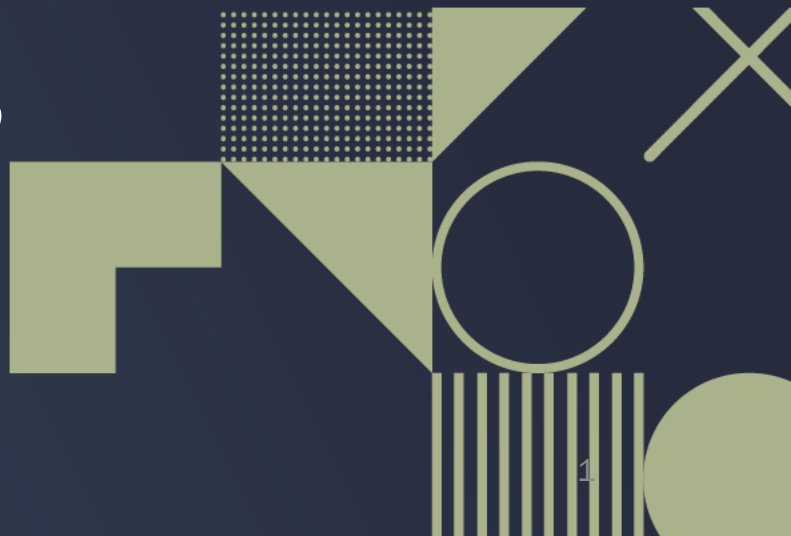
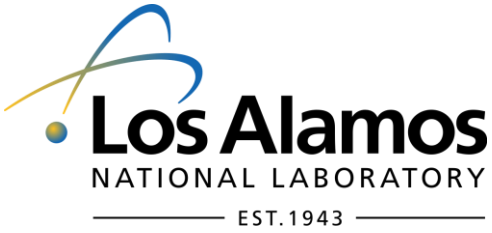
Index Launches: Scalable, Flexible Representation of Parallel Task Groups

Soi, Bauer, Treichler, Papadakis,
Lee, McCormick, Aiken, Slaughter

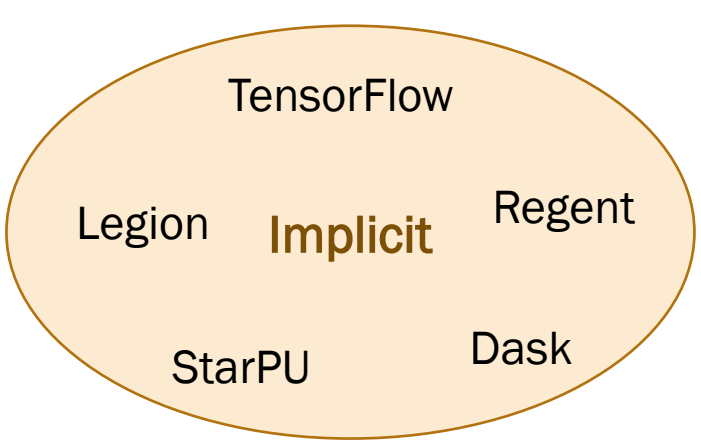


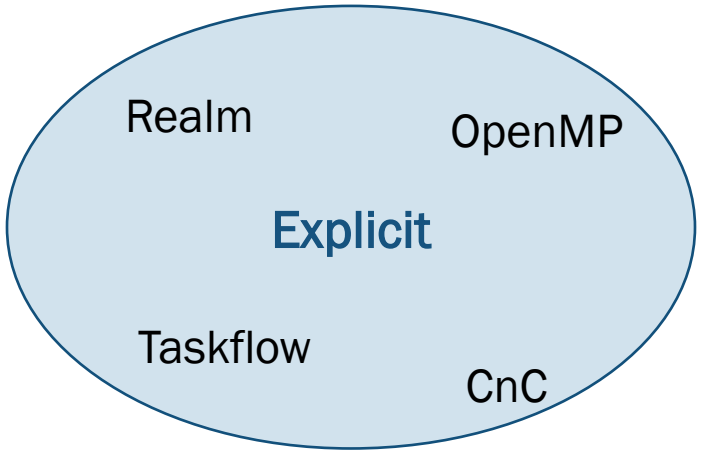
Index Launches: Scalable, Flexible Representation of Parallel Task Groups

Soi, Bauer, Treichler, Papadakis, Lee, McCormick, Aiken, Slaughter

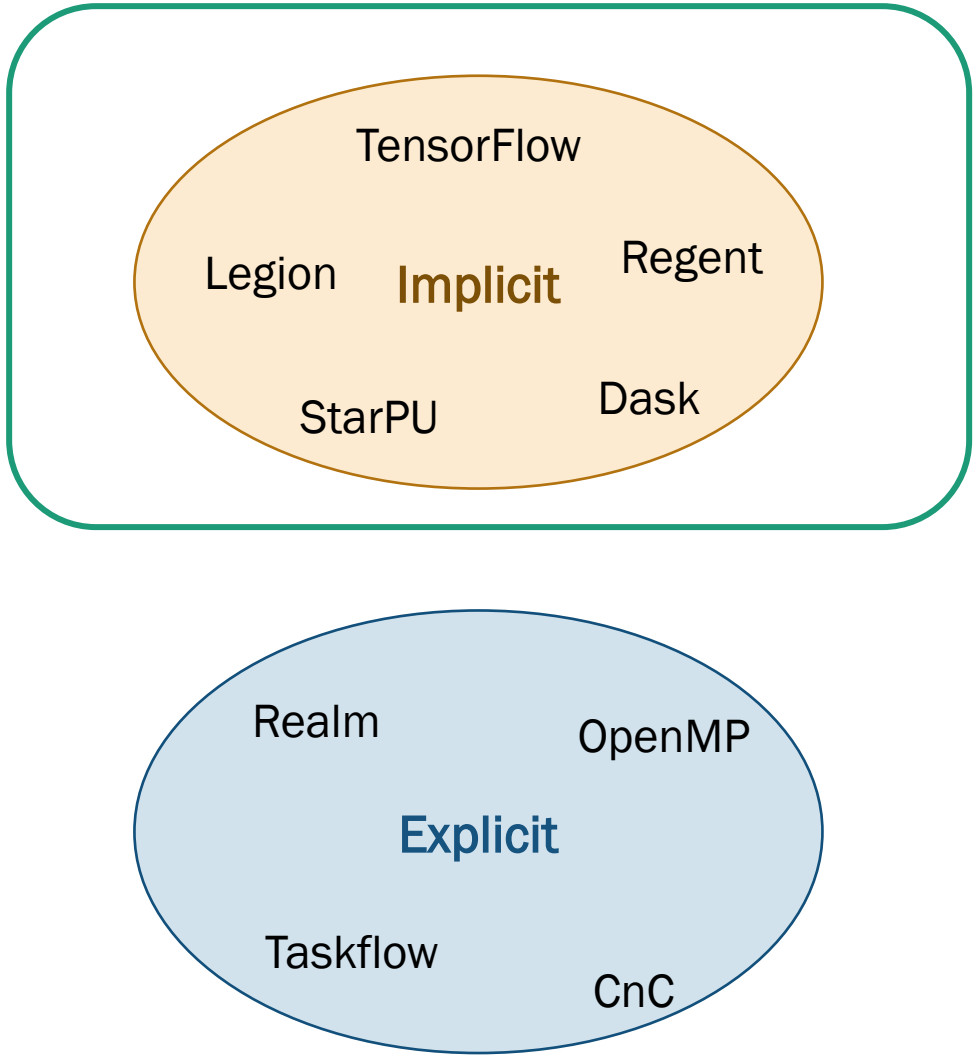


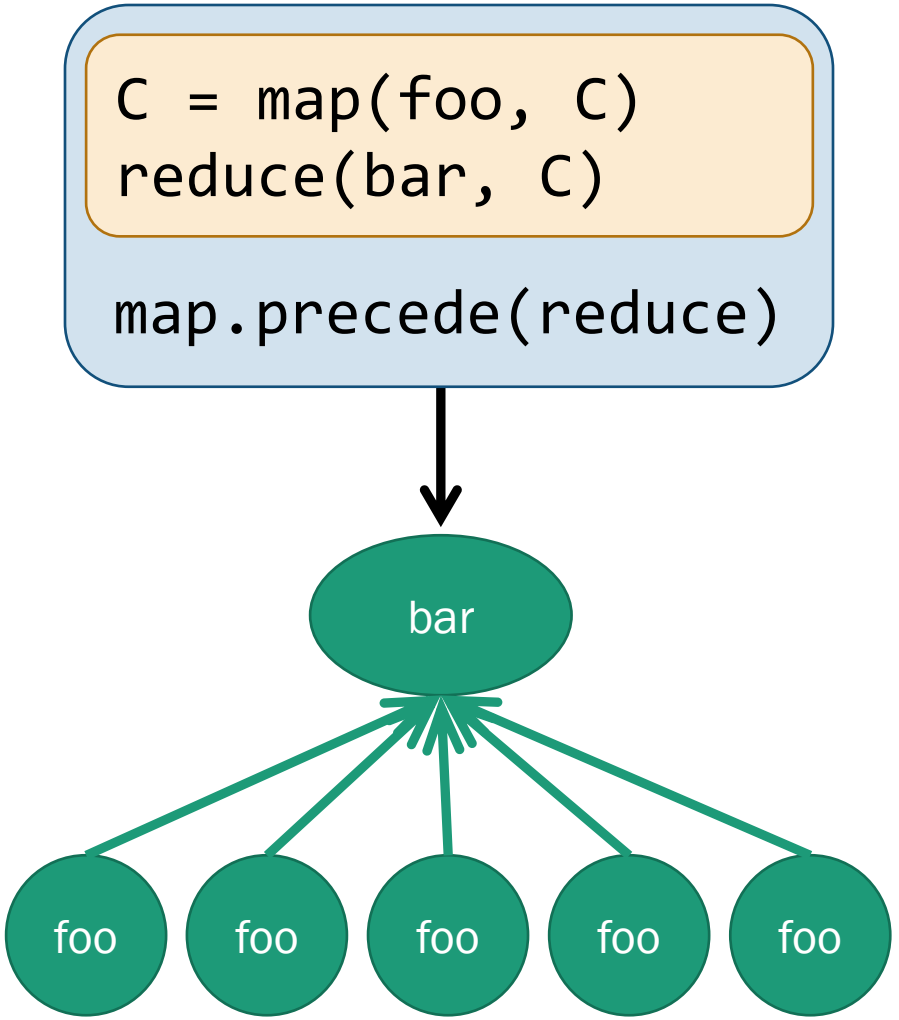
Task-Based Programming Systems





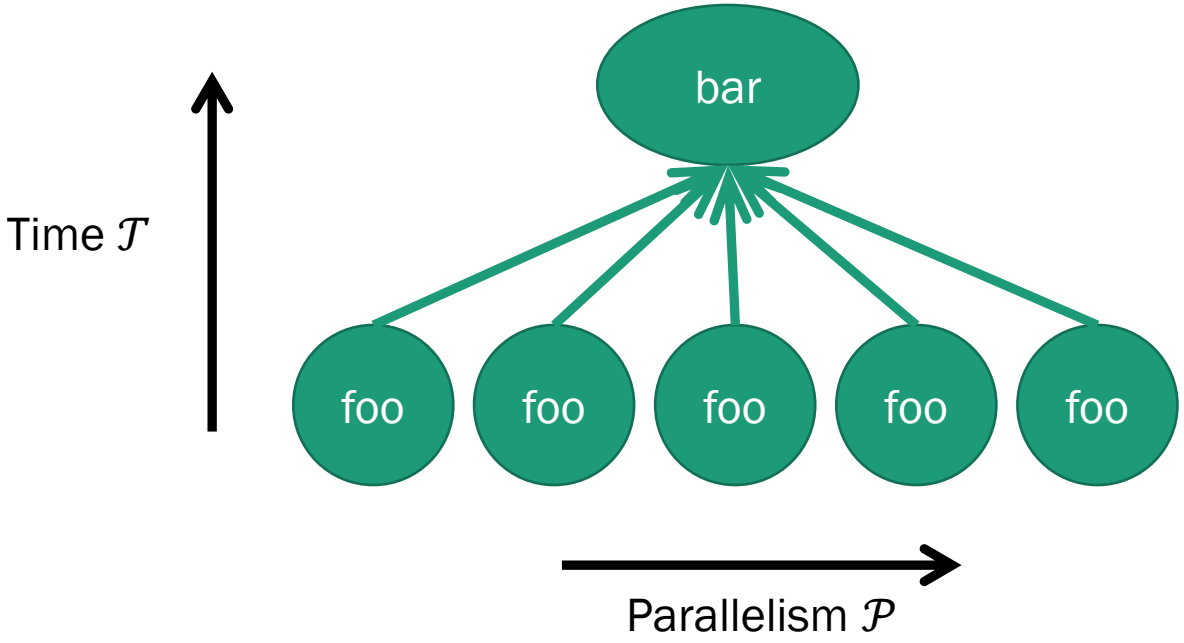
Task-Based Programming Systems





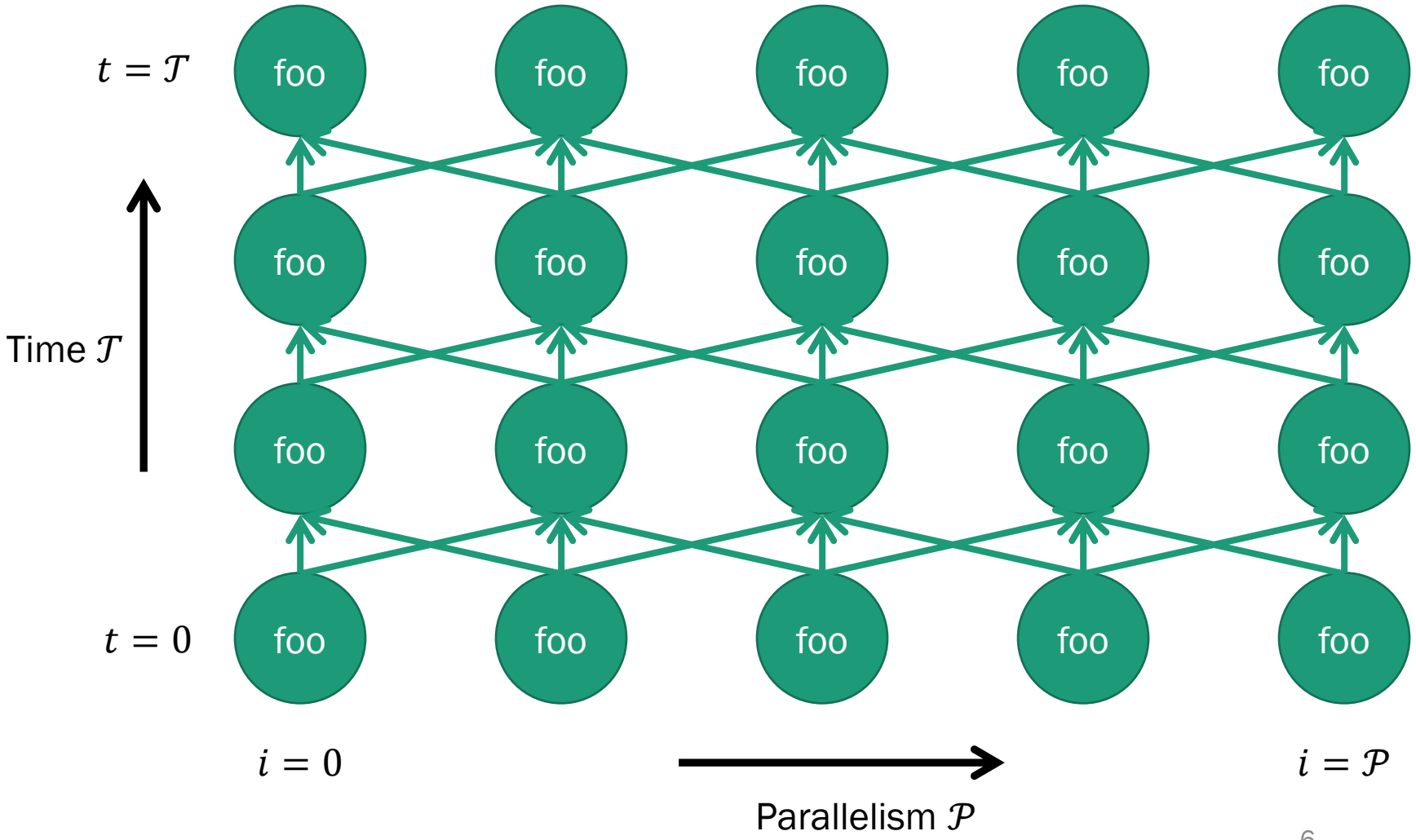
Task Graphs

- Typically, a DAG representing dependences between tasks
- An efficient representation is essential
  - Especially for implicit systems



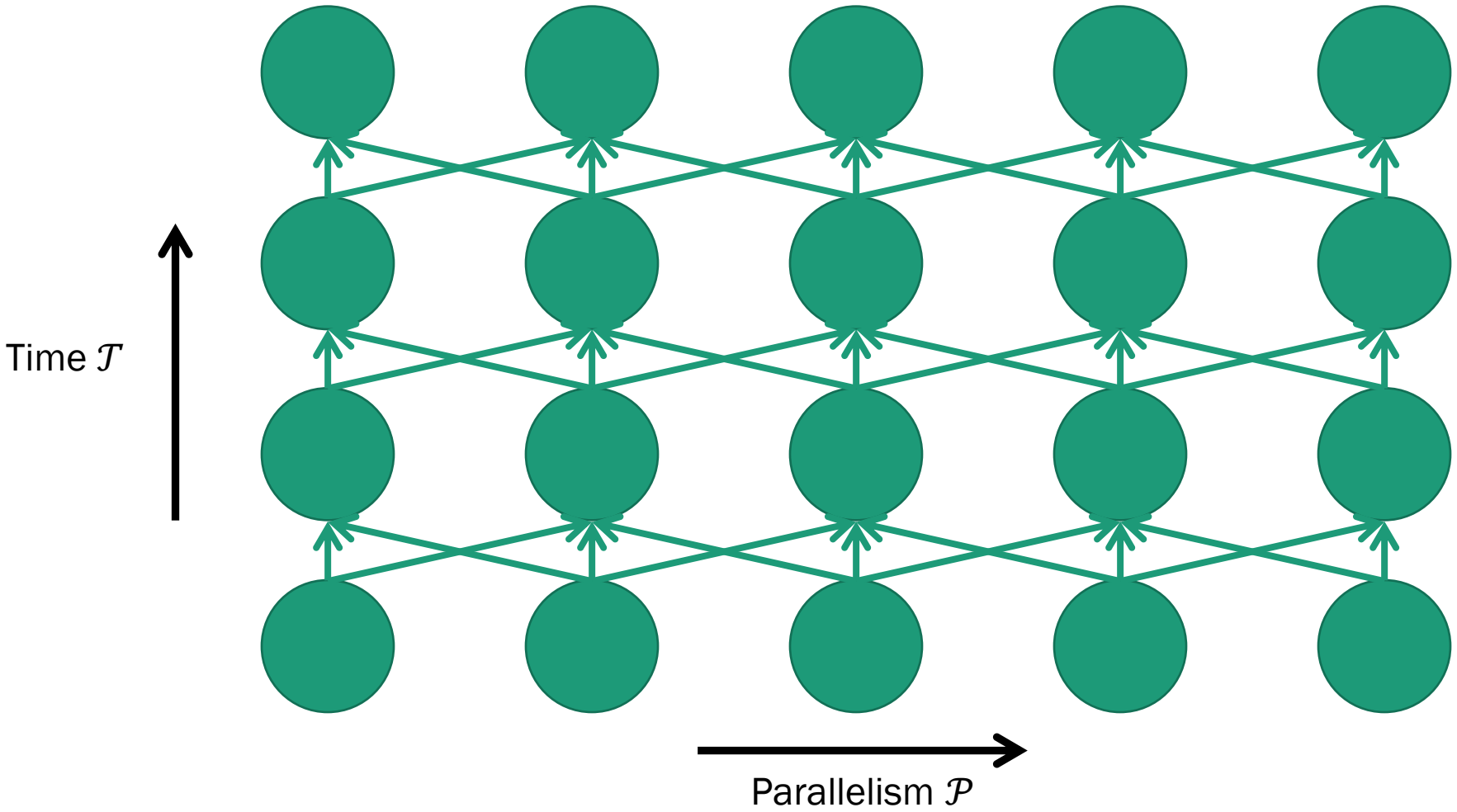
Task Graph Example: Stencil

```
for t = 0, T do
  for i = 0, P do
    foo(C[i-1],
        C[i],
        C[i+1],
        D[i])
  end
  swap(C, D)
end
```



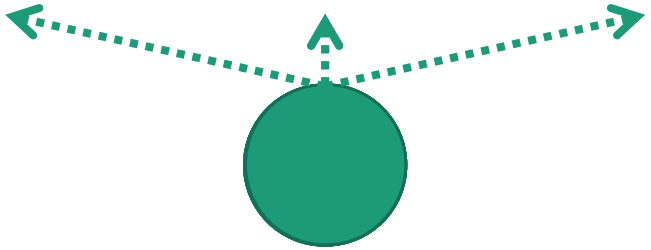
Naïve Representation: $\mathcal{O}(\mathcal{P}\mathcal{T})$

10^6 tasks/sec in large applications

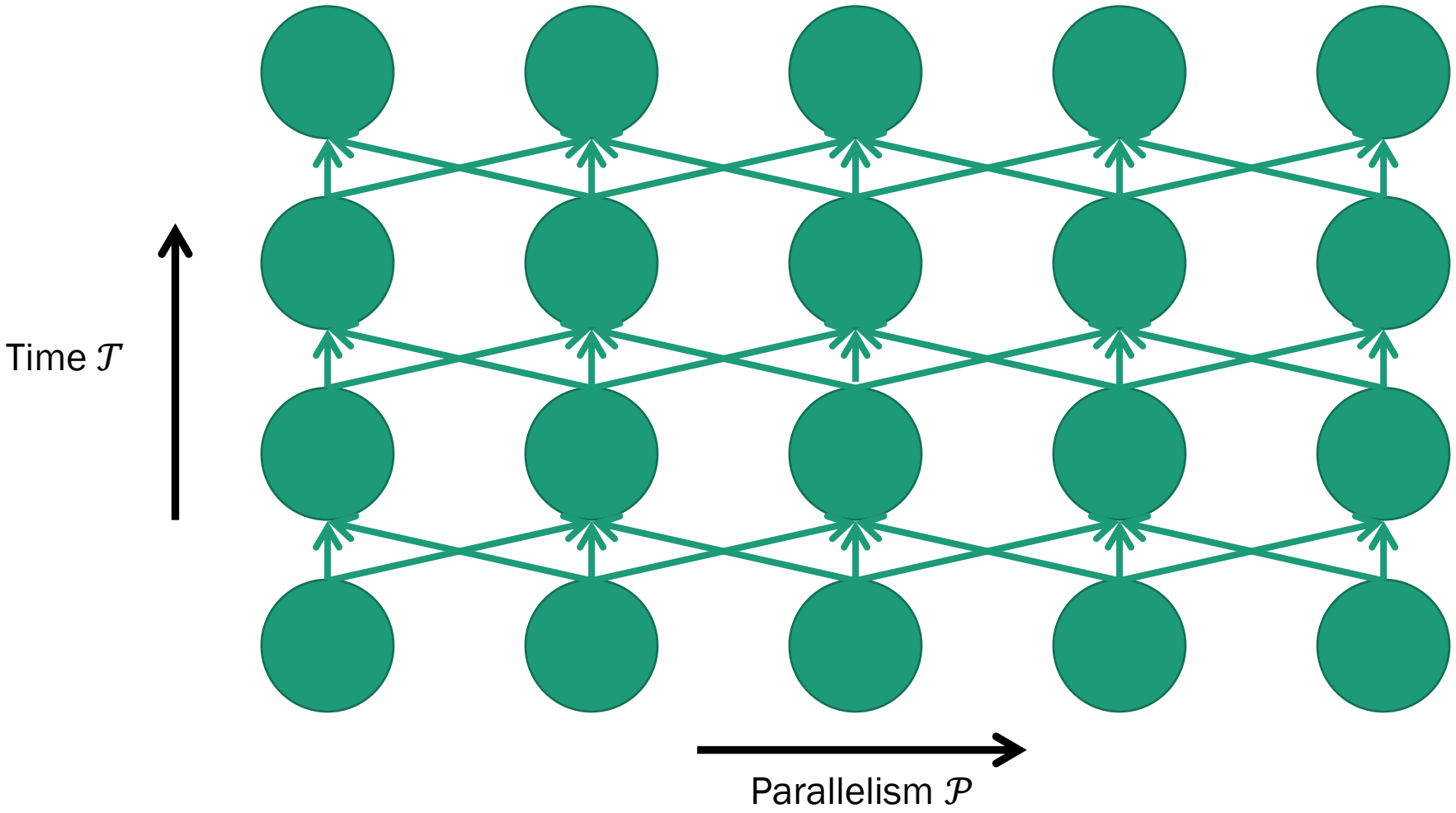


Collapsing Time & Parallelism: $\mathcal{O}(1)$

- PaRSEC PTG captures a static, algebraic task graph
- Maximally dense, but limited expressiveness
  - Cannot express dynamic task generation

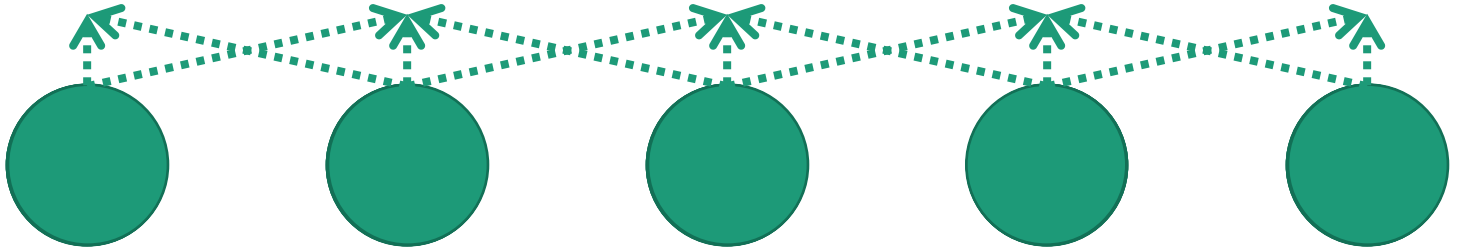


Naïve Representation: $\mathcal{O}(\mathcal{P}\mathcal{T})$

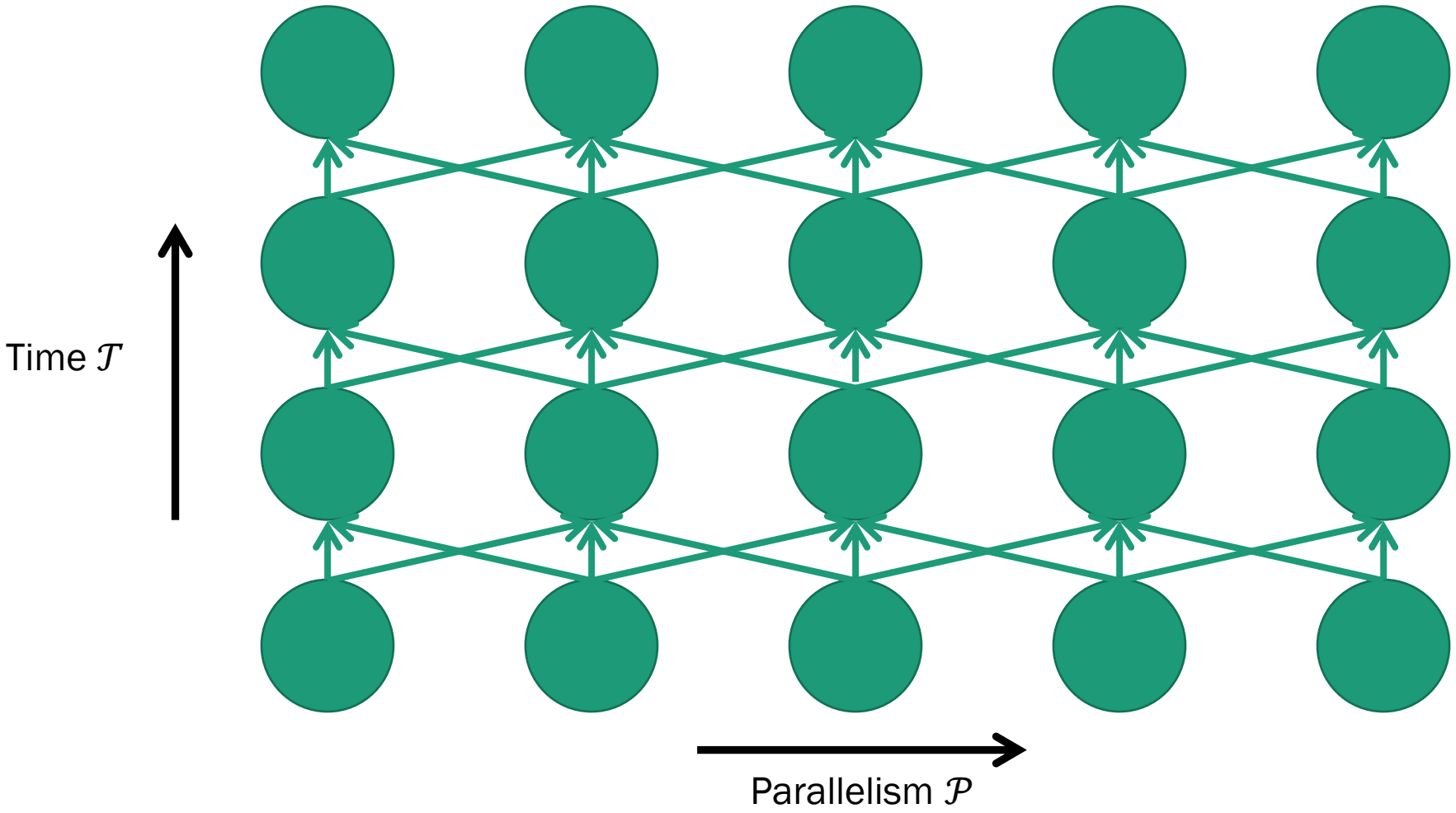


Collapsing Time: $\mathcal{O}(\mathcal{P})$

- TensorFlow represents control-flow in-graph
- Runtime needs to see the entire graph at once to collapse it. Impossible if:
  - the program is submitted incrementally, or
  - its control flow is not understood by the compiler

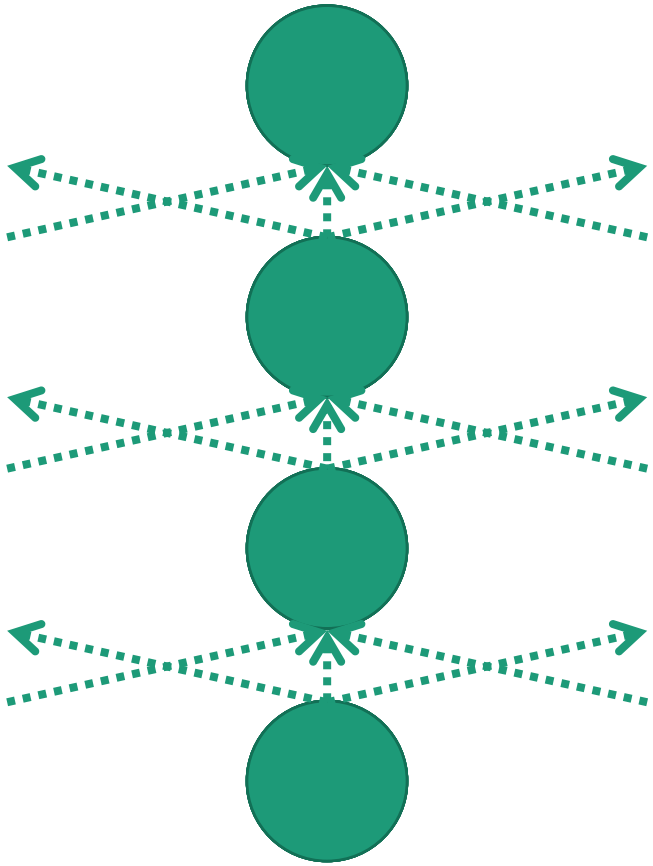


Naïve Representation: $\mathcal{O}(\mathcal{P}\mathcal{T})$



Collapsing Parallelism: $\mathcal{O}(\mathcal{T})$

- Our solution: Index Launches
- Programs benefit from the reduced runtime overhead
- We support arbitrary and dynamic control flow
- Implementation with Legion and Regent



Each node is now an index launch

Data Model

Data Model: Collections

- Any data structure containing homogeneous objects
- Often indexed in N-D
- The primary way to pass large data to tasks

(v_x, v_y)	(v_x, v_y)	(v_x, v_y)	(v_x, v_y)
(v_x, v_y)	(v_x, v_y)	(v_x, v_y)	(v_x, v_y)
(v_x, v_y)	(v_x, v_y)	(v_x, v_y)	(v_x, v_y)
(v_x, v_y)	(v_x, v_y)	(v_x, v_y)	(v_x, v_y)

Data Model: Partitions

- Dividing a collection into sub-collections
  - Tasks that access disjoint data can run in parallel
- Can be *disjoint* or *aliased*
  - Unlike mathematical partitions, which are disjoint by definition
- Exact method is unimportant for this talk

$|P| = 4$

(v_x, v_y)	(v_x, v_y)	(v_x, v_y)	(v_x, v_y)
(v_x, v_y)	(v_x, v_y)	(v_x, v_y)	(v_x, v_y)
(v_x, v_y)	(v_x, v_y)	(v_x, v_y)	(v_x, v_y)
(v_x, v_y)	(v_x, v_y)	(v_x, v_y)	(v_x, v_y)

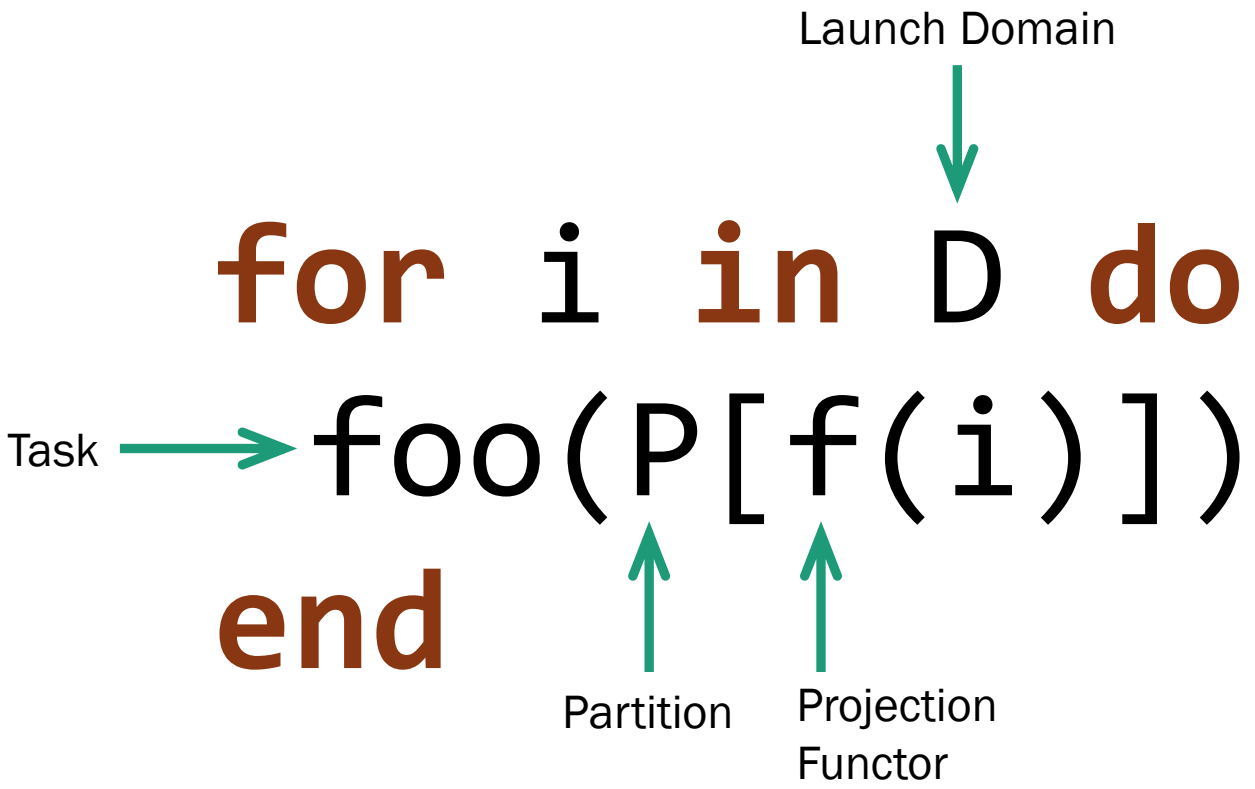
Data Model: Privileges

- Tasks declare privileges on collections they receive as input
- Privileges include read, write, and reduce (+, *, min, …)
- Required by the system for dependence analysis

```
task foo(C, D) reads(C) writes(D)
do
   -- computation
end
```

Index Launches

Elements of an Index Launch



When is an Index Launch Safe?

- When all tasks in the launch are safe to run in parallel
- Below: assume P is disjoint and *foo* has write privileges

```
for i in D do
  foo(P[0])
end
```

Unsafe, because two different tasks may write to P[0]

```
for i in D do
  foo(P[i])
end
```

Safe

Compiler Guarantees Safety

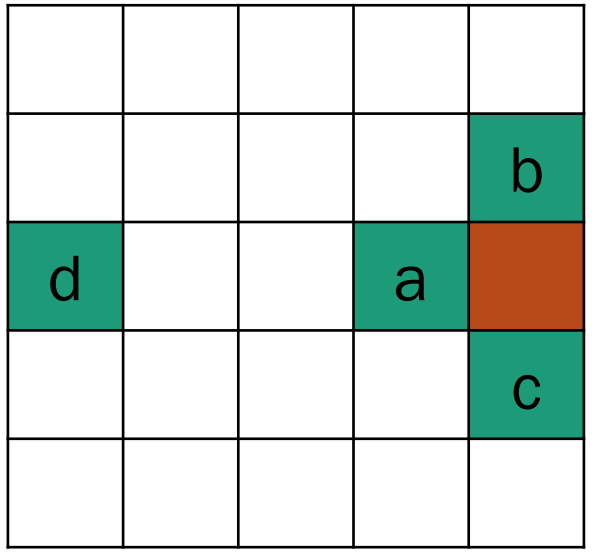
- Safety depends on:
  - Disjointess of the partition
  - Privileges of the task
  - Projection functor
- The programming model can facilitate the static analysis of privileges and disjointness:
  - Task with only read privileges on any partition is safe to index launch
  - Task with write privileges on an aliased partition is unsafe to index launch

Most common: disjoint partition with write privileges

The Problem of Projection Functors

- Can be arbitrary pure functions
- Hard to analyse statically

$$f_a(x, y) = (x - 1, y) \pmod N$$
$$f_b(x, y) = (x, y + 1) \pmod N$$
$$f_c(x, y) = (x, y - 1) \pmod N$$
$$f_d(x, y) = (x + 1, y) \pmod N$$



Periodic Launch Domain

Static and Dynamic Analysis for Projection Functors

- If the task has write privileges, then the projection functor must not assign two different calls the same data
  - Must be injective over the launch domain
- Static analysis for easy, common cases (e.g. identity)
- Dynamic analysis otherwise
  - Iterate over the launch domain and check for duplicate values
  - Precondition: the partition must be disjoint

Dynamic Analysis Algorithm

1. Create a bitmask as large as $|P|$
2. Iterate over D and set the bitmask at each value of f
3. Exit if a duplicate is found

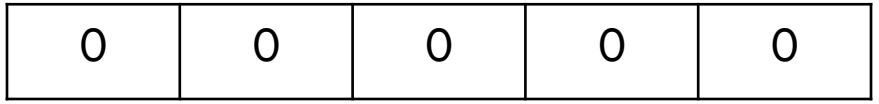
Our analysis works at the granularity of sub-collections

$$D = [0, 4]$$

$$|P| = 5$$

$$f(i) = i \pmod 3$$

Bitmask



Dynamic Analysis Algorithm

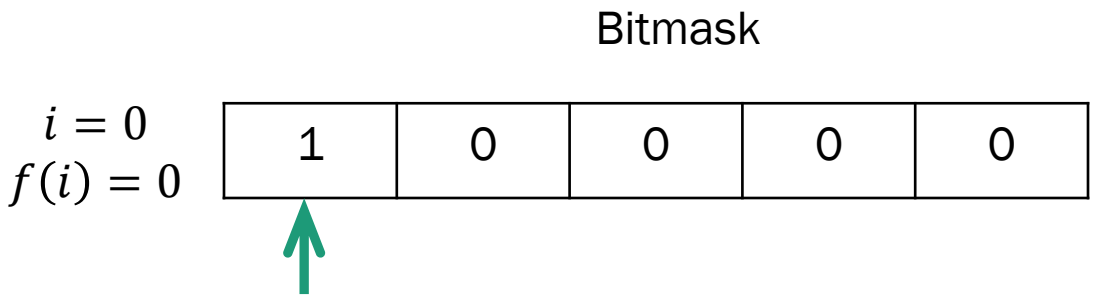
1. Create a bitmask as large as $|P|$
2. Iterate over D and set the bitmask at each value of f
3. Exit if a duplicate is found

Our analysis works at the granularity of sub-collections

$$D = [0, 4]$$

$$|P| = 5$$

$$f(i) = i \pmod 3$$



Dynamic Analysis Algorithm

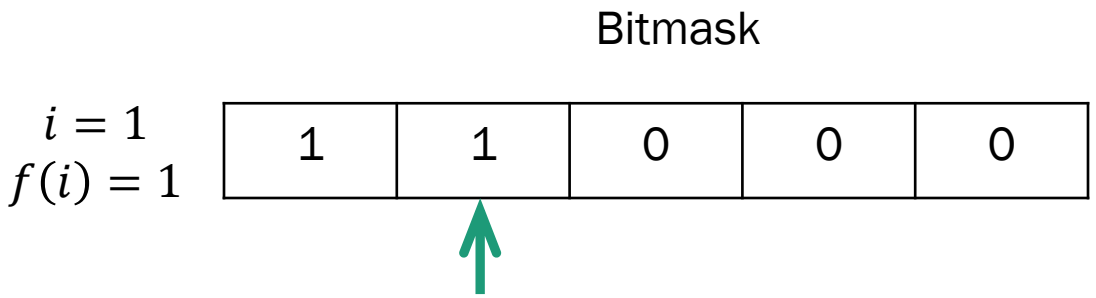
1. Create a bitmask as large as $|P|$
2. Iterate over D and set the bitmask at each value of f
3. Exit if a duplicate is found

Our analysis works at the granularity of sub-collections

$$D = [0, 4]$$

$$|P| = 5$$

$$f(i) = i \pmod 3$$



Dynamic Analysis Algorithm

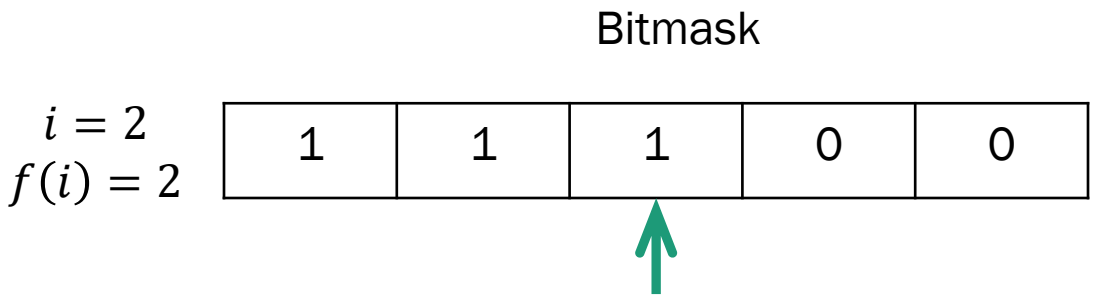
1. Create a bitmask as large as $|P|$
2. Iterate over D and set the bitmask at each value of f
3. Exit if a duplicate is found

Our analysis works at the granularity of sub-collections

$$D = [0, 4]$$

$$|P| = 5$$

$$f(i) = i \pmod 3$$



Dynamic Analysis Algorithm

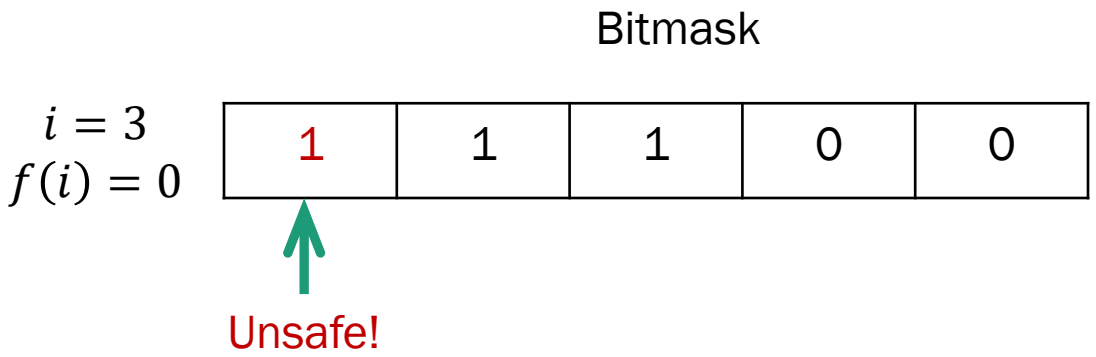
1. Create a bitmask as large as $|P|$
2. Iterate over D and set the bitmask at each value of f
3. Exit if a duplicate is found

Our analysis works at the granularity of sub-collections

$$D = [0, 4]$$

$$|P| = 5$$

$$f(i) = i \pmod 3$$

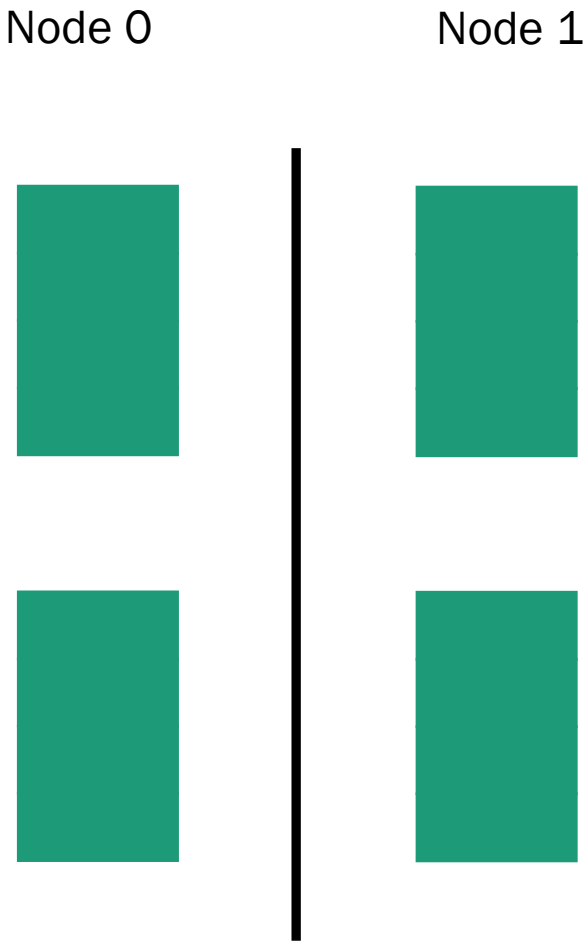


Runtime Implementation

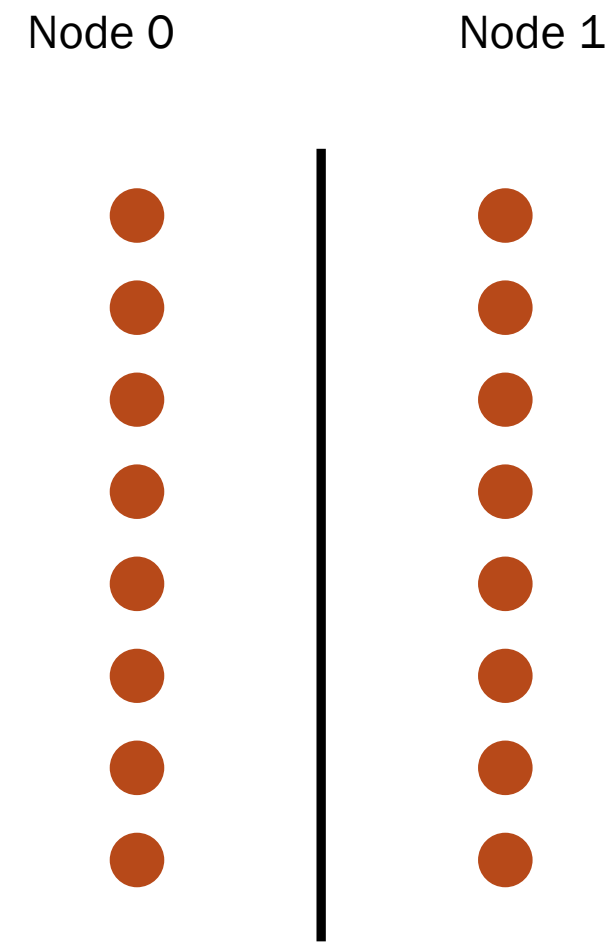
Legion Runtime

- Needs to efficiently execute index launches
- Two modes: this talk uses the newer, distributed mode
  - See Bauer PPoPP ’21
- Has an internal pipeline to process tasks

Pipeline Stage I: Task Issuance

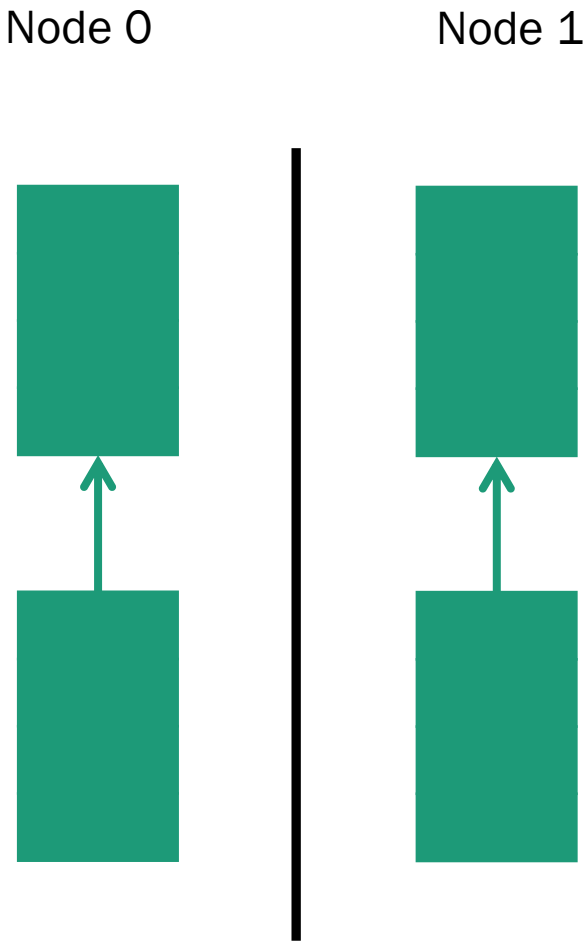


with Index Launches

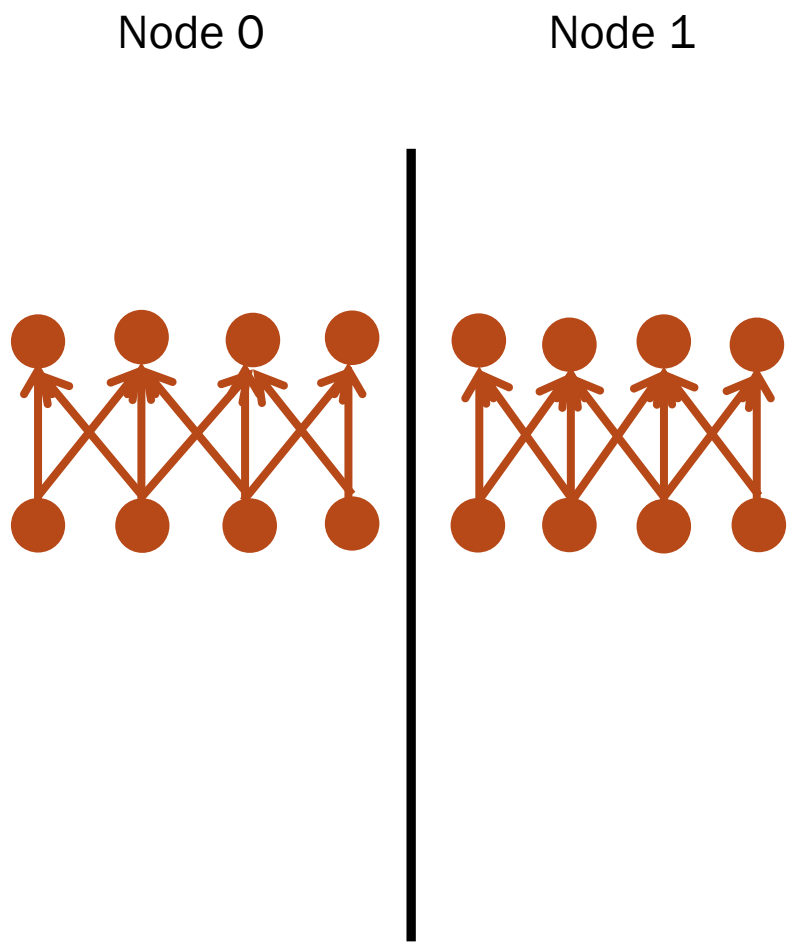


without Index Launches

Pipeline Stage II: Logical Analysis



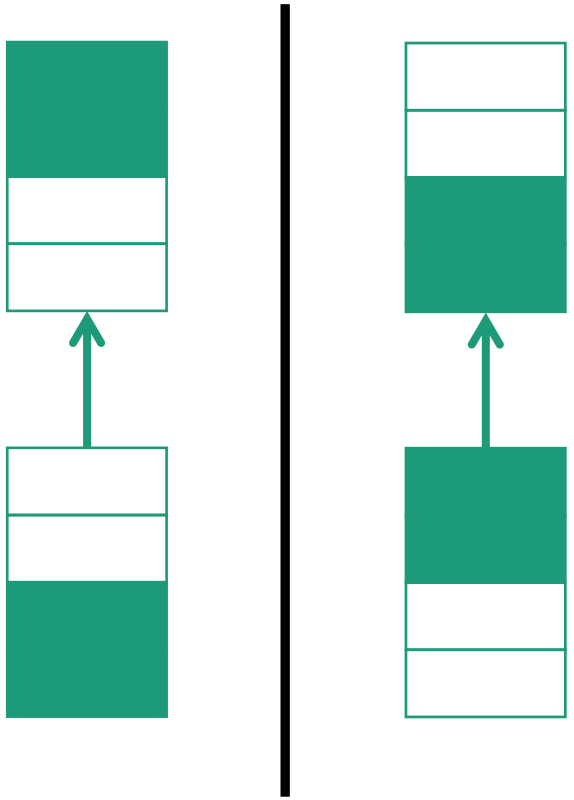
with Index Launches



without Index Launches

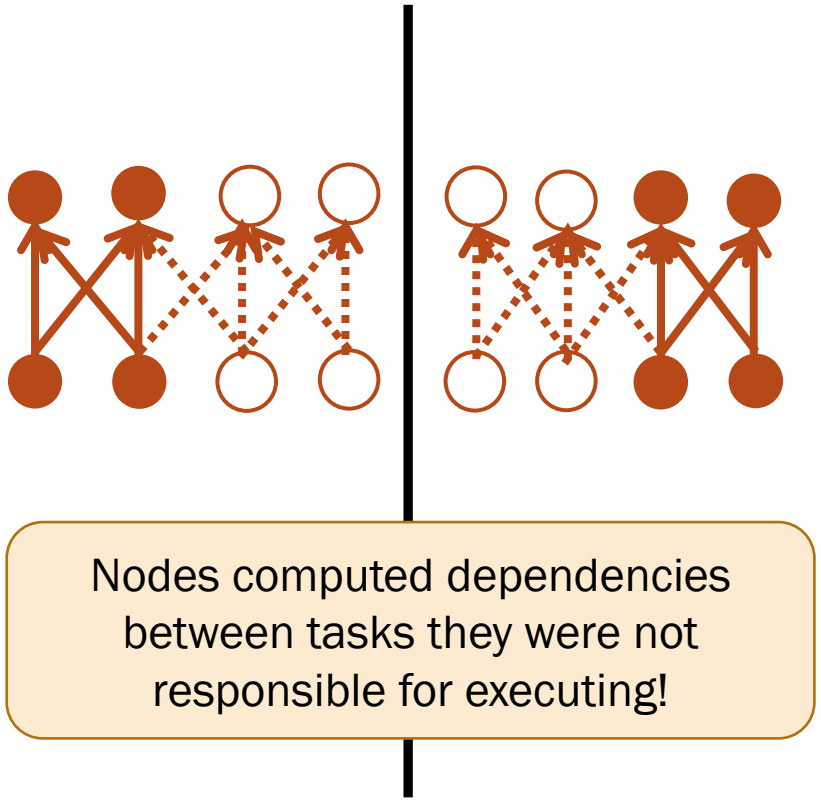
Pipeline Stage III: Distribution

Node 0 Node 1



with Index Launches

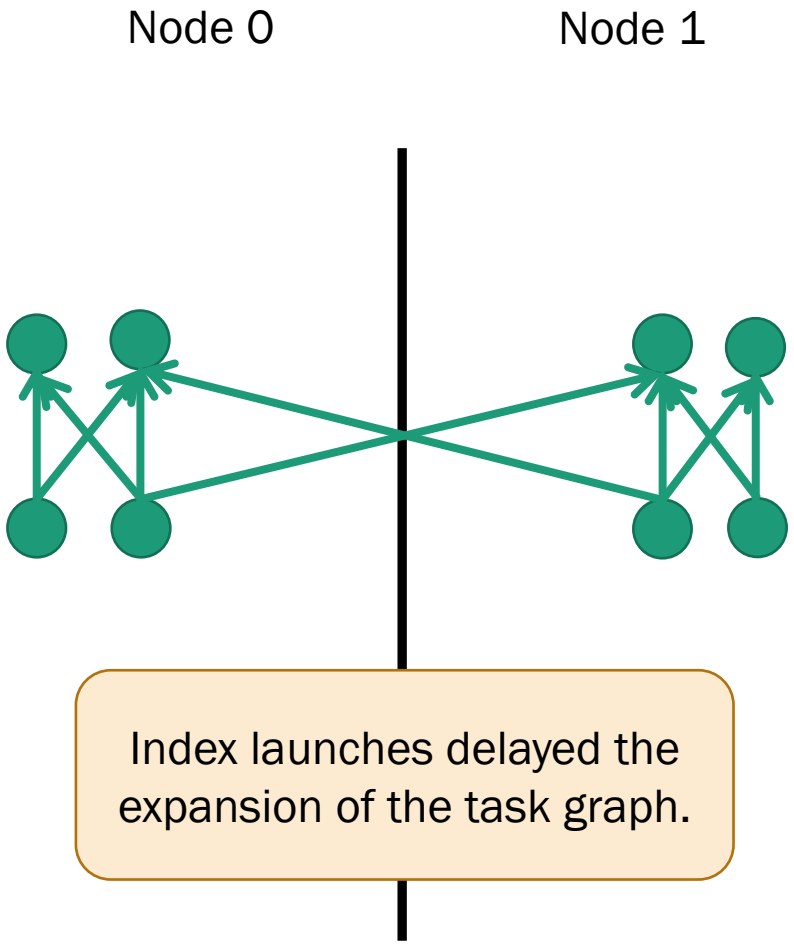
Node 0 Node 1



Nodes computed dependencies between tasks they were not responsible for executing!

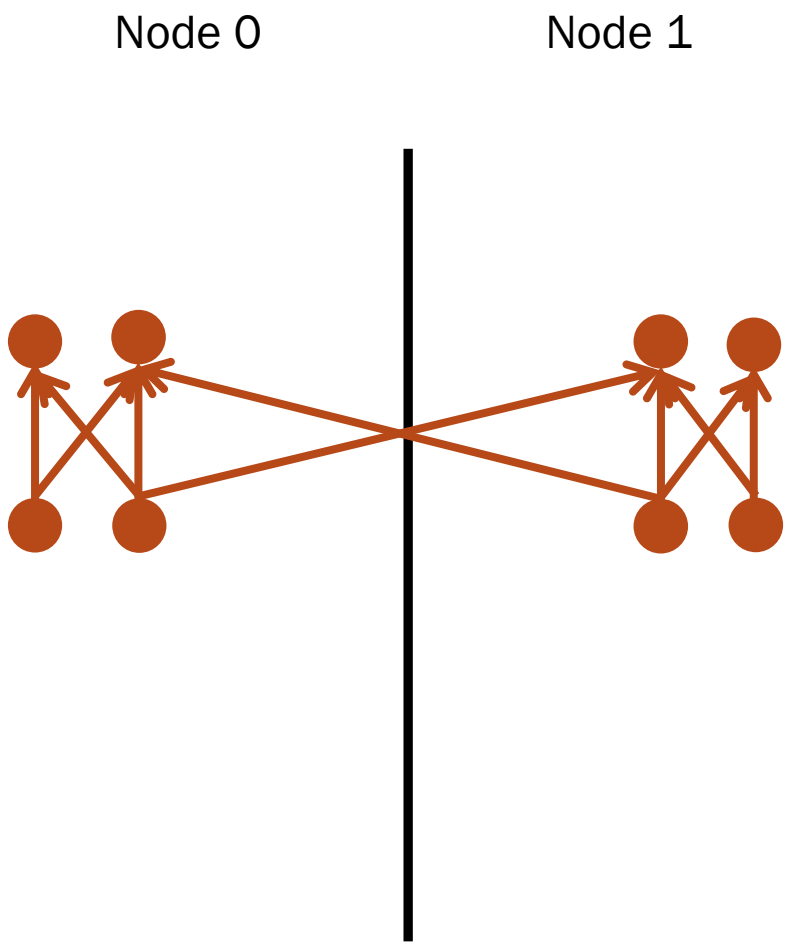
without Index Launches

Pipeline Stage IV: Physical Analysis



Index launches delayed the expansion of the task graph.

with Index Launches



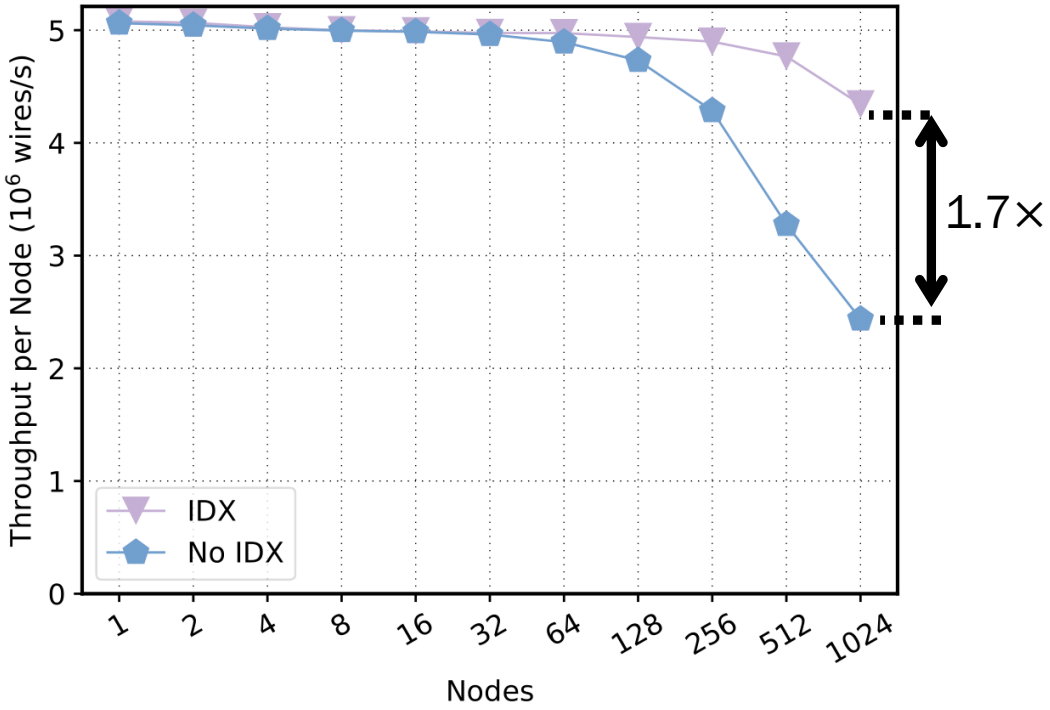
without Index Launches

Performance Evaluation

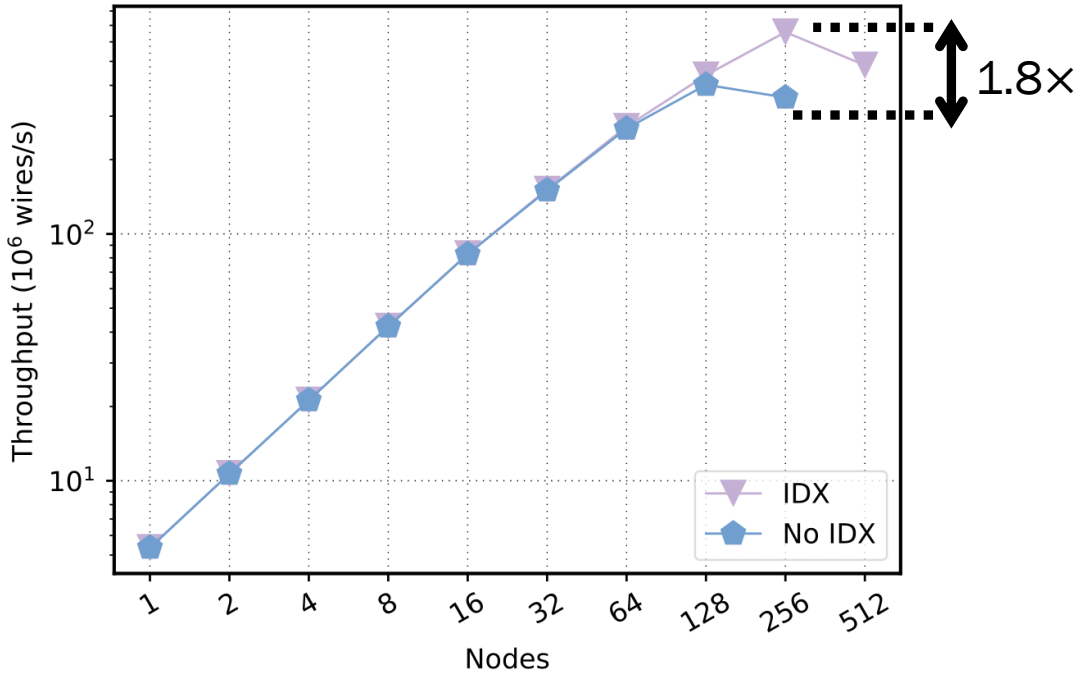
Experimental Setup: Piz Daint

Model	Cray XC50
CPU	Intel Xeon E5-2690 v3
GPU	NVIDIA Tesla P100
Interconnect	Cray Aries
Nodes	1024

Performance Evaluation: Circuit

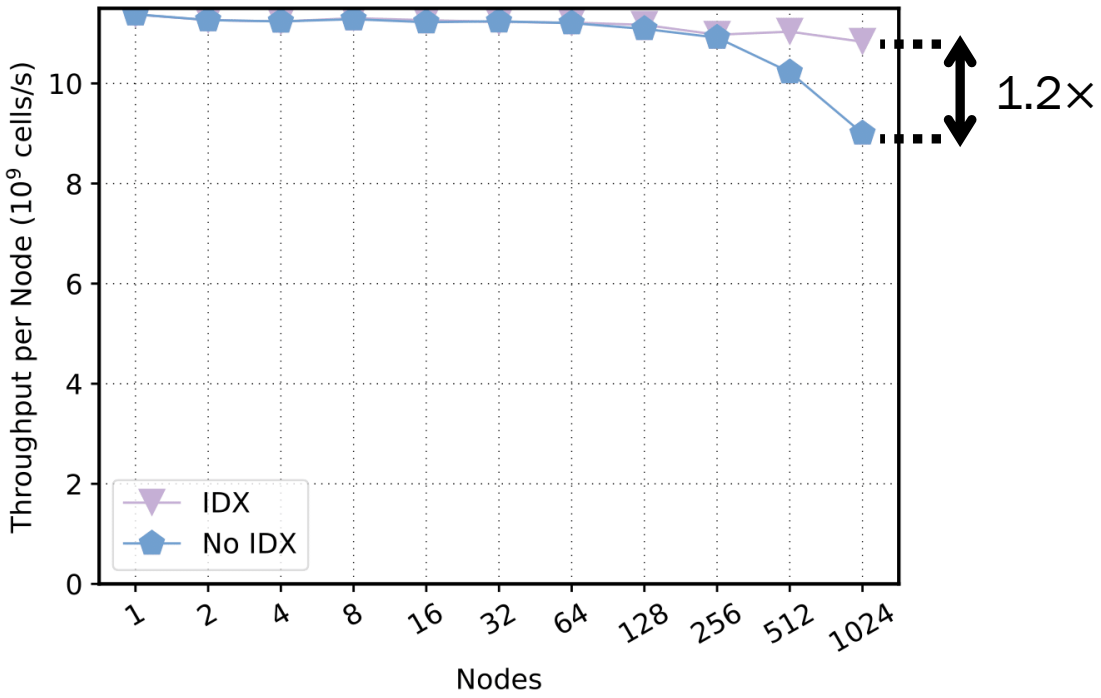


Weak Scaling

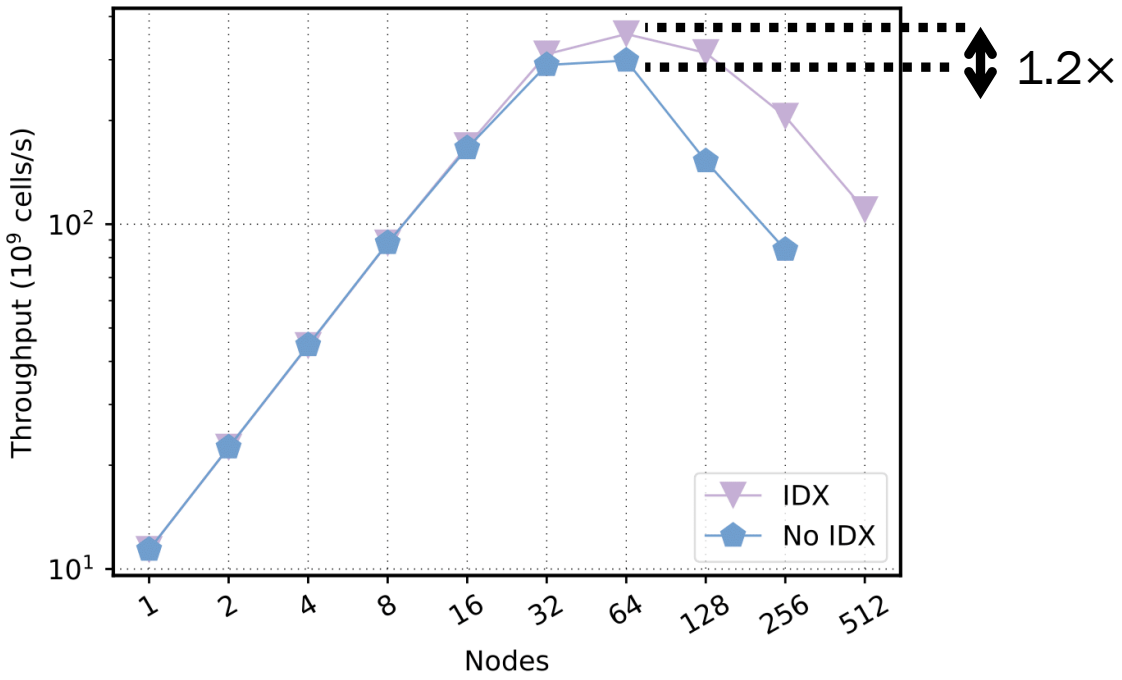


Strong Scaling

Performance Evaluation: Stencil

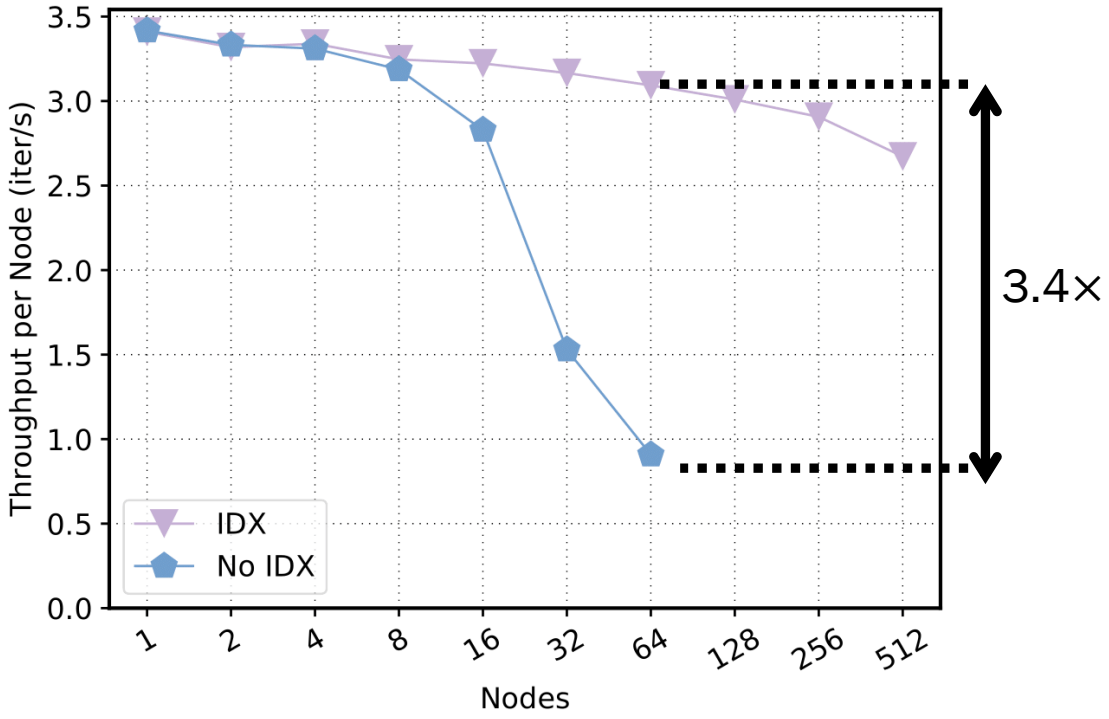


Weak Scaling

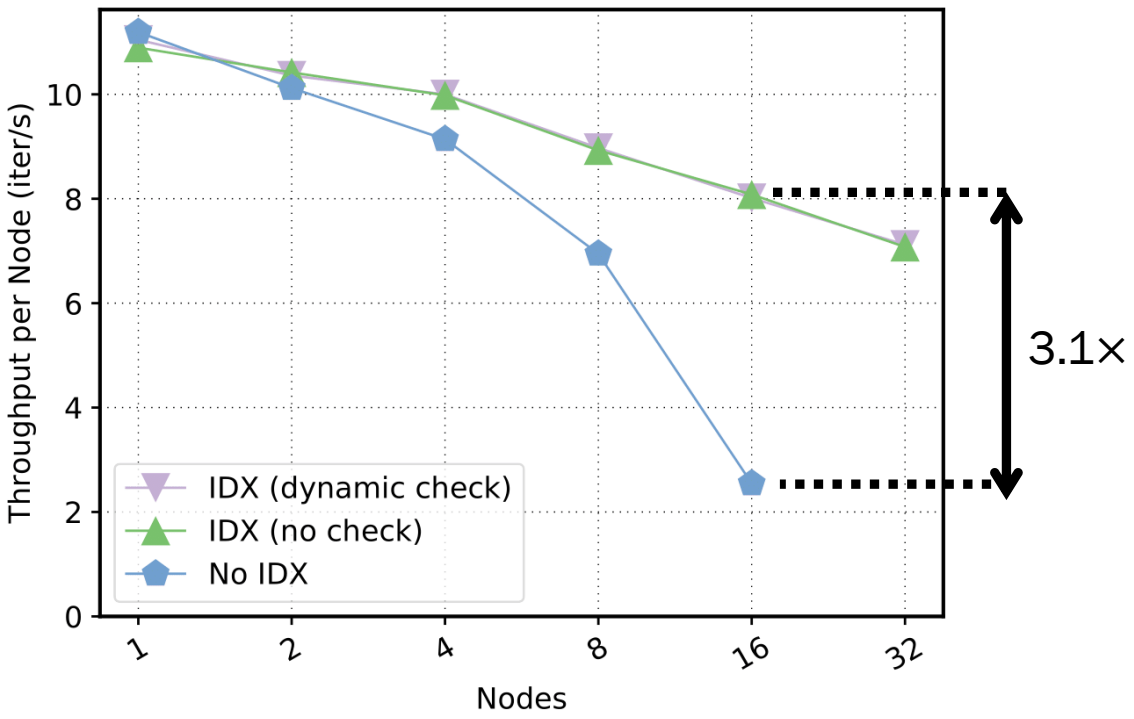


Strong Scaling

Performance Evaluation: Soleil-X



Weak Scaling, Fluid only



Weak Scaling, Fluid + Particle + DOM

Contributions

- Classification of task-based systems
  - Based on how they collapse the task graph
- Index launches as a general technique
  - Main elements: task, partition, projection functor
- Implementation in Legion and Regent
  - Index launches delay the expansion of the task graph at runtime
- Performance evaluation
  - Upto 3× speed-ups in scientific applications

More in the Paper!

- More sophisticated dynamic analysis
- Legion's centralized mode of dependence analysis
- More experiments
- …

Related Work

- Legion (Bauer SC ’12)
  Regent (Slaughter SC ’15)
  DCR (Bauer PPoPP ’21)
- TensorFlow (Yu EuroSys ’18)
- Dask (Rocklin SciPy ’15)
- PaRSEC (Bosilca ’13)
- Taskflow (Huang TPDS ’21)

Acknowledgments

- This research was supported by the Exascale Computing Project (17- SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration and funding from Department of Energy Office of Science, Office of Advanced Scientific Computing Research under the guidance of Drs. Laura Biven and Hal Finkel.
- Experiments on Piz Daint were supported by the Swiss National Supercomputing Centre (CSCS) under project ID d108.

Learn more at

legion.stanford.edu