TAPAS: Generating Parallel Accelerators from Parallel Programs

Steven Margerm[1], **Amirali Sharifian[1]**, Apala Guha[1]
Gilles Pokam[2], Arrvindh Shriraman[1]

https://github.com/sfu-arch/tapas

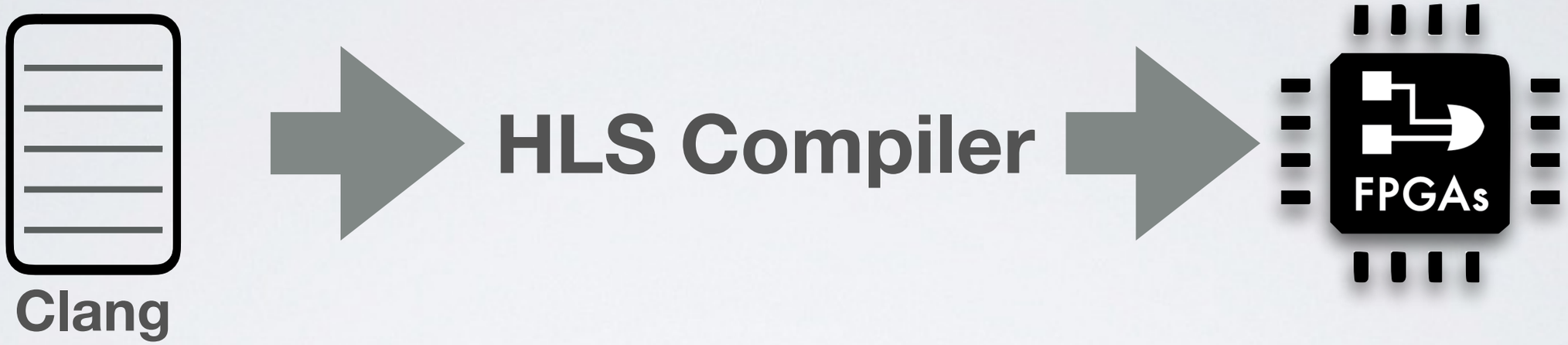
Simon Fraser University[1], Intel Corp.[2]

Motivation

FPGAs are everywhere

- Lots parallelism
  - 150$ Cyclone V SoC - 60 stencil tasks
- 10s of cycles for invoking a hardware “task”
- Fine-grain parallelism
  - Cyclone V. 512 arithmetic ops

High Level Synthesis



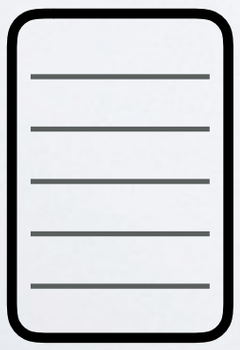
- Mixes schedule and algorithm
  - #pragma
- Static schedule
  - limited concurrency control
- Domain specific templates
  - generalizable ?

TAPAS: Auto generating Parallel Dataflow Accelerator

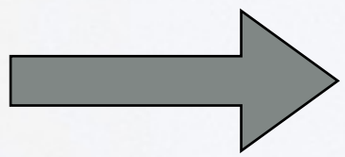
- **MIT's parallel compiler(TAPIR)**

- **Hardware component library:**
  - like UCB Rocket, but for accelerators

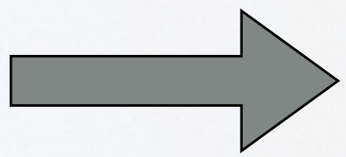
- **Generator**
  - synthesizing RTL from compiler IR



Cilk/Go/OpenMP



TAPAS



Parallel Accelerator

Overview

- HLS Challenge: Static Parallelism

- TAPAS : modular high level synthesis

- TAPAS: generating task units

HLS Challenge: Static Parallelism

```
for(i = 0 until n){
  if(node[i].valid){
    compute(&node[i]);
  }
}
```

Loop bound is unknown

Conditional Body

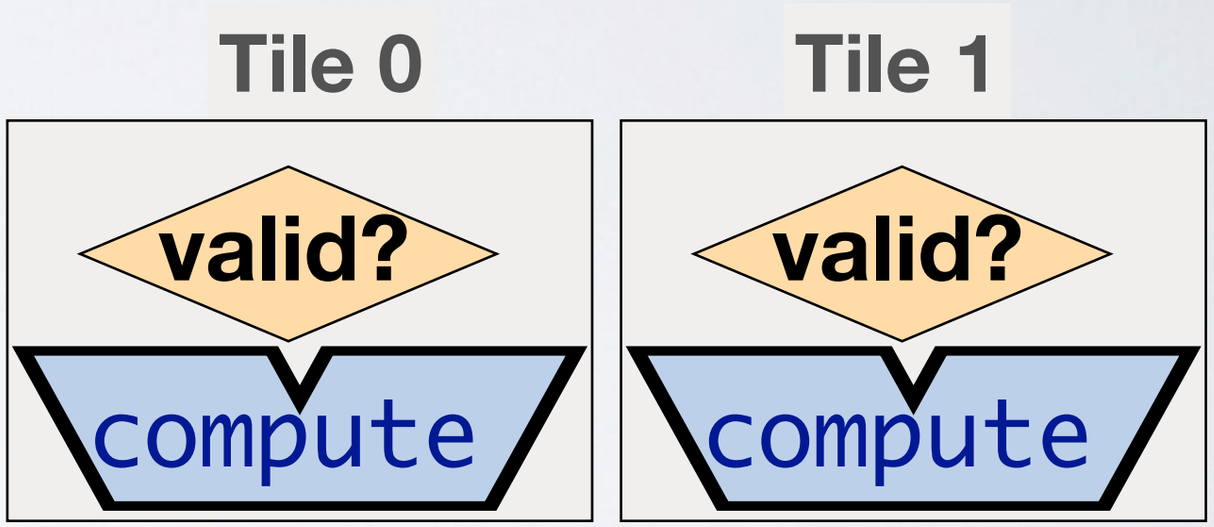
Non-deterministic latency

HLS Challenge: Static Parallelism

Unrolled Program

```
#pragma UNROLL 2
for(i = 0 until n){
  if(node[i].valid){
    compute(&node[i]);
  }
}
```

Hardware



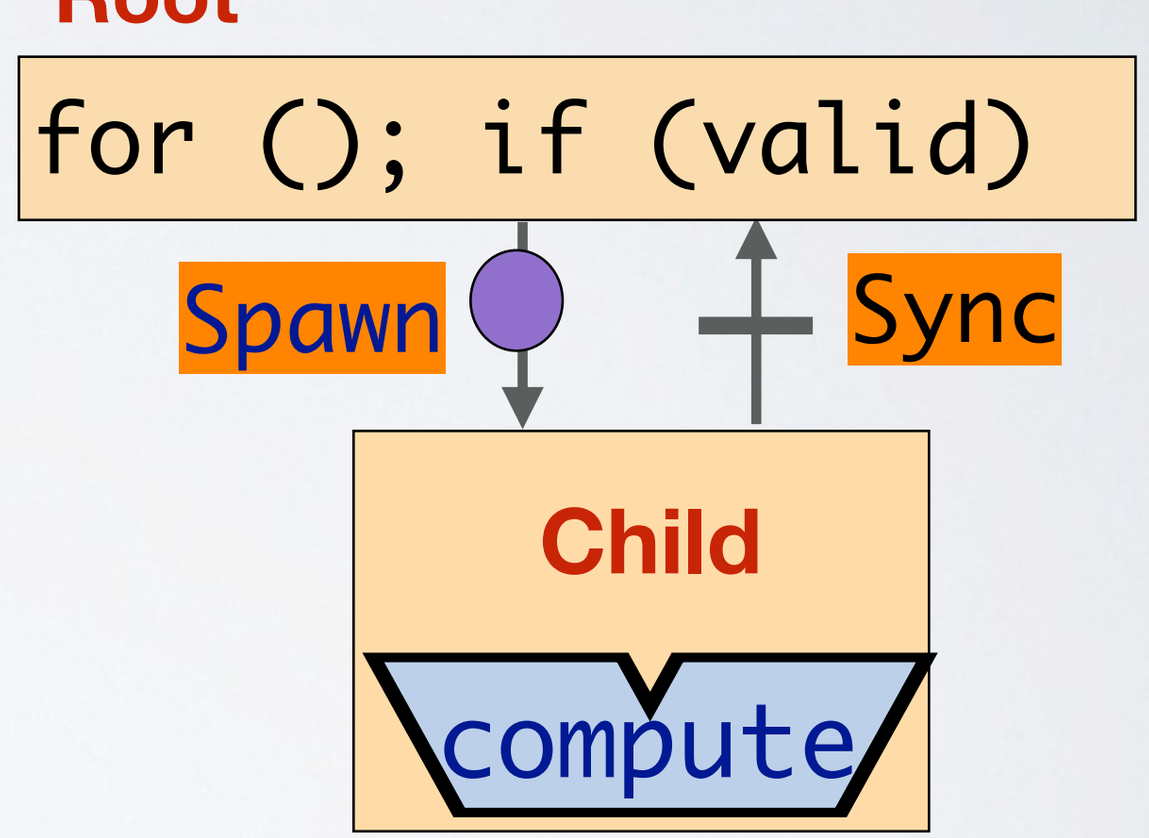
Worst case schedule —> Low utilization

Our Approach: Dynamic Parallelism

Task Program

```
for(i = 0 until n){
    if(node[i].valid){
Spawn compute(&node[i]);
    }
Sync;
```

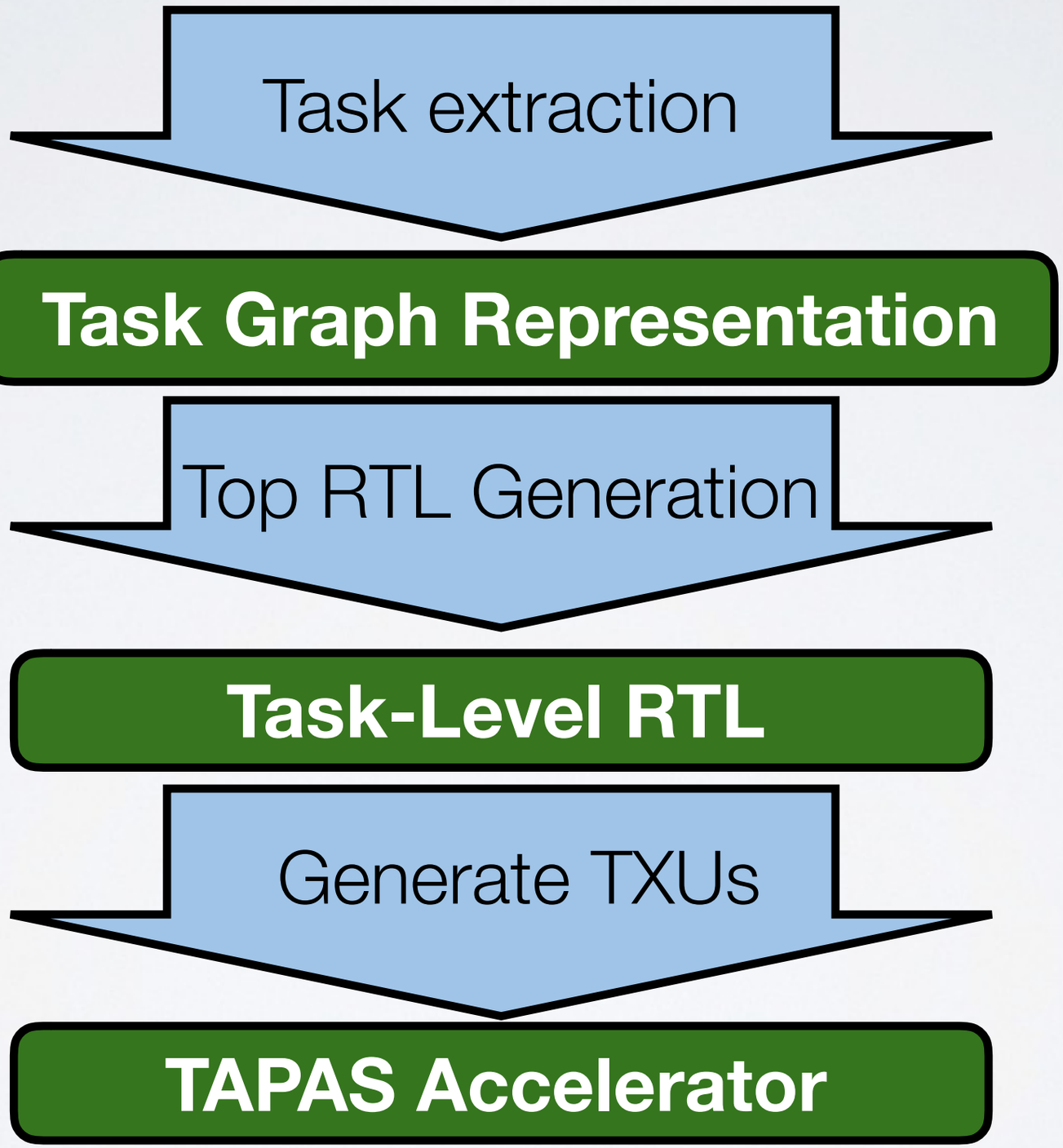
Task Hardware



Run time schedule —> High utilization

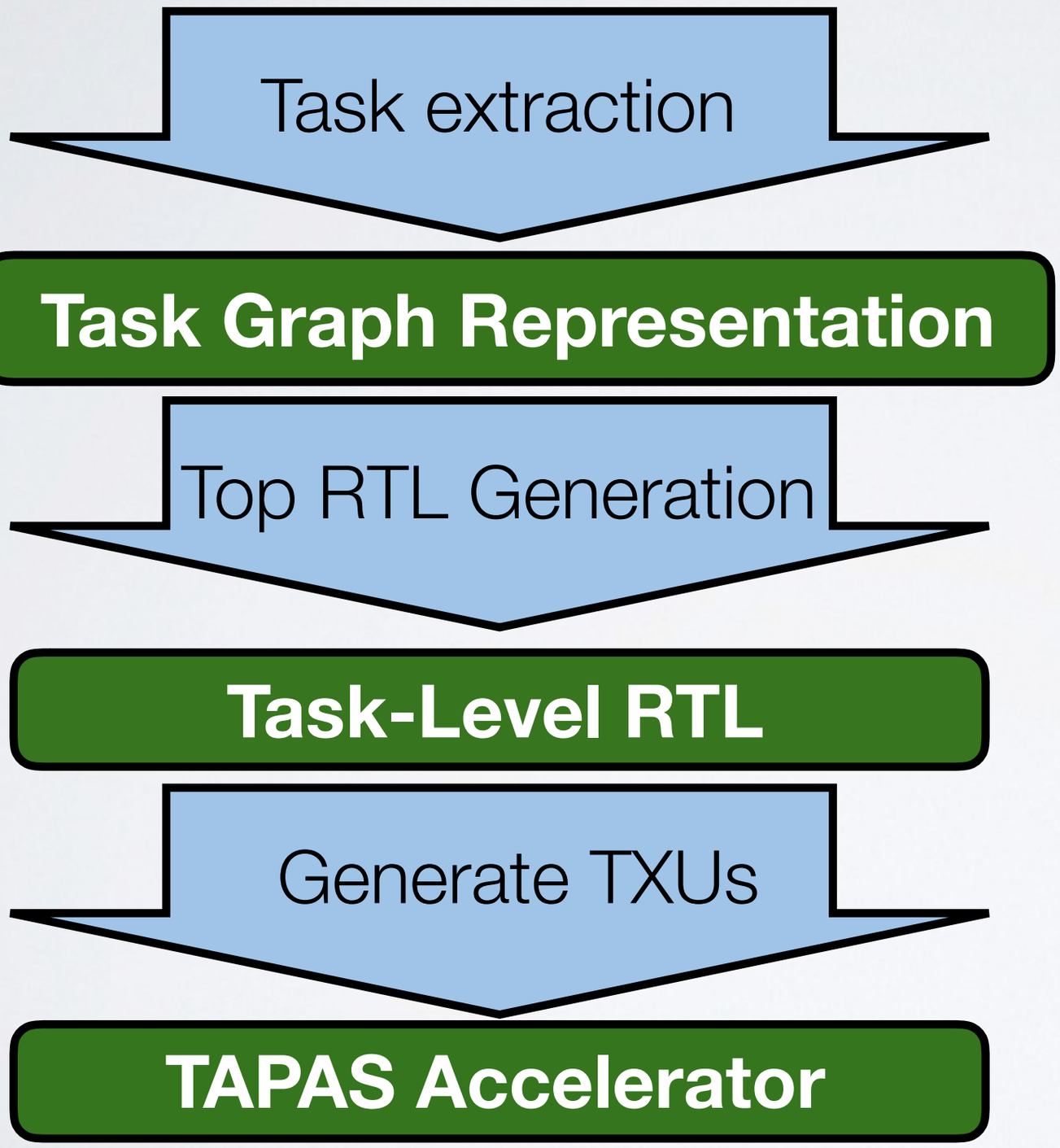
Compilation Flow

Cilk/Go



Compilation Flow

Cilk/Go




```
for(i = 0 until n){
  if(node[i].valid){
    compute(&node[i])
  }
}
```

Compilation Flow

Cilk/Go

Task extraction

Task Graph Representation

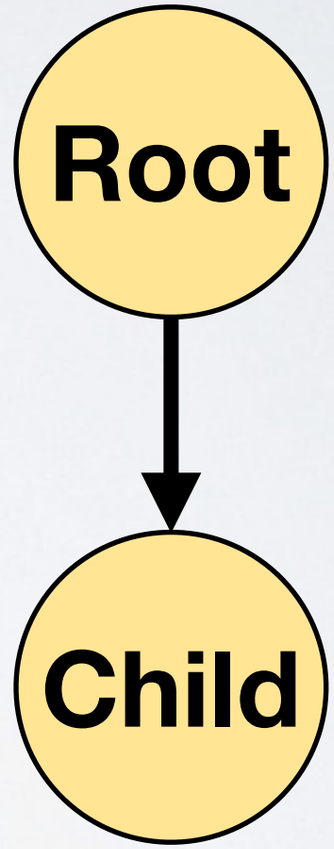
Top RTL Generation

Task-Level RTL

Generate TXUs

TAPAS Accelerator

Static Task Graph



Compilation Flow

Cilk/Go

Task extraction

Task Graph Representation

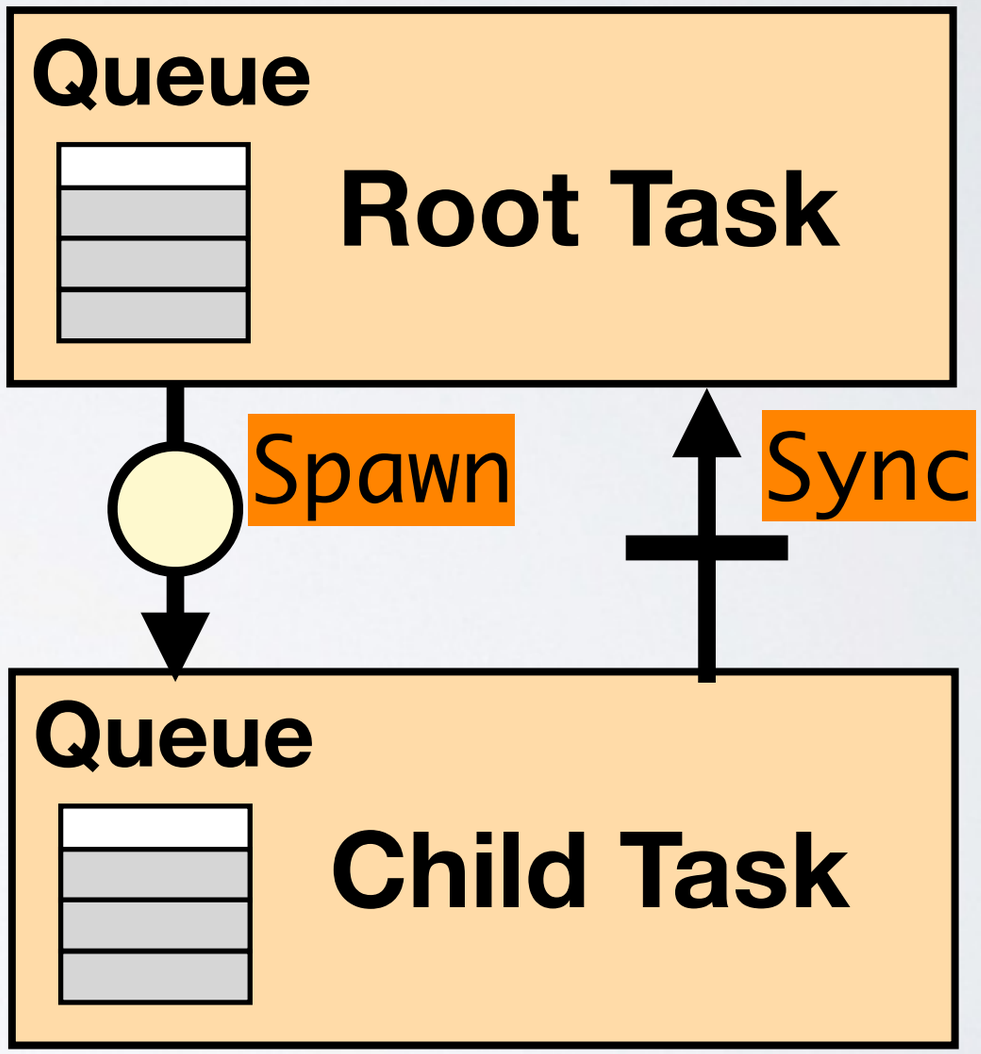
Top RTL Generation

Task-Level RTL

Generate TXUs

TAPAS Accelerator

Task-level Architecture



Compilation Flow

Cilk/Go

Task extraction

Task Graph Representation

Top RTL Generation

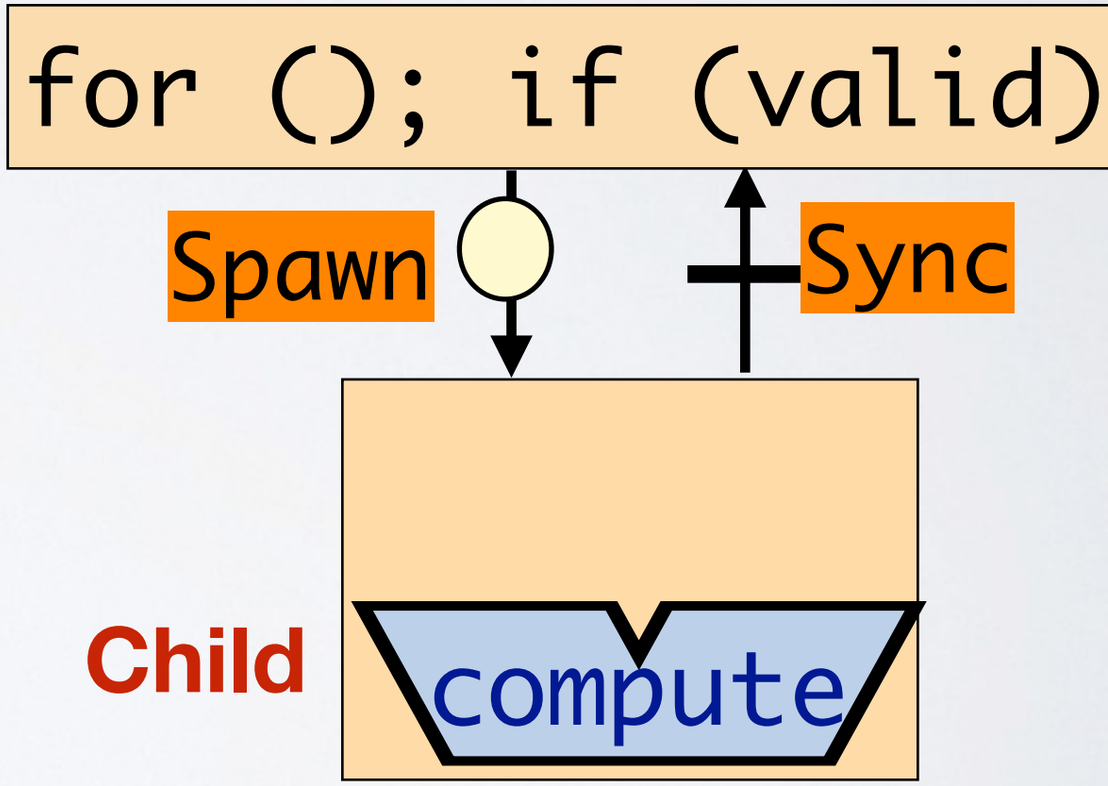
Task-Level RTL

Generate TXUs

TAPAS Accelerator

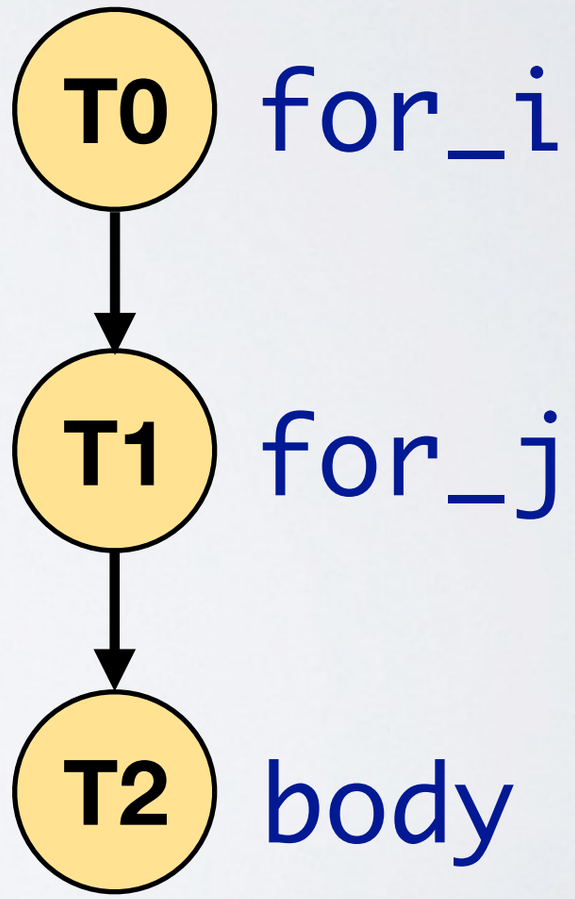
Task Execution Unit

Root



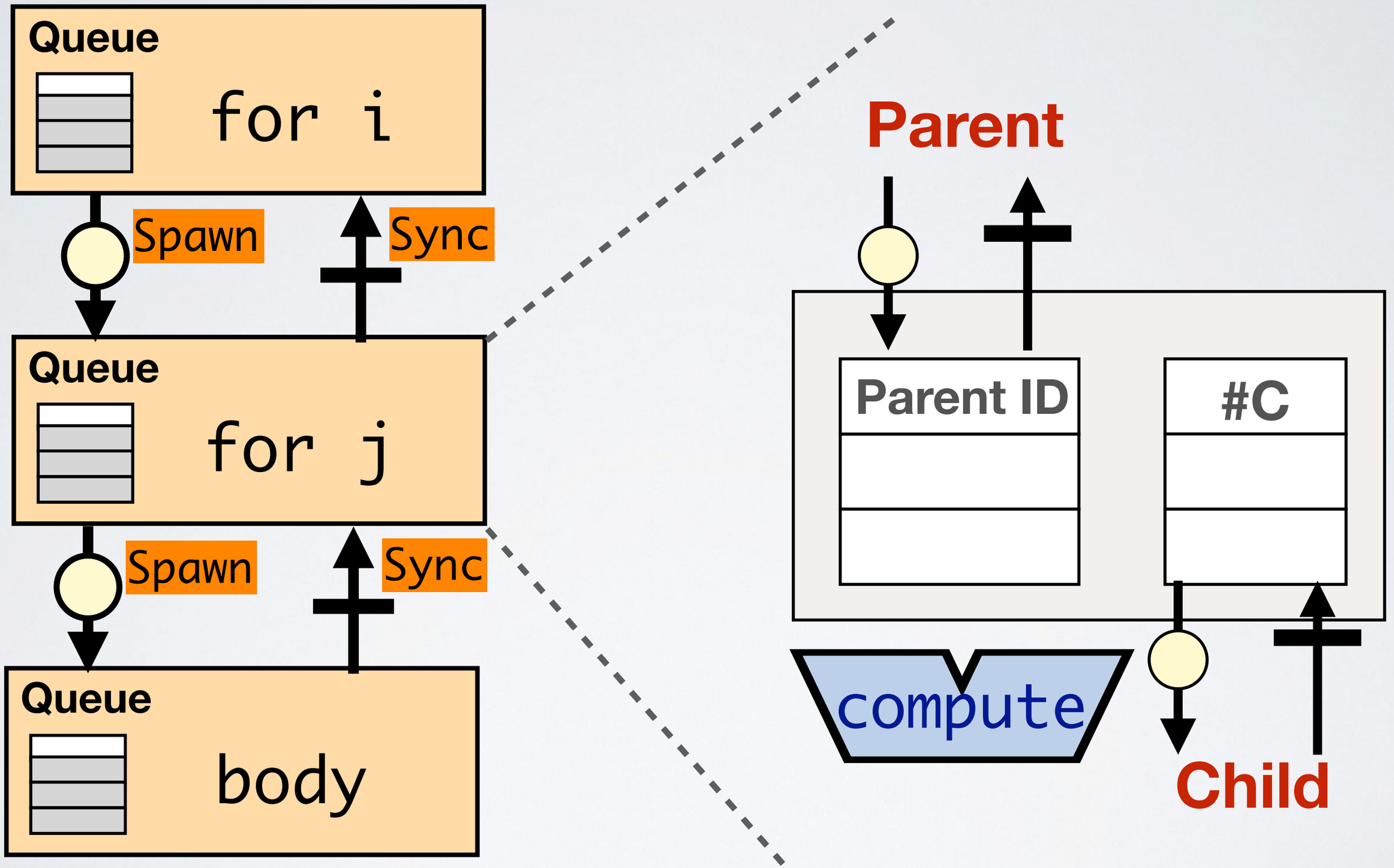
Static Task Graph

```
cilk_for(i = 0 until n){
   cilk_for(j = 0 until n){
      c[i][j] =
           a[i][j]+b[i][j];
   }
}
```



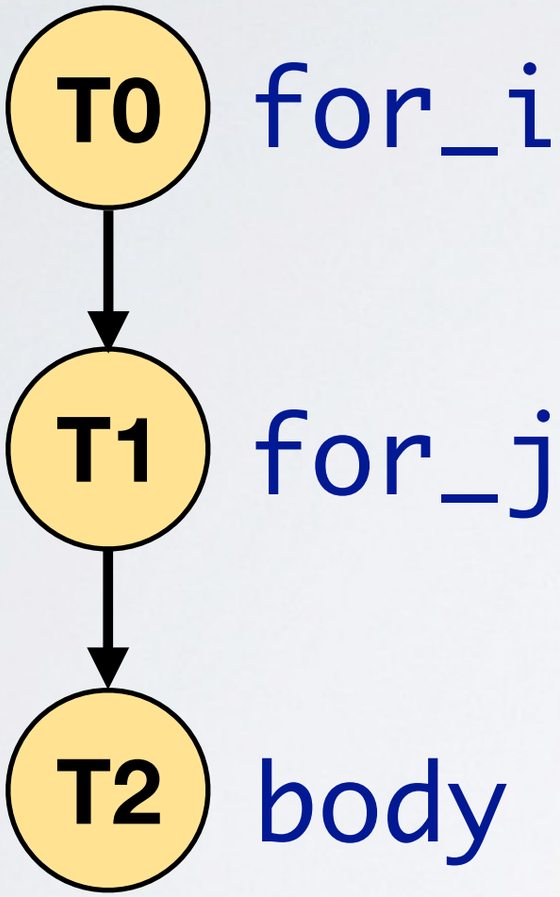
- **Parallel Compiler**
  - Captures Spawn and Sync from IR
- **Task Extractor:**
  - Wraps each task in a first class entity

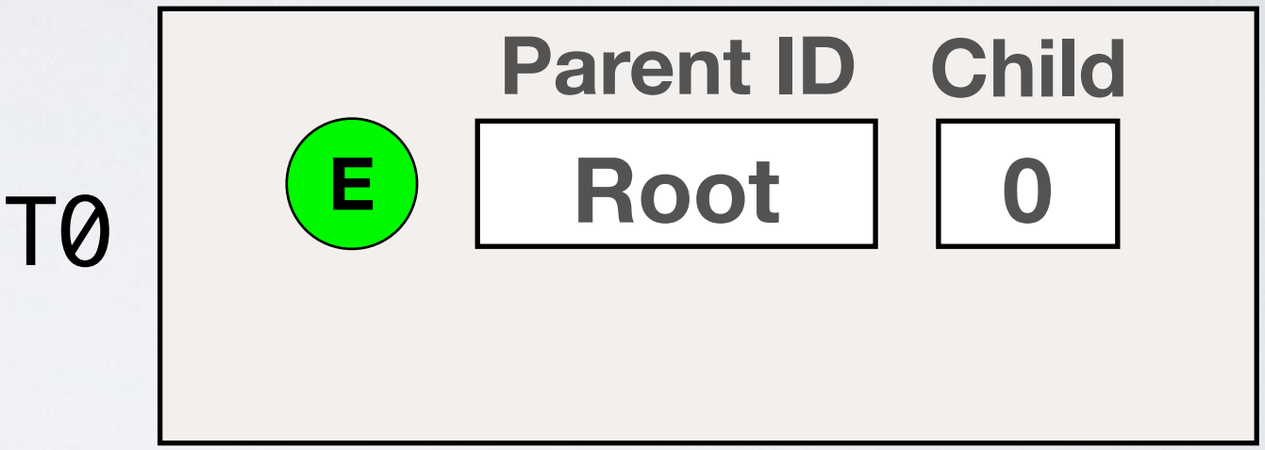
Task-Level Architecture

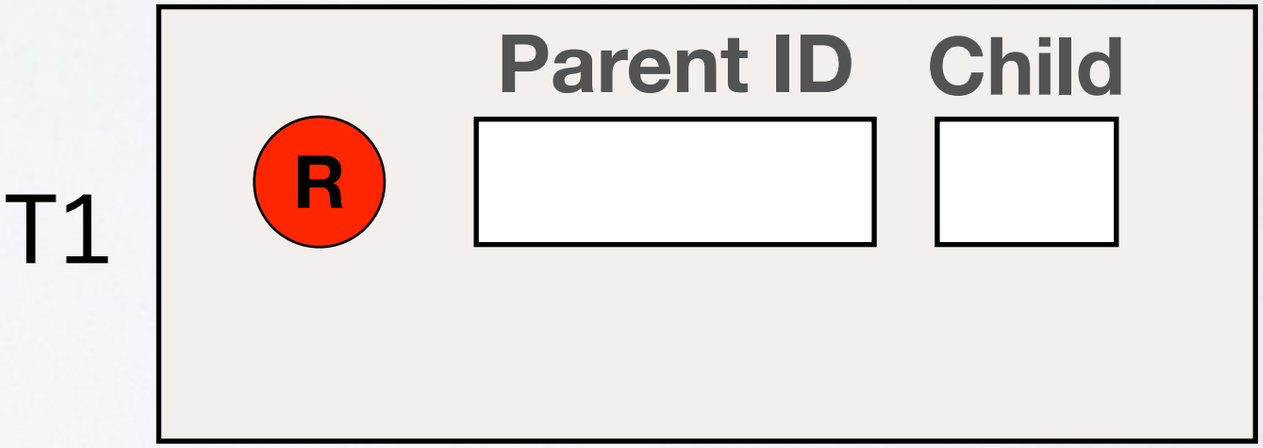


Heterogeneous! **Nested Parallel!** **Asynchronous!**

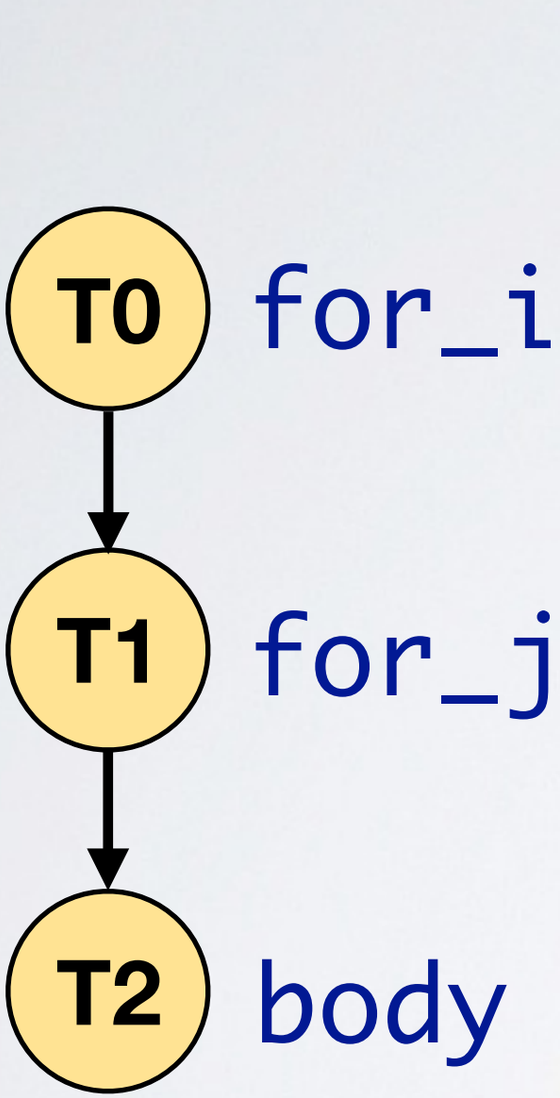
Task Level Execution

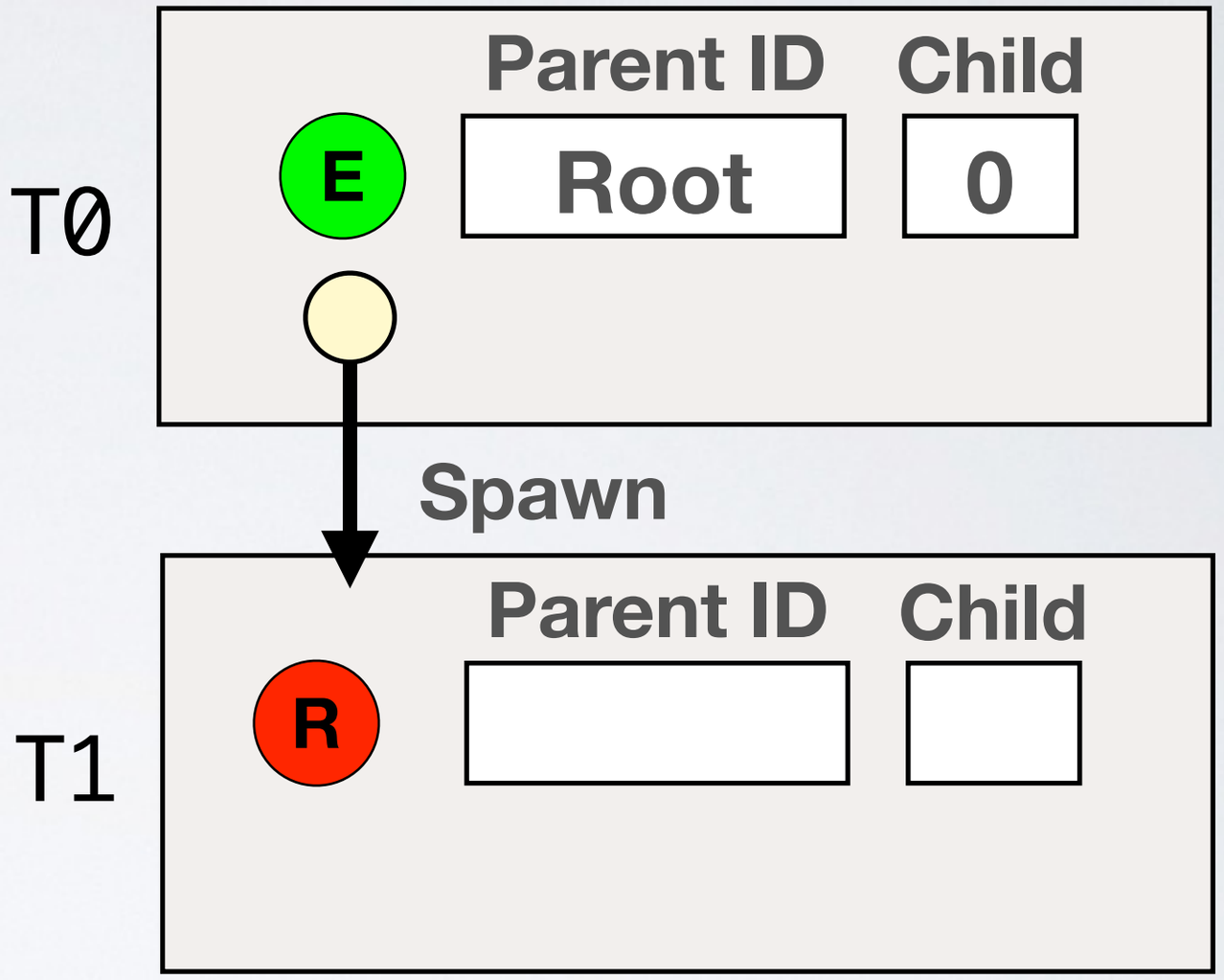




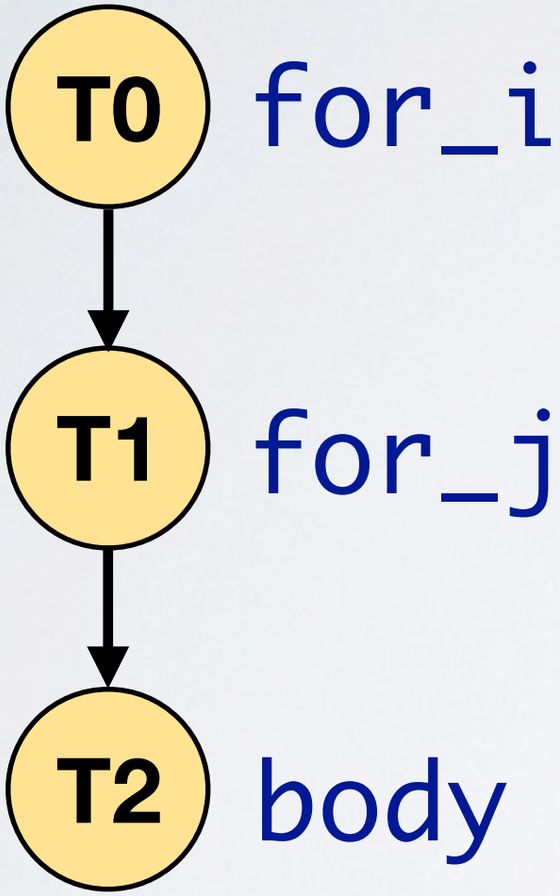


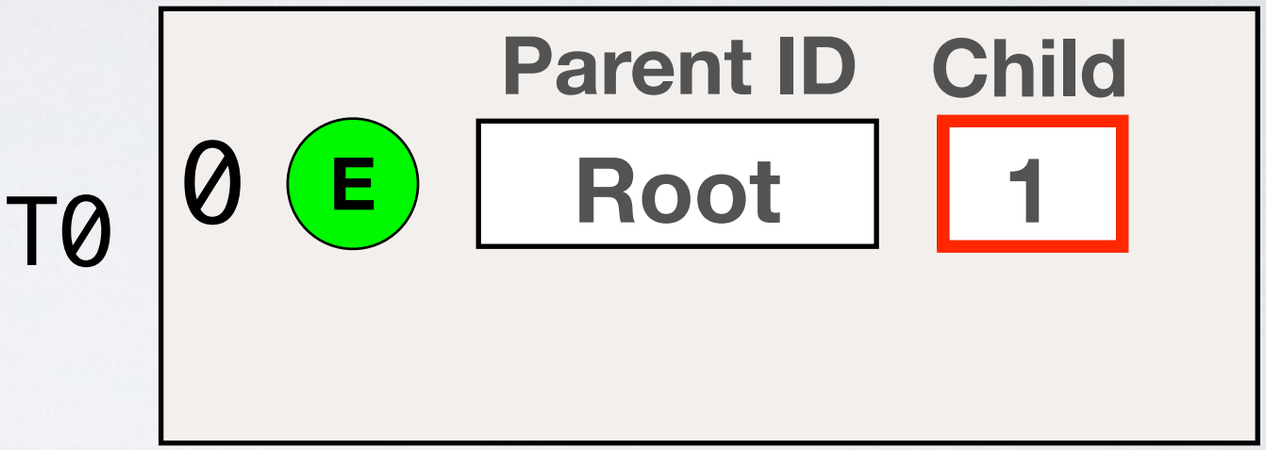
Task Level Execution

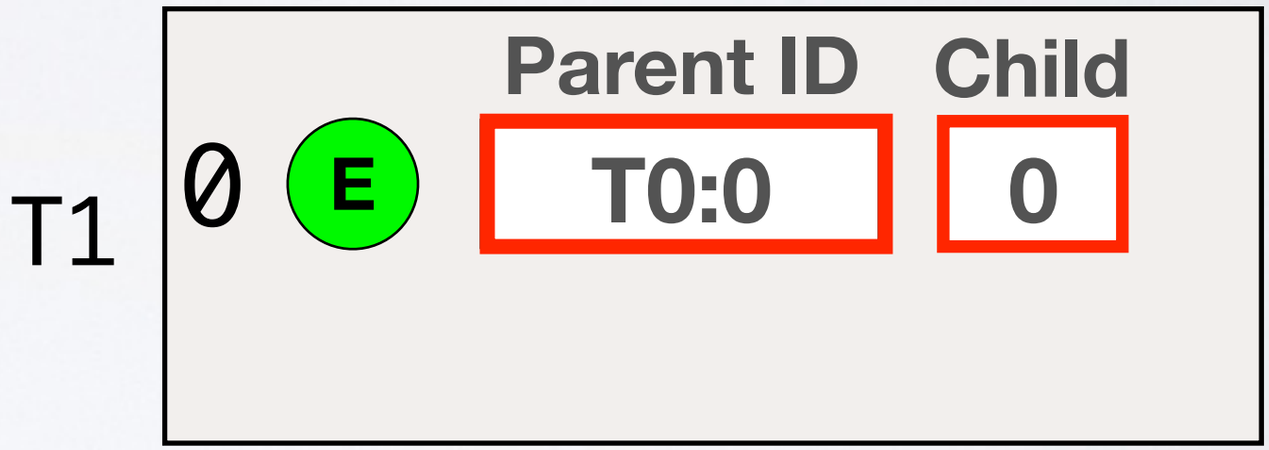




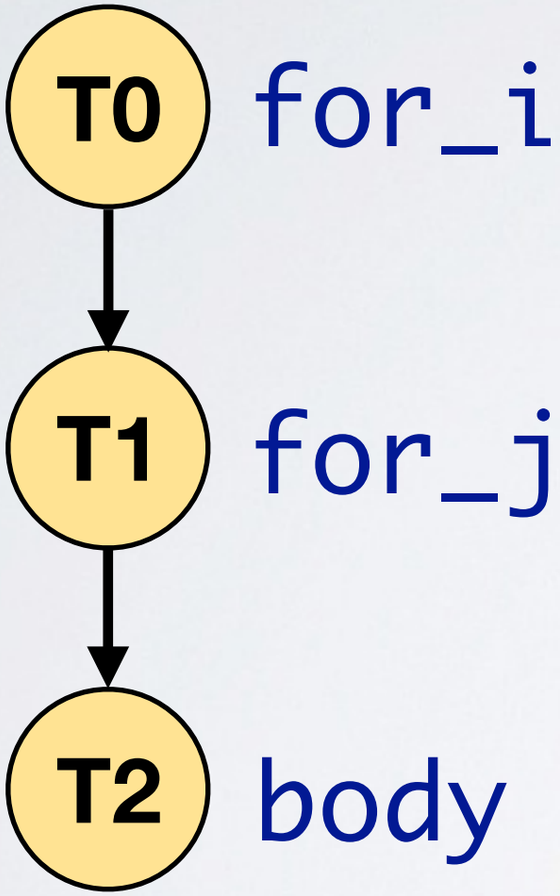
Task Level Execution

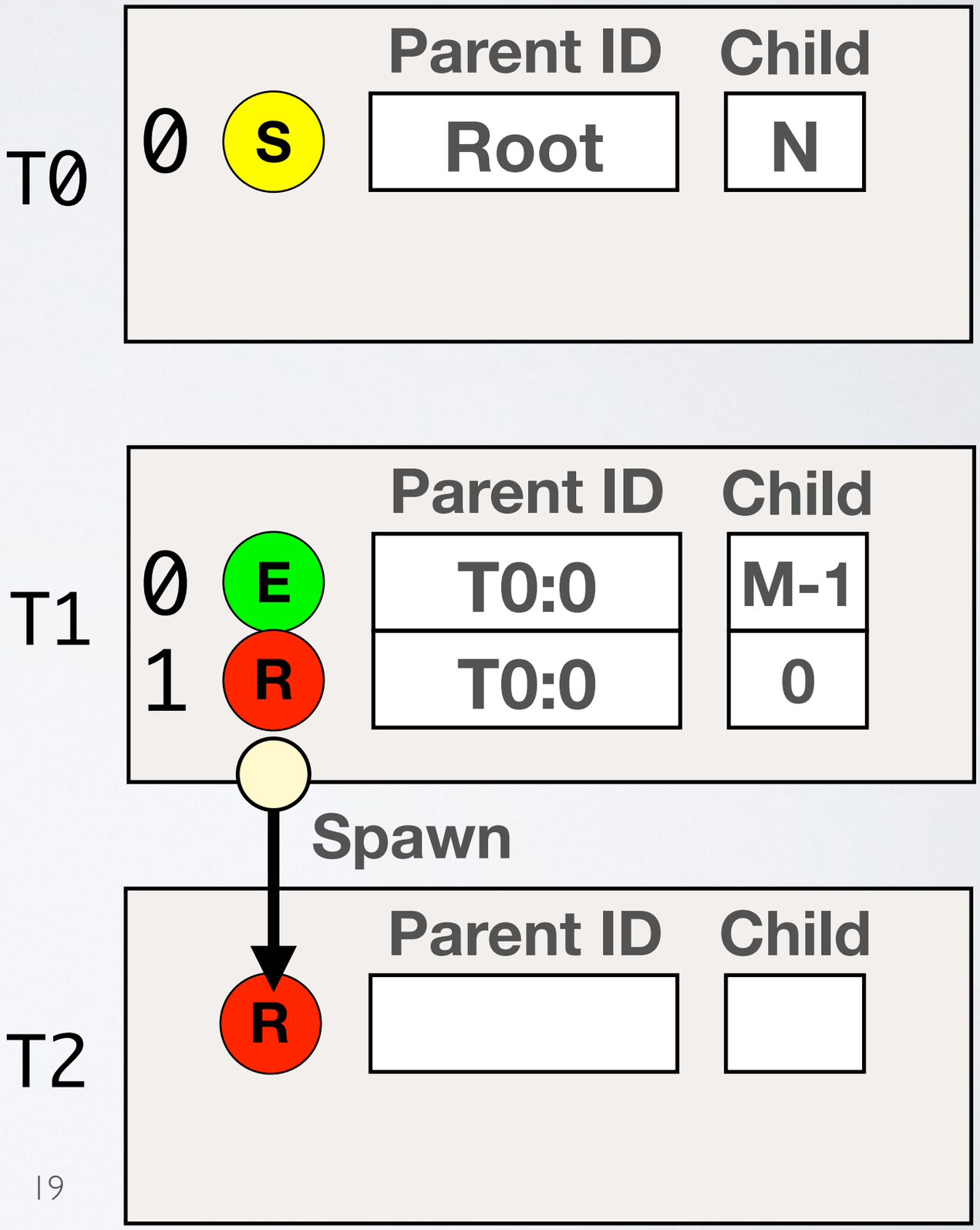




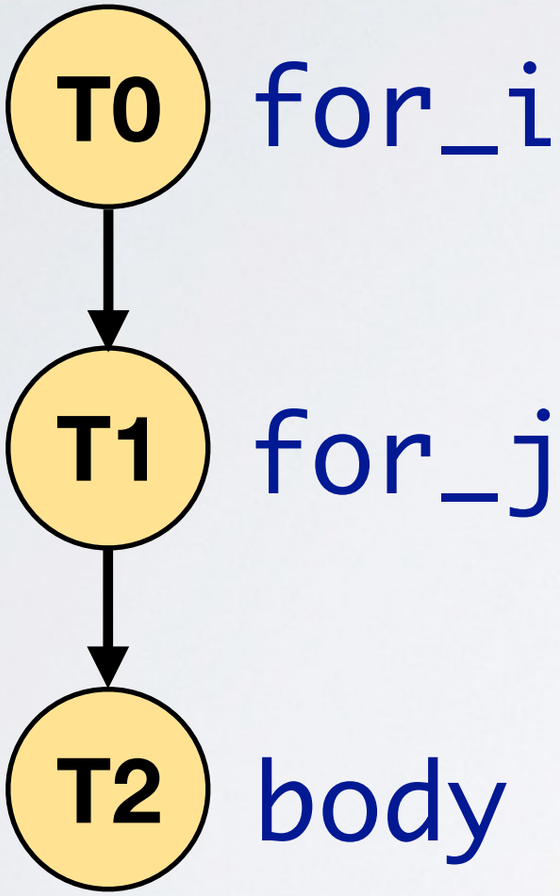


Task-Level Execution





Task Level Execution



T0

		Parent ID	Child
0	S	Root	N

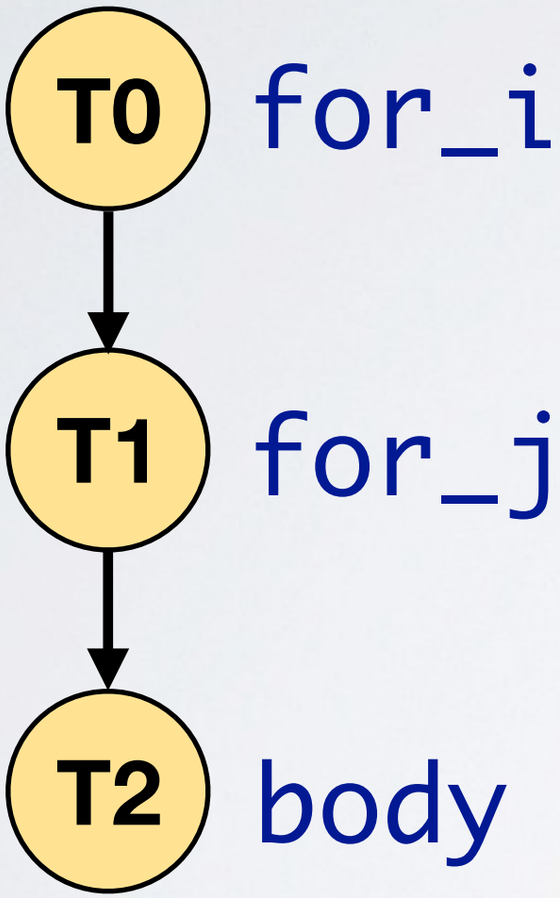
T1

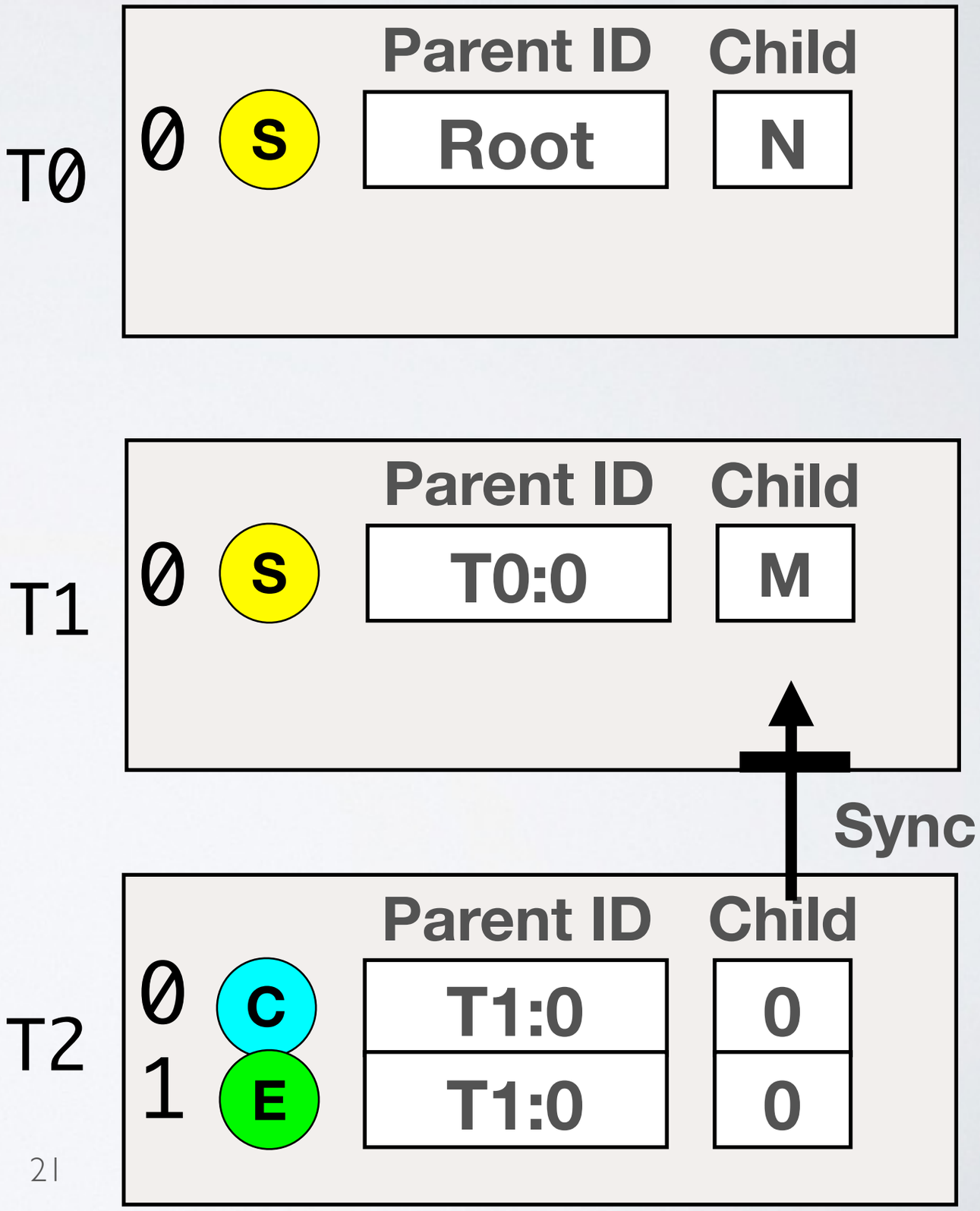
		Parent ID	Child
0	S	T0:0	M
1	E	T0:0	0

T2

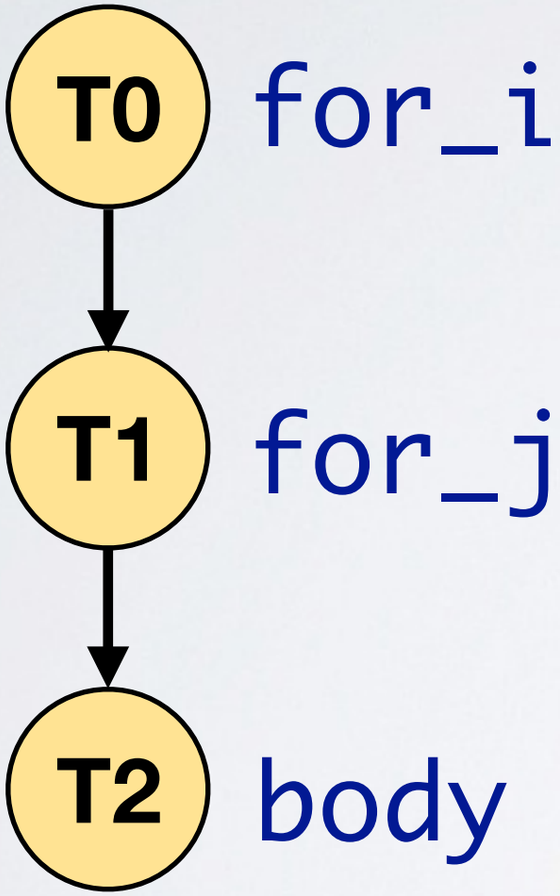
		Parent ID	Child
0	E	T1:0	0

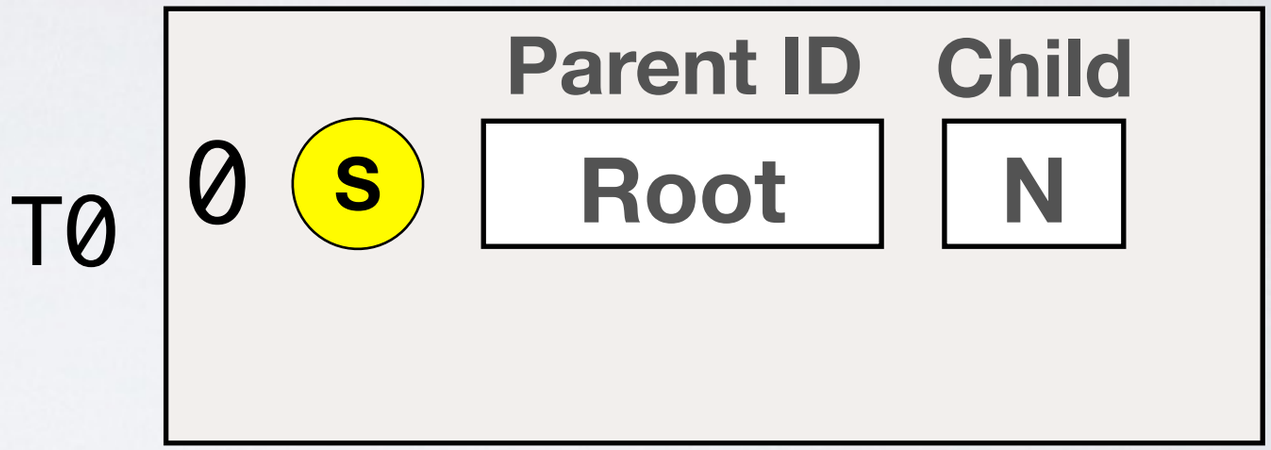
Task Level Execution

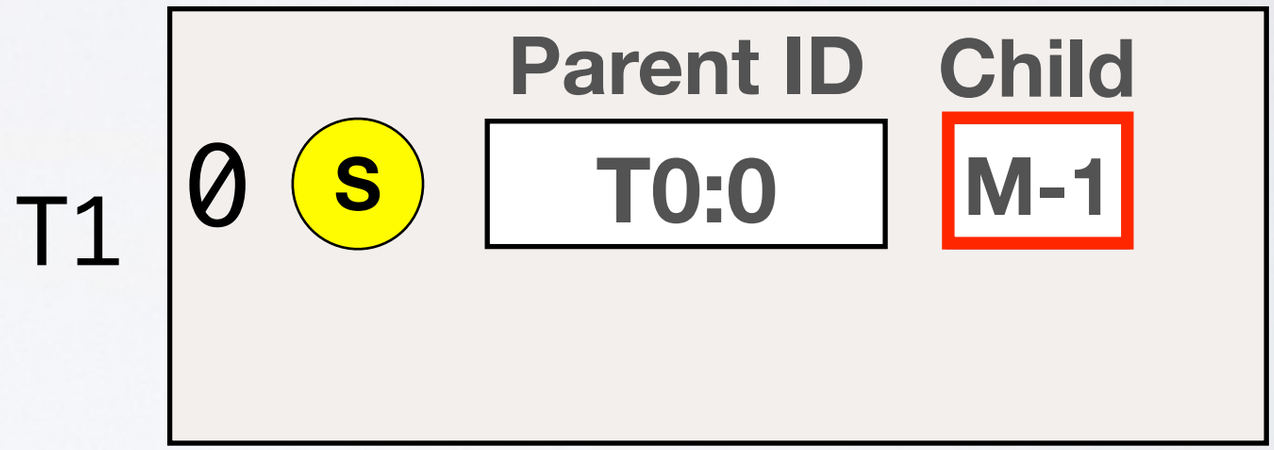


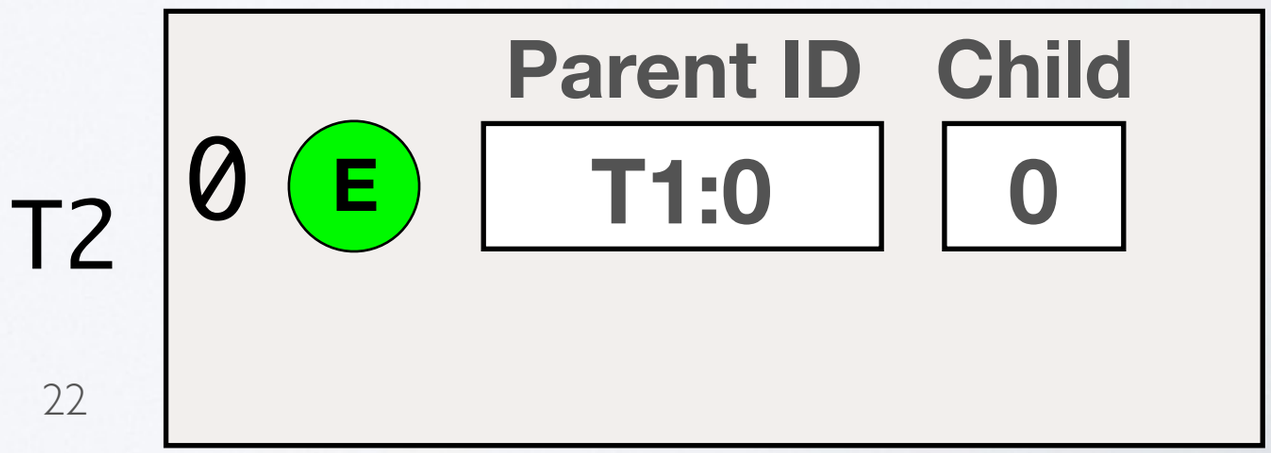


Task Level Execution





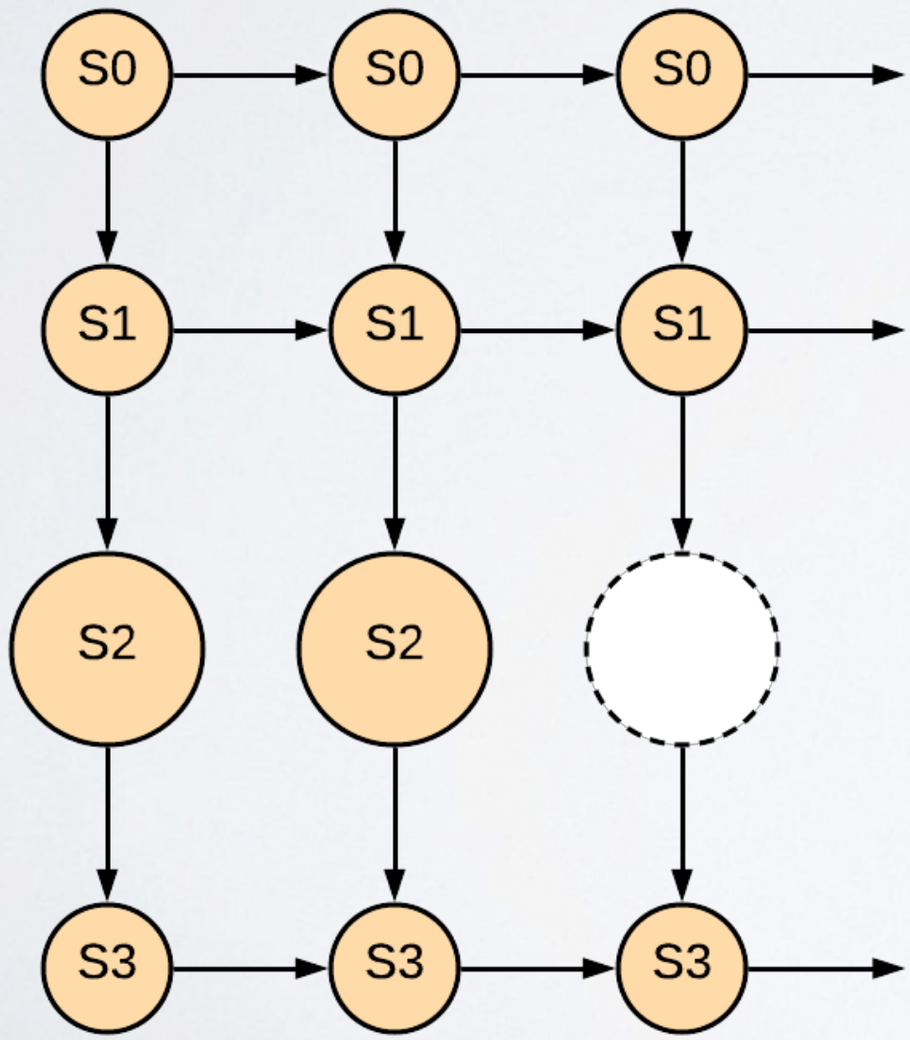




Supporting Dynamic Task Graphs

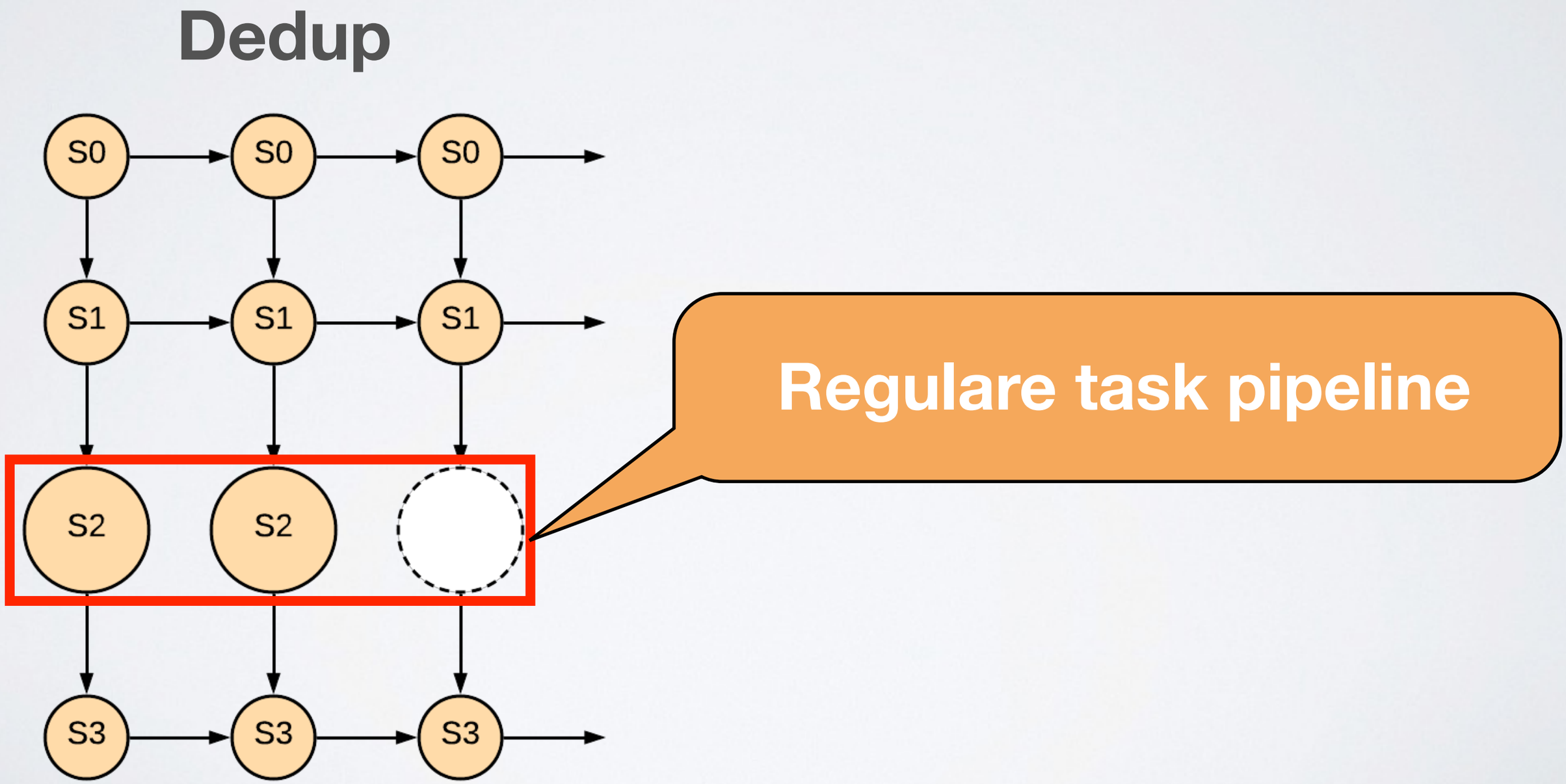
- What does task pipelining look like in TAPAS?

Dedup



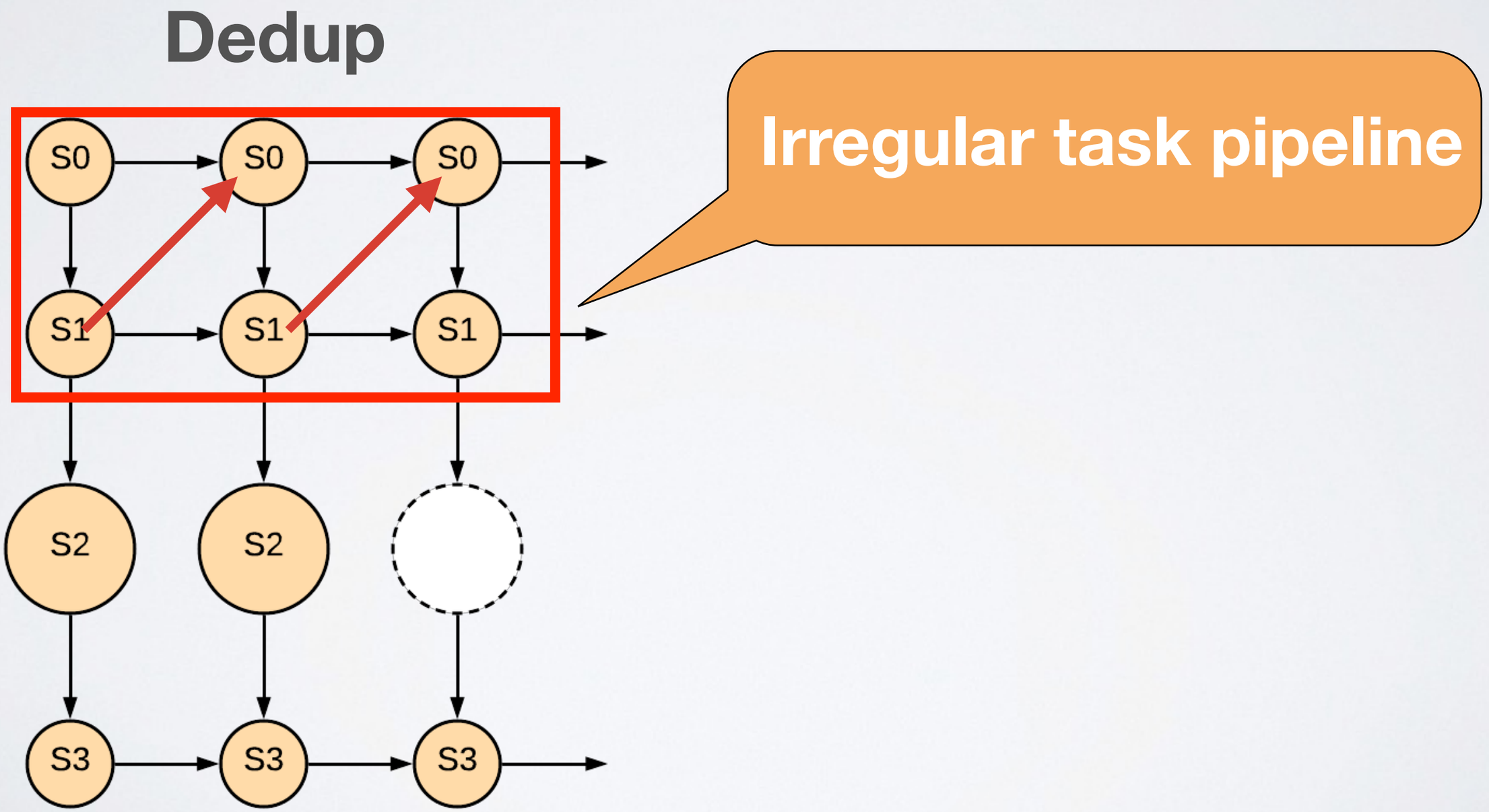
Supporting Dynamic Task Graphs

- What does task pipelining look like in TAPAS?



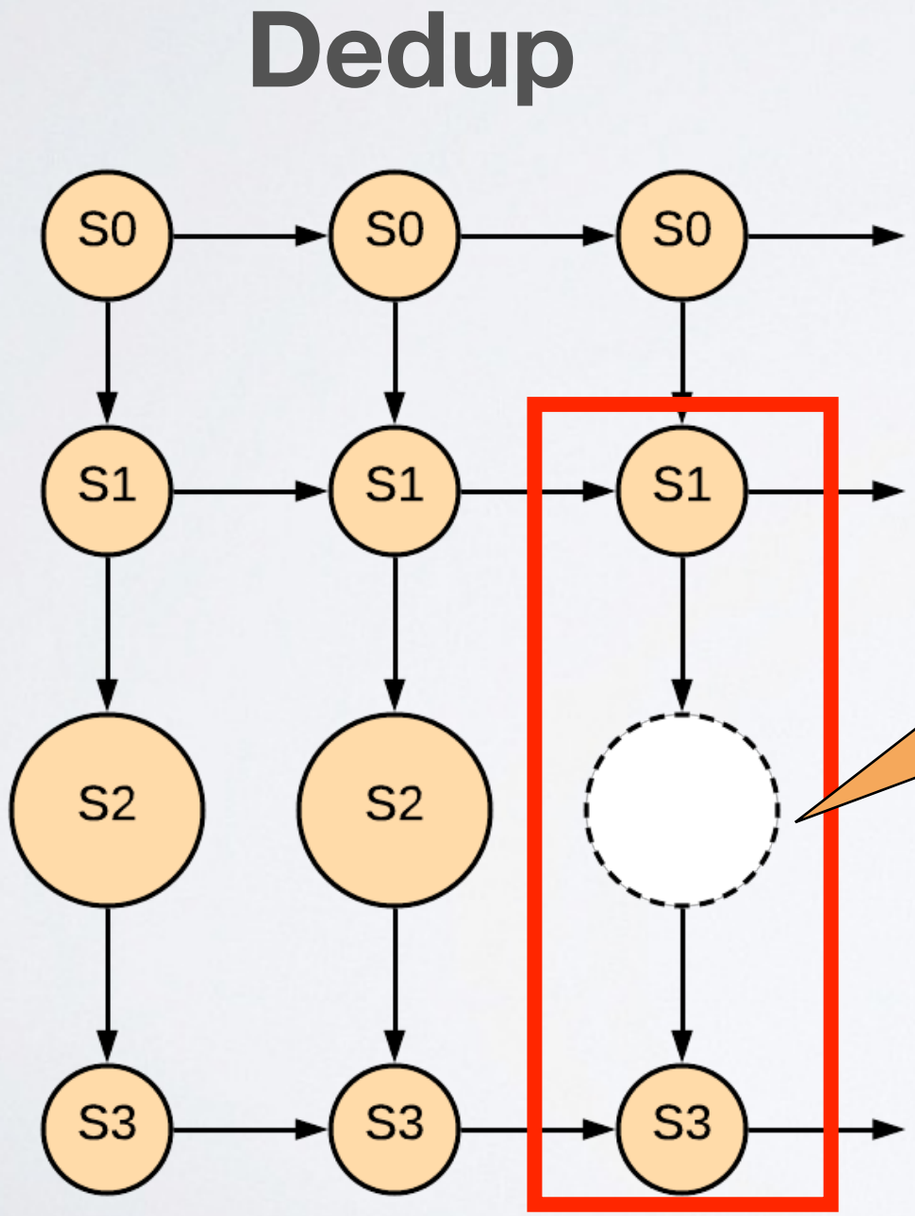
Supporting Dynamic Task Graphs

- What does task pipelining look like in TAPAS?



Supporting Dynamic Task Graphs

- What does task pipelining look like in TAPAS?



TAPAS supports recursive tasks as well (Paper: ⌘5)

Compilation Flow

Cilk/Go

Task extraction

Task Graph Representation

Top RTL Generation

Task-Level RTL

Generate TXUs

TAPAS Accelerator

Task Execution Unit

Root

```
for (); if (valid)
```

Spawn

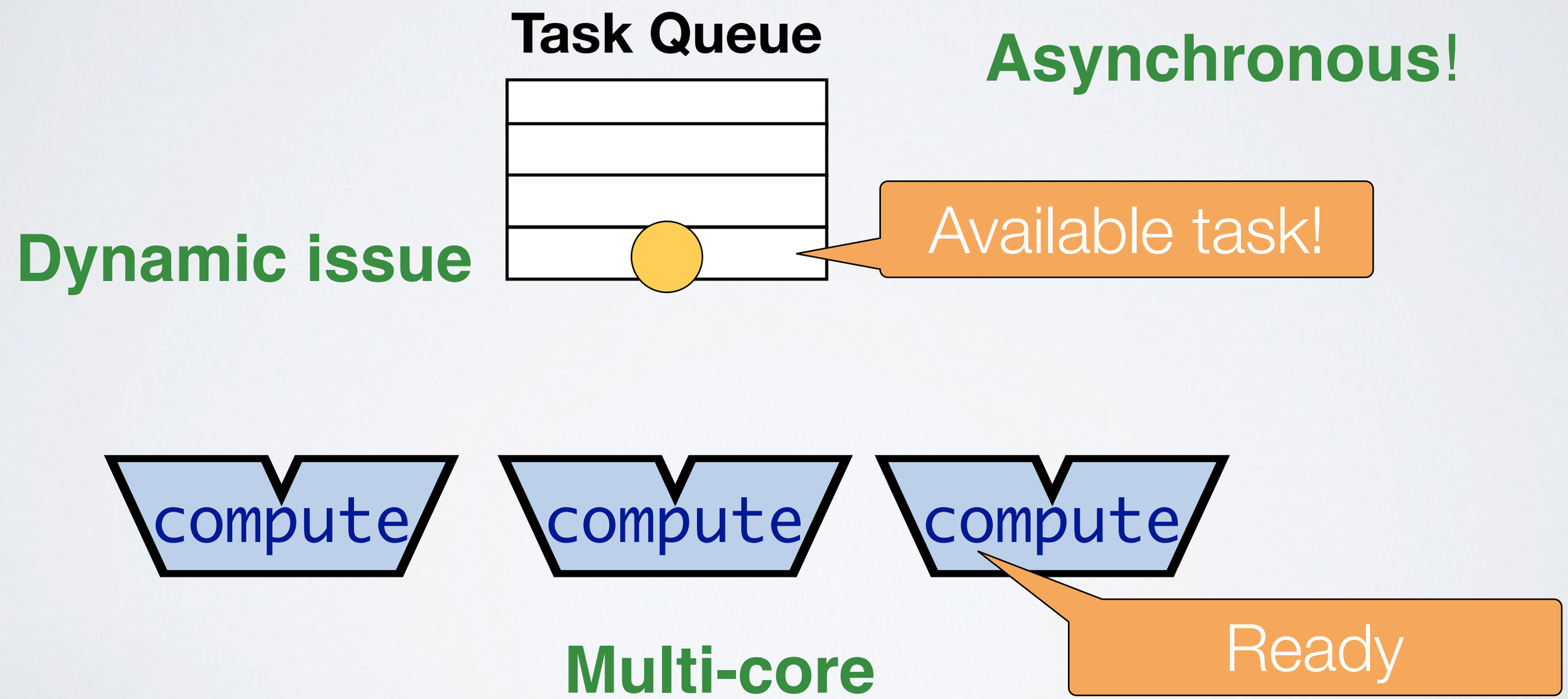
Sync

Child

compute

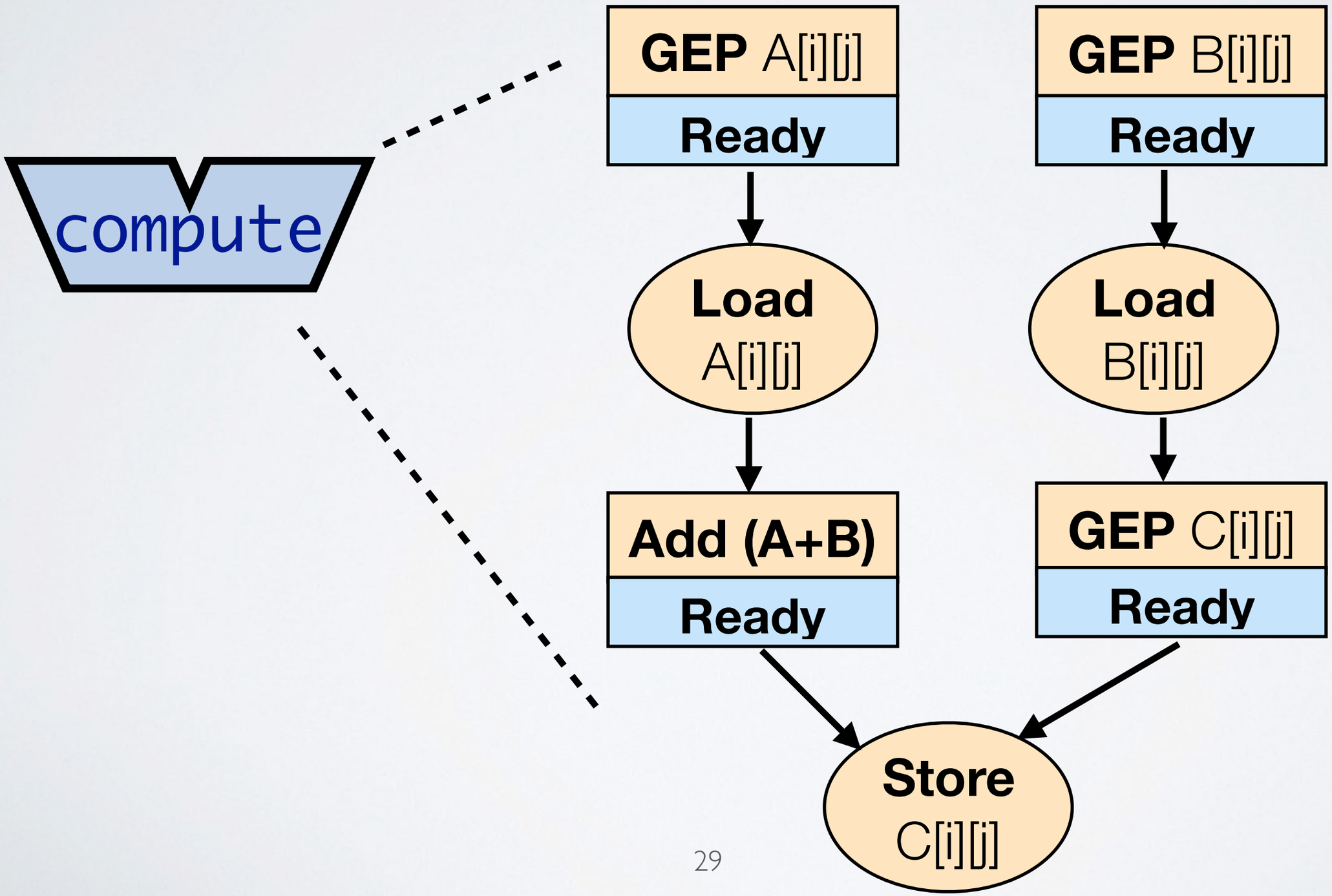
Task Execution Unit (TXU)

- What are the element inside each TXU?



Task Execution Unit (TXU)

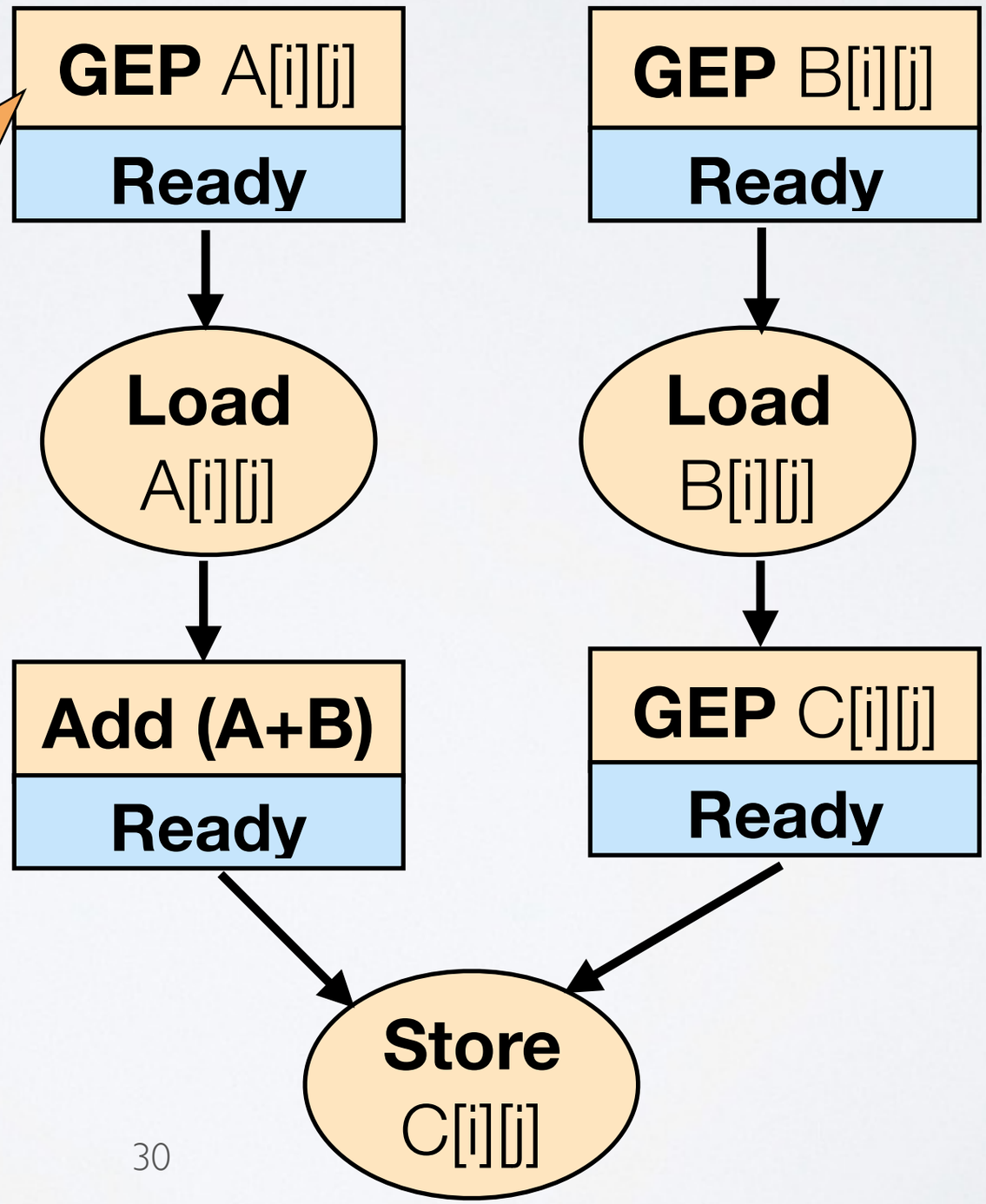
```
c[i][j] = a[i][j]+b[i][j];
```



Task Execution Unit (TXU)

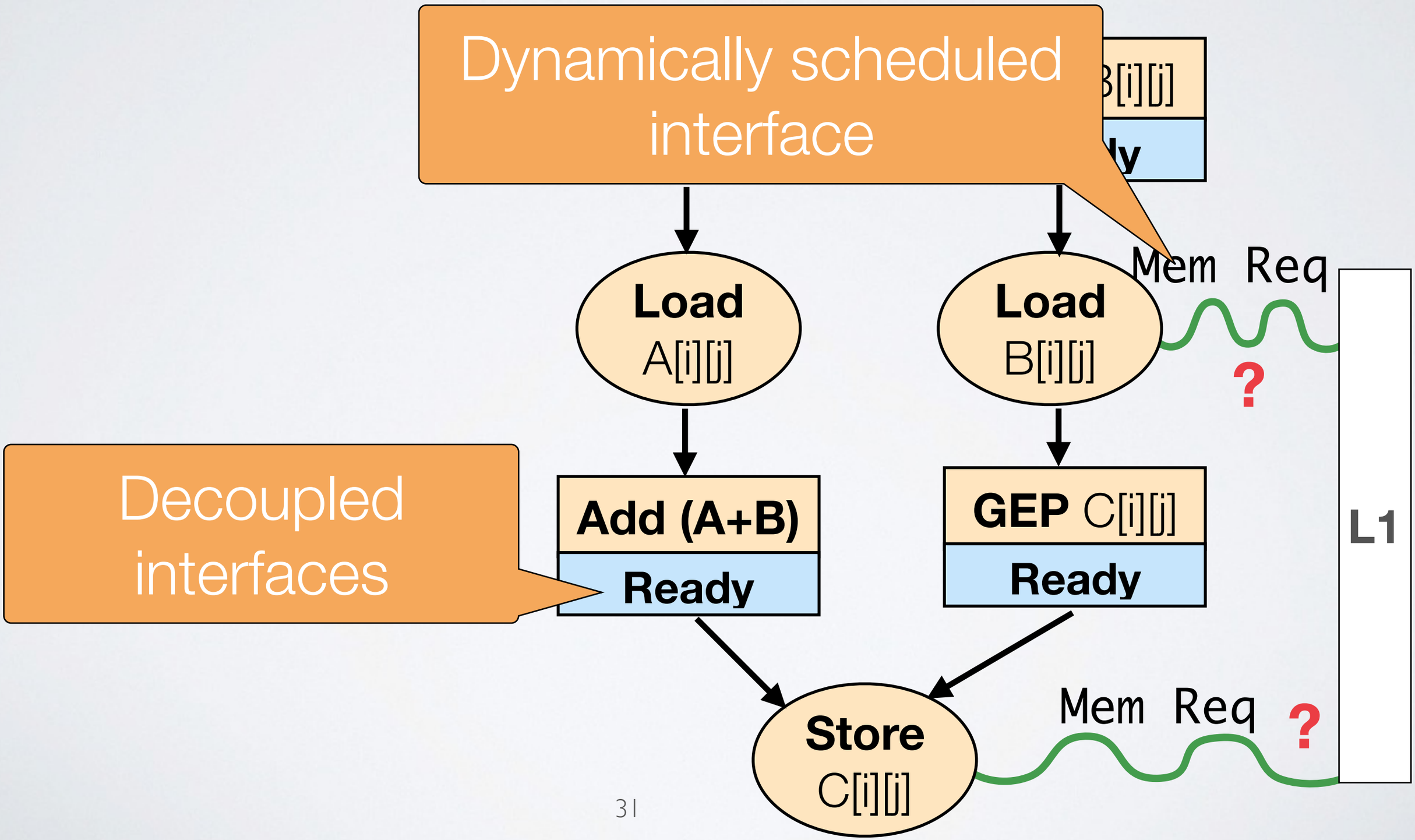
```
c[i][j] = a[i][j]+b[i][j];
```

Support for all LLVM semantics



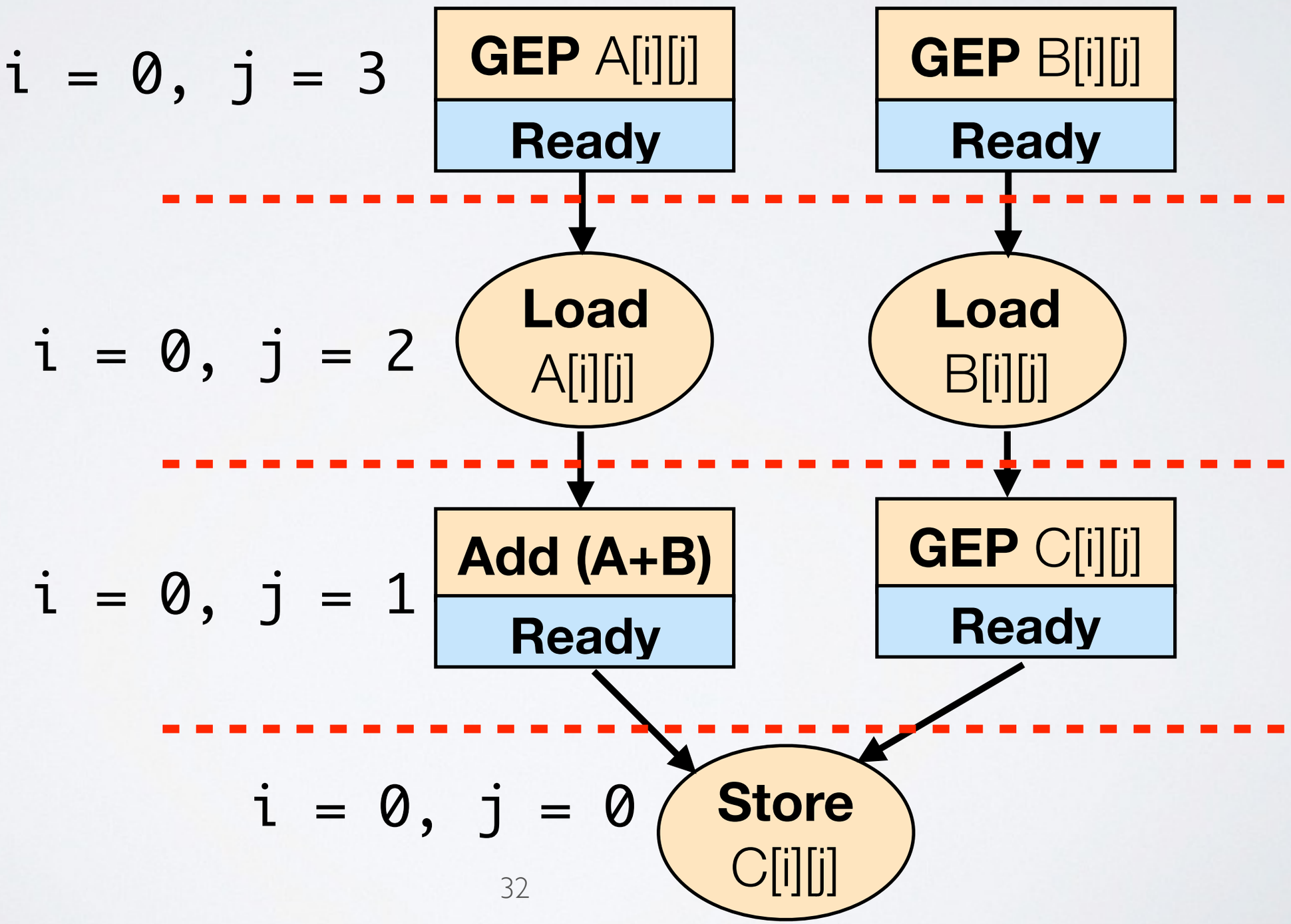
Task Execution Unit (TXU)

```
c[i][j] = a[i][j]+b[i][j];
```



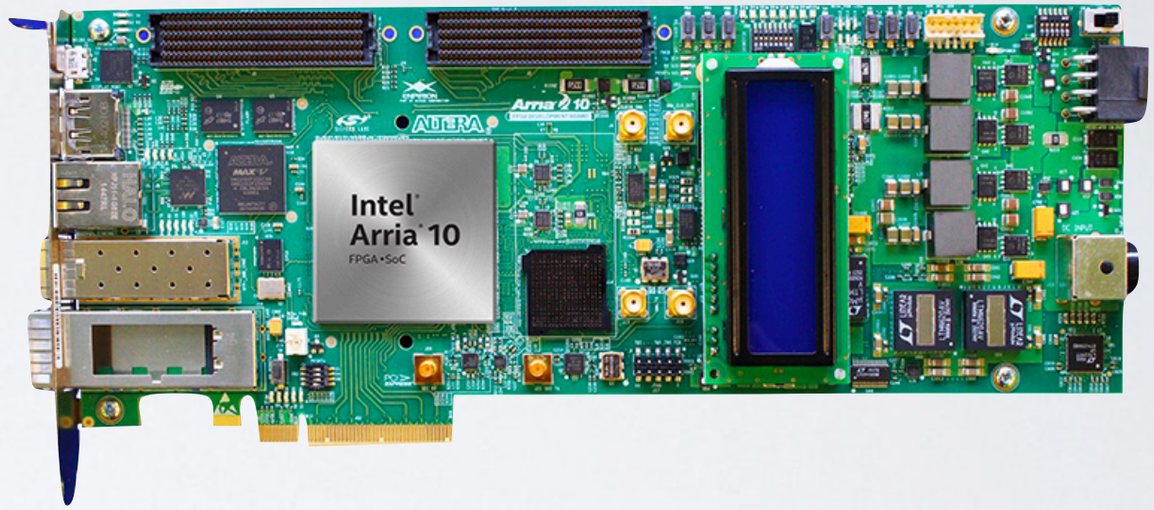
Task Execution Unit (TXU)

```
c[i][j] = a[i][j]+b[i][j];
```



Experiment

- Board:
  - Arria 10 SOC
  - Intel core i7

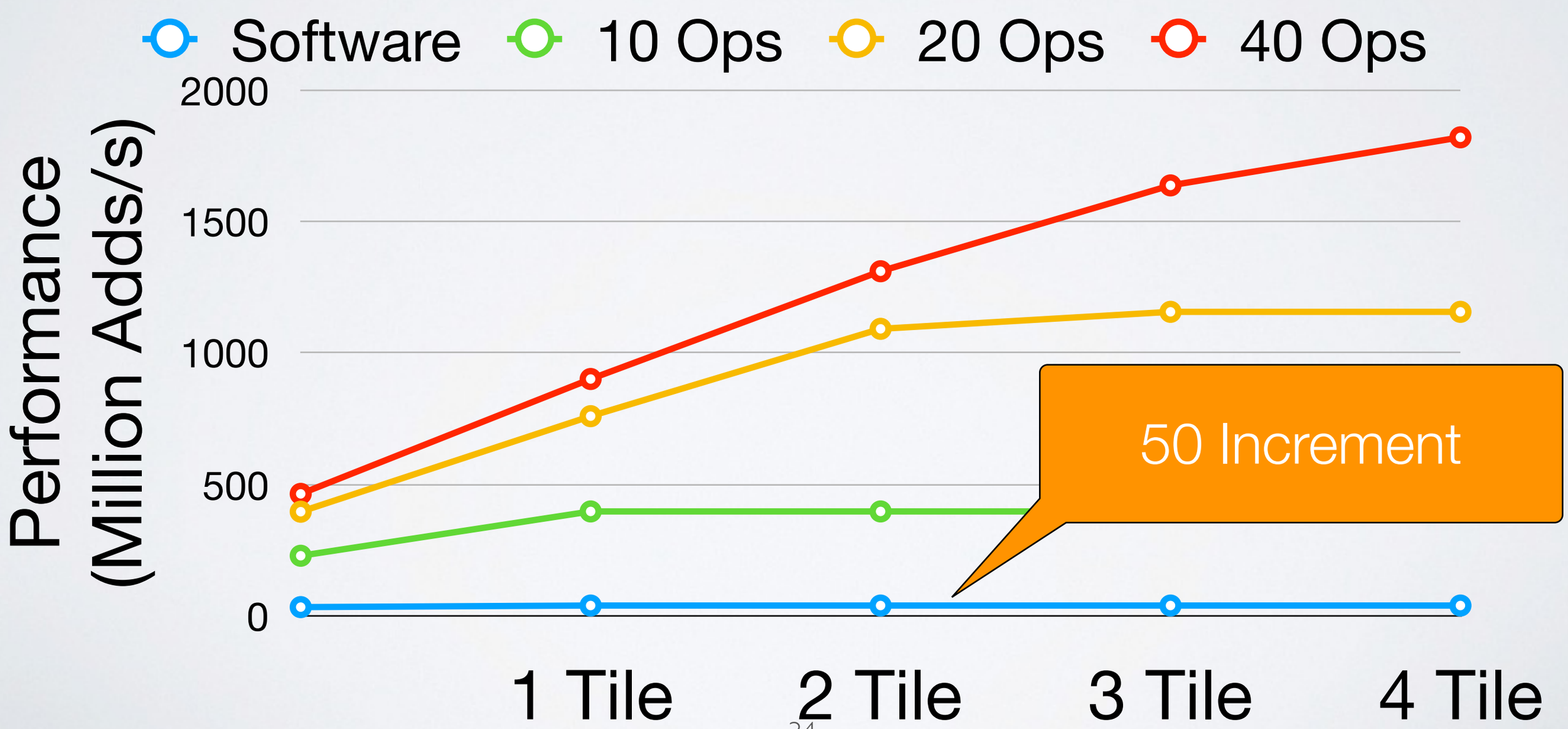


- Execution time reported
  - Number of Cycles

- Goal:
  - Performance/watt improvement
  - Reducing overhead of spawning tasks with few instructions

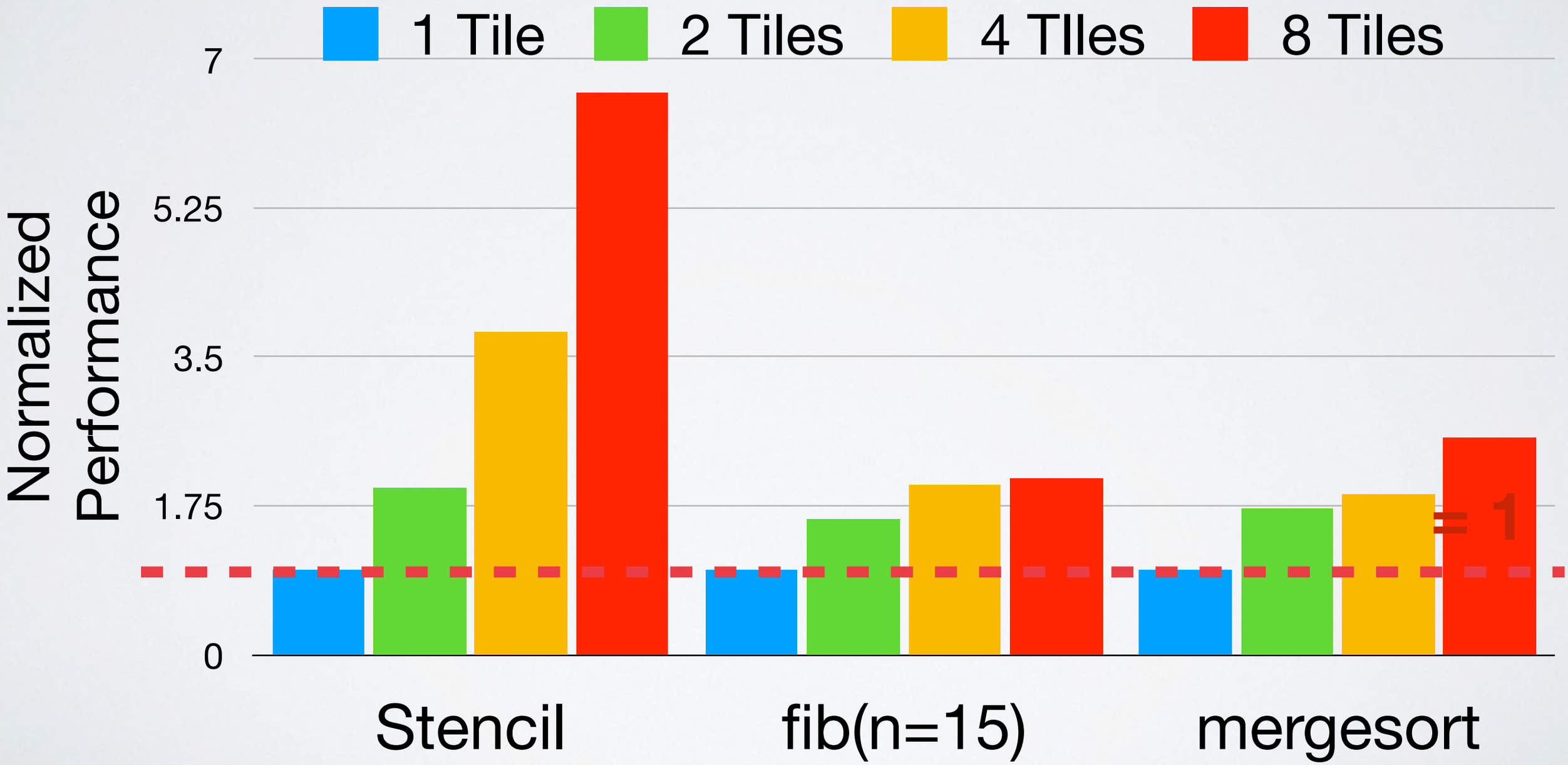
How does performance scale with workload size?

- Unlike a CPU, FPGA performance scales with **#TXU** even for ***fine grained*** parallelism.



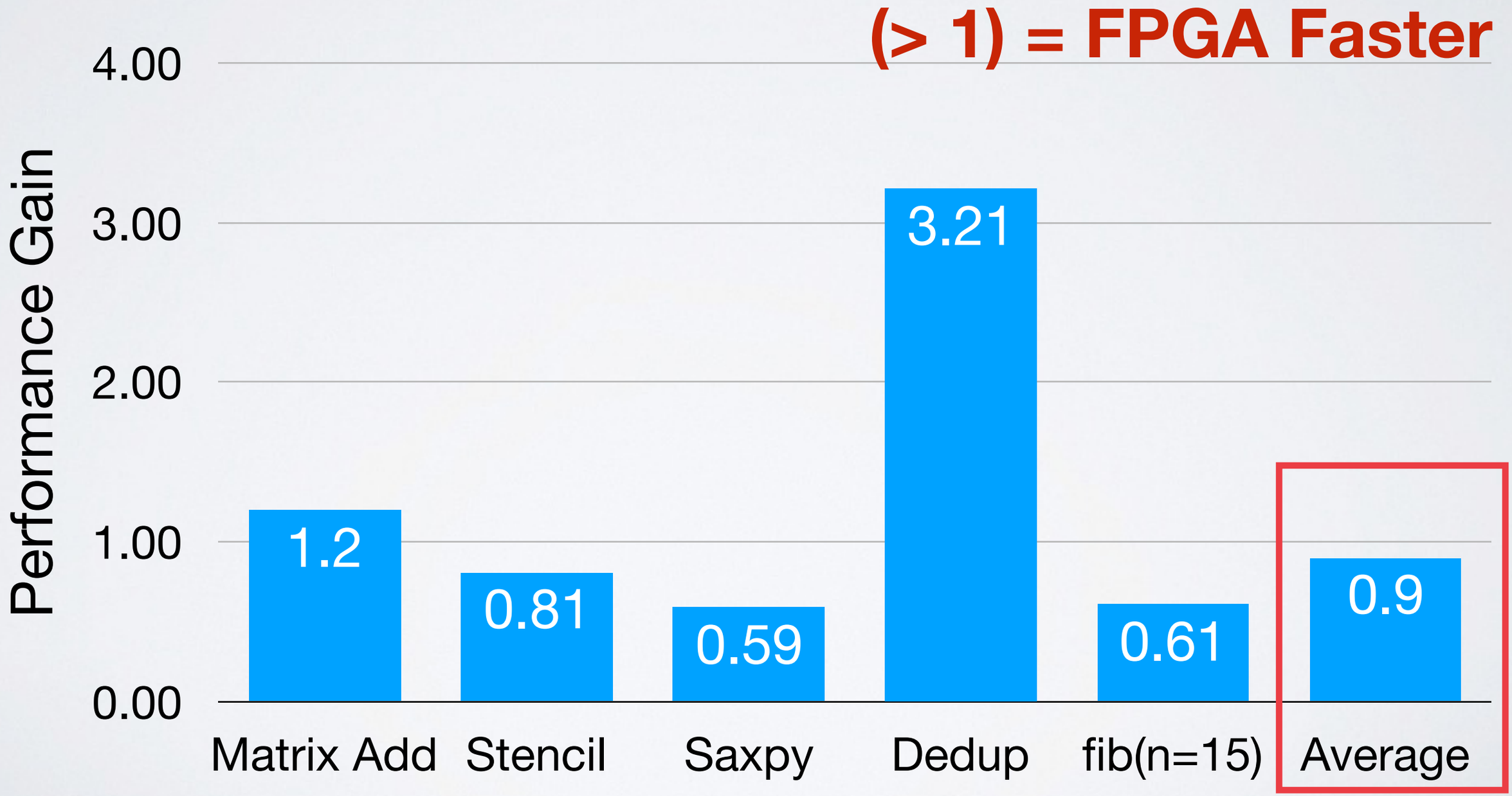
Does performance scale with recursion?

- Performance scales with recursive algorithms



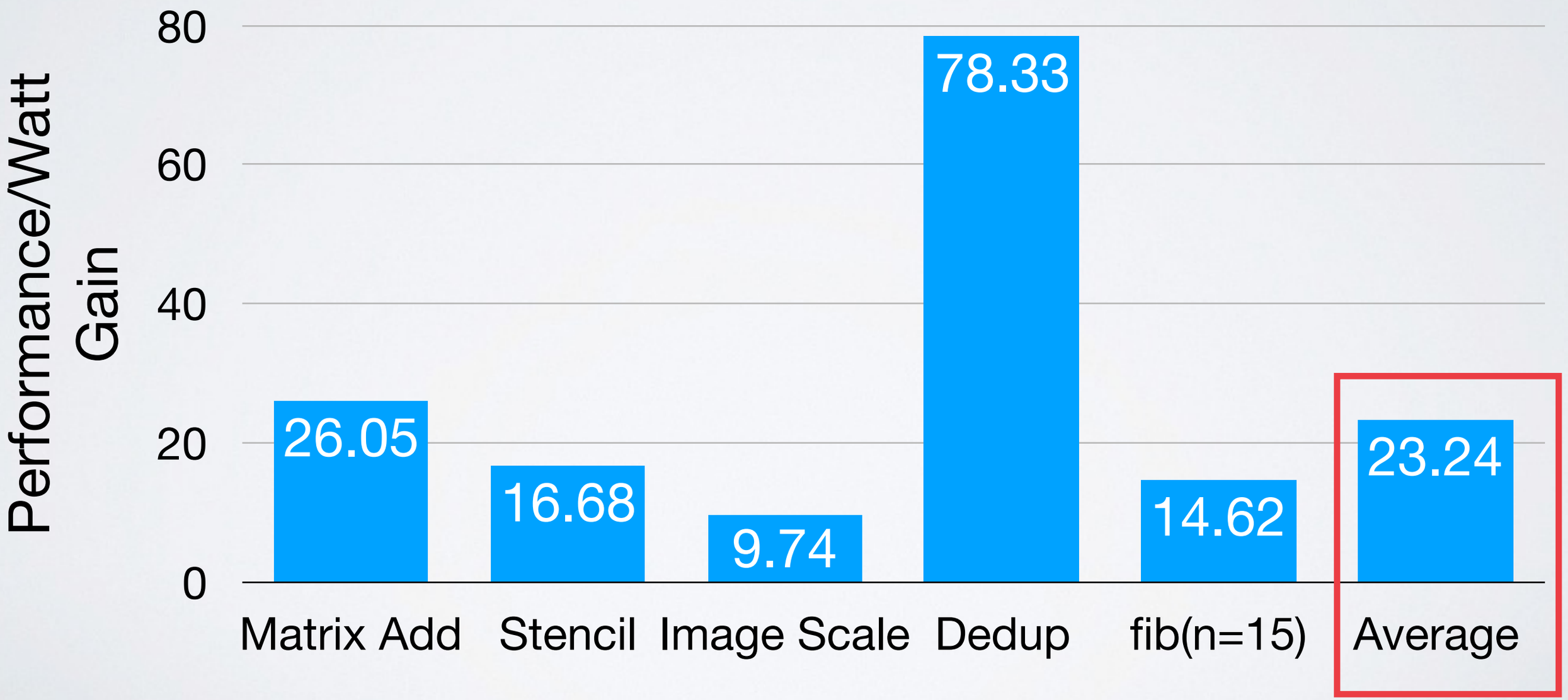
How does performance compare to CPU?

- Performance gain compare to a Intel core i7



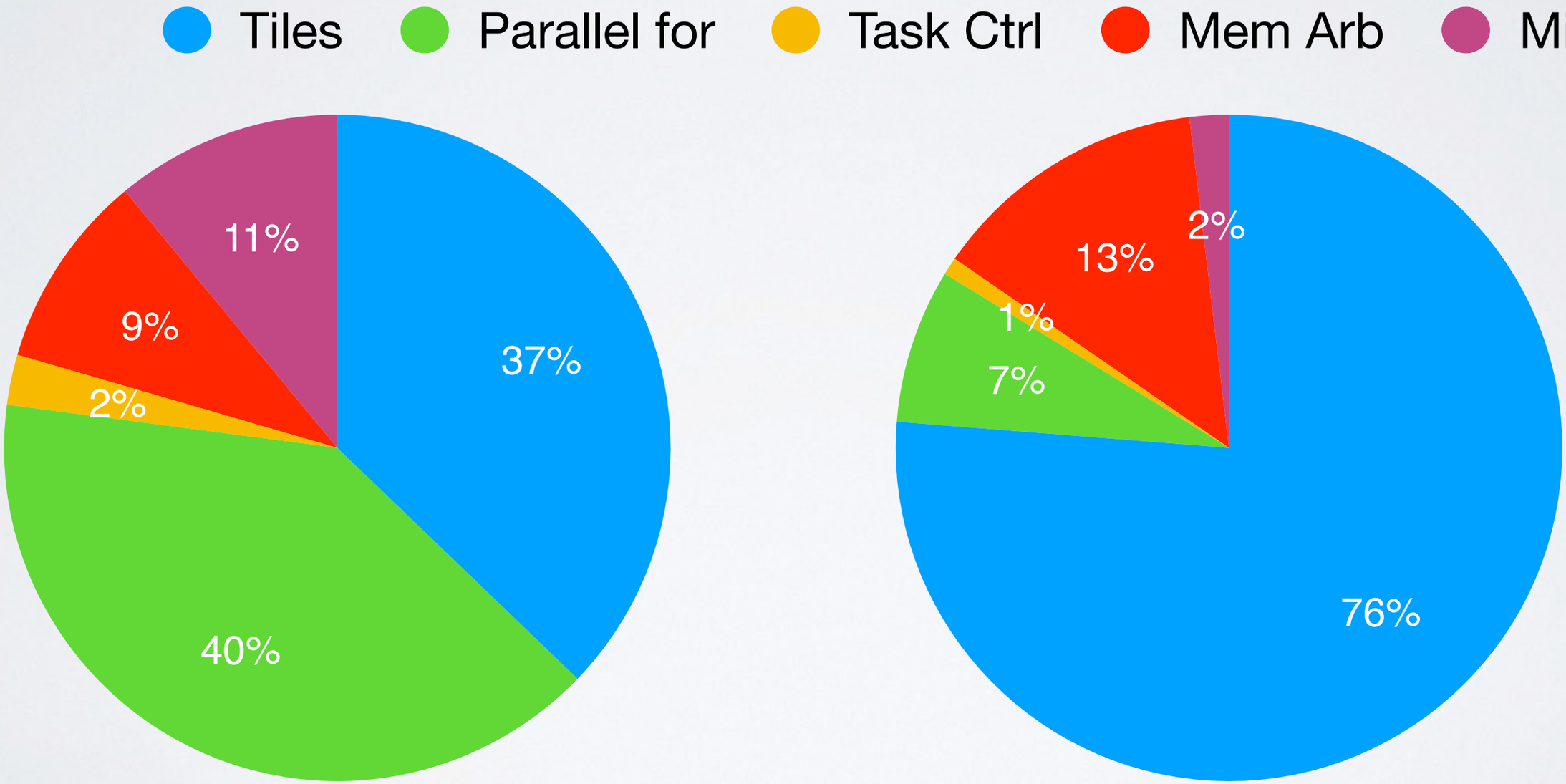
How does Performance/Watt compare to CPU?

- Performance/Watt has significant improvement



What is the overhead of task controller?

- ALM Utilization by Sub-block



1 Tile / 1 Ins

10 Tile / 1 Ins

Available now

https://github.com/sfu-arch/tapas

Thanks **Chisel** and **Tapir folks**

Shout out to related…

- An Architectural Framework for Accelerating Dynamic Parallel Algorithms on Reconfigurable Hardware **(MICRO51)**
- Dynamically scheduled high-level synthesis **(FPGA18`)**

Parametrization and Configuration

- TAPAS generated accelerator is *Parametrizable and Configurable*.
  - The number of TXUs can be set specifically for each task base on different criteria.
  - Datapath width can be set at this phase, supporting **mixed precision** as well.
  - Memory modules within each Task Unit are configurable like scratchpads, network and cache.