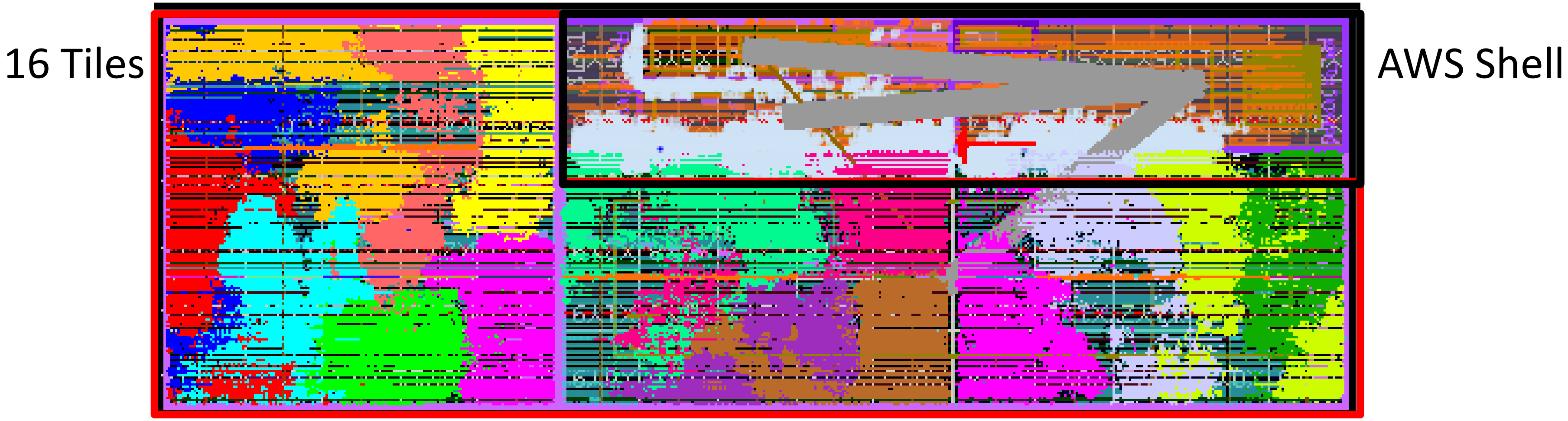
Achieves fast and parallel commits

Chronos FPGA implementation

Developed an FPGA implementation of Chronos – up to 16 tiles

Running at 125 MHz

High task throughput – can enqueue, dequeue, execute and commit 8 tasks per cycle on a 16-tile system



Experimental methodology

Four accelerators built using Chronos framework running on AWS FPGAs

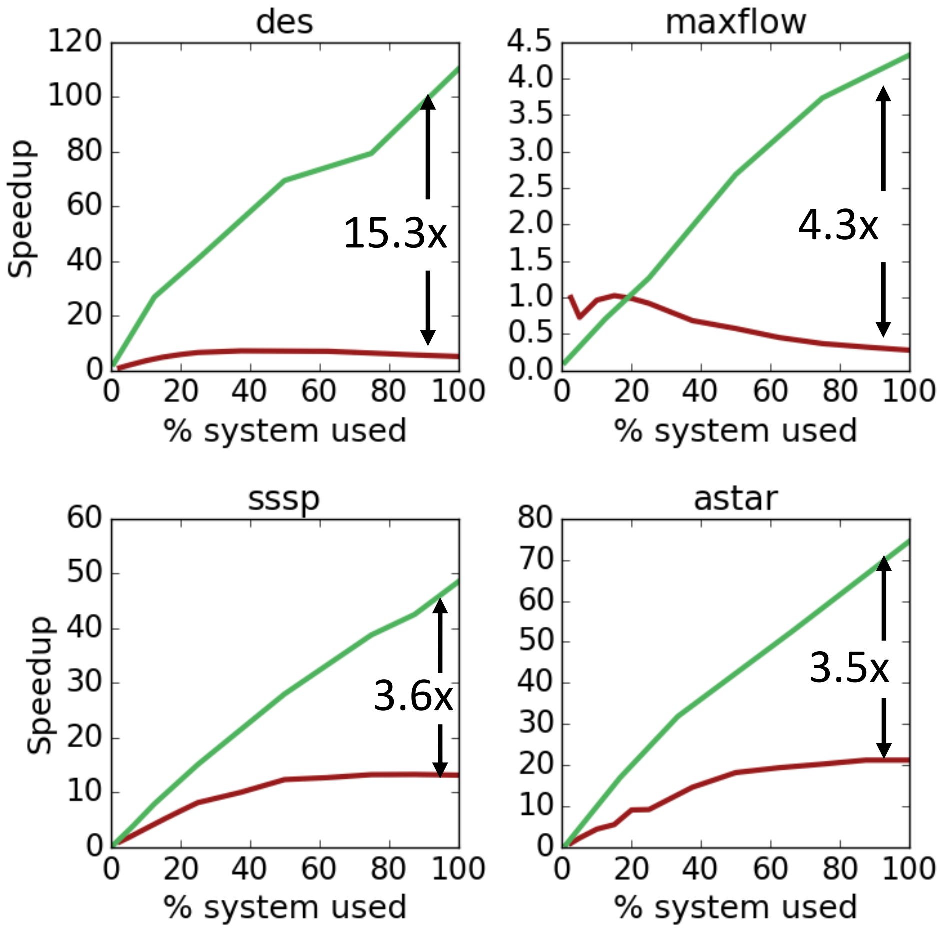
- Discrete Event Simulation (DES)
- Maxflow
- Single Source Shortest Paths (SSSP)
- Astar Search

Platform	AWS Instance	Price ($/hr)
Baseline CPU	M4.10xlarge	2.00
FPGA	F1.2xlarge	1.65

Custom PEs per application: 32-way multithreaded PE, single PE/tile

Baseline: Highly optimized software parallel implementations running on a 40-threaded Xeon AWS instance

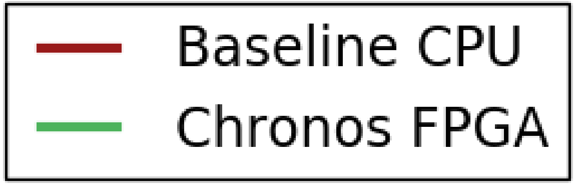
Chronos performance vs. 40-threaded Xeon



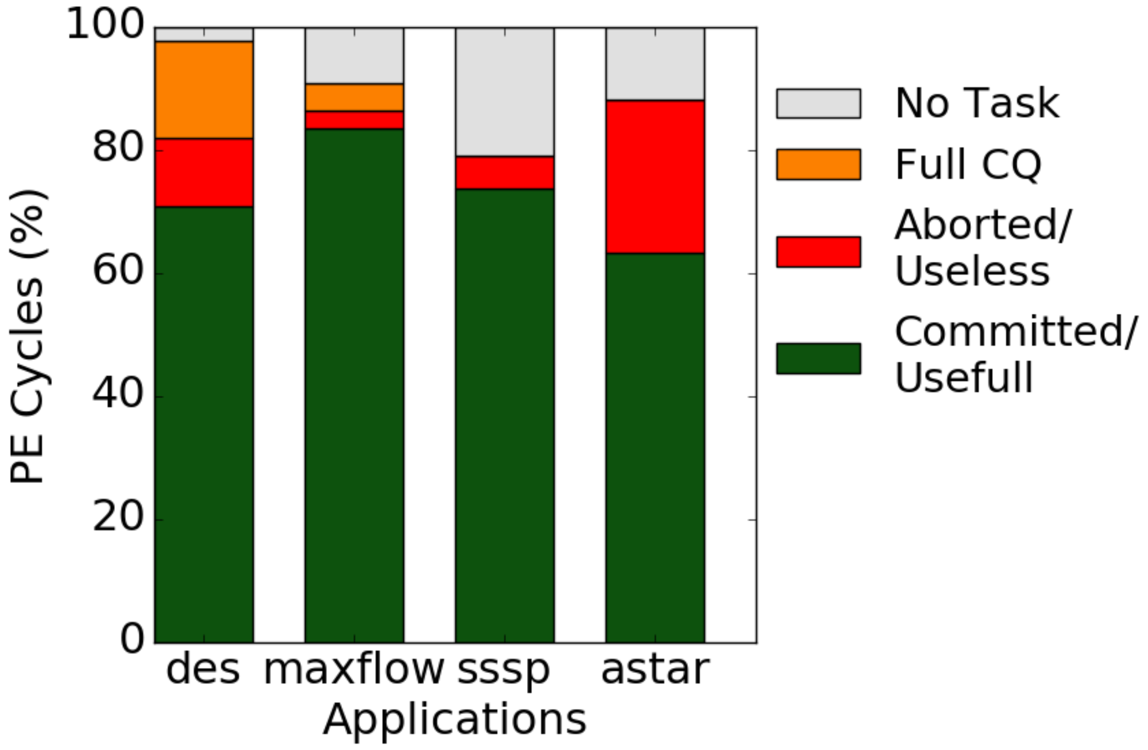
App	Concurrent Max. Tasks	FPGA 1t/ CPU 1t	Overall Speedup
des	256	2.45×	15.3×
maxflow	192	0.11×	4.3×
sssp	512	0.24×	3.6×
astar	192	0.58×	3.5×

Runs many more tasks in parallel

Specialization helps to run a single task efficiently (narrowing the 19× frequency gap with CPU)



Chronos performance analysis



Breakdown of aggregate PE cycles

Observation:

Most work is ultimately useful
(only 11% of cycles result in wasted work)

See the paper for more

Non-speculative applications

Non-rollback applications

Chronos with RISC-V cores

Projected performance on ASIC Chronos

Chronos resource utilization

Conclusion

Prior speculative parallel systems have relied on cache coherence to detect conflicts, precluding their use in accelerators

SLOT (Spatially Located Ordered Tasks): A new execution model that does not require coherence, but relies on task ordering and spatial task mapping to detect conflicts

Chronos: An implementation of SLOT that provides a common framework for acceleration of applications with speculative parallelism

- Use Chronos to build FPGA accelerators for four challenging applications providing up to 15x speedup over a multicore baseline

https://chronos-arch.csail.mit.edu/