Swarm: A Scalable Architecture for Ordered Parallelism

Mark C. Jeffrey, Suvinay Subramanian, Cong Yan, Joel Emer, Daniel Sanchez

{mcj, suvinay, congy, emer, sanchez}@csail.mit.edu http://bit.ly/swarmarch

1. Abstract

Current multicores **target easy parallelism**

- Coarse tasks with infrequent synchronization

Many applications are ignored

- Fine-grained tasks cause large SW overhead
- Limited support for complex synchronization

We target *ordered irregular* applications

- common, but **taxing to parallelize**
- comprise tasks that must follow some order
- tasks dynamically created

Swarm extracts **orders-of-magnitude of parallelism**

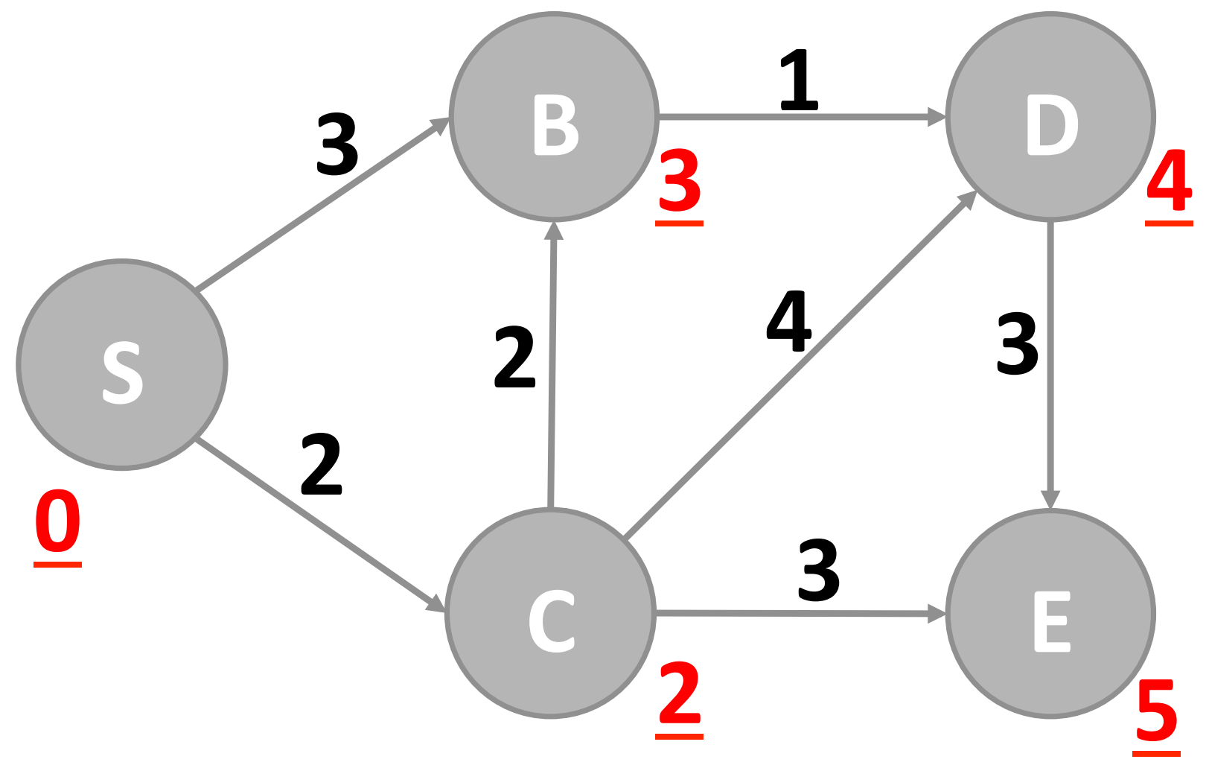
- a new execution model and microarchitecture
- enables efficient management of speculative tasks

2. Example: Parallelism in Dijkstra

Finds the shortest-path tree on a weighted graph

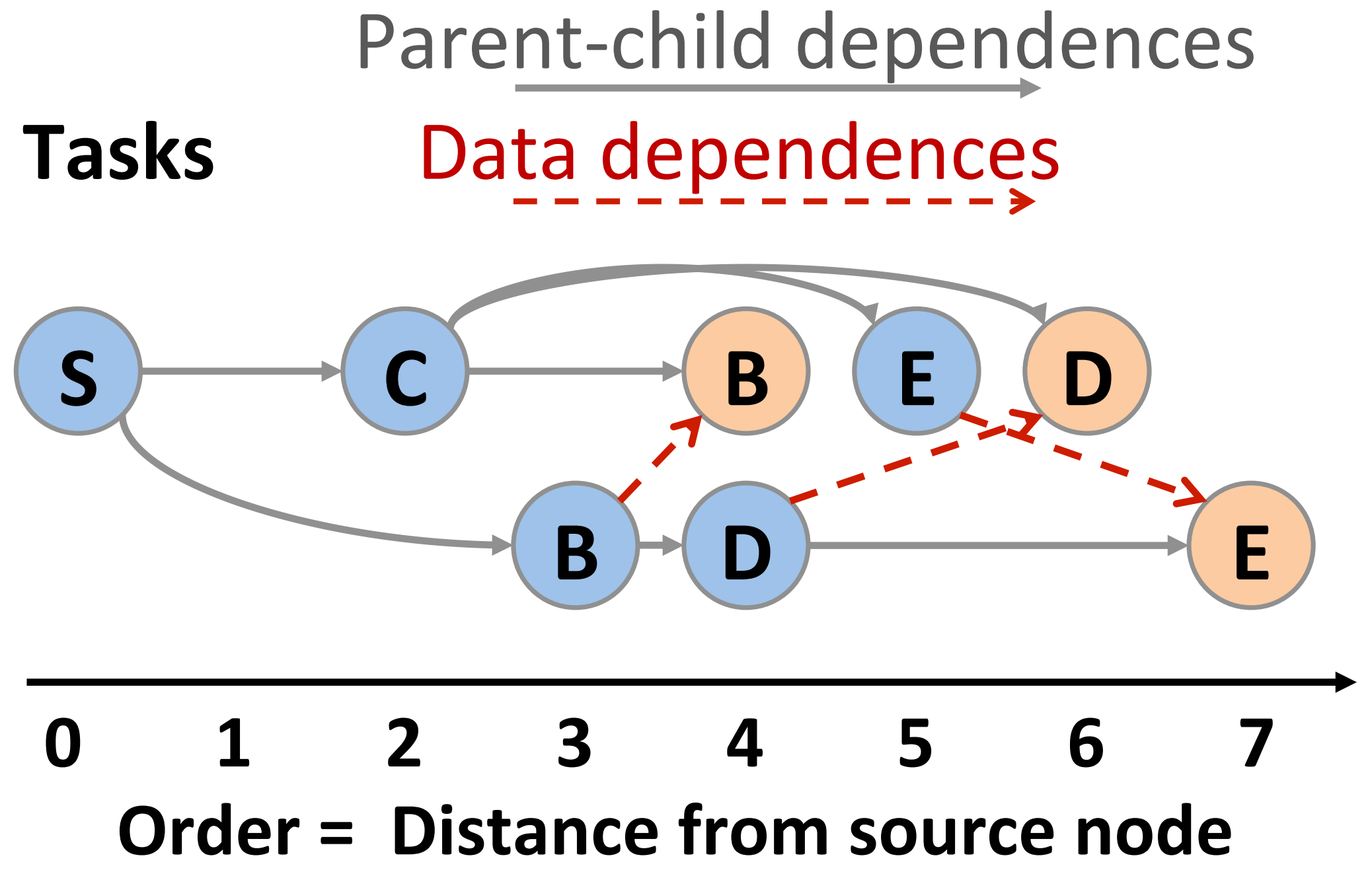
Each task:

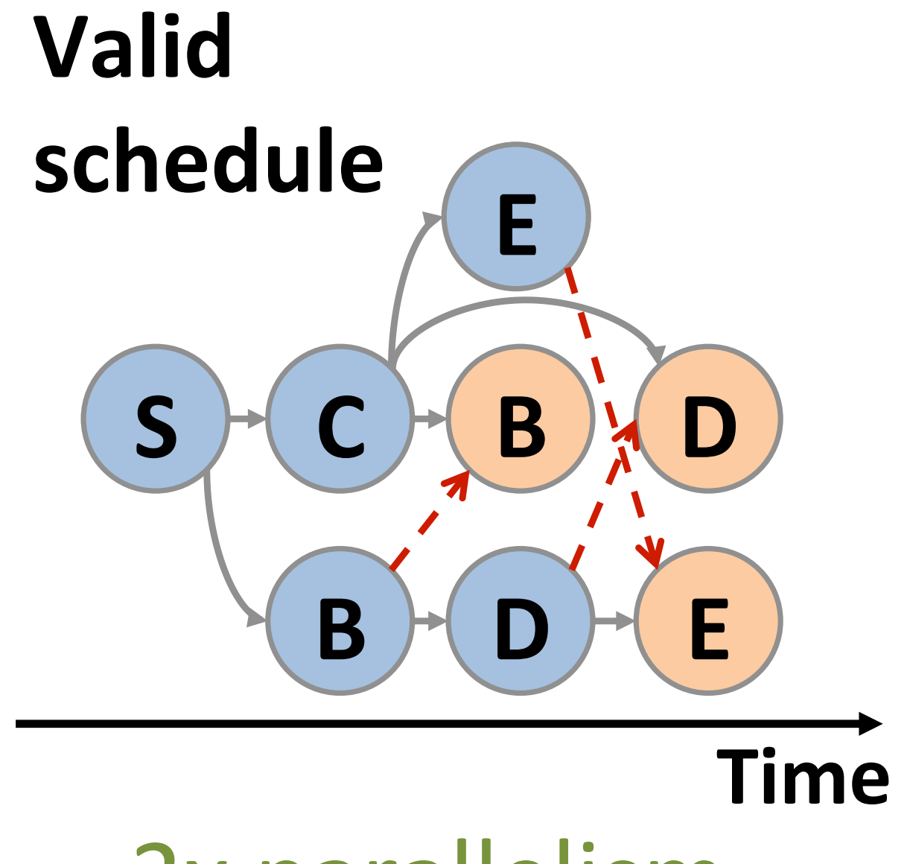
- operates on one node
- is ordered by distance



```
prioQueue.enqueue(source, 0)
while prioQueue not empty:
 (node, dist) =
prioQueue.dequeueMin()
 if node.distance not set:
  node.distance = dist
  for child in node.children:
   d = dist + distance(node, child)
   prioQueue.enqueue(child, d)
 else: // node already visited, skip
```

1. Task creation order ≠ execution order
2. Speculation that elides ordering constraints can uncover parallelism





2x parallelism

3. Parallelism Limit Study

Benchmark	Max. parallelism	Instrs. per task	Parallelism Window 64	Parallelism Window 1024
bfs (graph)	3440x	22	58x	827x
sssp	793x	32	26x	178x
astar	419x	195	16x	62x
msf	158x	40	49x	147x
des (sim.)	1440x	296	32x	198x
silo (DB)	318x	1969	17x	125x

1. With perfect speculation, parallelism is plentiful
2. Tasks are tiny
3. Independent tasks are far away in program order

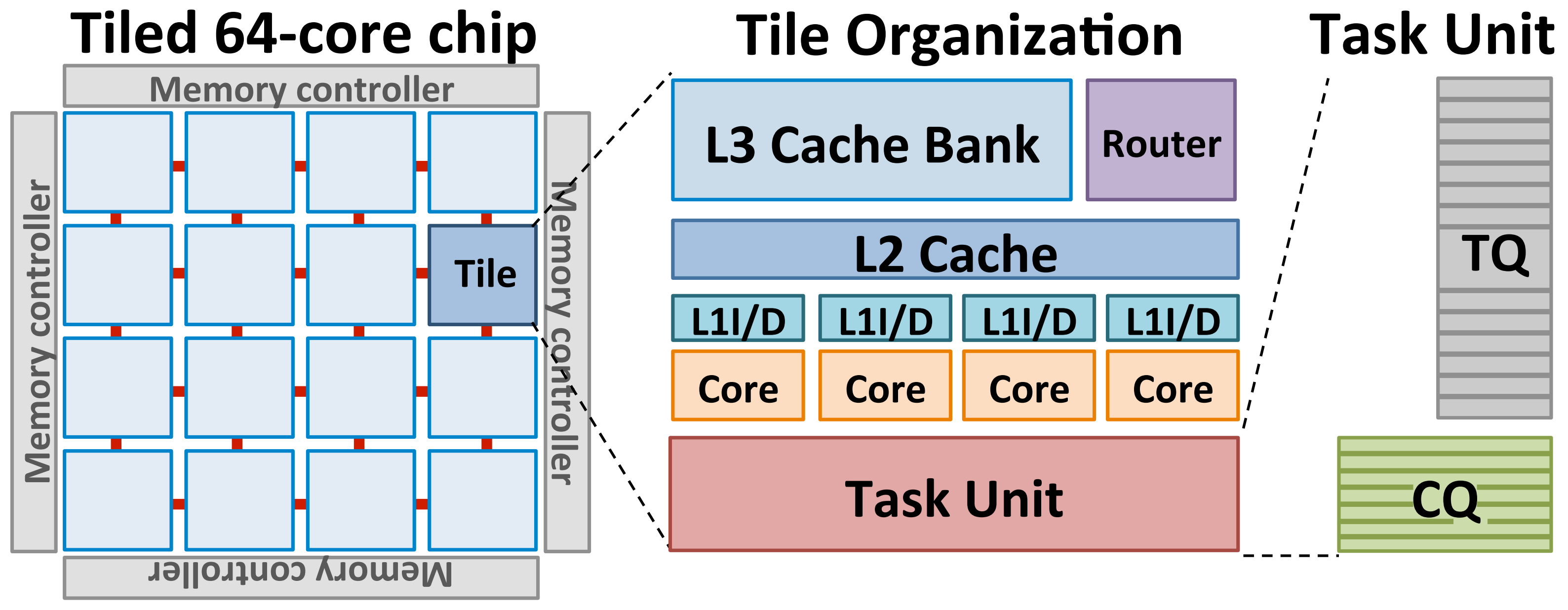
4. Swarm: Execution Model

- Programs comprise timestamped tasks
- Tasks can create children with >= timestamp (TS)
- Tasks appear to execute in timestamp order

```
swarm::enqueue(fptr, ts, args...);
```

Conveys new work as soon as possible

5. Swarm: Architecture



Task Queue: holds all task descriptors

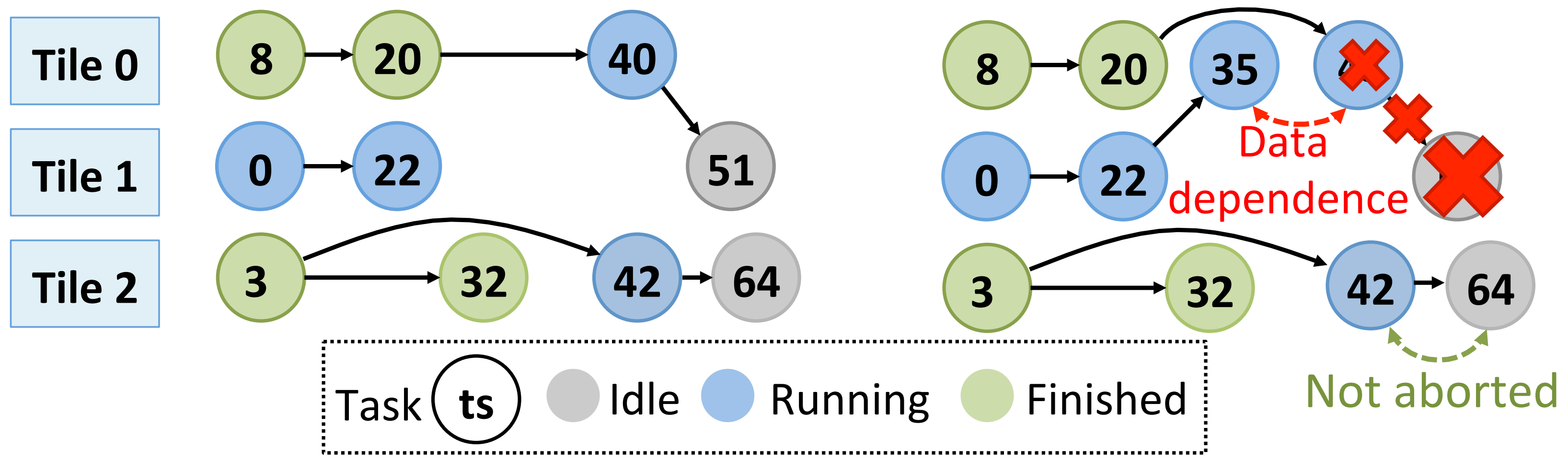
Commit Queue: holds speculative tasks' state

Eager versioning & TS-based conflict detection

- enables forwarding of still-speculative data
- uses memory hierarchy to filter conflict checks
- Bloom filters store read/write sets per task

Large queues enable a huge speculation window

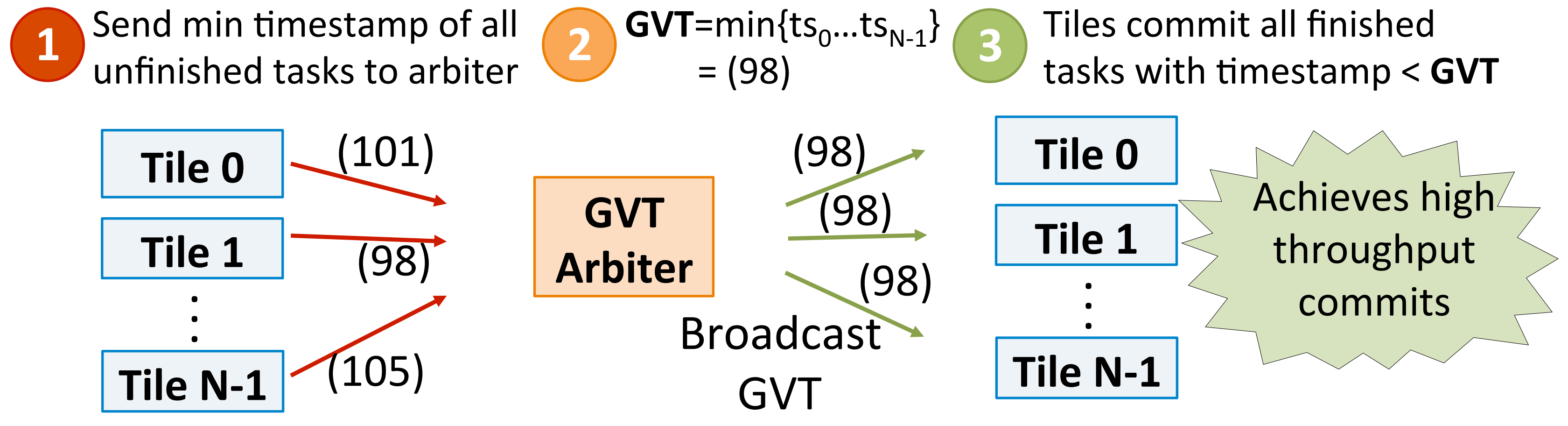
6. Selective Aborts



- Rollback reuses conflict-detection mechanism
- Dependences found via rollback's corrective writes

Avoids explicitly tracking inter-task dependences

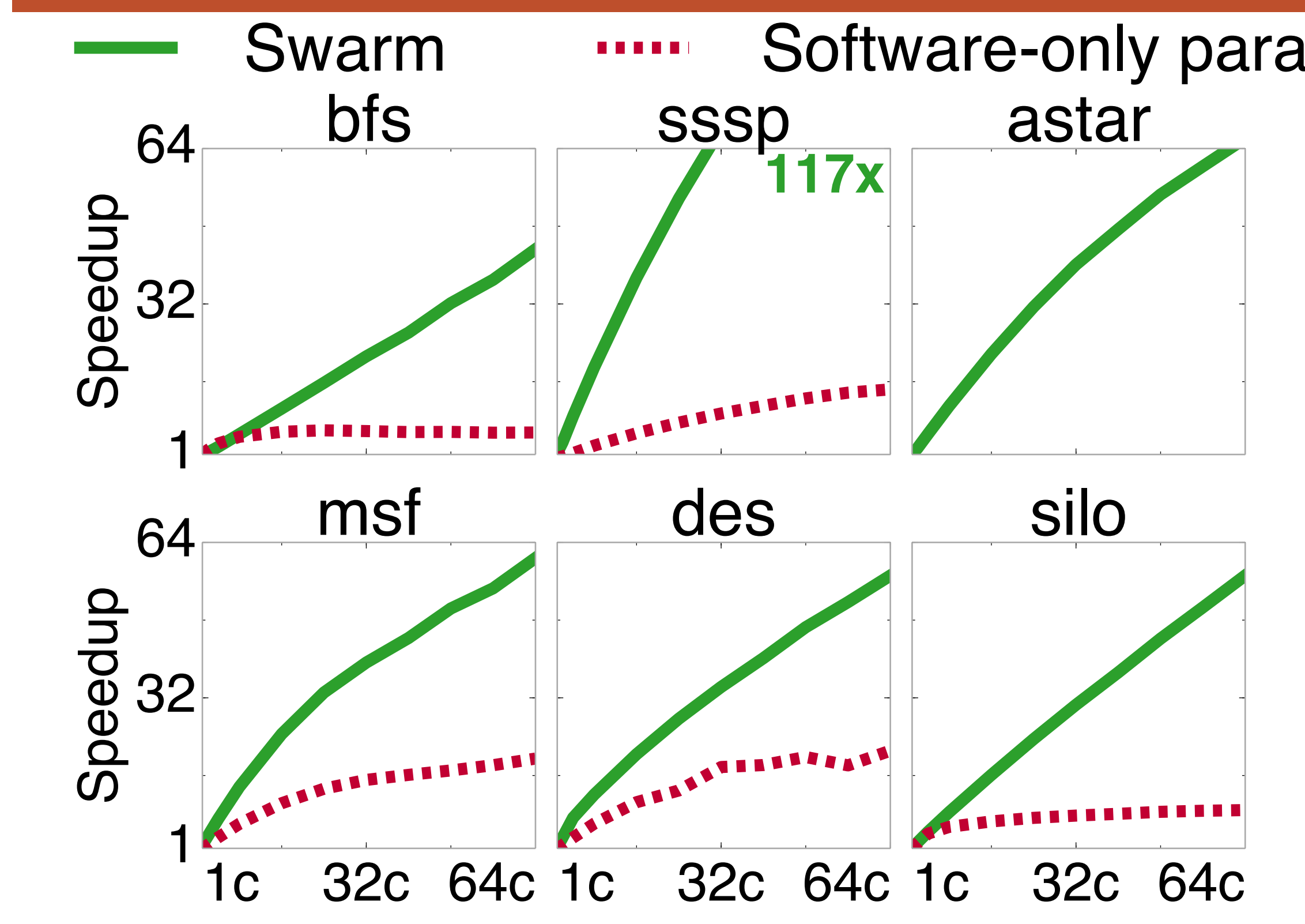
7. Distributed Commits


1
Send min timestamp of all unfinished tasks to arbiter
2
GVT=min{ts0…tsN-1} = (98)
3
Tiles commit all finished tasks with timestamp < GVT
Tile 0
Tile 1
Tile N-1
(101)
(98)
(105)
GVT Arbiter
(98)
(98)
(98)
Broadcast GVT
Tile 0
Tile 1
Tile N-1
Achieves high throughput commits

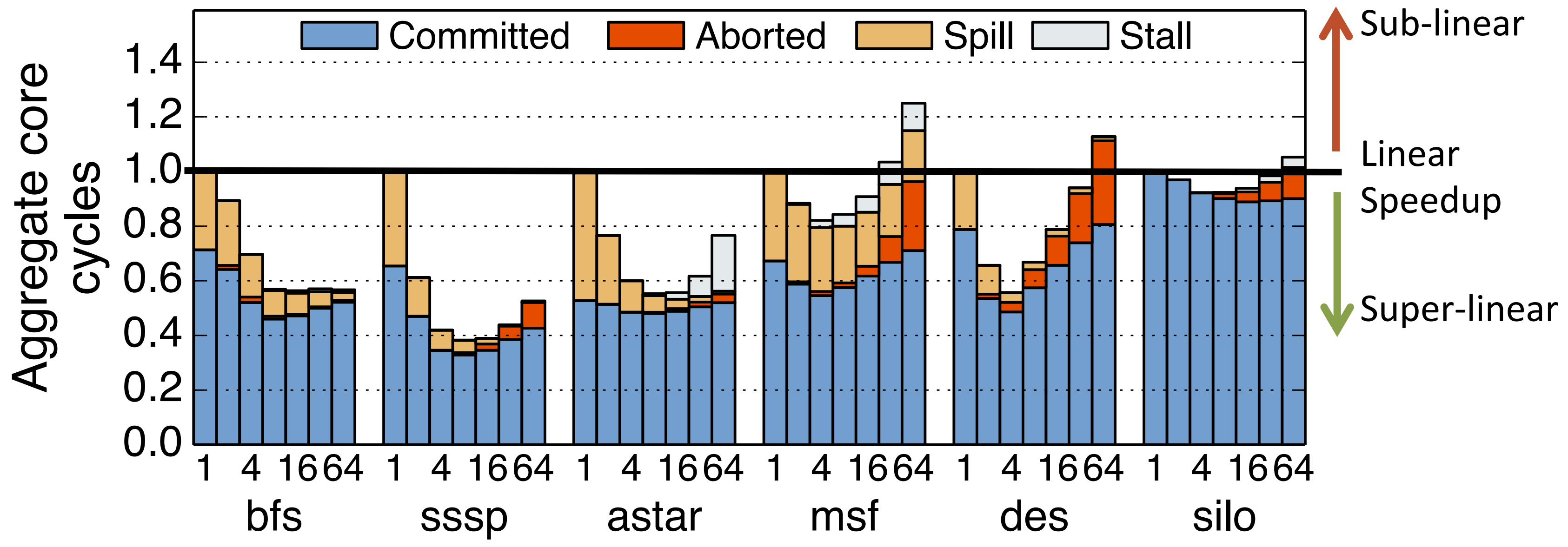
- Jefferson's 1985 "Virtual Time" used to retire tasks

Commit costs are amortized over many tasks

8. Evaluation



- Serial-relative speedups: 43x –117x
- Outperforms state-of-the-art parallel software by 3x – 18x


Committed
Aborted
Spill
Stall
Aggregate core cycles
bfs
sssp
astar
msf
des
silo
Sub-linear
Linear Speedup
Super-linear