Parallelism in SpECTRE: A (slightly) technical overview

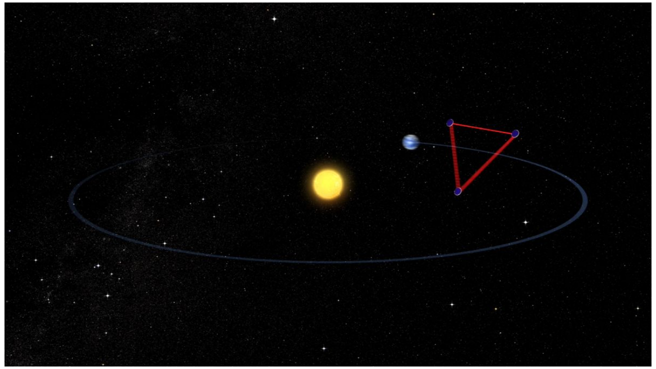
Kyle Nelli

DROP- Outline

- Async vs sync
  - What isn't guaranteed anymore
- Distributed Objects/Parallel Components
  - First C++ object
  - Different types
  - Placement on cores
  - How is it distributed
  - Proxy
- Actions
  - Different types
  - Calling them
- Phases
- Global Cache

Motivations for parallelism

- Future science
  - Higher accuracy
  - More gridpoints/computation
  - More resources
  - New methods
- Challenges
  - Time
  - Workload/Domain decomp
  - Global synchronizations
- Solutions
  - **Asynchronous**
  - GPUs


LISA Consortium

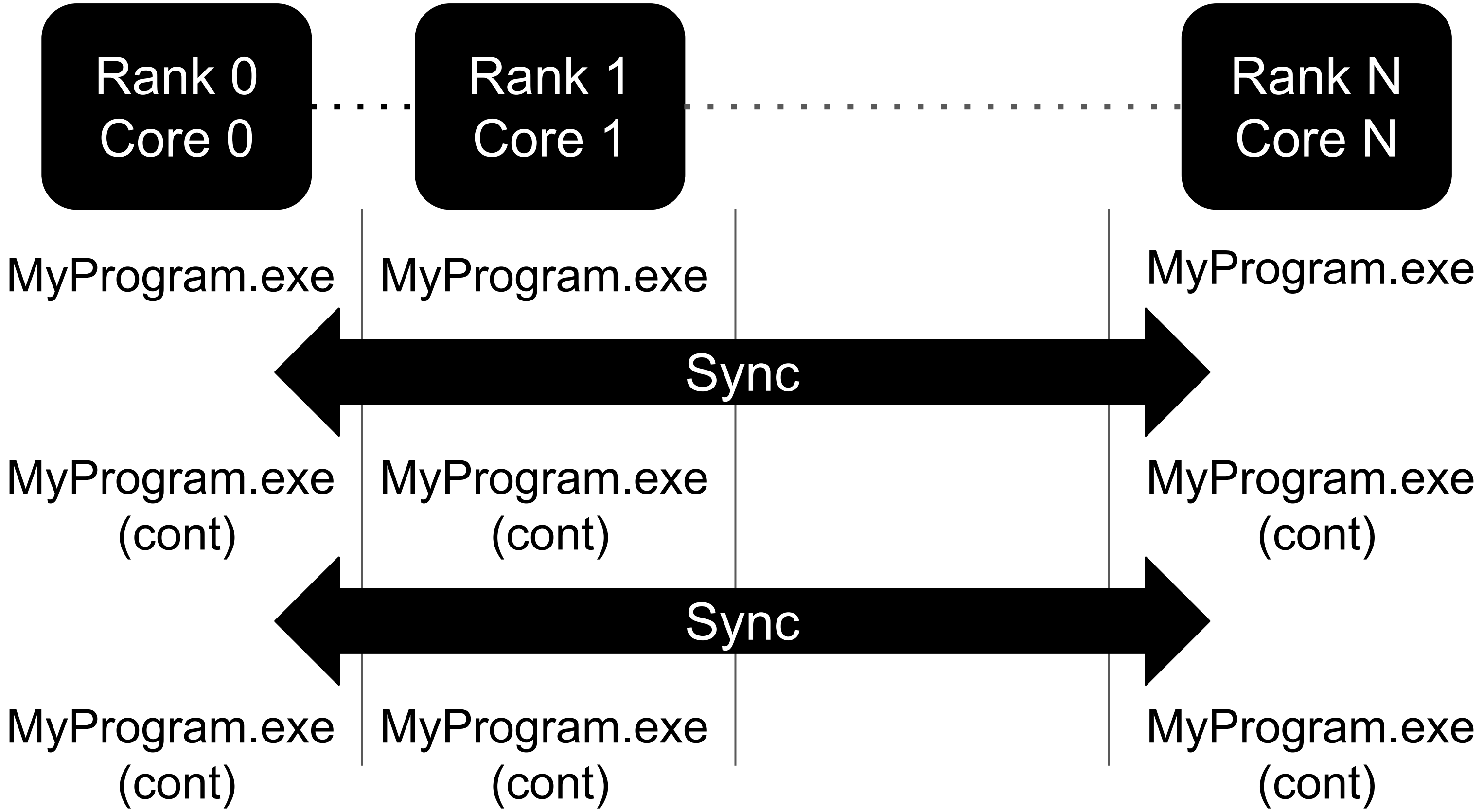
NERSC

ORNL

Big picture: Synchronous vs Asynchronous

SpECTRE jargon

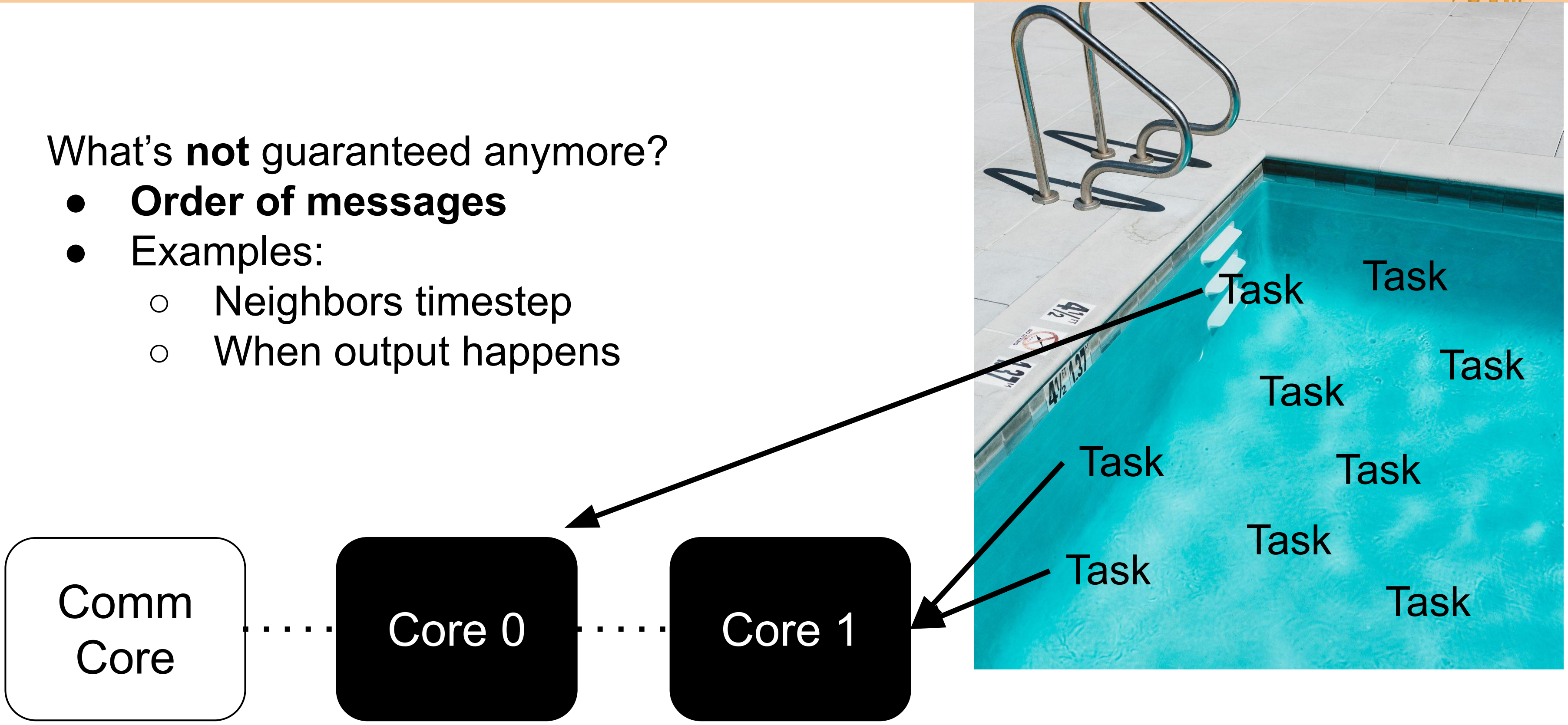
Synchronous



Asynchronous

What's **not** guaranteed anymore?

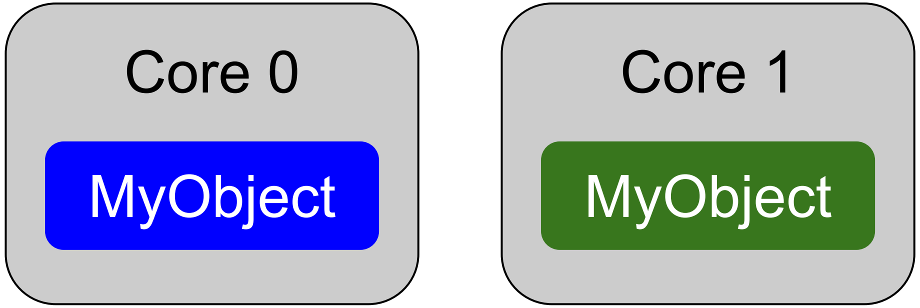
- **Order of messages**
- Examples:
  - Neighbors timestep
  - When output happens



How to parallelize the data: Distributed Objects

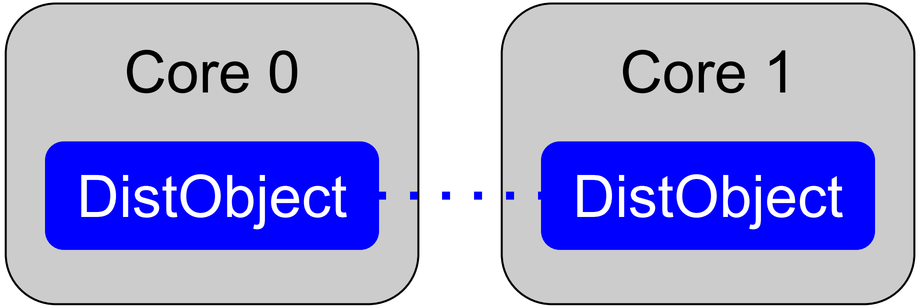
Regular ol’ C++ object

- Member functions
- Member variables (data)
- Lives where created
- Not “connected” to other objects of the same type



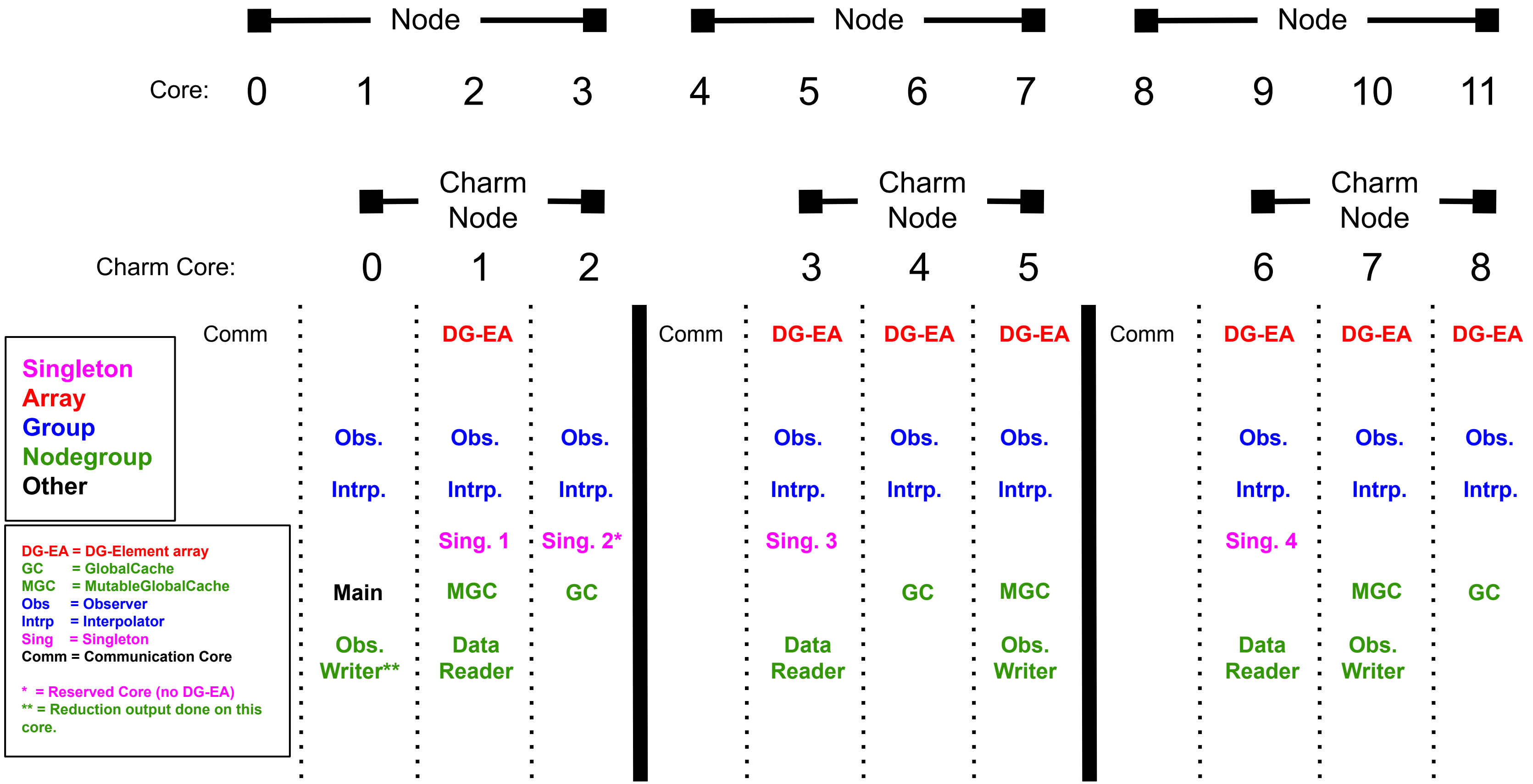
Distributed Object

- Member functions
- Member variables (data)
- “Abstract” notion of object
  - Distributed across resources
- 4 types in SpECTRE
  - Singleton
  - *Array
  - Group
  - Nodegroup



SpECTRE jargon:
Distributed Object = Parallel Component

Parallel Component placement on cores



	Node				Node				Node			
Core:	0	1	2	3	4	5	6	7	8	9	10	11
		Charm Node				Charm Node				Charm Node		
Charm Core:		0	1	2		3	4	5		6	7	8
	Comm		DG-EA		Comm	DG-EA	DG-EA	DG-EA	Comm	DG-EA	DG-EA	DG-EA
		Obs.	Obs.	Obs.		Obs.	Obs.	Obs.		Obs.	Obs.	Obs.
		Intrp.	Intrp.	Intrp.		Intrp.	Intrp.	Intrp.		Intrp.	Intrp.	Intrp.
			Sing. 1	Sing. 2*		Sing. 3				Sing. 4		
		Main	MGC	GC			GC	MGC			MGC	GC
		Obs. Writer**	Data Reader			Data Reader		Obs. Writer		Data Reader	Obs. Writer	

Singleton
Array
Group
Nodegroup
Other

DG-EA = DG-Element array
GC = GlobalCache
MGC = MutableGlobalCache
Obs = Observer
Intrp = Interpolator
Sing = Singleton
Comm = Communication Core

* = Reserved Core (no DG-EA)
** = Reduction output done on this core.

How to run code on Parallel Components

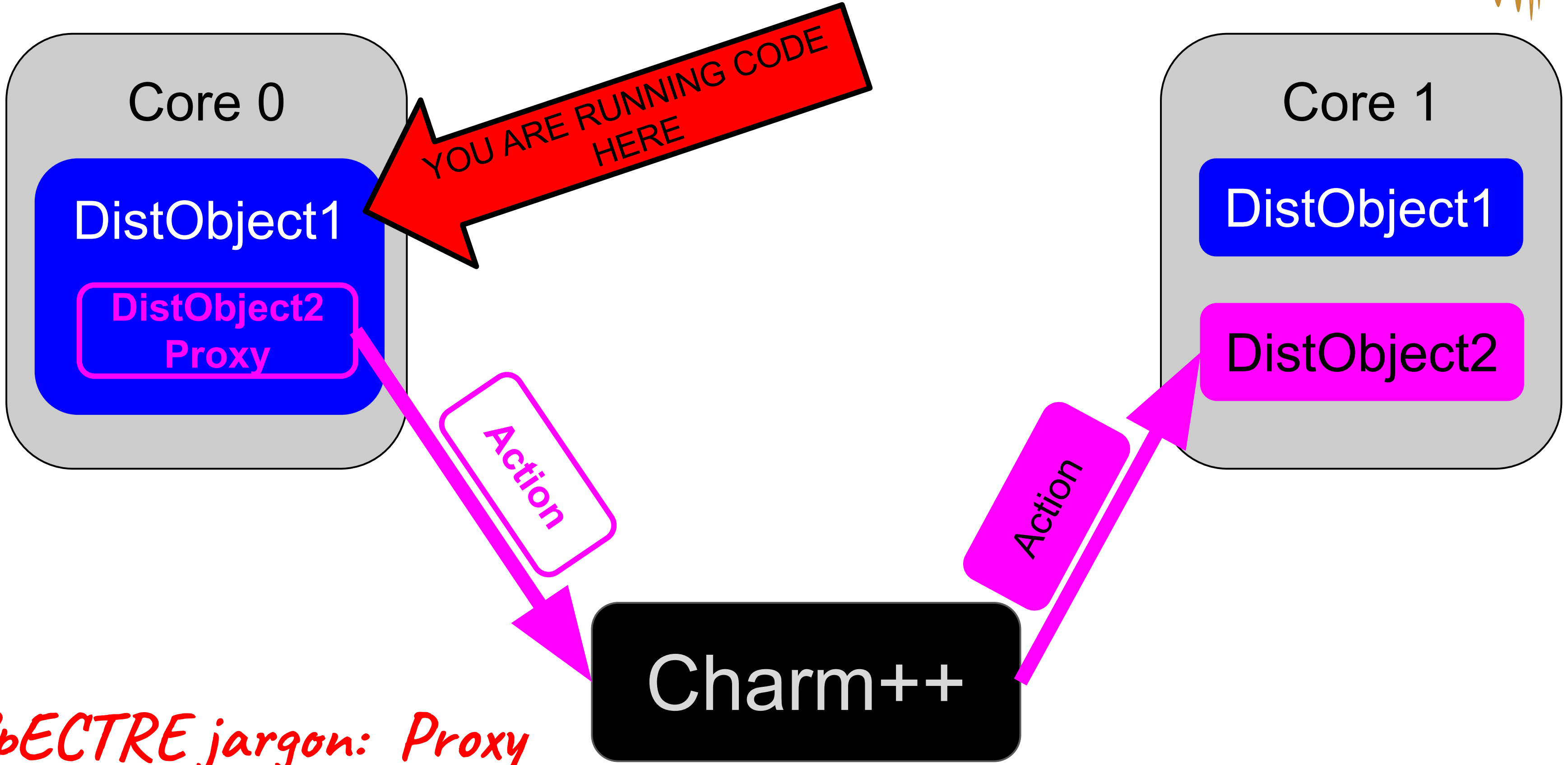
Member functions!

SpECTRE jargon:

Member function of Parallel Component = Action = Task!

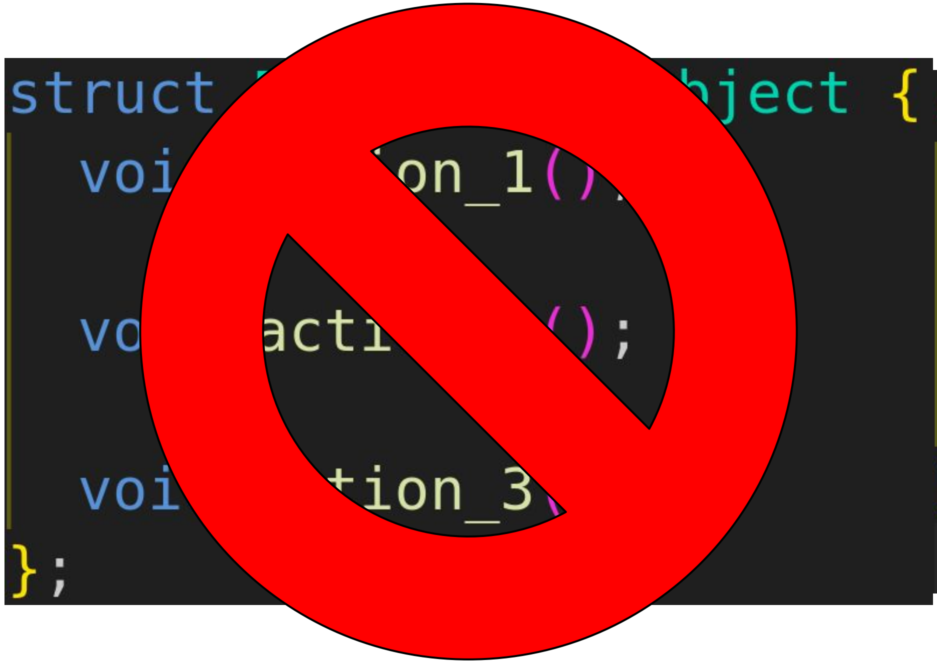
- Types of Actions
  - *Simple Action*
  - *Iterable Action*
  - *Reduction Action*
  - **Threaded Action*
  - **Local Synchronous Action*
- Has access to data of parallel component
- How do you run an action?

Parallel Component proxies



SpECTRE jargon: Proxy

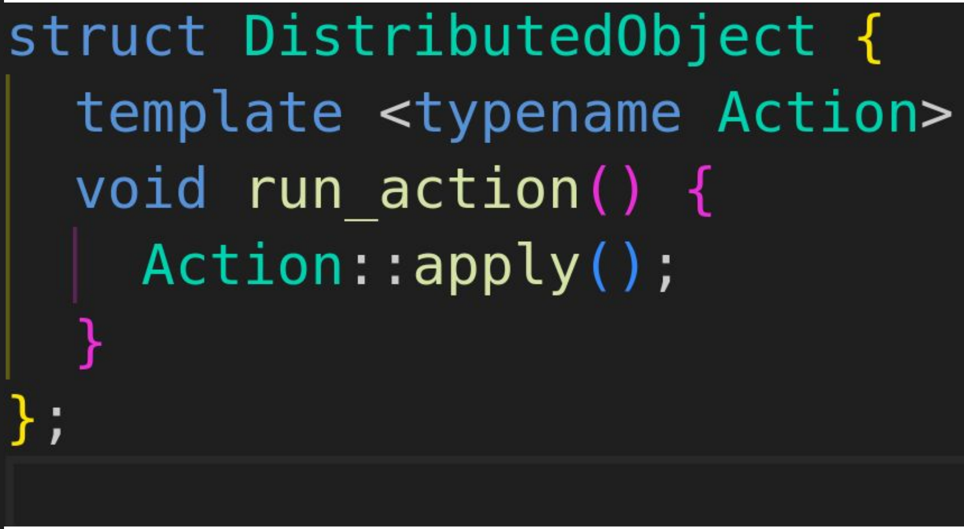
Side note about implementation



```
struct       bject {
  voi    on_1()

  vo   acti     ();

  voi    tion_3(
};
```



```
struct DistributedObject {
  template <typename Action>
  void run_action() {
    Action::apply();
  }
};
```

When to run Actions on Parallel Components

More on Iterable Actions

SpECTRE jargon:
Collection of iterable actions = The Algorithm

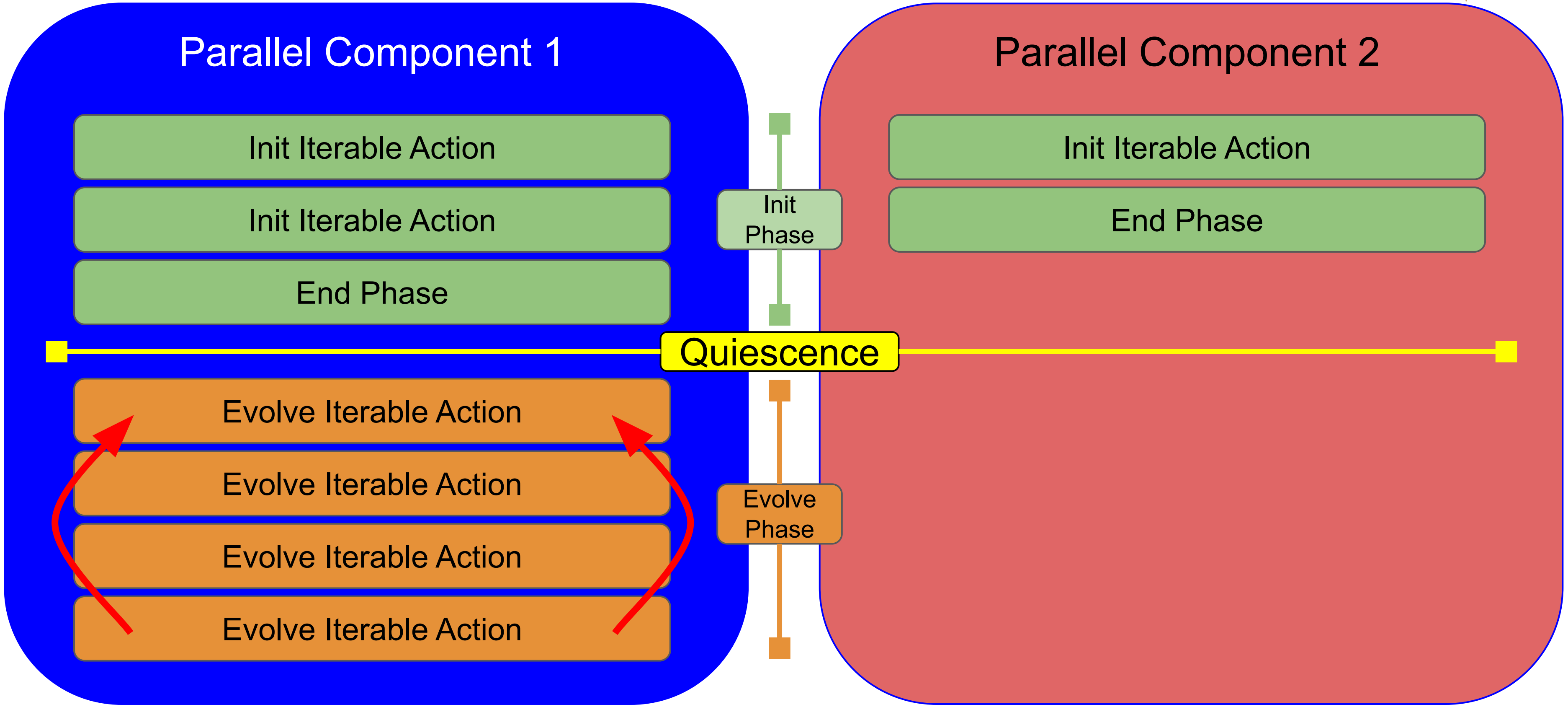
- Bulk of actual computation
- Algorithm continuously repeats
  - Until paused/terminated
  - Can be restarted
- Examples
  - Time derivative
  - Time step
  - Elliptic iteration

When to run Actions

SpECTRE jargon: Phase

- Periods when certain actions can be run
- Typical phases:
  - *Initialization*
  - *Registration*
  - *Evolution/Solve*
- How do phases end?
  - No “main”
  - No continuous code flow
  - Quiescence!

We need a picture of The Algorithm



Global Data

Storing global data

SpECTRE jargon: Global Cache

- Global info is constant
- Examples
  - # cores/nodes
  - Parallel component proxies
  - Domain info (blocks)
  - Coordinate maps
- Nodegroup
- Accessible most everywhere
  - All actions

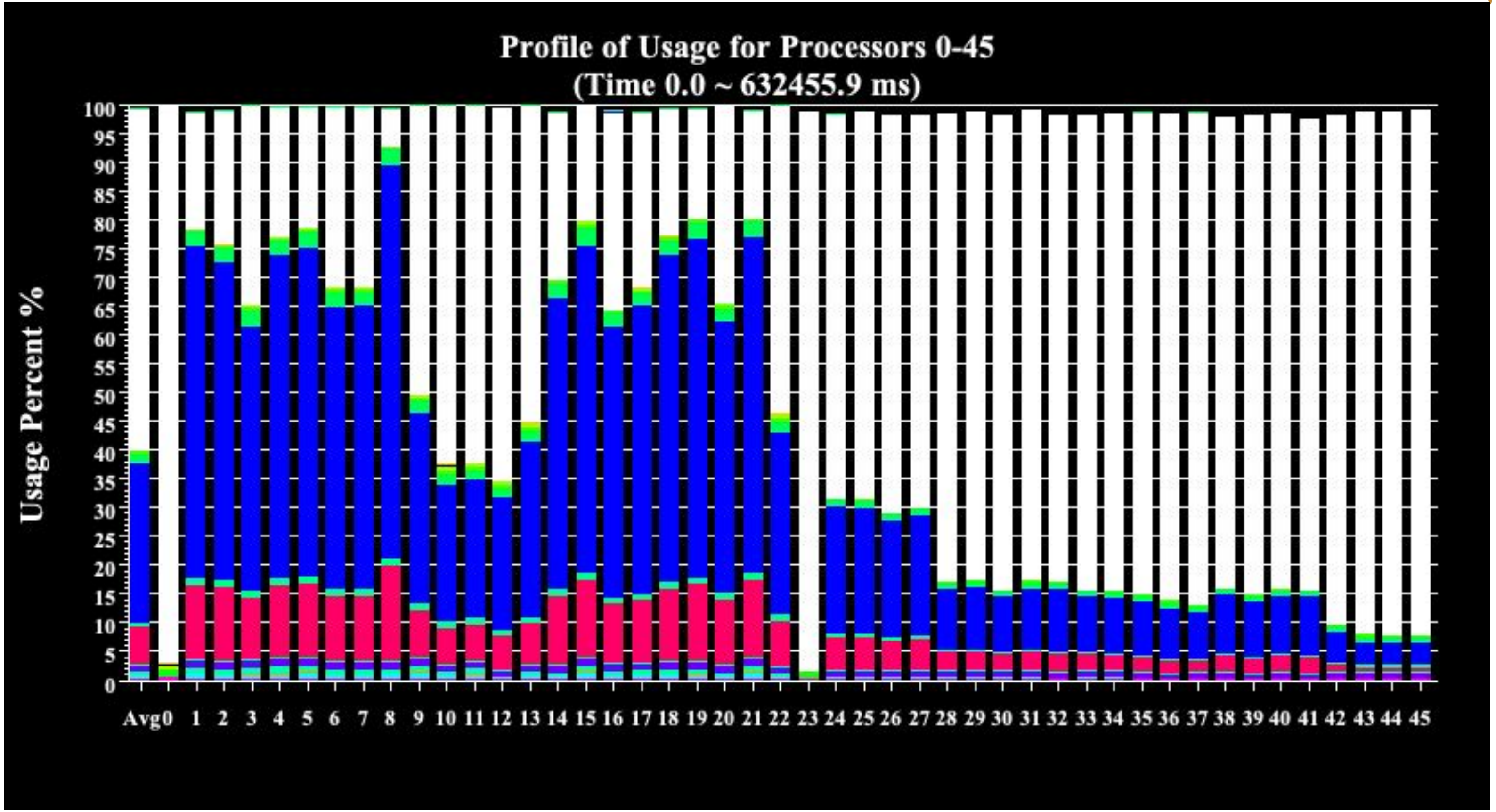
Changing global data

SpECTRE jargon: Mutable Global Cache

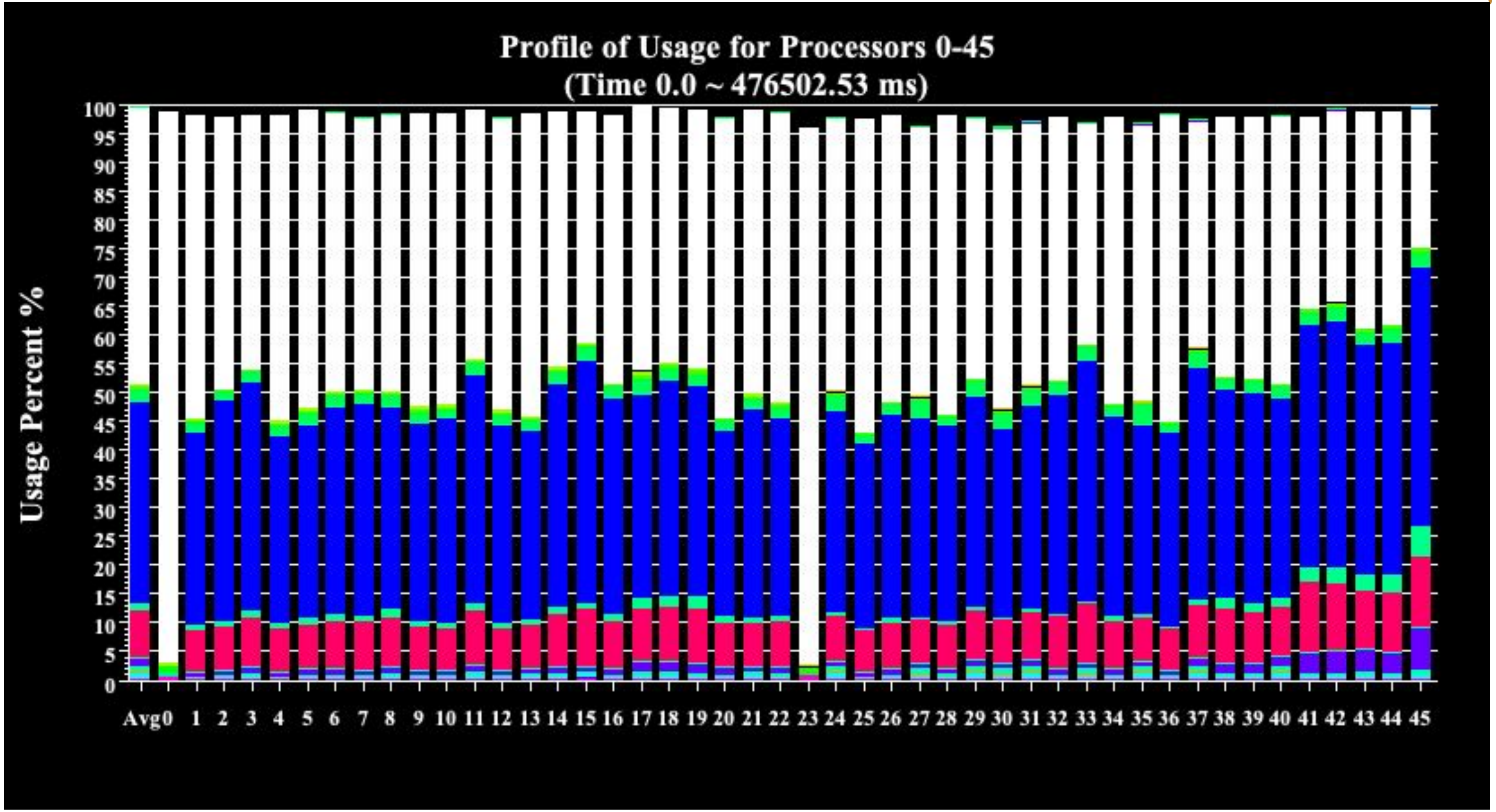
- Antithetical to async paradigm
- Avoid as much as possible
- Time dependent coordinate map parameters
- Many considerations when implementing
  - Synchronization
  - Concurrence
  - Race conditions
  - Messages out of order

Measuring Asynchronicity

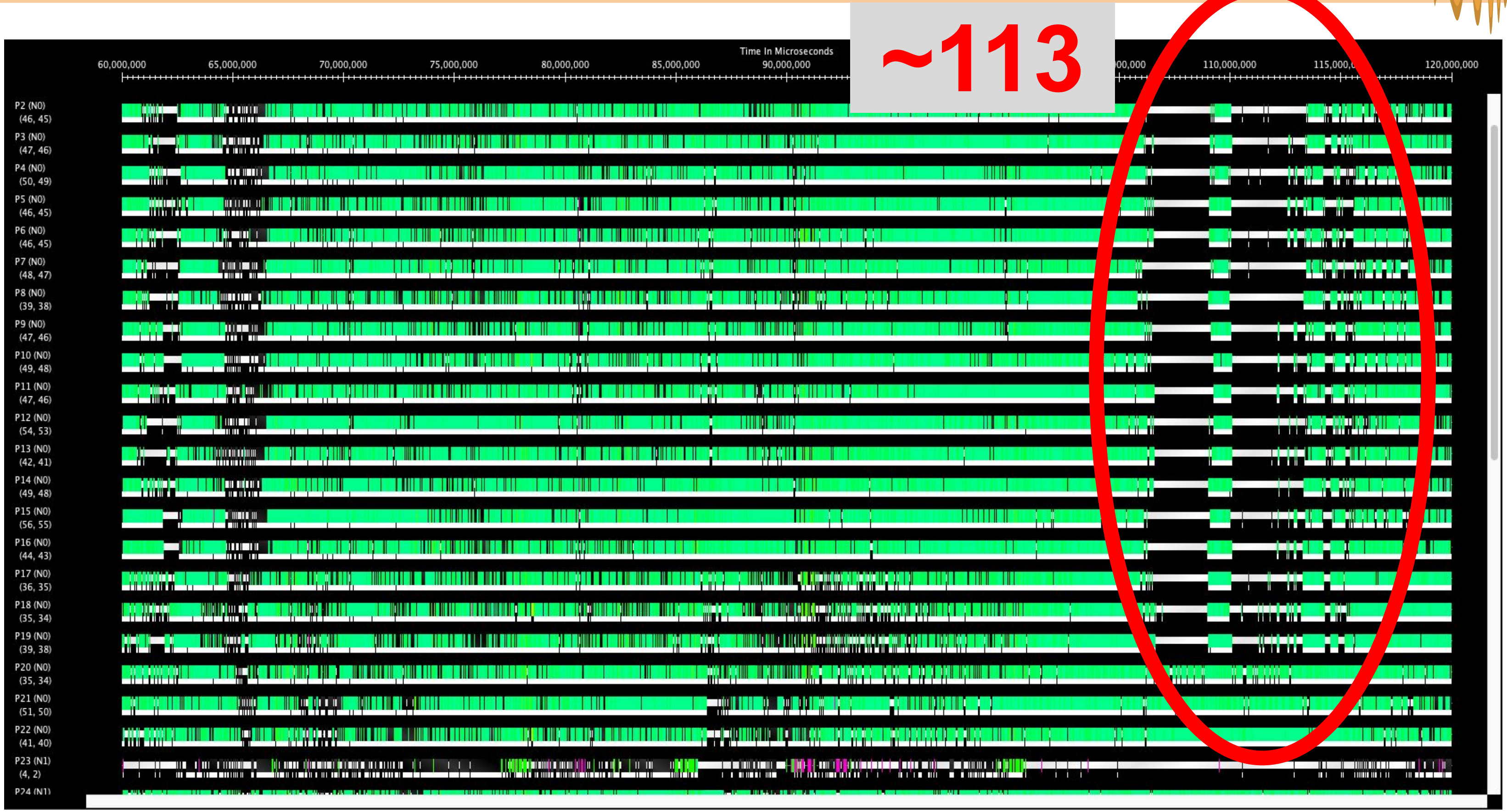
Utilization



Utilization



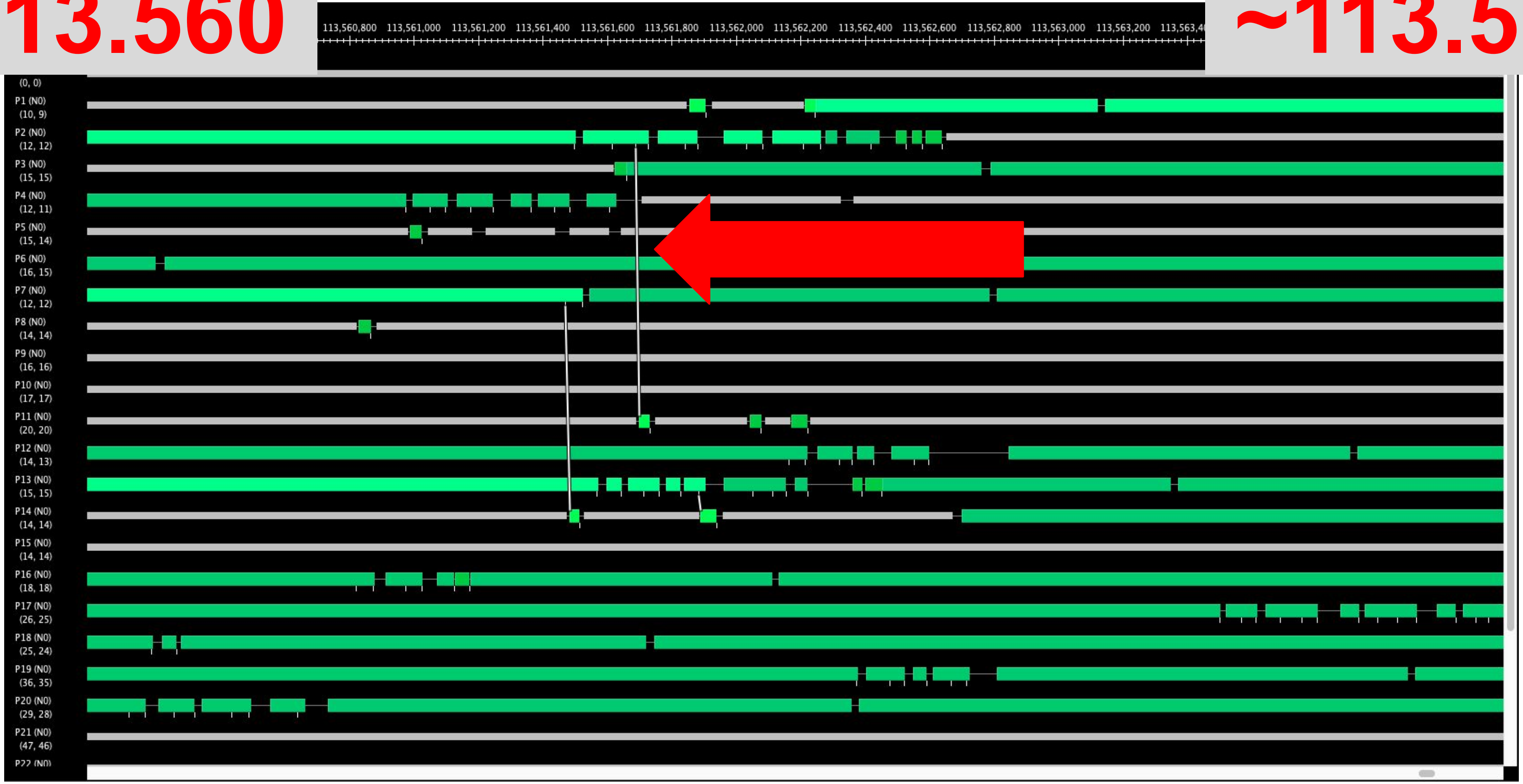
Timelines


~113
Time In Microseconds
60,000,000
65,000,000
70,000,000
75,000,000
80,000,000
85,000,000
90,000,000
110,000,000
120,000,000
P2 (N0) (46, 45)
P3 (N0) (47, 46)
P4 (N0) (50, 49)
P5 (N0) (46, 45)
P6 (N0) (46, 45)
P7 (N0) (48, 47)
P8 (N0) (39, 38)
P9 (N0) (47, 46)
P10 (N0) (49, 48)
P11 (N0) (47, 46)
P12 (N0) (54, 53)
P13 (N0) (42, 41)
P14 (N0) (49, 48)
P15 (N0) (56, 55)
P16 (N0) (44, 43)
P17 (N0) (36, 35)
P18 (N0) (35, 34)
P19 (N0) (39, 38)
P20 (N0) (35, 34)
P21 (N0) (51, 50)
P22 (N0) (41, 40)
P23 (N1) (4, 2)
P24 (N1)

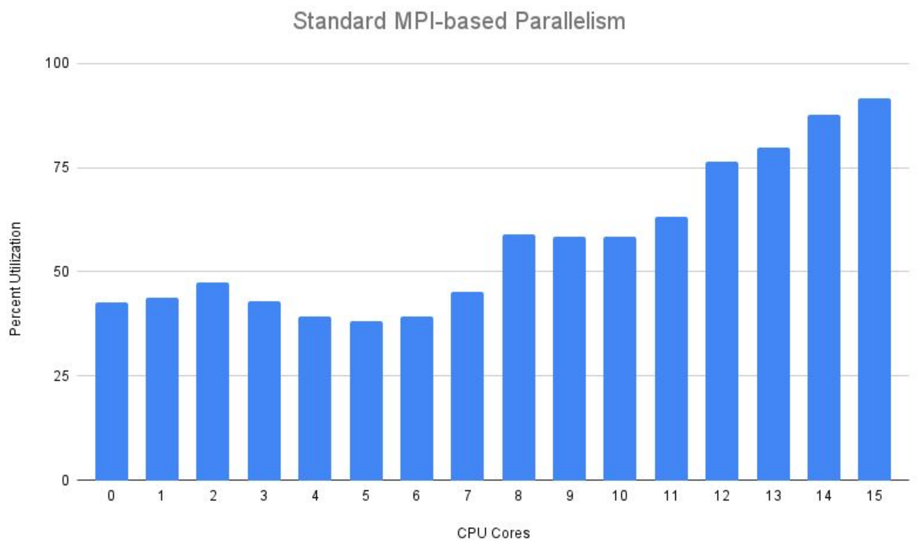
Timelines

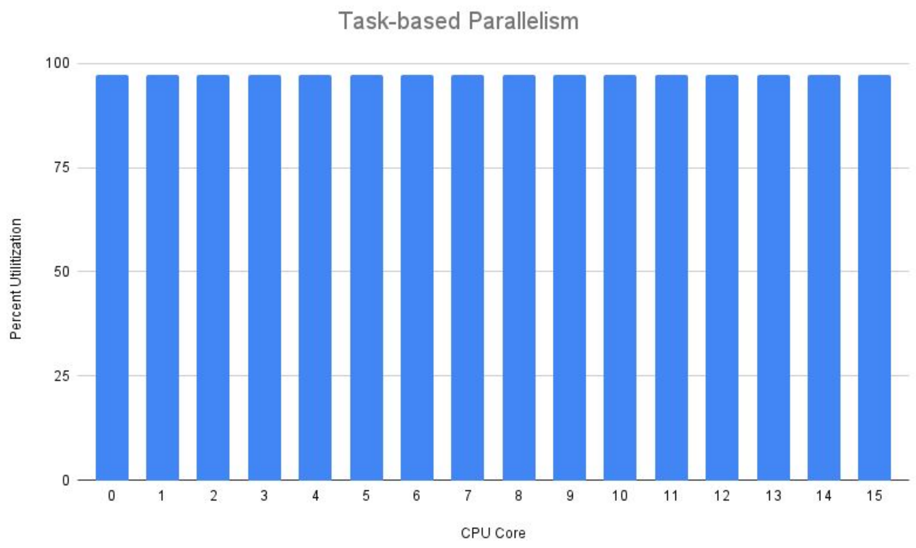
~113.560

~113.564



Comparison


Standard MPI-based Parallelism
Percent Utilization
100
75
50
25
0
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
CPU Cores


Task-based Parallelism
Percent Utilization
100
75
50
25
0
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
CPU Core

Overview

- *Parallel component*
- *Action*
- *Proxy*
- *The Algorithm*
- *Phase*
- *(Mutable) Global Cache*

Thank you!