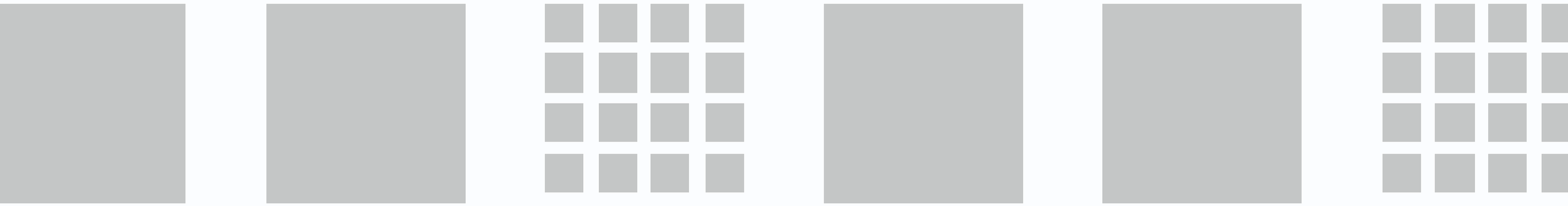
Liszt:

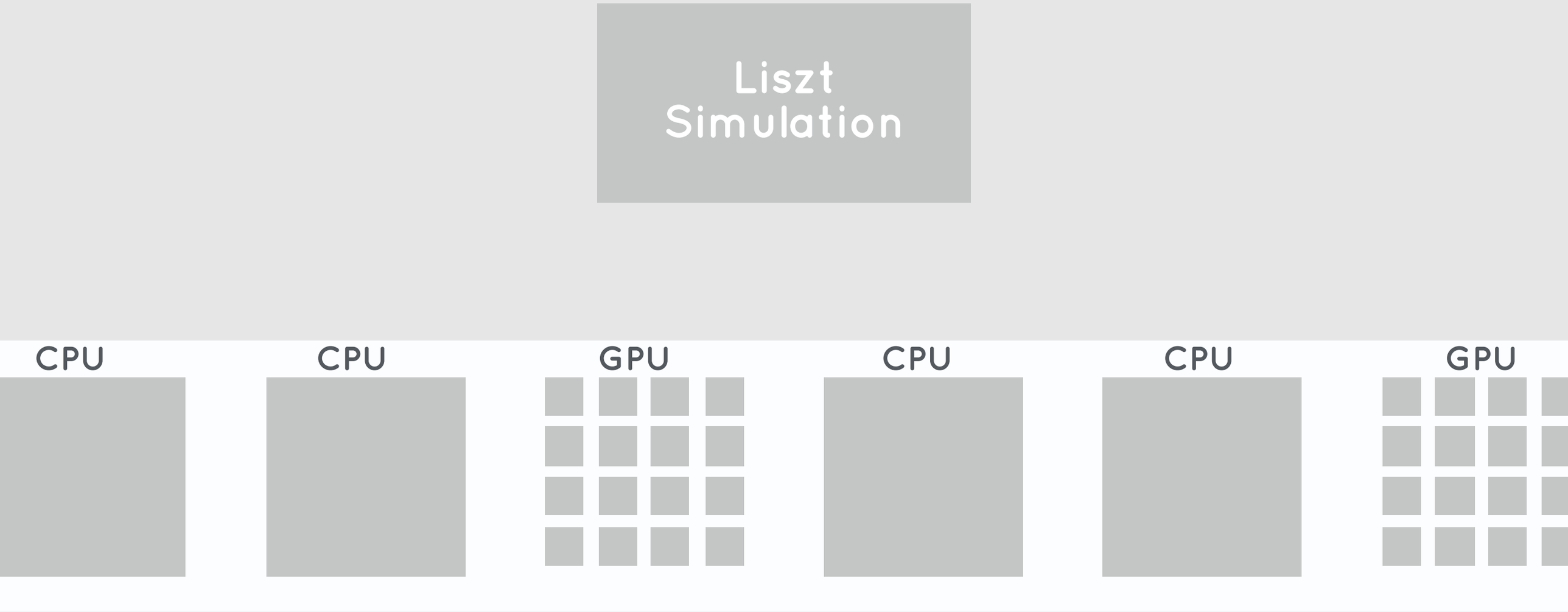
Running Parallel Simulations Across Heterogeneous Processors



talk: Chinmayee Shah

work with: Gilbert Bernstein, Zach Devito, Phil Levis, Pat Hanrahan

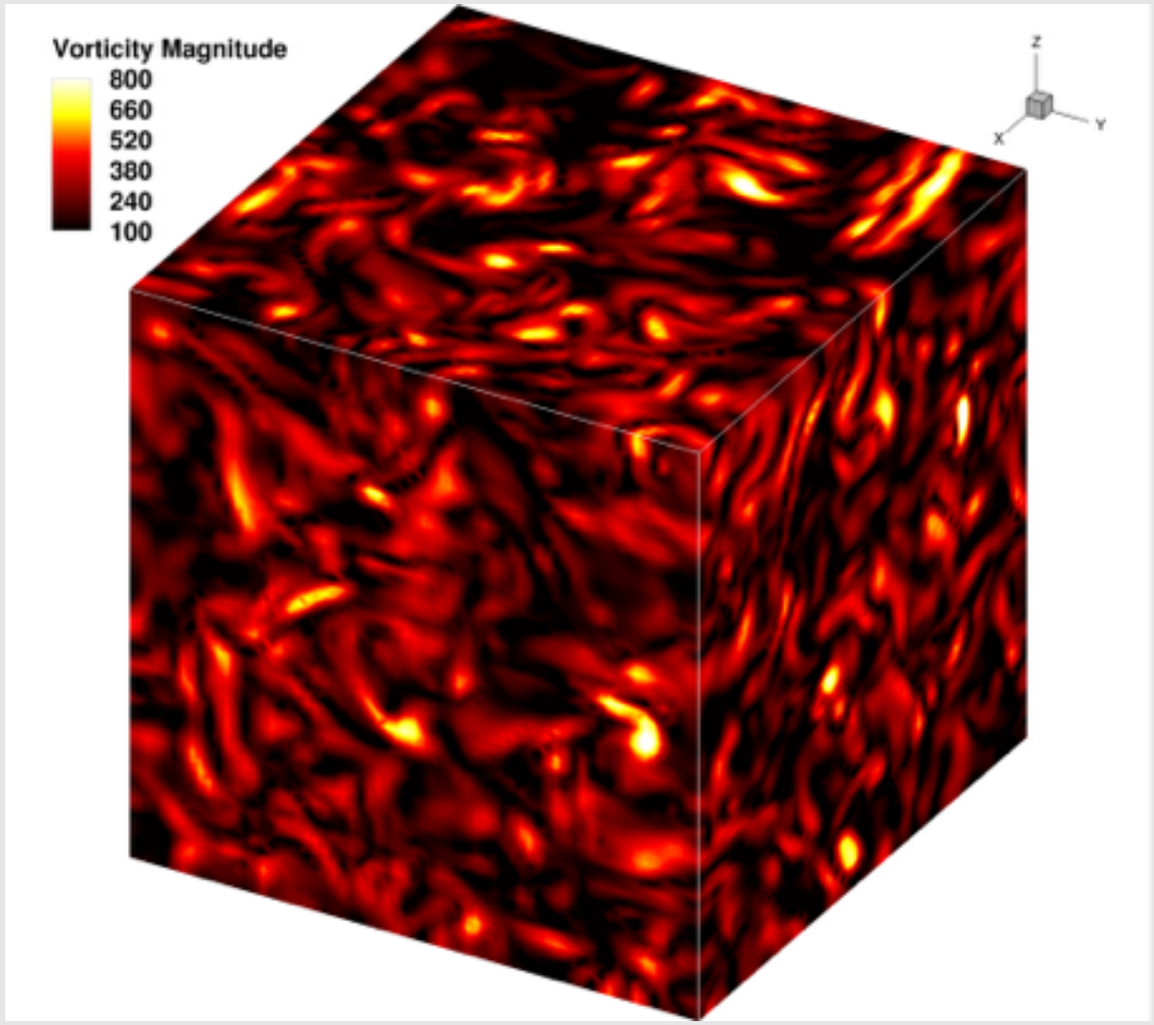
Liszt Makes Simulations Portable



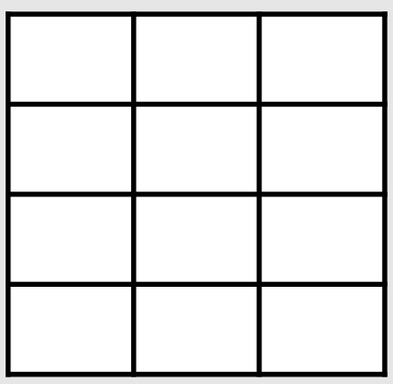
Soleil in Liszt

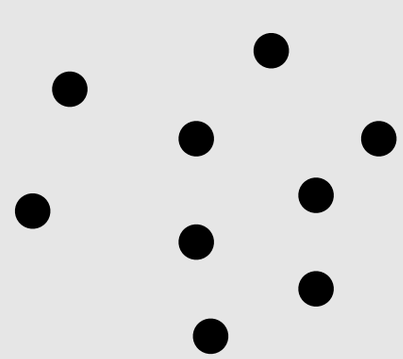
- 4000 lines of code
  - different configurations, IO, comments
  - similar reference codes in MPI over 10,000 lines

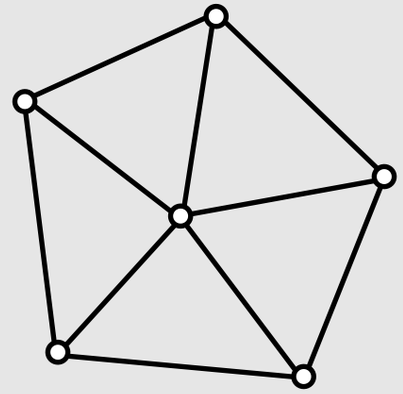
- Runs on CPUs & GPU



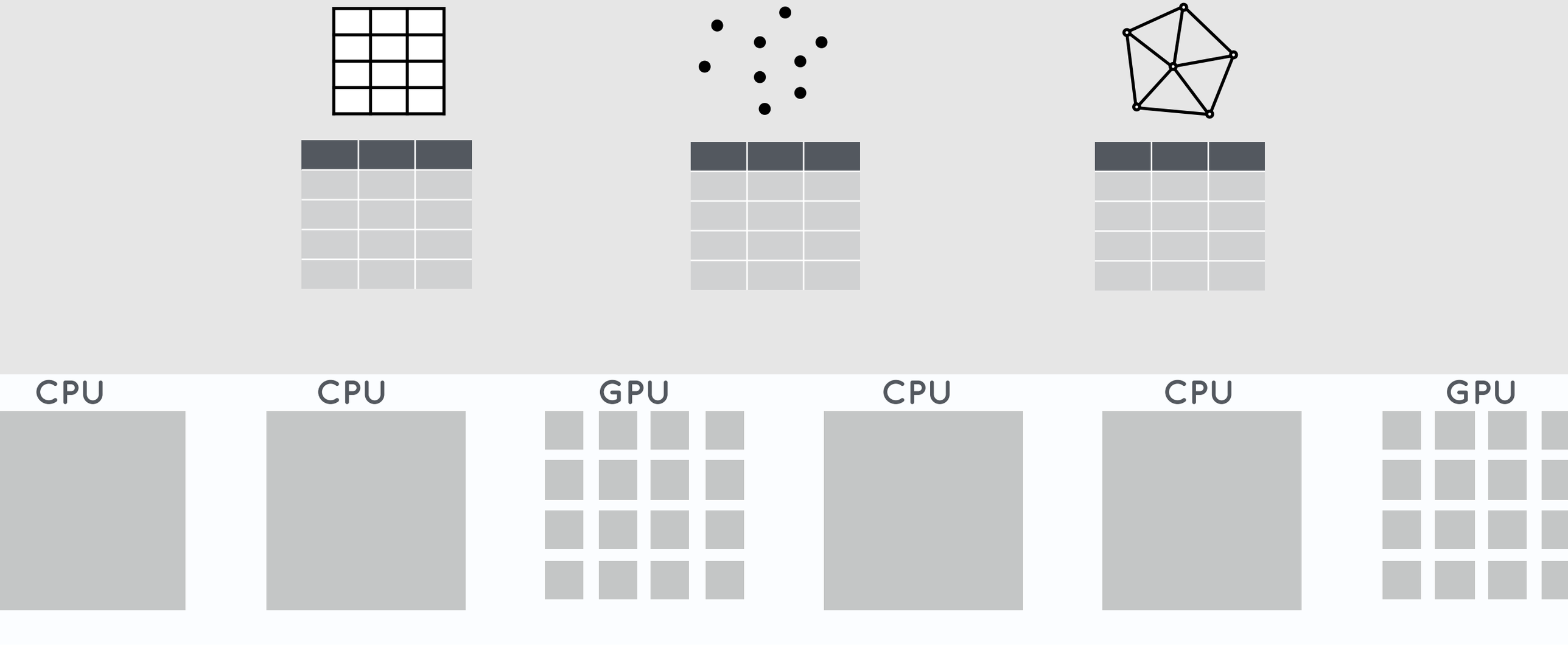
Liszt Supports Diverse Domains







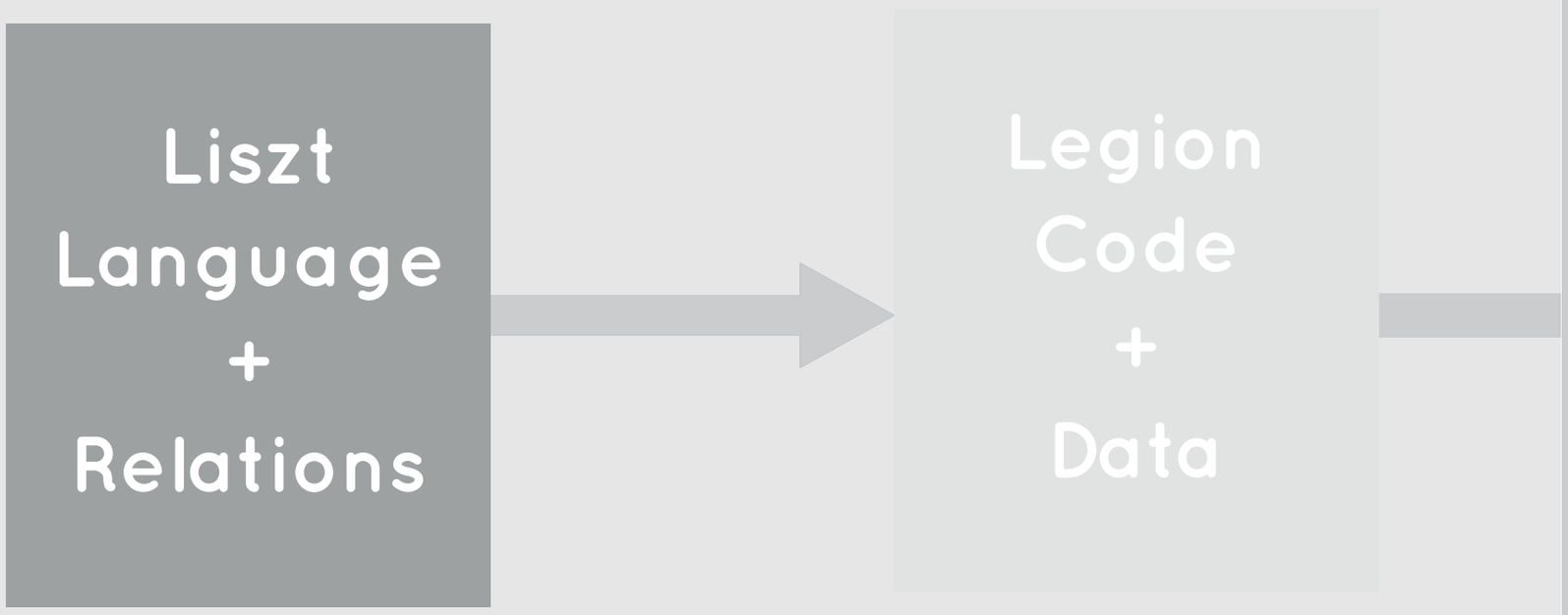
Unified Relational Data Model



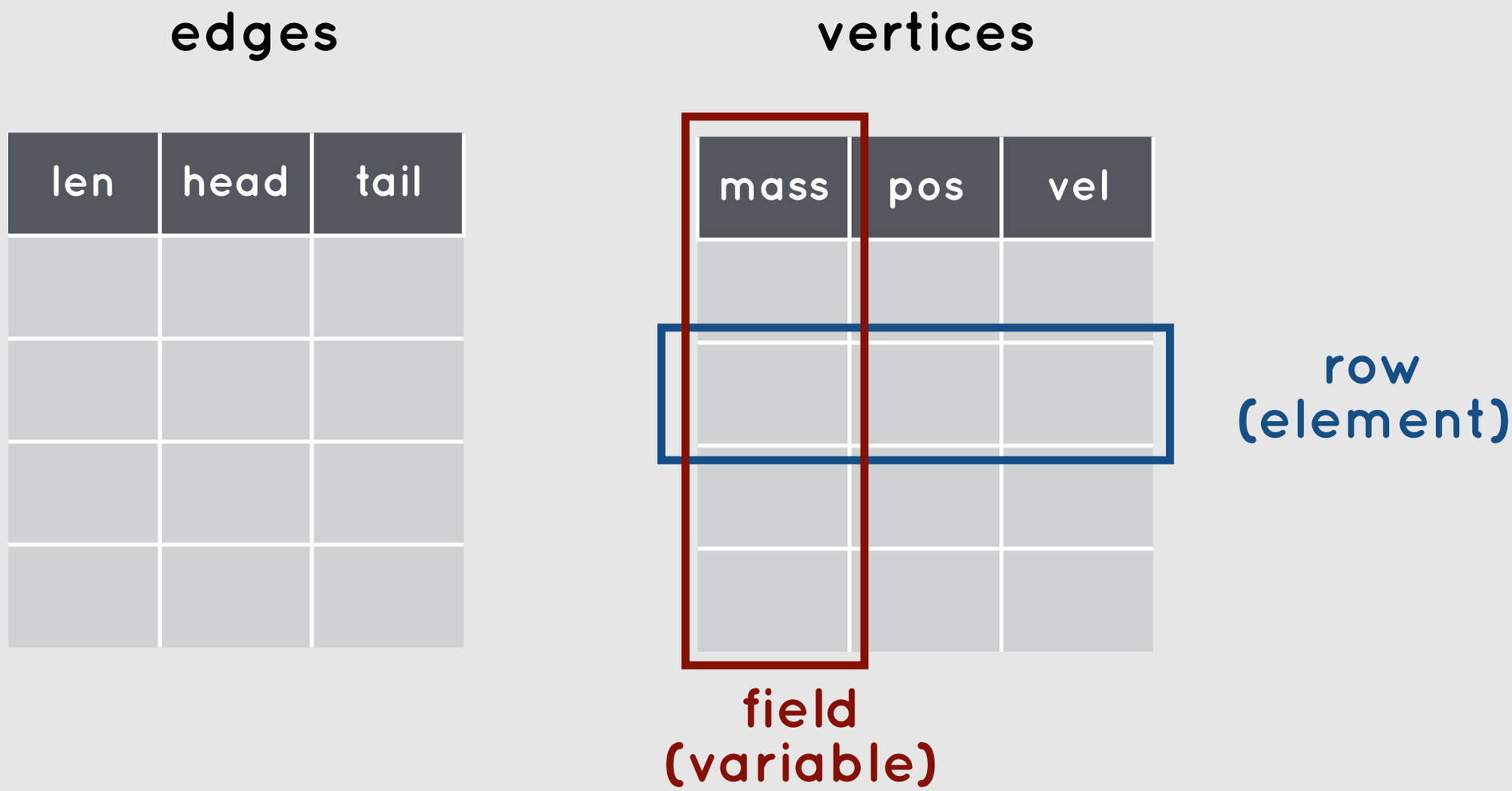
Outline

- Liszt background
- Task parallelism
- Data parallelism on one node
- Next steps for multiple nodes

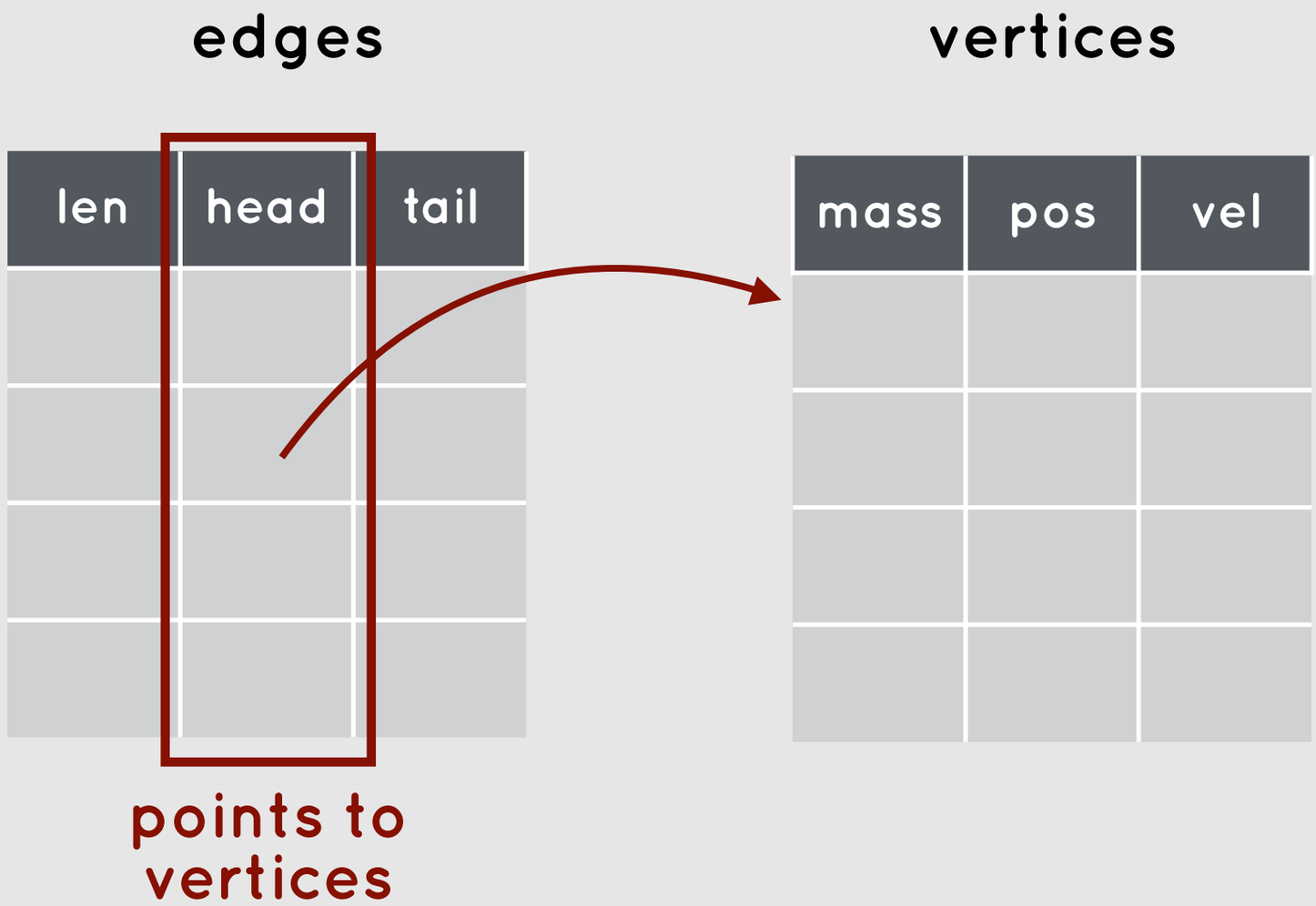
Background: Liszt Relations



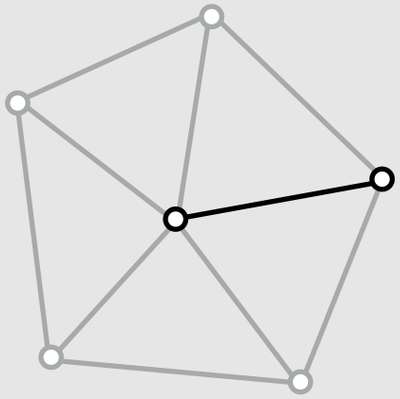
Relations


edges
len
head
tail
vertices
mass
pos
vel
row
(element)
field
(variable)

Relations

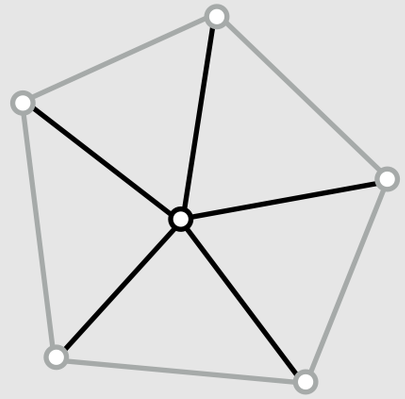

edges
vertices
len
head
tail
mass
pos
vel
points to
vertices

Relational Primitives



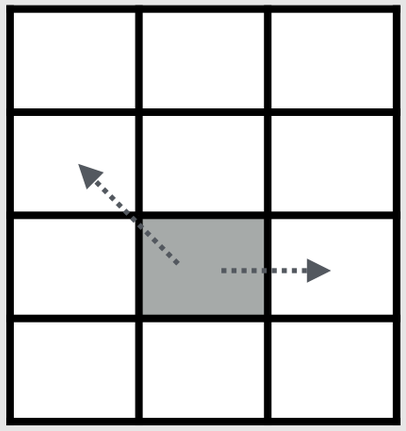
functional:

head, tail of an edge

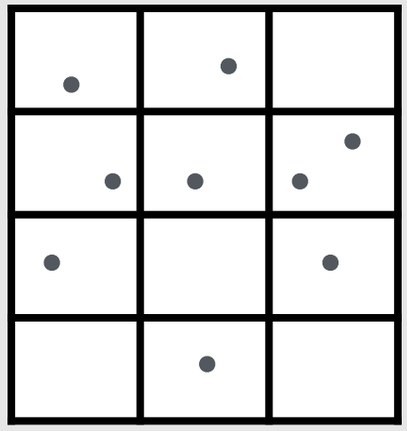


arbitrary:

edges of a vertex

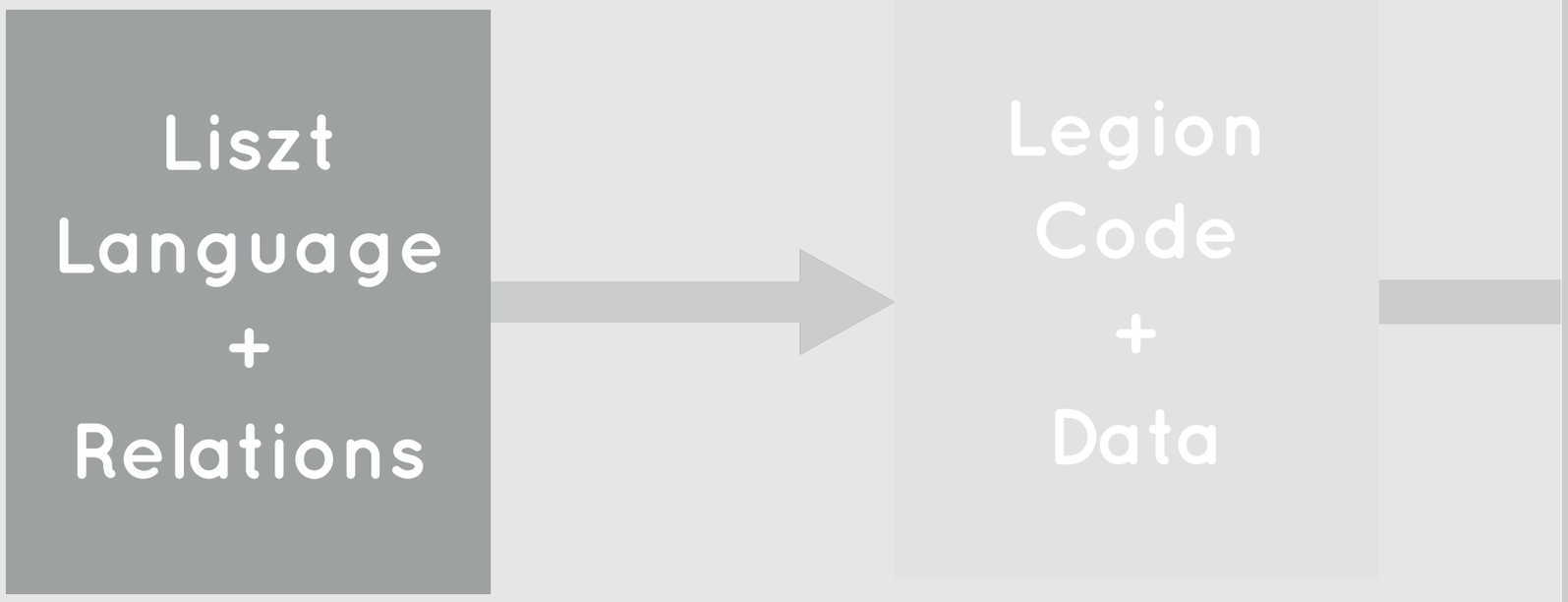


grid offsets



locate points in grid

Background: Example Code



Declare Simulation Data as Relations

```
local Tetmesh = L.require ‘domains.tetmesh’
local dragon  = Tetmesh.Load(‘dragon.veg’)

dragon.edges:NewField(‘rest_len’, L.float) :Load(0)
dragon.vertices:NewField(‘mass’, L.float)  :Load(1)
dragon.vertices:NewField(‘vel’, L.vec3f)   :Load({0,0,0})
dragon.vertices:NewField(‘acc’, L.vec3f)   :Load({0,0,0})
```

Computations over Elements

```
local Tetmesh = L.require 'domains.tetmesh'
local dragon  = Tetmesh.Load('dragon.veg')

dragon.edges:NewField('rest_len', L.float) :Load(0)
dragon.vertices:NewField('mass', L.float)  :Load(1)
dragon.vertices:NewField('vel', L.vec3f)   :Load({0,0,0})
dragon.vertices:NewField('acc', L.vec3f)   :Load({0,0,0})

local liszt InitLength(e : dragon.edges)
  var delta  = e.head.pos - e.tail.pos
  e.rest_len = sqrt(L.dot(delta, delta))
end
```

Phase Analysis

```
local Tetmesh = L.require 'domains.tetmesh'
local dragon  = Tetmesh.Load('dragon.veg')

dragon.edges:NewField('rest_len', L.float) :Load(0)
dragon.vertices:NewField('mass', L.float)  :Load(1)
dragon.vertices:NewField('vel', L.vec3f)   :Load({0,0,0})
dragon.vertices:NewField('acc', L.vec3f)   :Load({0,0,0})

local liszt InitLength(e : dragon.edges)
  var delta  = e.head.pos - e.tail.pos
  e.rest_len = sqrt(L.dot(delta, delta))
end
```

READ

WRITE

Computations over Elements

```
local Tetmesh = L.require 'domains.tetmesh'
local dragon  = Tetmesh.Load('dragon.veg')

dragon.edges:NewField('rest_len', L.float) :Load(0)
dragon.vertices:NewField('mass', L.float)  :Load(1)
dragon.vertices:NewField('vel', L.vec3f)   :Load({0,0,0})
dragon.vertices:NewField('acc', L.vec3f)   :Load({0,0,0})

local liszt InitLength(e : dragon.edges)
  var delta  = e.head.pos - e.tail.pos
  e.rest_len = sqrt(L.dot(delta, delta))
end

local liszt ComputeForces(e : dragon.edges)
  var force : L.vec3f = {0,0,0}
  var diff = e.head.pos - e.tail.pos
  var rest = e.rest_len * L.normalize(diff)
  e.head.force -= rest - diff
  e.tail.force += rest - diff
end

local liszt ApplyForces(v : dragon.vertices)
...
end
```

Simulation Loop

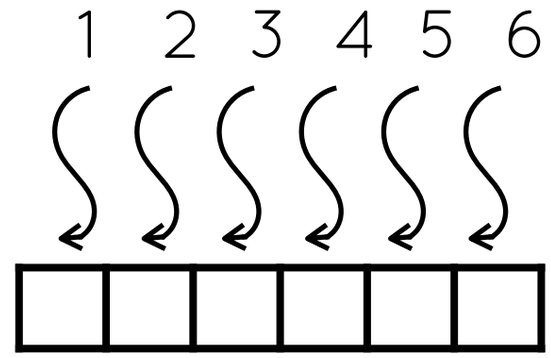
```
dragon.edges:foreach(InitLength)

for i = 1,300 do

  dragon.edges:foreach(ComputeForces)

  dragon.vertices:foreach(ApplyForces)

end
```

simulation loop

Running in Parallel is Simple

```
dragon.edges:foreach(InitLength)
for i = 1,300 do
  dragon.edges:foreach(ComputeForces)
  dragon.vertices:foreach(ApplyForces)
end
```

parallel “for loops”



Phase Restrictions

```
local Tetmesh = L.require 'domains.tetmesh'
local dragon  = Tetmesh.Load('dragon.veg')

dragon.edges:NewField('rest_len', L.float) :Load(0)
dragon.vertices:NewField('mass', L.float)  :Load(1)
dragon.vertices:NewField('vel', L.vec3f)   :Load({0,0,0})
dragon.vertices:NewField('acc', L.vec3f)   :Load({0,0,0})

local liszt InitLength(e : dragon.edges)
  var delta  = e.head.pos - e.tail.pos
  e.rest_len = sqrt(L.dot(delta, delta))
end

local liszt ComputeForces(e : dragon.edges)
  var force : L.vec3f = {0,0,0}
  var diff = e.head.pos - e.tail.pos
  var rest = e.rest_len * L.normalize(diff)
  e.head.force -= rest - diff
  e.tail.force += rest - diff
end

local liszt ApplyForces(v : dragon.vertices)
...
end
```

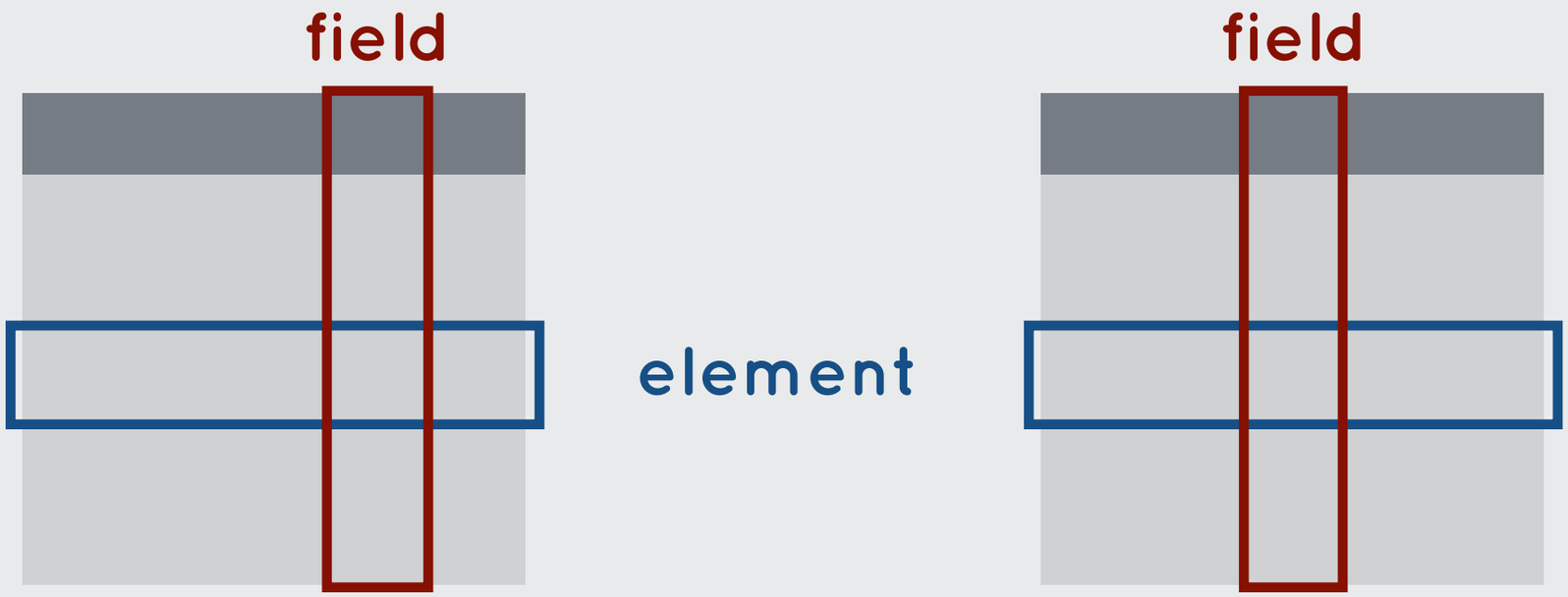
WRITE —
directly access
argument field

Progress: Liszt to Legion & Inter-Task Parallelism

Liszt Language + Relations

Legion Code + Data

Relations as Logical Regions



Relations

Legion Regions

Liszt Functions as Legion Tasks

CT

```
...

liszt InitLength(e : dragon.edges)
  var delta  =   e.head.pos
                - e.tail.pos
  e.rest_len = L.len(delta)
end

dragon.edges:foreach(InitLength)

...
```

IL

...

Control Task

Compute Tasks

Data Dependencies

```
liszt ComputeForces(e : dragon.edges)
  var force : L.vec3f = {0,0,0}
  var diff = e.head.pos - e.tail.pos
  var rest = e.rest_len *
             L.normalize(diff)
  e.head.force -= rest - diff
  e.tail.force += rest - diff
end
```

READ

REDUCE

```
vertices.pos    : READ,
                  EXCLUSIVE
                  VERTICES_PARTN

vertices.force  : REDUCE,
                  EXCLUSIVE,
                  ATOMIC_ADD_OP,
                  VERTICES_PARTN
```

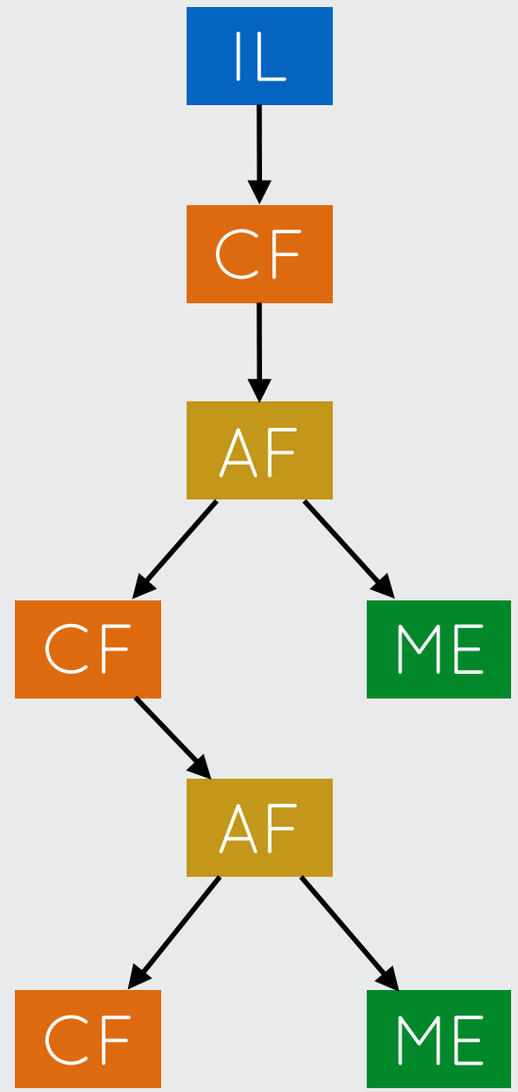
Phase Analysis

Legion Requirements

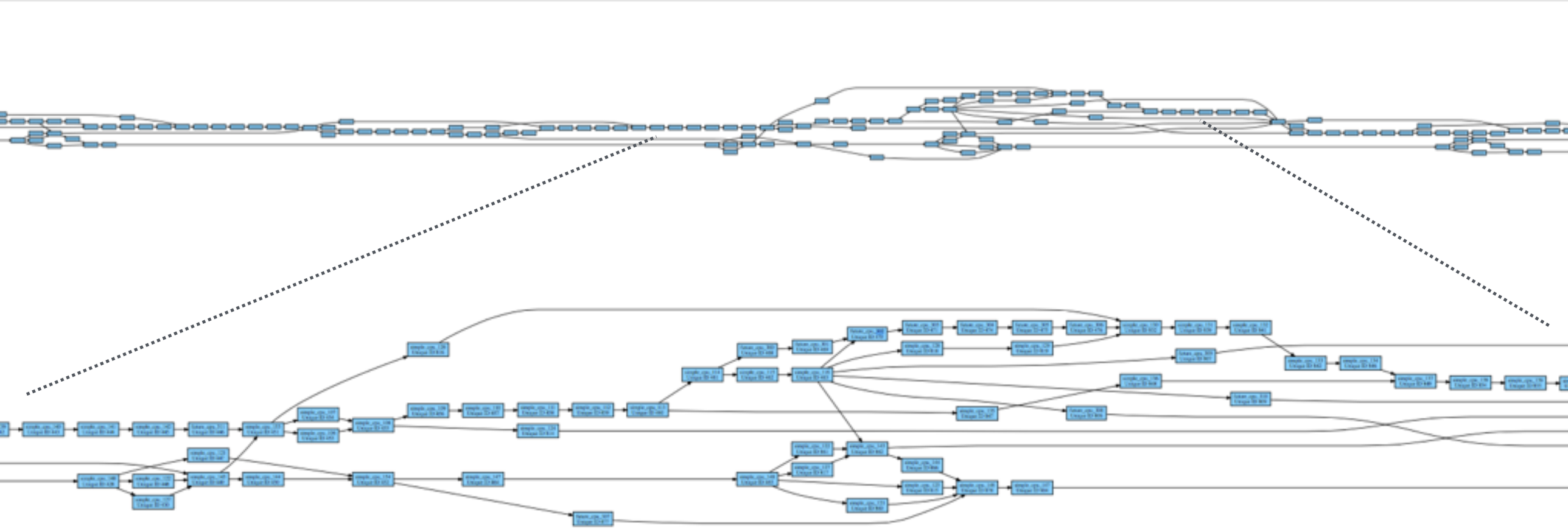
Legion Gives Inter-Task Parallelism

```
dragon.edges:map(InitLength)

for i = 1,300 do

  dragon.vertices:foreach(ComputeForces)

  dragon.vertices:foreach(ApplyForces)

  dragon.vertices:foreach(MeasureEnergy)

end
```

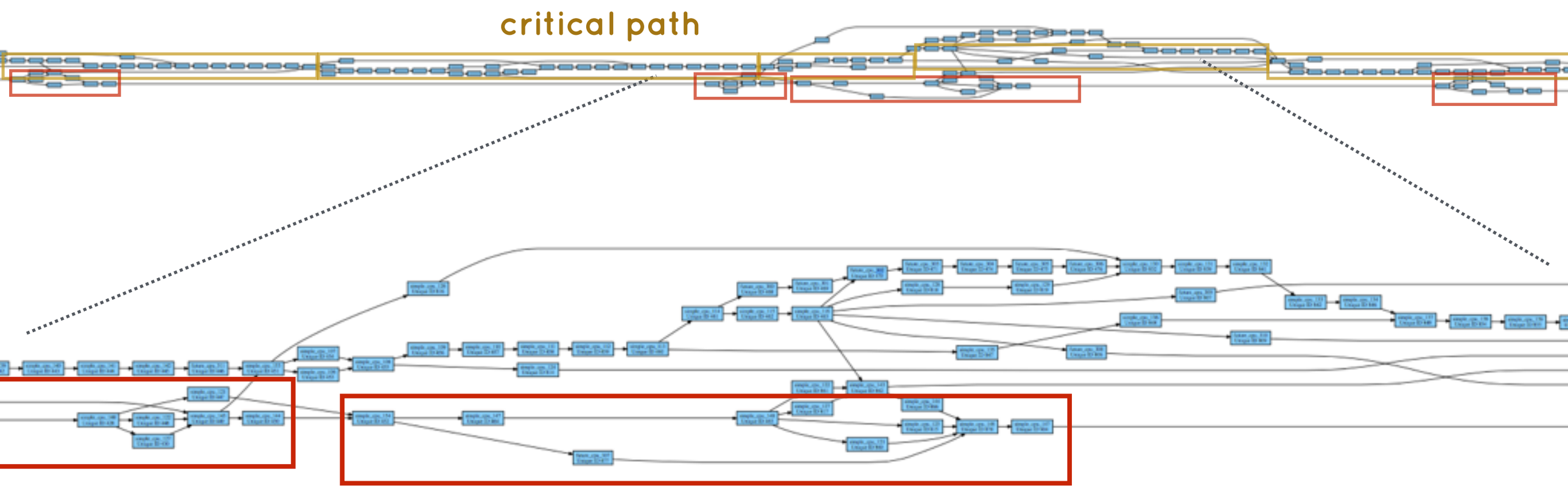
Sequential Functions



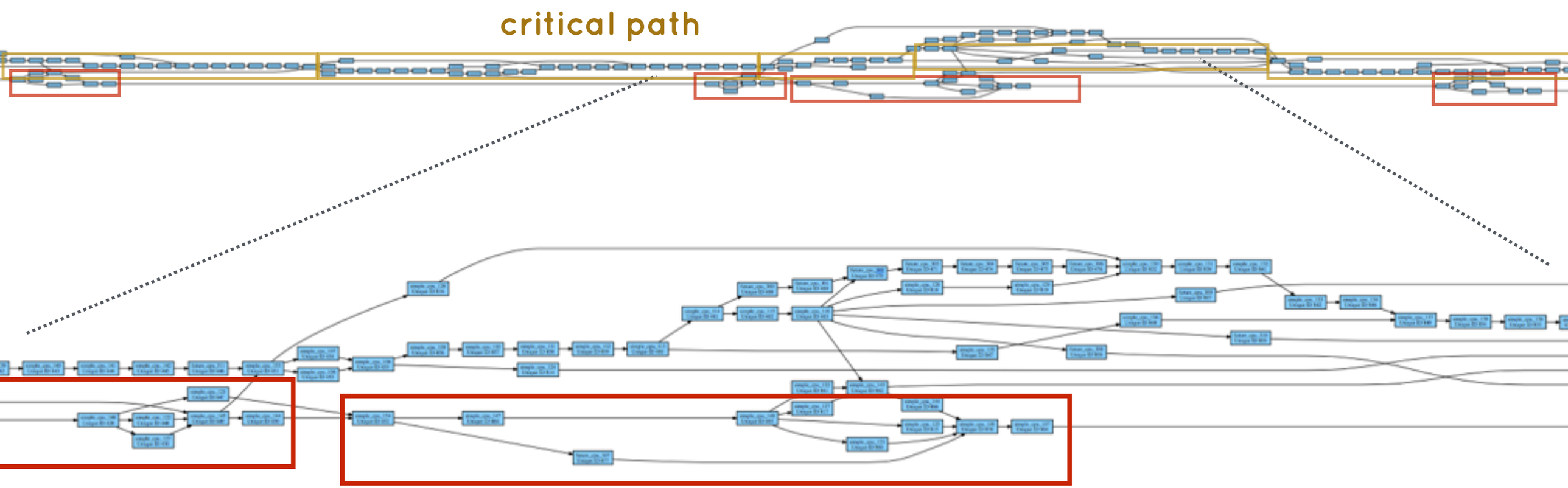
Soleil Task Graph



Soleil Task Graph

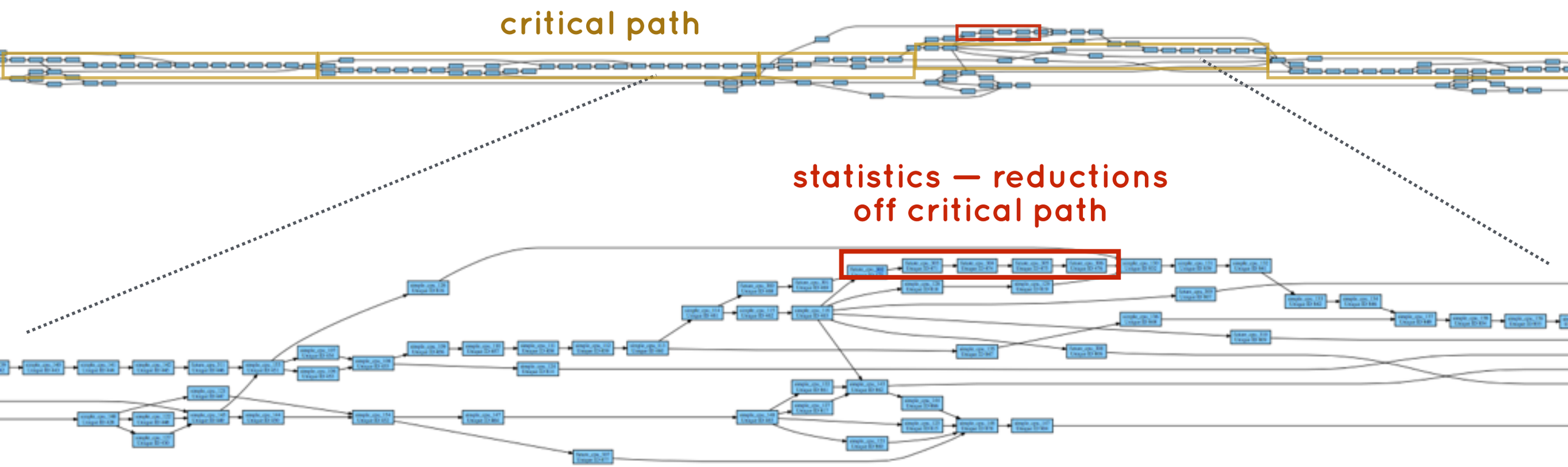


Soleil Task Graph



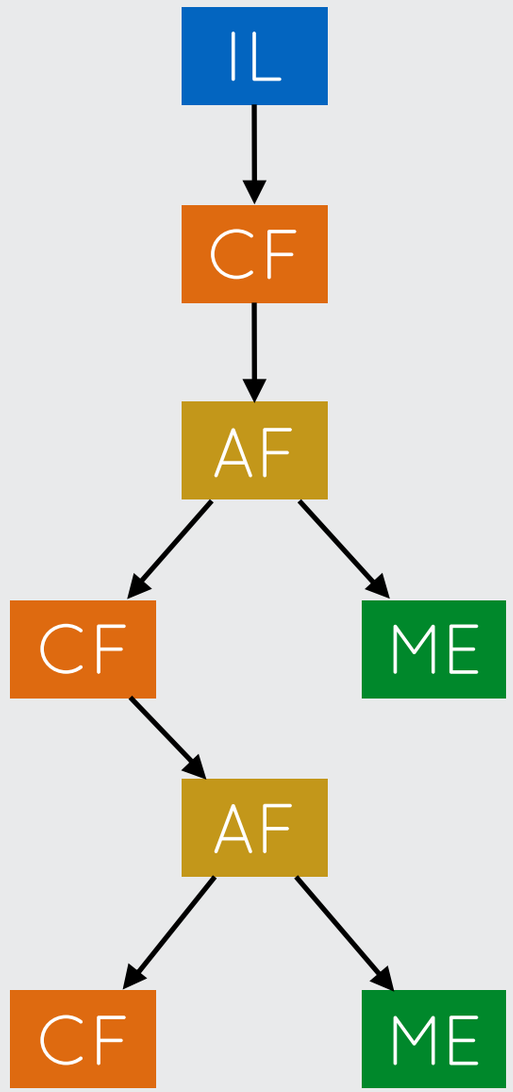
more phenomena (multi-physics)
⇒ more inter-task parallelism

Soleil Task Graph



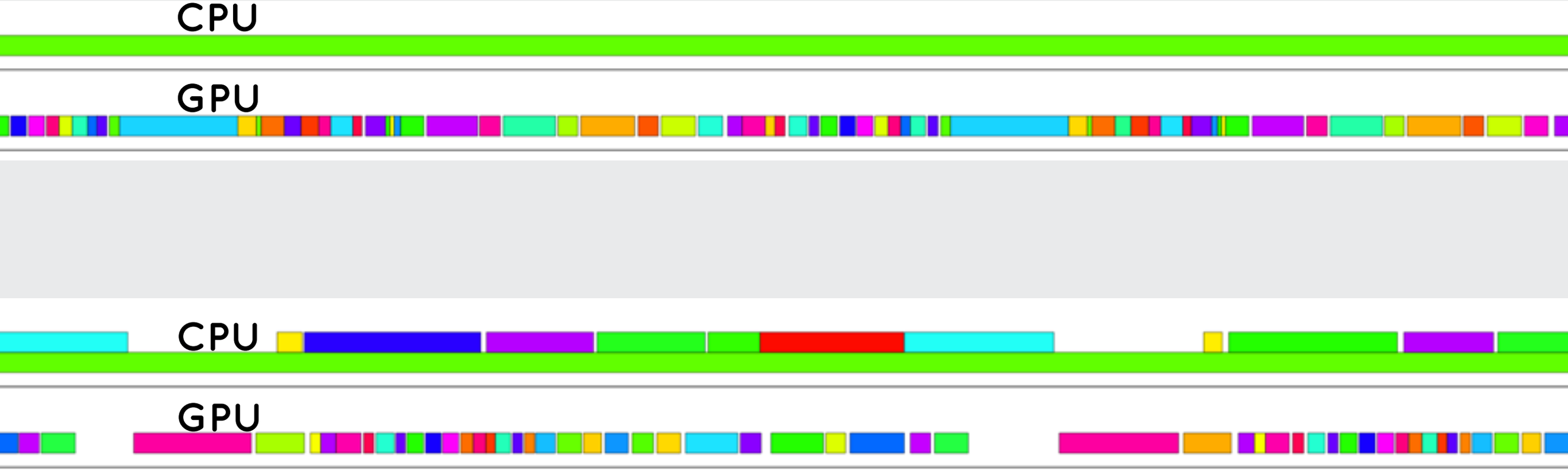
Using CPUs and GPUs: Pin Tasks

```
dragon.edges:map(InitLength)

for i = 1,300 do

  dragon.vertices:foreach(ComputeForces)

  dragon.vertices:foreach(ApplyForces)

  dragon.vertices:foreach(MeasureEnergy,
                          location = {L.CPU})
end
```



Sequential Functions

Soleil Runs On CPU + GPU



Progress: Data Parallelism

Liszt Language + Relations

Legion Code + Data

Data Parallelism with Parallel Tasks

CT

```
...

liszt InitLength(e : dragon.edges)
  var delta  =   e.head.pos
                - e.tail.pos
  e.rest_len = L.len(delta)
end

dragon.edges:foreach(InitLength)

...
```

IL ••• IL

... ••• ...

Control Task

Parallel Tasks

Non-conflicting Parallel Tasks

```
local liszt InitLength(e : dragon.edges)
  var delta  = e.head.pos - e.tail.pos
  e.rest_len = sqrt(L.dot(delta, delta))
end
```

READ does not conflict

```
liszt ComputeForces(e : dragon.edges)
  var force : L.vec3f = {0,0,0}
  var diff = e.head.pos - e.tail.pos
  var rest = e.rest_len *
             L.normalize(diff)
  e.head.force -= rest - diff
  e.tail.force += rest - diff
end
```

WRITE disjoint — directly access argument field

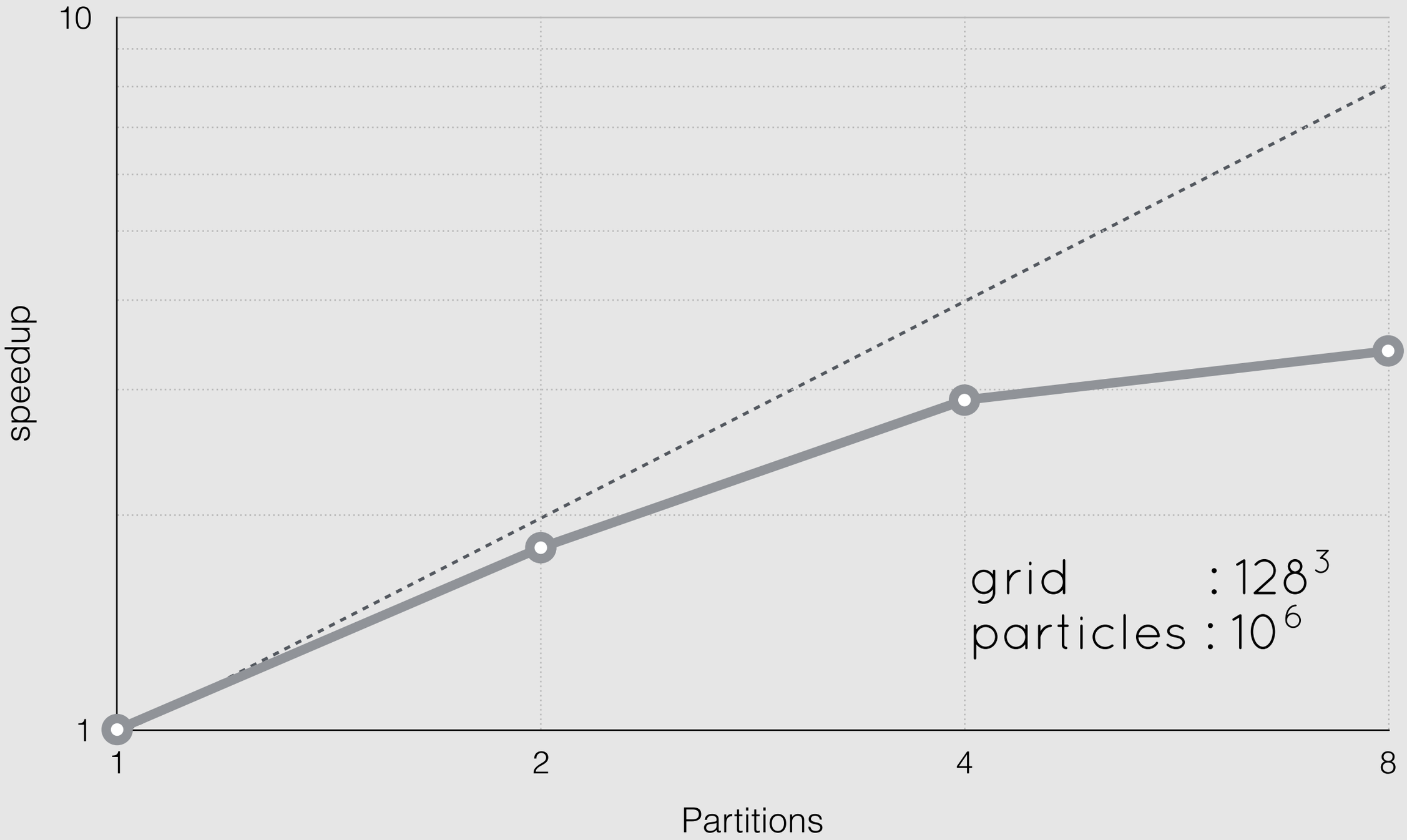
REDUCE → Safe atomics

Verified Correctness

- Correct results
  - inter-task parallelism, CPU + GPU
  - with data parallelism, on multiple cores

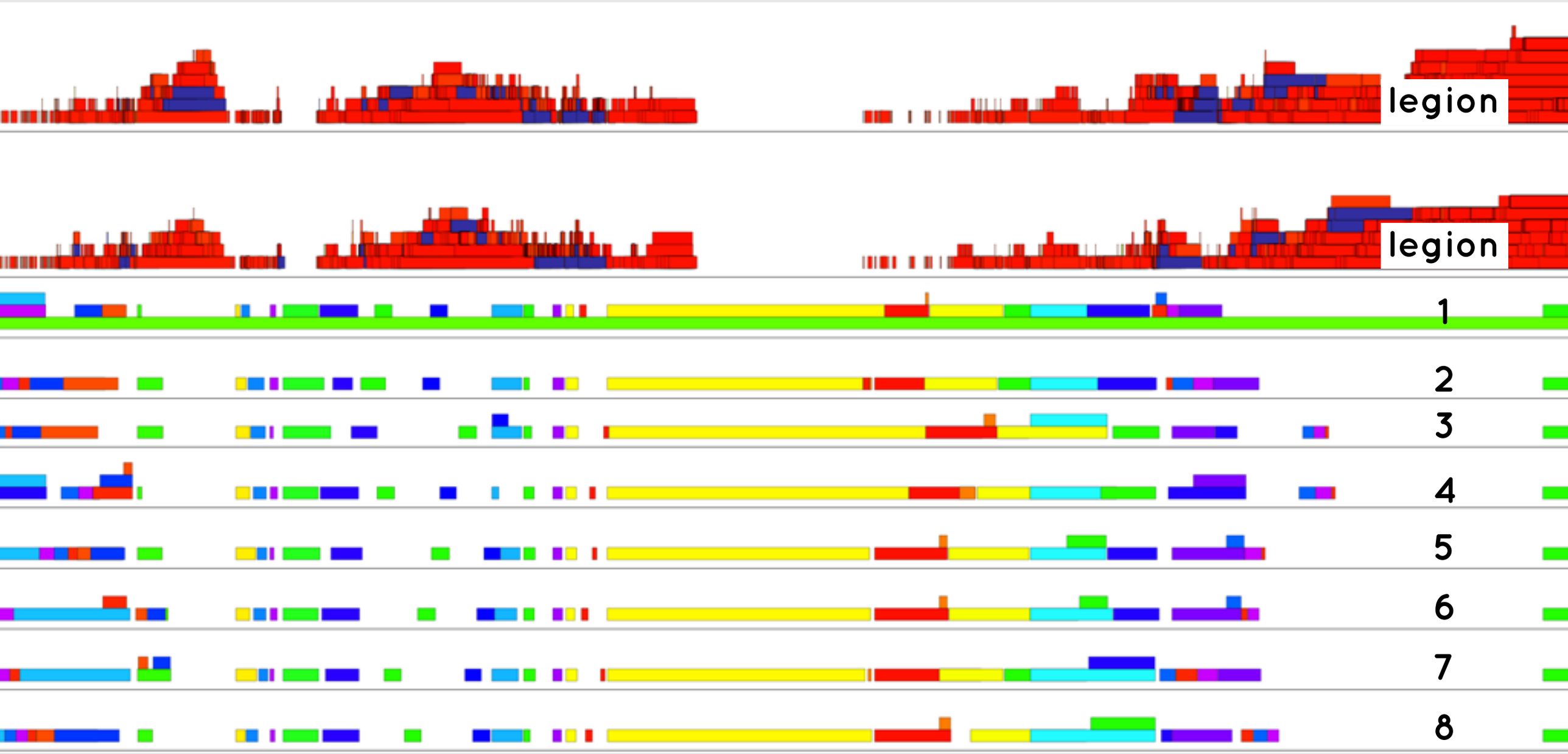
- Application and test benchmarks, and Soleil give correct results

Soleil Performance



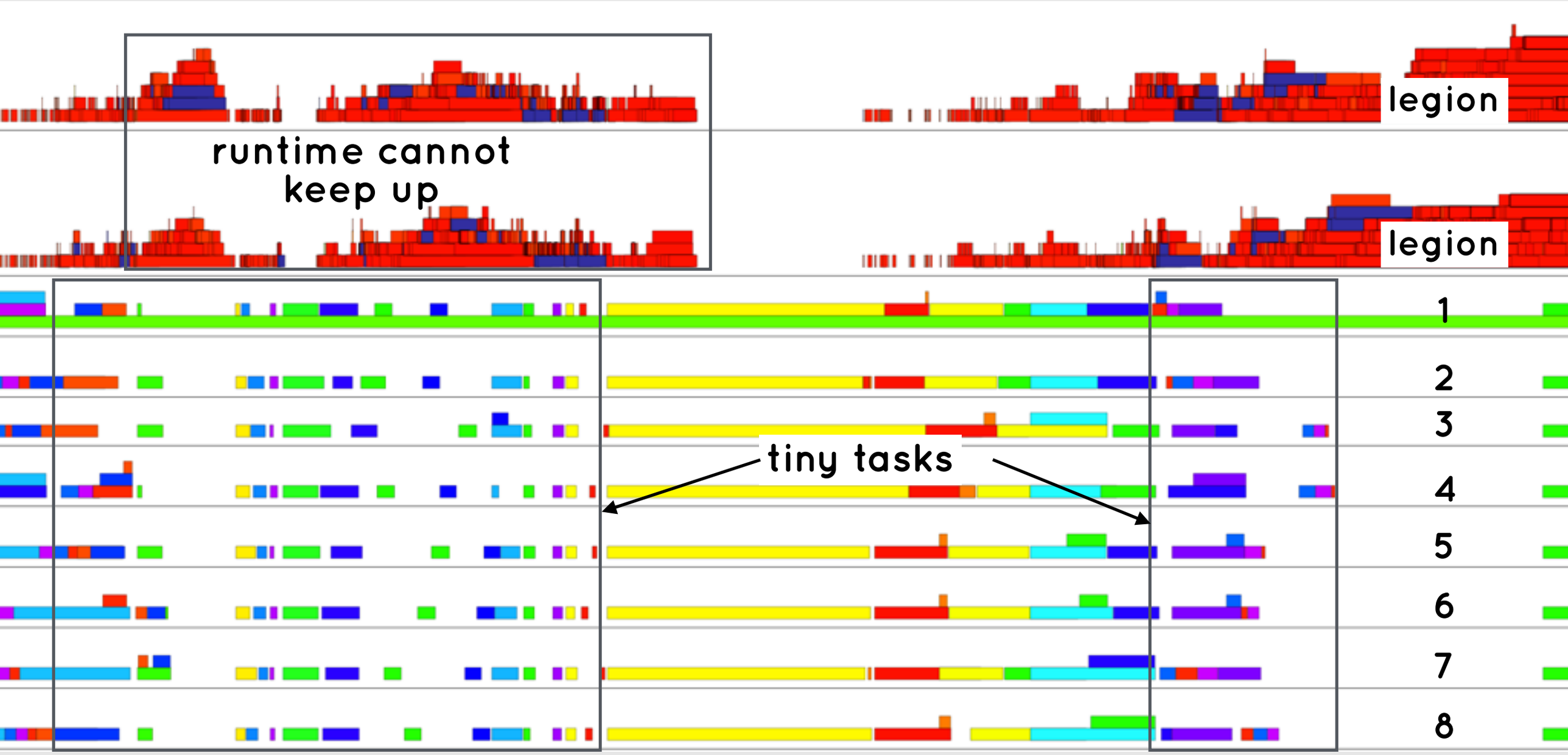
On Sapling, 2 sockets/ node, 6 cpus/socket

In Progress: Performance with Legion



8 partitions on Sapling, 2 sockets/ node, 6 cpus/socket

In Progress: Performance with Legion



8 partitions on Sapling, 2 sockets/ node, 6 cpus/socket

Status

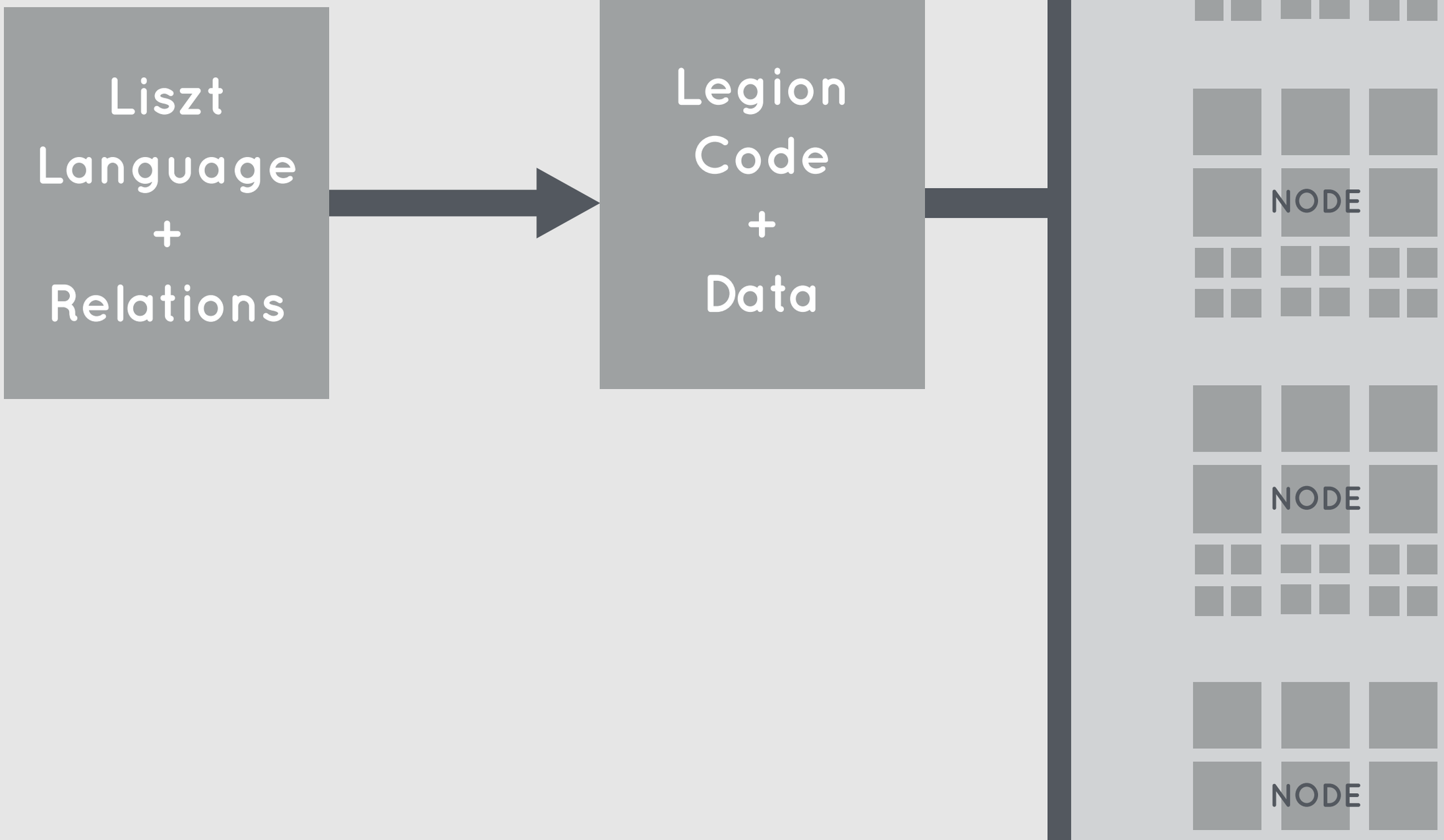
Done

- Correctly compiling Liszt to Legion
- Single partition performance on CPU and GPU
- Correctness with inter-task parallelism, CPU + GPU
- Correctness with data parallelism

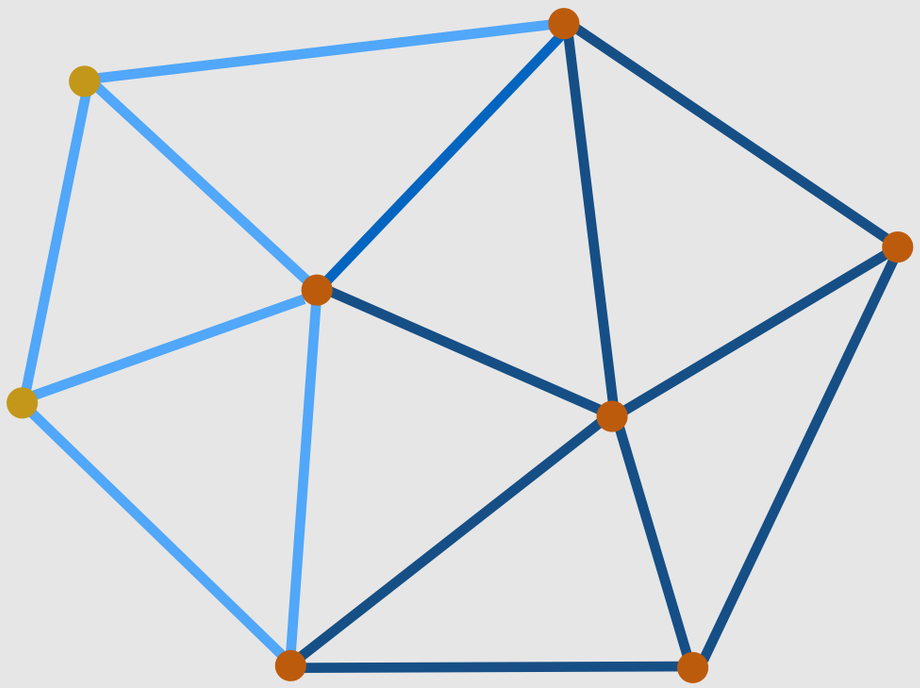
In Progress

- Performance on 8 and more cores on a node

Next:
Multiple Nodes

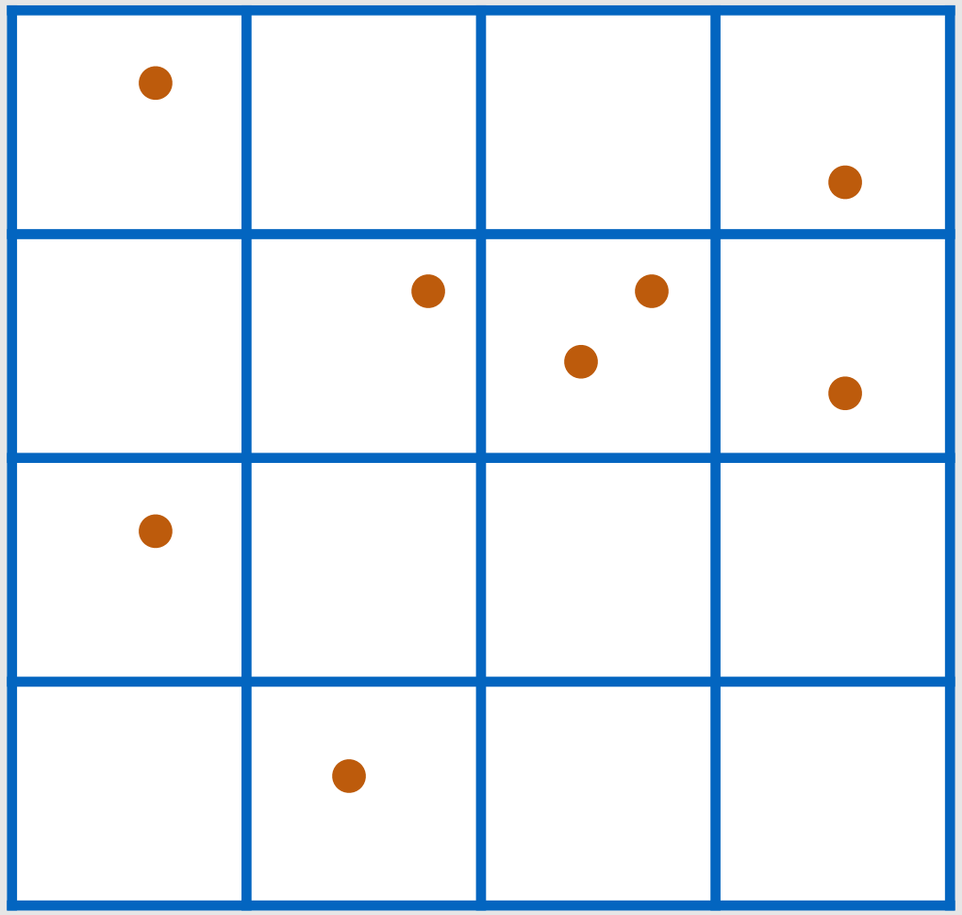


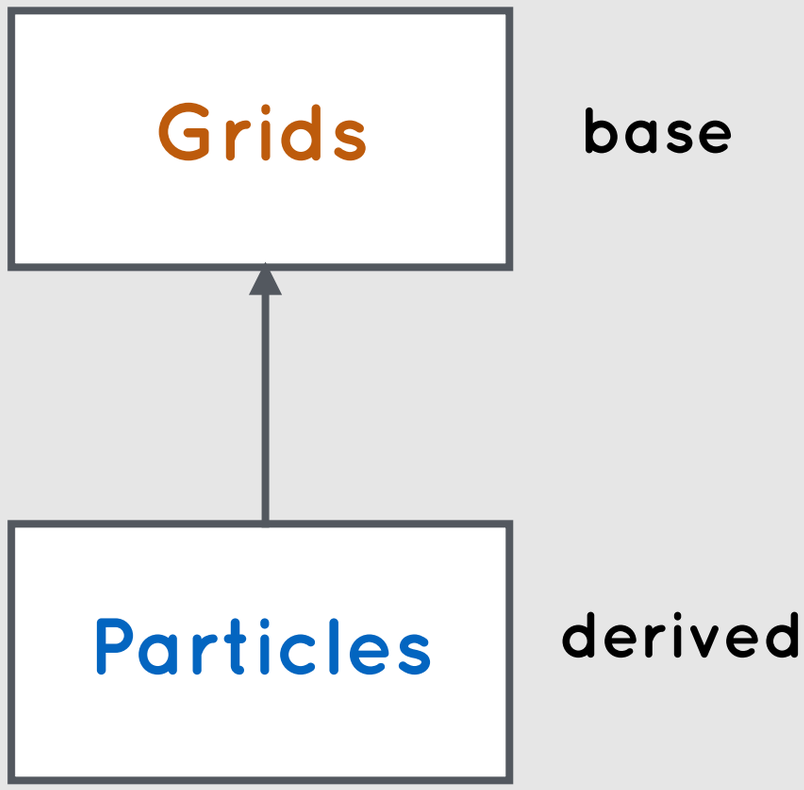
Stencil Analysis



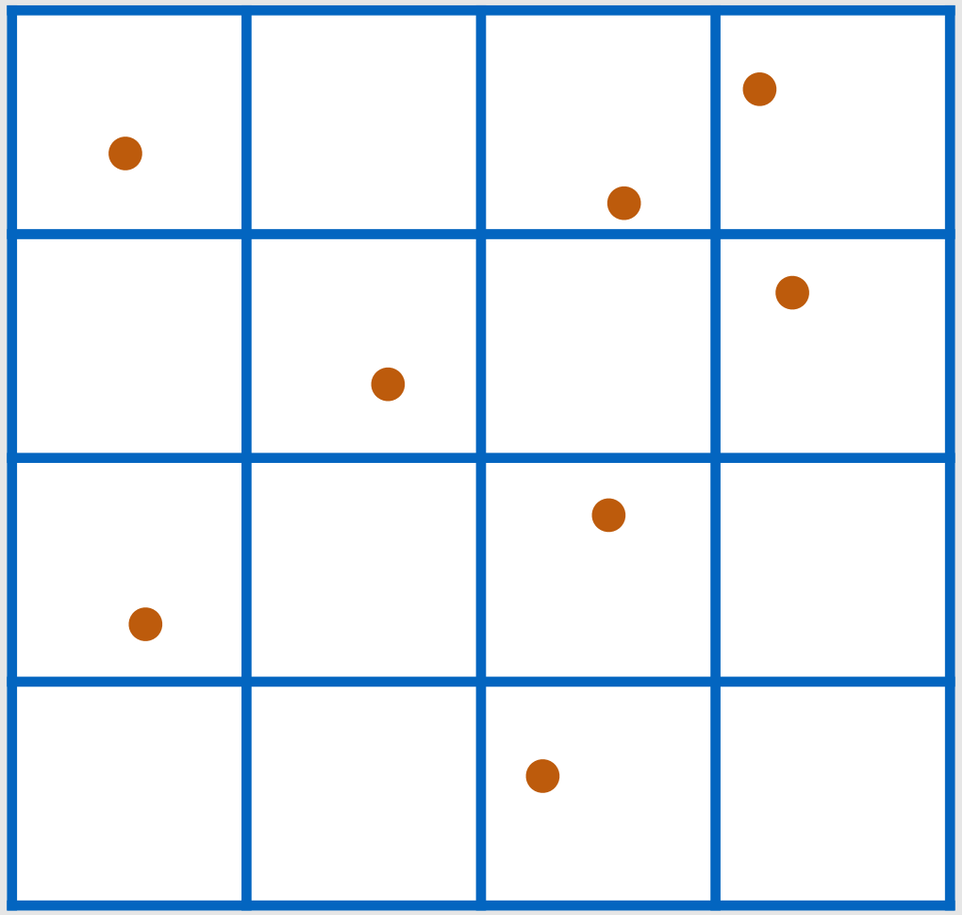
```
liszt ComputeForces(e : dragon.edges)
  var force : L.vec3f = {0,0,0}
  var diff = e.head.pos - e.tail.pos
  var rest = e.rest_len *
             L.normalize(diff)
  e.head.force -= rest - diff
  e.tail.force += rest - diff
end
```

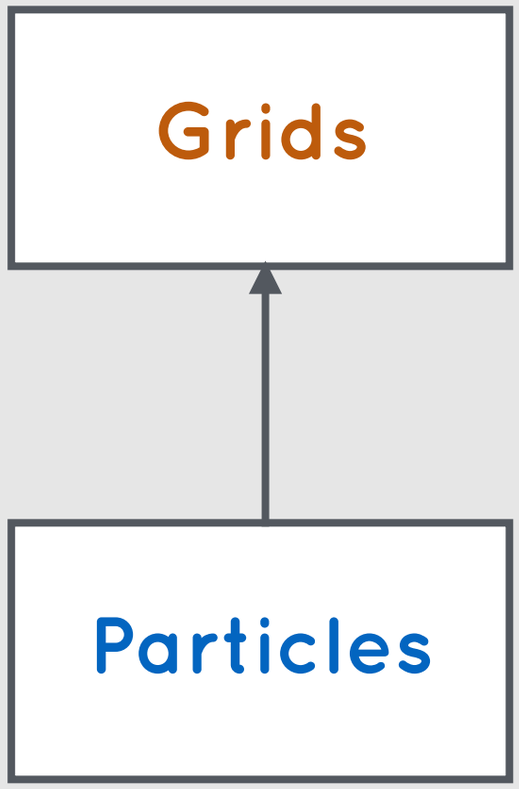
Jointly Partition Multiple Relations





Partitioning Dynamic Relations





Liszt Language + Relations

Legion Code + Data

NODE

NODE

NODE

NODE

NODE

End