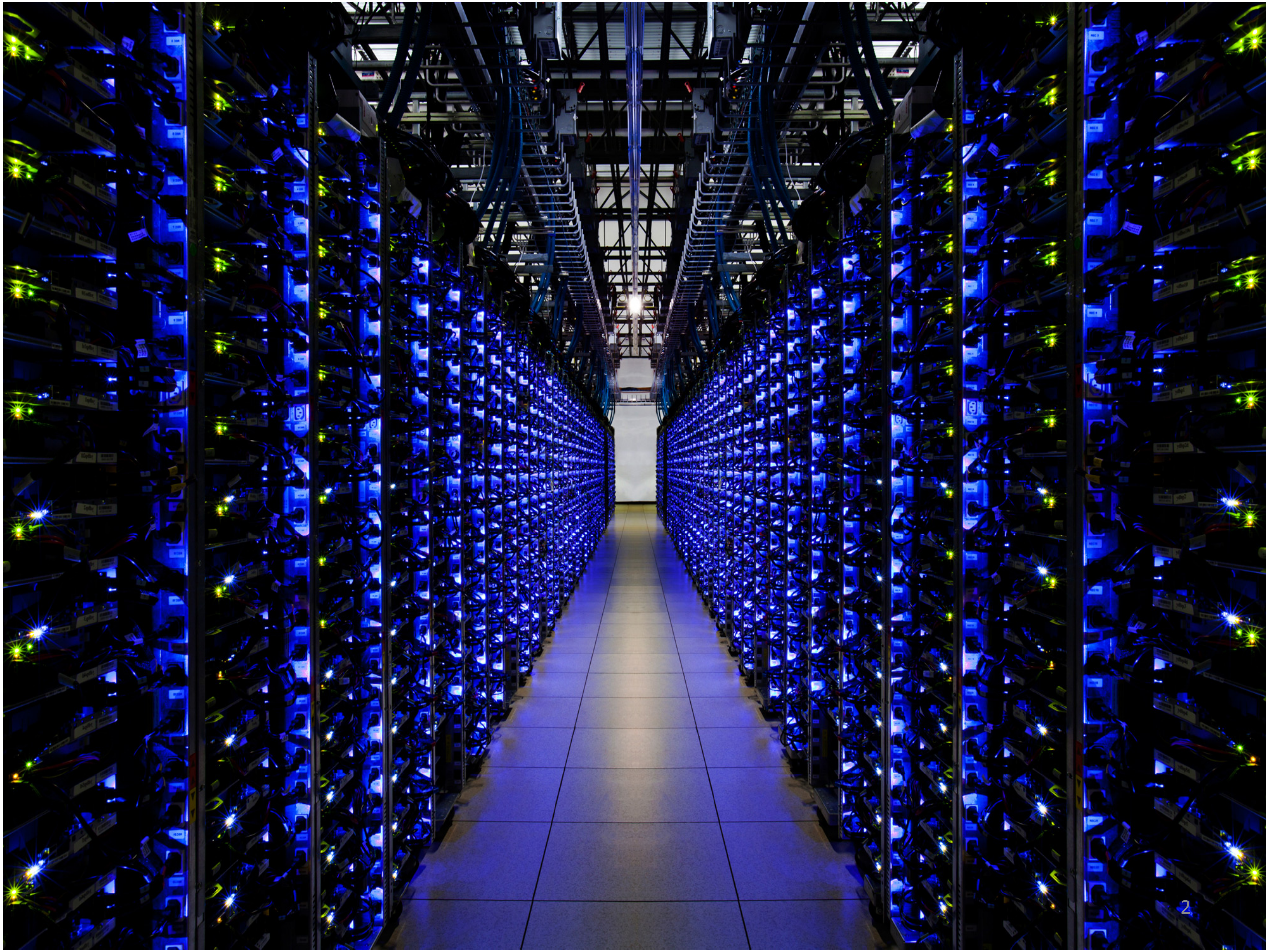
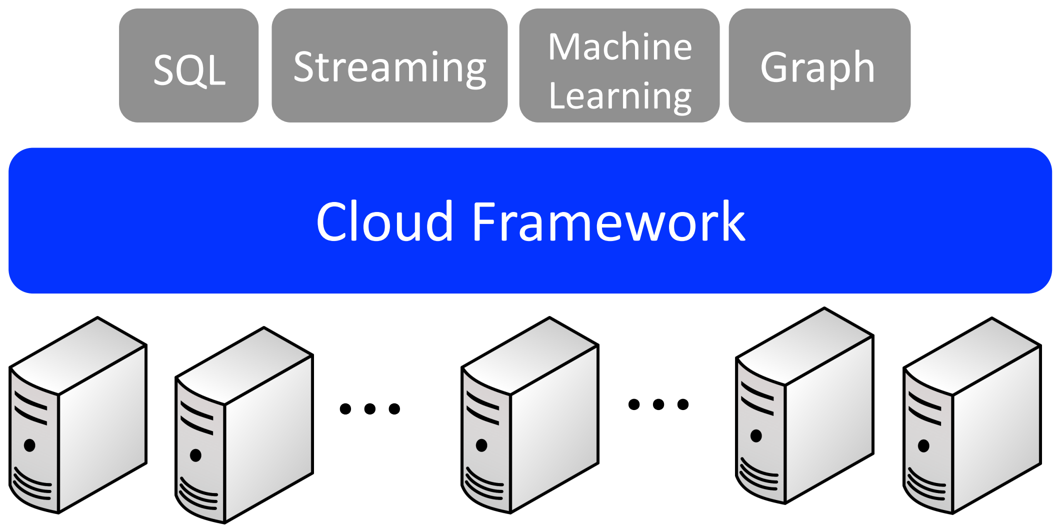
Fast, Distributed Computations in the Cloud

Omid Mashayekhi
Advisor: Philip Levis

April 7, 2017



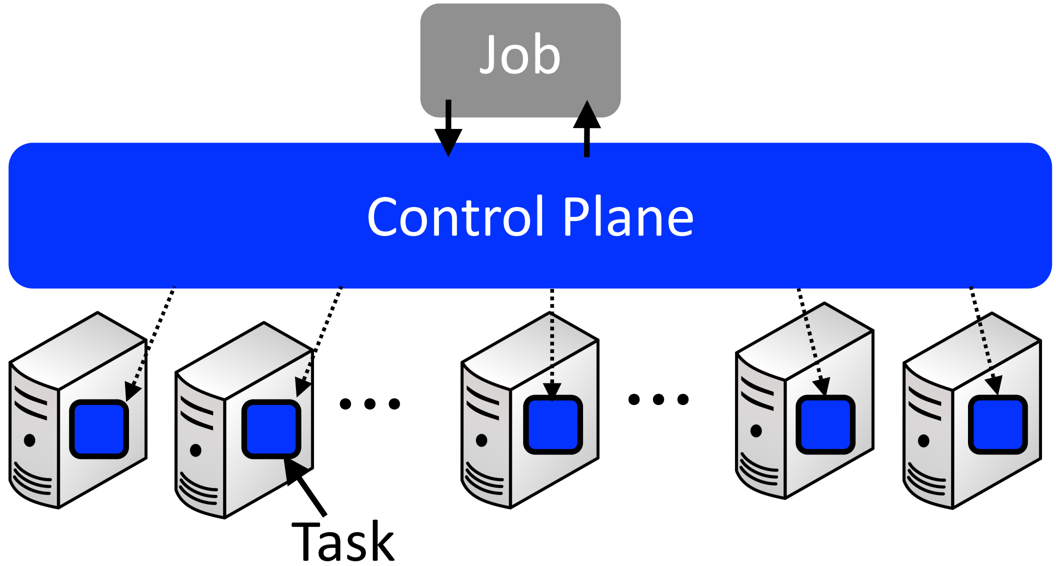
Cloud Frameworks



Cloud frameworks abstract away the complexities of the cloud infrastructure from the application developers:

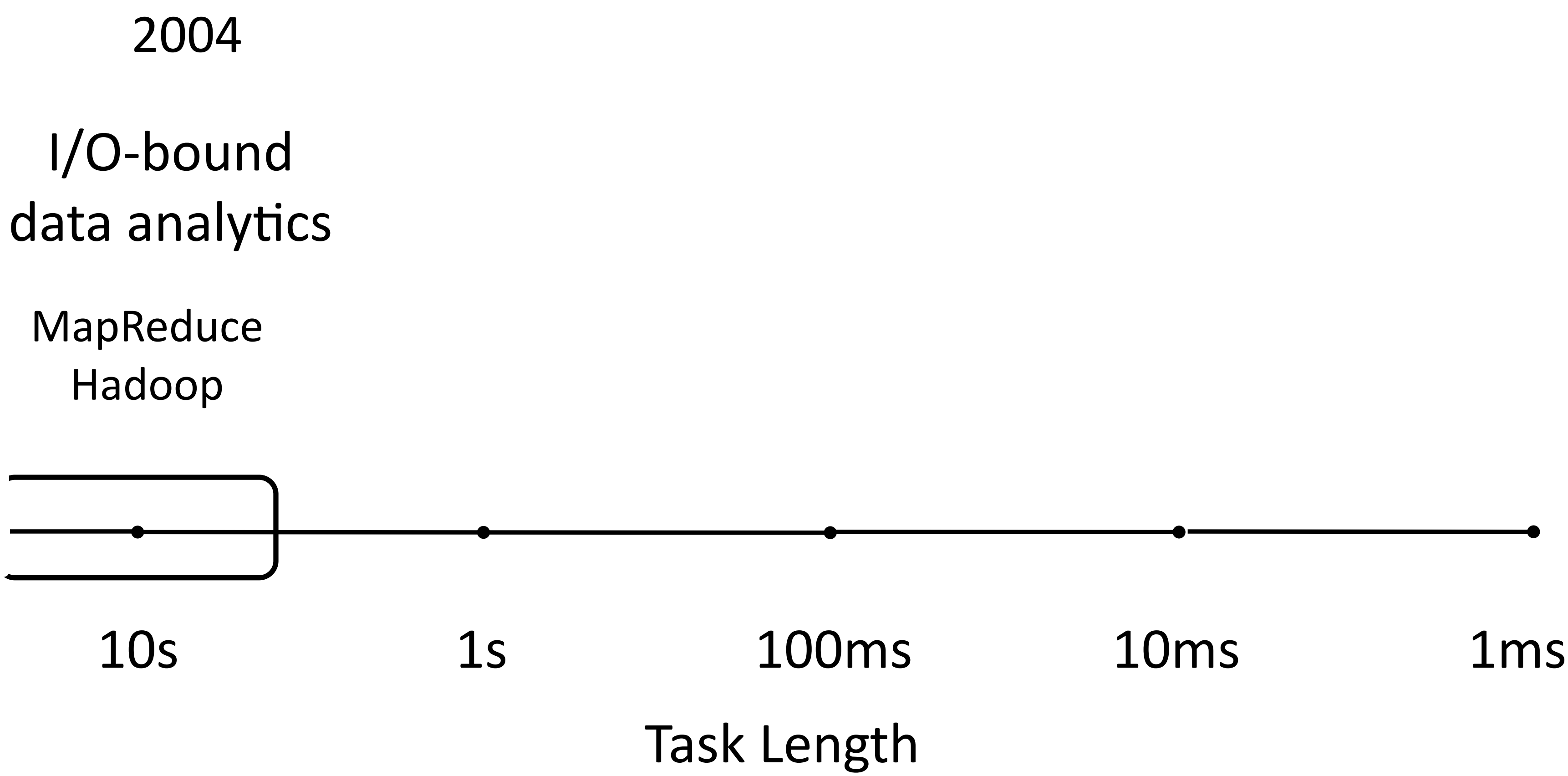
1. Automatic distribution
2. Elastic scalability
3. Multitenant applications
4. Load balancing
5. Fault tolerance

Cloud Frameworks



- **Job** is an instance of the application running in the framework.
- **Task** is the unit of computation.
- **Control plane** makes the magic happen:
  - Partitioning job in to tasks
  - Scheduling tasks
  - Load balancing
  - Fault recovery

Evolution of Cloud Frameworks



Evolution of Cloud Frameworks

2004

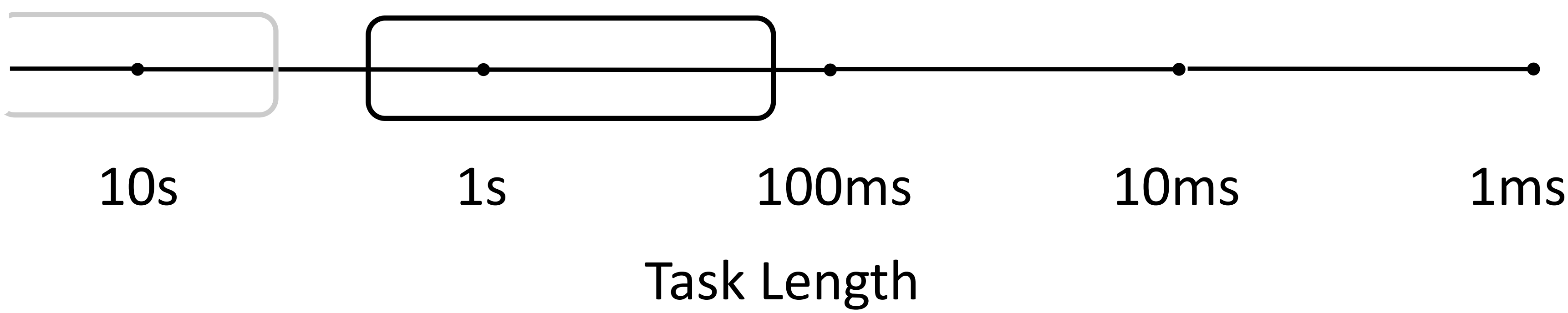
I/O-bound
data analytics

MapReduce
Hadoop

2012

In-memory
data analytics

Spark
Naiad



Evolution of Cloud Frameworks

2004

I/O-bound
data analytics

MapReduce
Hadoop

2012

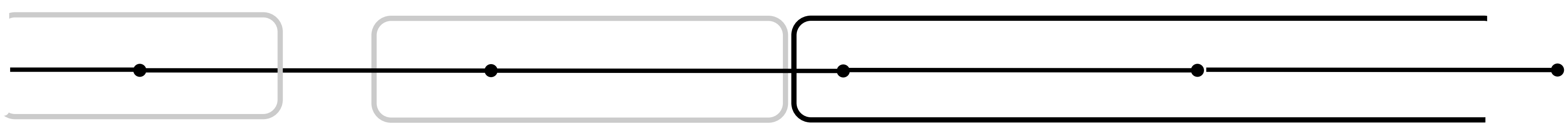
In-memory
data analytics

Spark
Naiad

2016

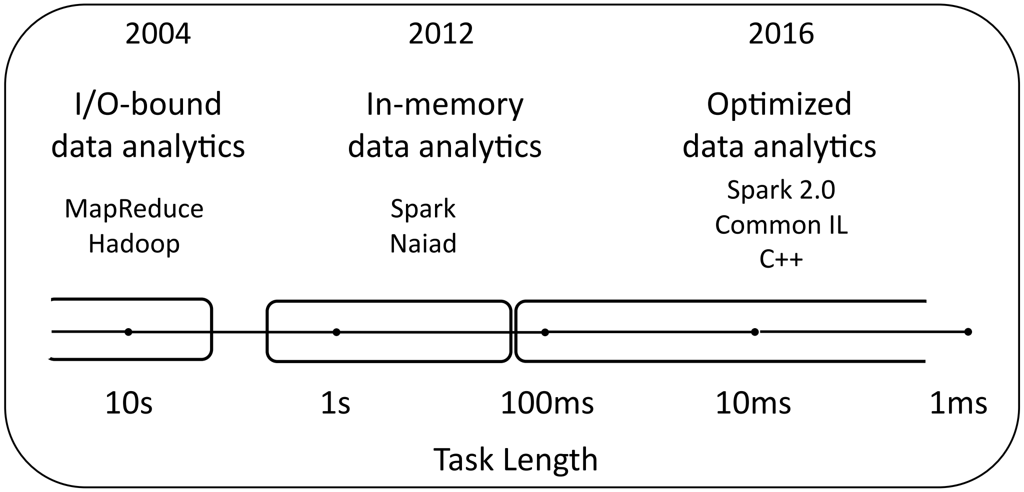
Optimized
data analytics

Spark 2.0
Common IL
C++

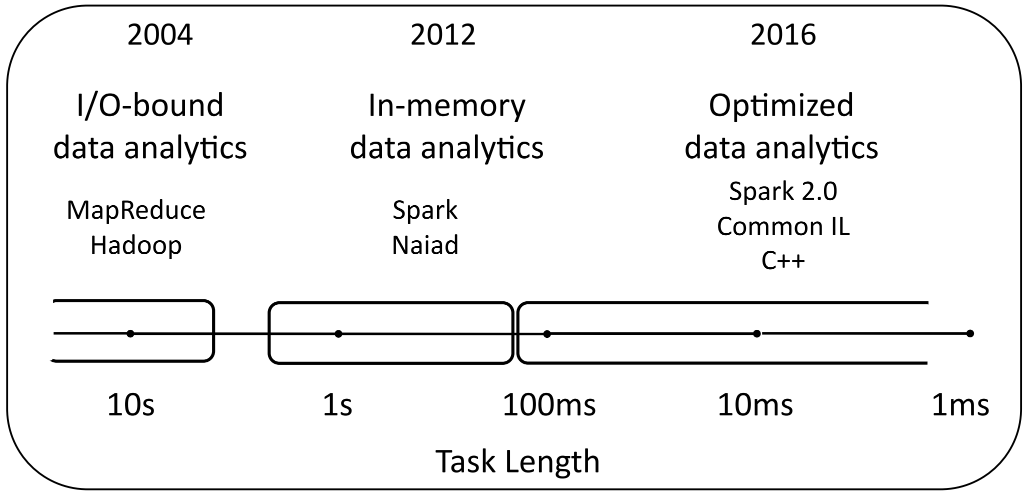


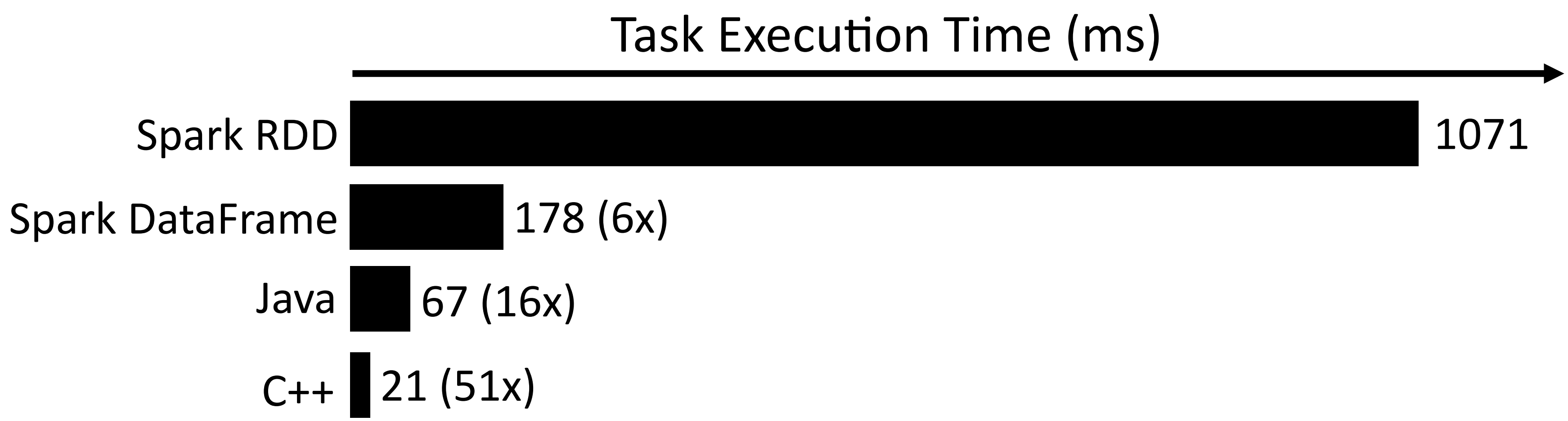
10s 1s 100ms 10ms 1ms

Task Length



2004
2012
2016
I/O-bound data analytics
In-memory data analytics
Optimized data analytics
MapReduce
Hadoop
Spark
Naiad
Spark 2.0
Common IL
C++
10s
1s
100ms
10ms
1ms
Task Length



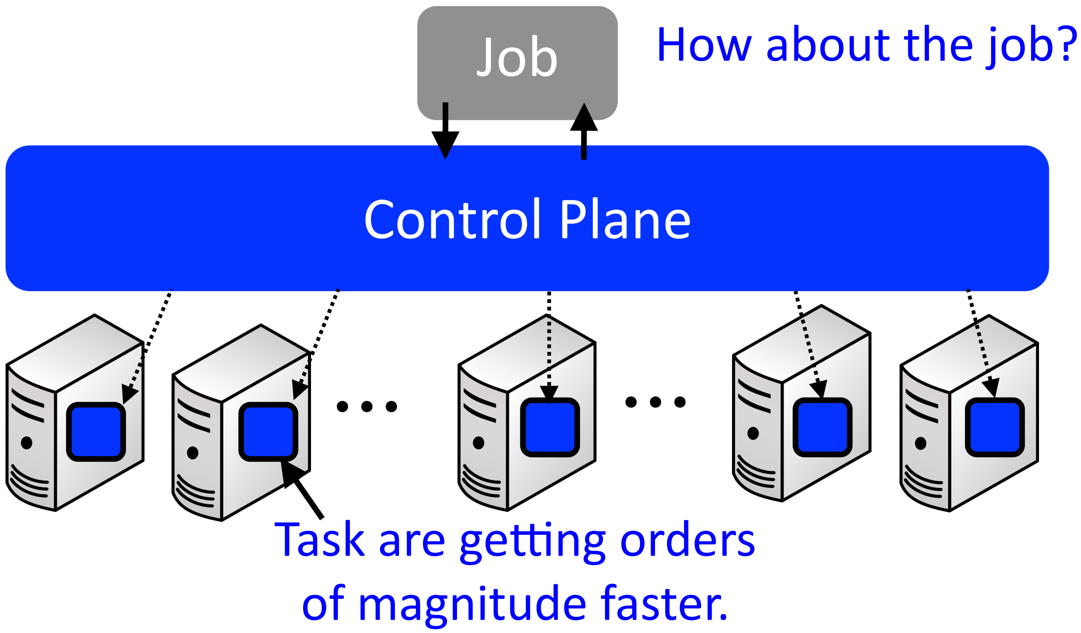


- One iteration of logistic regression over a data set of size 64MB.
- Tasks implemented efficiently, could run **50x** faster.

Individual tasks are getting faster.

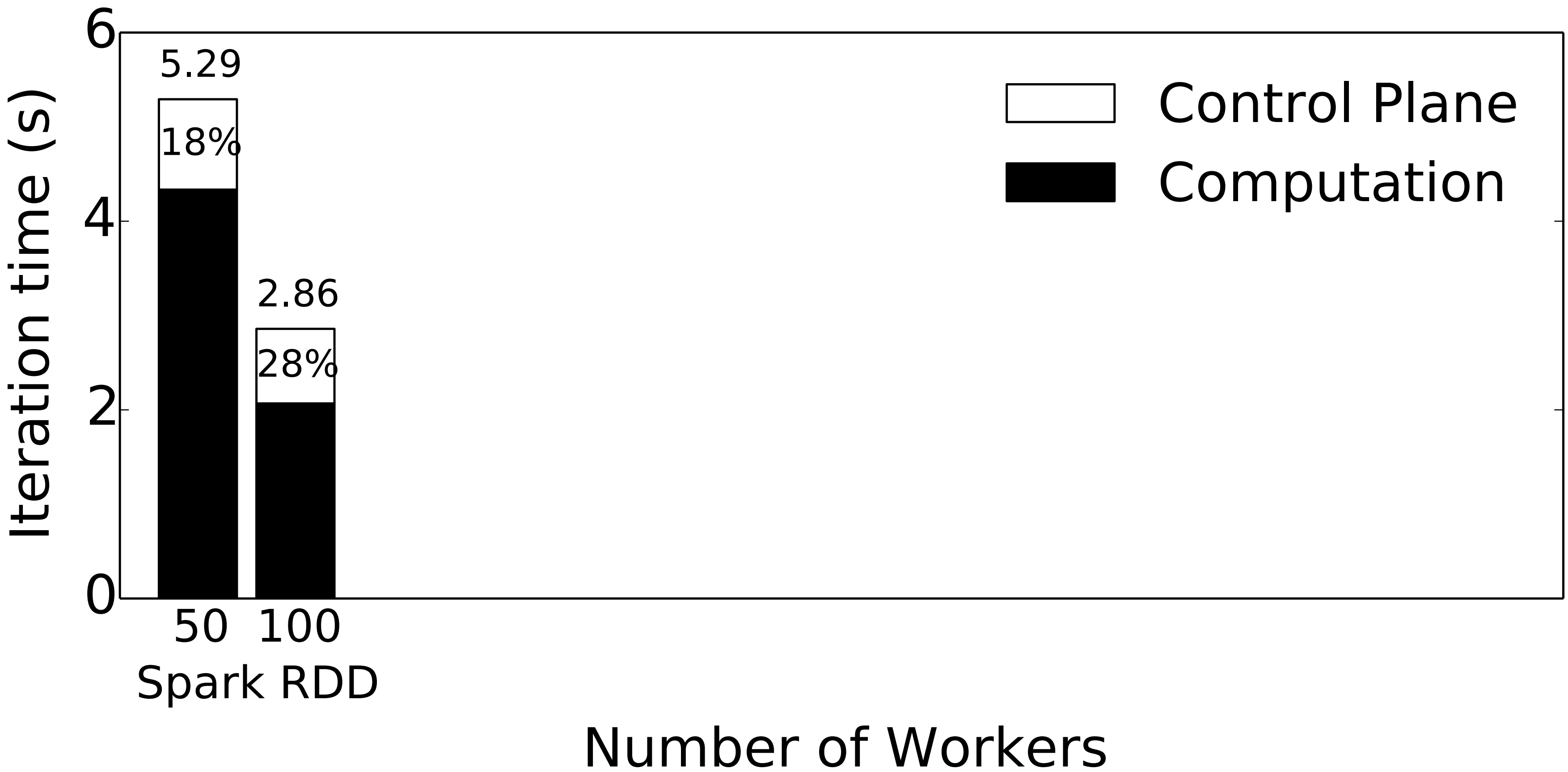
But does it necessarily mean that
job completion time is getting shorter?

Cloud Frameworks



Control Plane

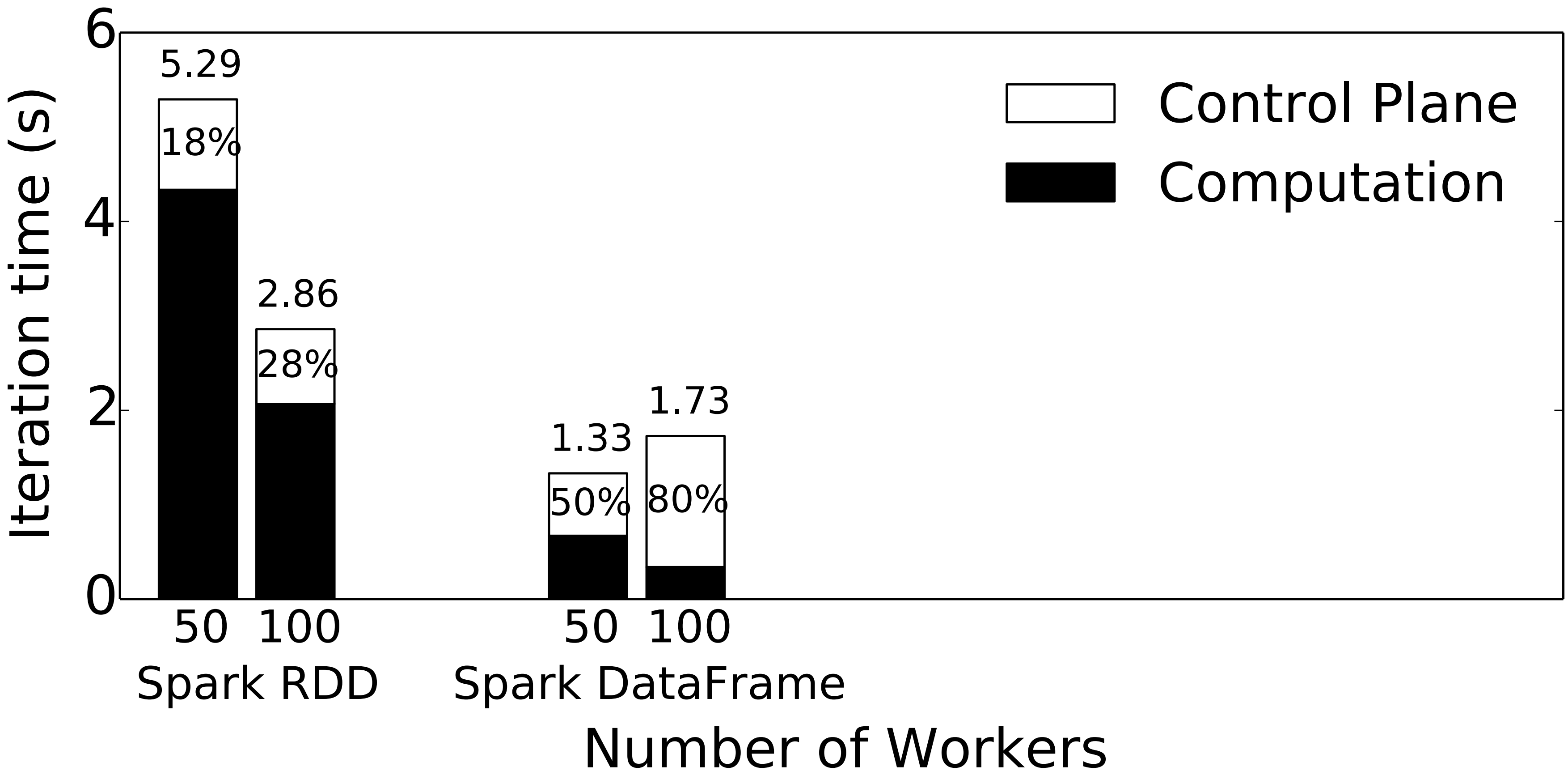
The New Bottleneck



- Logistic regression over a data set of size 100GB.
- Classic Spark used to be **CPU-bound**.

Control Plane

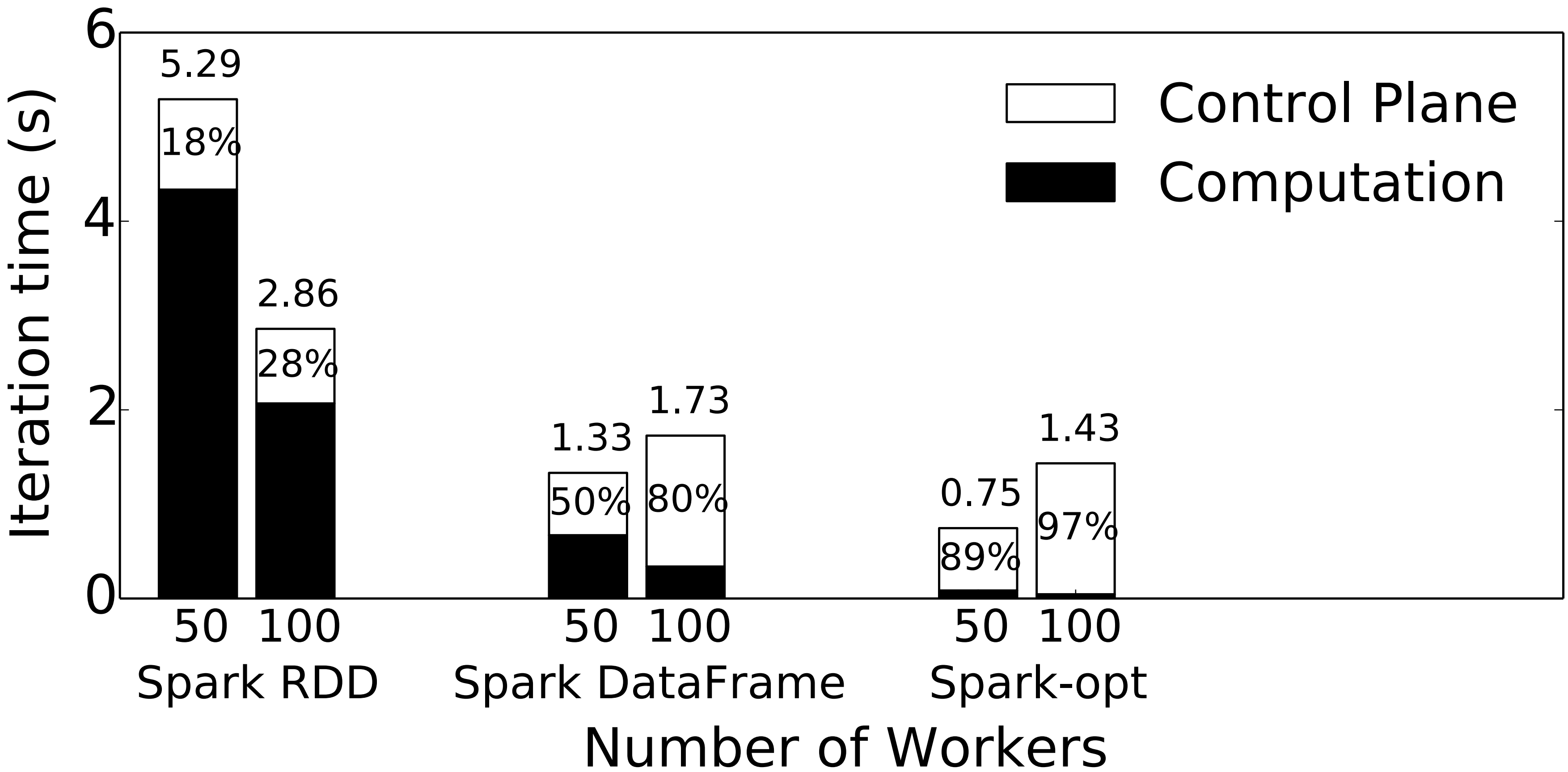
The New Bottleneck



- Logistic regression over a data set of size 100GB.
- Spark 2.0 is already **control-bound**.

Control Plane

The New Bottleneck

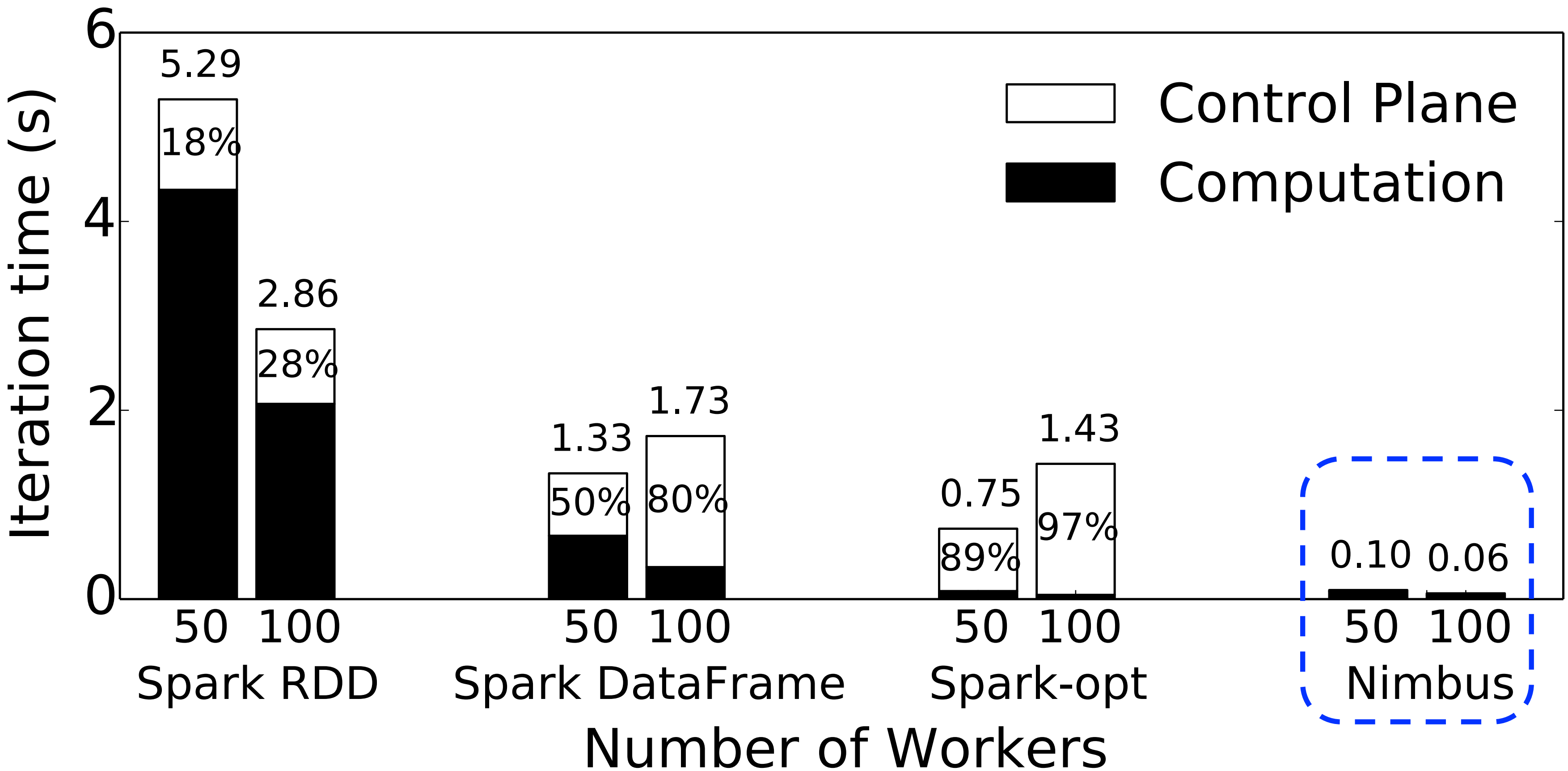


- Logistic regression over a data set of size 100GB.
- Spark-opt: hypothetical case where Spark runs tasks as fast as C++.

Control plane is the emerging bottleneck for the cloud computing frameworks.

Control Plane

The New Bottleneck



- Logistic regression over a data set of size 100GB.
- **Nimbus** with **execution templates** scales almost linearly.

Contributions

- Demonstrating how the **control plane** is the emerging **bottleneck** for data analytics frameworks.
- **Execution Templates** as an abstraction for the control plane of cloud computing frameworks, that enables orders of magnitude higher task throughput, while keeping the fine-grained, flexible scheduling.
- The design, implementation, and evaluation of **Nimbus**, a distributed cloud computing framework that embeds execution templates.
- A demonstration of a single-core **graphical simulation** that Nimbus **automatically distributes** in the cloud showing execution templates in practice for complex applications.

This talk

- Control Plane: the Emerging Bottleneck
- Design Scope of the Control Plane
- Execution Templates
- Nimbus: a Framework with Templates
- Evaluation

This talk

- Control Plane: the Emerging Bottleneck
- Design Scope of the Control Plane
- Execution Templates
- Nimbus: a Framework with Templates
- Evaluation

Cloud Frameworks Design

- Currently, there are two approaches:
  1. Centralized control model.
     - Controller generates and assigns tasks to the worker.
     - Limited task throughput, but reactive scheduling.
  2. Distributed data flow model.
     - Nodes generate and spawn tasks locally.
     - Great scalability, but static scheduling.

Design Spectrum

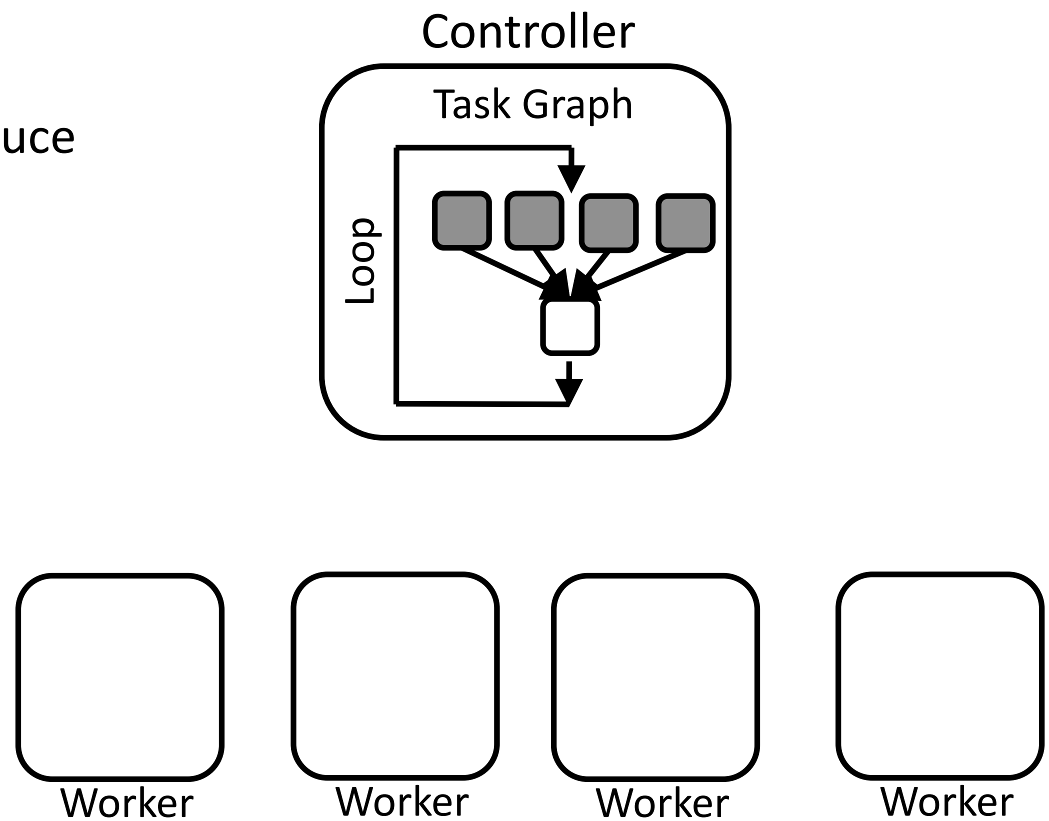
Centralized Controllers ⟷ Distributed Controllers

Design Spectrum

Centralized Controllers

Distributed Controllers

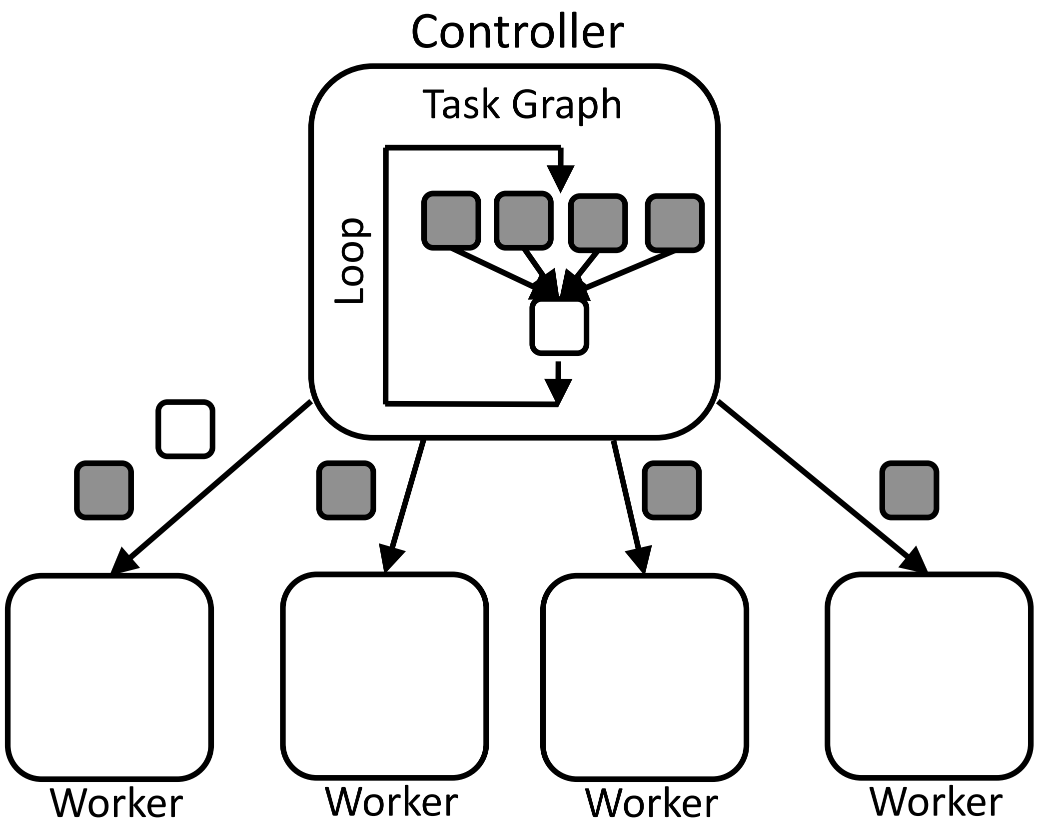
- MapReduce
- Hadoop
- Spark



- Controller centrally schedules and spawns tasks.

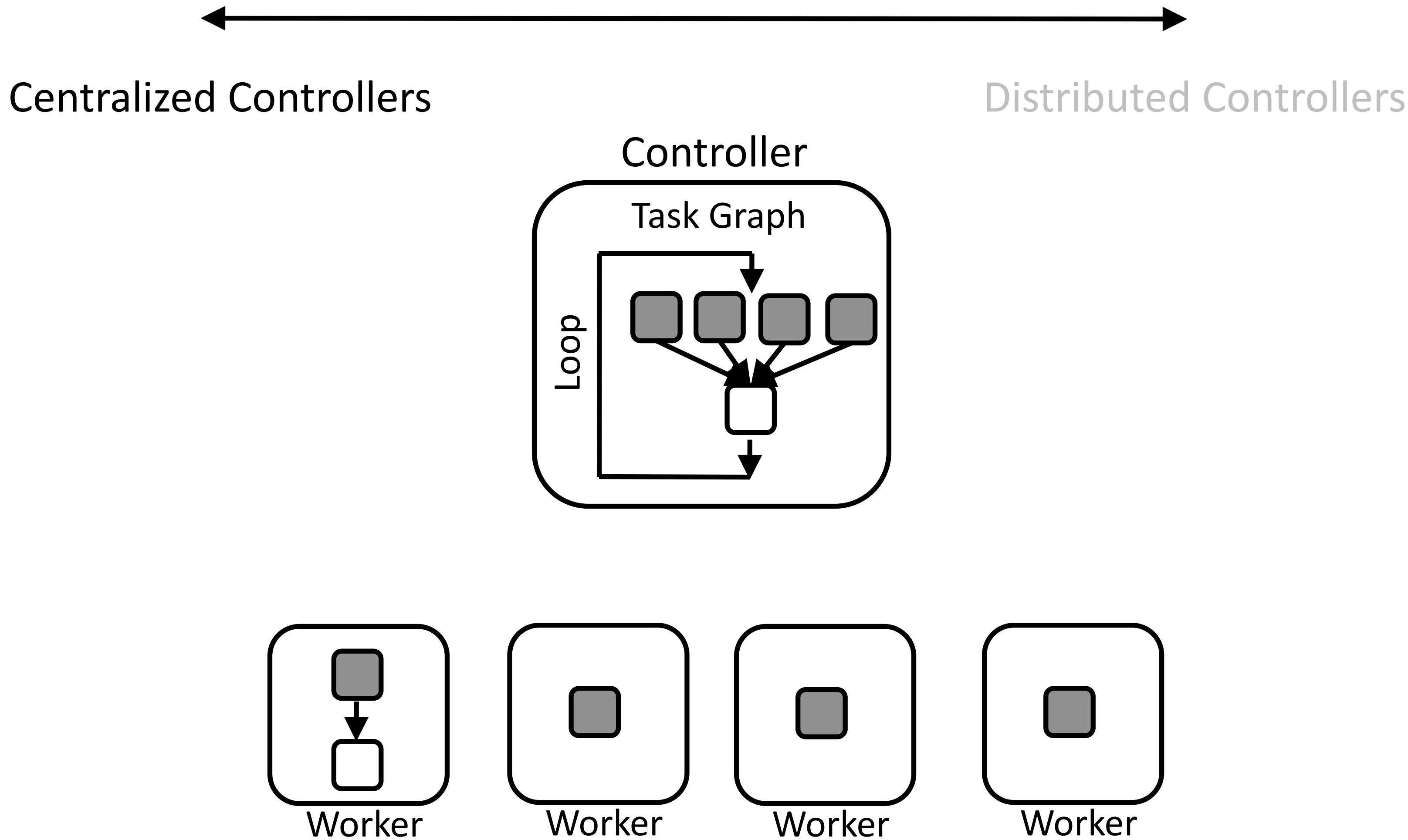
Design Spectrum

Centralized Controllers — Distributed Controllers



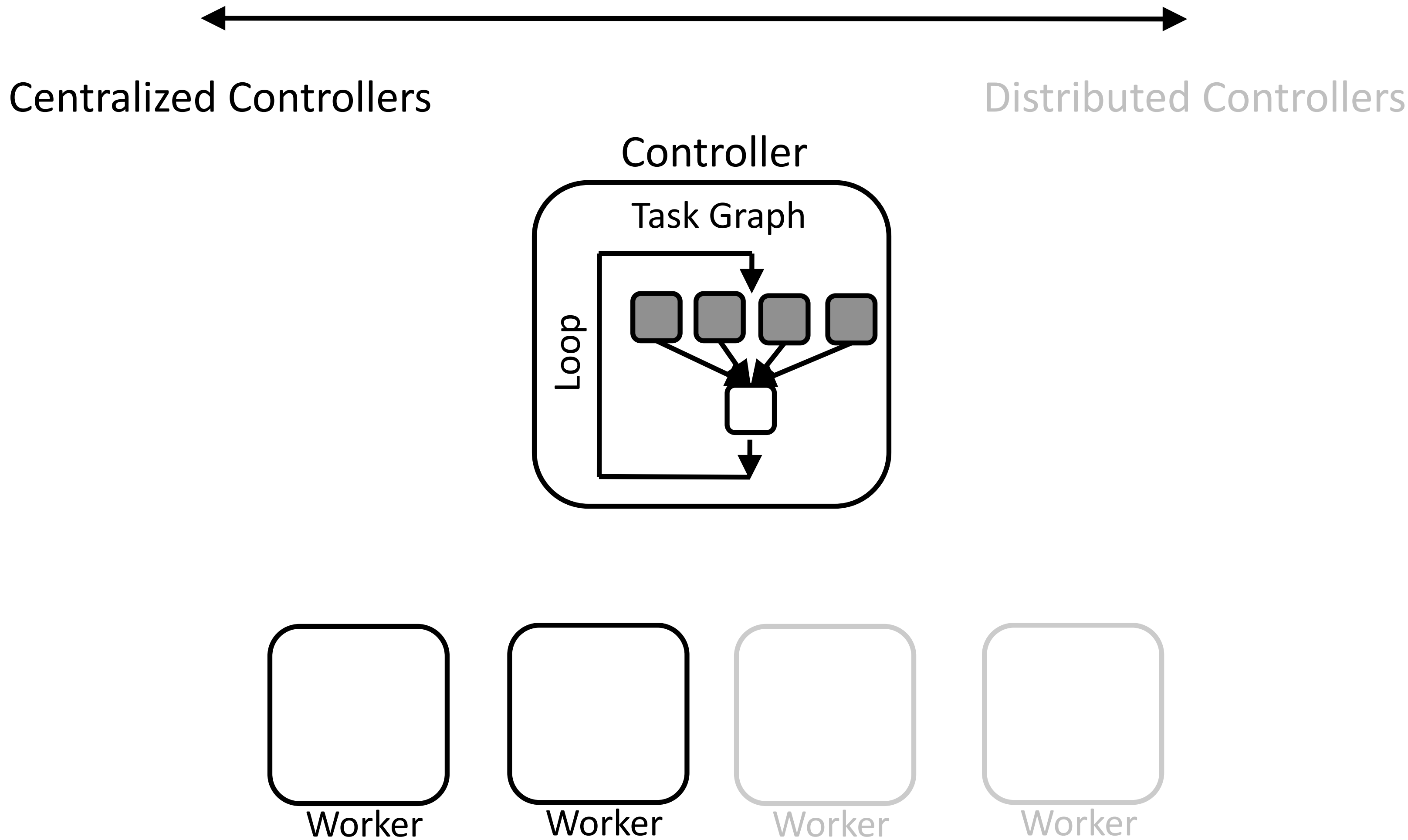
- Controller centrally schedules and spawns tasks.

Design Spectrum



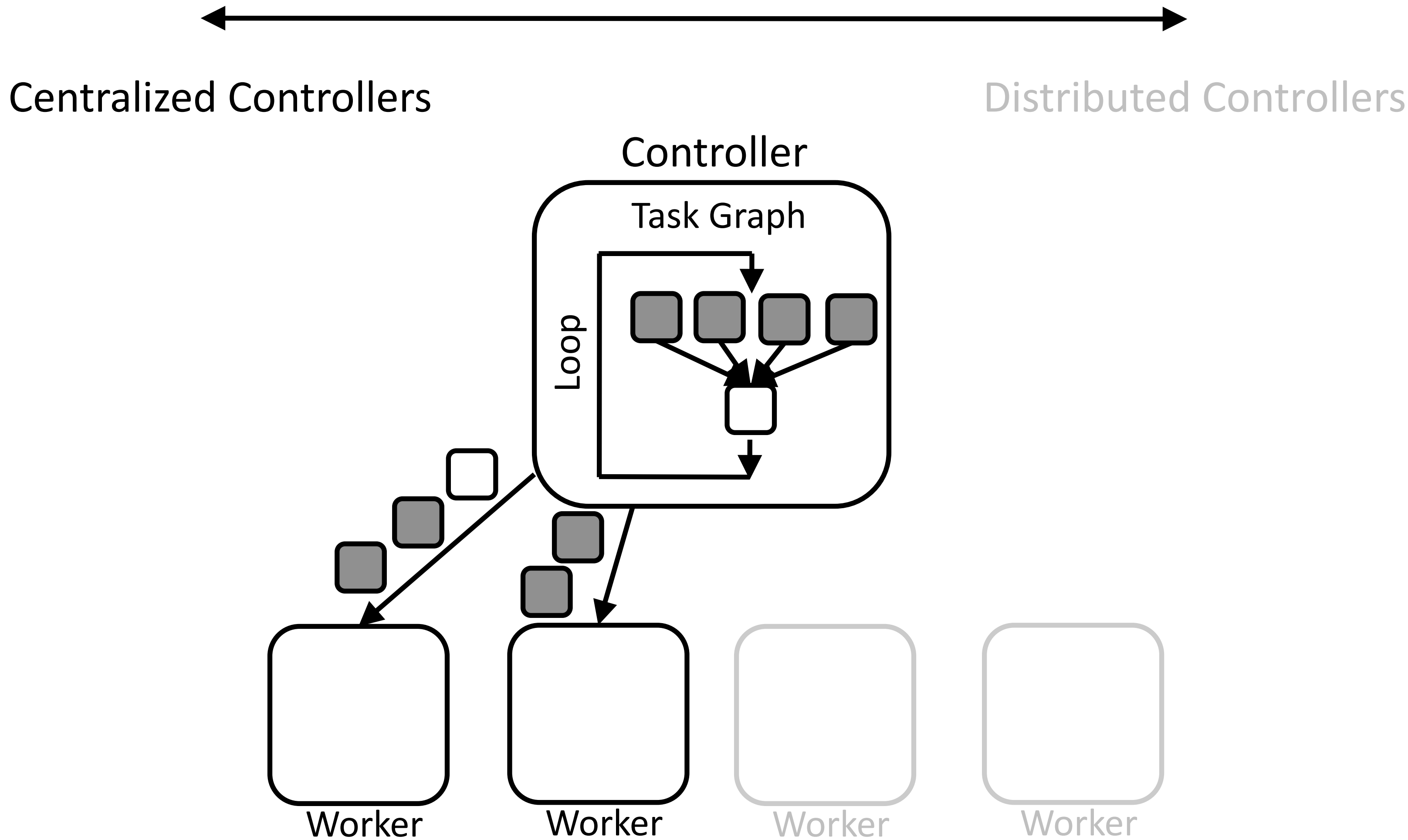
- Controller centrally schedules and spawns tasks.

Design Spectrum



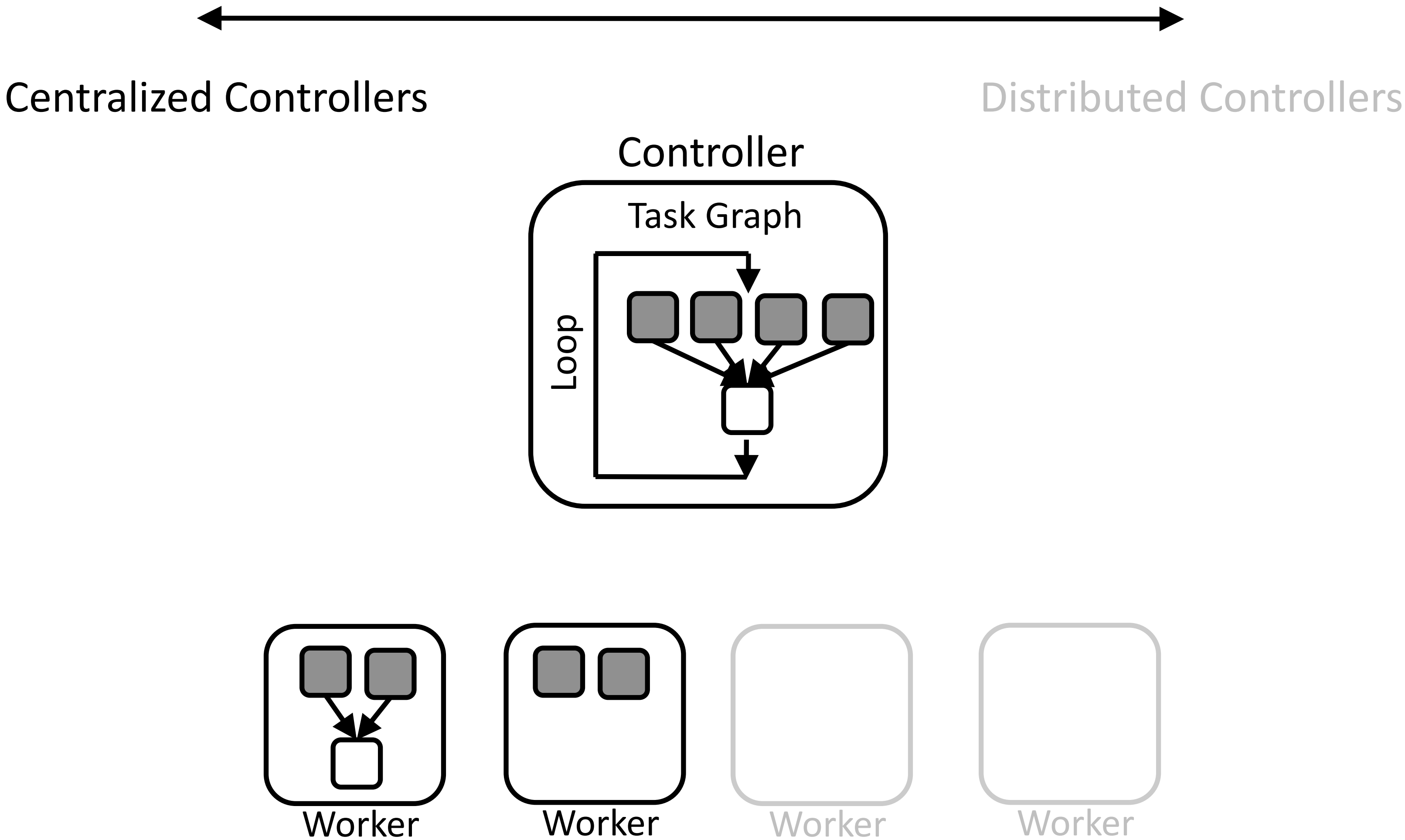
- Controller could reactively and dynamically change the schedule.

Design Spectrum



• Controller could reactively and dynamically change the schedule.

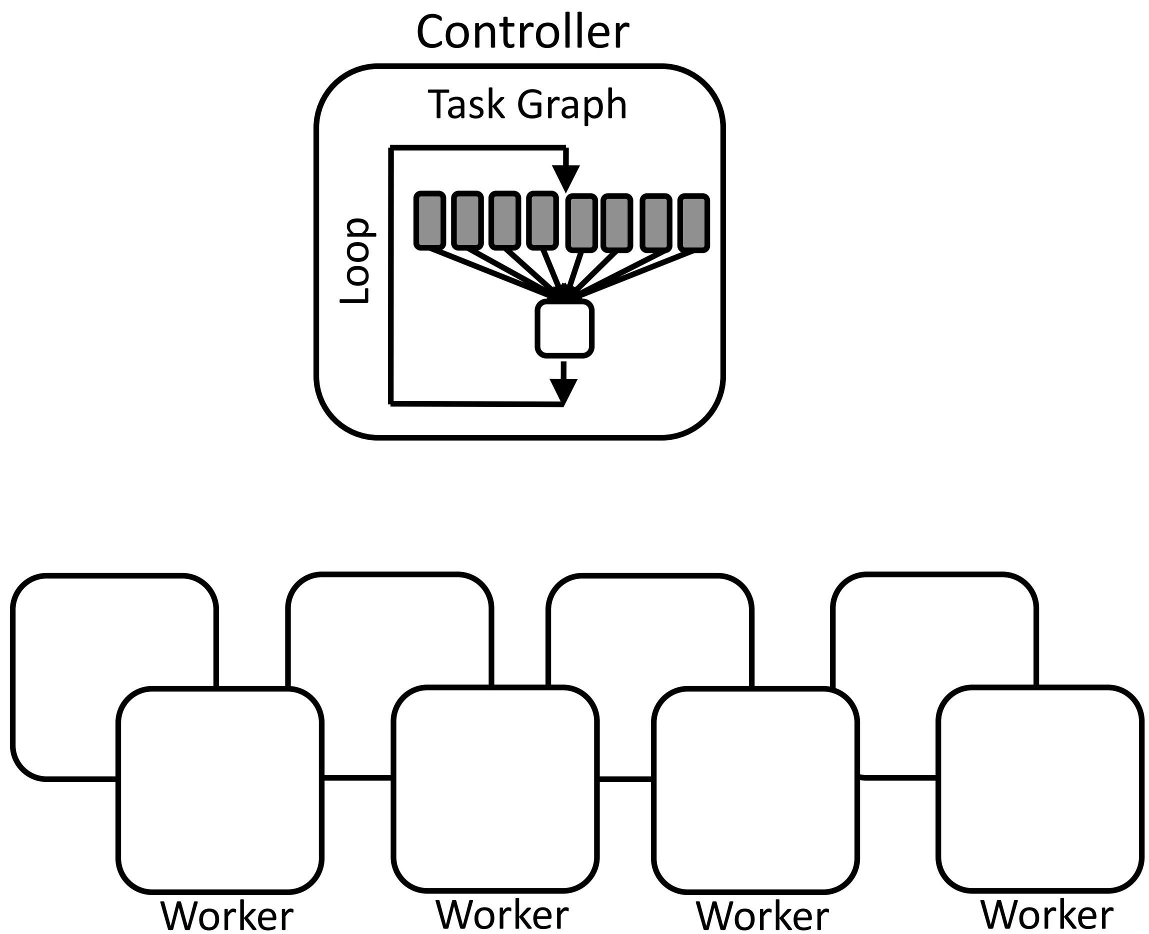
Design Spectrum



- Controller could reactively and dynamically change the schedule.

Design Spectrum

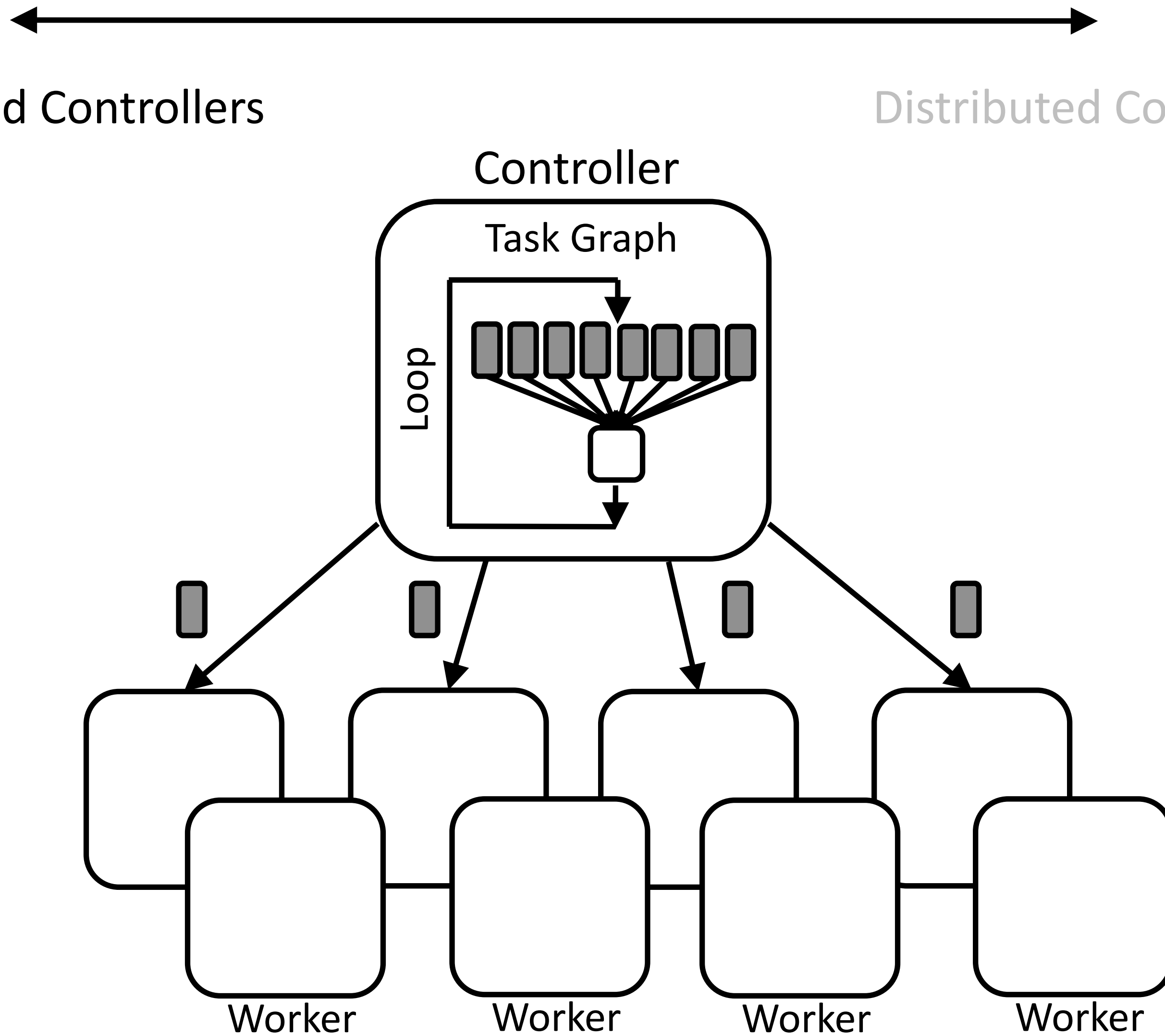
Centralized Controllers | Distributed Controllers



- But controller bottlenecks at scale.

Design Spectrum

Centralized Controllers | Distributed Controllers

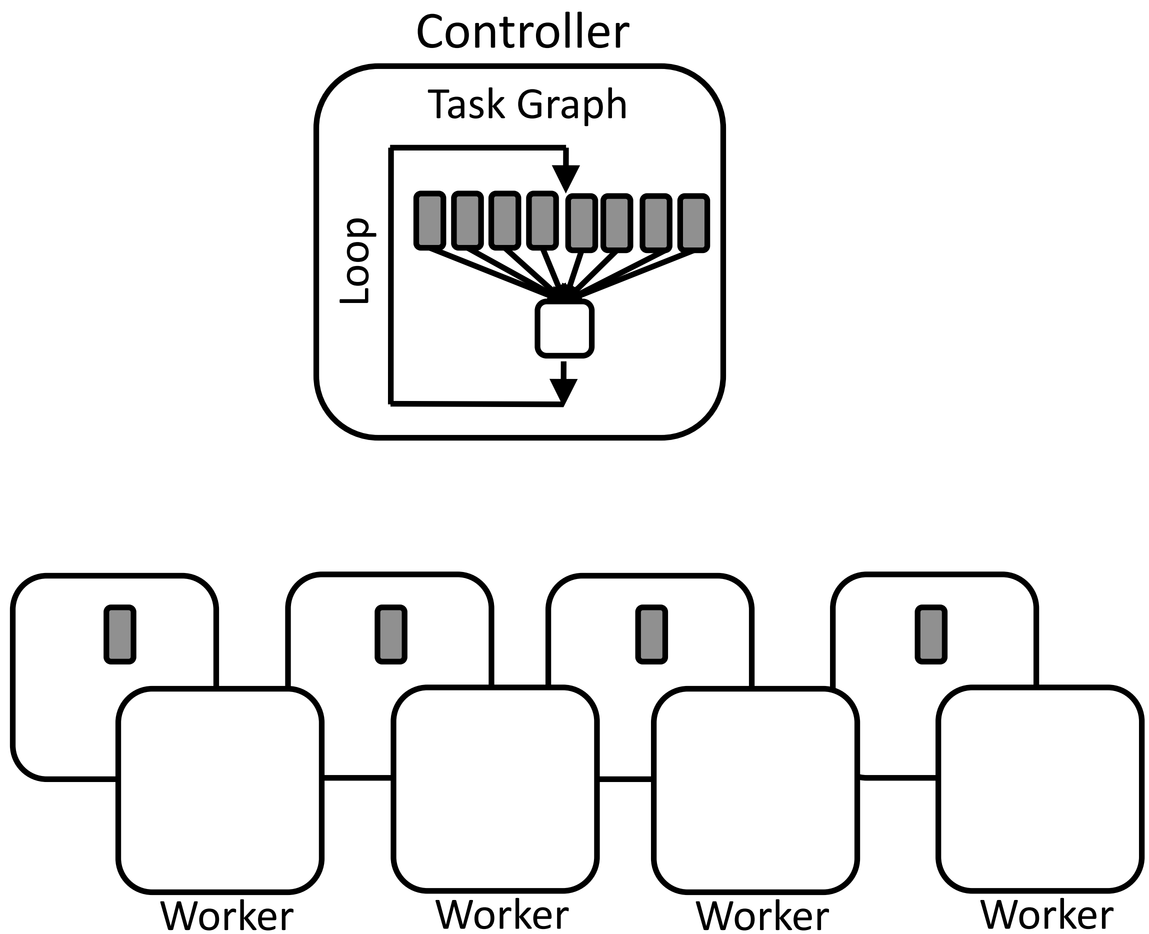


- But controller bottlenecks at scale.

Design Spectrum

Centralized Controllers

Distributed Controllers

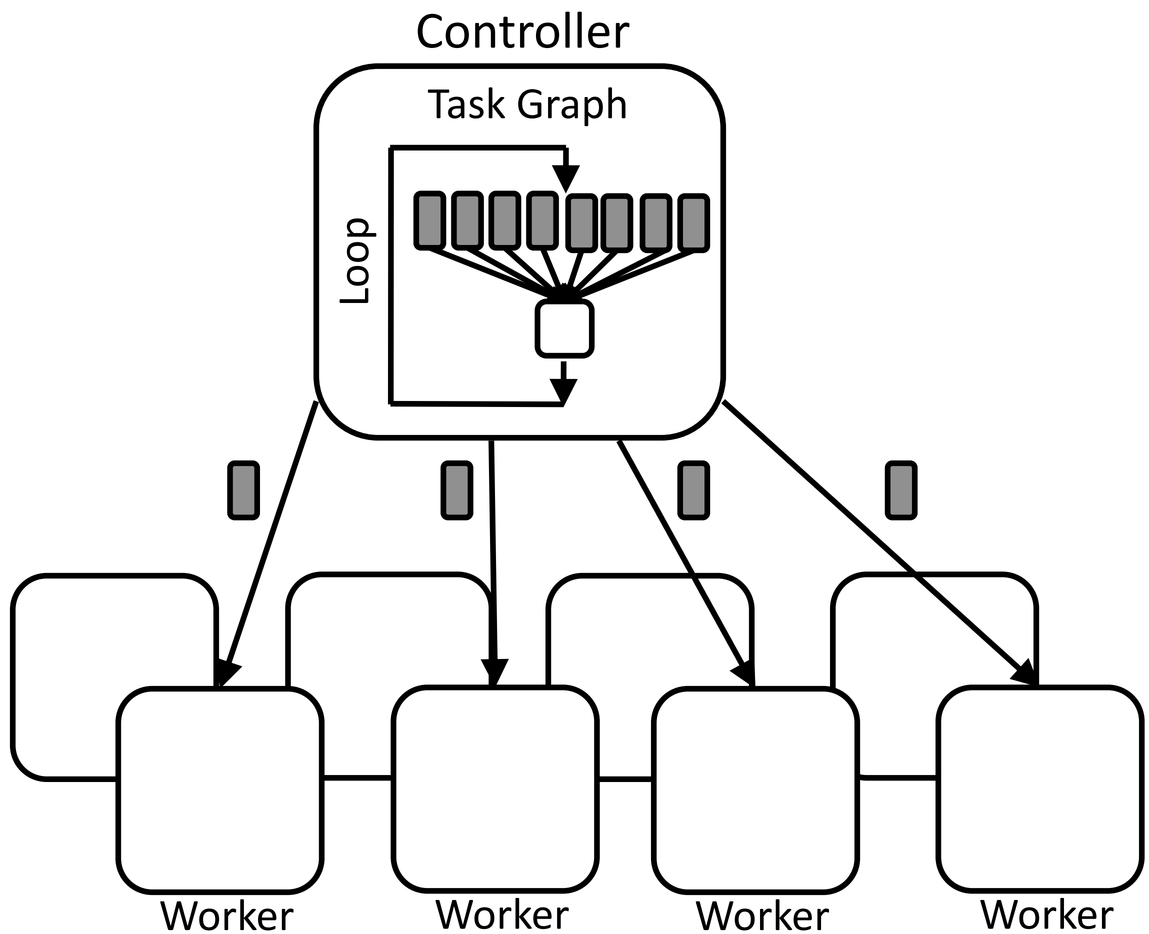


- But controller bottlenecks at scale.

Design Spectrum

Centralized Controllers

Distributed Controllers

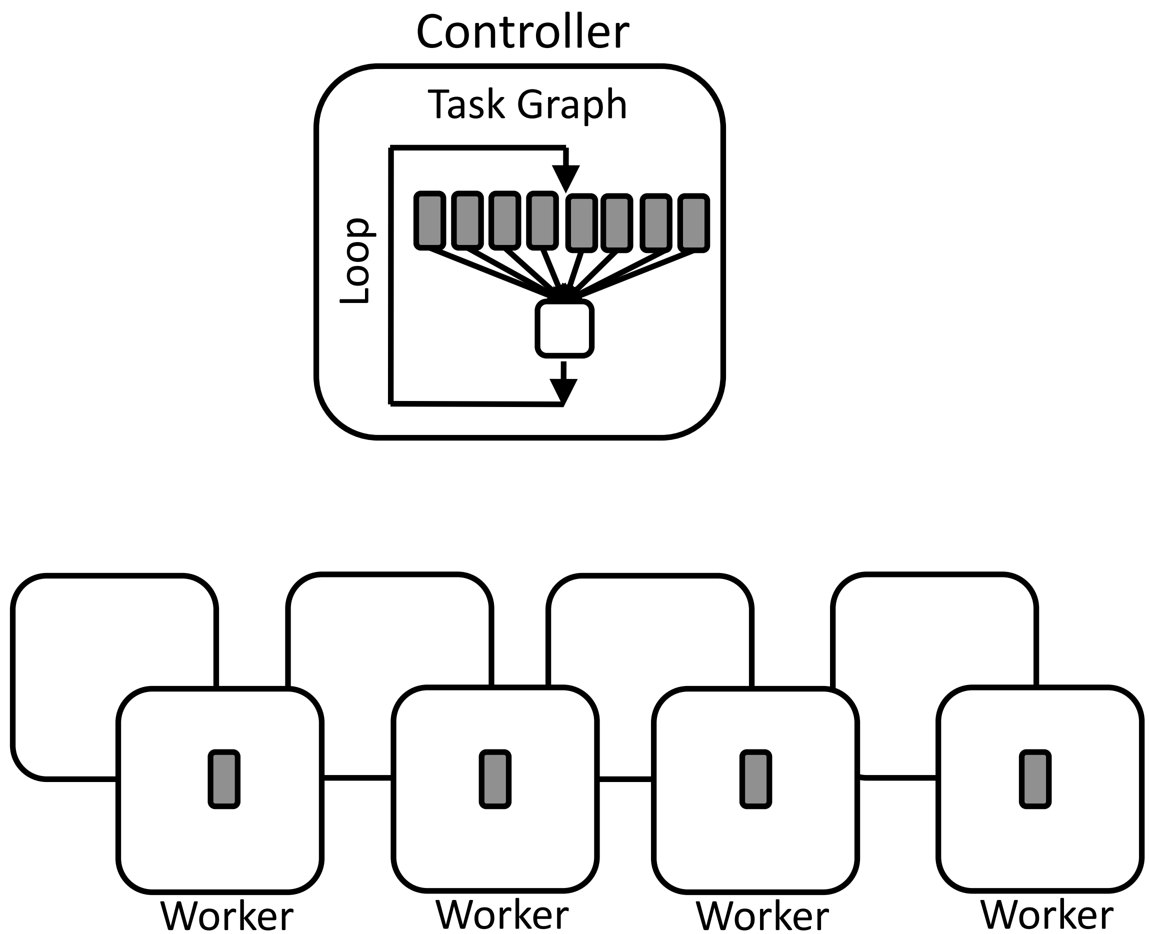


- But controller bottlenecks at scale.

Design Spectrum

Centralized Controllers

Distributed Controllers



- But controller bottlenecks at scale.

Design Spectrum

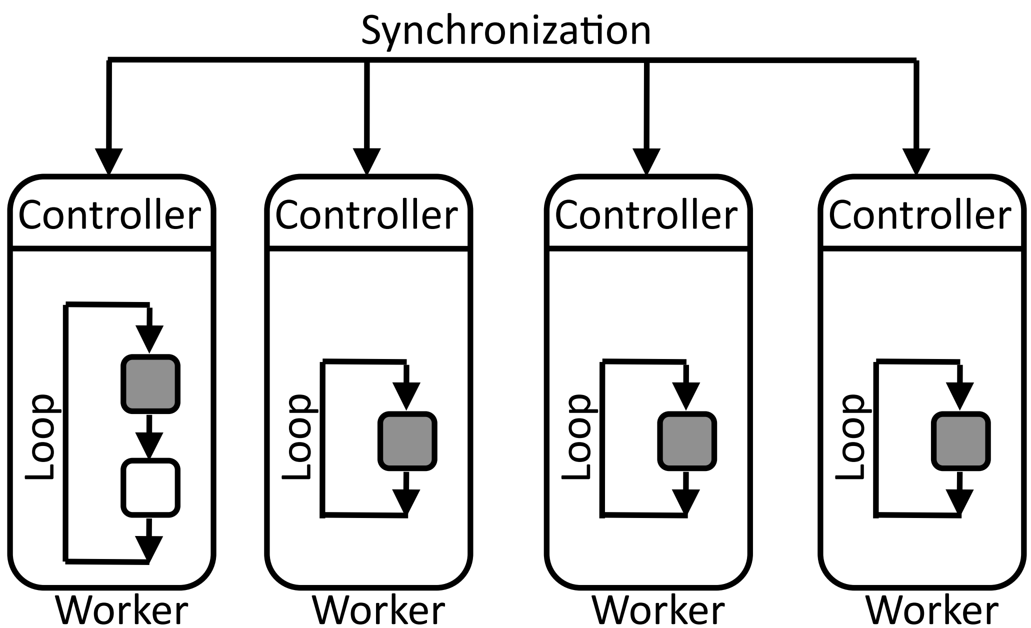
Centralized Controllers

Distributed Controllers

- Logistic regression over a data set of size 100GB in Spark 2.0 MLlib.
- Control Plane **bottlenecks at scale**, generating and spawning tasks.

Design Spectrum

Centralized Controllers — Distributed Controllers

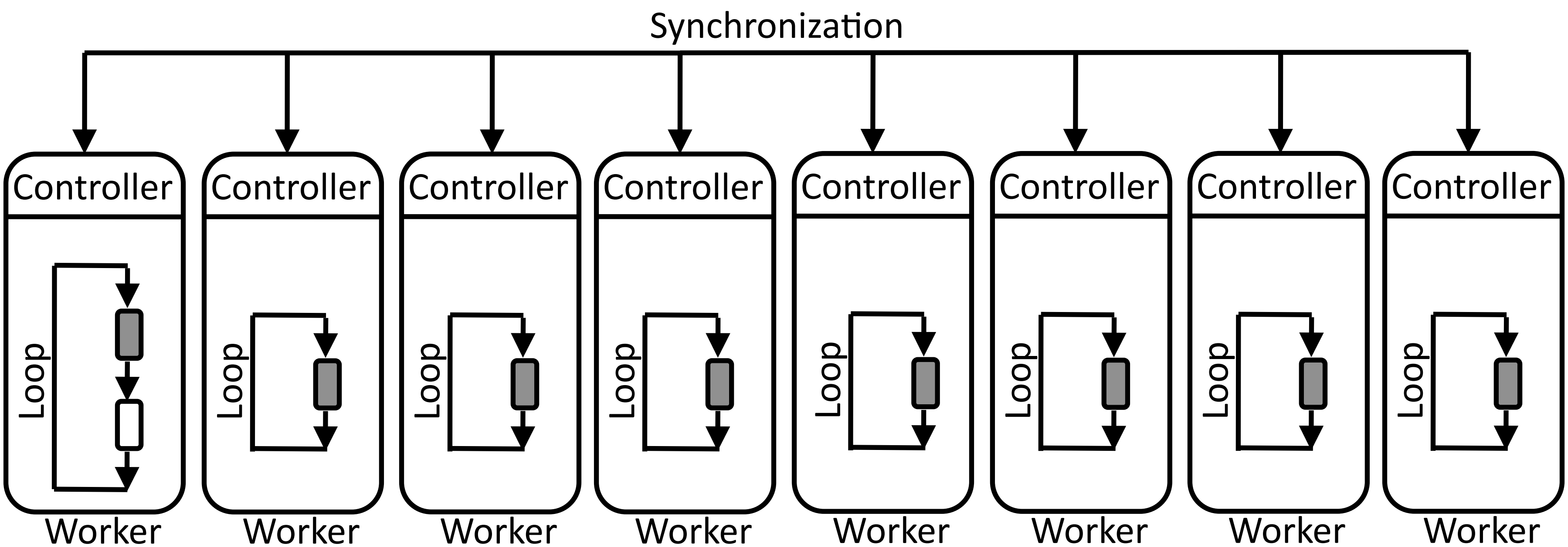


- Naiad
- TensorFlow

- Each node generates and executes tasks locally.

Design Spectrum

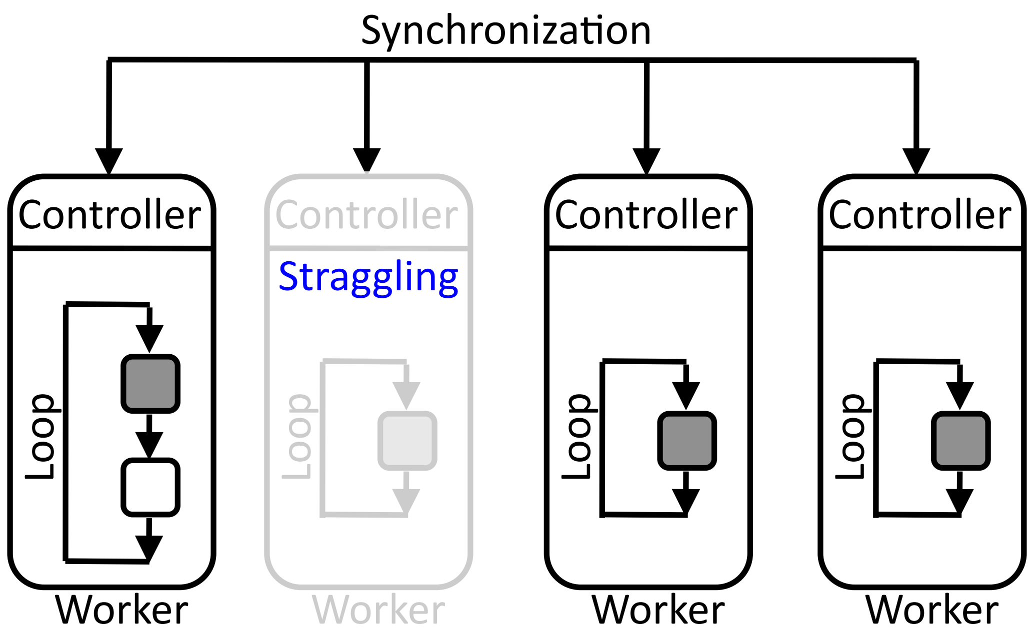
Centralized Controllers | Distributed Controllers



- The design scales well as there is no single bottleneck.

Design Spectrum

Centralized Controllers | Distributed Controllers

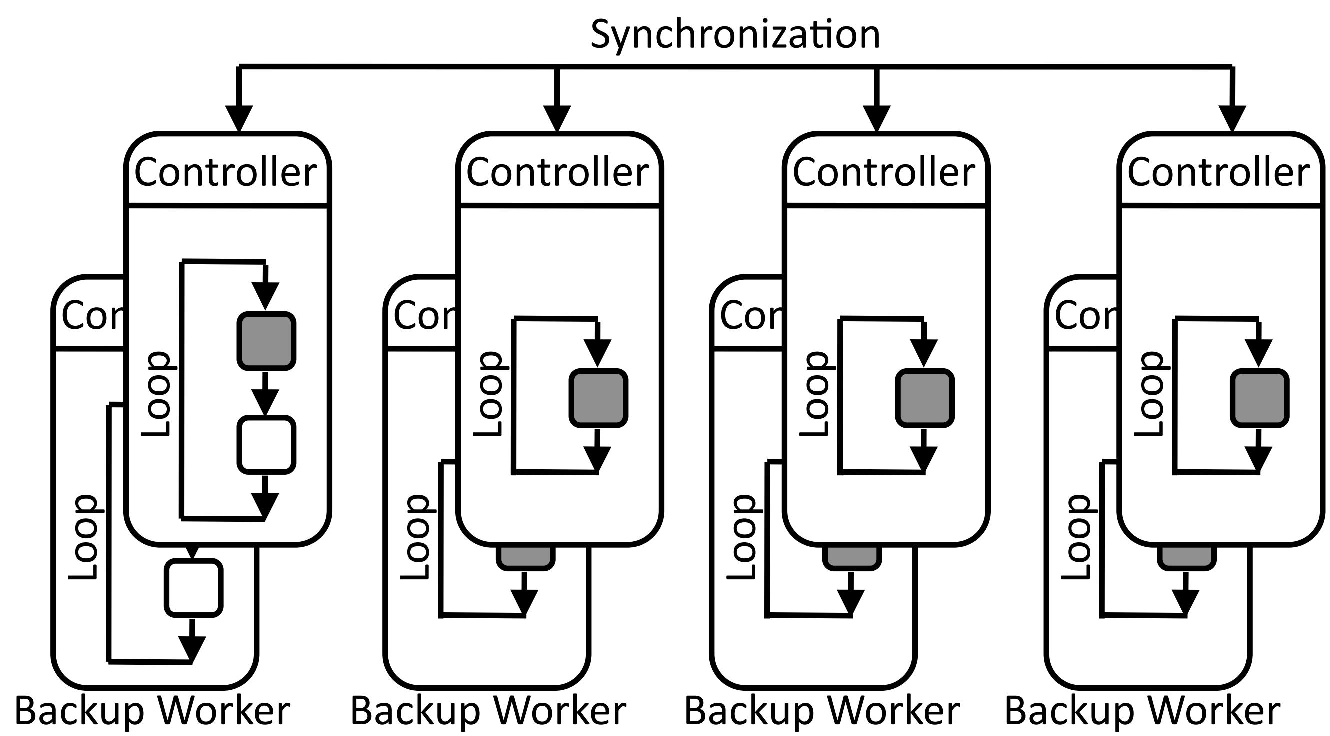


- But, the scheduling is static.
    - The progress speed is bound to the speed of the slowest node.
    - Any change requires stopping all nodes and installing new data flow.

Design Spectrum

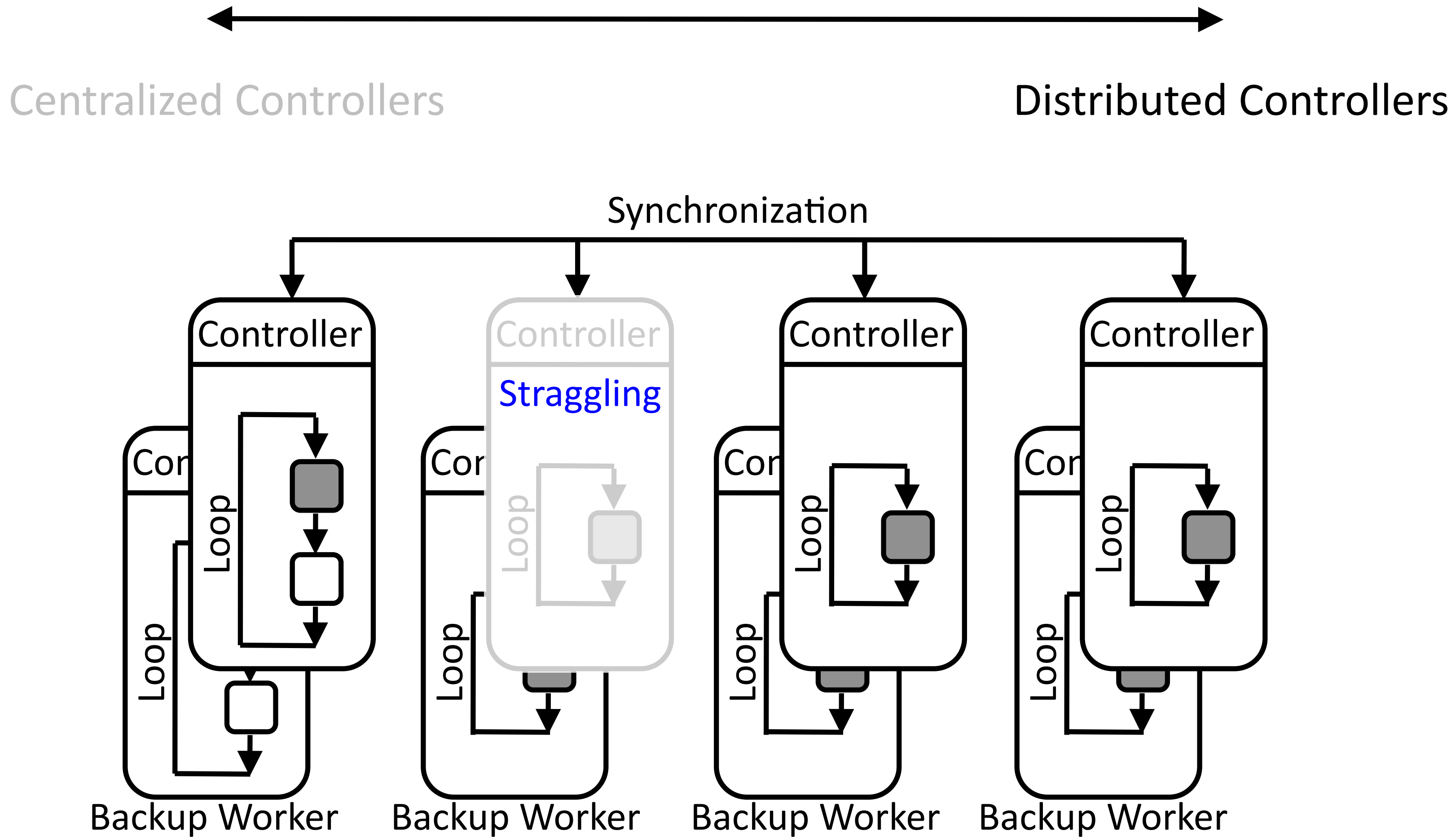
Centralized Controllers

Distributed Controllers



- In practice the straggler mitigation is only **proactive**:
  - Avoiding stragglers by meticulous engineering work.
  - Launching backup workers (at least doubling the resources).

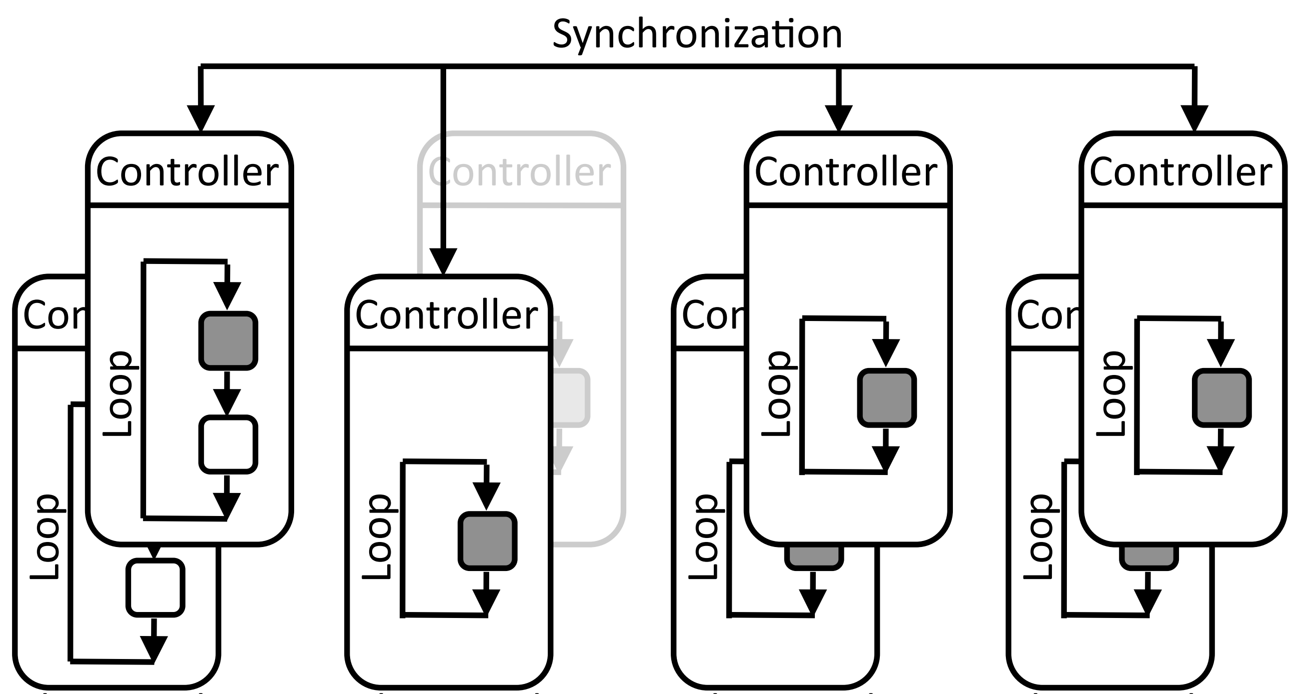
Design Spectrum



- In practice the straggler mitigation is only **proactive**:
  - Avoiding stragglers by meticulous engineering work.
  - Launching backup workers (at least doubling the resources).

Design Spectrum

Centralized Controllers — Distributed Controllers



- In practice the straggler mitigation is only **proactive**:
  - Avoiding stragglers by meticulous engineering work.
  - Launching backup workers (at least doubling the resources).

Design Space

Summary

Control Plane Design	Example Framework	Task Throughput	Task Scheduling
Centralized	MapReduce Hadoop Spark	Low	Dynamic
Distributed	Naiad TensorFlow	High	Static

Design Space

Summary

Control Plane Design	Example Framework	Task Throughput	Task Scheduling
Centralized	MapReduce Hadoop Spark	Low	Dynamic
Distributed	Naiad TensorFlow	High	Static

We would like to have the best of both worlds:

- High task throughput for fast computations.
- Dynamic, fine-grained scheduling decisions.

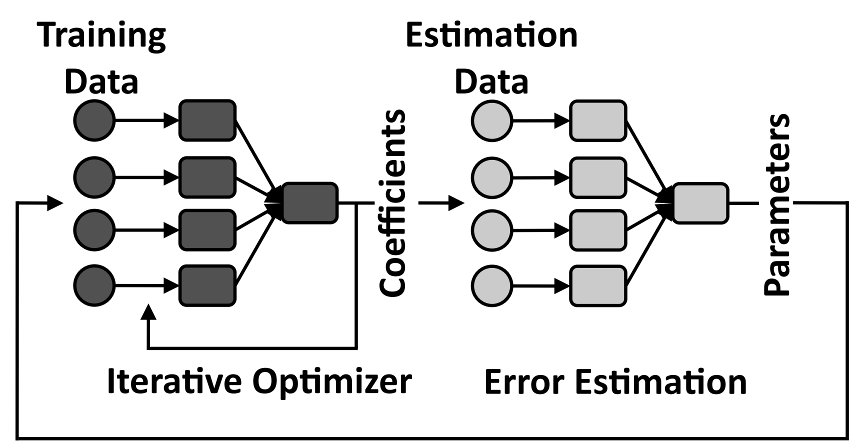
Repetitive Patterns

- Advanced data analytics are iterative in nature.
  - Machine learning, graph processing, image recognition, etc.
- This results in repetitive patterns in the control plane.
  - Similar tasks execute with minor differences.

Repetitive Patterns

- Advanced data analytics are iterative in nature.
  - Machine learning, graph processing, image recognition, etc.
- This results in repetitive patterns in the control plane.
  - Similar tasks execute with minor differences.

```
while (error > threshold_e) {
   while (gradient > threshold_g) {
      // Optimization code block
      gradient = Gradient(tdata, coeff, param)
      coeff += gradient
   }
   // Estimation code block
   error = Estimate(edata, coeff, param)
   param = update_model(param, error)
}
```

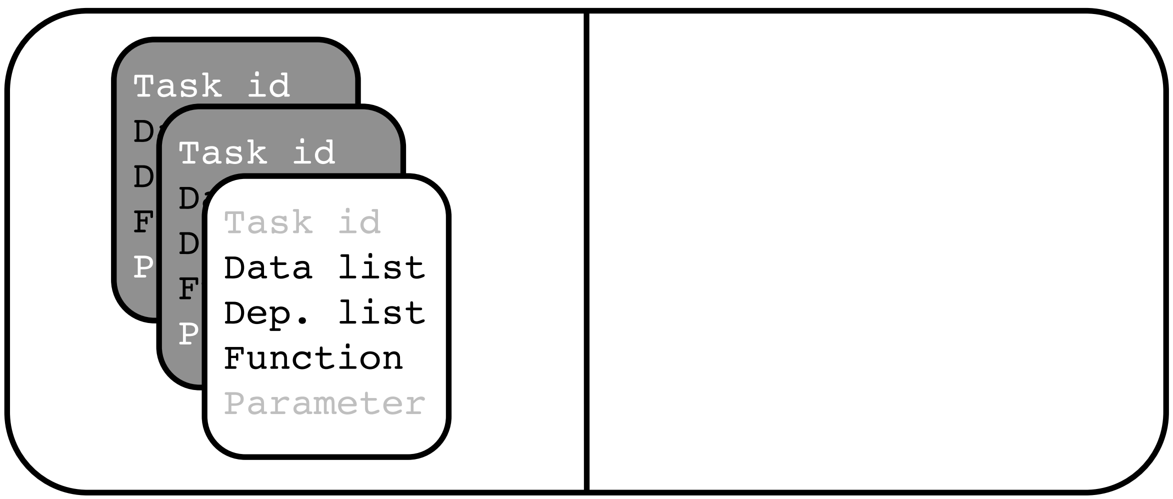


This talk

- Control Plane: the Emerging Bottleneck
- Design Scope of the Control Plane
- Execution Templates
- Nimbus: a Framework with Templates
- Evaluation

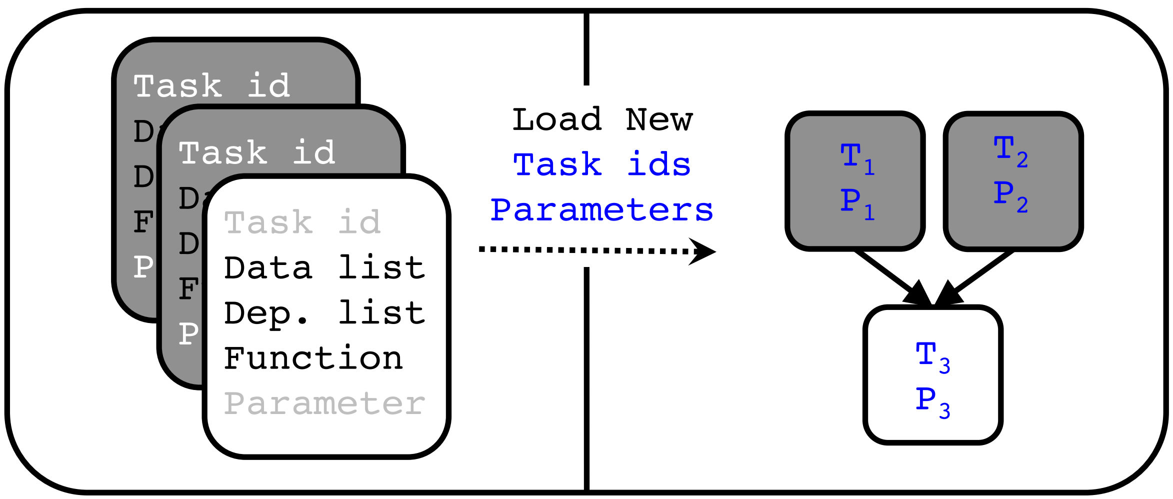
Execution Templates

- Tasks are cached as **parameterizable blocks** on nodes.
- Instead of assigning the tasks from scratch, templates are **instantiated** by filling in only changing parameters.



Execution Templates

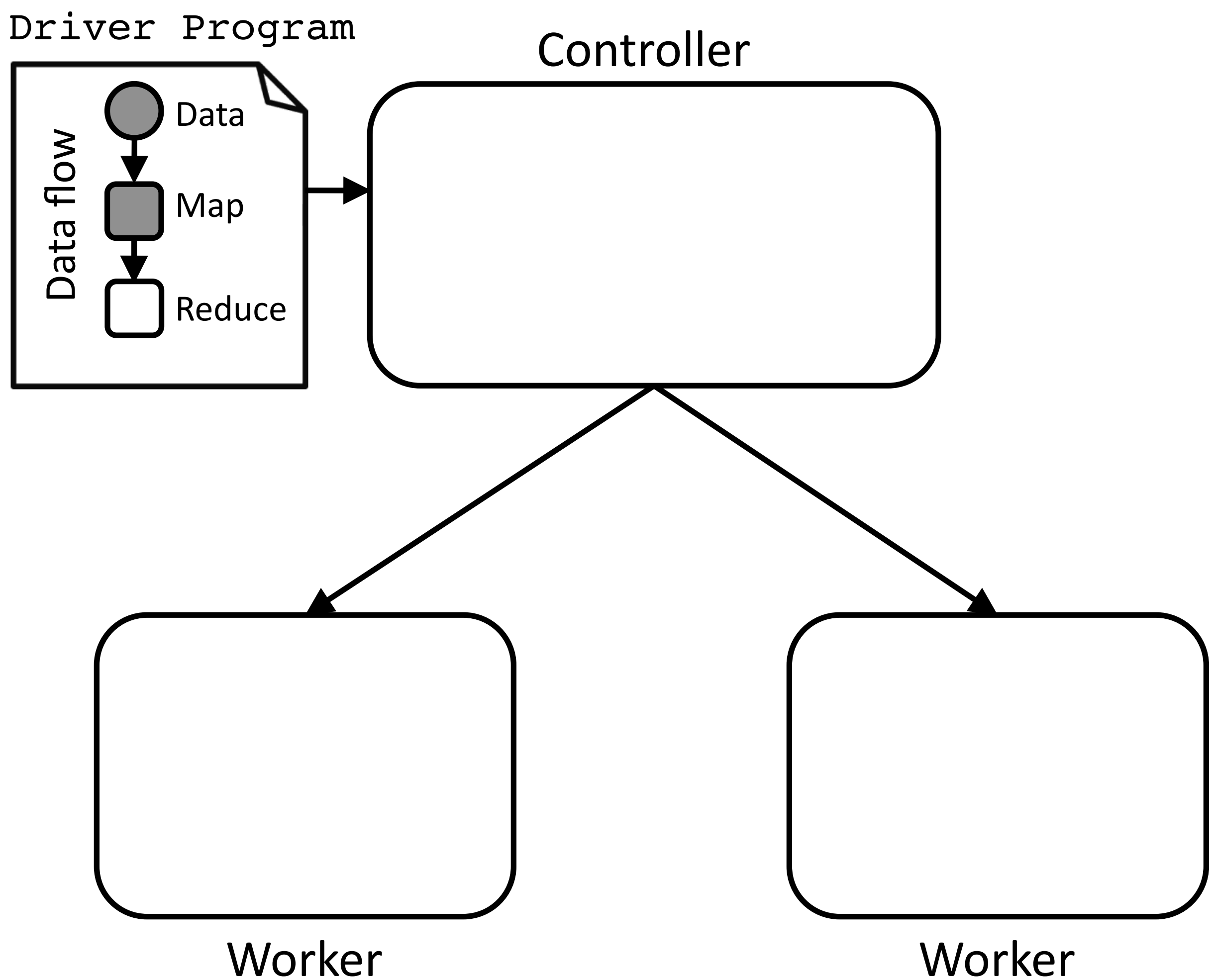
- Tasks are cached as **parameterizable blocks** on nodes.
- Instead of assigning the tasks from scratch, templates are **instantiated** by filling in only changing parameters.



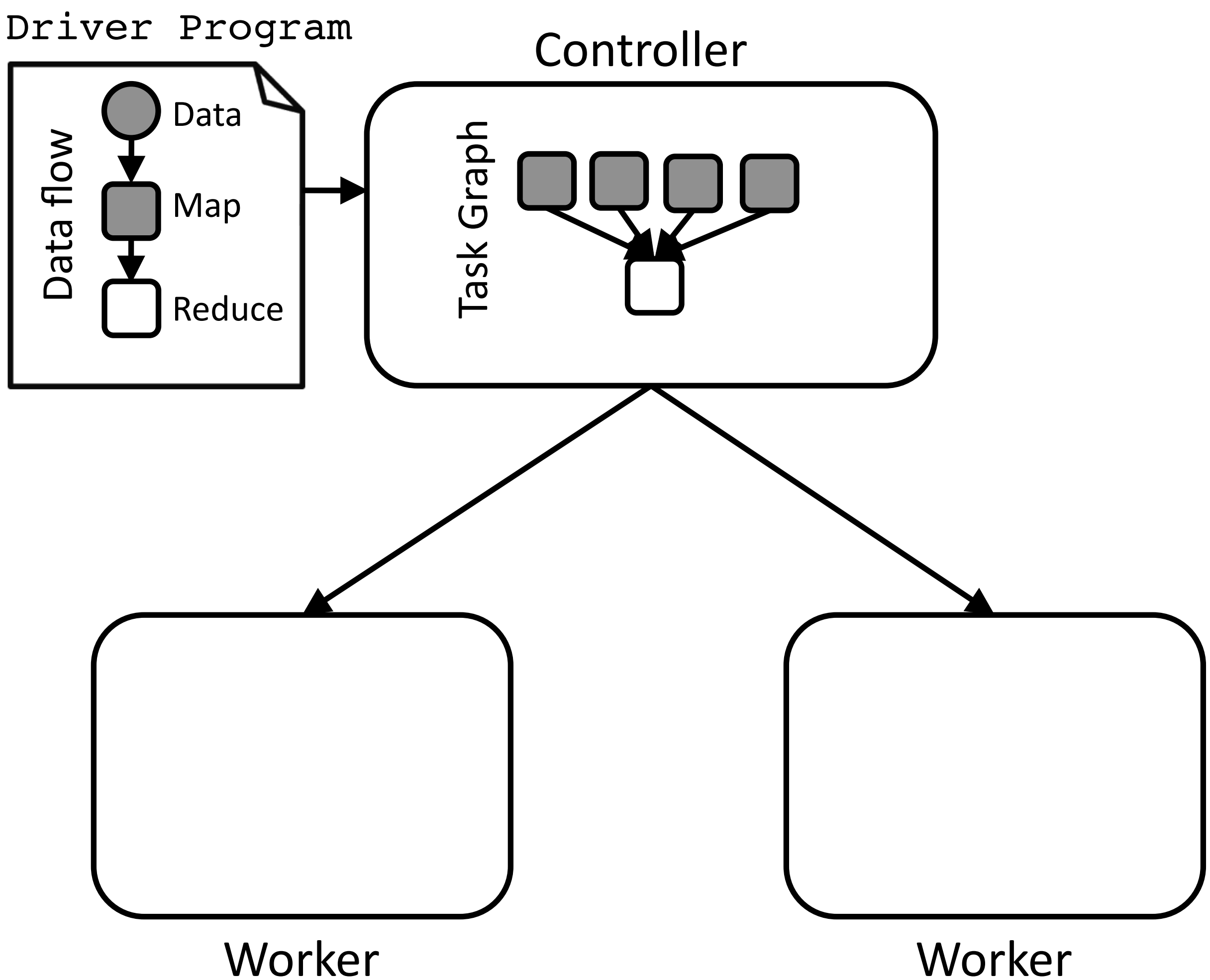
Execution Templates
Mechanisms Summary

- **Instantiation**: spawn a block of tasks without processing each task individually from scratch. It helps increase the **task throughput**.
- **Edits**: modifies the content of each template at the granularity of tasks. It enables fine-grained, **dynamic scheduling**.
- **Patches**: In case the state of the worker does not match the preconditions of the template. It enables **dynamic control flow**.

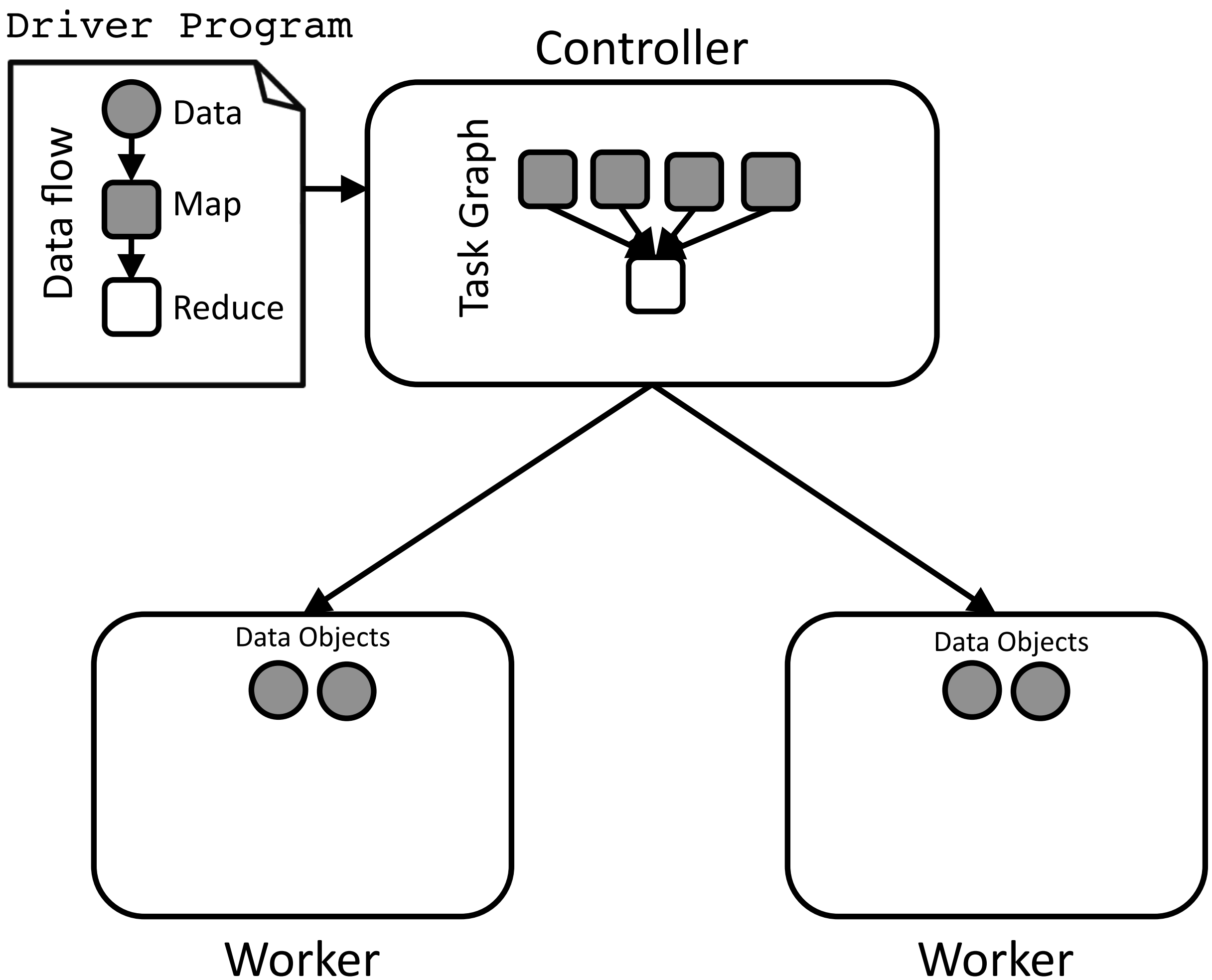
Execution Model



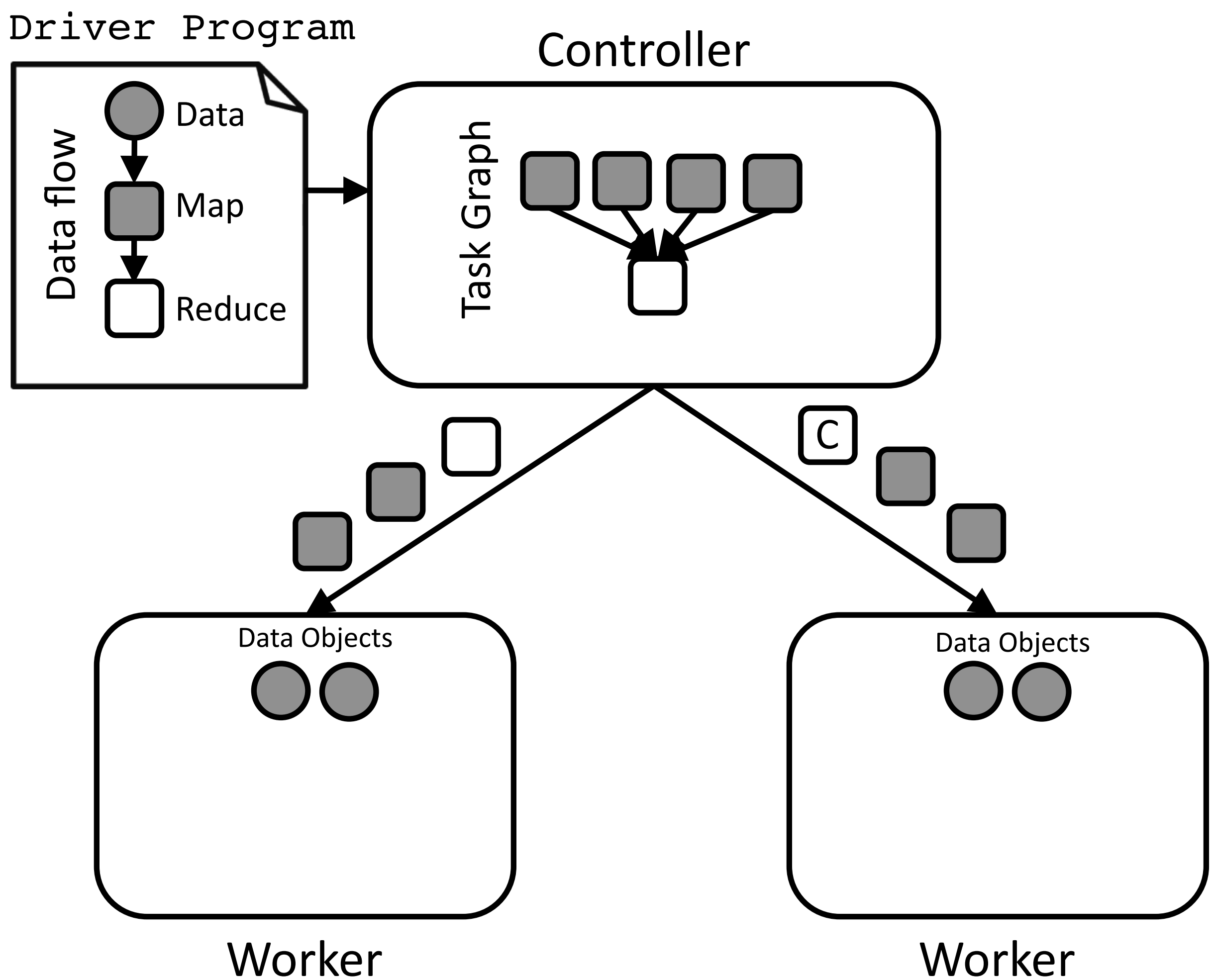
Execution Model



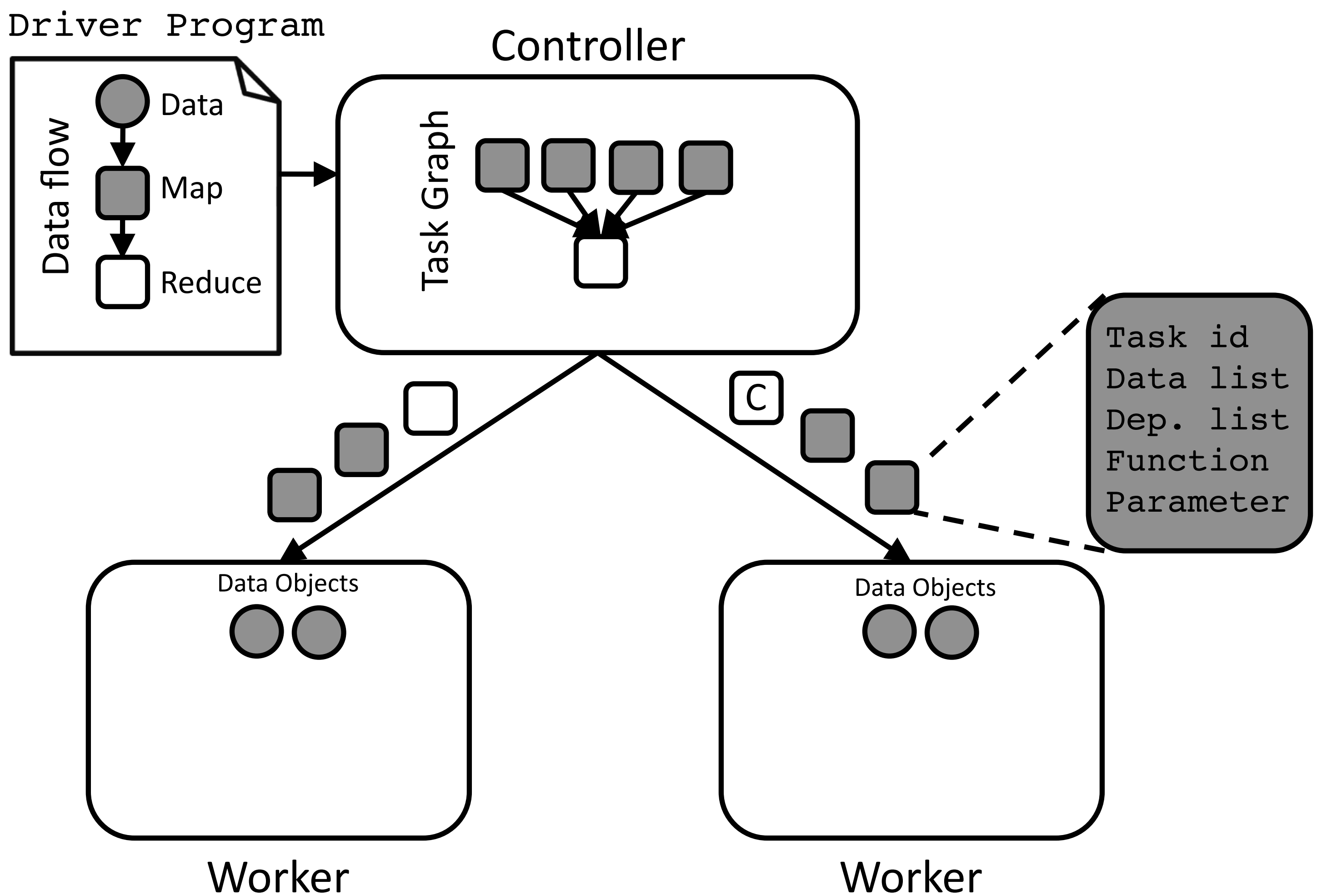
Execution Model



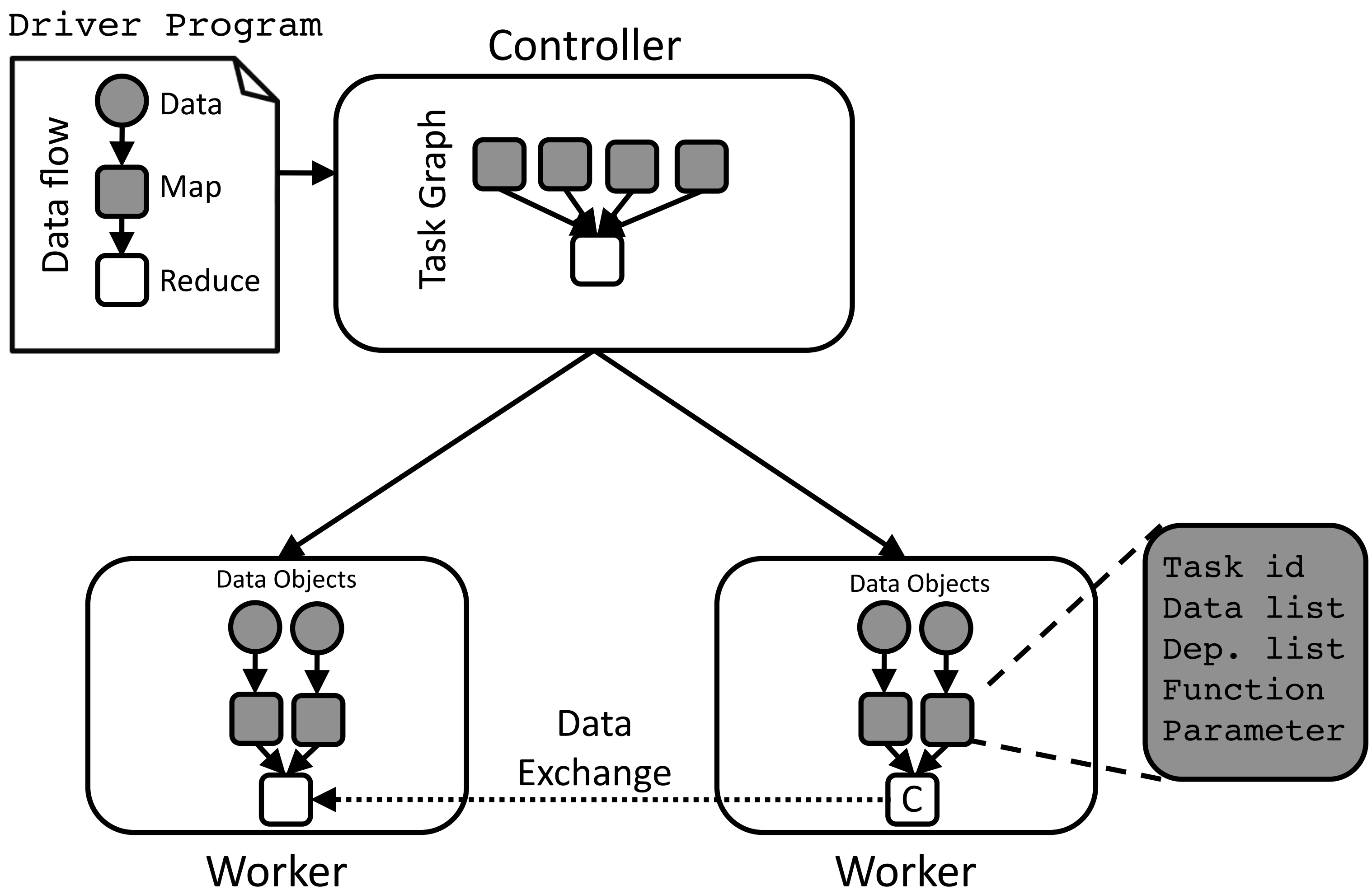
Execution Model



Execution Model



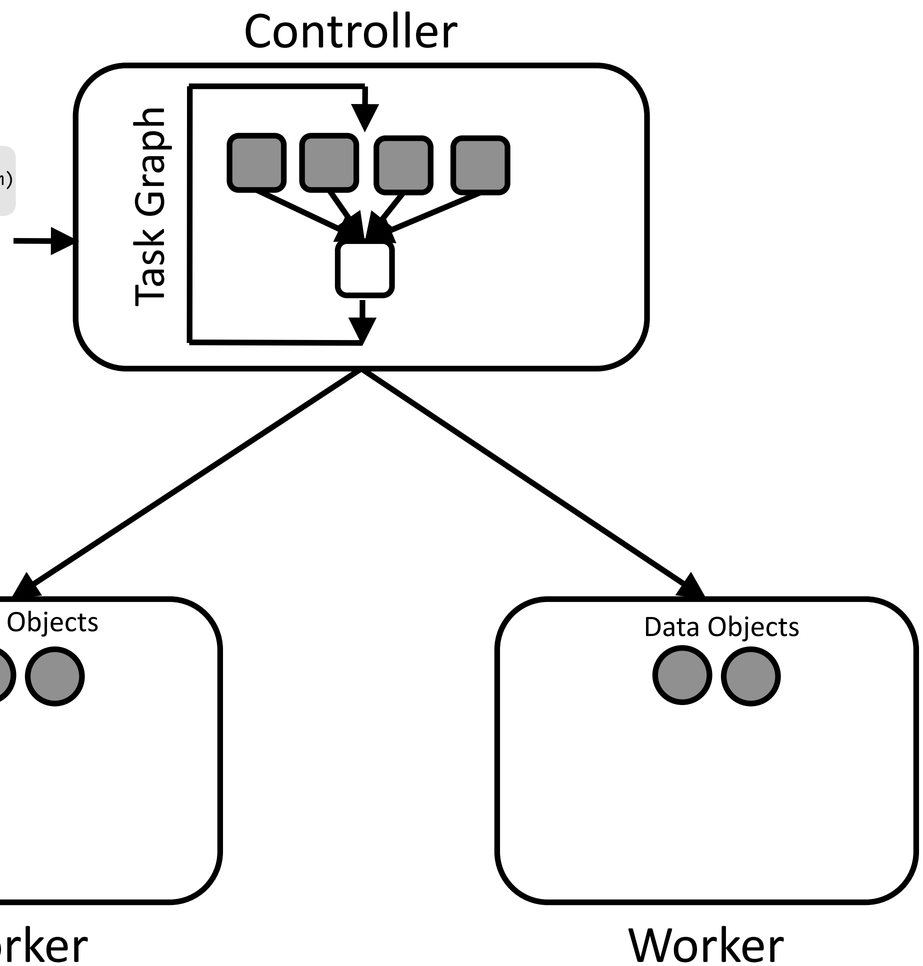
Execution Model



Repetitive Patterns

Driver Program

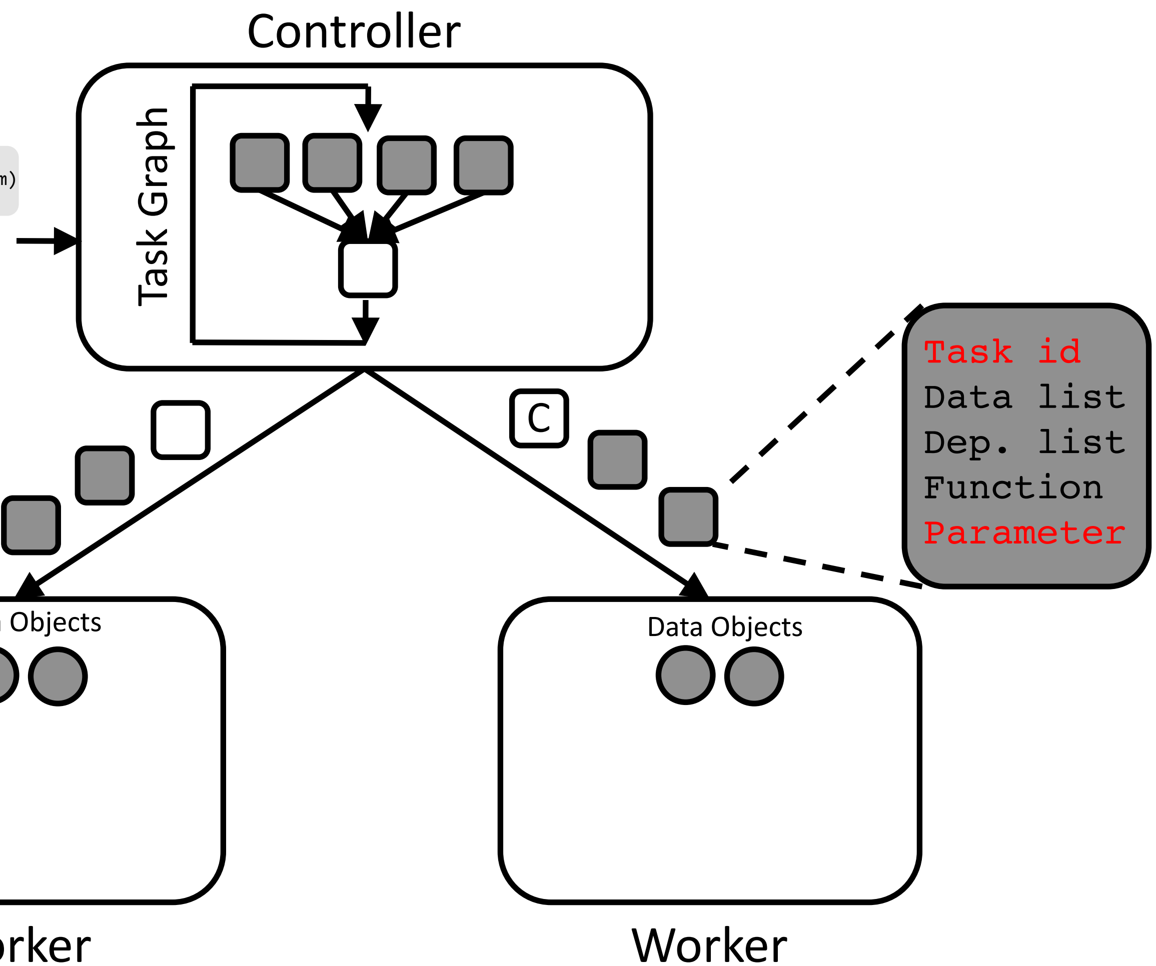
```
while (error > threshold_e) {
   while (gradient > threshold_g) {
      // Optimization code block
      gradient = Gradient(tdata, coeff, param)
      coeff += gradient
   }
   // Estimation code block
   error = Estimate(edata, coeff, param)
   param = update_model(param, error)
}
```



Repetitive Patterns

Driver Program

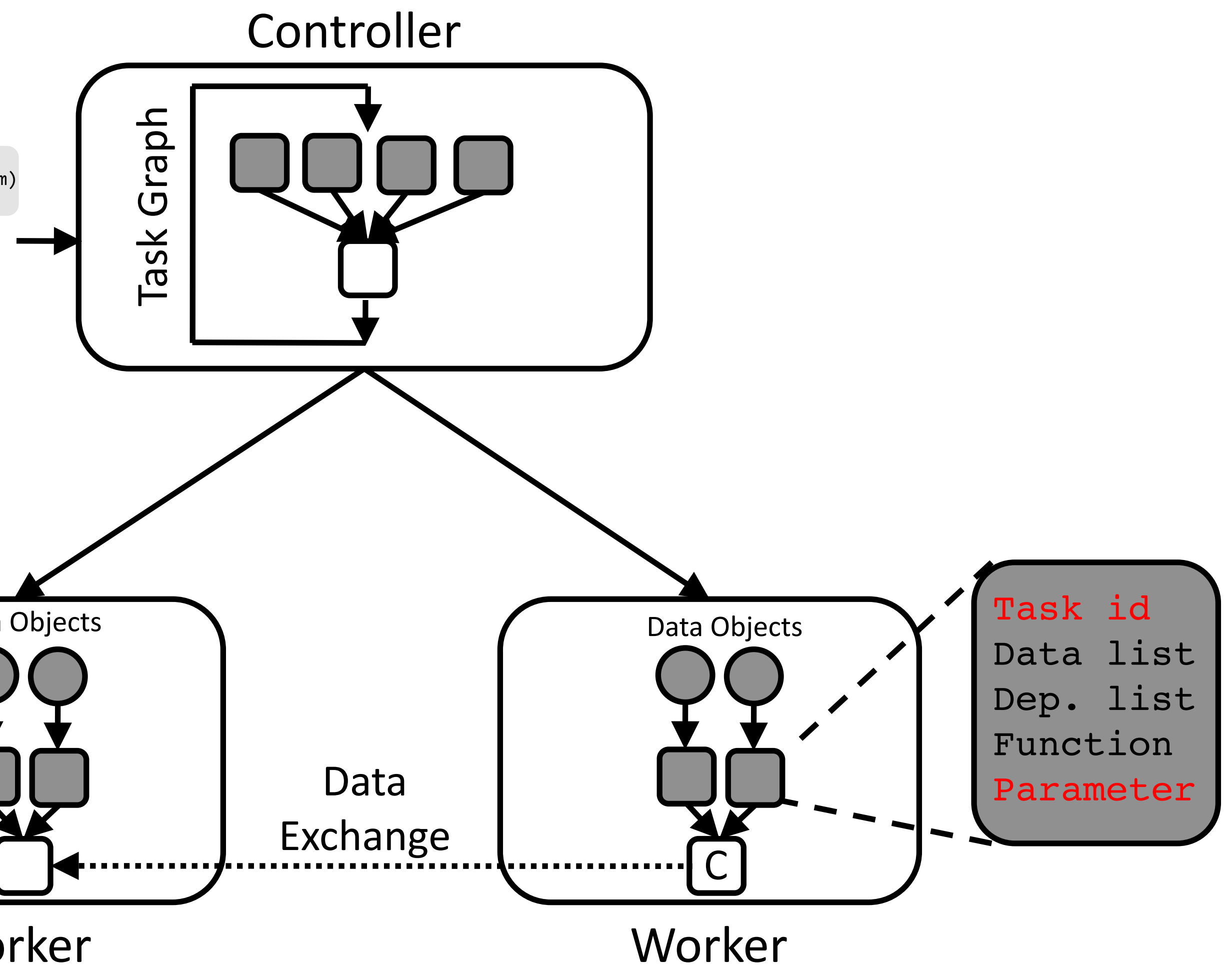
```
while (error > threshold_e) {
   while (gradient > threshold_g) {
      // Optimization code block
      gradient = Gradient(tdata, coeff, param)
      coeff += gradient
   }
   // Estimation code block
   error = Estimate(edata, coeff, param)
   param = update_model(param, error)
}
```



Repetitive Patterns

Driver Program

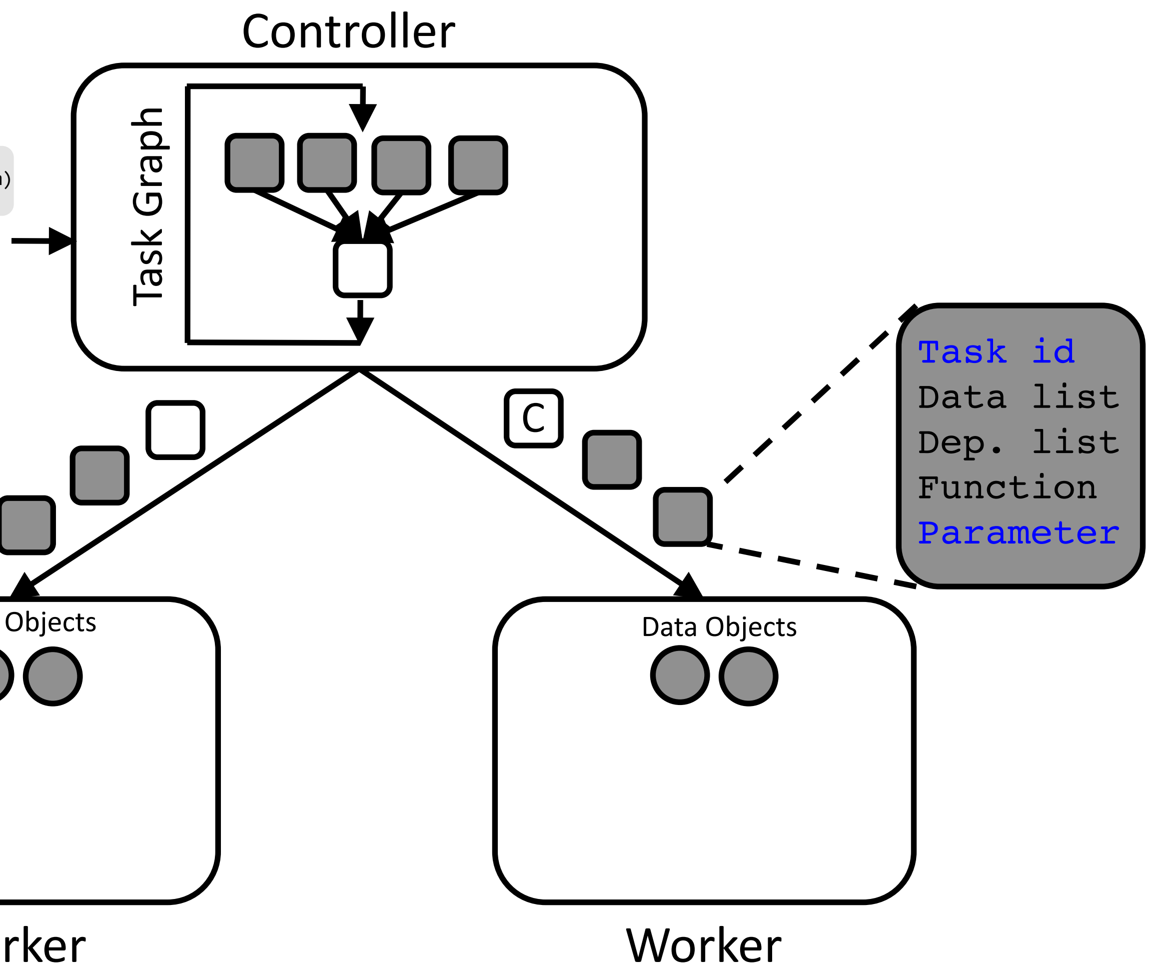
```
while (error > threshold_e) {
   while (gradient > threshold_g) {
      // Optimization code block
      gradient = Gradient(tdata, coeff, param)
      coeff += gradient
   }
   // Estimation code block
   error = Estimate(edata, coeff, param)
   param = update_model(param, error)
}
```



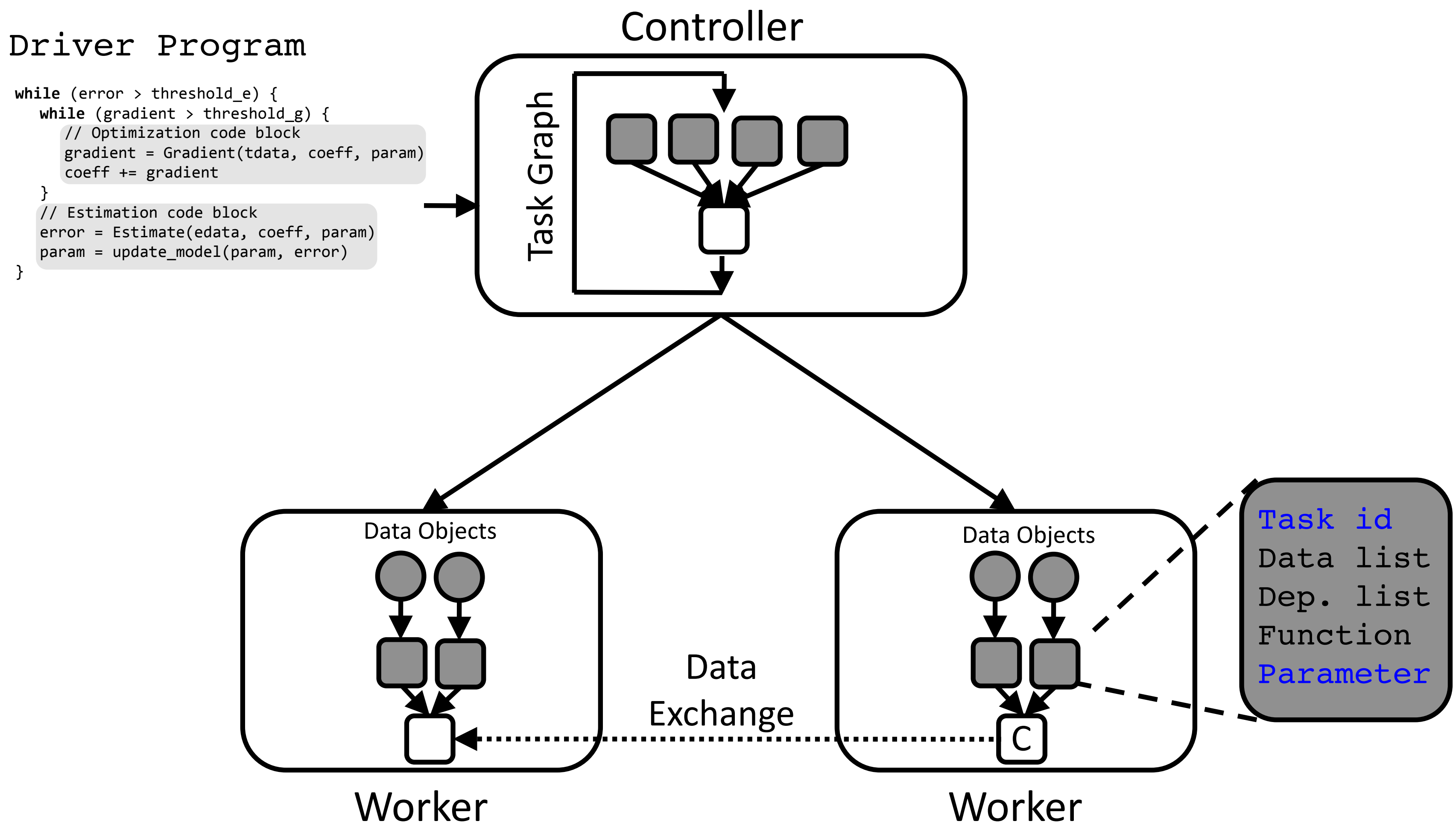
Repetitive Patterns

Driver Program

```
while (error > threshold_e) {
  while (gradient > threshold_g) {
    // Optimization code block
    gradient = Gradient(tdata, coeff, param)
    coeff += gradient
  }
  // Estimation code block
  error = Estimate(edata, coeff, param)
  param = update_model(param, error)
}
```

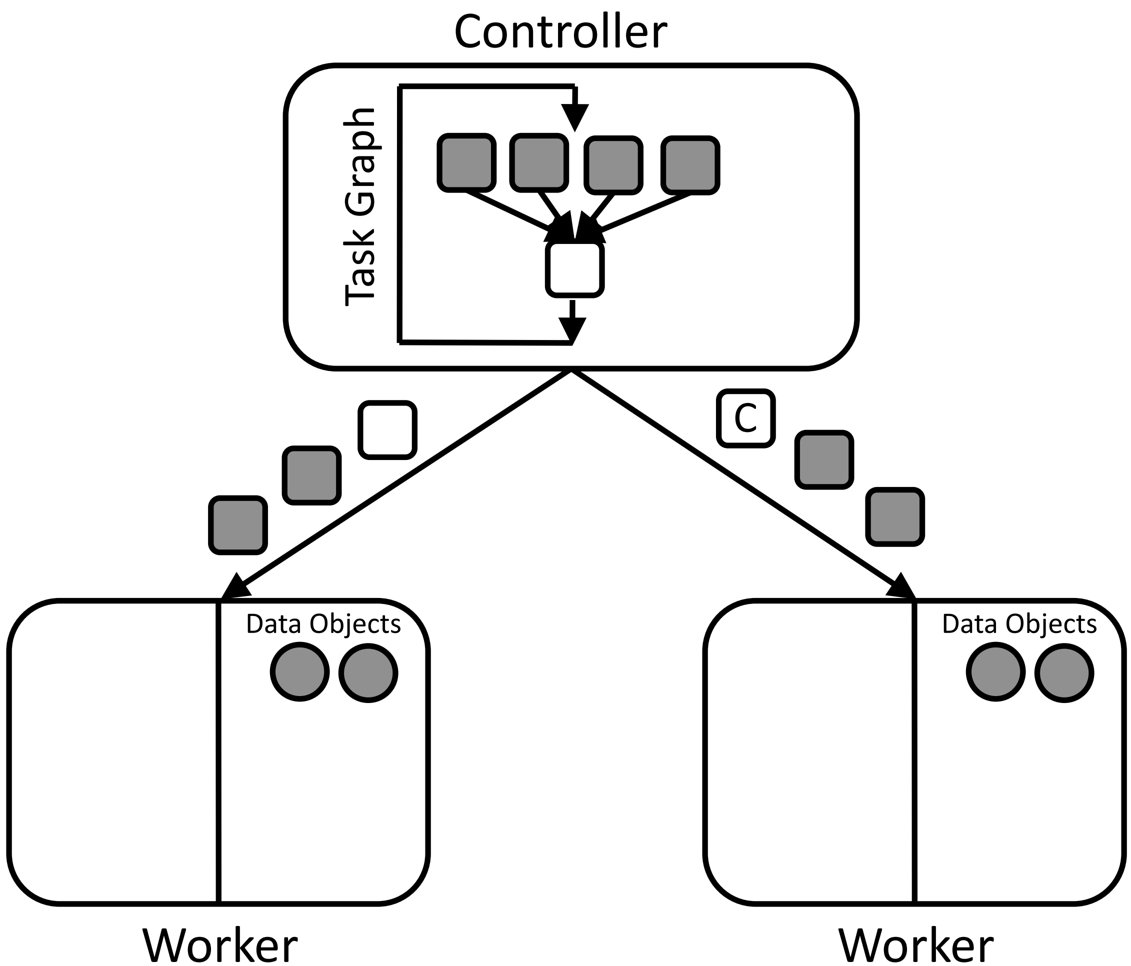


Repetitive Patterns



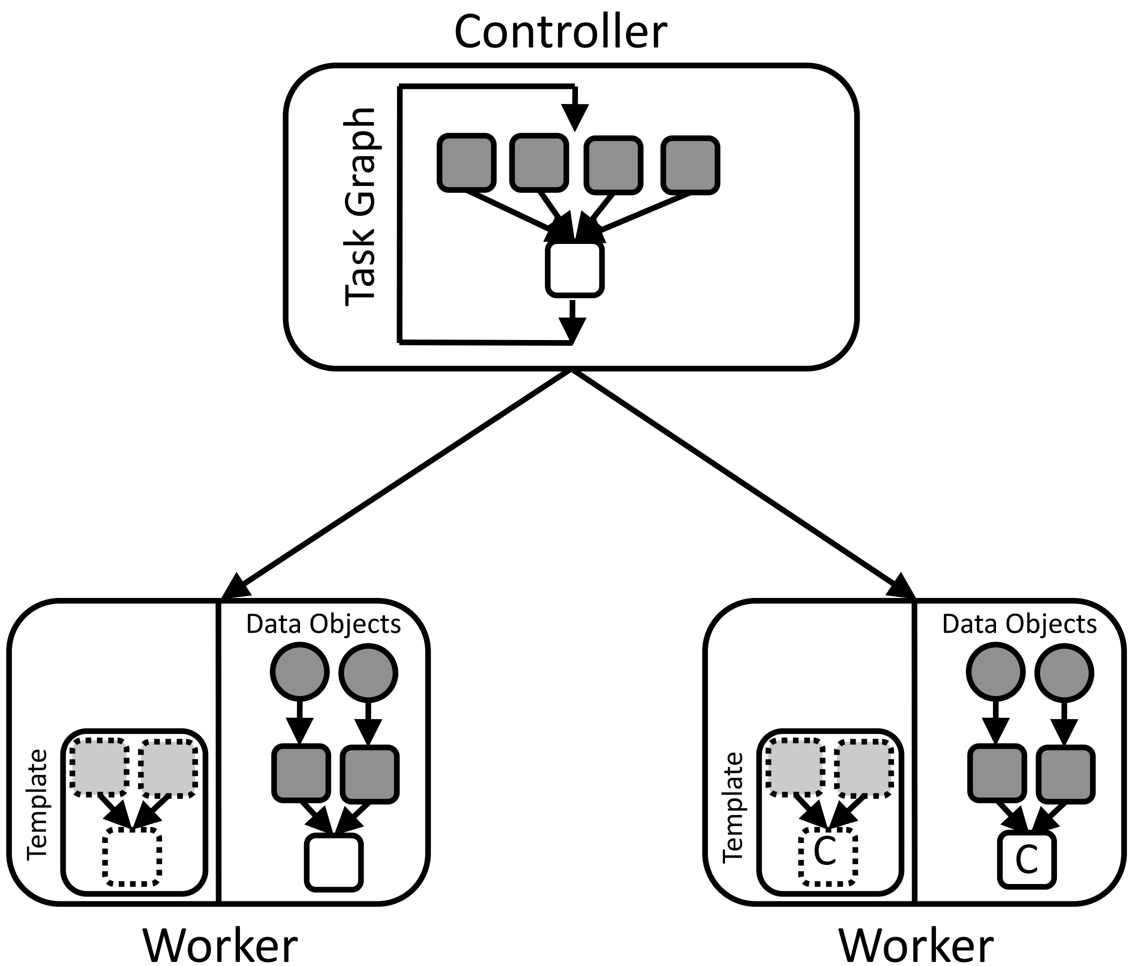
Execution Templates

Abstraction



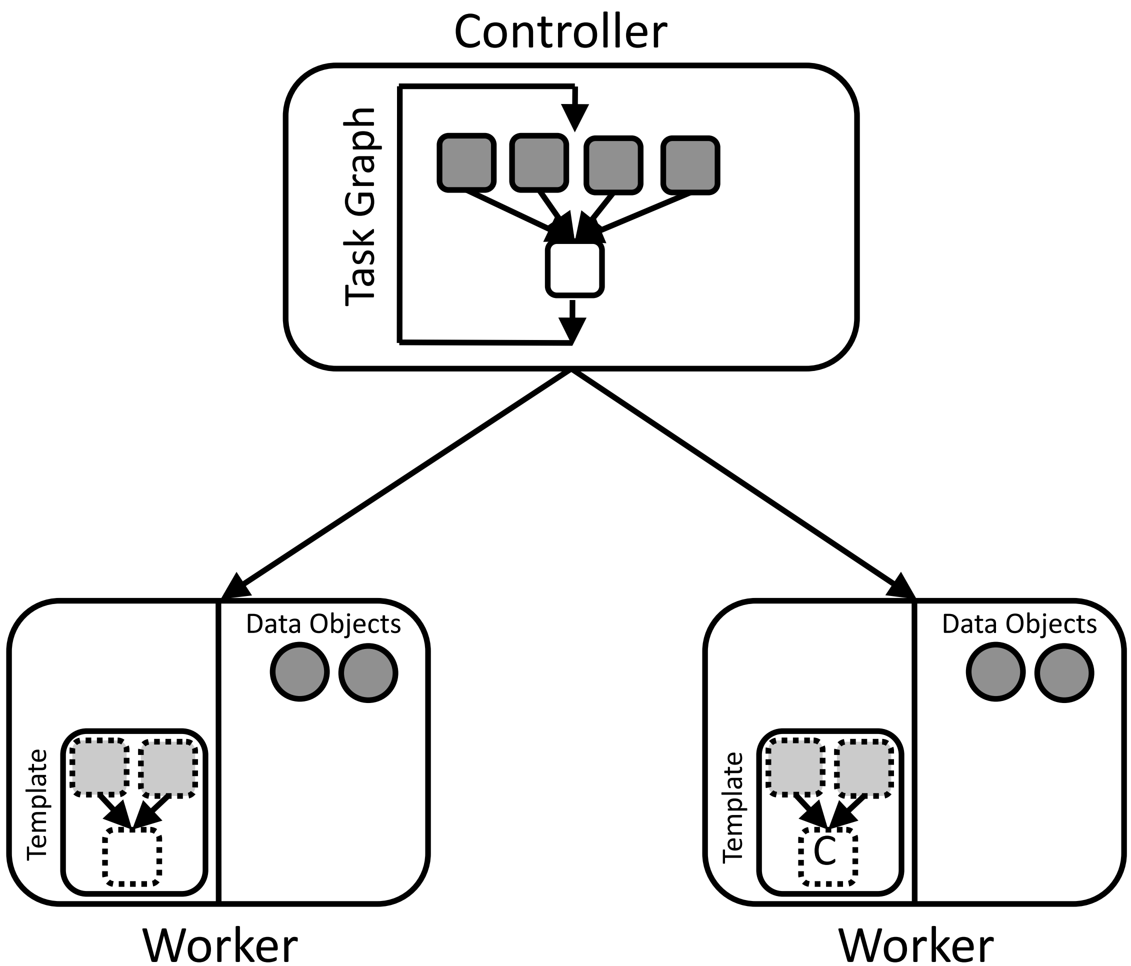
Execution Templates

Abstraction



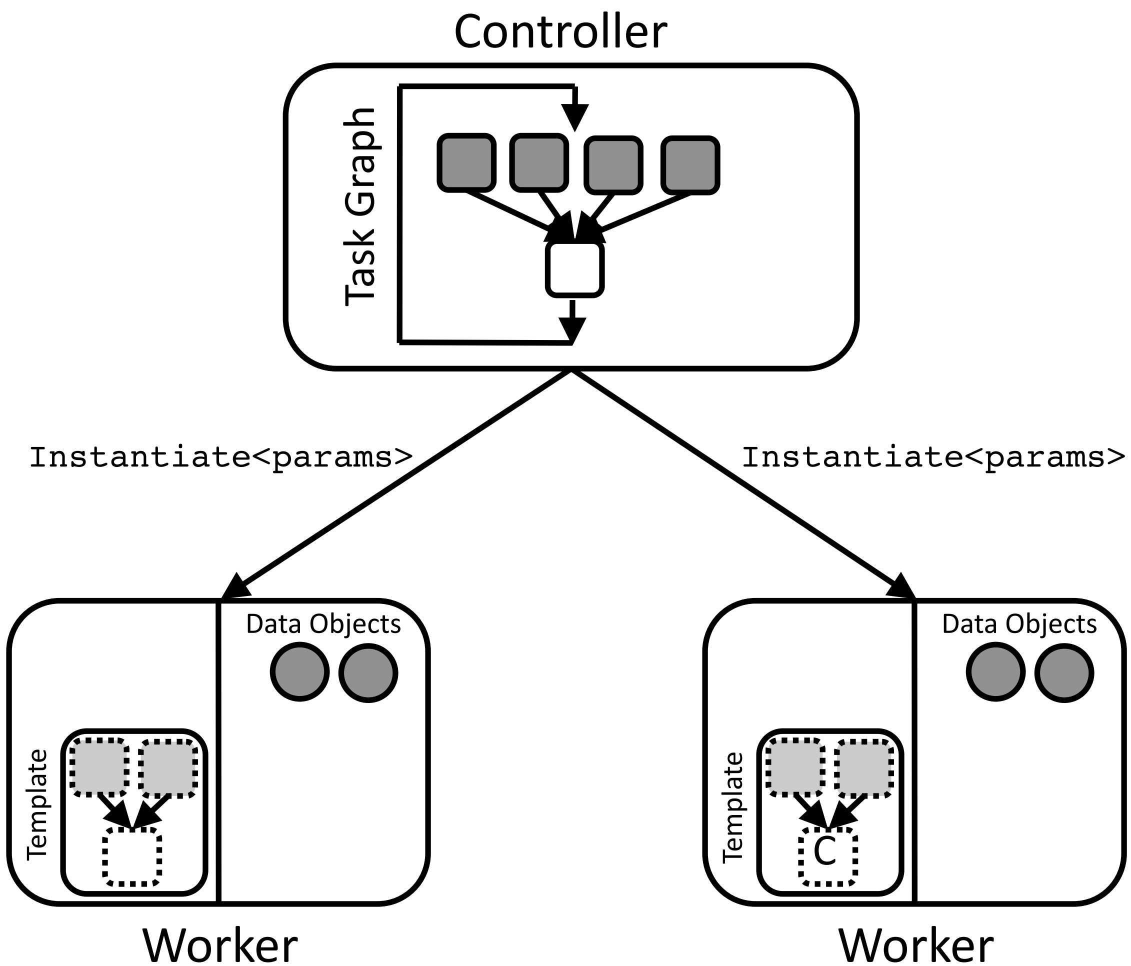
Execution Templates

Abstraction



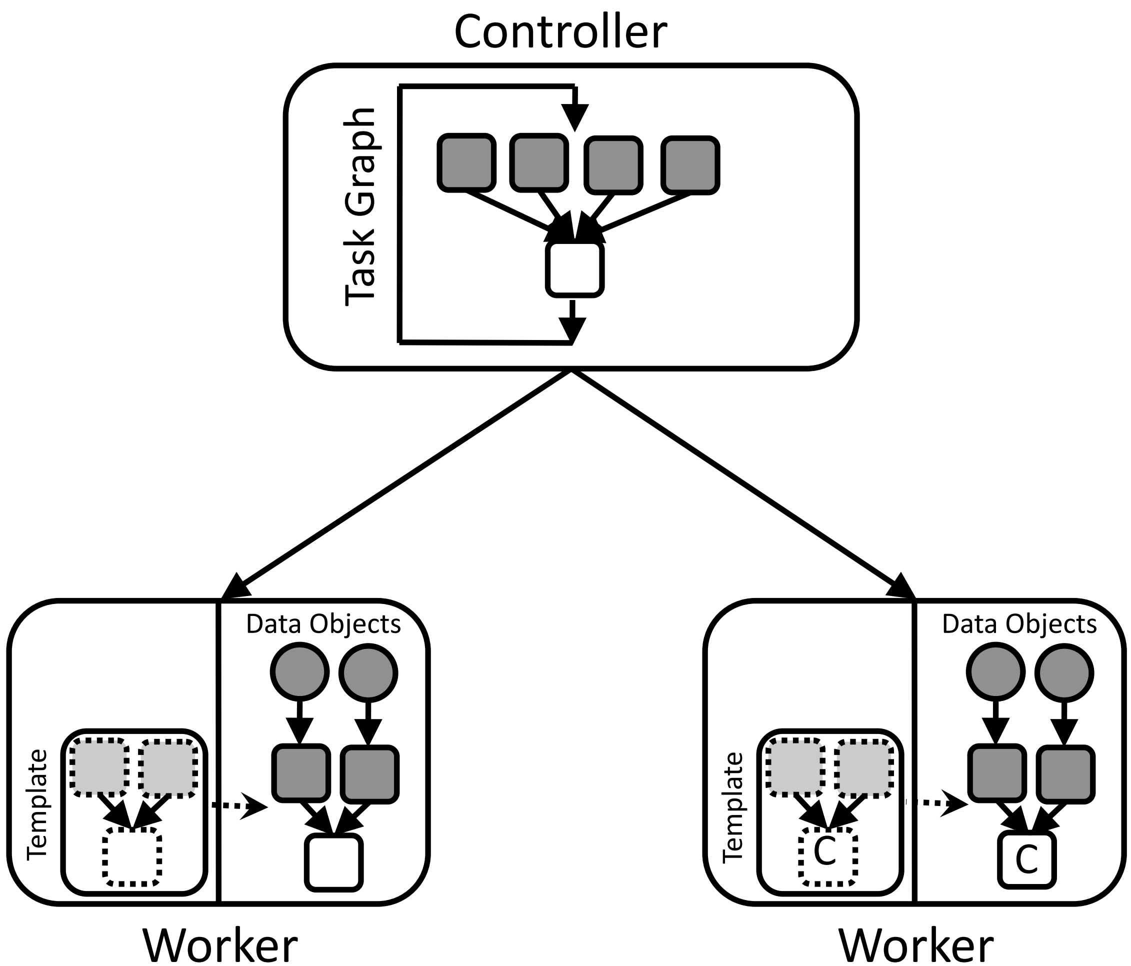
Execution Templates

Abstraction



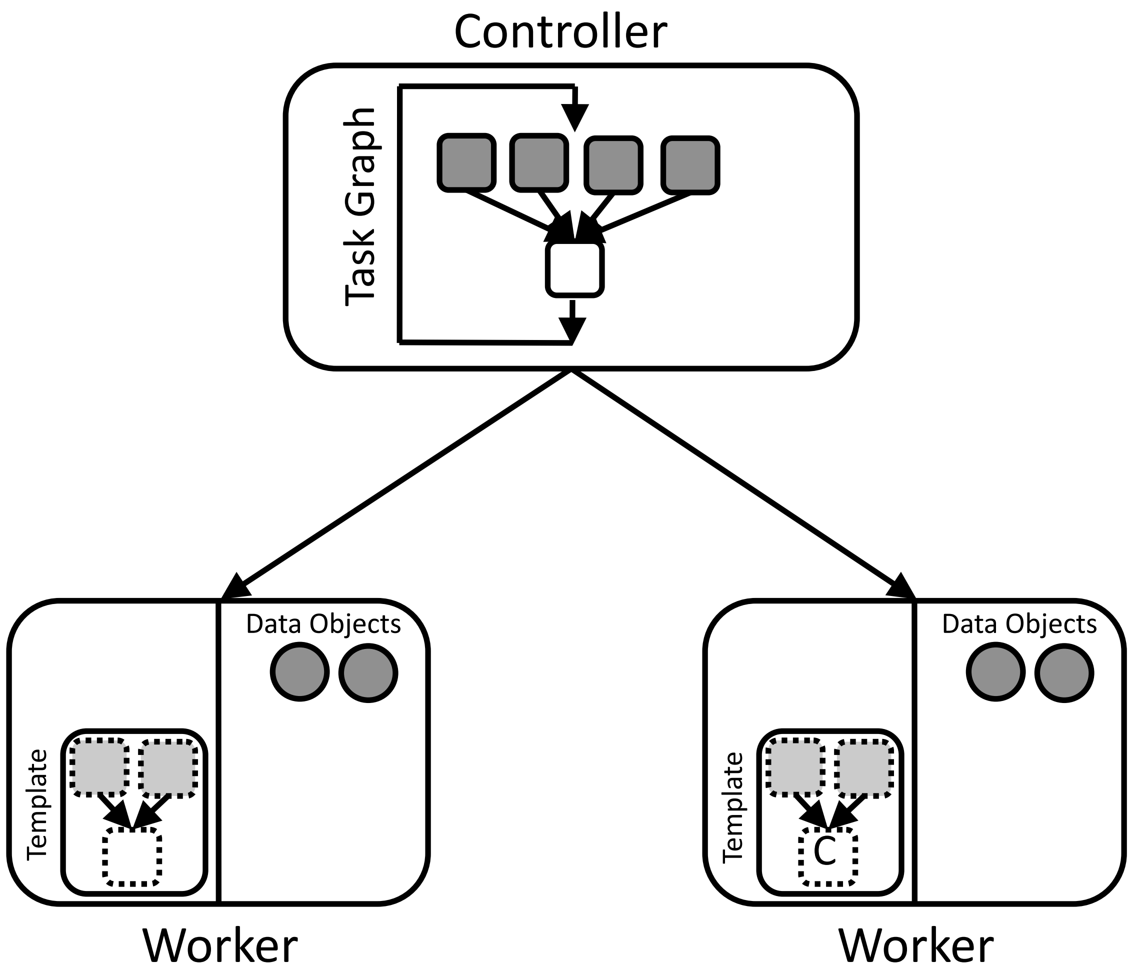
Execution Templates

Abstraction



Execution Templates

Abstraction



Execution Templates

The Devil is in the details.

- Caching tasks implies static behavior:
  - Templates and **dynamic scheduling**?
    - Reactive scheduling changes for load balancing.
    - Scheduling changes at the task granularity.
  - Templates and **dynamic control flow**?
    - Need to support nested loops.
    - Need to support data dependent branches.

Execution Templates

The Devil is in the details.

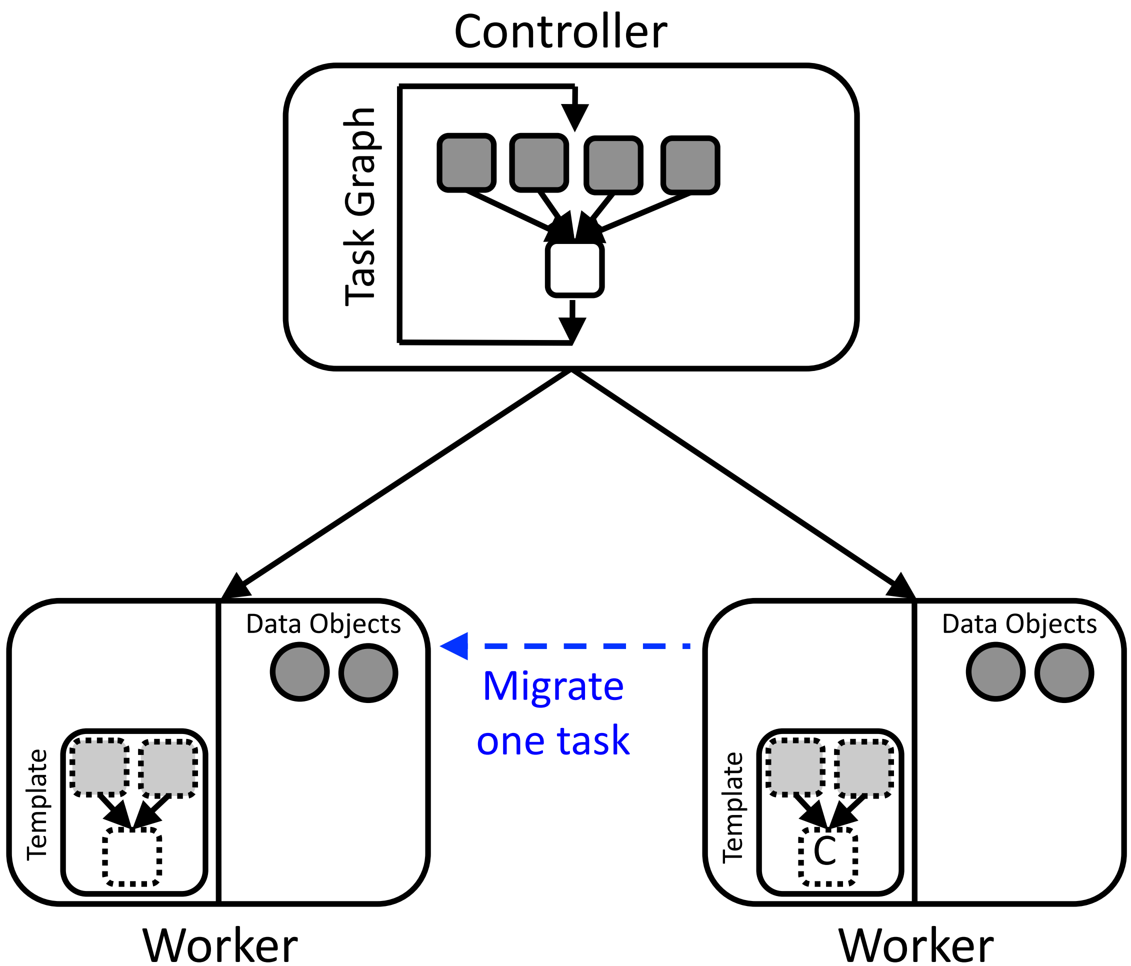
- Caching tasks implies static behavior:
  - Templates and **dynamic scheduling**?
    - Reactive scheduling changes for load balancing.
    - Scheduling changes at the task granularity.
  - Templates and **dynamic control flow**?
    - Need to support nested loops.
    - Need to support data dependent branches.

Execution Templates
Edits

- If scheduling changes, even slightly, the templates are obsolete.
  - For example migrating tasks among workers.
- Instead of paying the substantial cost of installing templates for every changes, templates allow **edit**, to change their structure.
- **Edits** enable adding or removing tasks from the template and modifying the template content, in-place.
- Controller has the general view of the task graph so it can update the dependencies properly, needed by the edits.

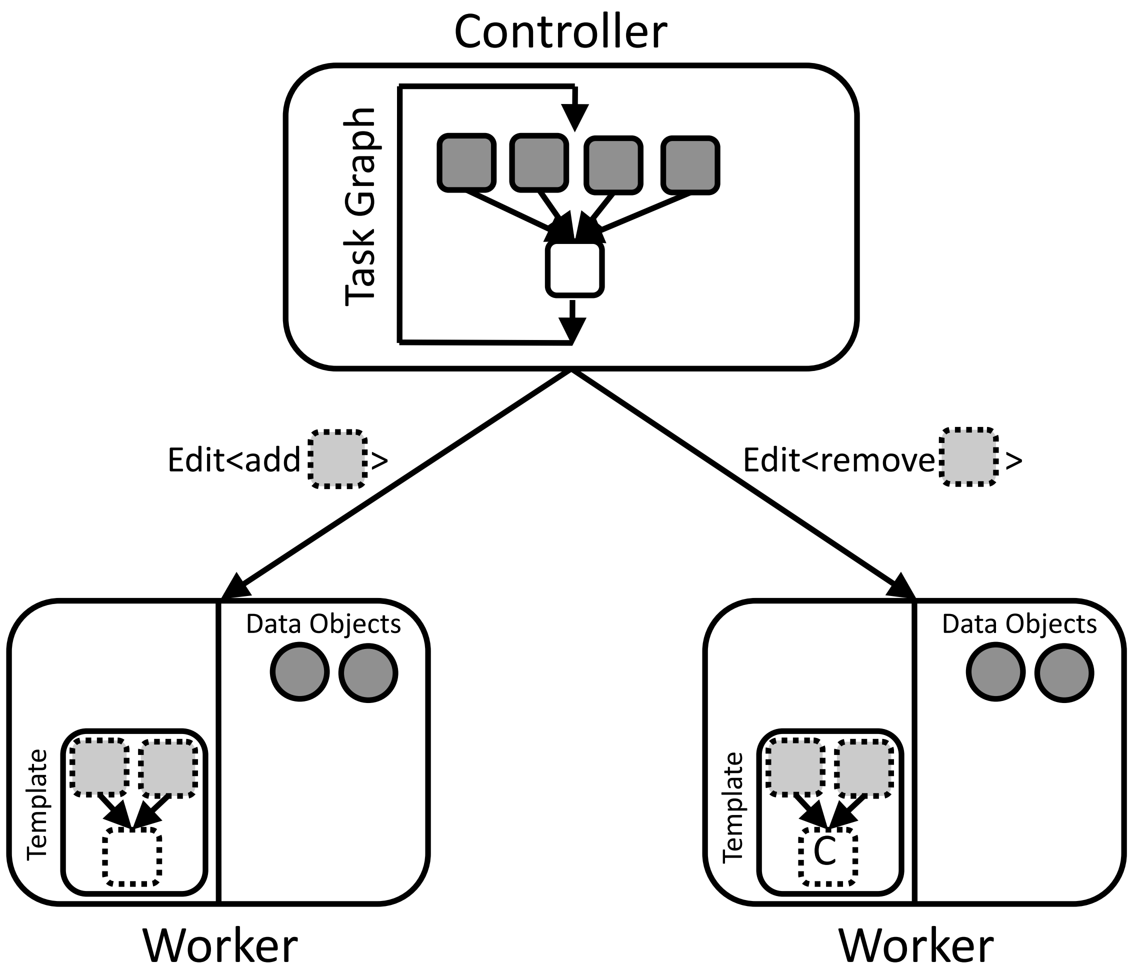
Execution Templates

Edits



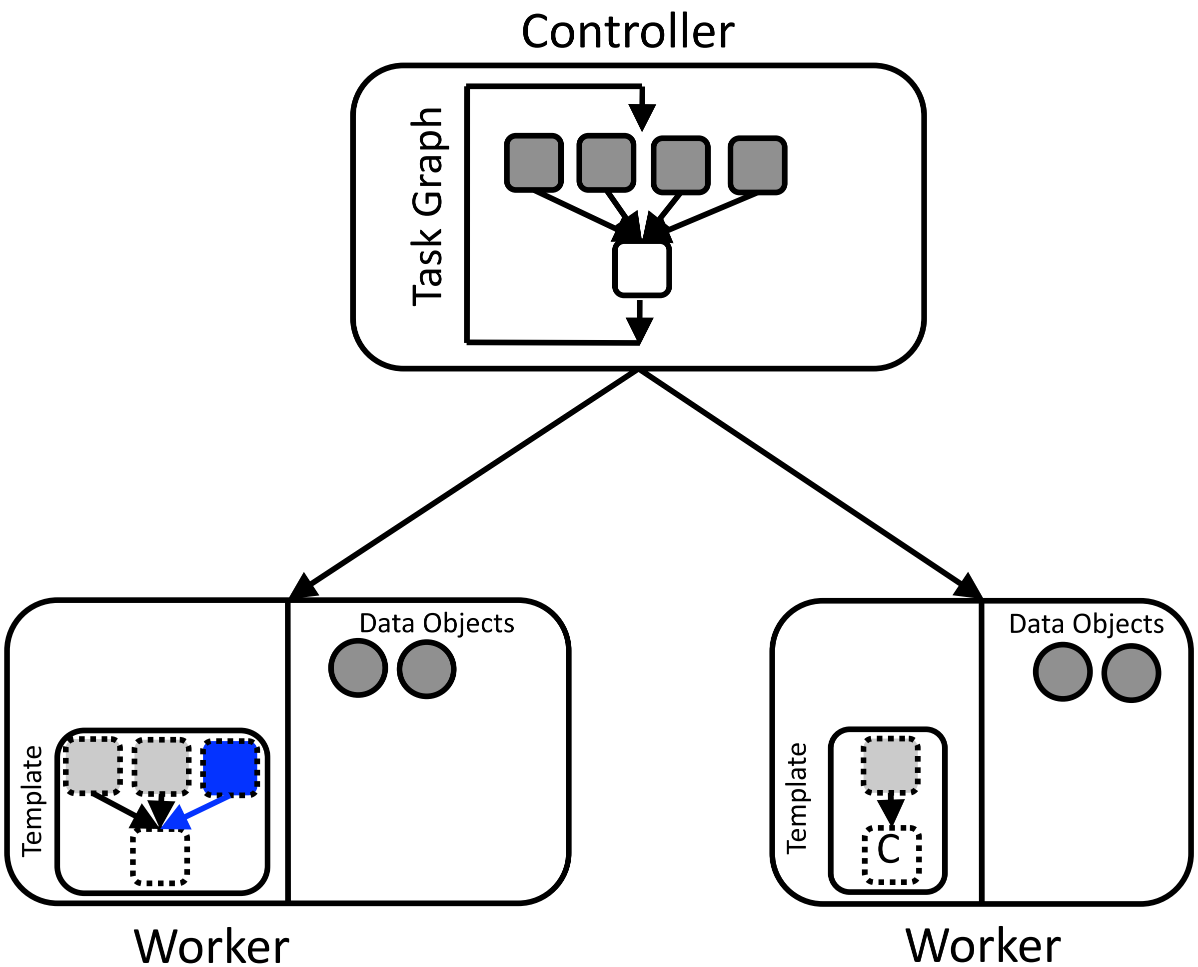
Execution Templates

Edits



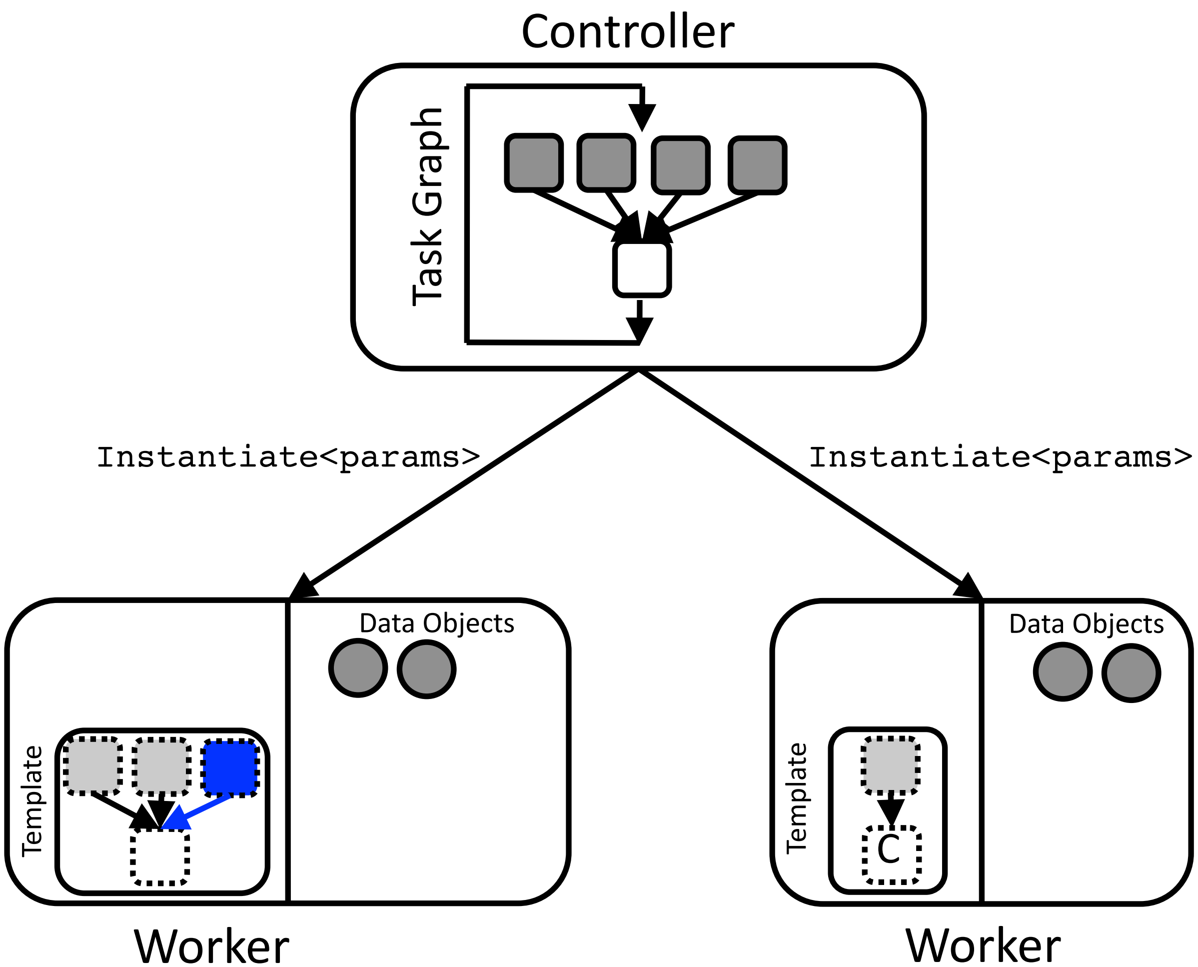
Execution Templates

Edits



Execution Templates

Edits



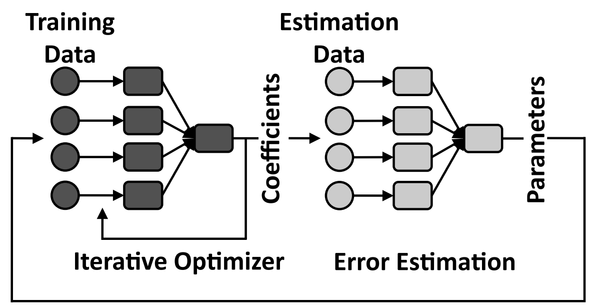
Execution Templates

The Devil is in the details.

- Caching tasks implies static behavior:
  - Templates and **dynamic scheduling**?
    - Reactive scheduling changes for load balancing.
    - Scheduling changes at the task granularity.
  - Templates and **dynamic control flow**?
    - Need to support nested loops.
    - Need to support data dependent branches.

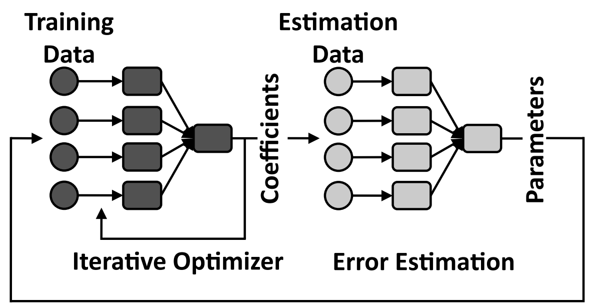
Execution Templates

Granularity



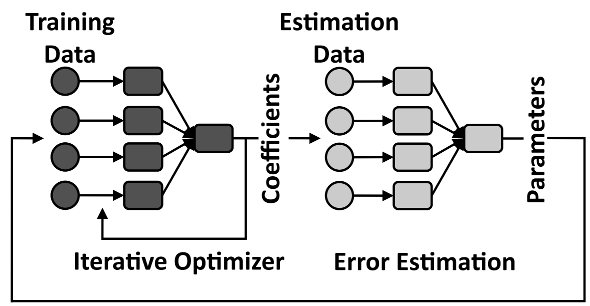
Execution Templates

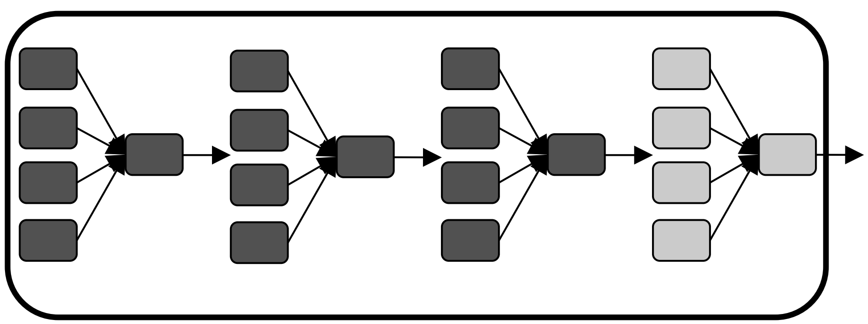
Granularity



- The more tasks cached in the template the better.
  - The cost of template instantiation is amortized over greater number of tasks.
  - But **loop unrolling** only works for static control flow.

Execution Templates
Granularity

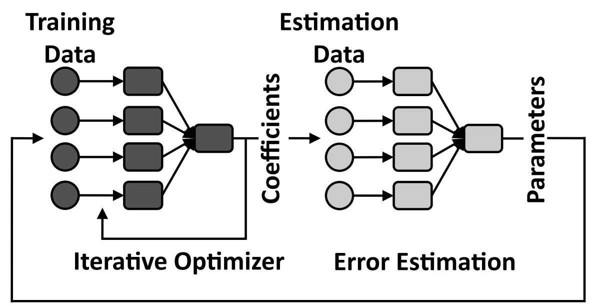


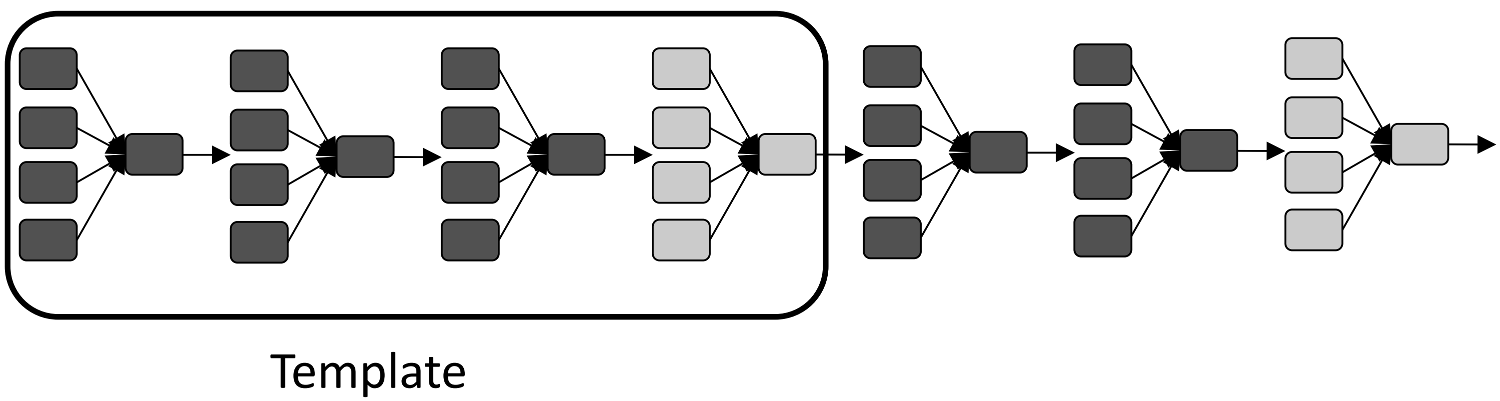


Template

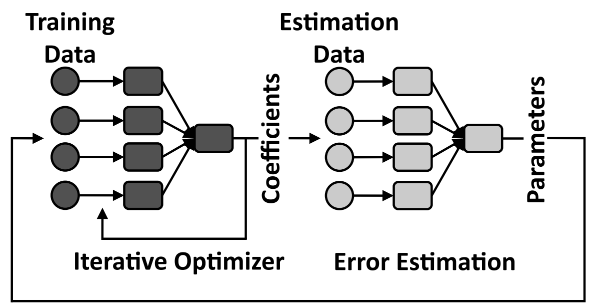
Execution Templates

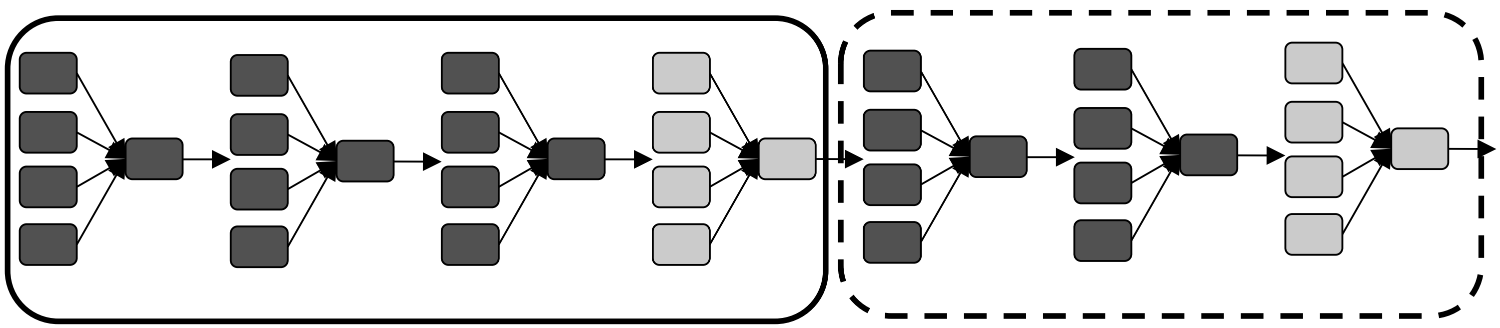
Granularity





Execution Templates
Granularity



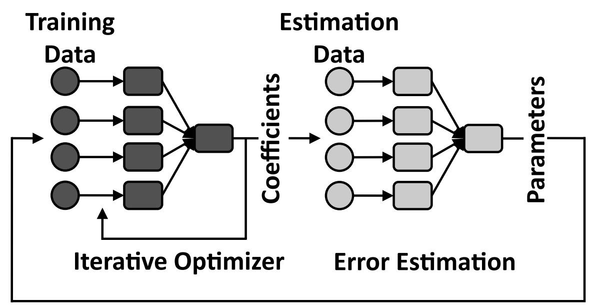


Template

- Cannot reuse the template (only two iterations of the inner loop).

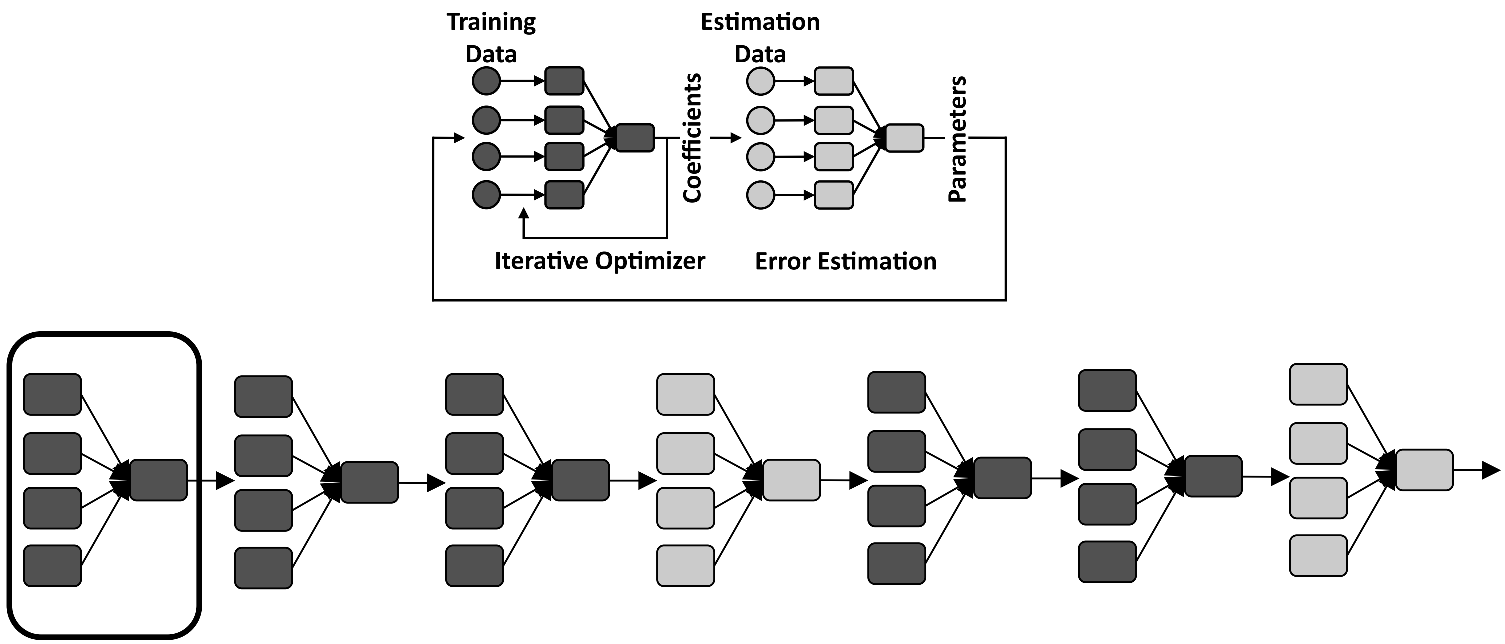
Execution Templates

Granularity



- Templates cannot go beyond a branch in the driver program.
- Execution templates operates at the granularity of **basic blocks**:
  - A code block with single entry and no branches except at the end.
  - It is the biggest block without sacrificing **dynamic control flow**.

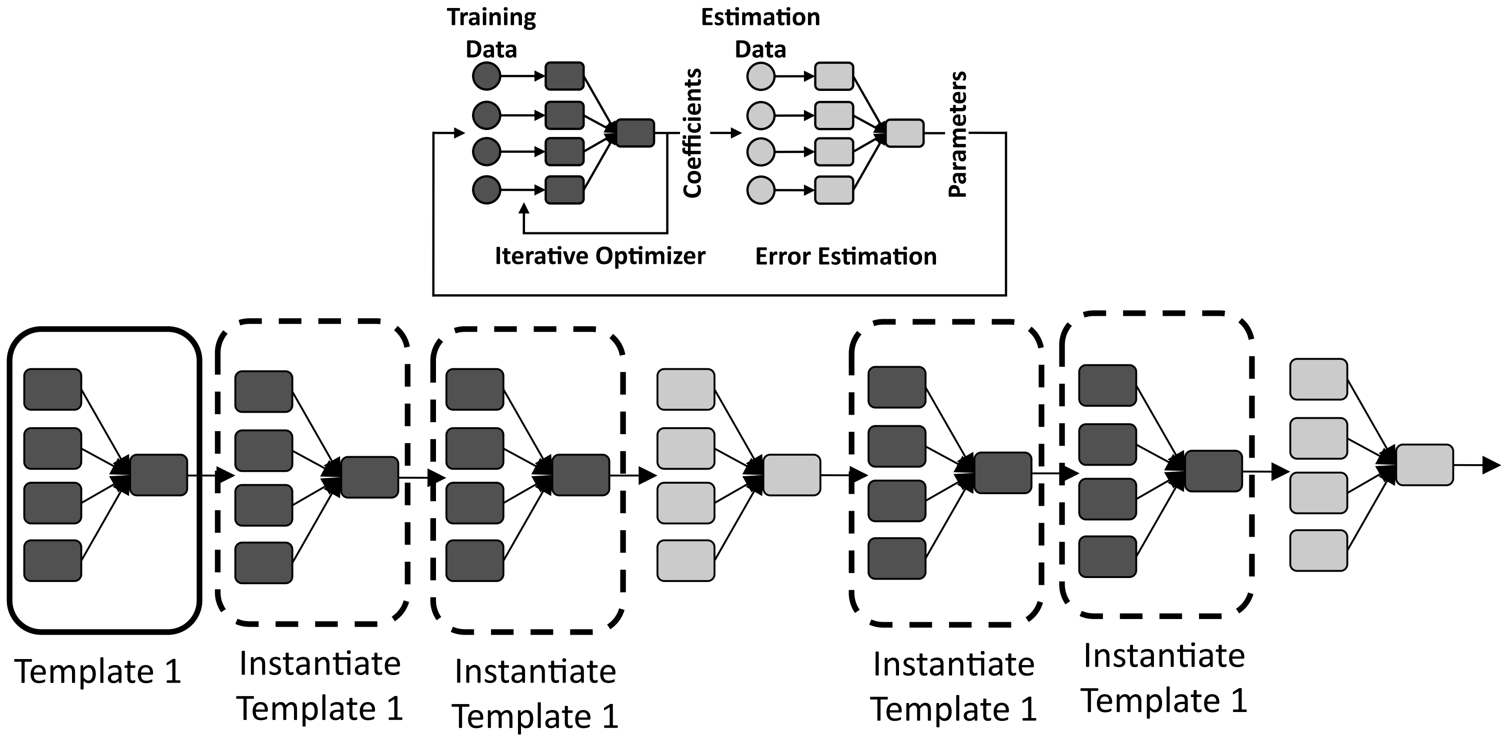
Execution Templates
Granularity



Template 1

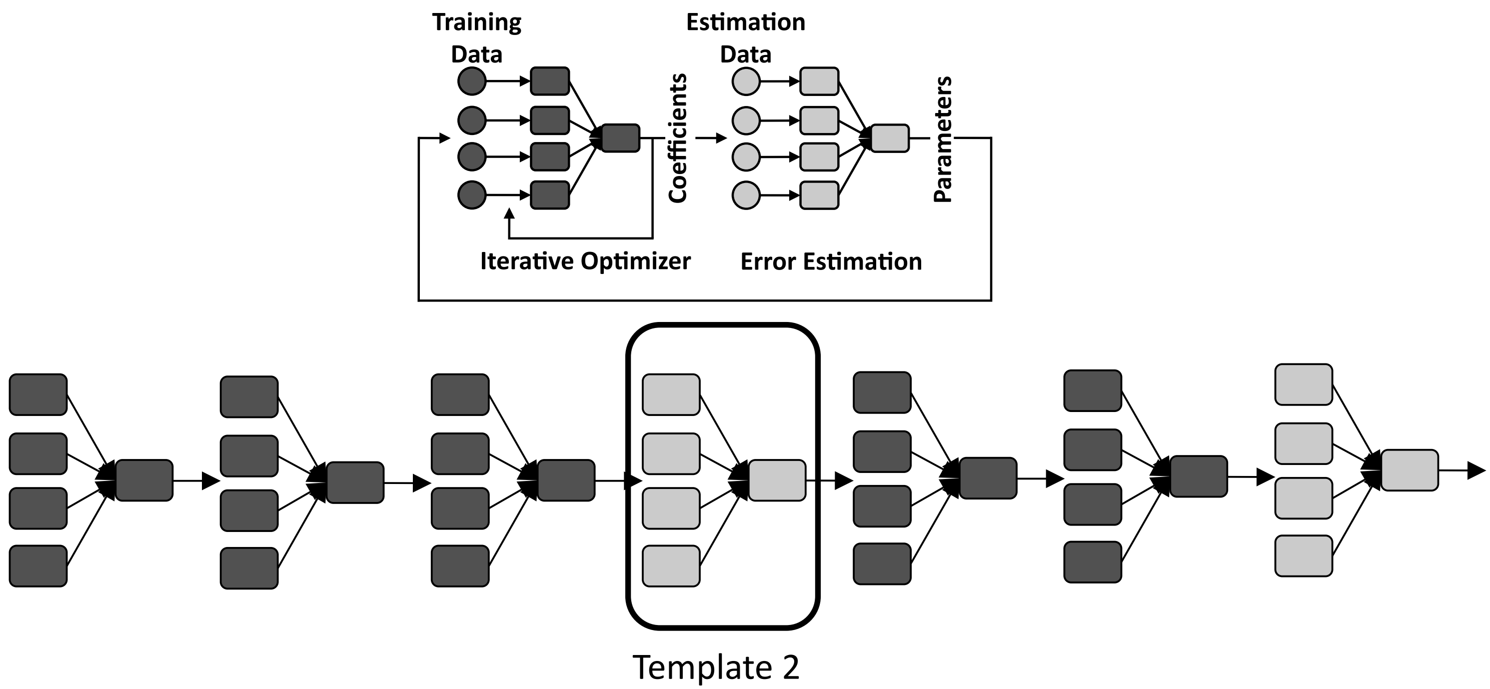
Execution Templates

Granularity



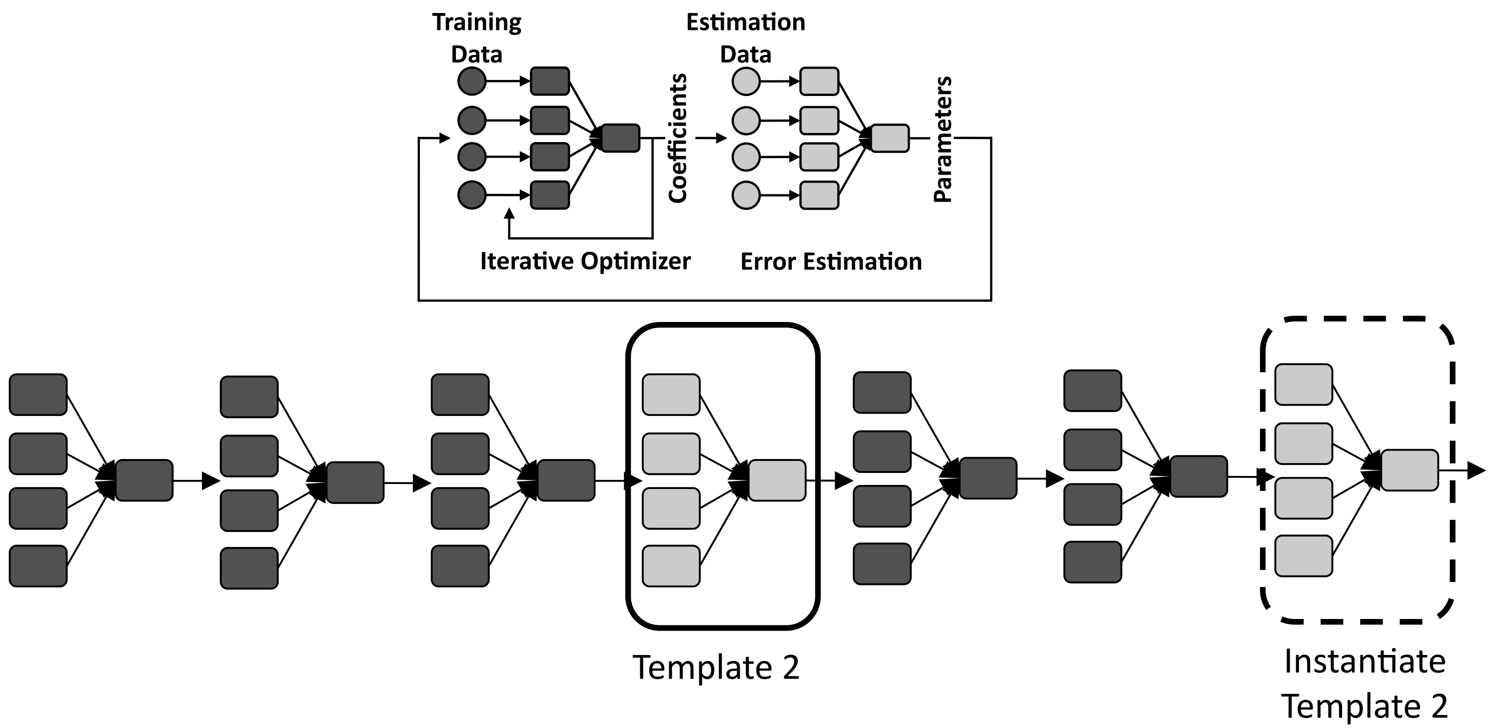
Execution Templates

Granularity



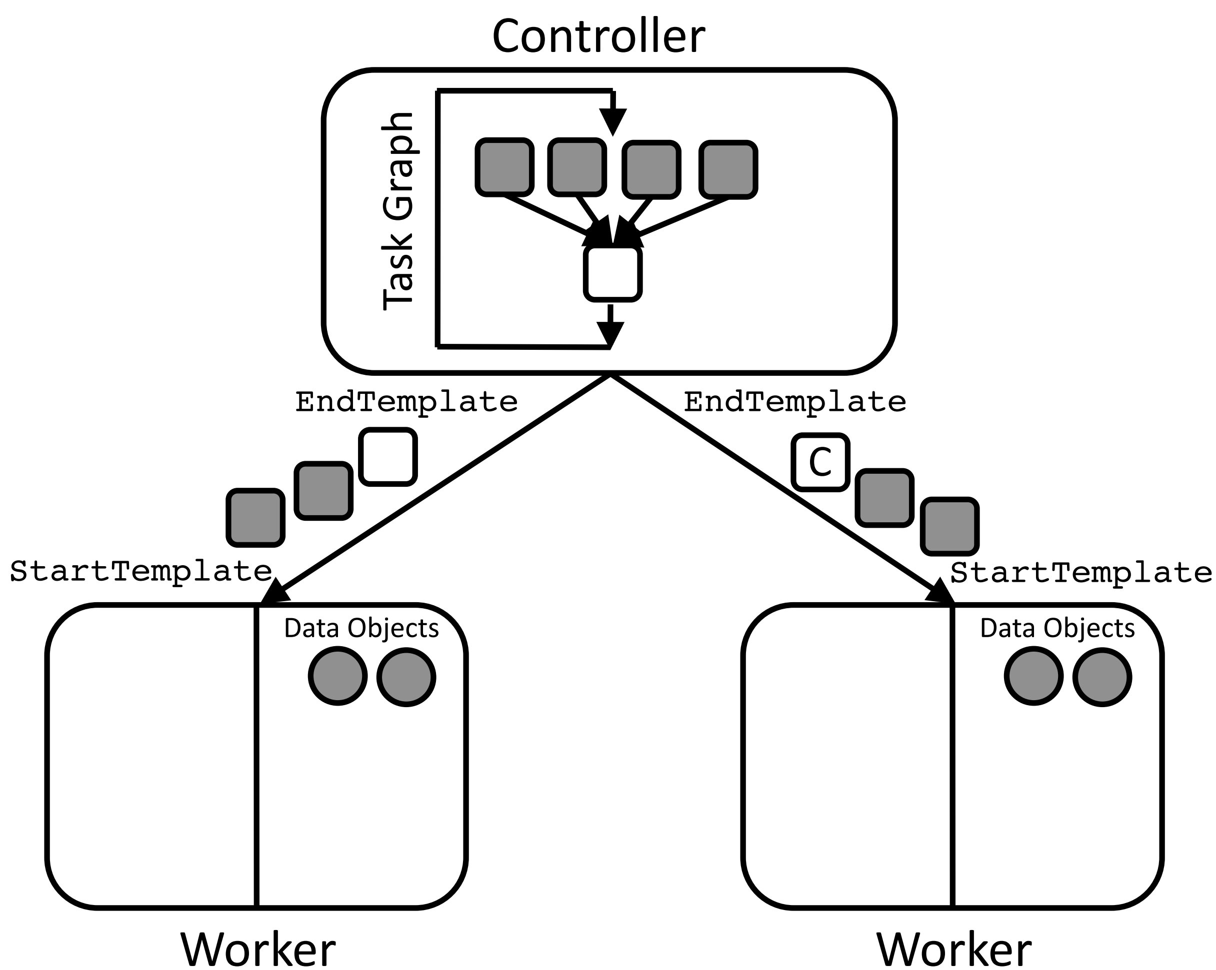
Execution Templates

Granularity



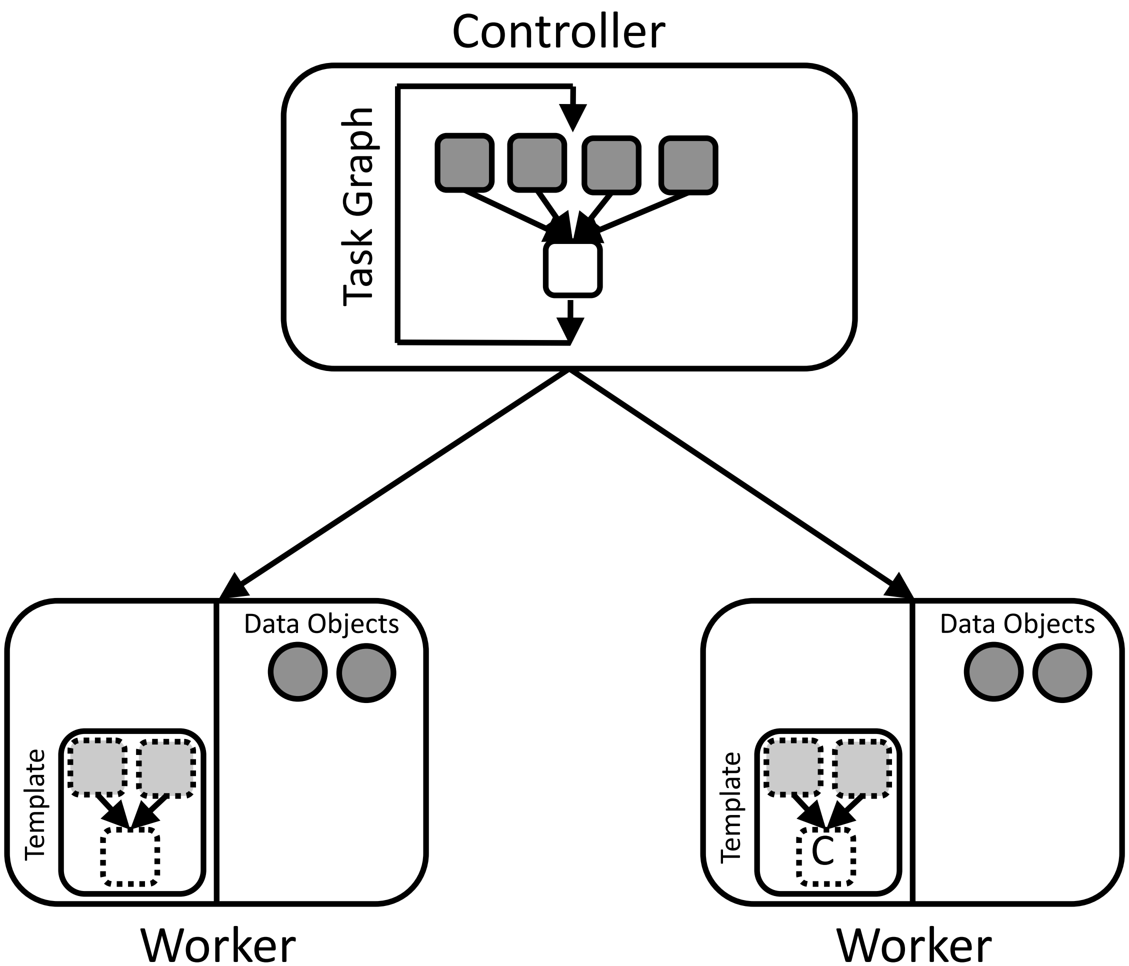
Execution Templates

Granularity

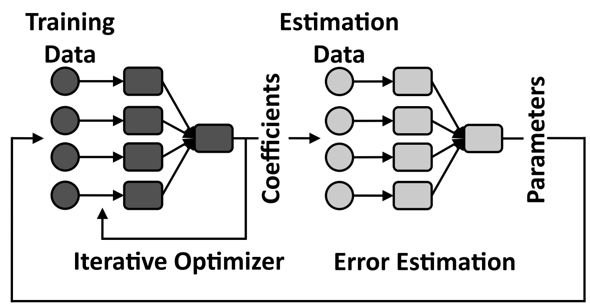


Execution Templates

Granularity



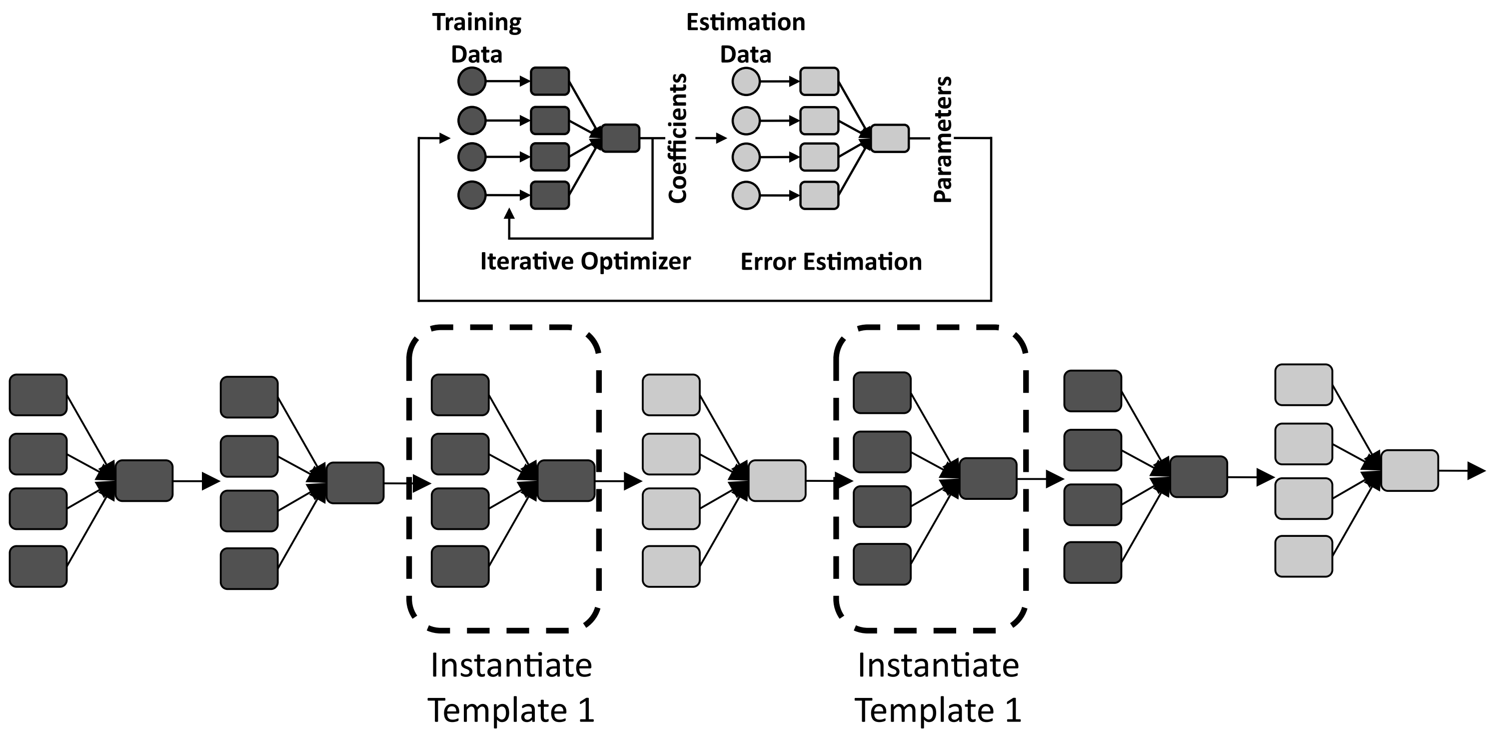
Execution Templates
Patching



- With dynamic control flow a basic block can have different entries.
- The execution state is not similar in all circumstances.

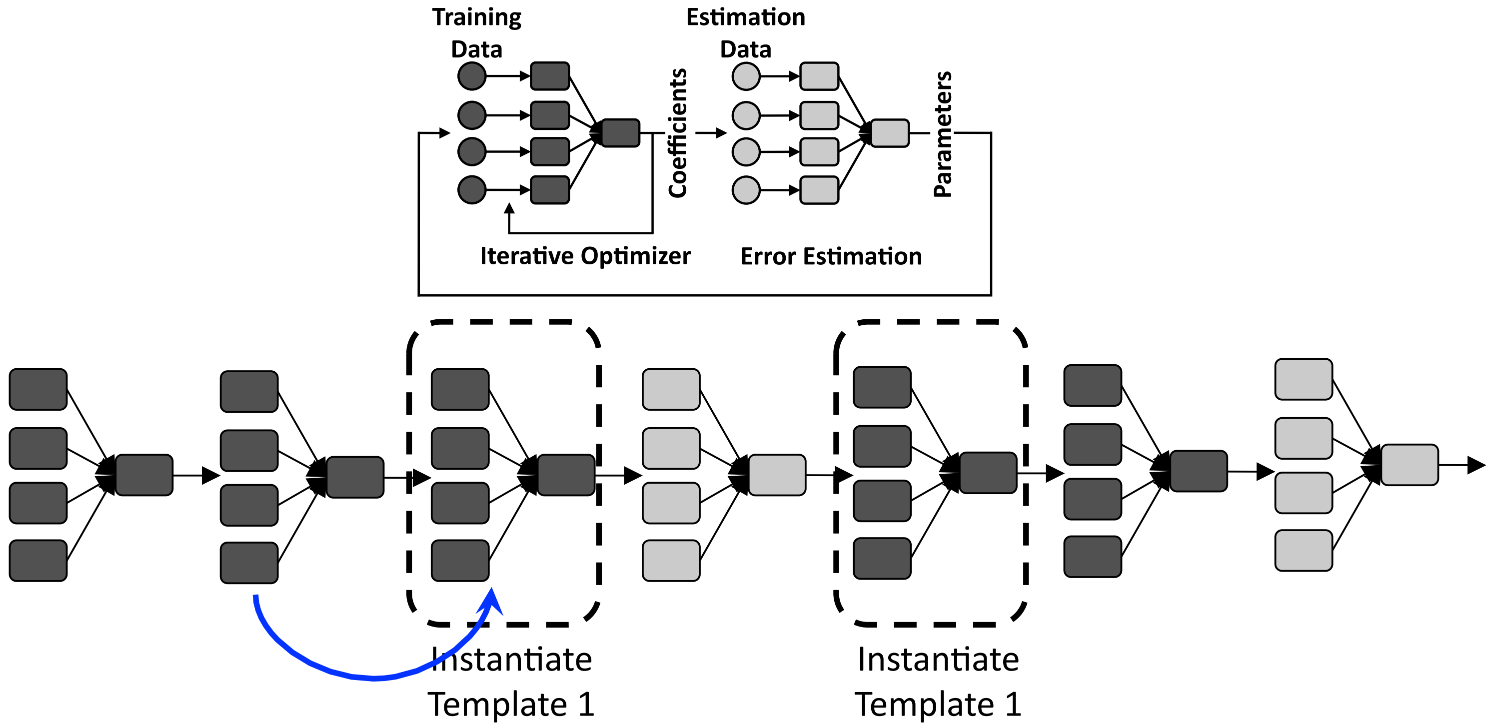
Execution Templates

Patching



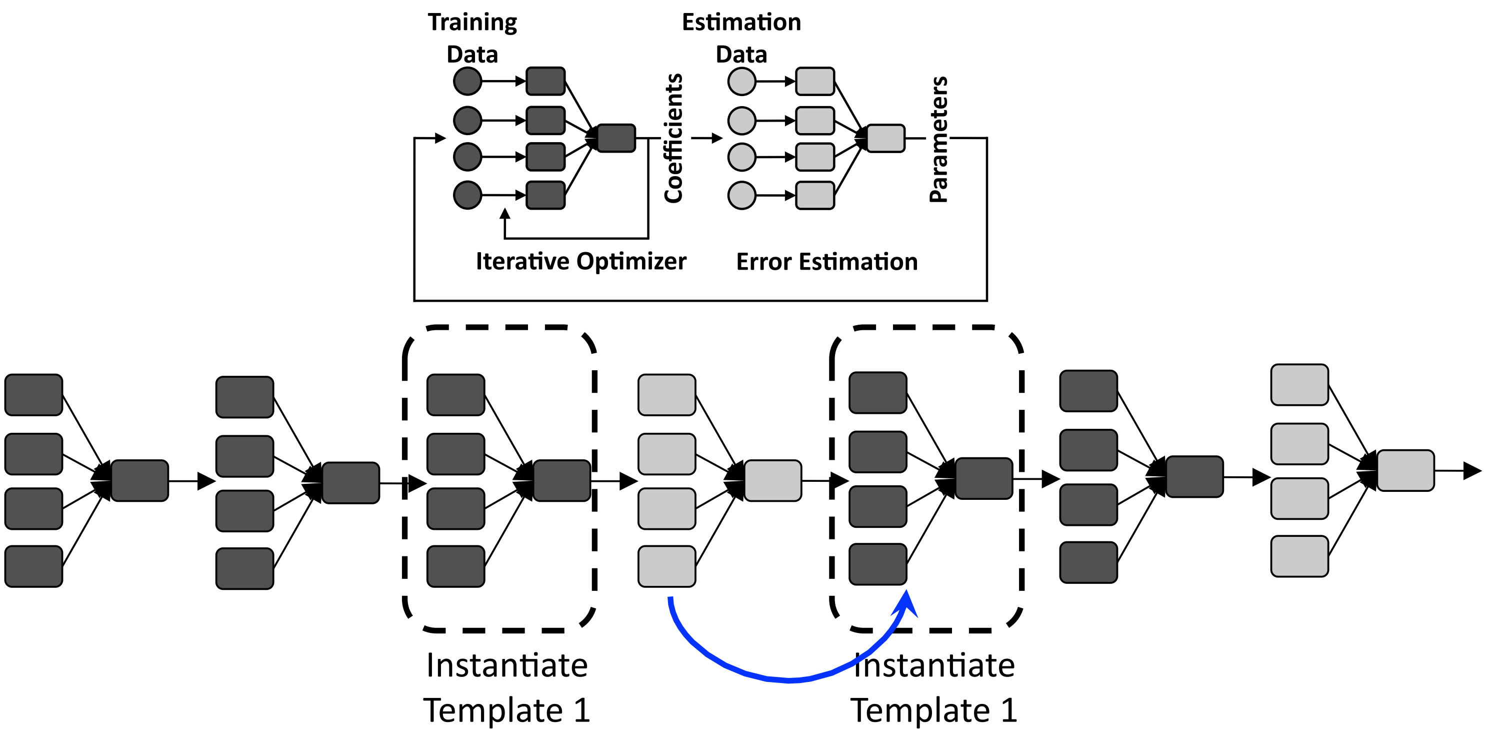
Execution Templates

Patching



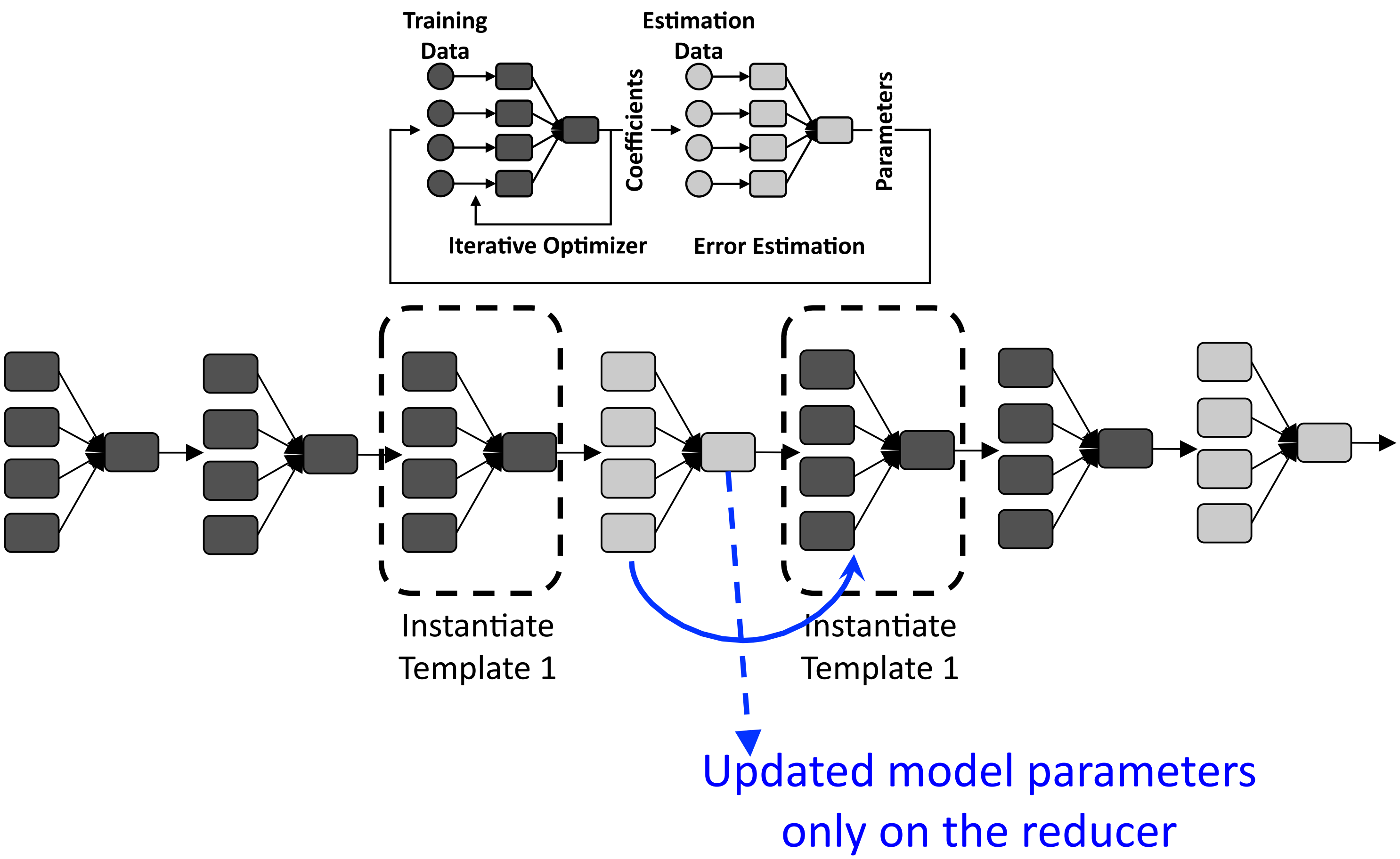
Execution Templates

Patching



Execution Templates

Patching



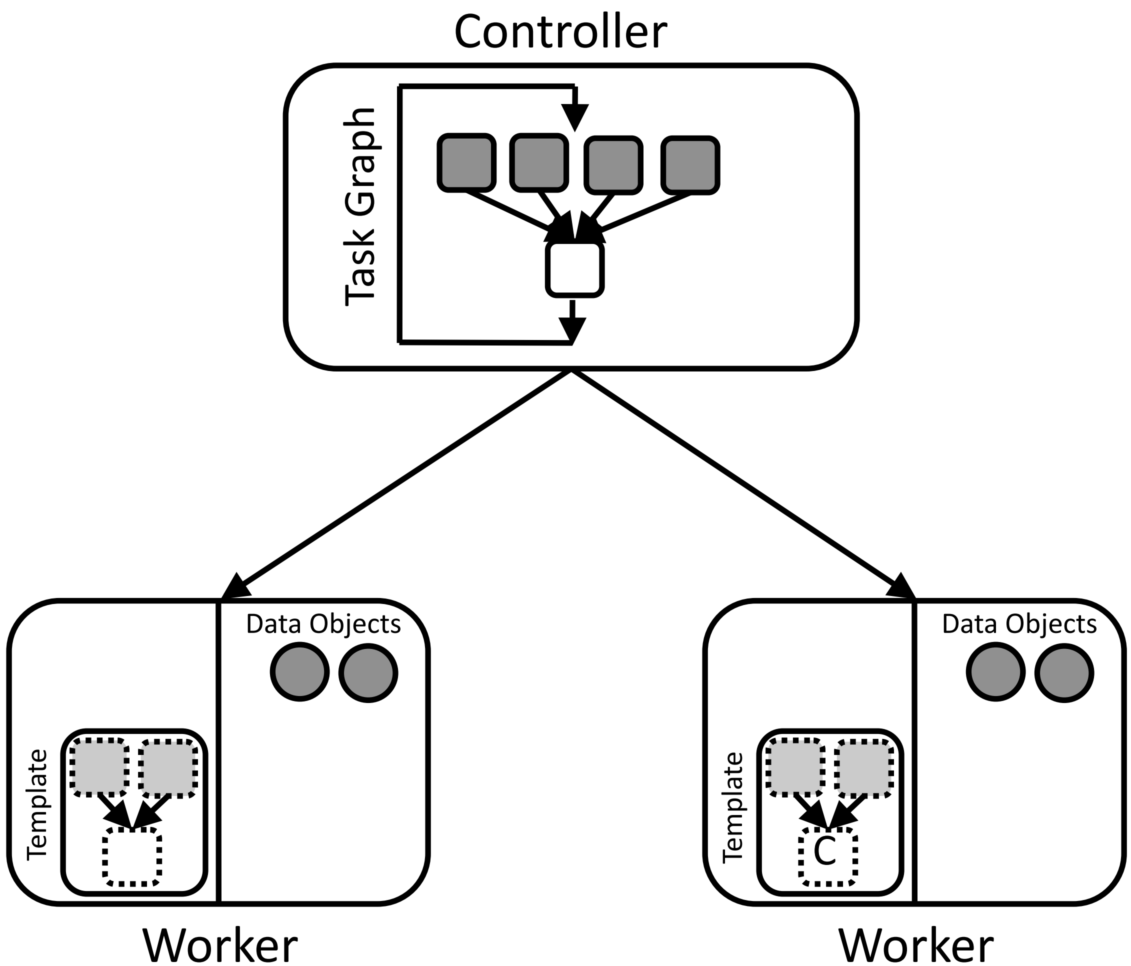
Execution Templates

Patching

- Each template has a set of **preconditions** that need to be satisfied before it can be instantiated.
  - For example the set of data objects in memory, accessed by the tasks cached in the template.

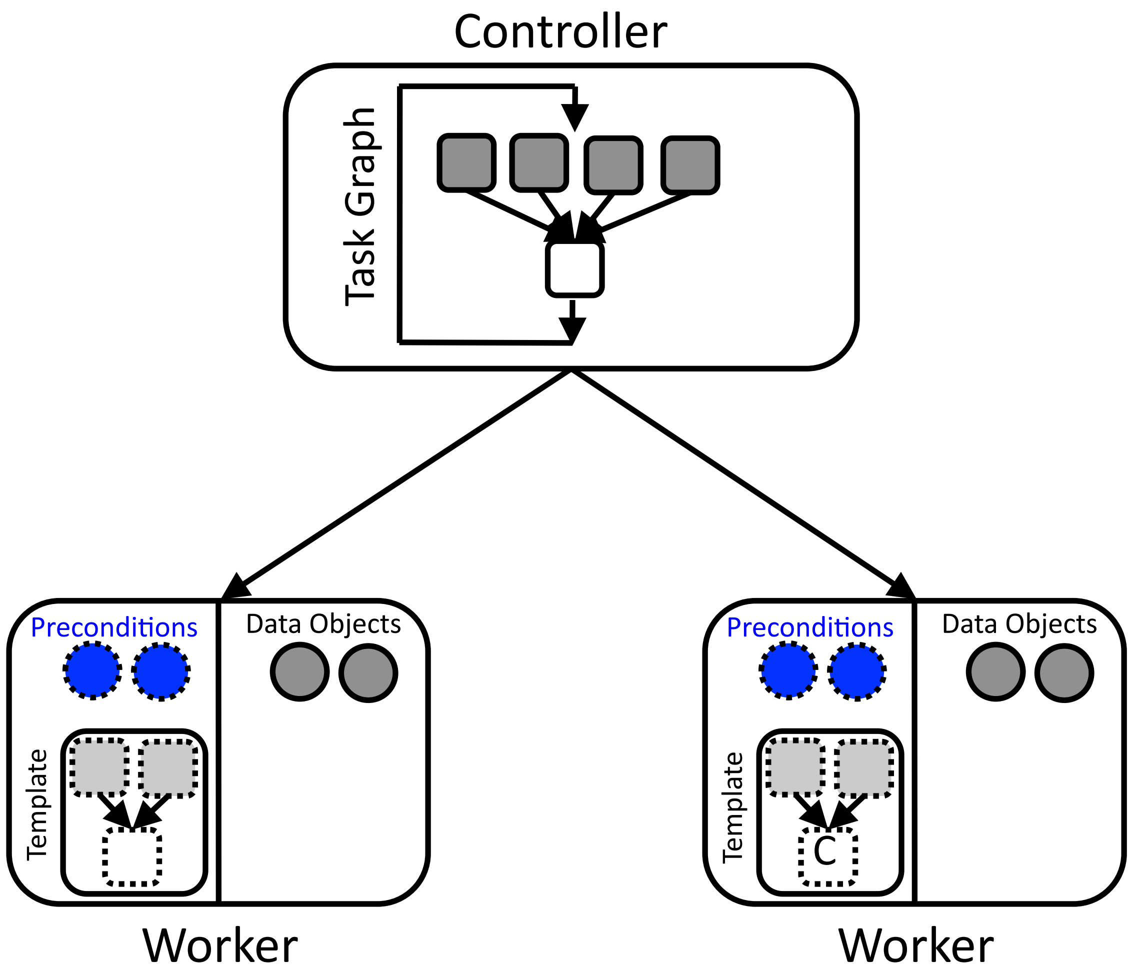
Execution Templates

Patching



Execution Templates

Patching

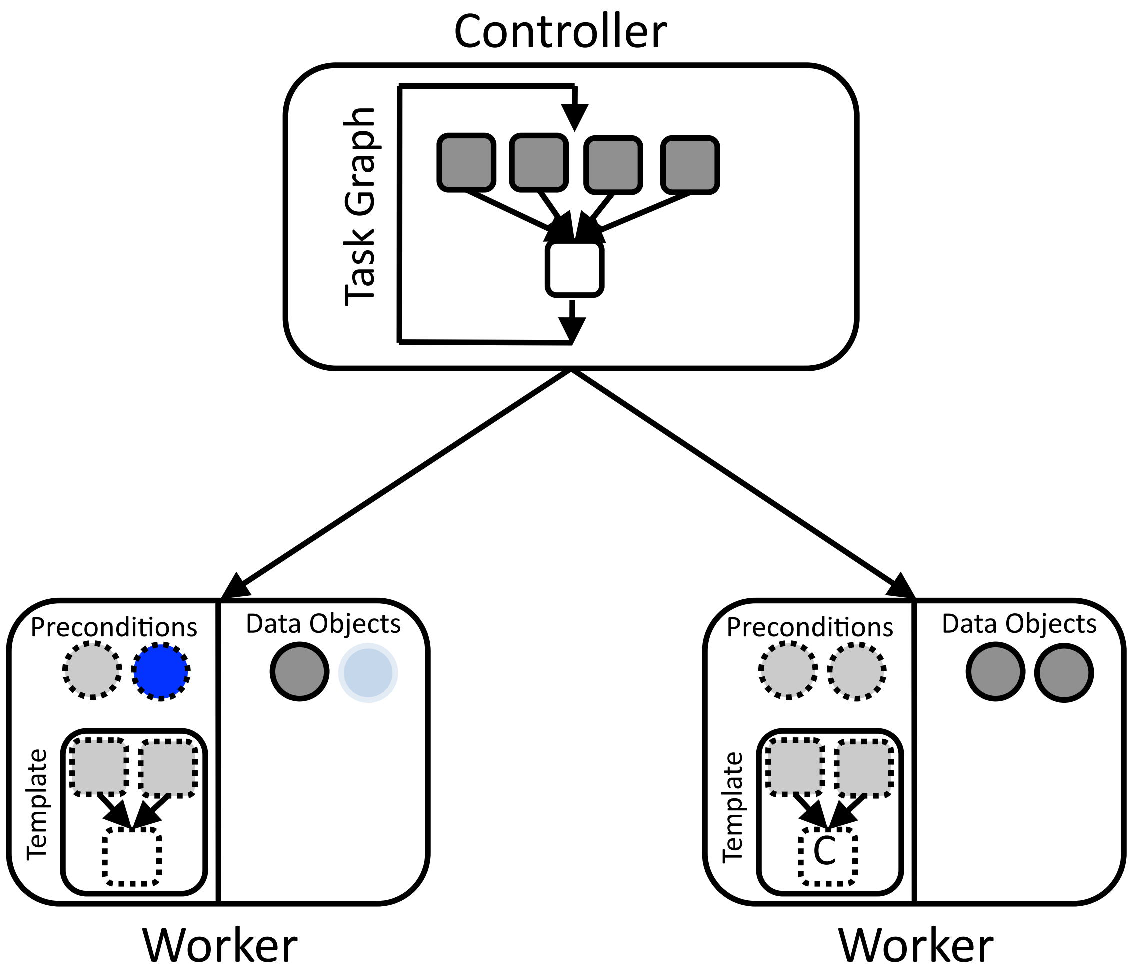


Execution Templates
Patching

- Each template has a set of **preconditions** that need to be satisfied before it can be instantiated.
  - For example the set of data objects in memory, accessed by the tasks cached in the template.
- Worker state might not match the preconditions of the template in all circumstances.
- Controller **patches** the worker state before template instantiation, to satisfy the preconditions.

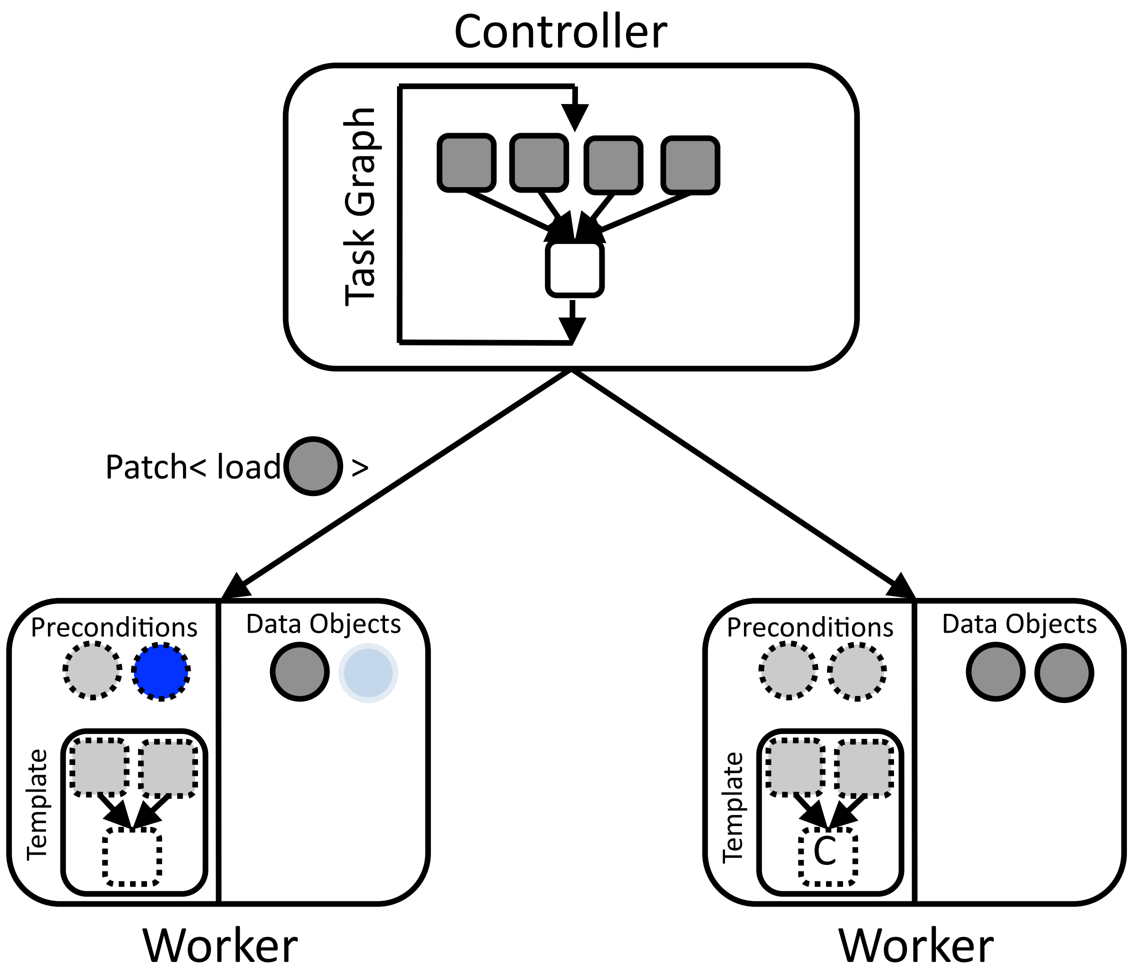
Execution Templates

Patching



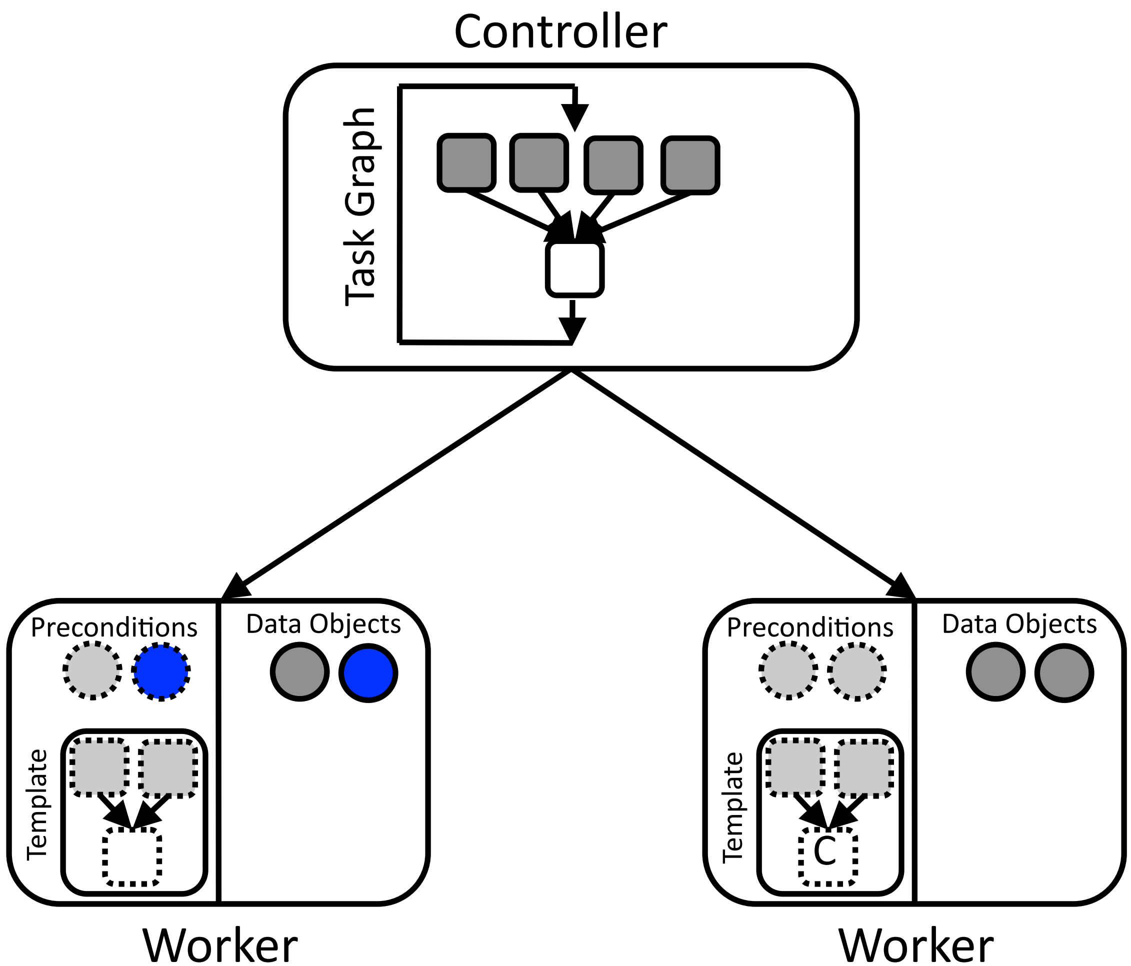
Execution Templates

Patching



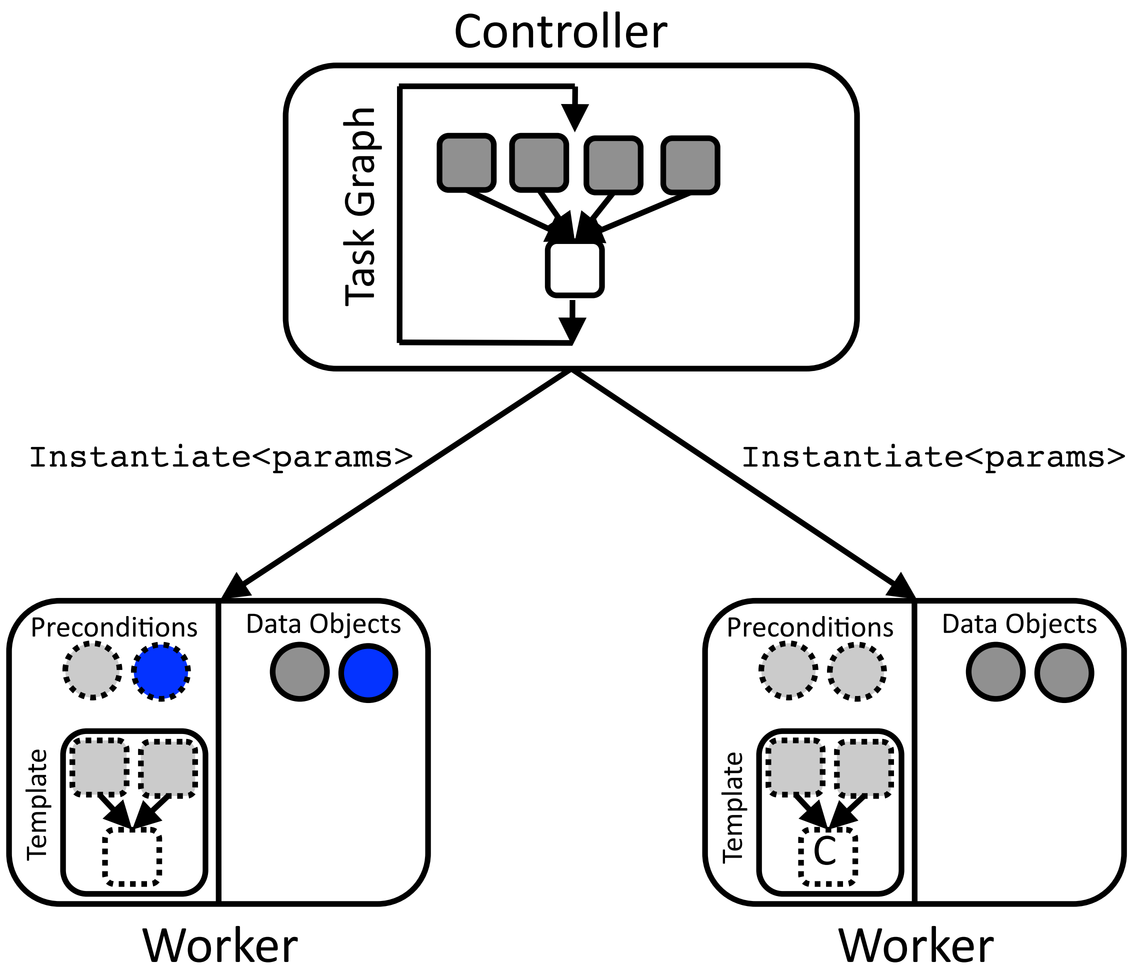
Execution Templates

Patching



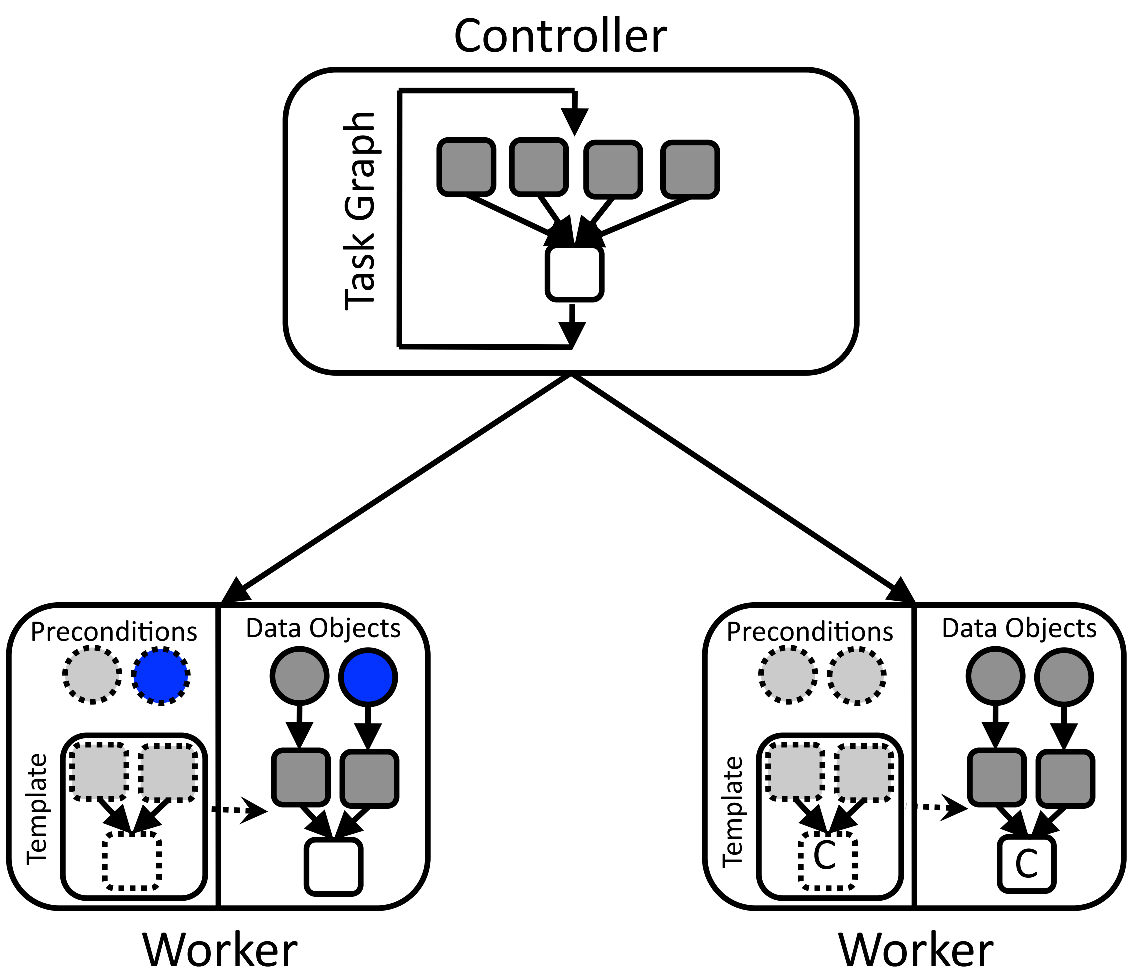
Execution Templates

Patching



Execution Templates

Patching



Execution Templates
Mechanisms Summary

- **Instantiation**: spawn a block of tasks without processing each task individually from scratch. It helps increase the **task throughput**.
- **Edits**: modifies the content of each template at the granularity of tasks. It enables fine-grained, **dynamic scheduling**.
- **Patches**: In case the state of the worker does not match the preconditions of the template. It enables **dynamic control flow**.

This talk

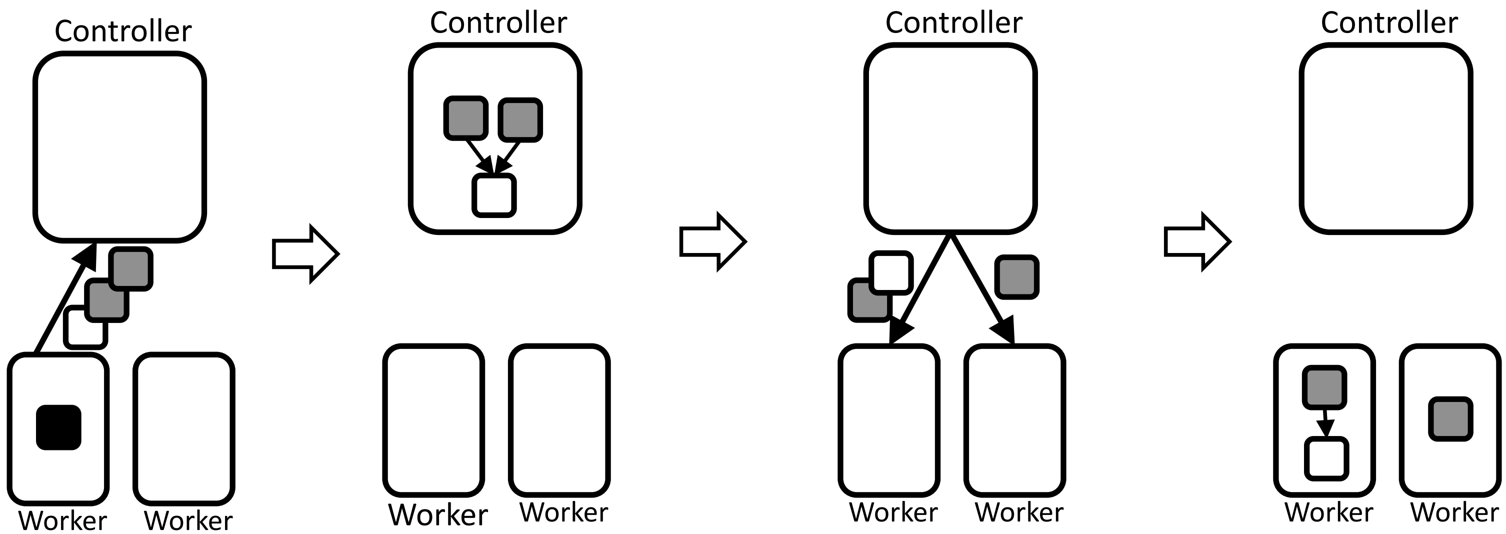
- Control Plane: the Emerging Bottleneck
- Design Scope of the Control Plane
- Execution Templates
- Nimbus: a Framework with Templates
- Evaluation

Nimbus

- Nimbus is designed for low latency, fast computations in the cloud.
  - Implemented in C++ (the core library is ~35,000 semicolons).
  - Mutable data model to allow in-place operations.
- Nimbus embeds execution templates for its control plane.
  - The centralized controller allows dynamic scheduling and resource allocation.
  - Execution templates help deliver high task throughput at scale.
- Nimbus supports traditional data analytics as well as Eulerian and hybrid graphical simulations; for the first time in a cloud framework.
  - Supervised/unsupervised learning algorithms, graph library.
  - PhysBAM library (water, smoke, etc.)

Nimbus

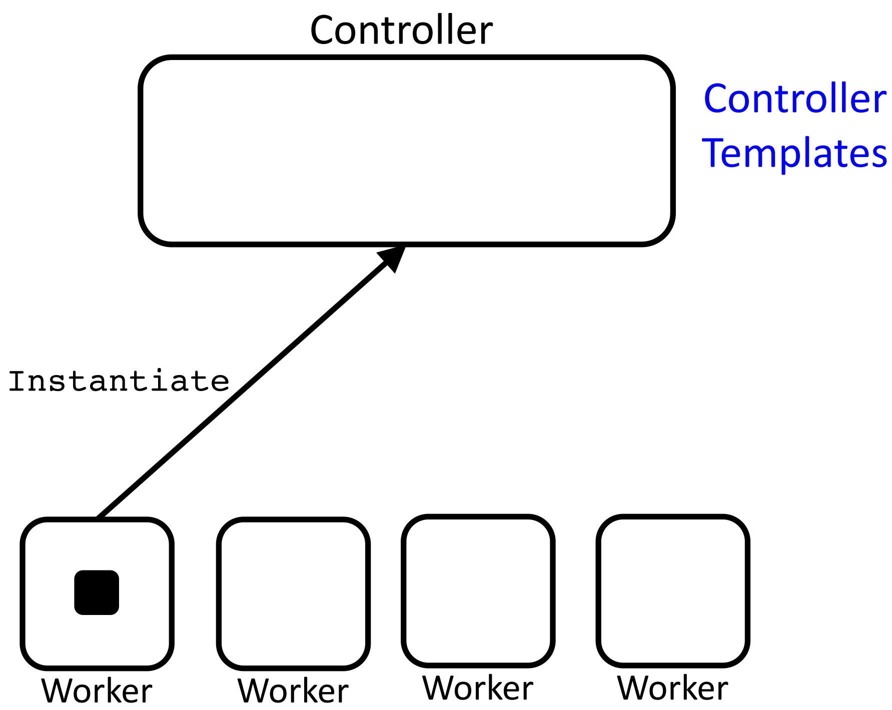
Control Flow



- Tasks spawn other tasks for execution (similar to Legion).
- Driver program is a lineage of tasks executing on the workers.
- More flexible DAG for the task graph.
  - Not just narrow and wide dependencies.
  - Needed for graphical simulations.

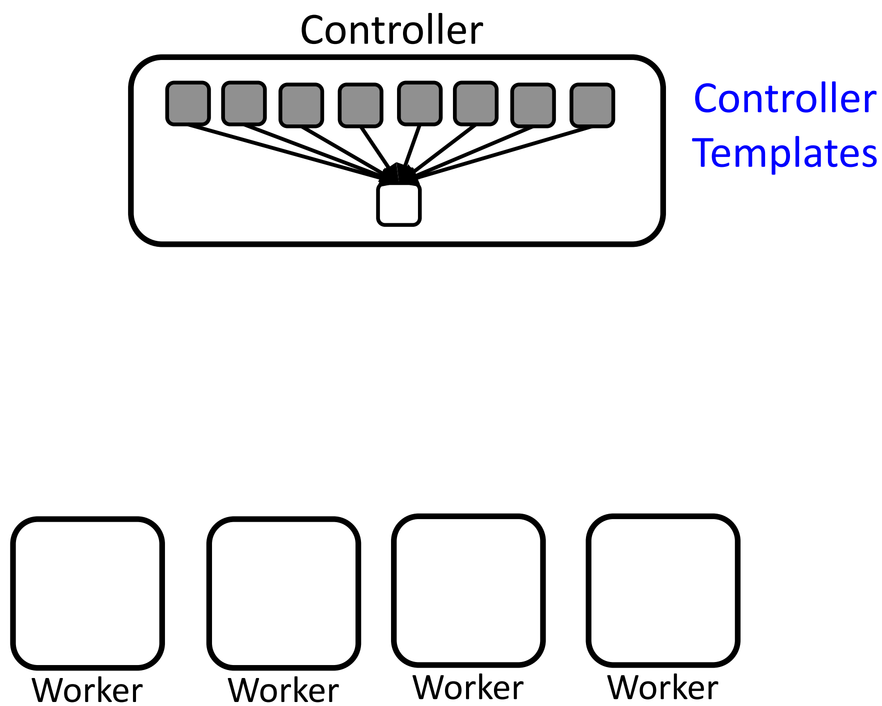
Nimbus

Controller and Worker Templates



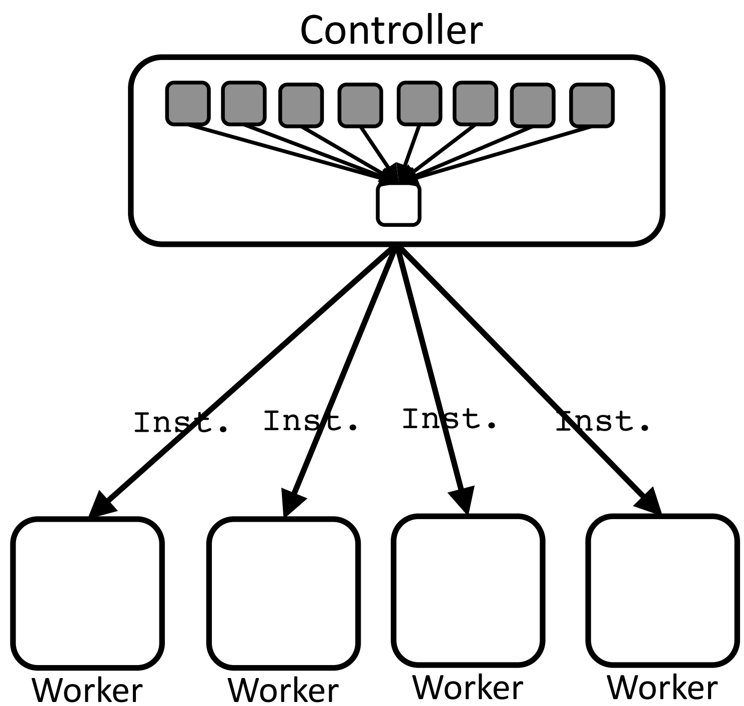
Nimbus

Controller and Worker Templates



Nimbus

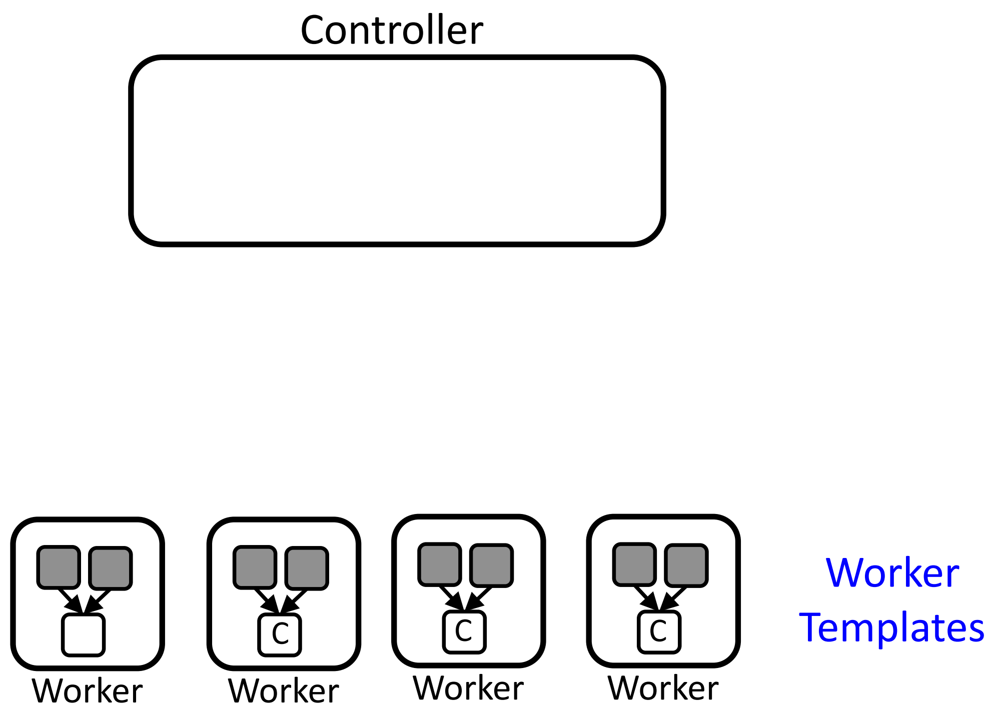
Controller and Worker Templates



Worker Templates

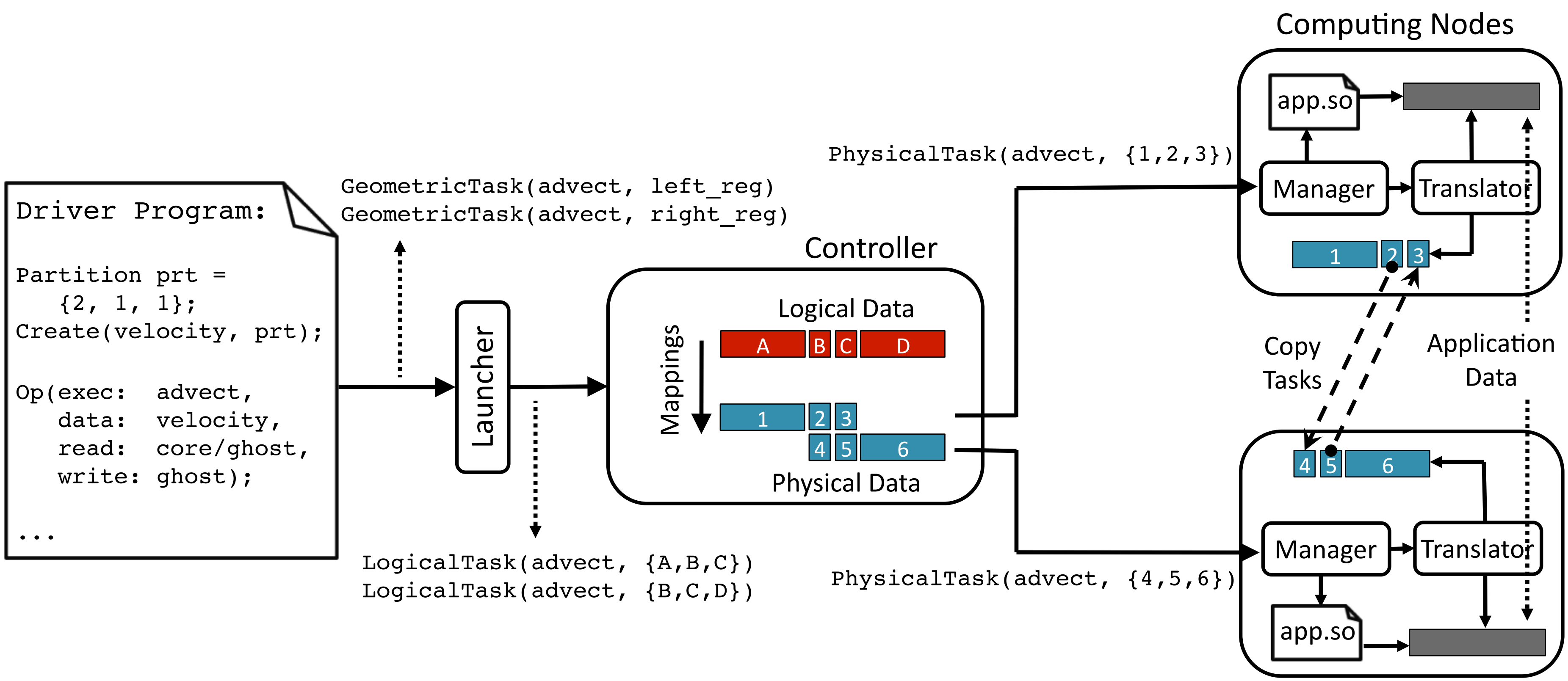
Nimbus

Controller and Worker Templates



Nimbus

Graphical Simulations



- The goal is to automatically distribute sequential library kernels.
- Four layer data abstraction (geometric, logical, physical, application).
- Automatic translation and caching between the data layers.

nimbus.stanford.edu

- For more information you can visit Nimbus website.

This talk

- Control Plane: the Emerging Bottleneck
- Design Scope of the Control Plane
- Execution Templates
- Nimbus: a Framework with Templates
- Evaluation

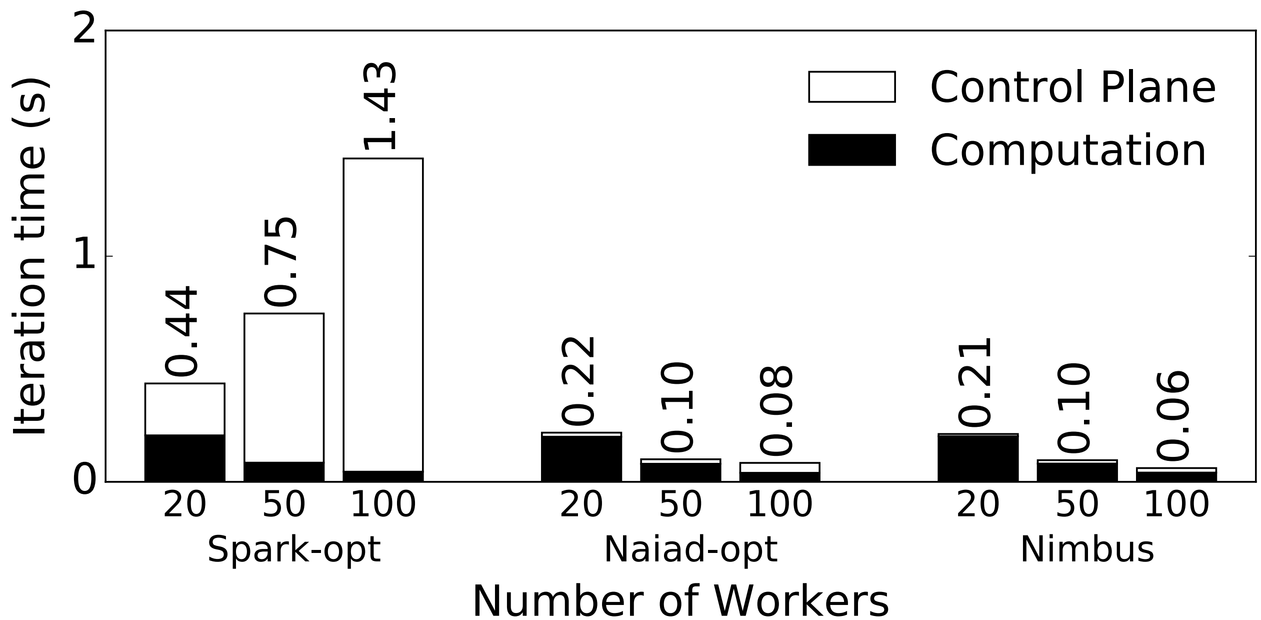
Evaluation

Results Summary

- Control plane task throughput:
  - Execution templates match the strong scaling performance of frameworks with distributed control plane design.
- Dynamic scheduling:
  - Execution templates allows low cost, reactive scheduling and dynamic resource allocation similar to a centralized frameworks.
- Dynamic control flow:
  - Execution templates can handle applications with nested loops and data dependent branches with low overhead.

Evaluation

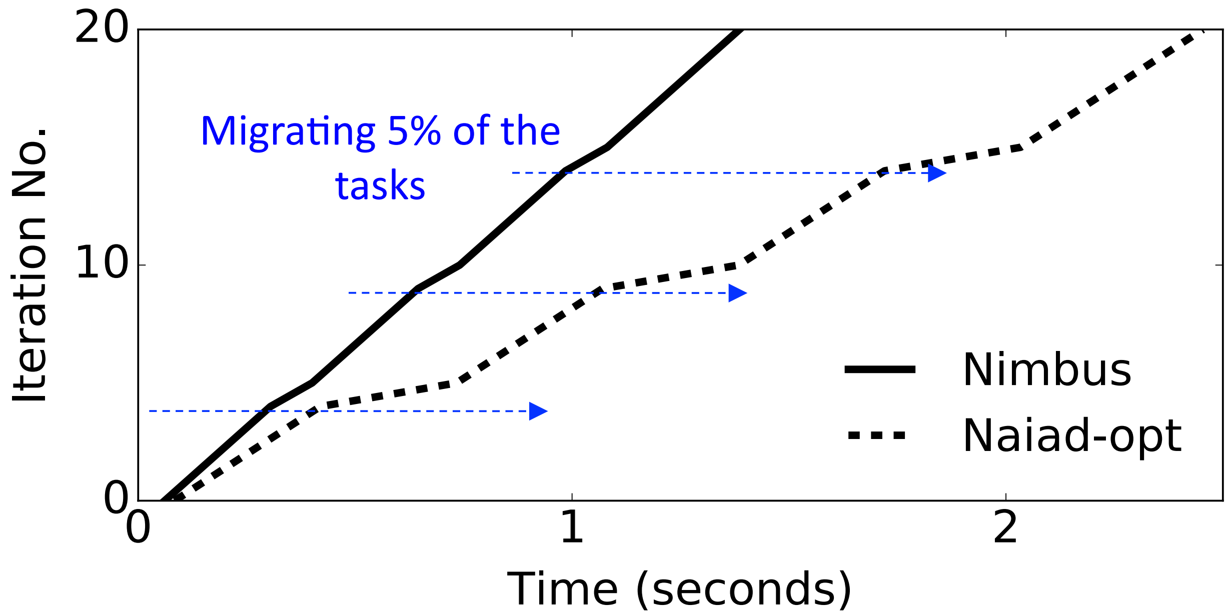
Strong Scalability with Templates



- Logistic regression over data set of size 100GB.
- Spark-opt and Naiad-opt, runs tasks as fast as C++ implementation.
- Nimbus centralized controller with execution templates matches the performance of Naiad with a distributed control plane.

Evaluation

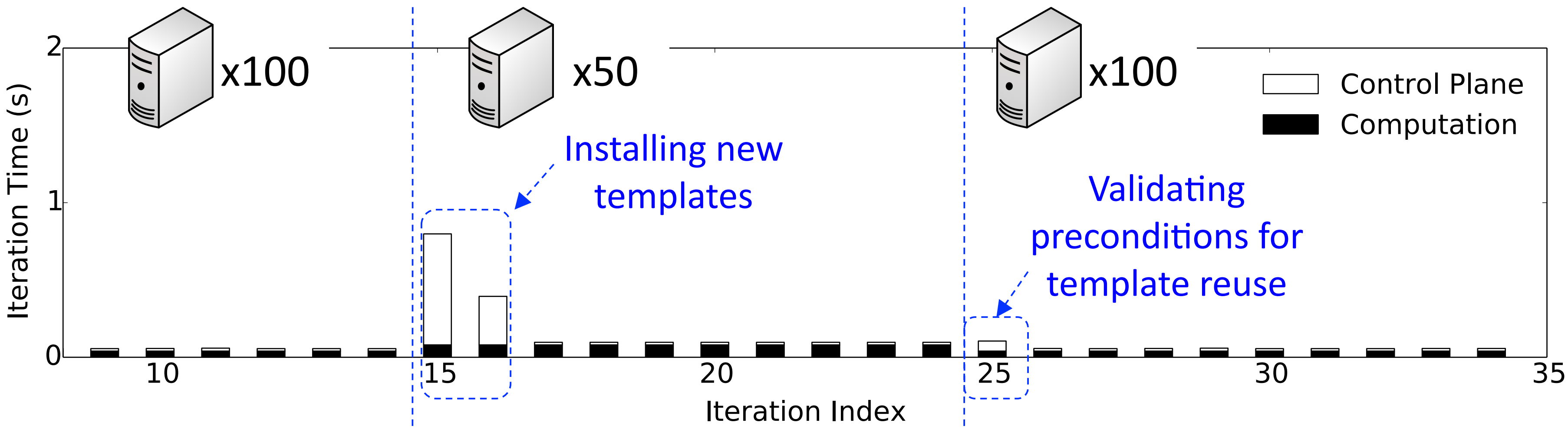
Reactive, Fine-Grained Scheduling with Templates



- Logistic regression over data set of size 100GB, on 100 workers.
- Naiad-opt curve is simulated (migrations every 5 iterations).
- Execution templates allow low cost, reactive scheduling changes through edits at task granularity.
  - Single edit overhead is only 41μs (in average).

Evaluation

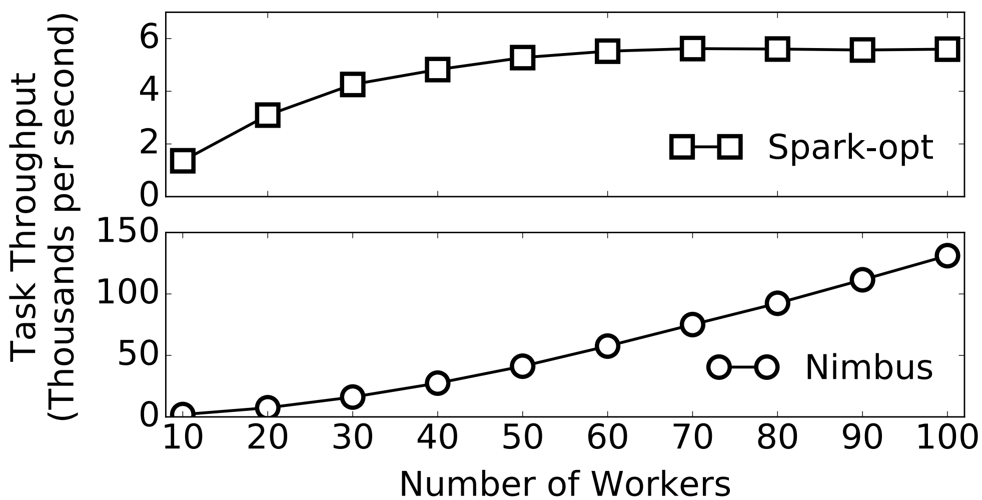
Dynamic Resource Allocation with Templates



- Logistic regression over 100GB of data, on 50/100 workers.
- One-time template installation cost is ~40% of direct task scheduling.
- Nimbus allows dynamic resource allocation.
- Nimbus installs multiple versions of a template depending on resources.

Evaluation

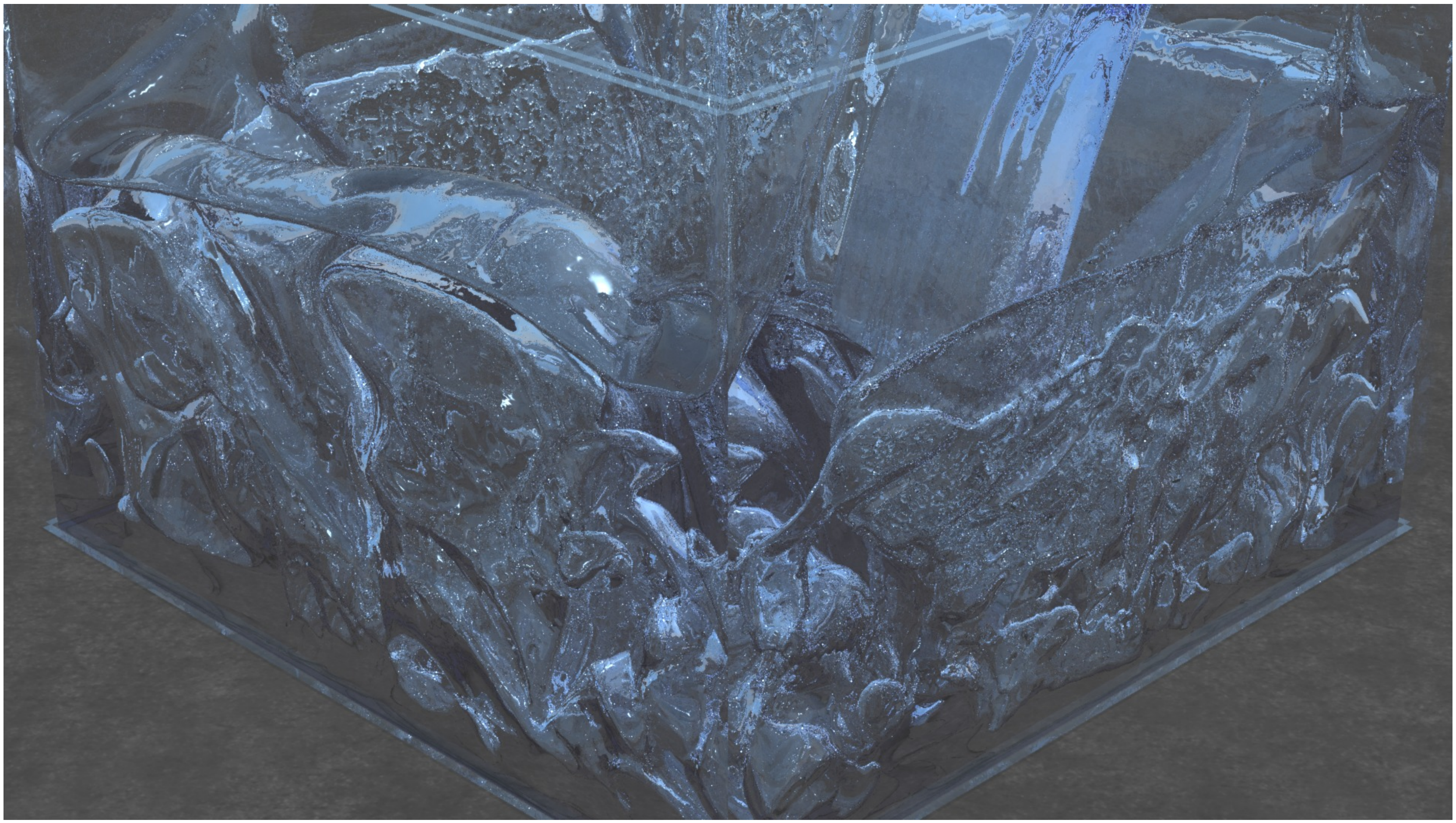
High Task Throughput with Templates



- Spark and Nimbus both have centralized controller.
- Nimbus task throughput scales super linearly with more workers.
  - $O(N^2)$: more tasks and shorter tasks, simultaneously.
- For a task graphs with single stage:
  - Instantiation cost is <2µs per task (500,000 tasks per second).

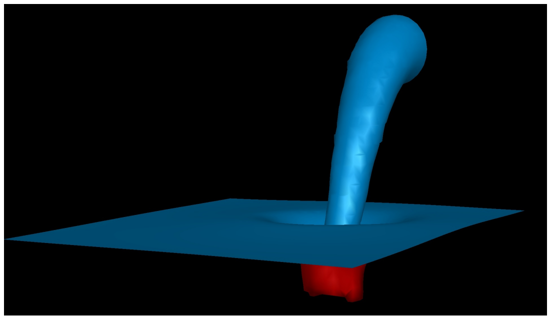
Evaluation

Graphical Simulations Distributed in Nimbus

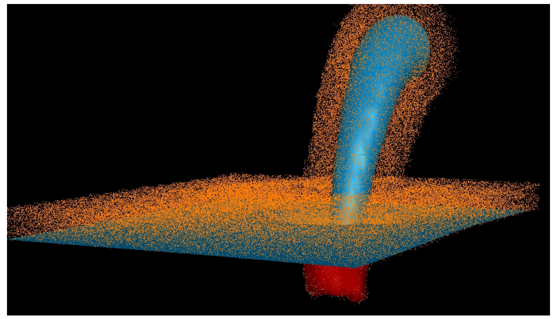


Evaluation

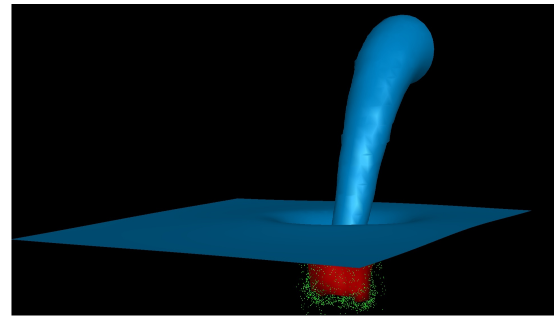
Complexities of Graphical Simulations



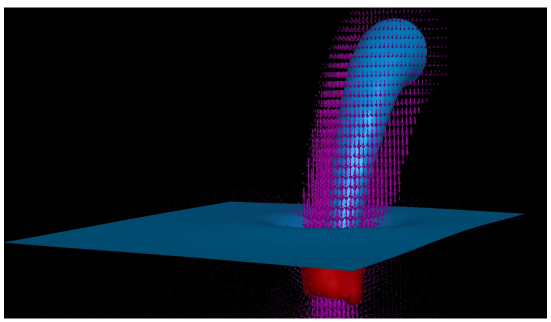
Levelset



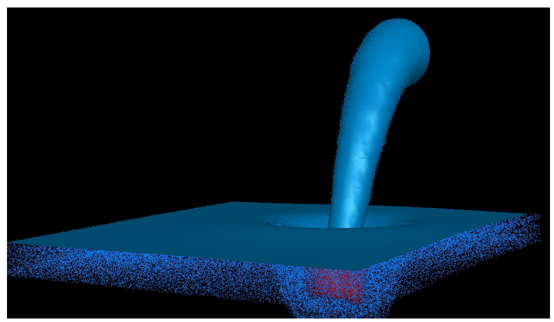
Positive Particles



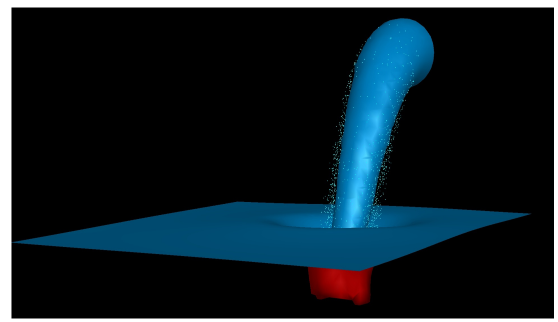
Positive Removed Particles



Velocity



Negative Particles

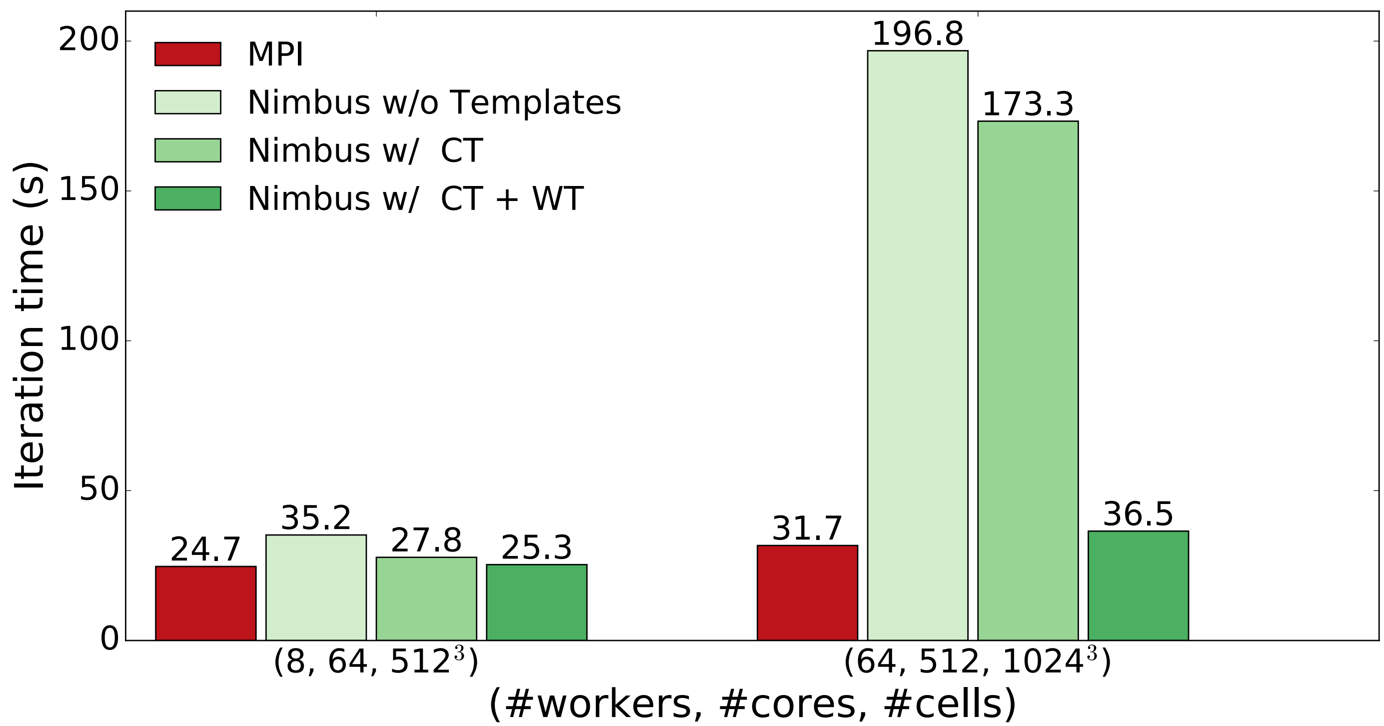


Negative Removed Particles

- 40 different variables: scalar, vector, particle.
- Triply nested loop with data dependent branches.
    - 9 different templates (basic blocks).
    - 3 branches that need patching.

Evaluation

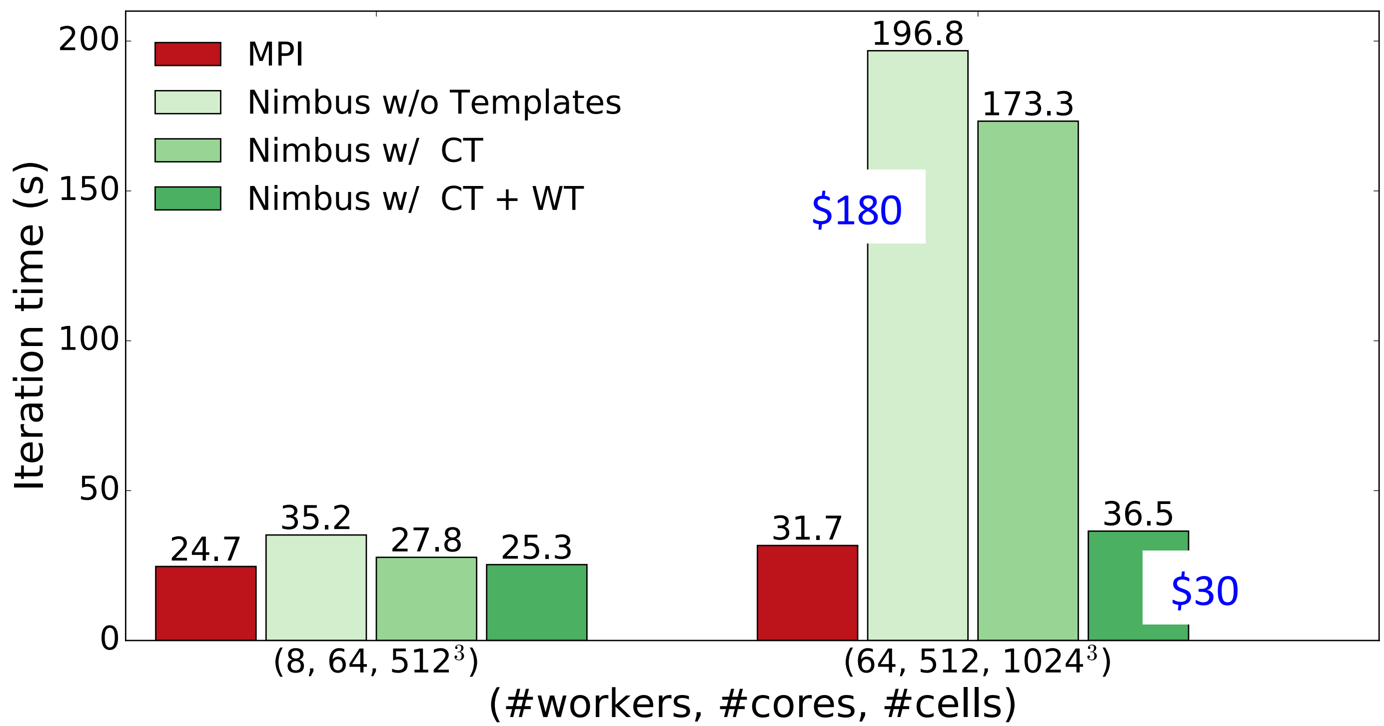
Speedup with Templates



- Canonical water simulations under Nimbus and MPI.
- Without templates, Nimbus is almost 6x slower than MPI.
- Slow down means either lower resolution or more time/money.

Evaluation

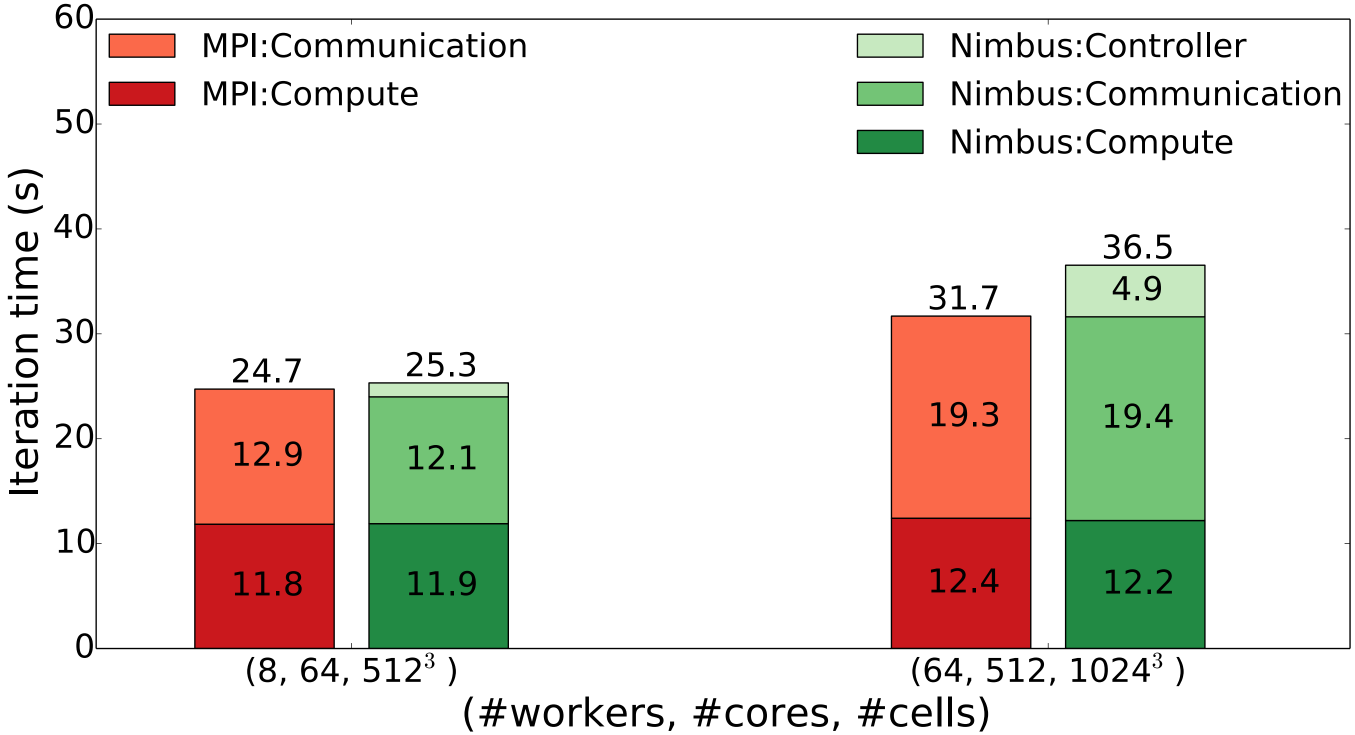
Speedup with Templates



- Canonical water simulations under Nimbus and MPI.
- Without templates, Nimbus is almost 6x slower than MPI.
- Slow down means either lower resolution or more time/money.

Evaluation

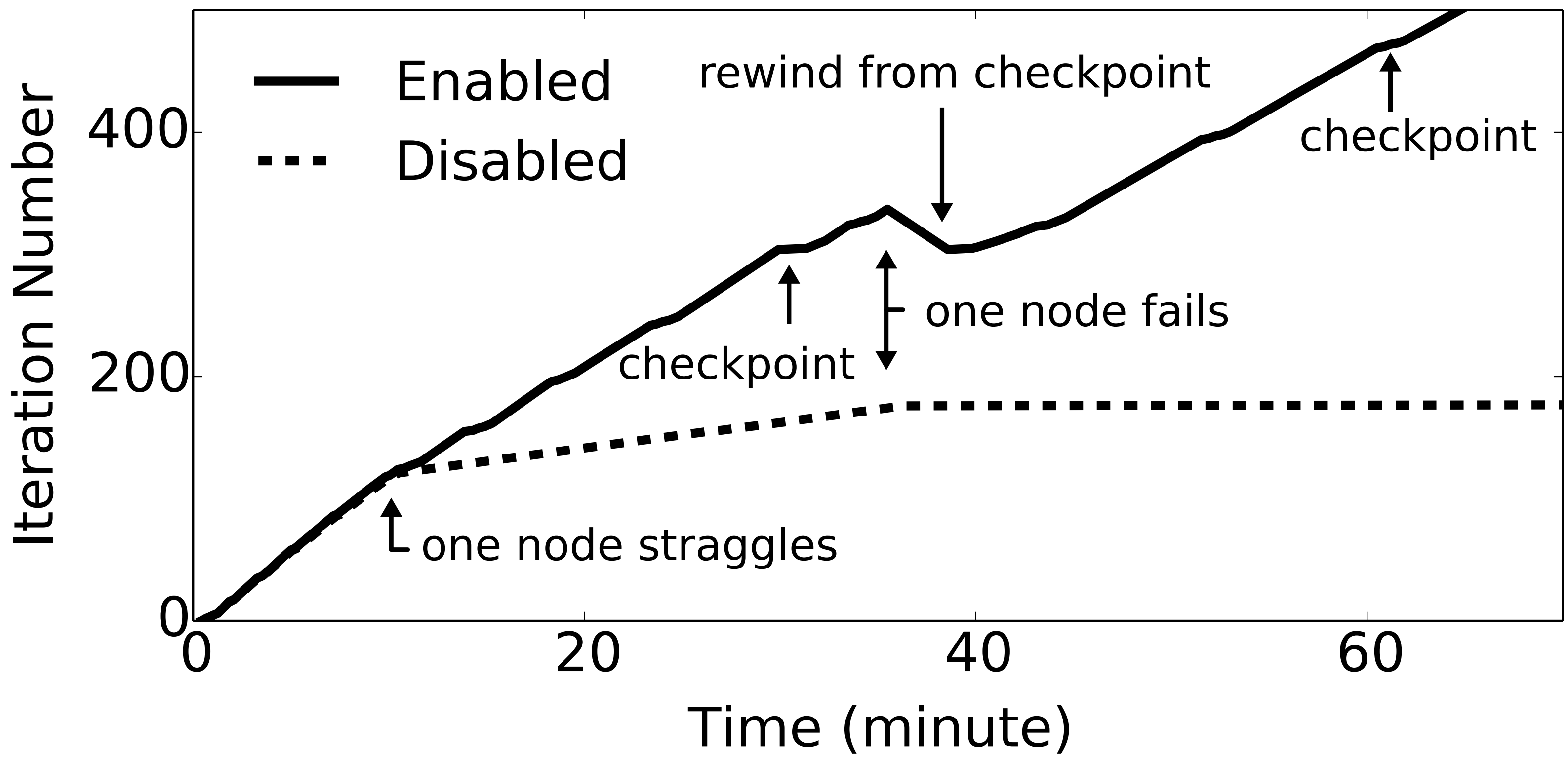
Comparison with Hand-Tuned MPI



- Canonical water simulations under Nimbus and MPI.
- Nimbus performance is within 3-15% of the hand-tuned MIP.
- At 512 cores, there are more than 1 million distinct data objects and task throughput picks at 460,000 tasks per second.

Evaluation

Load Balancing and Fault Recovery with Templates



- Nimbus controller adapts to the stragglers and worker failures.
- Templates are seamlessly installed as schedule changes.

Contributions

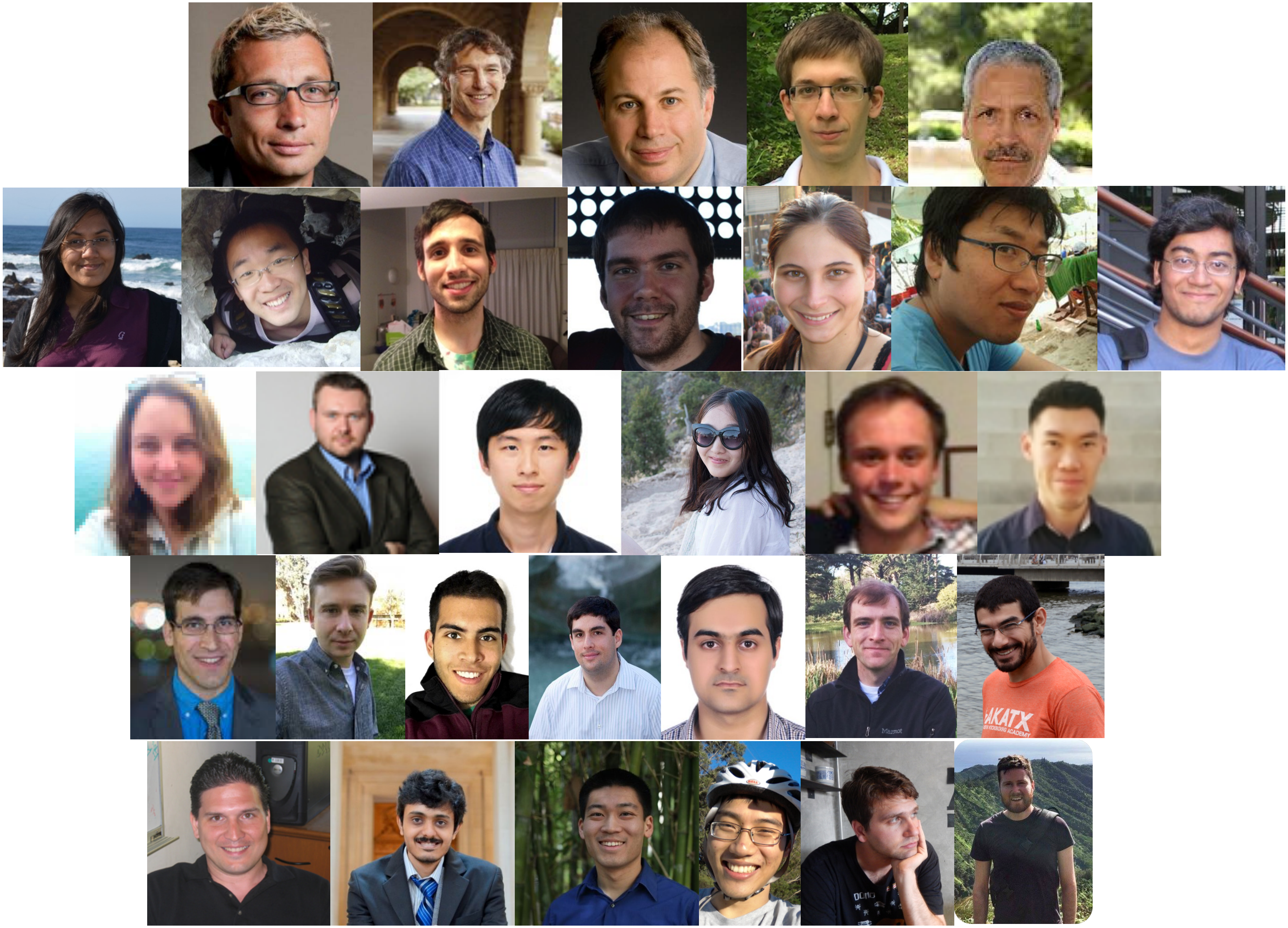
- Demonstrating how the **control plane** is the emerging **bottleneck** for data analytics frameworks.
- **Execution Templates** as an abstraction for the control plane of cloud computing frameworks, that enables orders of magnitude higher task throughput, while keeping the fine-grained, flexible scheduling.
- The design, implementation, and evaluation of **Nimbus**, a distributed cloud computing framework that embeds execution templates.
- A demonstration of a single-core **graphical simulation** that Nimbus **automatically distributes** in the cloud showing execution templates in practice for complex applications.

Conclusion

Control Plane Design	Example Framework	Task Throughput	Task Scheduling
Centralized	MapReduce Hadoop Spark	Low	Dynamic
Distributed	Naiad TensorFlow	High	Static
Centralized w/ Execution Templates	Nimbus	High	Dynamic

Thank You!

Acknowledgements



Thank You!

Backup Slides

Evaluation

Micro-benchmarks

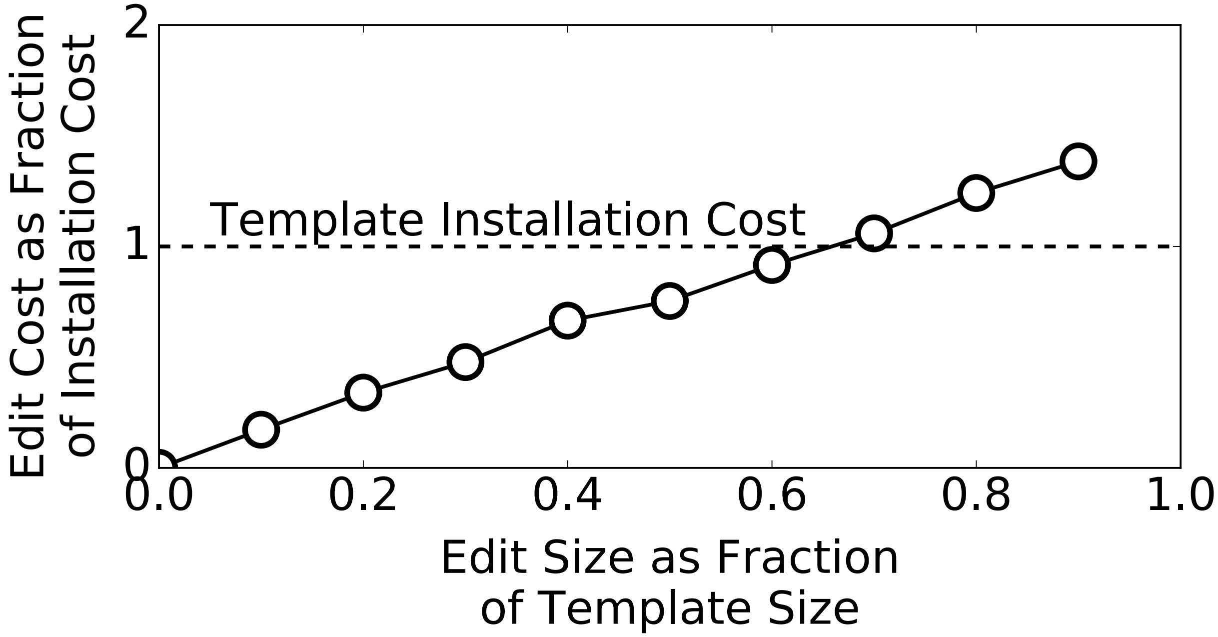
	Per-task cost
Installing controller template	25μs
Installing worker template on controller	15μs
Installing worker template on worker	9μs
Nimbus schedule task	134μs
Spark schedule task	166μs

	Per-task cost
Instantiate controller template	$0.2\mu s$
Instantiate worker template (auto-validation)	$1.7\mu s$
Instantiate worker template (validation)	$7.3\mu s$

	Cost
Nimbus single edit	$\approx 41\mu s$
Nimbus 5% task migration (800 edits)	$35ms$
Nimbus complete installation (8000 tasks)	$203ms$
Naiad any change	$230ms$

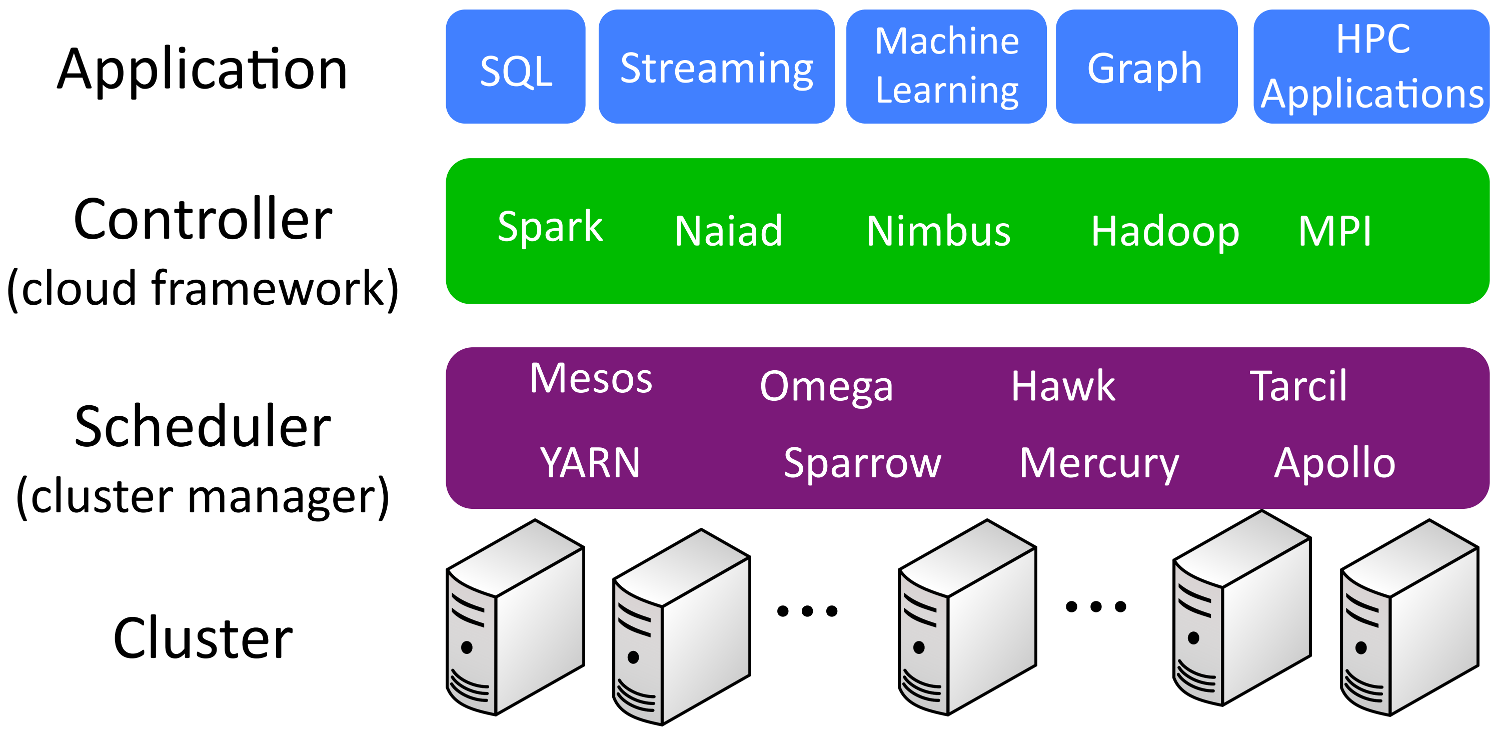
Evaluation

Low Edit Cost



- Single edit overhead is only 41us (in average).
- Edit cost increases linearly with the extent of changes.
- For extensive changes it is better to install new templates.

Schedule Plane vs. Control Plane



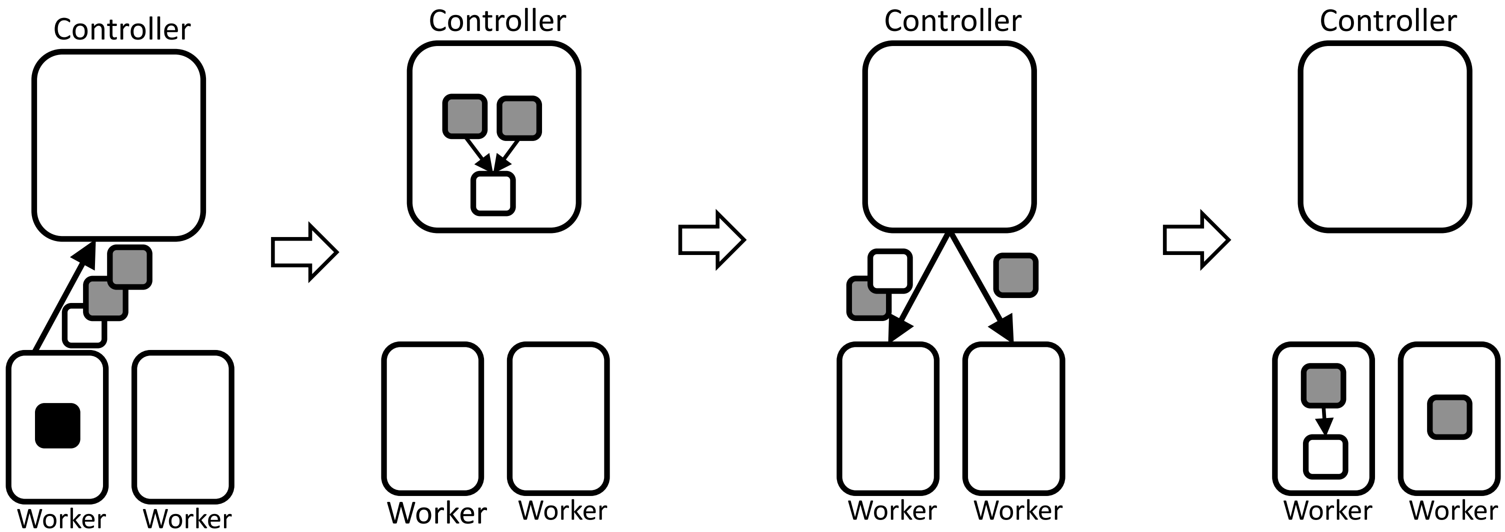
Nimbus

Applications

- Nimbus support traditional data analytics:
  - supervised/unsupervised learning, graph library.
- To show the generality of execution templates, we considered graphical simulations in Nimbus:
  - Complex, and memory intensive from PhysBAM library.
  - High tasks throughput requirements (400,000 tasks per second).
  - Nested loops and data dependent branches.
  - Require patching in very subtle cases.

Nimbus

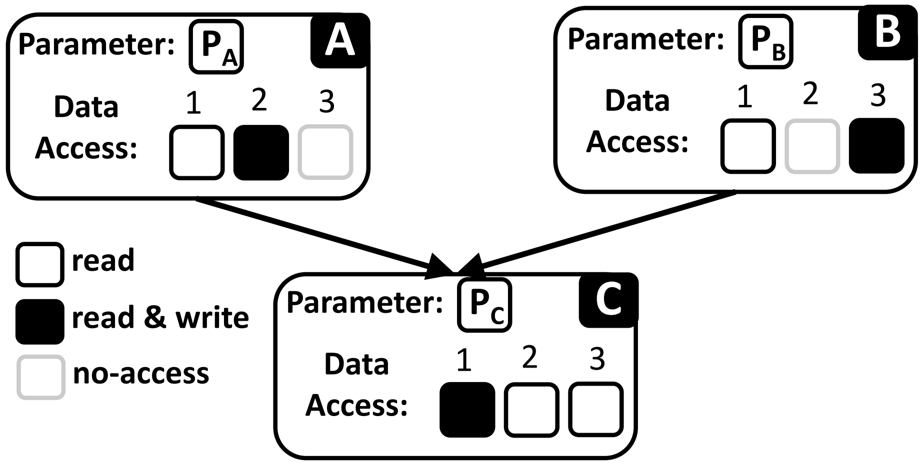
Task Spawning Control Flow



- Control flow:
  - Tasks that execute on the workers spawn other tasks.
  - Driver program is a lineage of tasks executing on the workers.

Nimbus

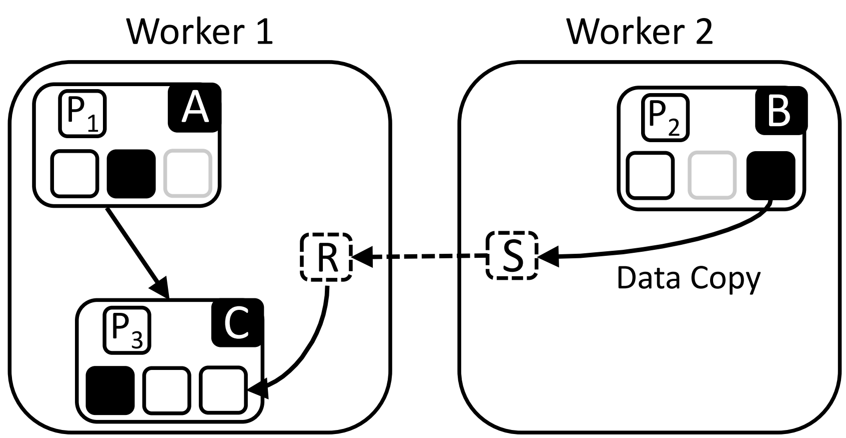
Data Model



- Mutable data model:
  - In-place operations are crucial for memory intense applications (e.g. graphical simulations).
  - Data identifiers could be part of the static segment of the template.

Nimbus

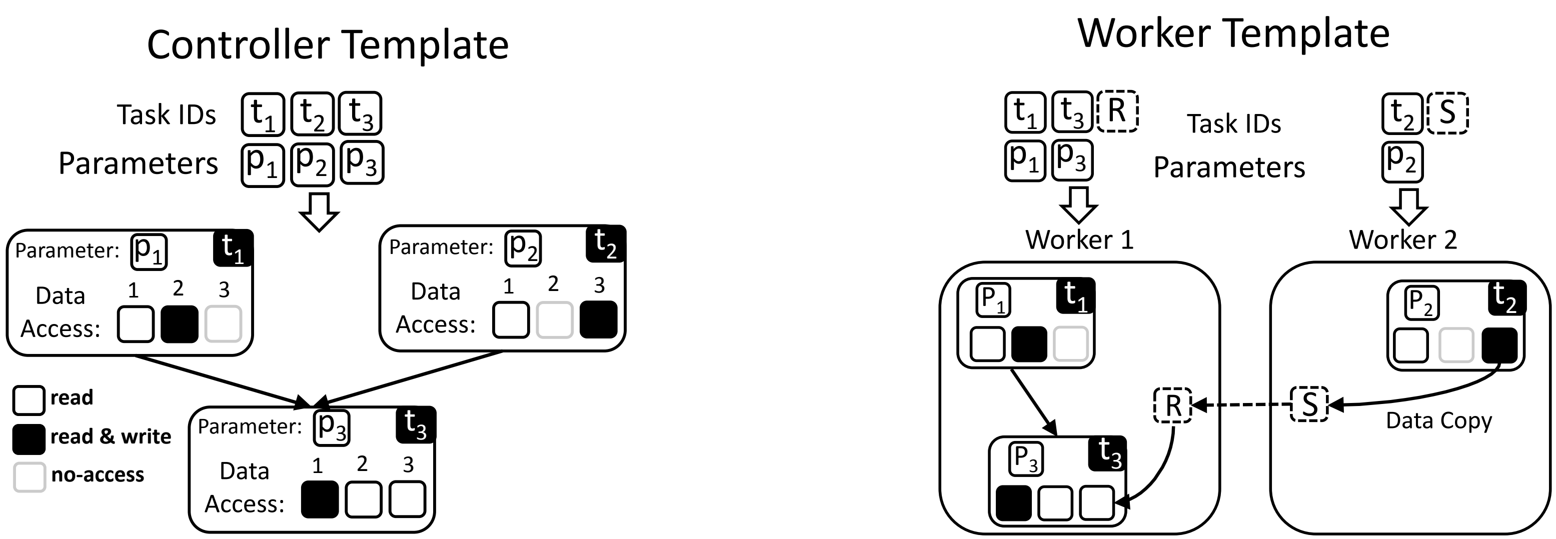
Execution Model



- Explicit dependencies:
  - Task dependency set: event based model.
  - Explicit data exchange for inter worker dependencies.
  - Worker can queue tasks and order them locally.

Nimbus

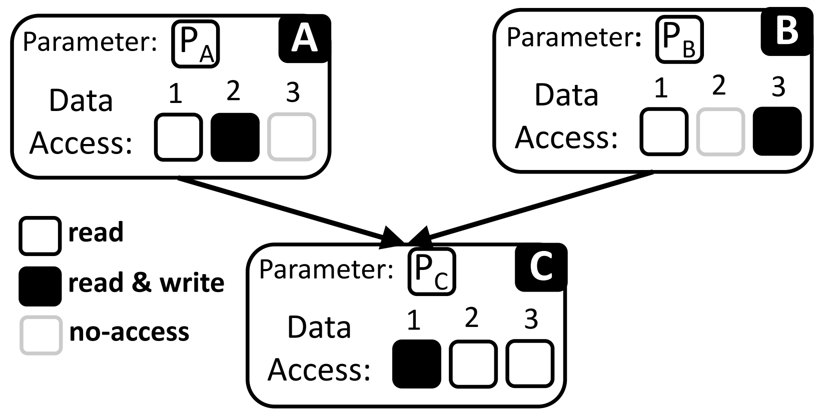
Controller and Worker Templates

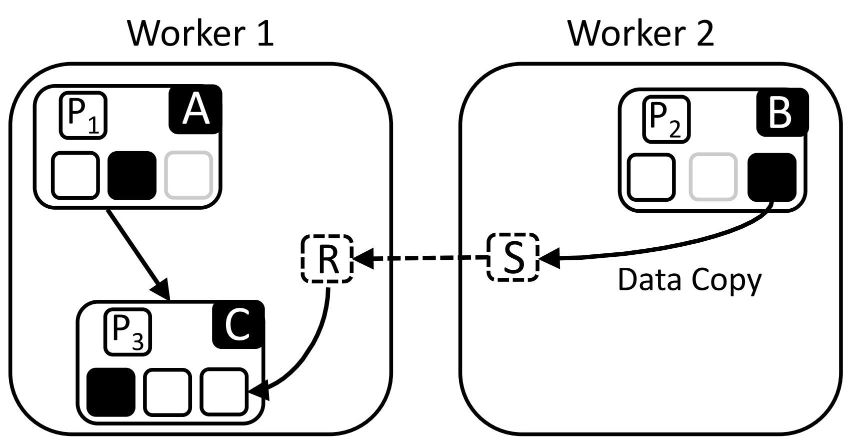


- Separate scheduling and control flow:
  - Controller template: instantiates entire task graph.
  - Worker Template: instantiates per worker execution plan.

Nimbus

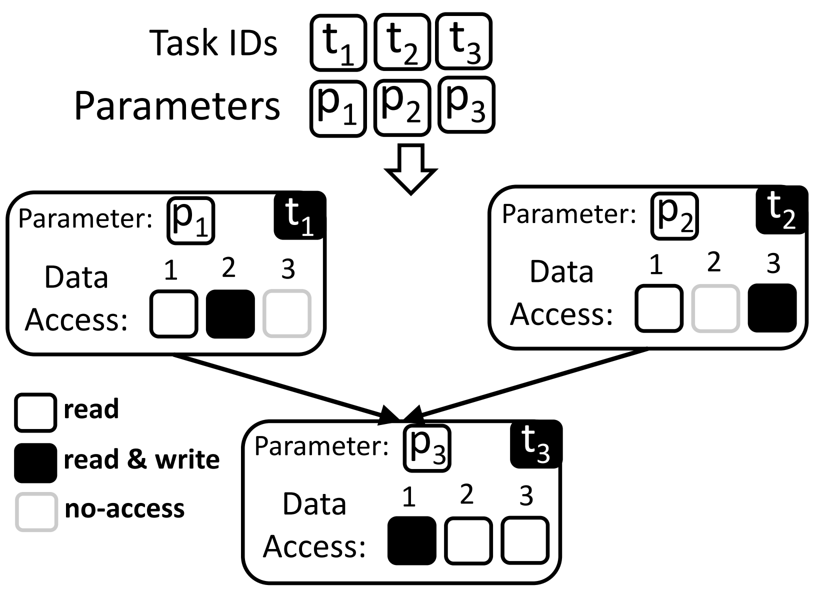
Execution Model





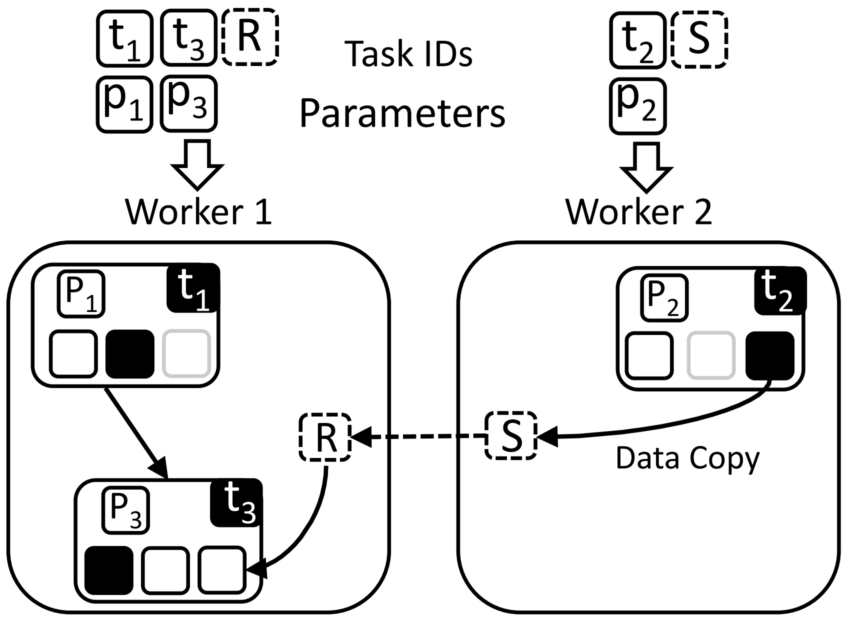
Nimbus

Controller Templates



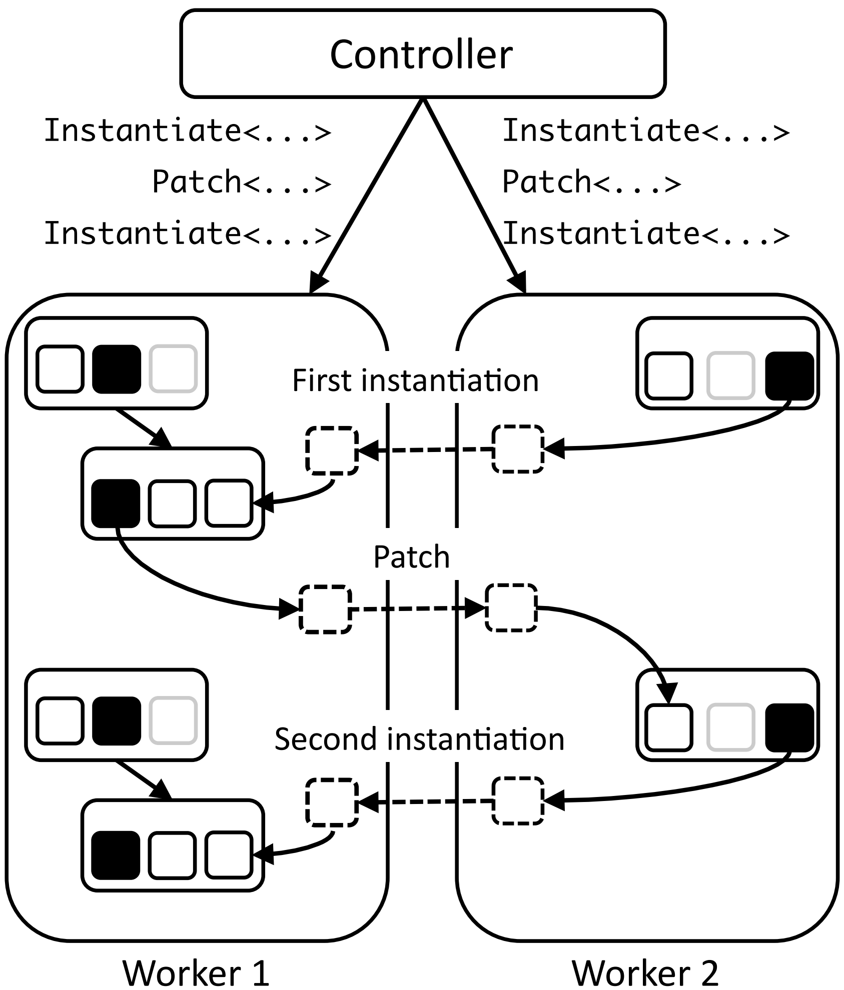
Nimbus

Worker Templates



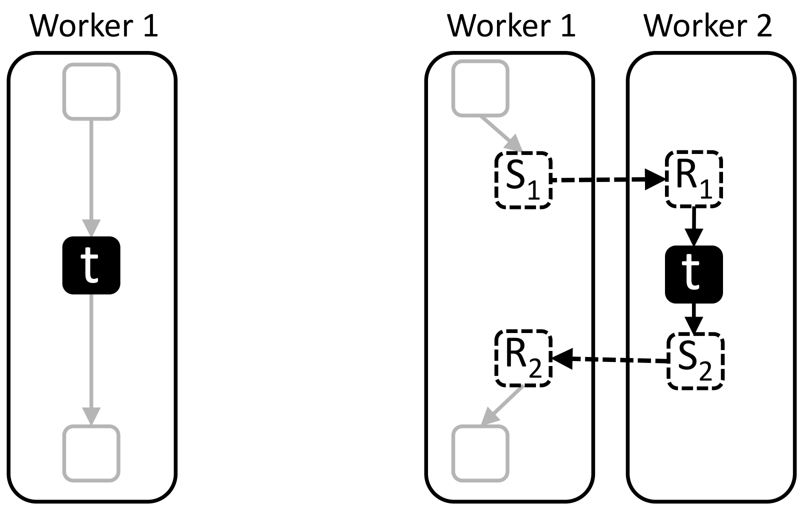
Nimbus

Patches



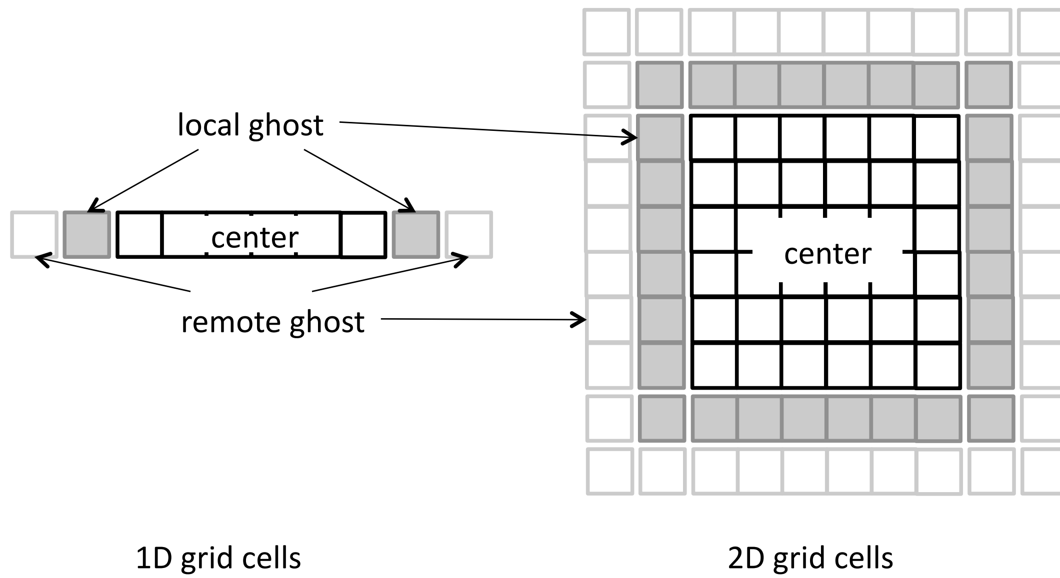
Nimbus

Edits



Nimbus

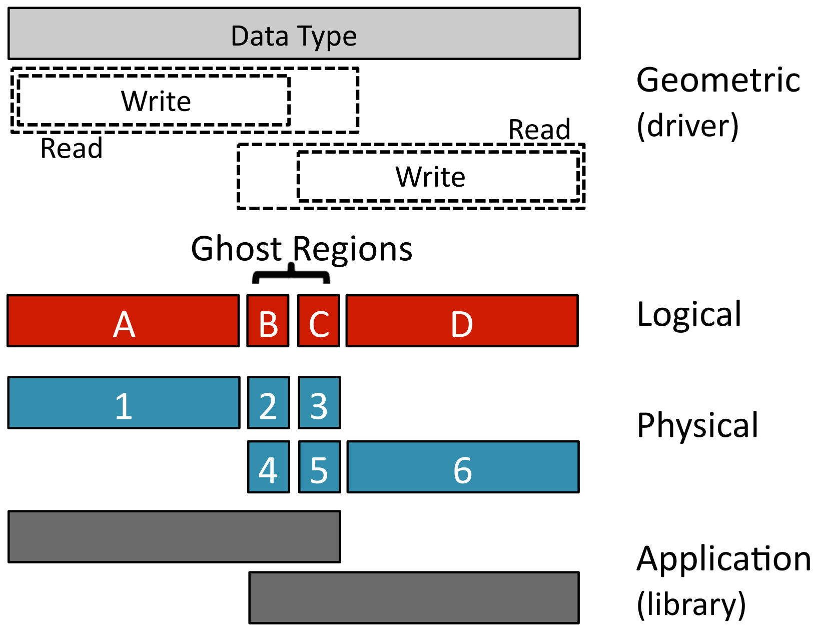
Graphical Simulations



- The notion of ghost value is different than traditional analytics data model.
  - They have geometric locality.
  - Could be captured by **GroupBy** operations, but very inefficiently.

Nimbus

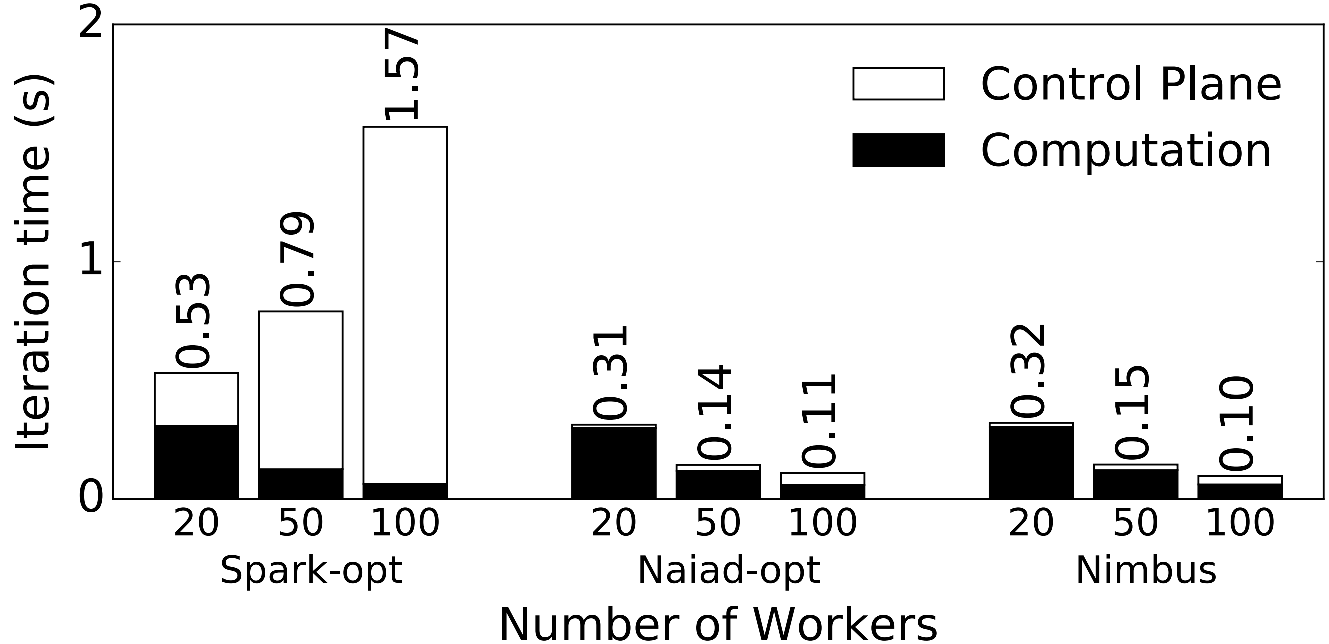
Graphical Simulations



- Geometric based driver program.
- Contiguous data for the library kernels.
- Automatic translation and synchronization by Nimbus

Evaluation

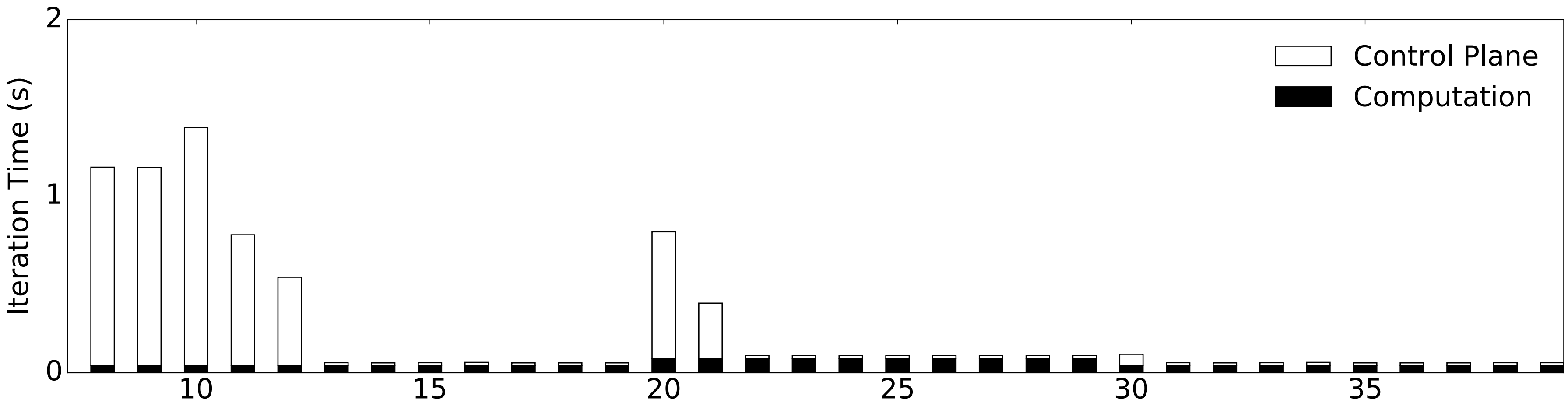
Data Analytics (K-means)



- K-means over data set of size 100GB.
- Spark-opt and Naiad-opt, runs tasks as fast as C++ implementation.
- Nimbus centralized controller with execution templates matches the performance of Naiad with a distributed control plane.

Evaluation

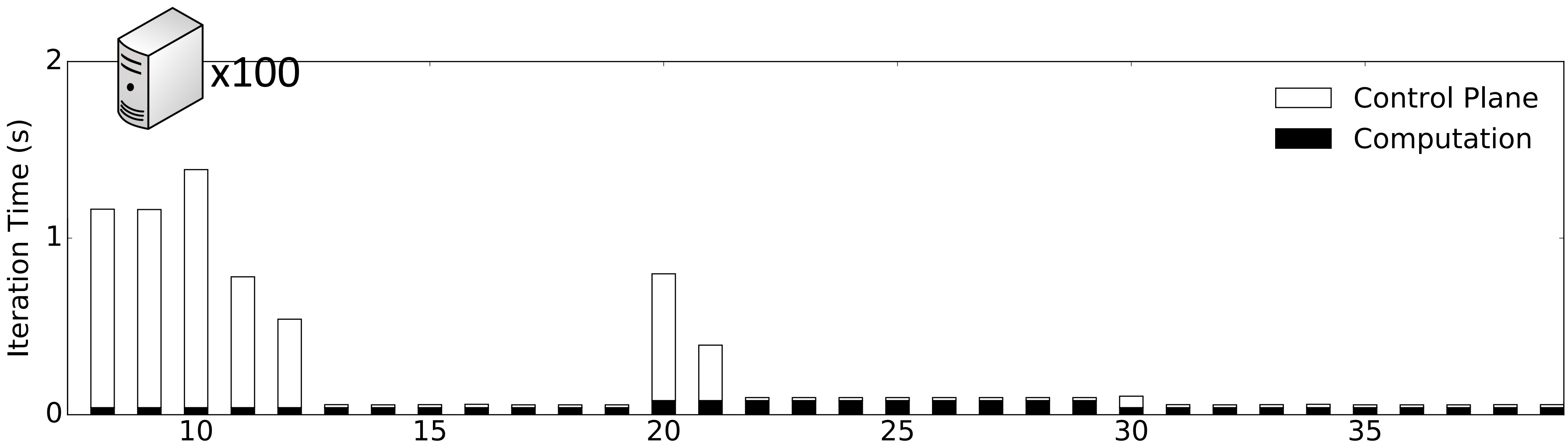
Dynamic Scheduling



- Logistic regression over 100GB of data, on 50/100 workers.
- One-time template installation cost is <40% of task scheduling.
- Nimbus allows dynamic resource allocation.
- Nimbus installs multiple versions of a template depending on resources.

Evaluation

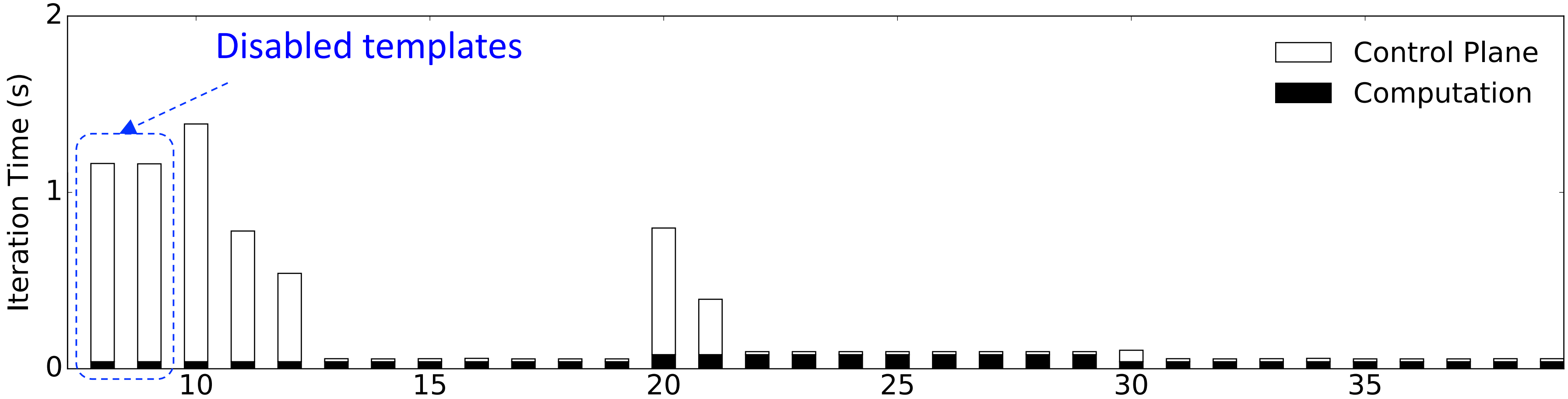
Dynamic Scheduling



- Logistic regression over 100GB of data, on 50/100 workers.
- One-time template installation cost is <40% of task scheduling.
- Nimbus allows dynamic resource allocation.
- Nimbus installs multiple versions of a template depending on resources.

Evaluation

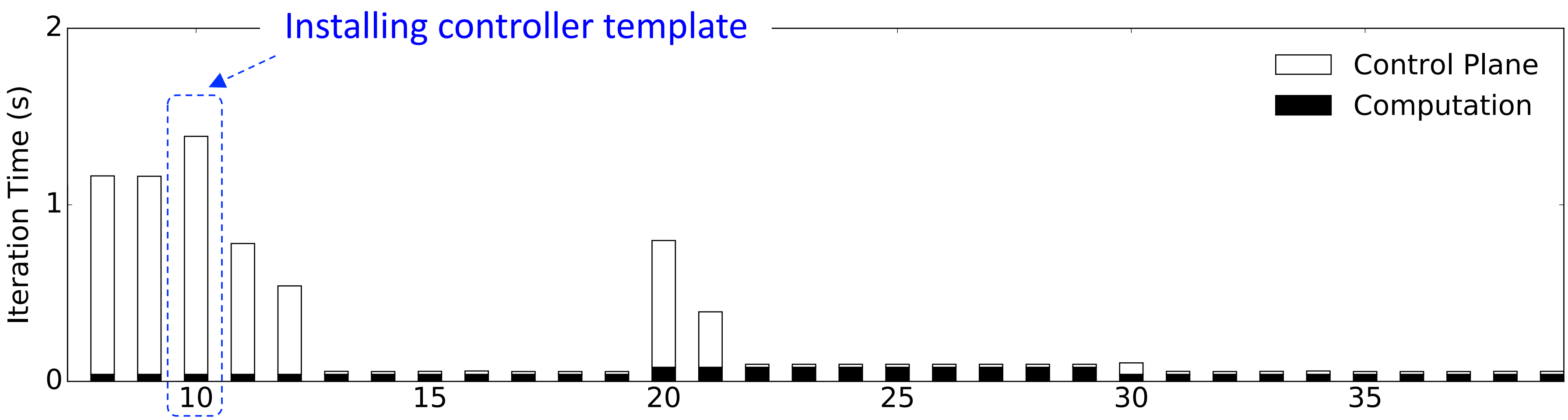
Dynamic Scheduling



- Logistic regression over 100GB of data, on 50/100 workers.
- One-time template installation cost is <40% of task scheduling.
- Nimbus allows dynamic resource allocation.
- Nimbus installs multiple versions of a template depending on resources.

Evaluation

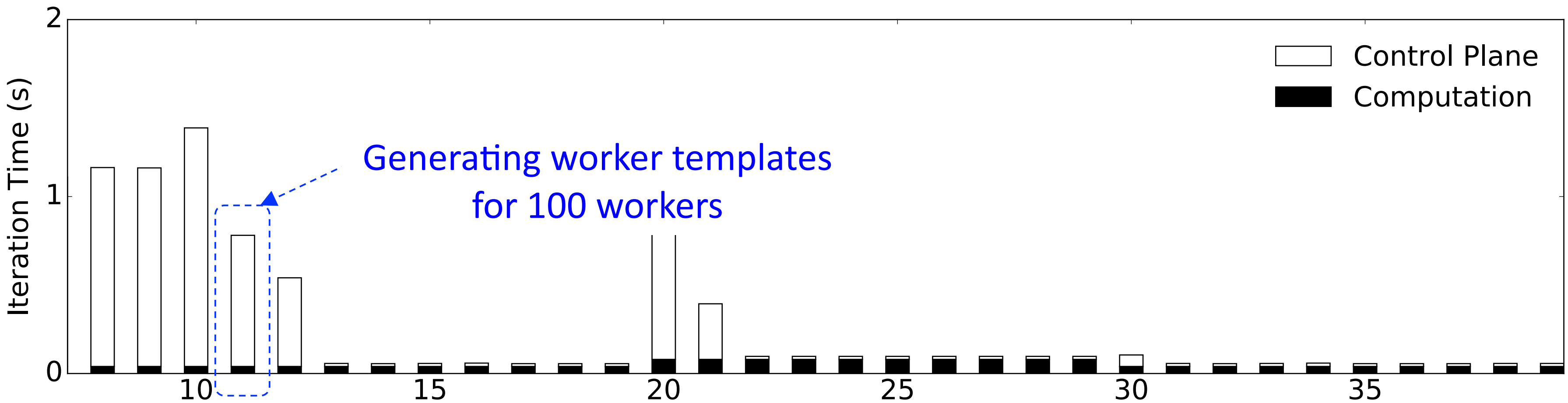
Dynamic Scheduling



- Logistic regression over 100GB of data, on 50/100 workers.
- One-time template installation cost is <40% of task scheduling.
- Nimbus allows dynamic resource allocation.
- Nimbus installs multiple versions of a template depending on resources.

Evaluation

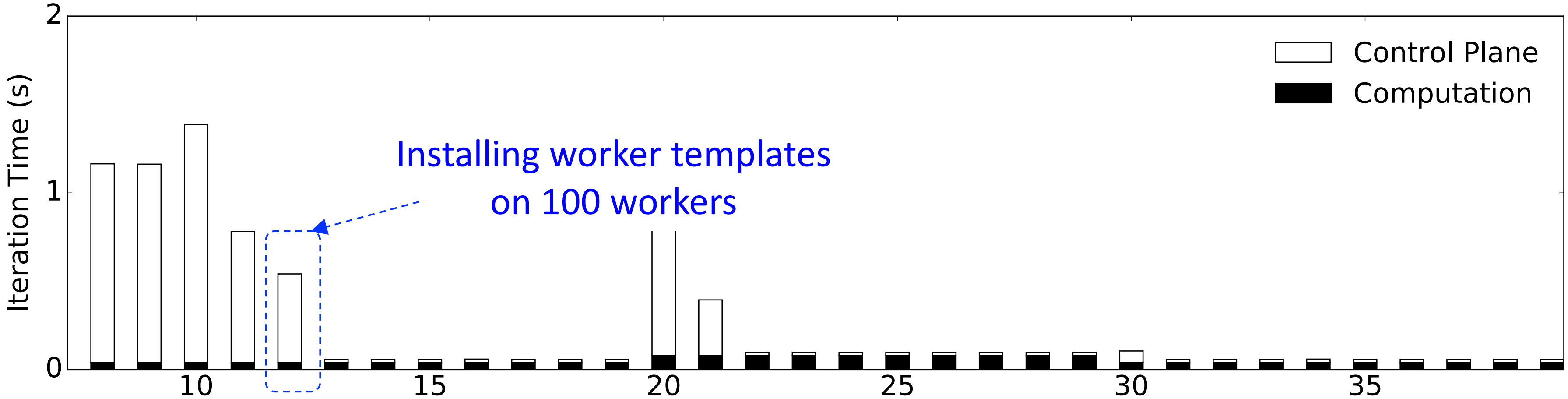
Dynamic Scheduling



- Logistic regression over 100GB of data, on 50/100 workers.
- One-time template installation cost is <40% of task scheduling.
- Nimbus allows dynamic resource allocation.
- Nimbus installs multiple versions of a template depending on resources.

Evaluation

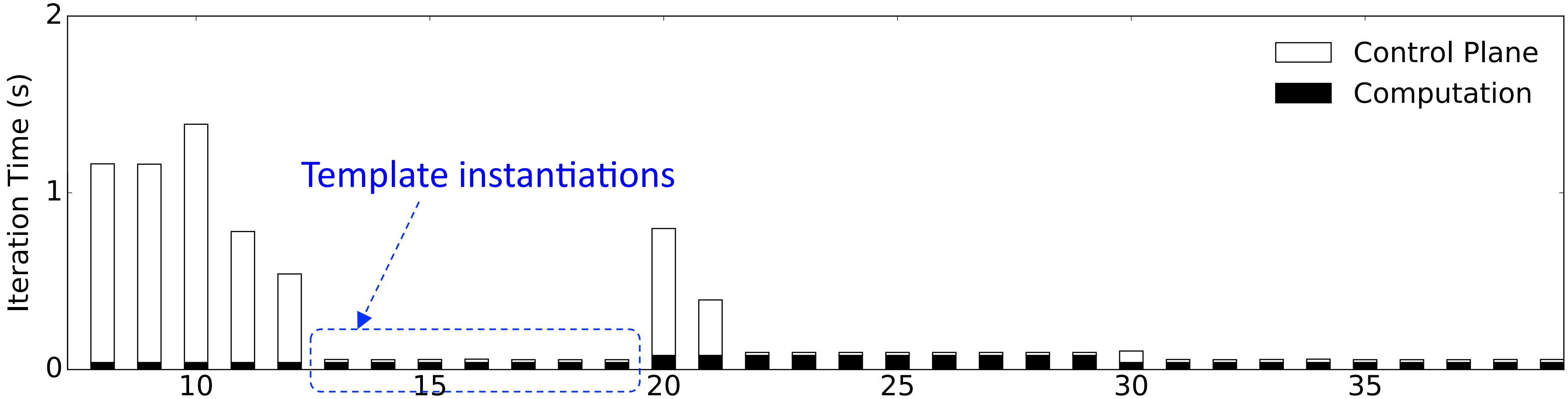
Dynamic Scheduling



- Logistic regression over 100GB of data, on 50/100 workers.
- One-time template installation cost is <40% of task scheduling.
- Nimbus allows dynamic resource allocation.
- Nimbus installs multiple versions of a template depending on resources.

Evaluation

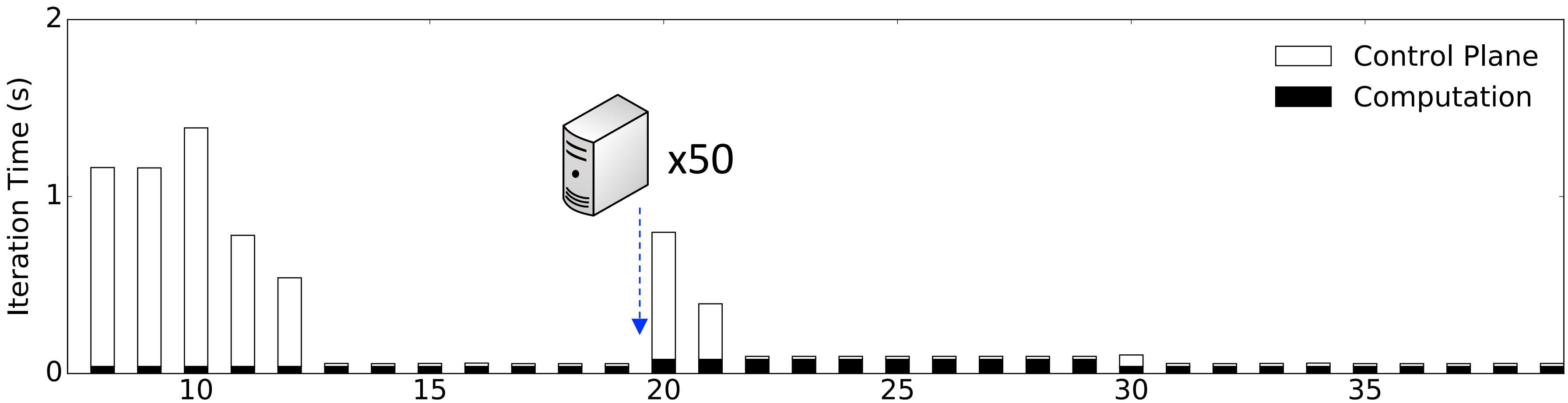
Dynamic Scheduling



- Logistic regression over 100GB of data, on 50/100 workers.
- One-time template installation cost is <40% of task scheduling.
- Nimbus allows dynamic resource allocation.
- Nimbus installs multiple versions of a template depending on resources.

Evaluation

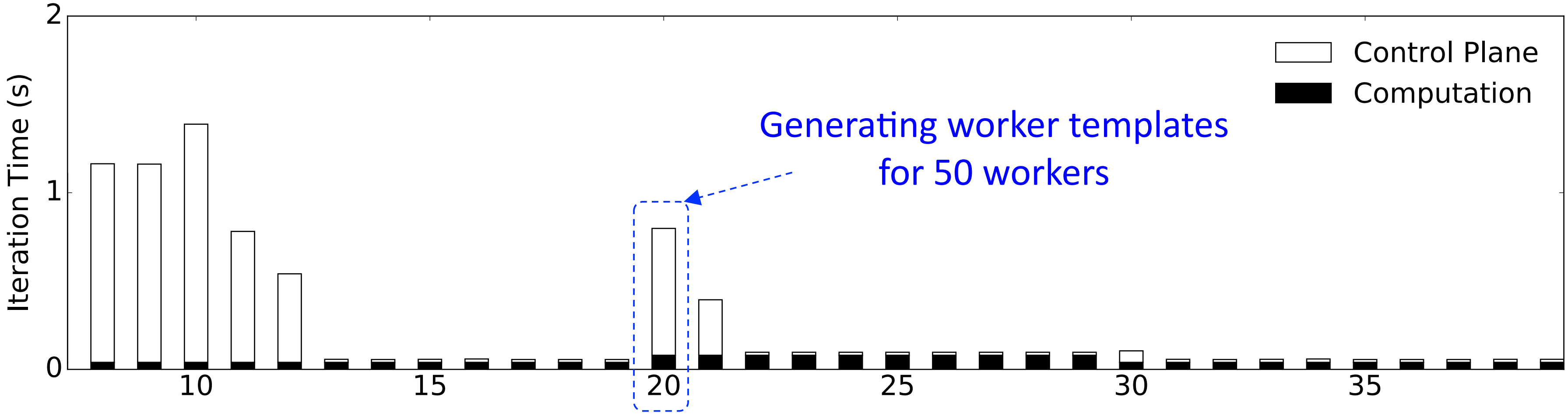
Dynamic Scheduling



- Logistic regression over 100GB of data, on 50/100 workers.
- One-time template installation cost is <40% of task scheduling.
- Nimbus allows dynamic resource allocation.
- Nimbus installs multiple versions of a template depending on resources.

Evaluation

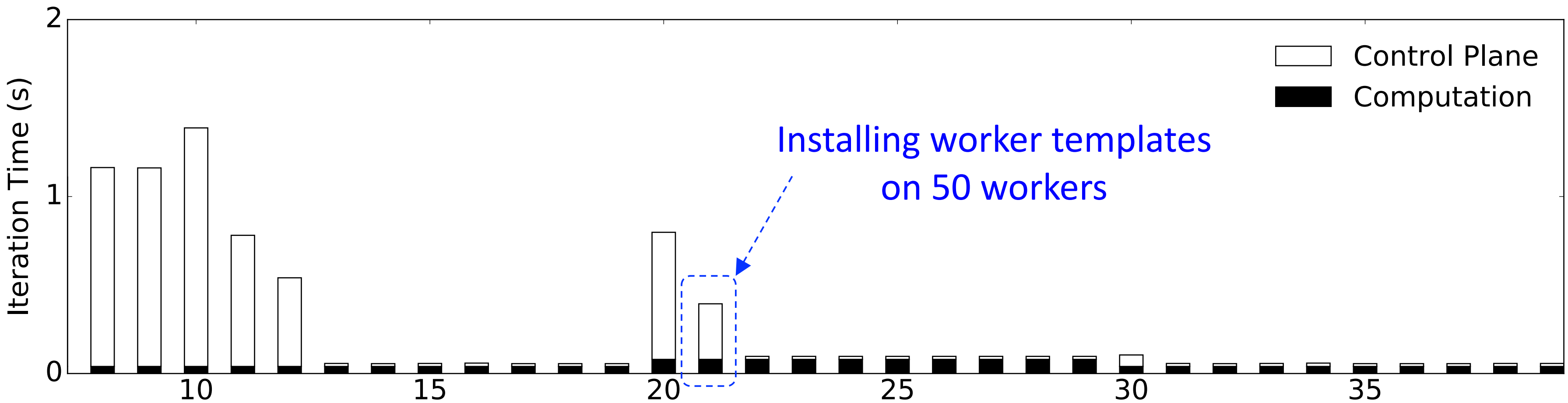
Dynamic Scheduling



- Logistic regression over 100GB of data, on 50/100 workers.
- One-time template installation cost is <40% of task scheduling.
- Nimbus allows dynamic resource allocation.
- Nimbus installs multiple versions of a template depending on resources.

Evaluation

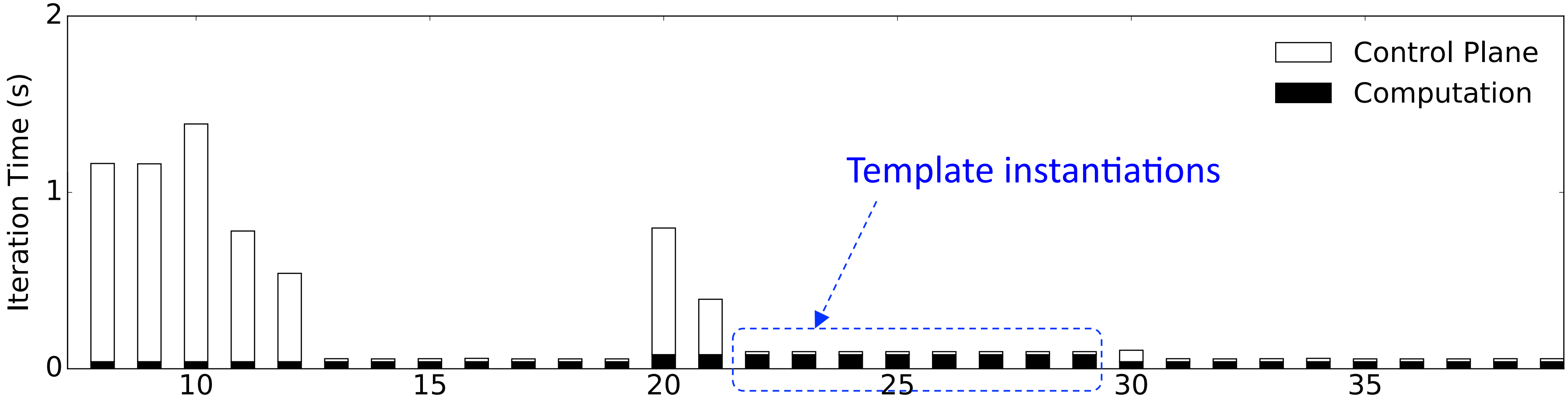
Dynamic Scheduling



- Logistic regression over 100GB of data, on 50/100 workers.
- One-time template installation cost is <40% of task scheduling.
- Nimbus allows dynamic resource allocation.
- Nimbus installs multiple versions of a template depending on resources.

Evaluation

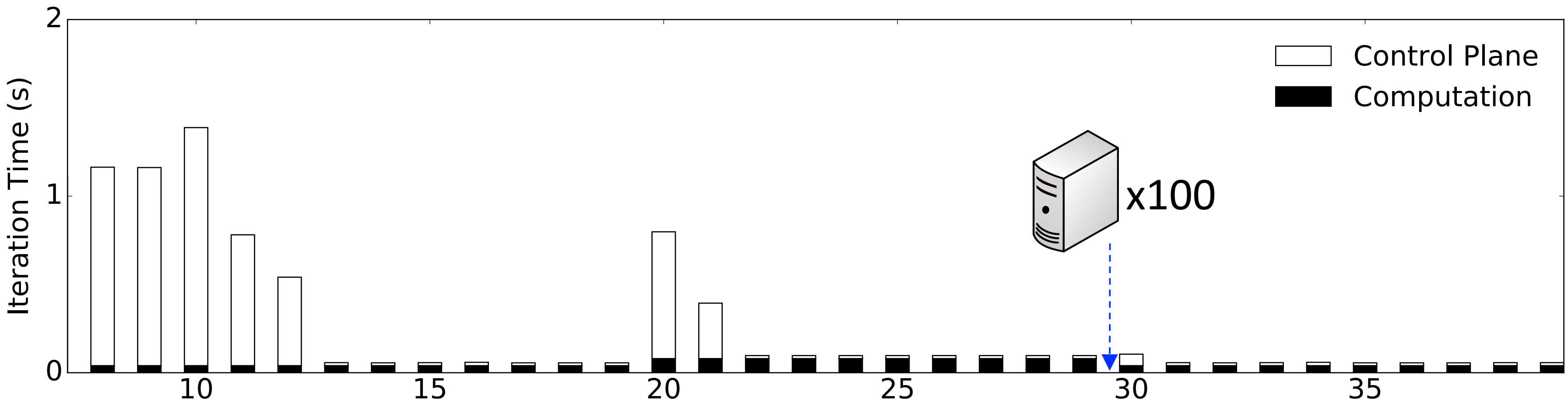
Dynamic Scheduling



- Logistic regression over 100GB of data, on 50/100 workers.
- One-time template installation cost is <40% of task scheduling.
- Nimbus allows dynamic resource allocation.
- Nimbus installs multiple versions of a template depending on resources.

Evaluation

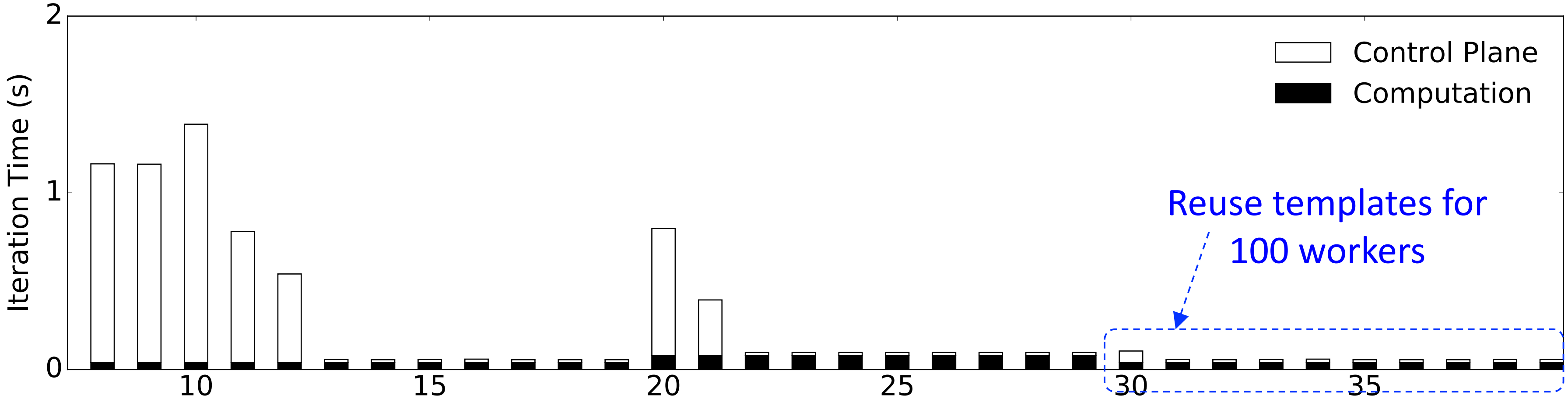
Dynamic Scheduling



- Logistic regression over 100GB of data, on 50/100 workers.
- One-time template installation cost is <40% of task scheduling.
- Nimbus allows dynamic resource allocation.
- Nimbus installs multiple versions of a template depending on resources.

Evaluation

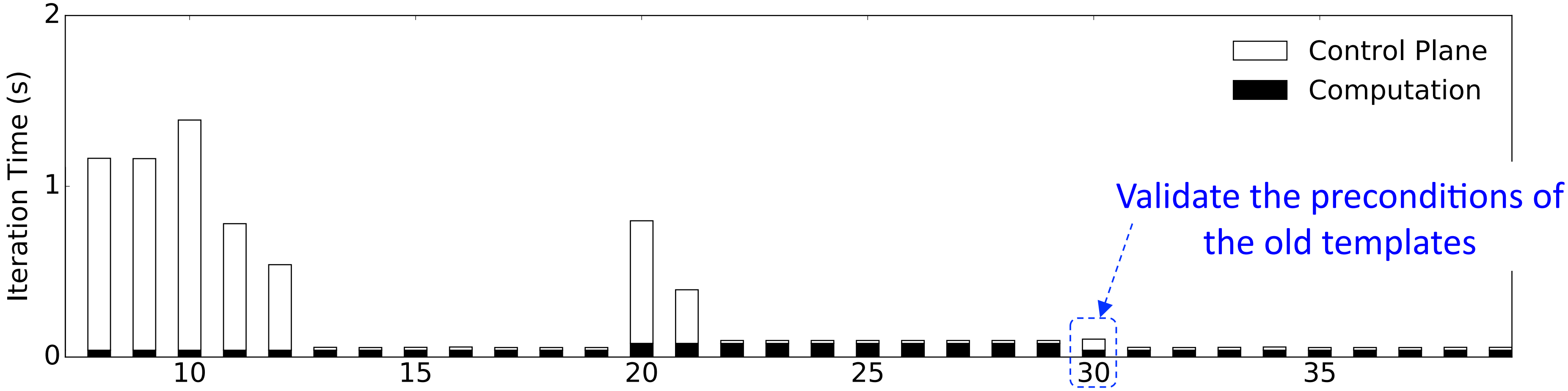
Dynamic Scheduling



- Logistic regression over 100GB of data, on 50/100 workers.
- One-time template installation cost is <40% of task scheduling.
- Nimbus allows dynamic resource allocation.
- Nimbus installs multiple versions of a template depending on resources.

Evaluation

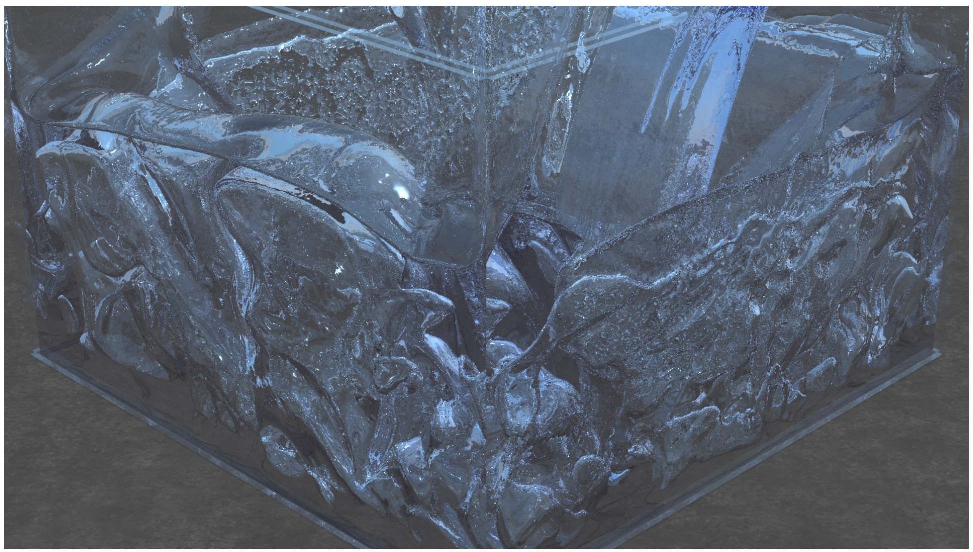
Dynamic Scheduling

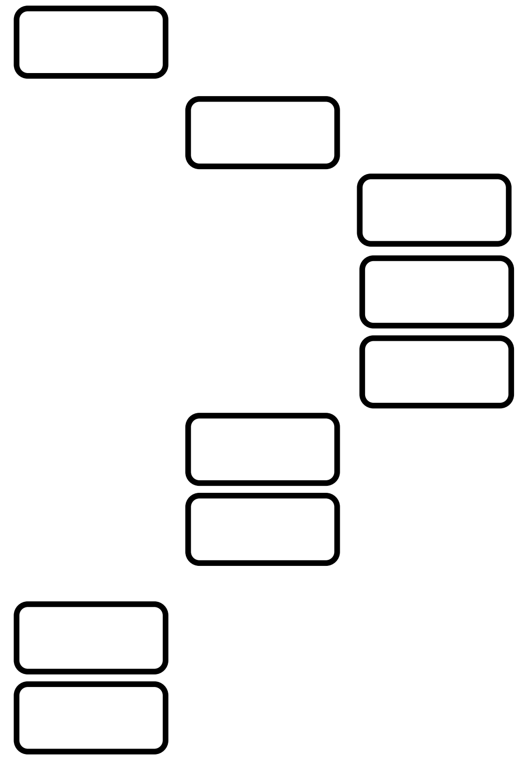


- Logistic regression over 100GB of data, on 50/100 workers.
- One-time template installation cost is <40% of task scheduling.
- Nimbus allows dynamic resource allocation.
- Nimbus installs multiple versions of a template depending on resources.

Evaluation

Dynamic Control Flow and Patching with Templates





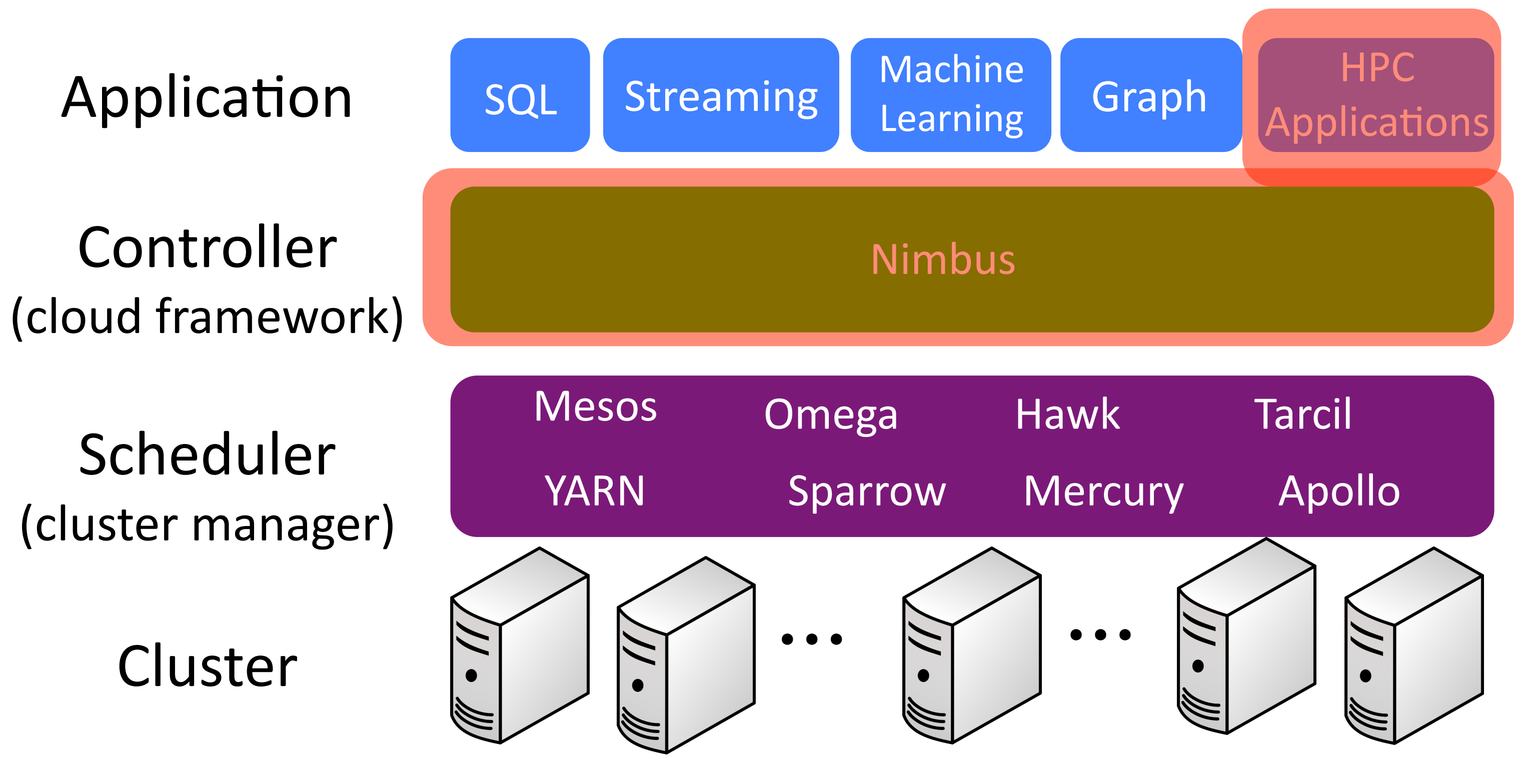
- Triply nested loop with data dependent branches.
- 9 different templates (basic blocks).
- 3 branches that need patching.

Old Slides

Scaling Data Analytics

- Logistic regression over a data set of size 100GB in Spark 2.0 MLlib.
- Scaling hurts the overall job completion time.
- Control Plane becomes a **bottleneck**, generating and spawning tasks.

Schedule Plane vs. Control Plane



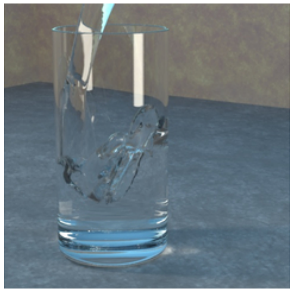
Cloud Frameworks

SQL | Streaming | Machine Learning | Graph | HPC Applications

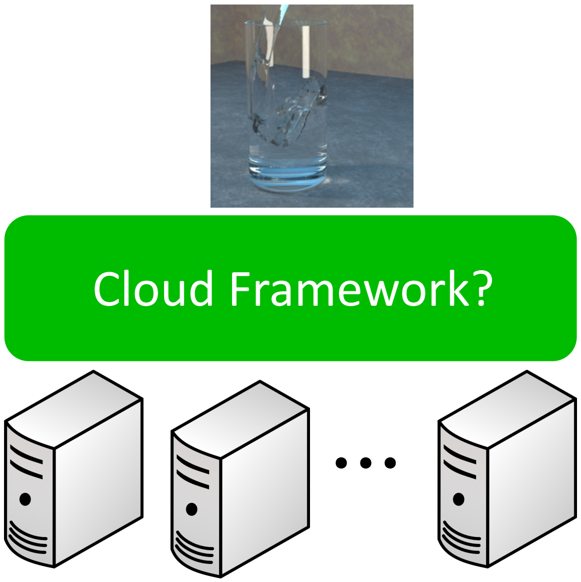
Cloud Framework

Research Question:

Could you run applications traditionally classified in the HPC domain, within a cloud framework?

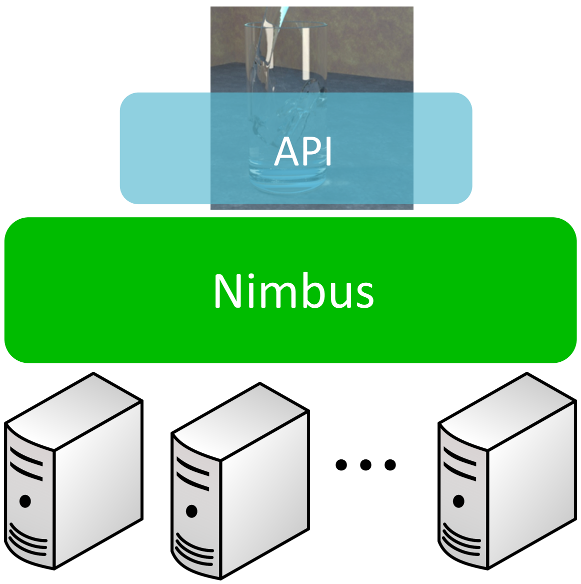


Graphical Simulations



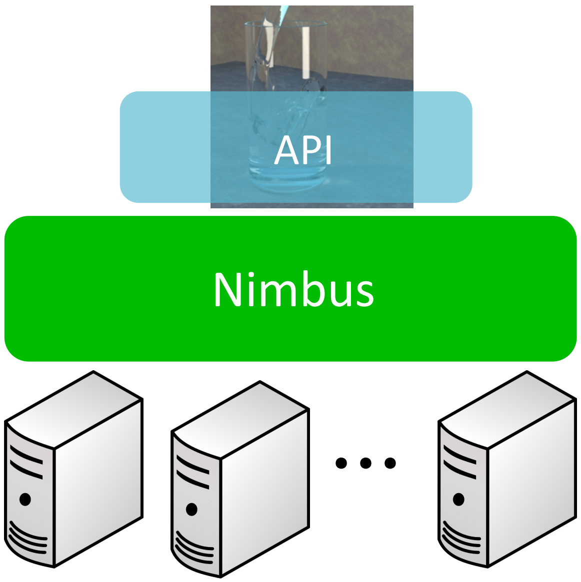
- In-place computations (mutable data).
  - Cannot afford multiple placeholders.
- Geometric locality and dependency.
  - Cannot afford GroupBy operations.

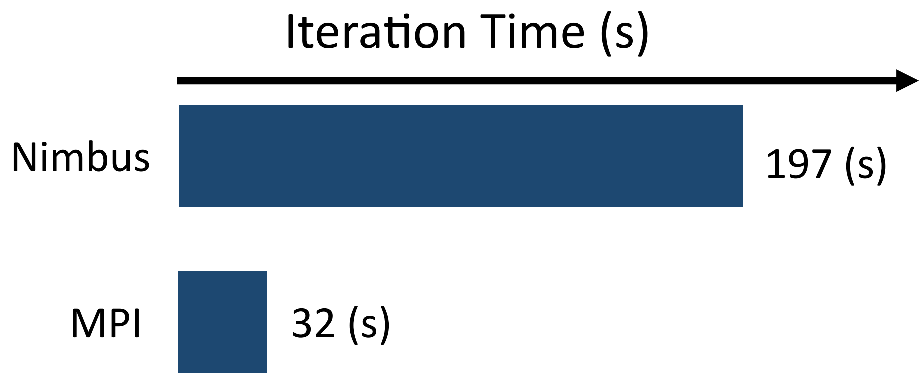
Graphical Simulations



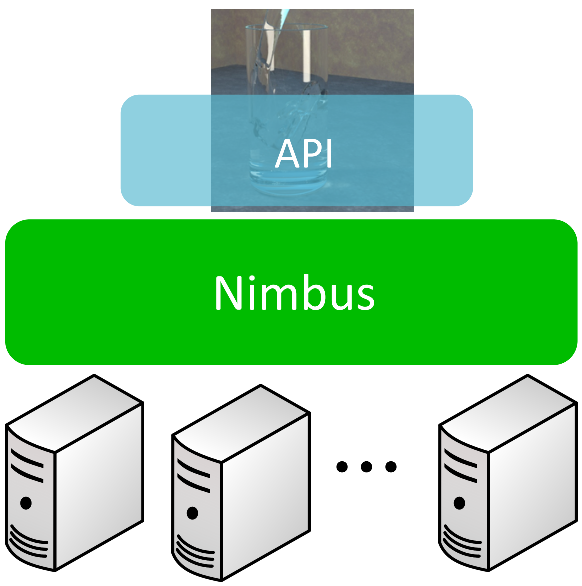
- Explicit control flow (task graph).
- Explicit data dependency (lineage)
- Preserved geometric locality.
- Application cache layer.

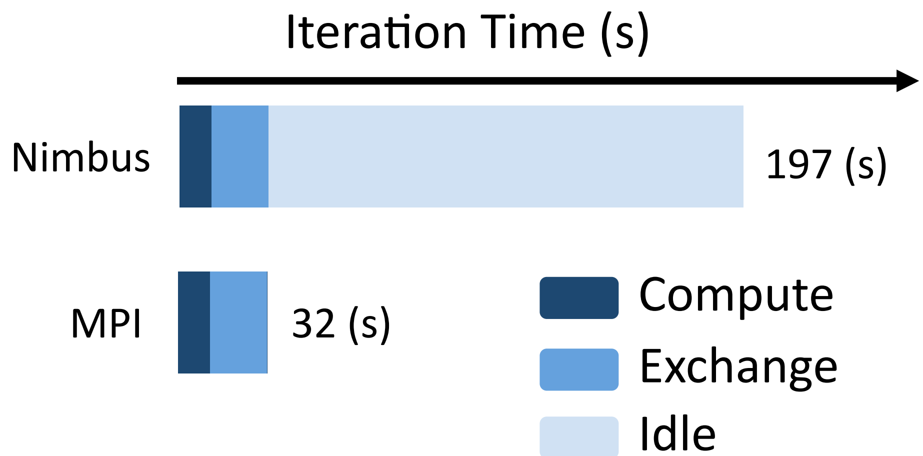
Graphical Simulations



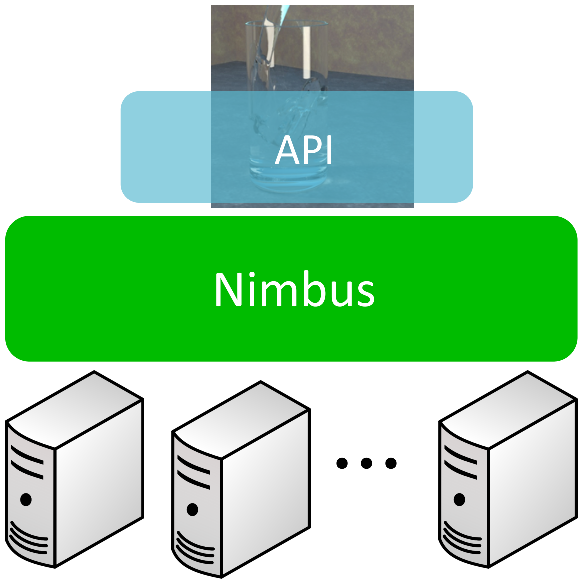


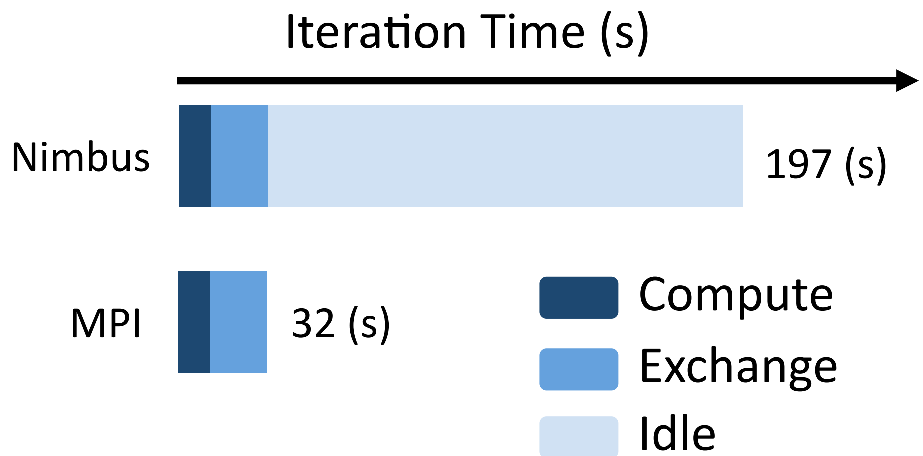
Graphical Simulations

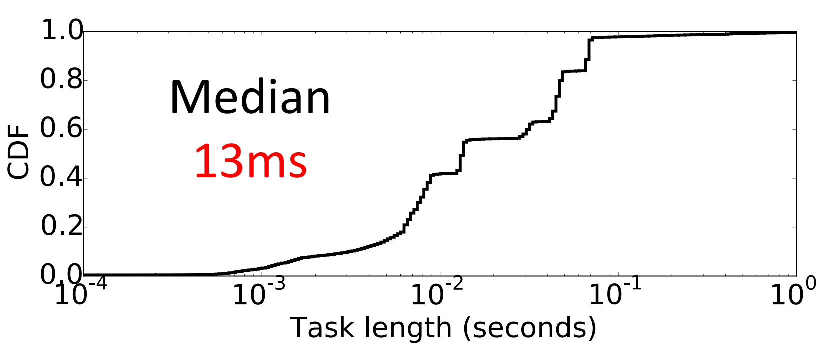




Graphical Simulations







Task rates as high as
~460,000 tasks/sec

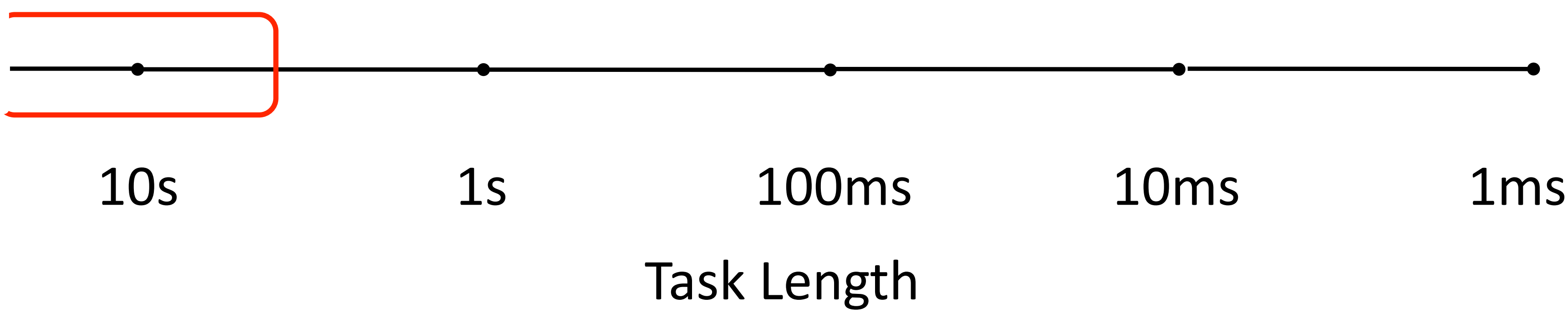
Available Frameworks could handle
only few thousands tasks/sec

Evolution of Cloud Frameworks

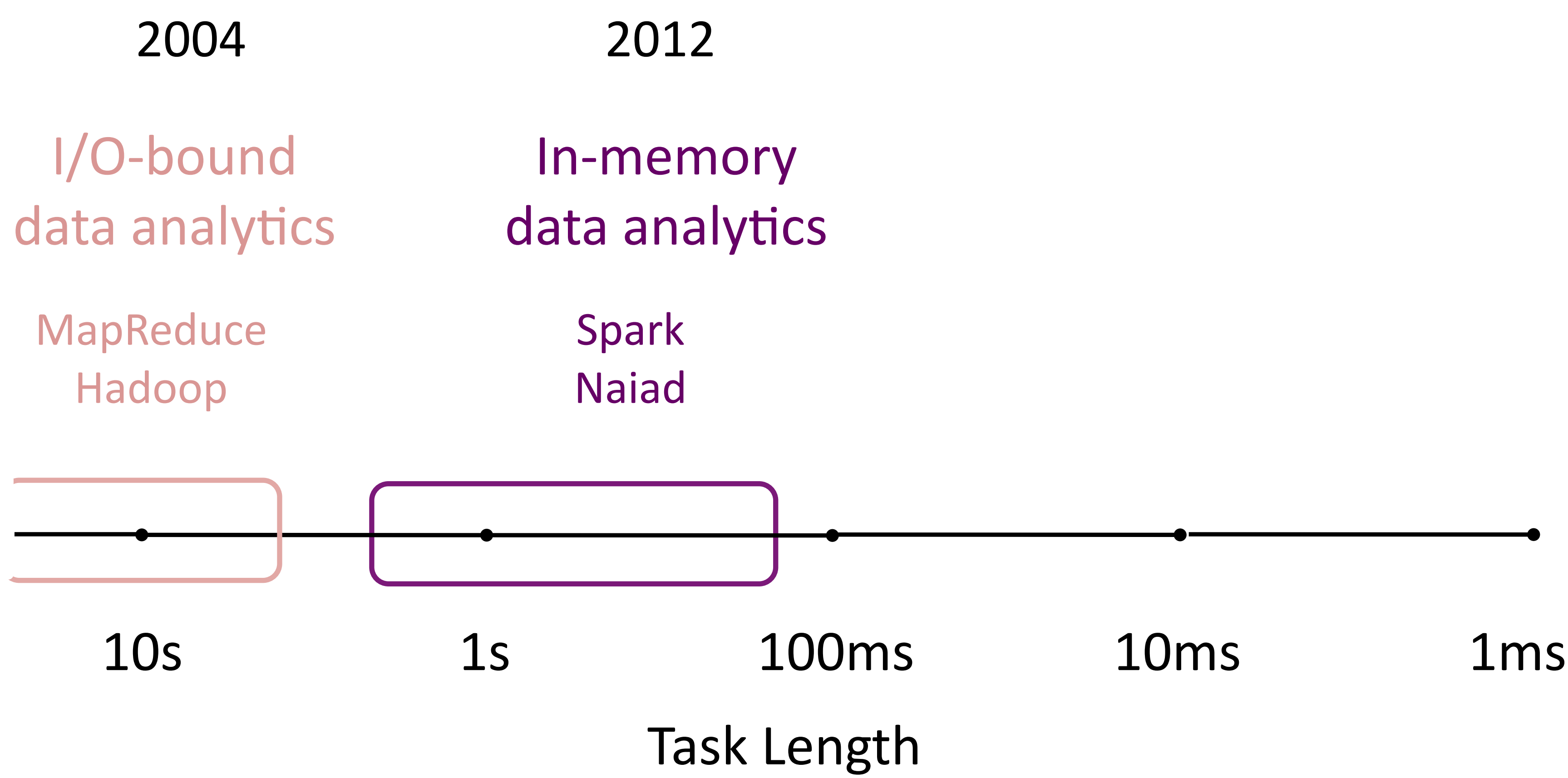
2004

I/O-bound
data analytics

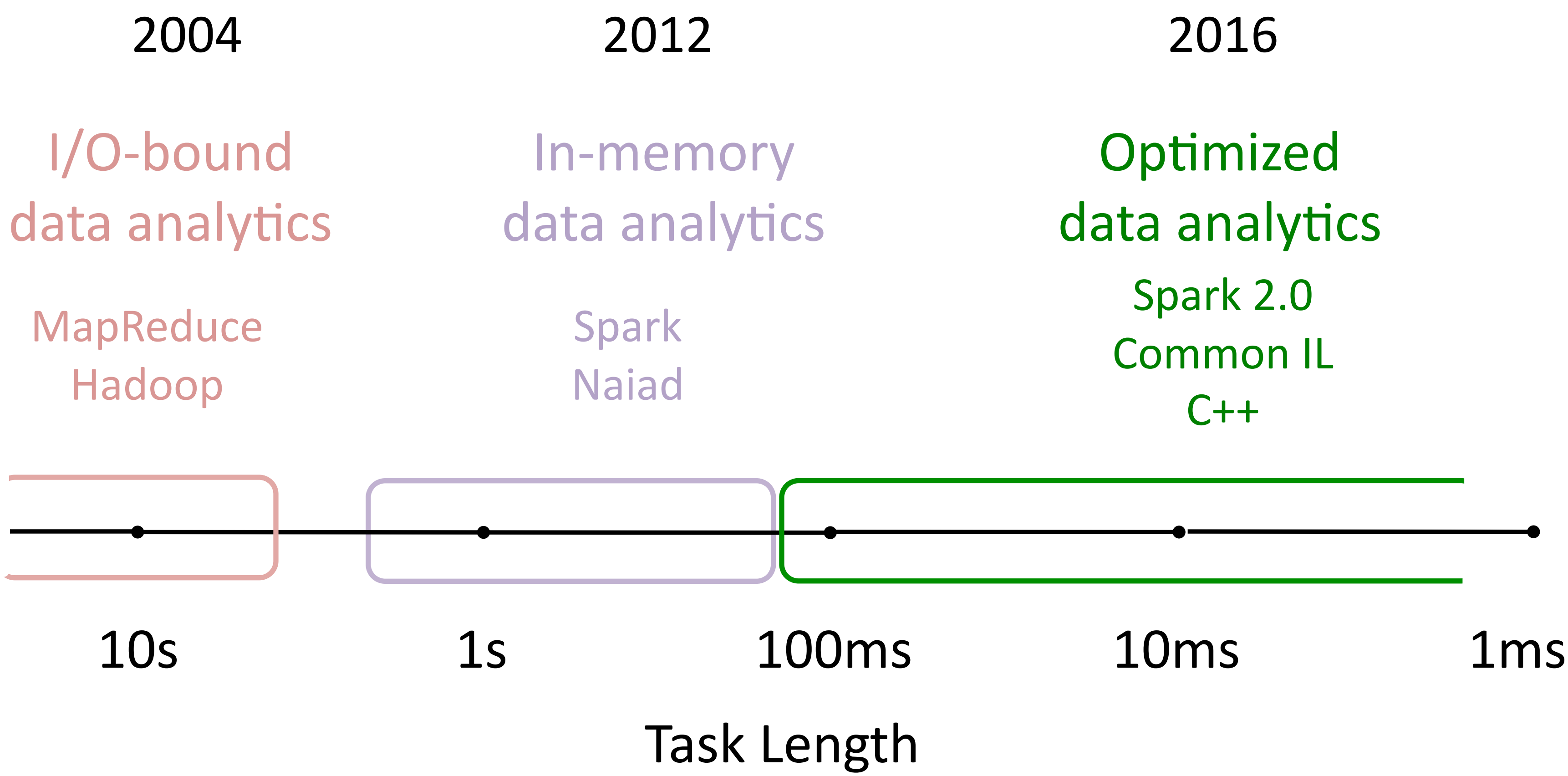
MapReduce
Hadoop

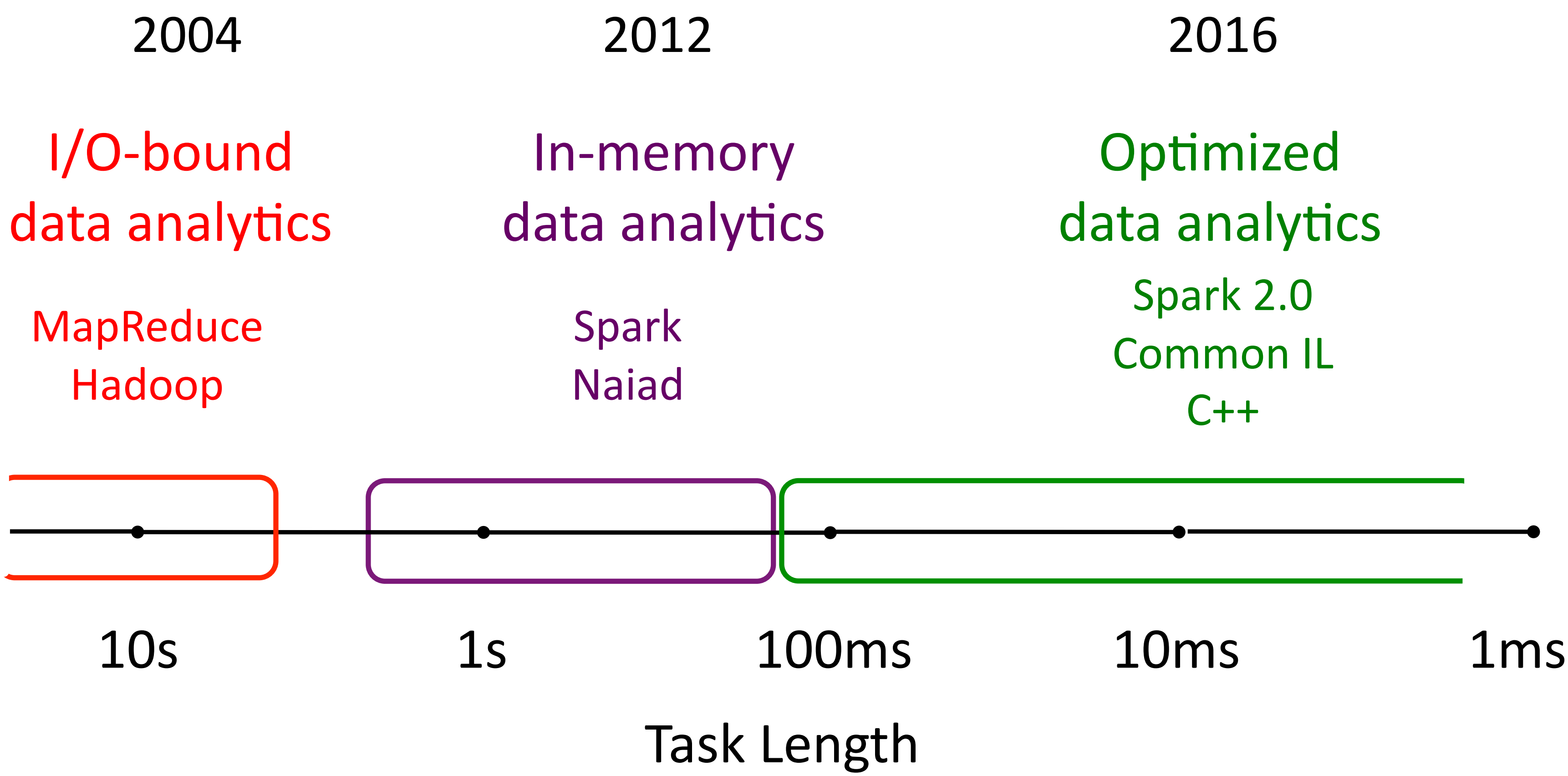


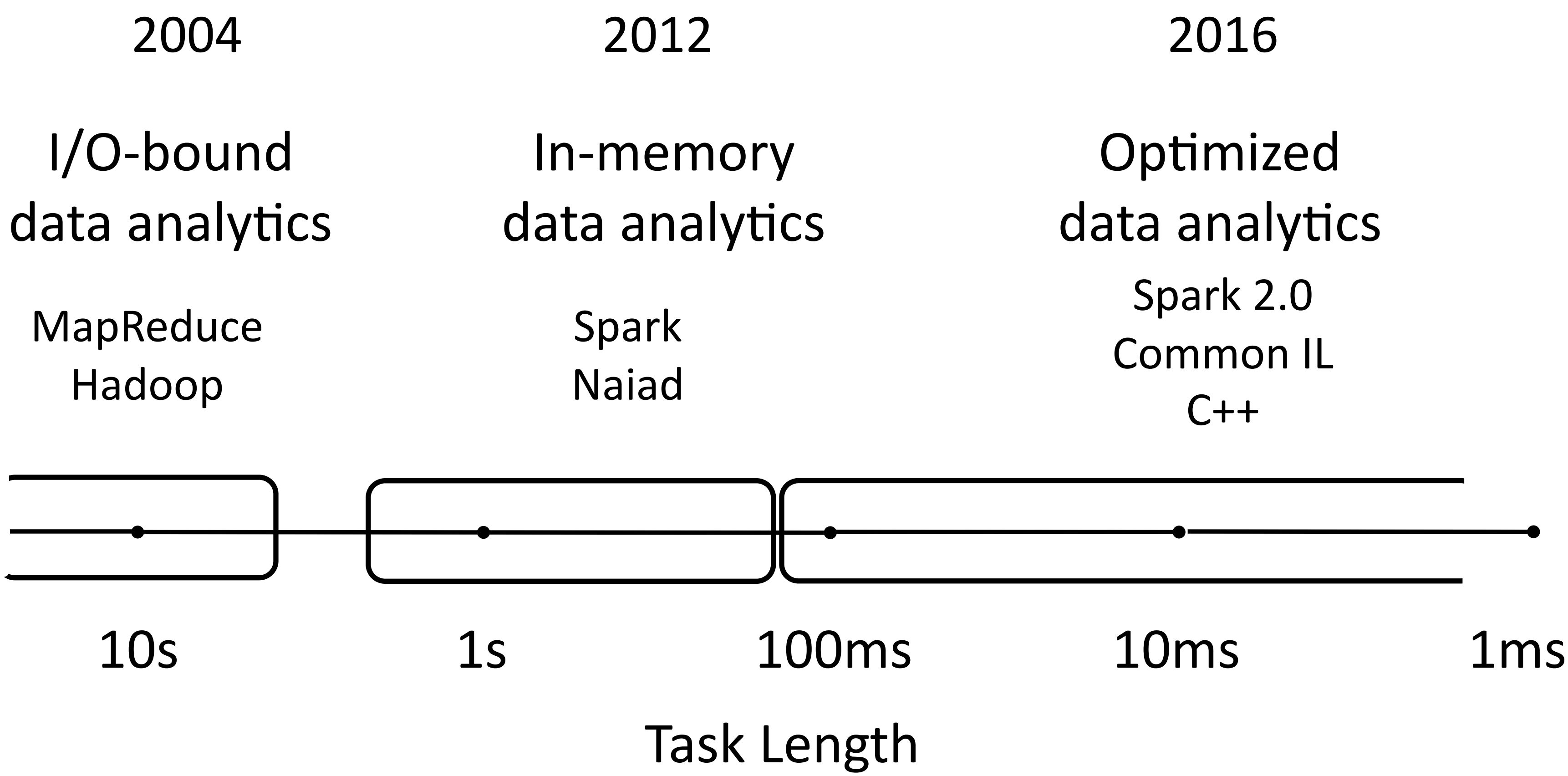
Evolution of Cloud Frameworks

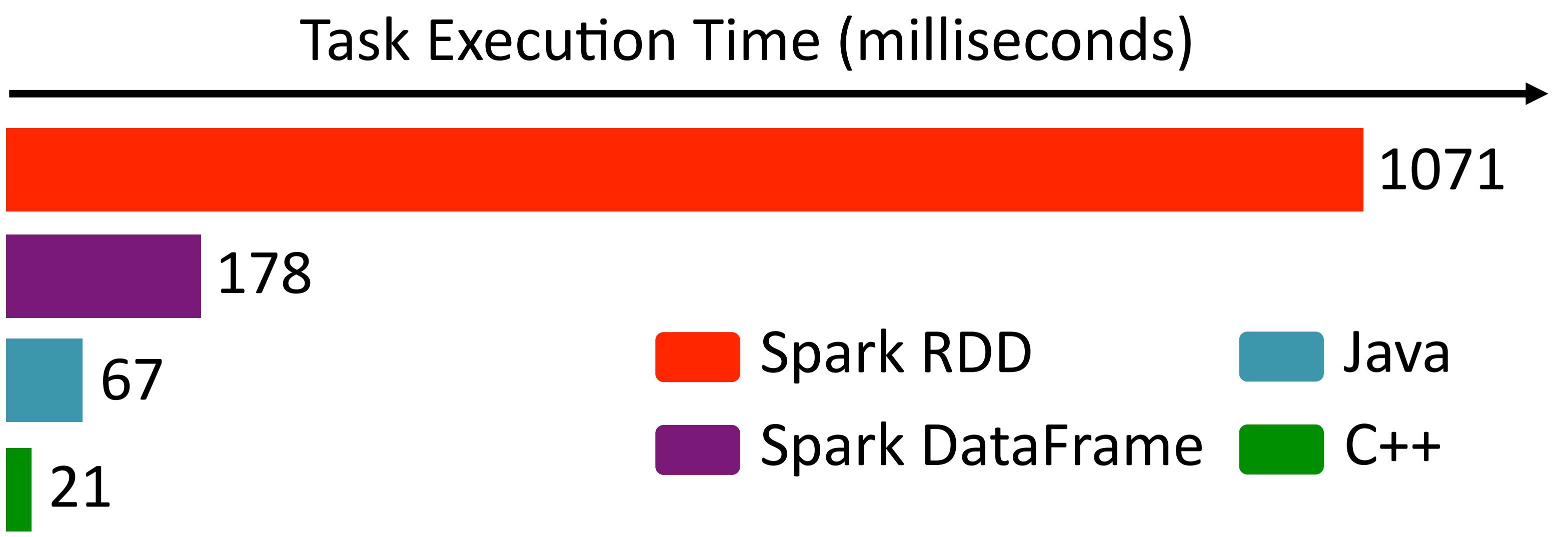


Evolution of Cloud Frameworks



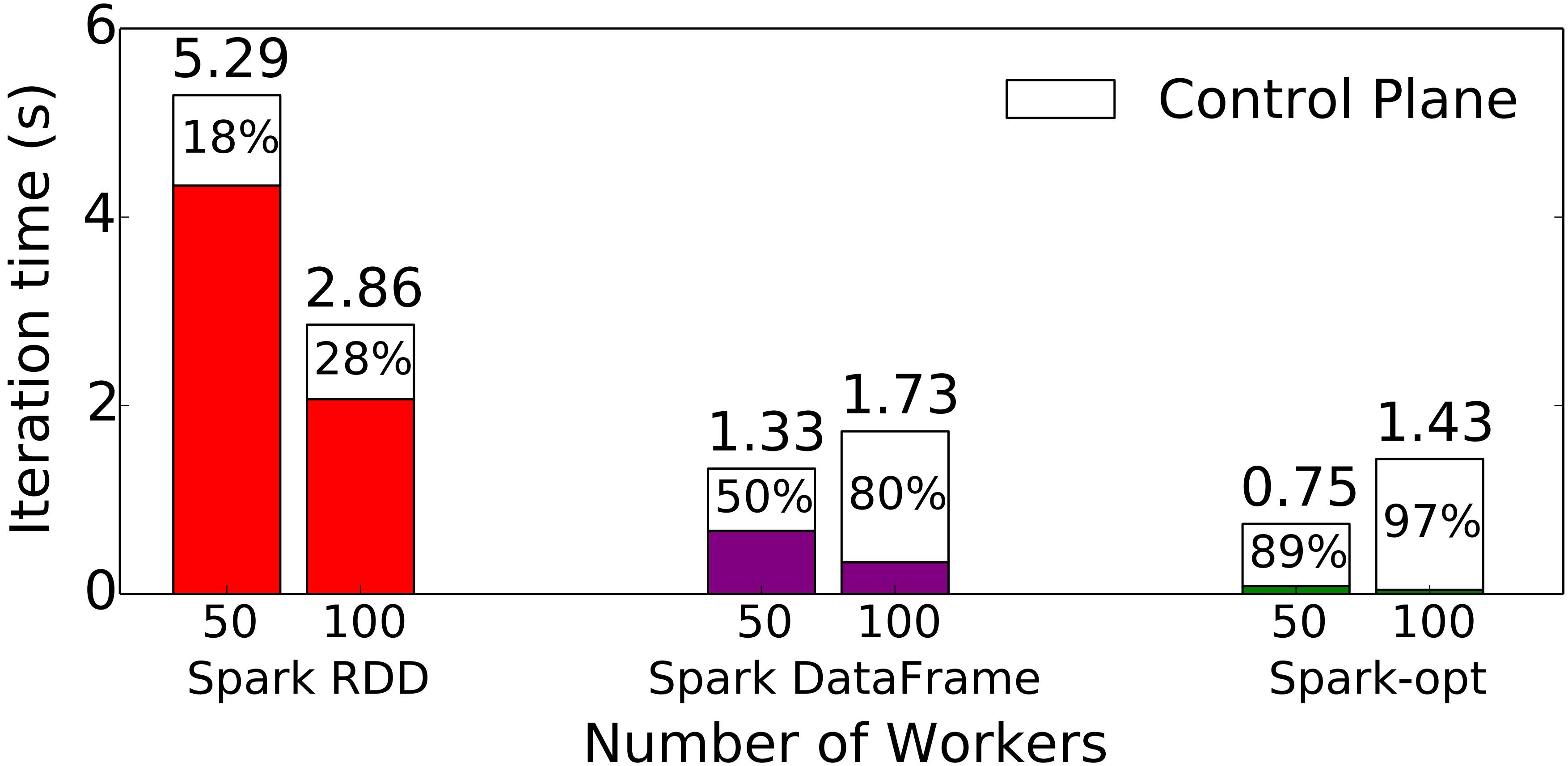

2004
2012
2016
I/O-bound
data analytics
In-memory
data analytics
Optimized
data analytics
MapReduce
Hadoop
Spark
Naiad
Spark 2.0
Common IL
C++
10s
1s
100ms
10ms
1ms
Task Length


2004
I/O-bound data analytics
MapReduce
Hadoop
2012
In-memory data analytics
Spark
Naiad
2016
Optimized data analytics
Spark 2.0
Common IL
C++
10s
1s
100ms
10ms
1ms
Task Length


Task Execution Time (milliseconds)
1071
178
67
21
Spark RDD
Java
Spark DataFrame
C++

Control Plane

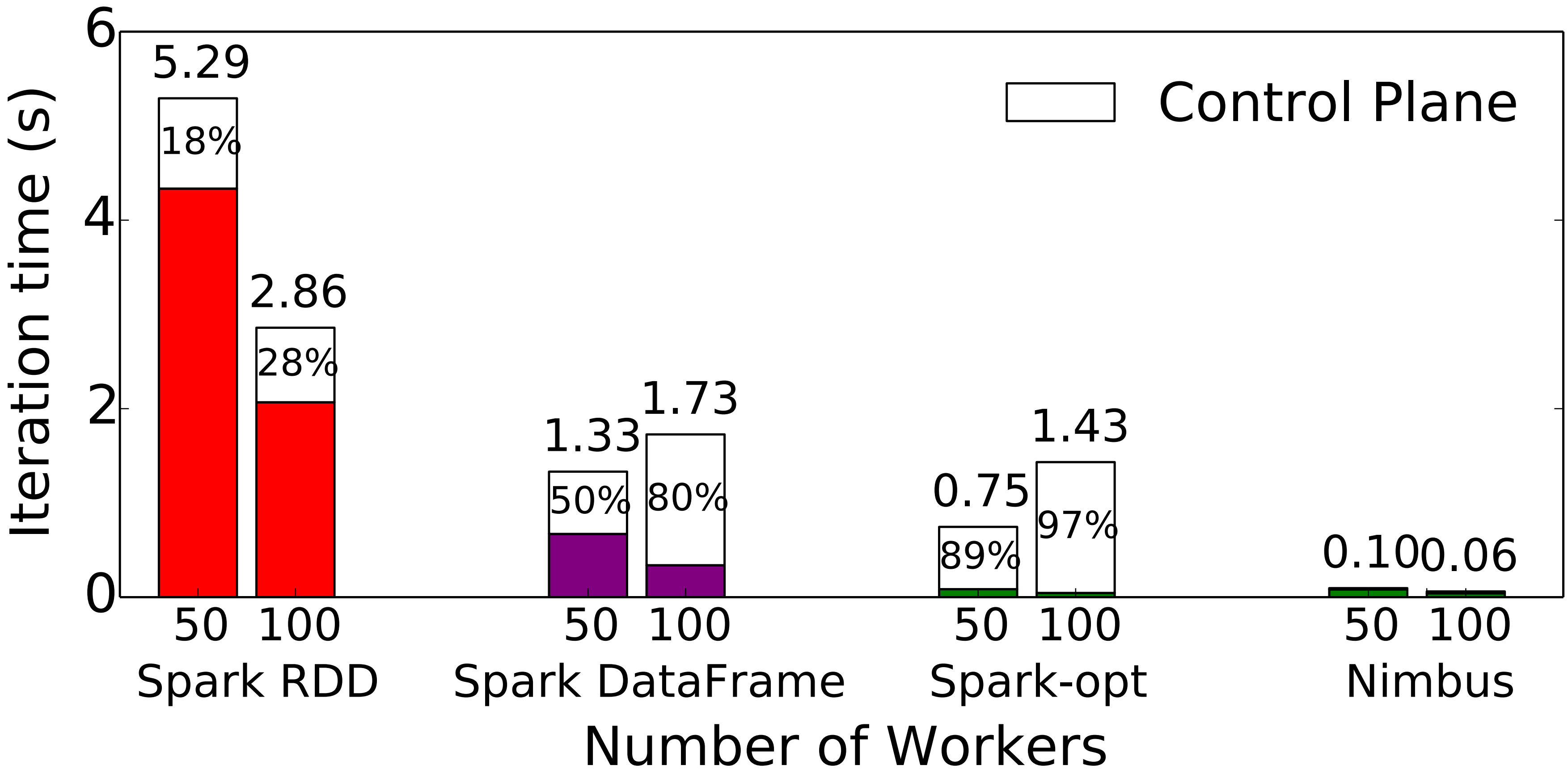
The New Bottleneck



- Logistic regression over a data set of size 100GB in different frameworks and settings.

Control Plane

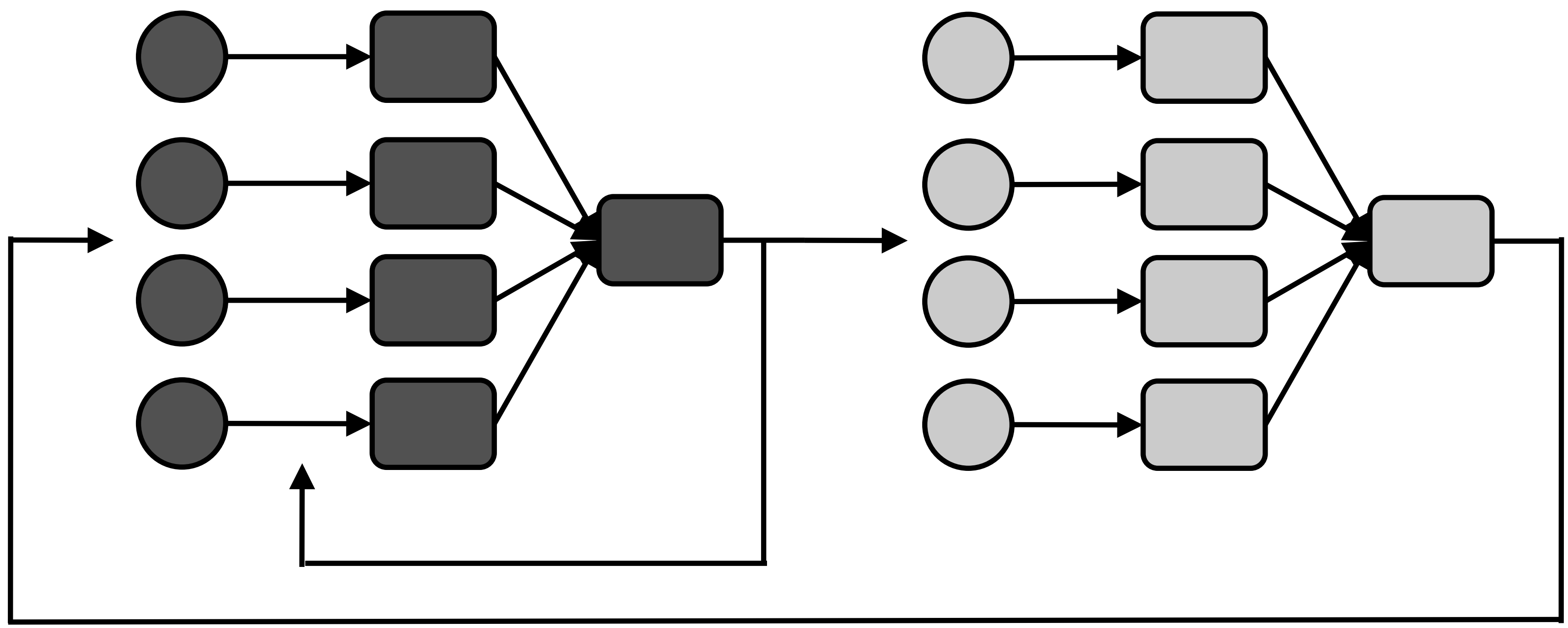
The New Bottleneck



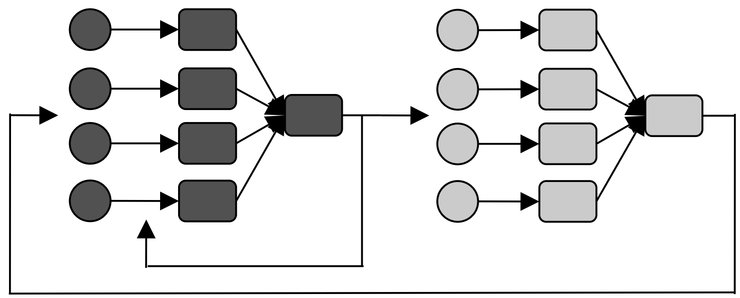
- Logistic regression over a data set of size 100GB in different frameworks and settings.


Google
Microsoft
facebook
amazon
Natural Language Processing
Speech Recognition
Graph Processing
Neural Networks

Example



Example



Conclusion

- Control plane is a new bottleneck for cloud computing frameworks.
- Execution templates enable orders of magnitude higher task throughput while keeping the dynamic scheduling.
- Execution templates are general enough to support classic data analytics, as well as complex applications.