



# Björn Knafla

# Parallelization of Game AI

Mittwoch, 17. Juni 2009

Motivation - why this talks?

Today I will give an overview of ways to parallelize game ai

... in no way complete, just a brief view

... purely technology oriented

Mittwoch, 17. Juni 2009

Developing Game AI is a challenge  
bigger worlds, more details, more interactions, more entities, dynamic worlds,  
more realism, expectations of lifelike behavior, emotions, individuals, ...

# More

Mittwoch, 17. Juni 2009

Developing Game AI is a challenge  
bigger worlds, more details, more interactions, more entities, dynamic worlds,  
more realism, expectations of lifelike behavior, emotions, individuals, ...

# More

# Dynamic

# More

# Dynamic

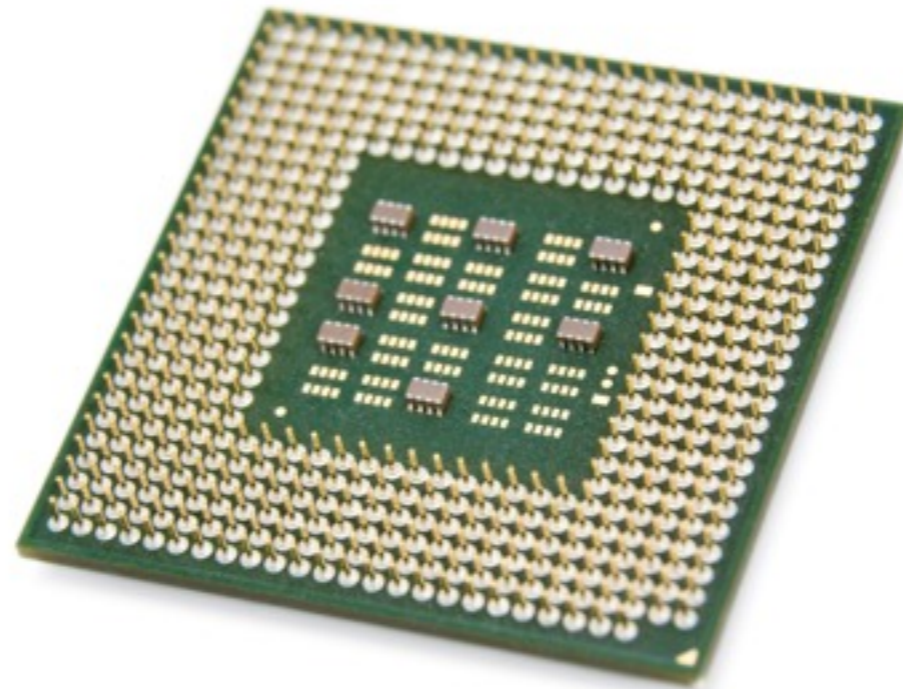
# Lifelike

# More

# Dynamic

# Lifelike

# Realism





- Errors due to non-synchronized access to shared resources
- Errors due to tasks blocking each other because they need resources the other holds
- Contention due to synchronization which limits performance

# Parallel Programming Problems

- Errors due to non-synchronized access to shared resources
- Errors due to tasks blocking each other because they need resources the other holds
- Contention due to synchronization which limits performance

# Parallel Programming Problems

## **Errors**

- Errors due to non-synchronized access to shared resources
- Errors due to tasks blocking each other because they need resources the other holds
- Contention due to synchronization which limits performance

# Parallel Programming Problems

## **Errors**

Race conditions

- Errors due to non-synchronized access to shared resources
- Errors due to tasks blocking each other because they need resources the other holds
- Contention due to synchronization which limits performance

# Parallel Programming Problems

## **Errors**

Race conditions

Deadlocks

- Errors due to non-synchronized access to shared resources
- Errors due to tasks blocking each other because they need resources the other holds
- Contention due to synchronization which limits performance

# Parallel Programming Problems

## **Errors**

Race conditions

Deadlocks

## **Performance**

- Errors due to non-synchronized access to shared resources
- Errors due to tasks blocking each other because they need resources the other holds
- Contention due to synchronization which limits performance

# Parallel Programming Problems

## **Errors**

Race conditions

Deadlocks

## **Performance**

Contention

- Errors due to non-synchronized access to shared resources
- Errors due to tasks blocking each other because they need resources the other holds
- Contention due to synchronization which limits performance



# Not Deterministic







... but will show you some methods and approaches that can help you exploit parallelism and push your AI further



# Outline

# Outline

## I. Parallel Hardware

# Outline

1. Parallel Hardware
2. Example AI

# Outline

1. Parallel Hardware
2. Example AI
3. Parallelization Approaches

# Outline

1. Parallel Hardware
2. Example AI
3. Parallelization Approaches
4. Conclusion

# Parallel Hardware

Mittwoch, 17. Juni 2009

12

... lets lift off

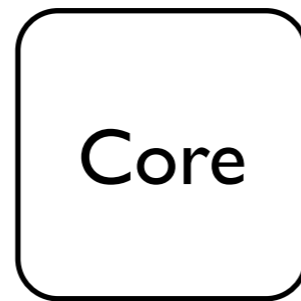
Its the parallel hardware that forces us to program in parallel - we need to understand it to understand parallel programming - a different programming model than Von Neumann Architecture (Model for sequential programming)

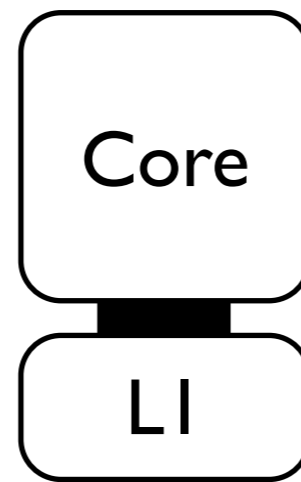
- high-level view of processor

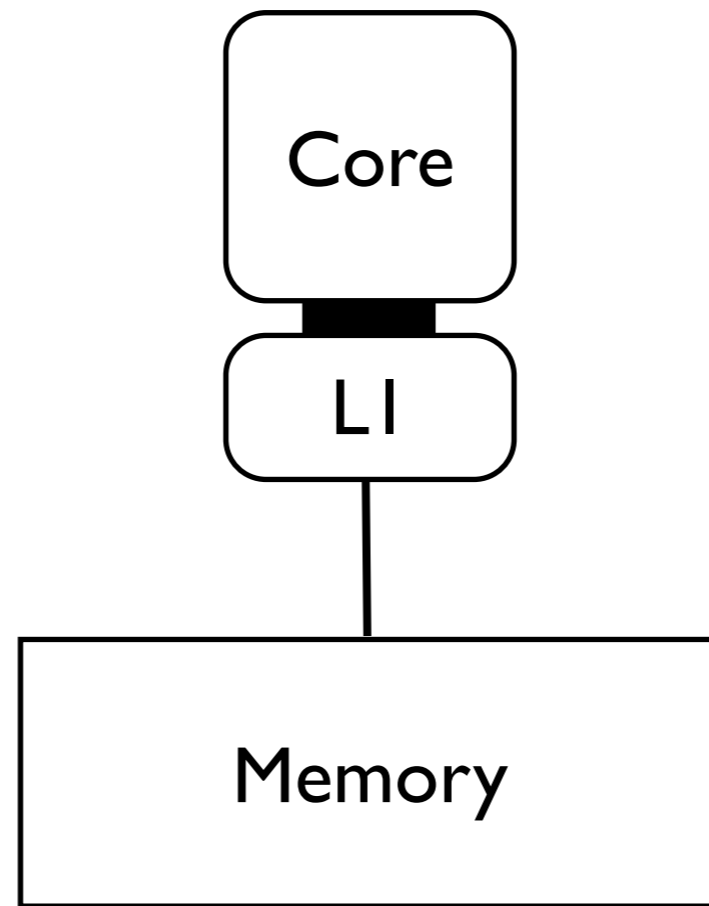
Mittwoch, 17. Juni 2009

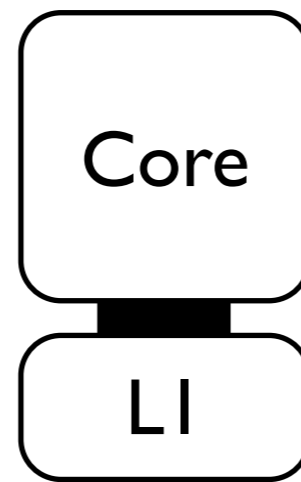
assume single address space and shared memory

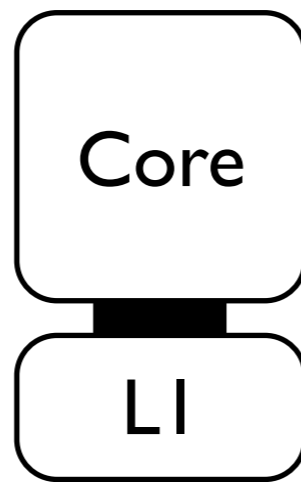
core like a CPU, a processing unit

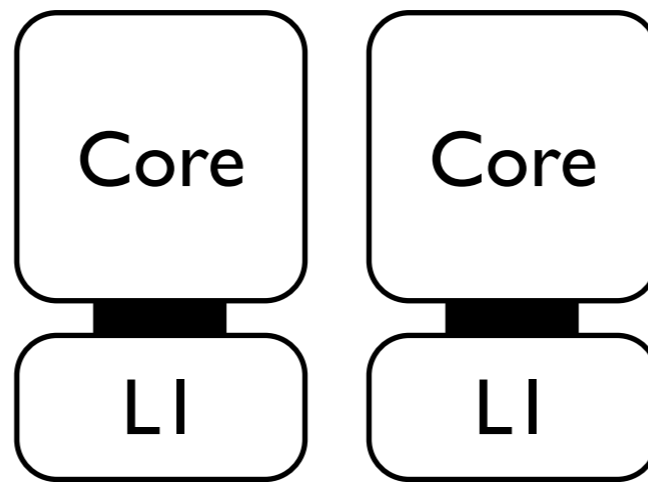


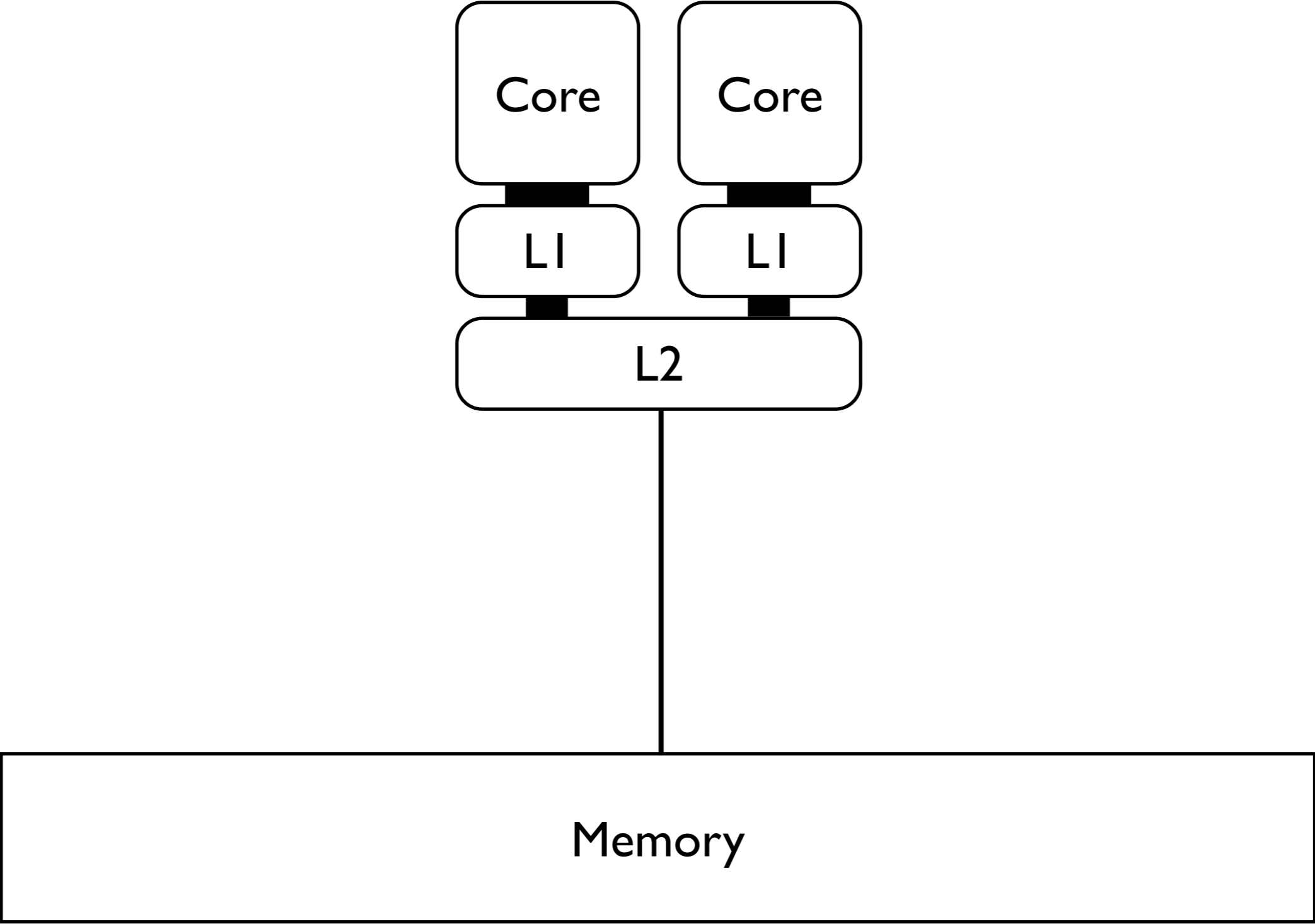


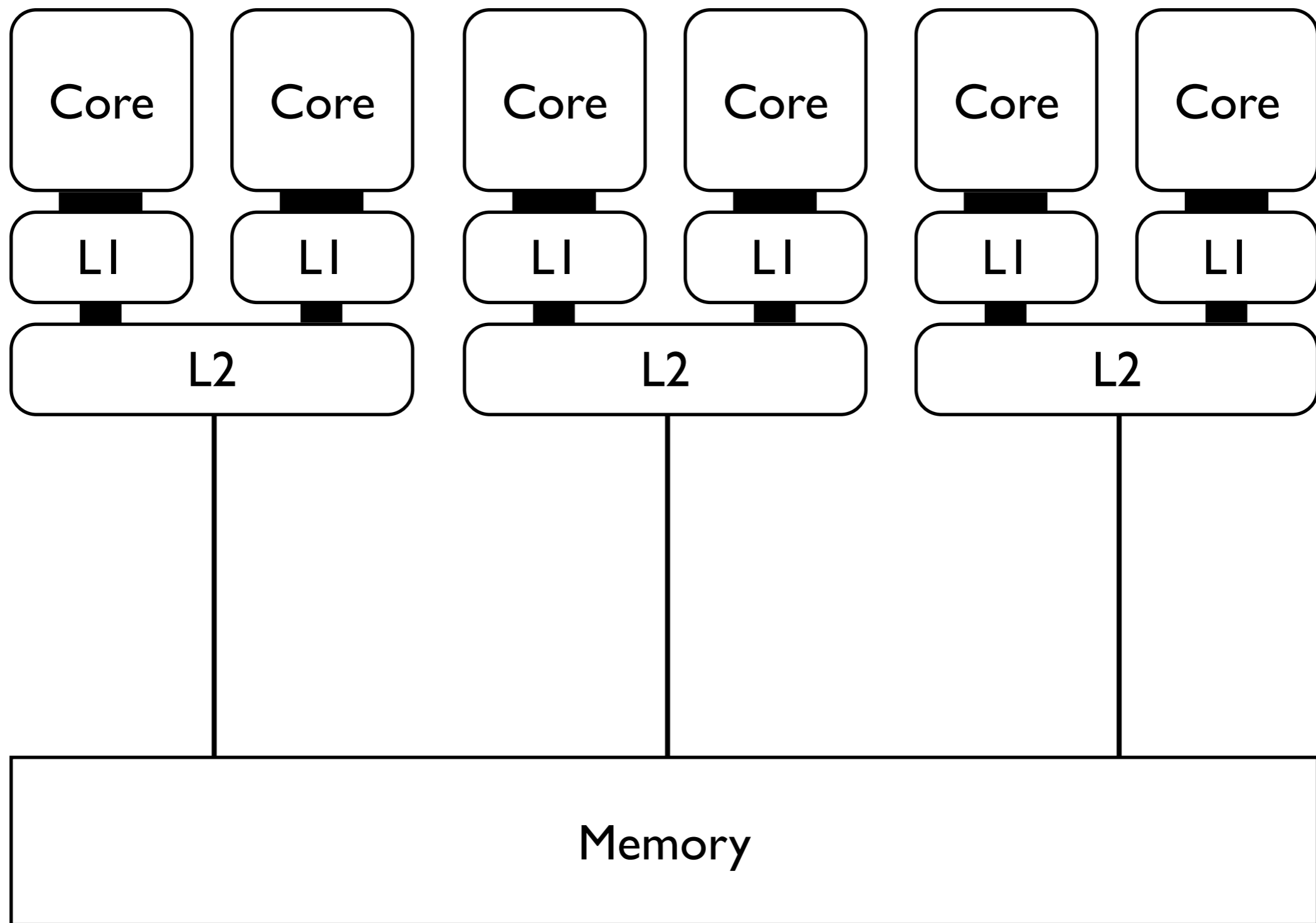


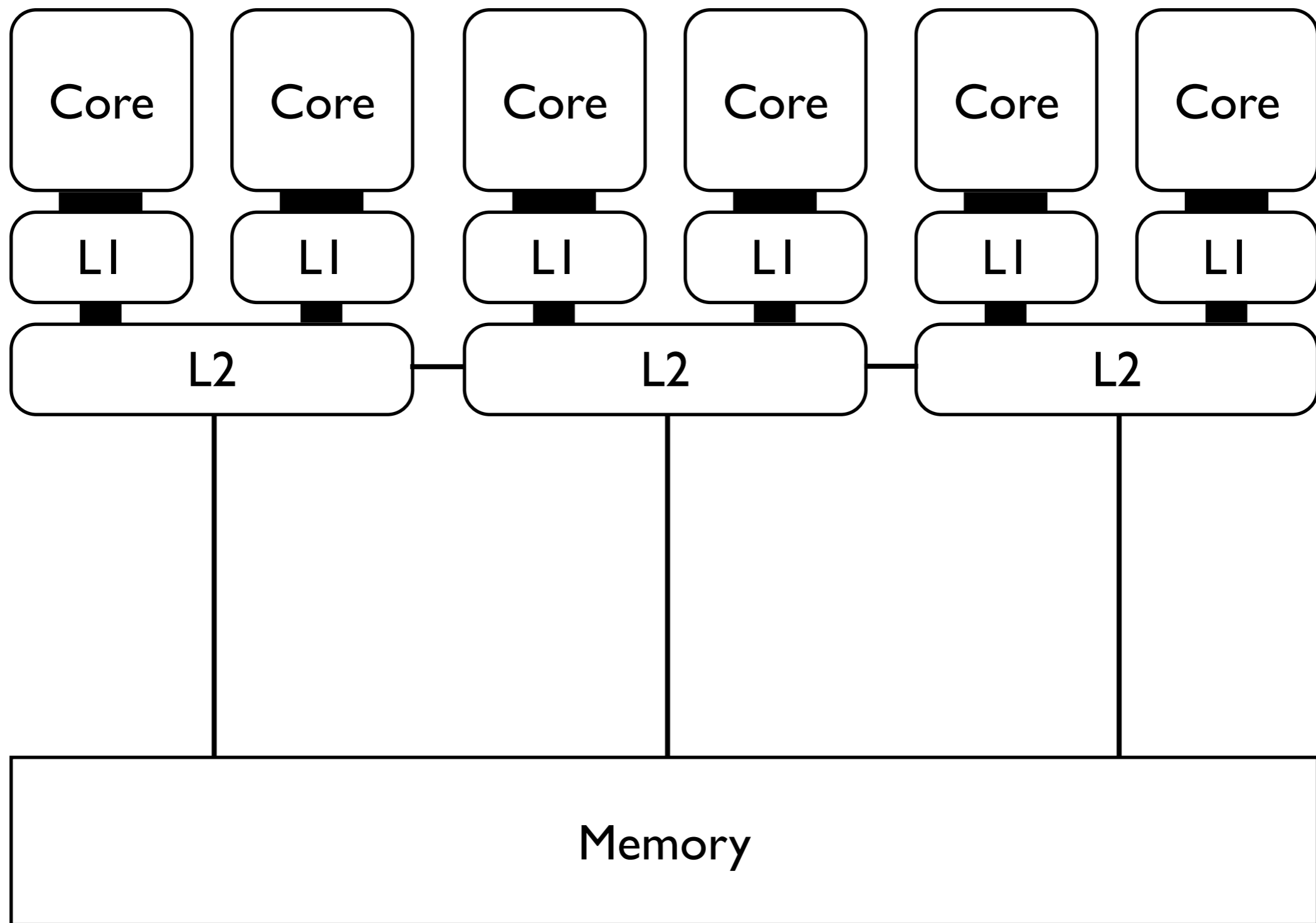












Mittwoch, 17. Juni 2009

Problem revisited: memory wall -> latency fighting, bandwidth usage  
Triggered by faster and faster processors - or  
more and more cores that want to be fed

# Memory Wall

Mittwoch, 17. Juni 2009

Problem revisited: memory wall -> latency fighting, bandwidth usage  
Triggered by faster and faster processors - or  
more and more cores that want to be fed

# Latency

# Latency

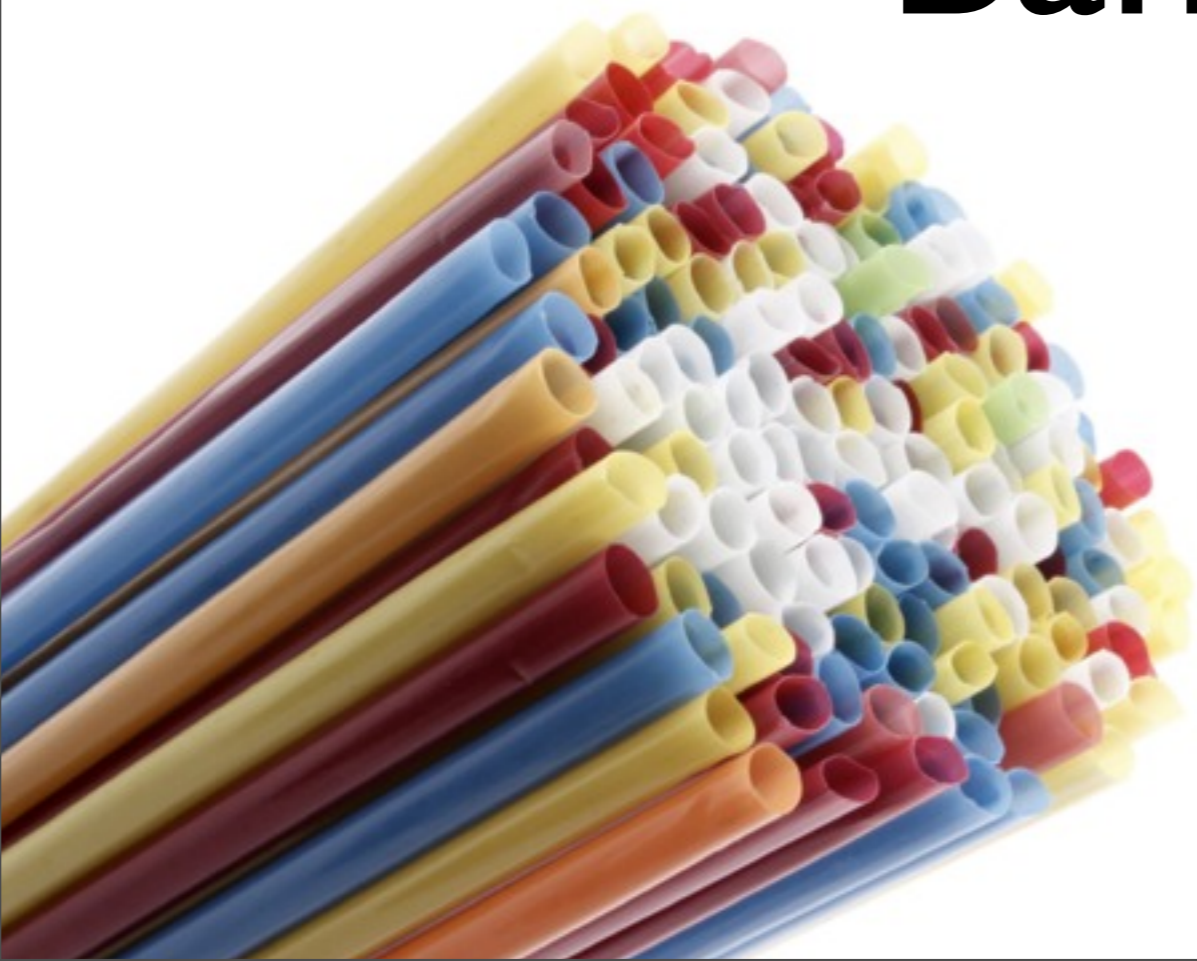


Mittwoch, 17. Juni 2009

Latency doesn't decrease as fast as core speed went / goes up

# Bandwidth

# Bandwidth



Mittwoch, 17. Juni 2009

Bandwidth increases - but precious resource when we have tons of cores (thousands? ten thousands? ...) that can easily saturate memory "bus"/connection

Mittwoch, 17. Juni 2009

sharing is the root of contention - sharing needs synchronization

cache-coherency protocol ping-pong

communication used bandwidth and takes time due to latency

synchronization cheaper for cores that share a cache or local storage

# Combined Problem

Mittwoch, 17. Juni 2009

sharing is the root of contention - sharing needs synchronization

cache-coherency protocol ping-pong

communication used bandwidth and takes time due to latency

synchronization cheaper for cores that share a cache or local storage

# Combined Problem

## Sharing



Mittwoch, 17. Juni 2009

sharing is the root of contention - sharing needs synchronization  
cache-coherency protocol ping-pong  
communication used bandwidth and takes time due to latency  
synchronization cheaper for cores that share a cache or local storage

# Solutions

# Latency

# Maximize **Throughput**

Mittwoch, 17. Juni 2009

Fight latency

Out-of-order execution

Caches or local storage

Prefetching

Simultaneous Multithreading (SMT)

# Bandwidth

# Maximize Memory **Locality**

Mittwoch, 17. Juni 2009

Fight bandwidth problem

especially between SMT hardware threads that share a cache

Reuse cache or local storage

Increase arithmetic intensity

Contiguous memory accesses - not random

Coalesced memory accesses

# Sharing

# Minimize (False) **Sharing**

Mittwoch, 17. Juni 2009

minimize sharing to prevent cache-coherency protocol ping-pong  
minimize sharing to minimize need for syncing  
minimize syncing

# Balance

# Balance

maximize      minimize  
**Locality**    vs.    **Sharing**

Mittwoch, 17. Juni 2009

27

Ok, so we have covered the background - we know the basic properties of parallel hardware, lets take a "real" example to see what all of this has to do with Game AI

- Computer controlled entities = agents
- real-time game, agents cooperate to capture a flag or clear an area

# Example AI

Mittwoch, 17. Juni 2009

27

Ok, so we have covered the background - we know the basic properties of parallel hardware, lets take a "real" example to see what all of this has to do with Game AI

- Computer controlled entities = agents
- real-time game, agents cooperate to capture a flag or clear an area

# Agent

Mittwoch, 17. Juni 2009

28

- agents have higher-level planning AI, lower-level reactive AI (FSM or BTs)
- they sense their environment and use pathfinding and locomotion planning to create effector actions
- reactive agent (higher level might be planning + terrain analysis)
- uses sensing to perceive its surrounding
- plans paths with pathfinding and follows paths using steering and locomotion planning
- generates actions (movement, taking ammunition, firing a gun) and animations

# Agent

Reactive behavior

- agents have higher-level planning AI, lower-level reactive AI (FSM or BTs)
- they sense their environment and use pathfinding and locomotion planning to create effector actions
- reactive agent (higher level might be planning + terrain analysis)
- uses sensing to perceive its surrounding
- plans paths with pathfinding and follows paths using steering and locomotion planning
- generates actions (movement, taking ammunition, firing a gun) and animations

# Agent

Sensing

Reactive behavior

- agents have higher-level planning AI, lower-level reactive AI (FSM or BTs)
- they sense their environment and use pathfinding and locomotion planning to create effector actions
- reactive agent (higher level might be planning + terrain analysis)
- uses sensing to perceive its surrounding
- plans paths with pathfinding and follows paths using steering and locomotion planning
- generates actions (movement, taking ammunition, firing a gun) and animations

# Agent

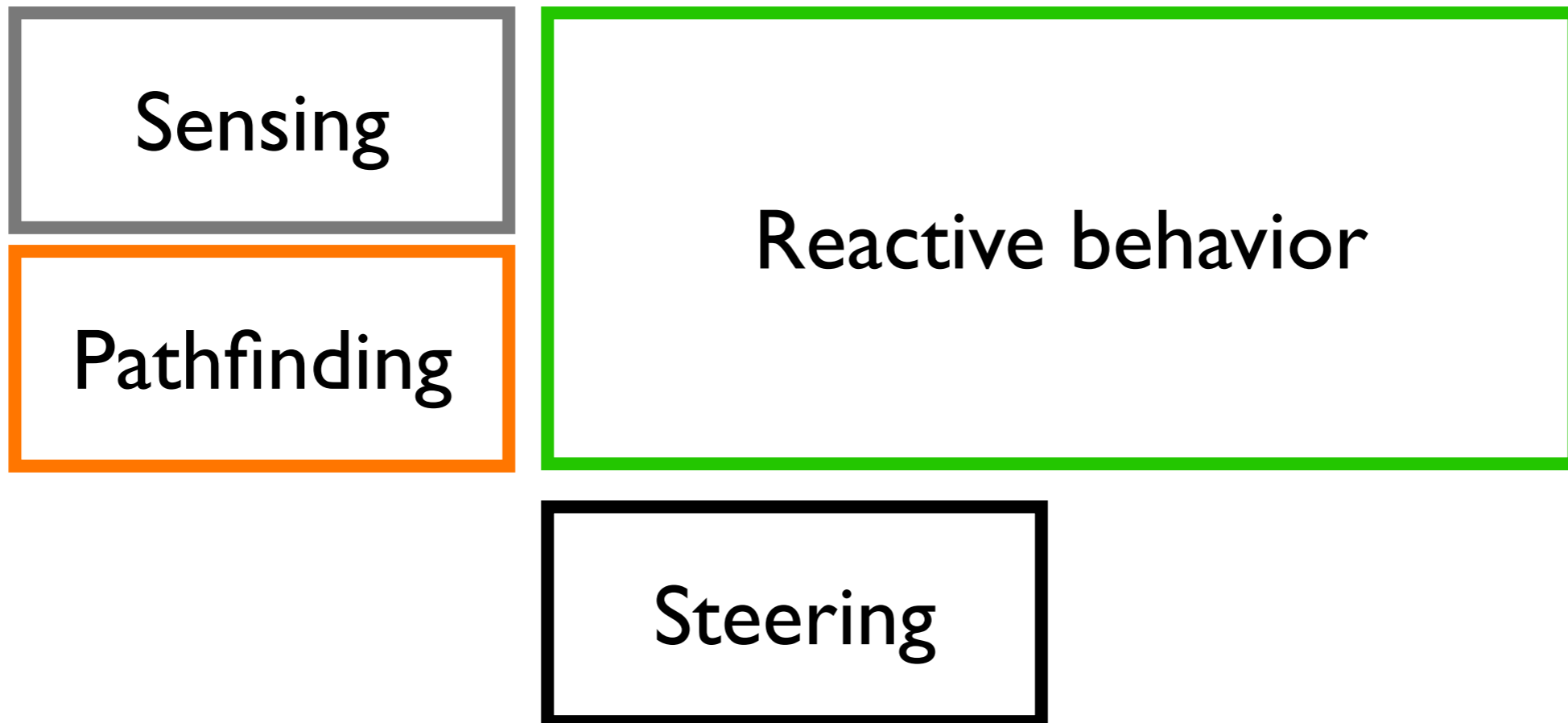
Sensing

Pathfinding

Reactive behavior

- agents have higher-level planning AI, lower-level reactive AI (FSM or BTs)
- they sense their environment and use pathfinding and locomotion planning to create effector actions
- reactive agent (higher level might be planning + terrain analysis)
- uses sensing to perceive its surrounding
- plans paths with pathfinding and follows paths using steering and locomotion planning
- generates actions (movement, taking ammunition, firing a gun) and animations

# Agent

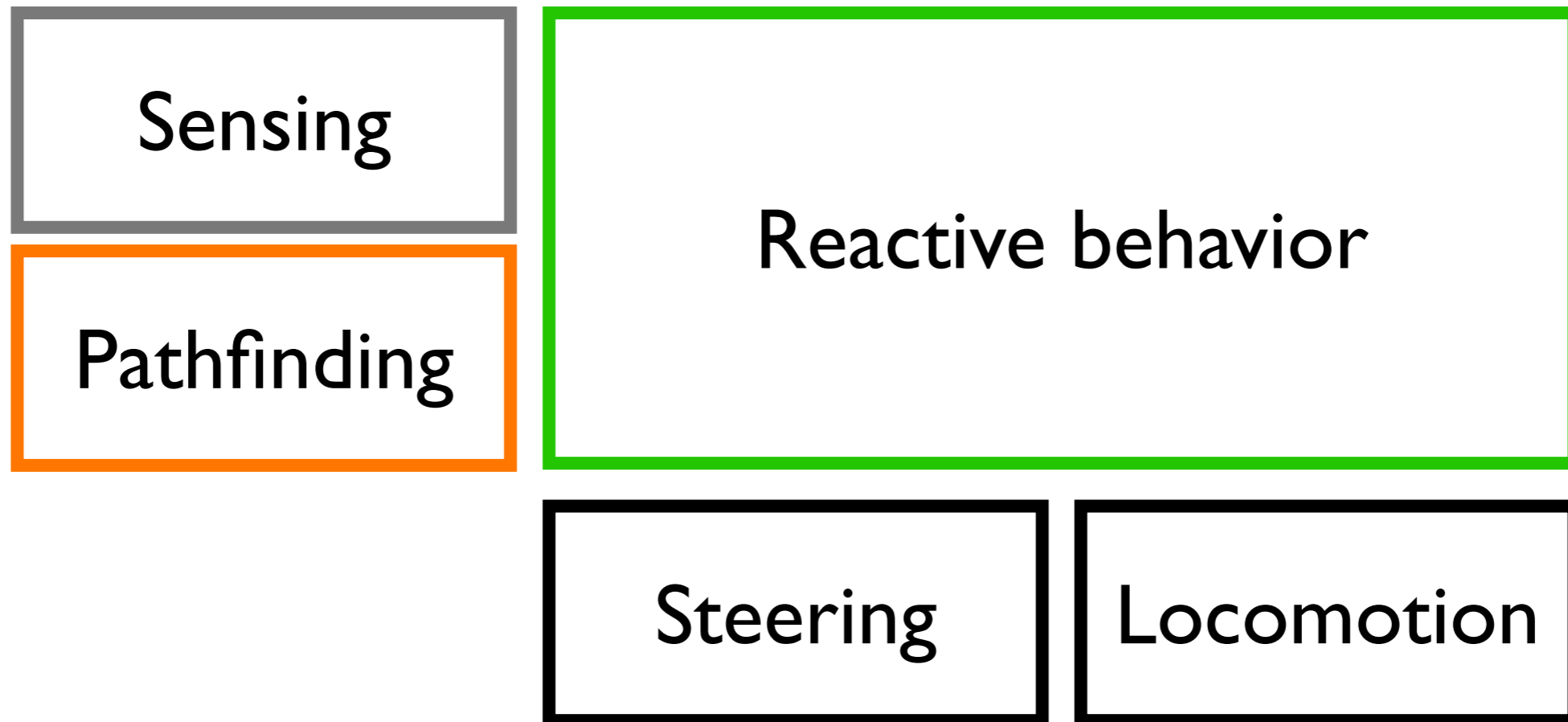


Mittwoch, 17. Juni 2009

28

- agents have higher-level planning AI, lower-level reactive AI (FSM or BTs)
- they sense their environment and use pathfinding and locomotion planning to create effector actions
- reactive agent (higher level might be planning + terrain analysis)
- uses sensing to perceive its surrounding
- plans paths with pathfinding and follows paths using steering and locomotion planning
- generates actions (movement, taking ammunition, firing a gun) and animations

# Agent

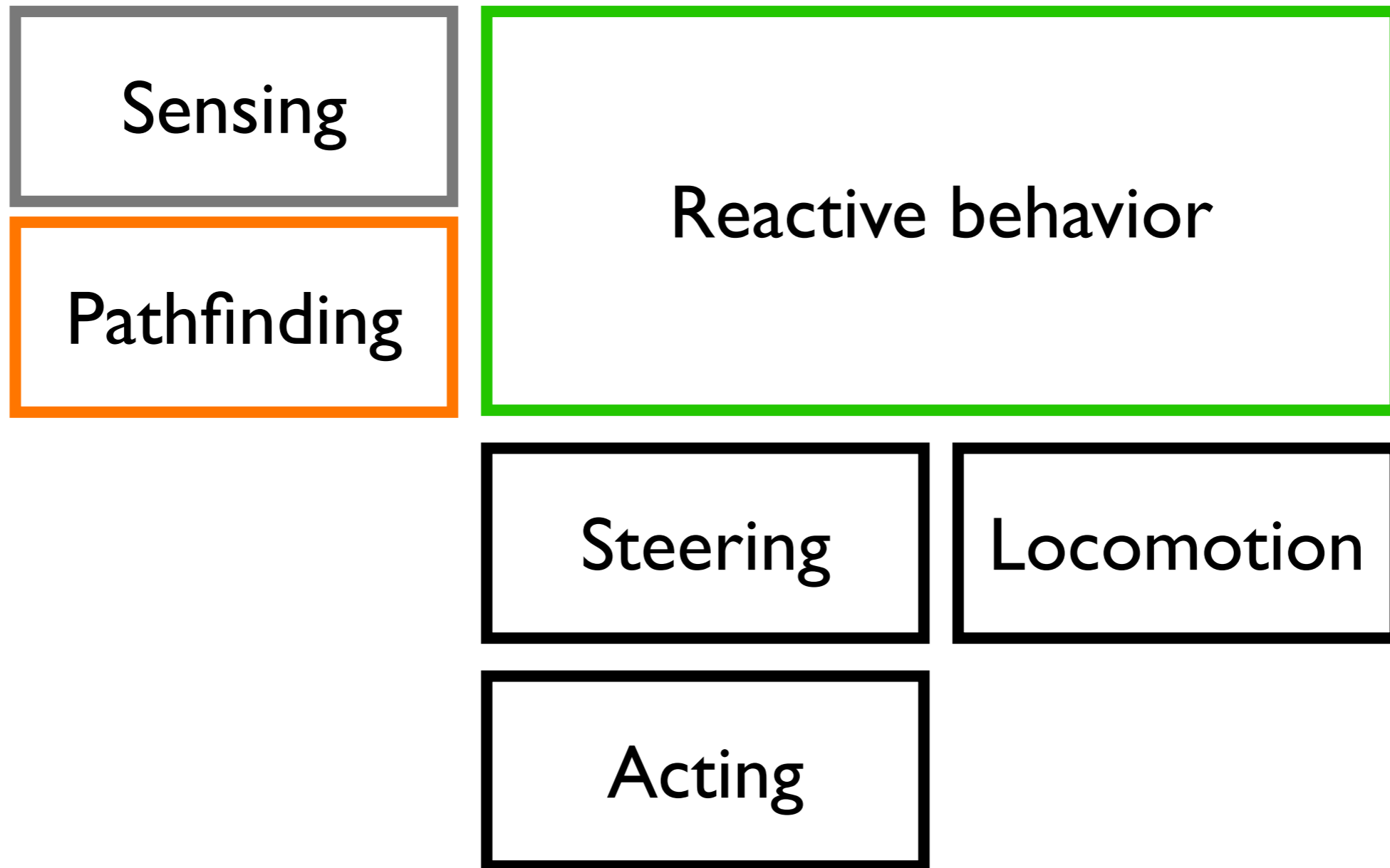


Mittwoch, 17. Juni 2009

28

- agents have higher-level planning AI, lower-level reactive AI (FSM or BTs)
- they sense their environment and use pathfinding and locomotion planning to create effector actions
- reactive agent (higher level might be planning + terrain analysis)
- uses sensing to perceive its surrounding
- plans paths with pathfinding and follows paths using steering and locomotion planning
- generates actions (movement, taking ammunition, firing a gun) and animations

# Agent

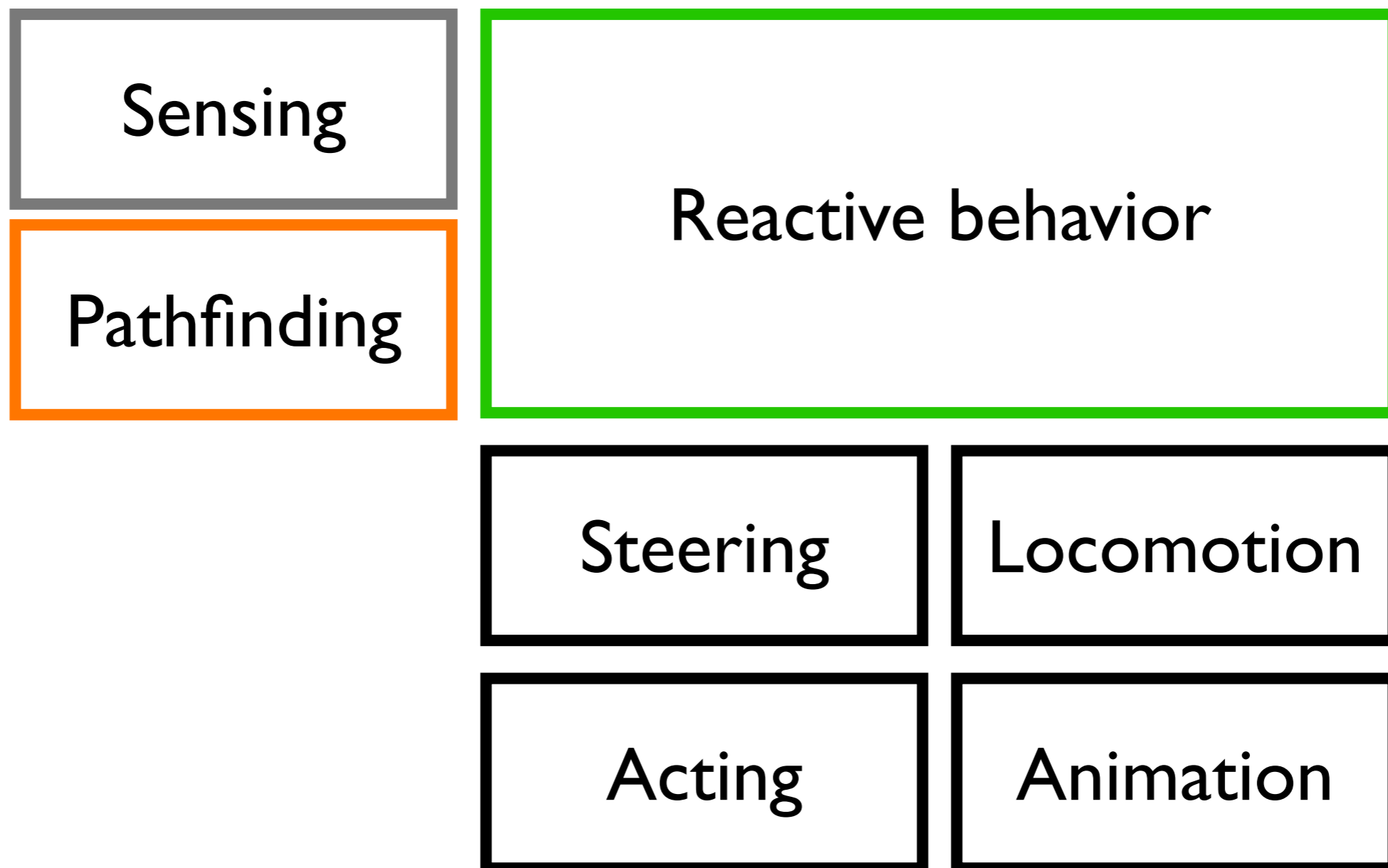


Mittwoch, 17. Juni 2009

28

- agents have higher-level planning AI, lower-level reactive AI (FSM or BTs)
- they sense their environment and use pathfinding and locomotion planning to create effector actions
- reactive agent (higher level might be planning + terrain analysis)
- uses sensing to perceive its surrounding
- plans paths with pathfinding and follows paths using steering and locomotion planning
- generates actions (movement, taking ammunition, firing a gun) and animations

# Agent



Mittwoch, 17. Juni 2009

28

- agents have higher-level planning AI, lower-level reactive AI (FSM or BTs)
- they sense their environment and use pathfinding and locomotion planning to create effector actions
- reactive agent (higher level might be planning + terrain analysis)
- uses sensing to perceive its surrounding
- plans paths with pathfinding and follows paths using steering and locomotion planning
- generates actions (movement, taking ammunition, firing a gun) and animations

# Parallelization Approaches

Mittwoch, 17. Juni 2009

29

Now on towards real parallelim, simultaneous execution of multiple tasks

Parallelize because conceptual planning shows need

Parallelize because profiling shows need

I show parallelization for performance (game logic relevant) - other alternative, parallelization for eye candy - non-game logic relevant but nice and fluffy

# I. Asynchronous Calls

Ok, lets concentrate purely on AI from now on  
Profiler driven  
Valve's Source Engine supports this

Sequential agent update-loop

Run pathfinding **async** on other thread

Freestep using raw threads -> not deterministic

Uncontrollable scaling with raw threads (over- or under-subscribed, context switching)

Expensive to create and destroy raw threads often

Frames

Core



- Sequential agent update-loop
- Run pathfinding **async** on other thread
- Freestep using raw threads -> not deterministic
- Uncontrollable scaling with raw threads (over- or under-subscribed, context switching)
- Expensive to create and destroy raw threads often

# Frames



Core

1

2

3

4

5

⋮

- Sequential agent update-loop
- Run pathfinding **async** on other thread
- Freestep using raw threads -> not deterministic
- Uncontrollable scaling with raw threads (over- or under-subscribed, context switching)
- Expensive to create and destroy raw threads often

# Frames



Agent 1

Core  
1  
2  
3  
4  
5  
⋮

- Sequential agent update-loop
- Run pathfinding **async** on other thread
- Freestep using raw threads -> not deterministic
- Uncontrollable scaling with raw threads (over- or under-subscribed, context switching)
- Expensive to create and destroy raw threads often

# Frames



Core  
1  
2  
3  
4  
5  
⋮

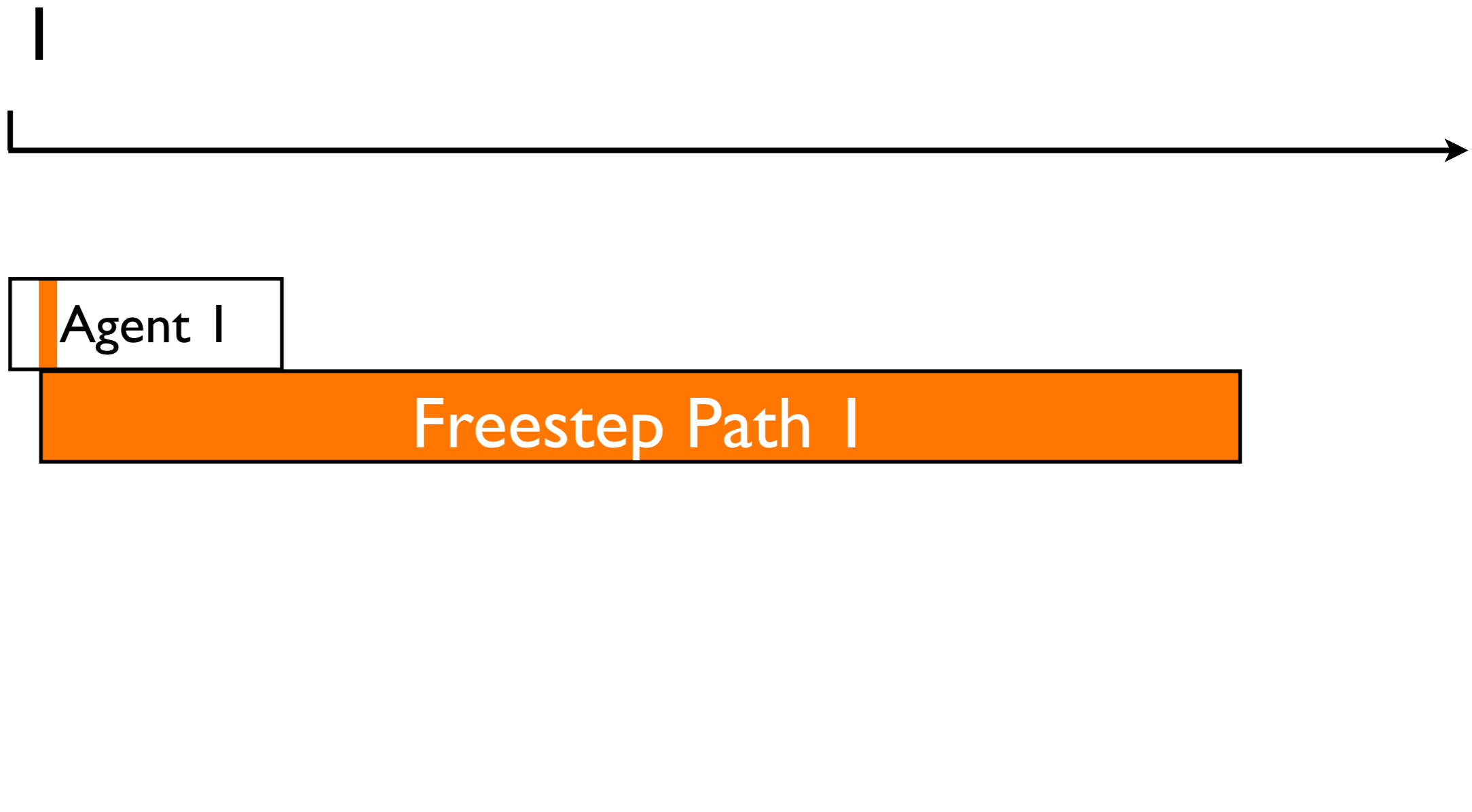


Agent 1

- Sequential agent update-loop
- Run pathfinding **async** on other thread
- Freestep using raw threads -> not deterministic
- Uncontrollable scaling with raw threads (over- or under-subscribed, context switching)
- Expensive to create and destroy raw threads often

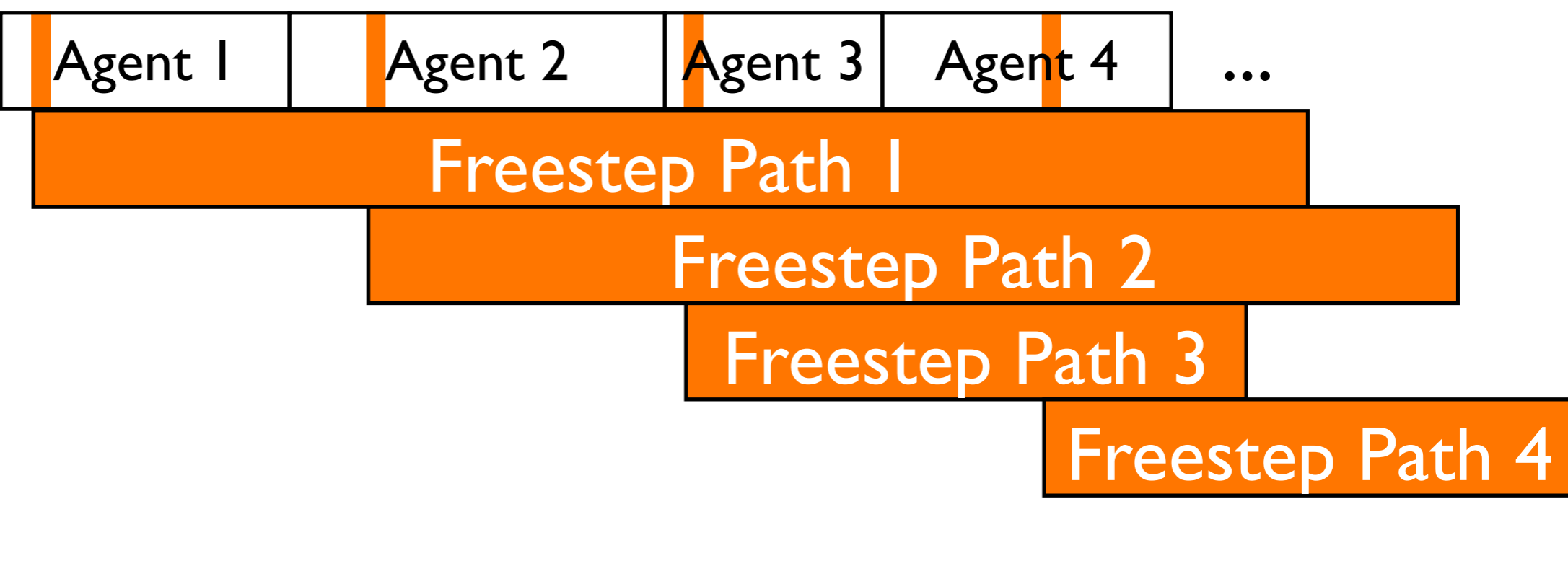
# Frames

Core



- Sequential agent update-loop
- Run pathfinding **async** on other thread
- Freestep using raw threads -> not deterministic
- Uncontrollable scaling with raw threads (over- or under-subscribed, context switching)
- Expensive to create and destroy raw threads often

# Frames



- Sequential agent update-loop
- Run pathfinding **async** on other thread
- Freestep using raw threads -> not deterministic
- Uncontrollable scaling with raw threads (over- or under-subscribed, context switching)
- Expensive to create and destroy raw threads often

## Tasks

Creates threads it manages once – user enters tasks, doesn't interact directly with raw threads

Enables scaling, automatic load balancing

Central queue (thread pool) or central + queue per thread (task pool)

Tasks can spawn (sub-)tasks in local thread queue

Work stealing

# Task Pool

## Tasks

Creates threads it manages once – user enters tasks, doesn't interact directly with raw threads

Enables scaling, automatic load balancing

Central queue (thread pool) or central + queue per thread (task pool)

Tasks can spawn (sub-)tasks in local thread queue

Work stealing

# Task Pool

Mittwoch, 17. Juni 2009

32

## Tasks

Creates threads it manages once – user enters tasks, doesn't interact directly with raw threads

Enables scaling, automatic load balancing

Central queue (thread pool) or central + queue per thread (task pool)

Tasks can spawn (sub-)tasks in local thread queue

Work stealing

A diagram illustrating a task pool. On the left, a vertical blue rectangle is labeled "Task Pool". Three horizontal black arrows originate from the right side of this rectangle, pointing towards the right edge of the slide. The arrows are labeled "Thread 1", "Thread 2", and "Thread n" from top to bottom, respectively. The labels are positioned to the right of the rectangle and to the left of the arrows.

Task Pool

Thread 1

Thread 2

Thread n

## Tasks

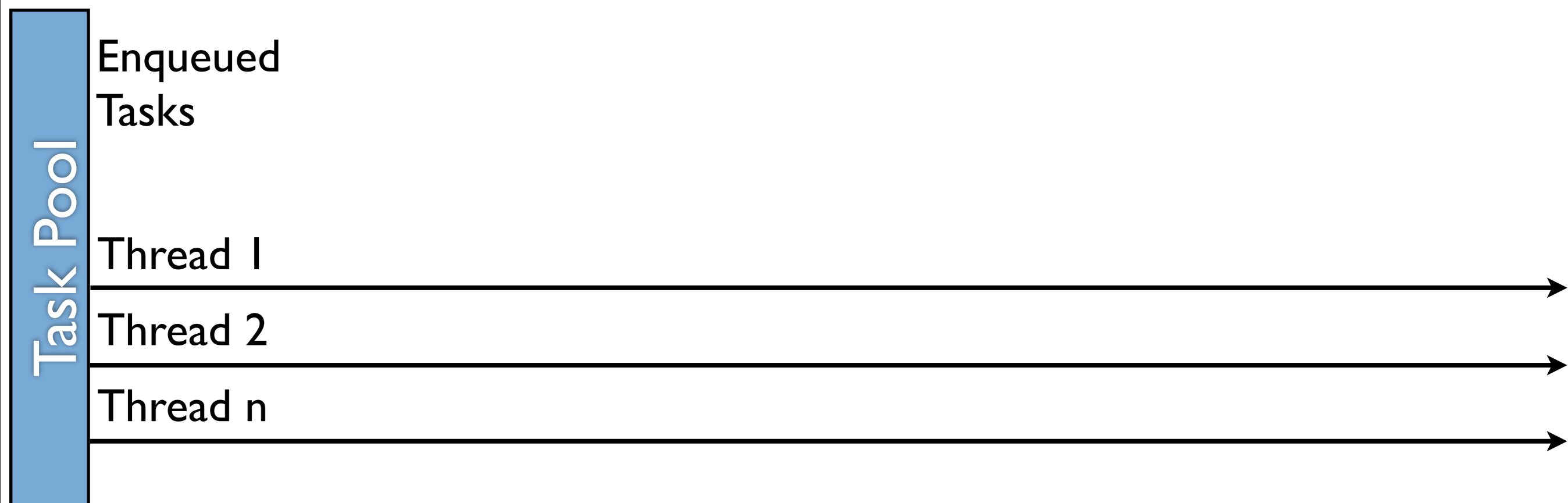
Creates threads it manages once – user enters tasks, doesn't interact directly with raw threads

Enables scaling, automatic load balancing

Central queue (thread pool) or central + queue per thread (task pool)

Tasks can spawn (sub-)tasks in local thread queue

Work stealing



## Tasks

Creates threads it manages once – user enters tasks, doesn't interact directly with raw threads

Enables scaling, automatic load balancing

Central queue (thread pool) or central + queue per thread (task pool)

Tasks can spawn (sub-)tasks in local thread queue

Work stealing

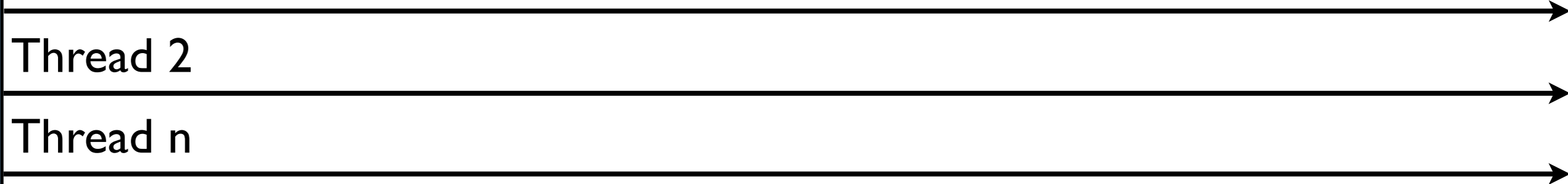
Task Pool

Enqueued  
Tasks

Thread 1

Thread 2

Thread n



# Frames

1

2



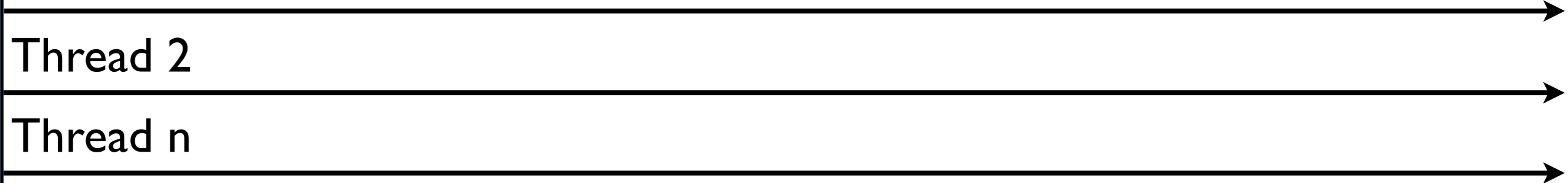
Task Pool

Enqueued  
Tasks

Thread 1

Thread 2

Thread n



# Frames



Task Pool

Enqueued  
Tasks

Thread 1

Thread 2

Thread n

# Frames

1

2



Agent 1

Task Pool

Enqueued  
Tasks

Thread 1

Thread 2

Thread n



# Frames

1

2



Task Pool

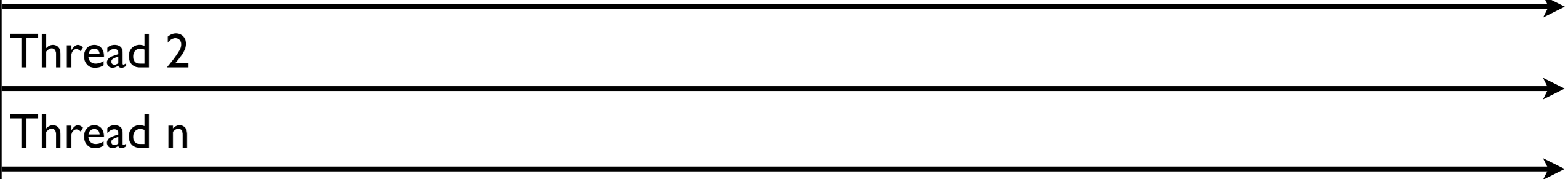
Enqueued  
Tasks



Thread 1

Thread 2

Thread n



# Frames

1

2



Agent 1

Enqueued  
Tasks

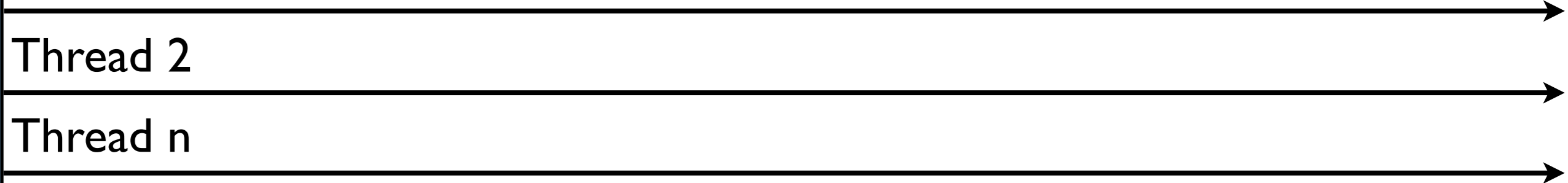


Thread 1

Thread 2

Thread n

Task Pool



# Frames



## Task Pool

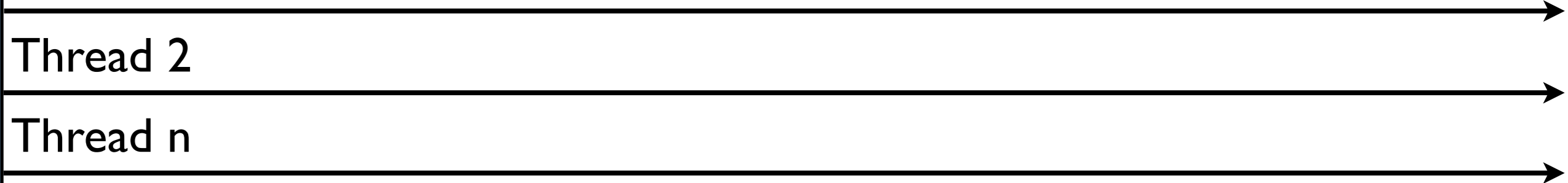
Enqueued  
Tasks



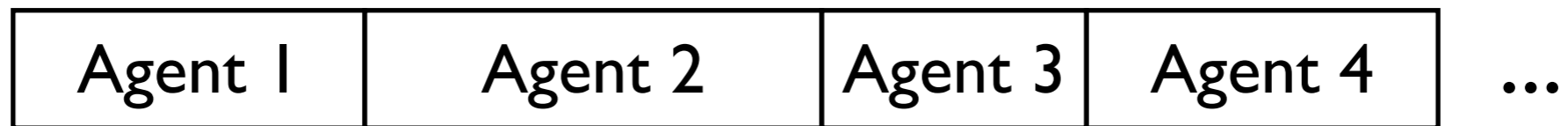
Thread 1

Thread 2

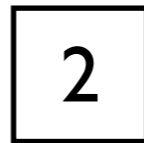
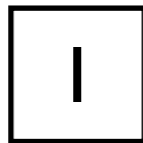
Thread n



# Frames



Enqueued  
Tasks



Task Pool

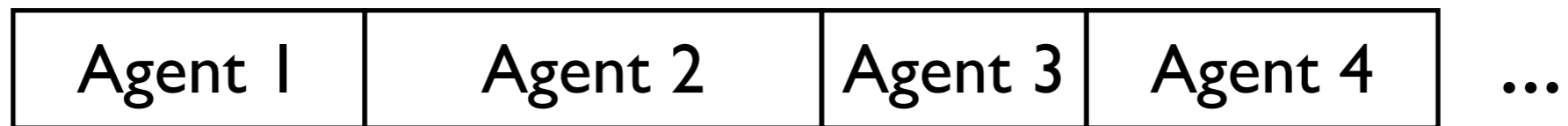
Thread 1

Thread 2

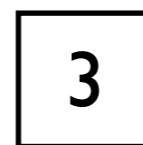
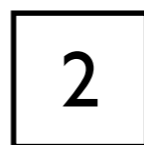
Thread n



# Frames



Enqueued  
Tasks



Thread 1

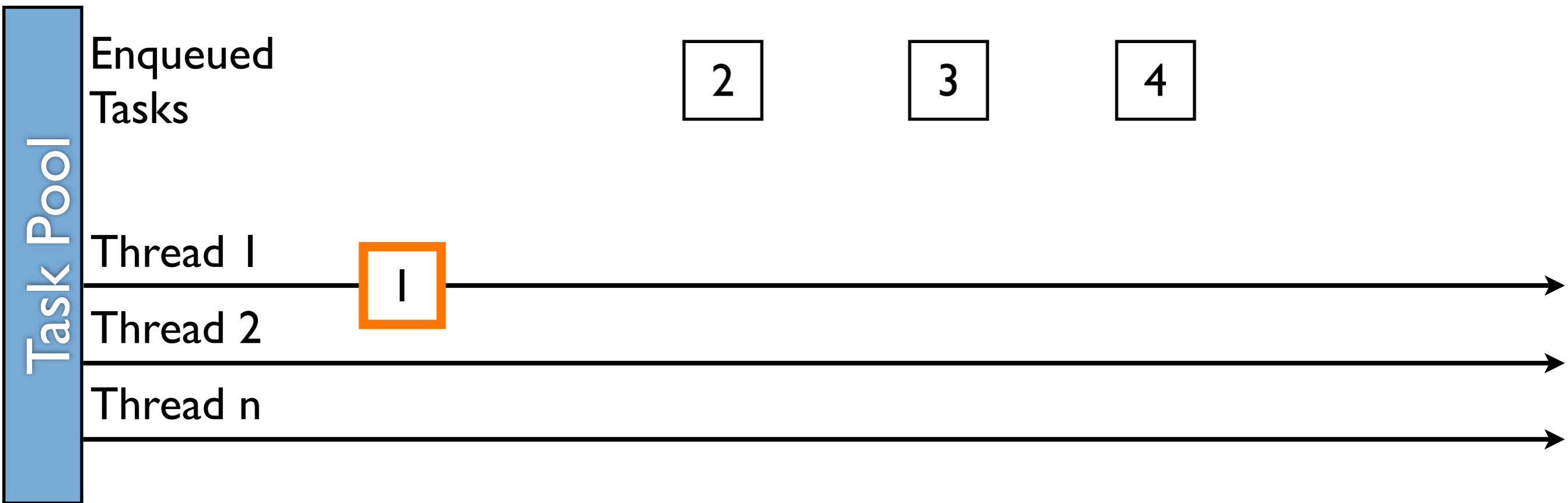
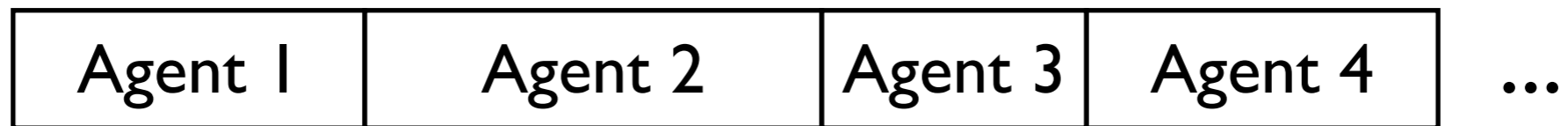
Thread 2

Thread n

Task Pool

Task pool schedules tasks onto free threads  
... for all tasks enqueued  
... observes queues

# Frames



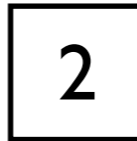
Task pool schedules tasks onto free threads  
... for all tasks enqueued  
... observes queues

# Frames



Task Pool

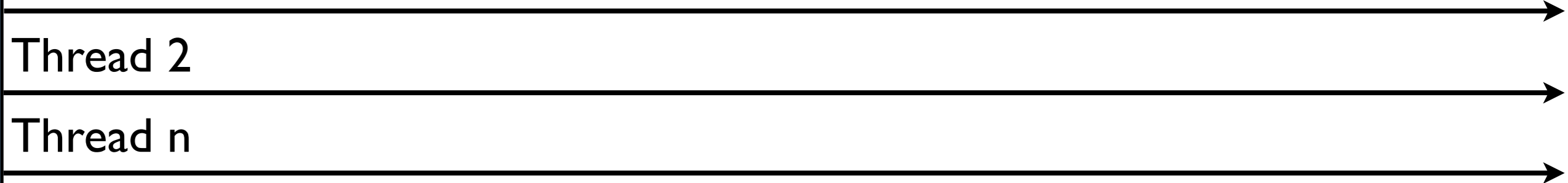
Enqueued  
Tasks



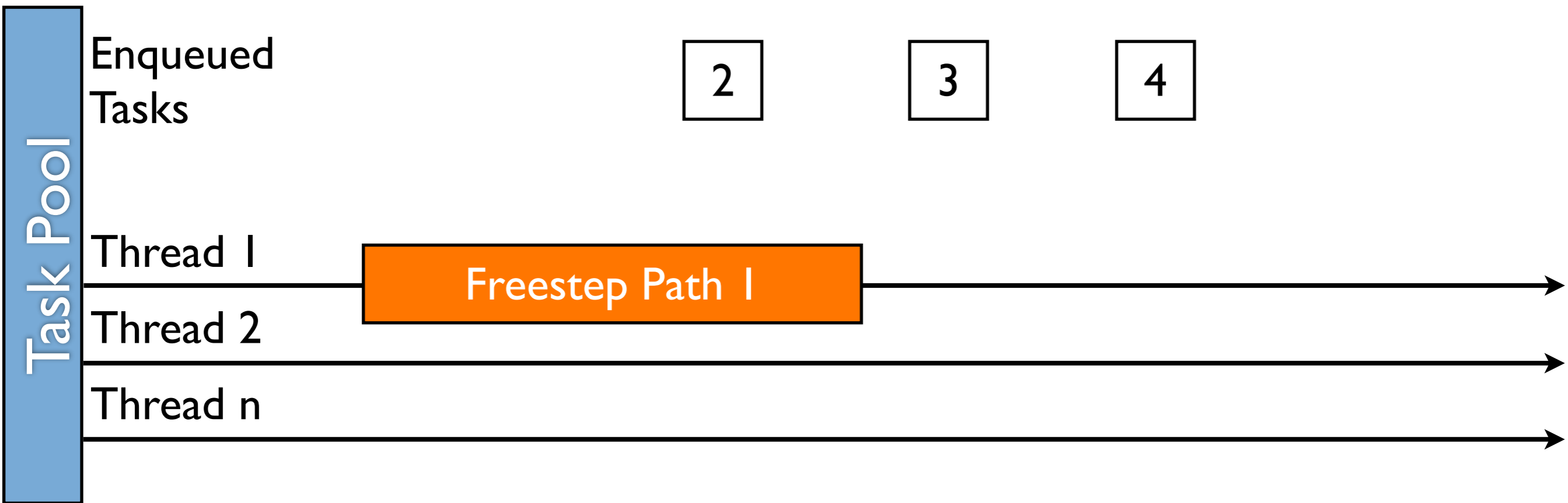
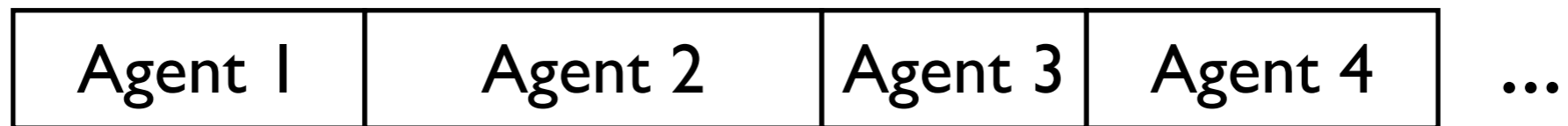
Thread 1

Thread 2

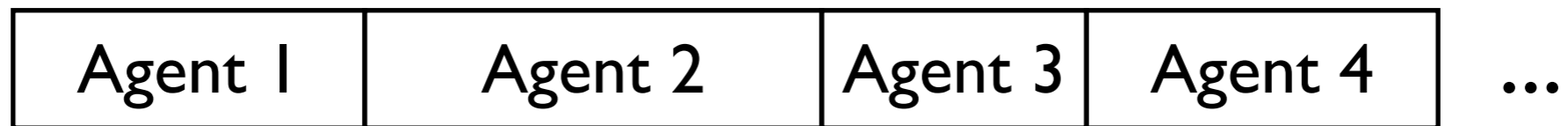
Thread n



# Frames



# Frames



Task Pool

Enqueued  
Tasks



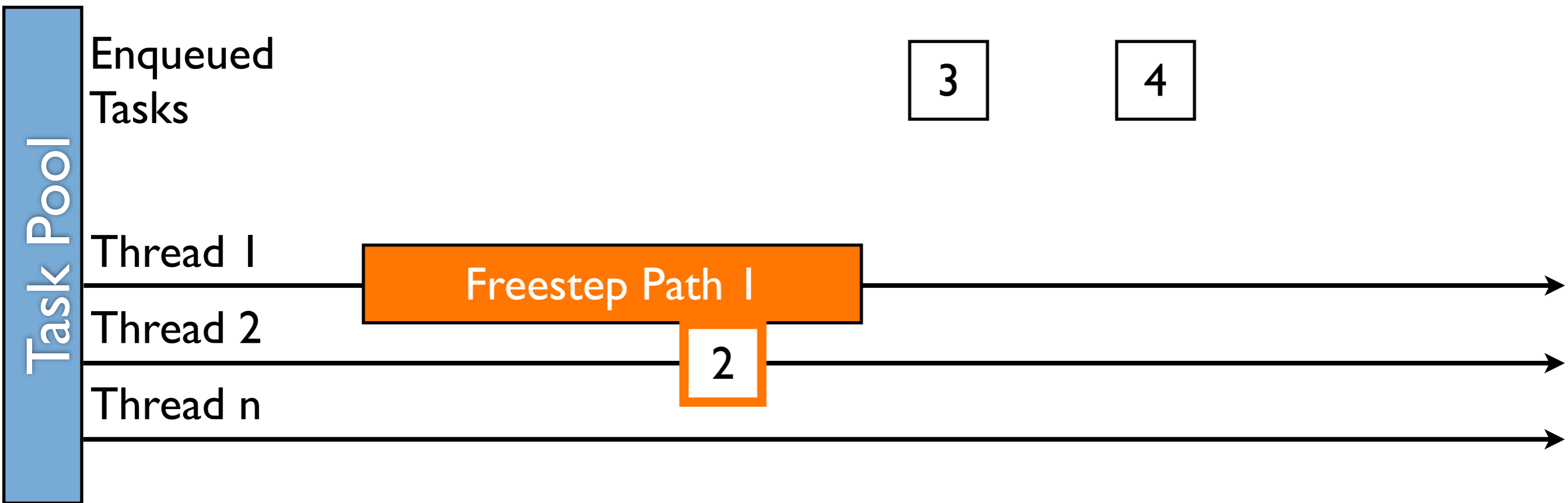
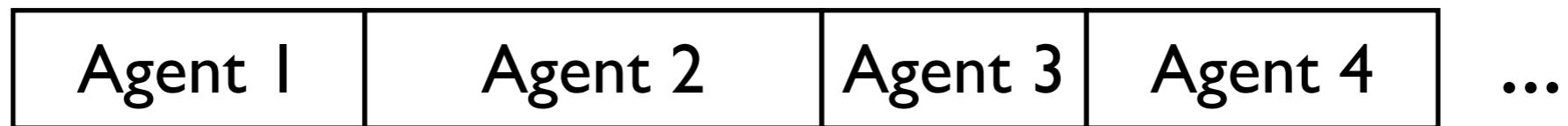
Thread 1



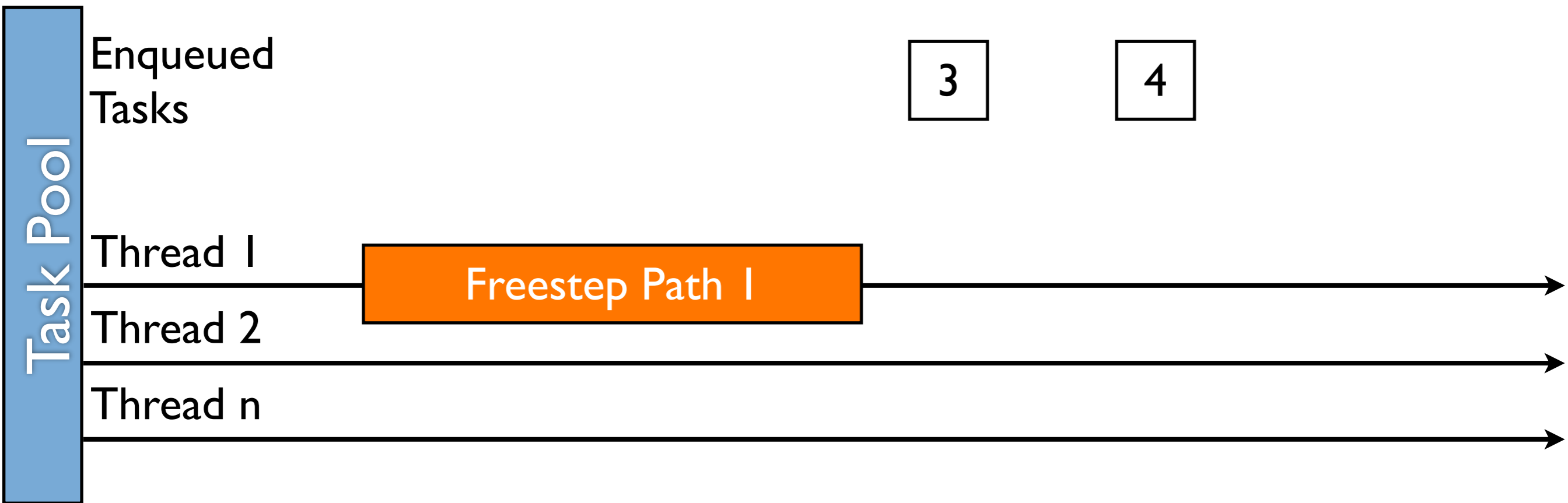
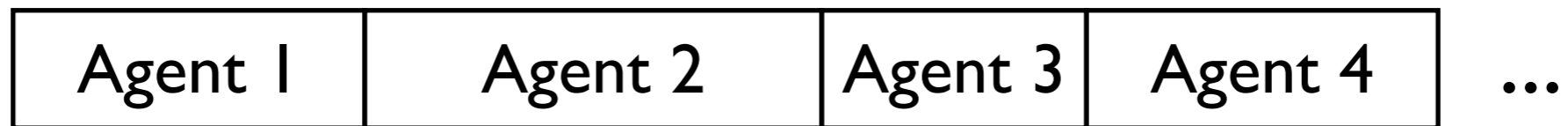
Thread 2

Thread n

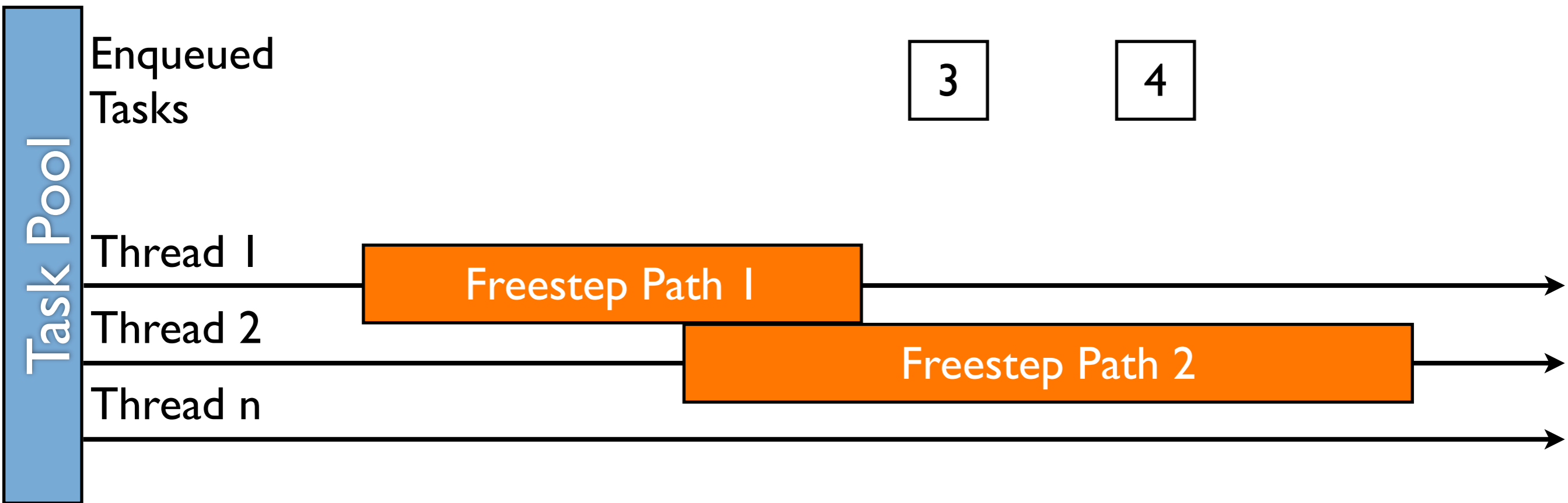
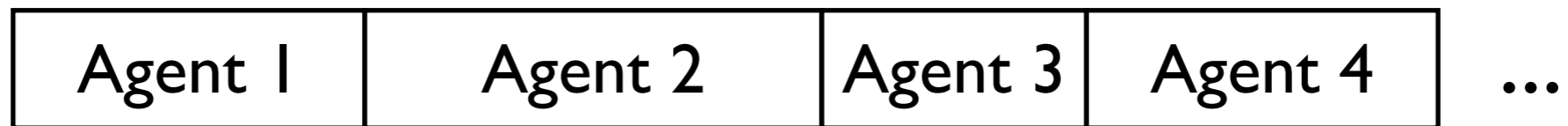
# Frames



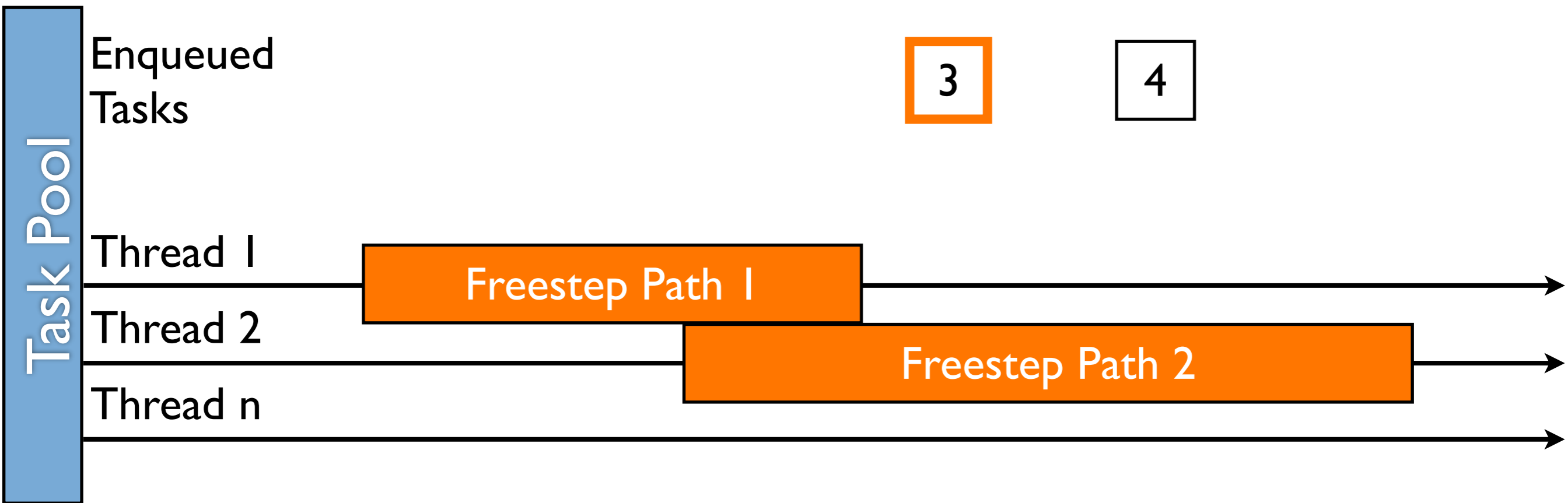
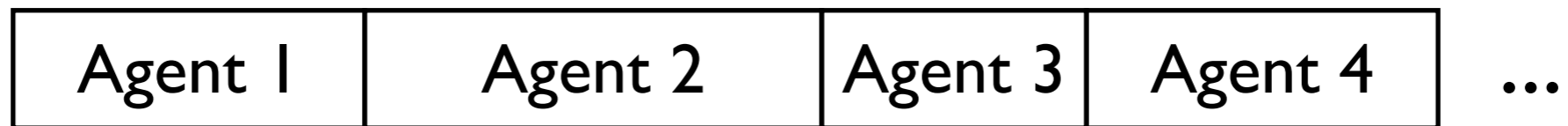
# Frames



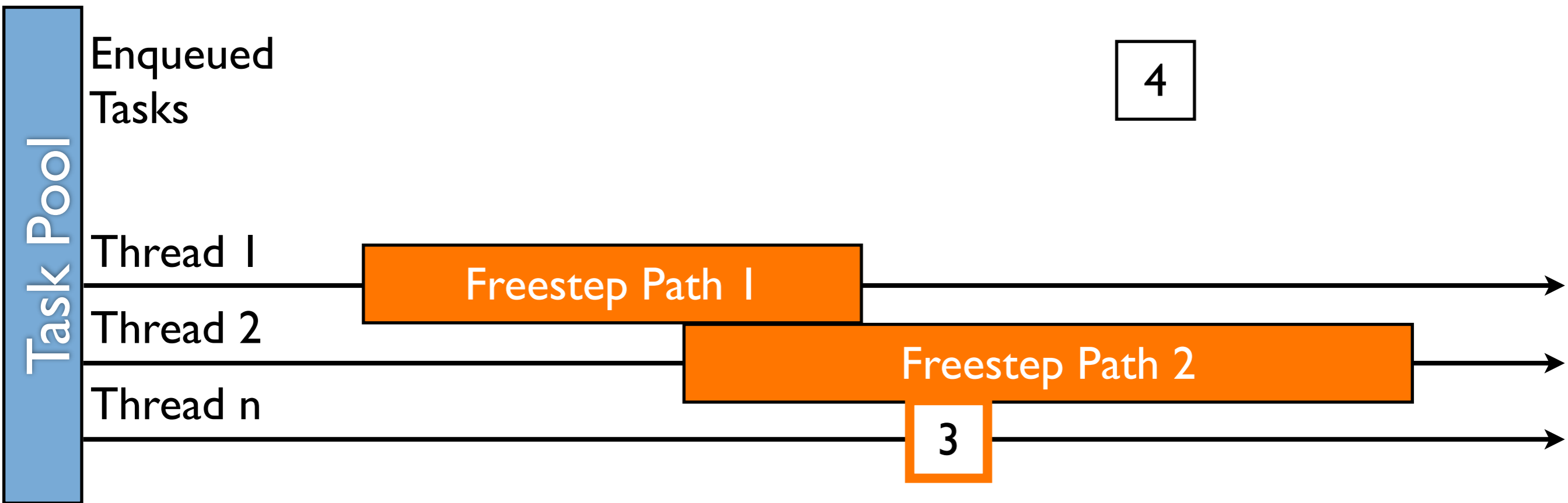
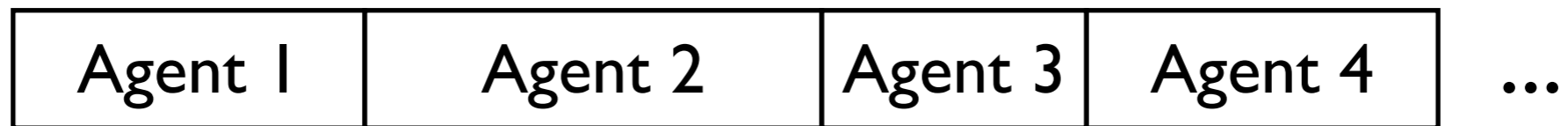
# Frames



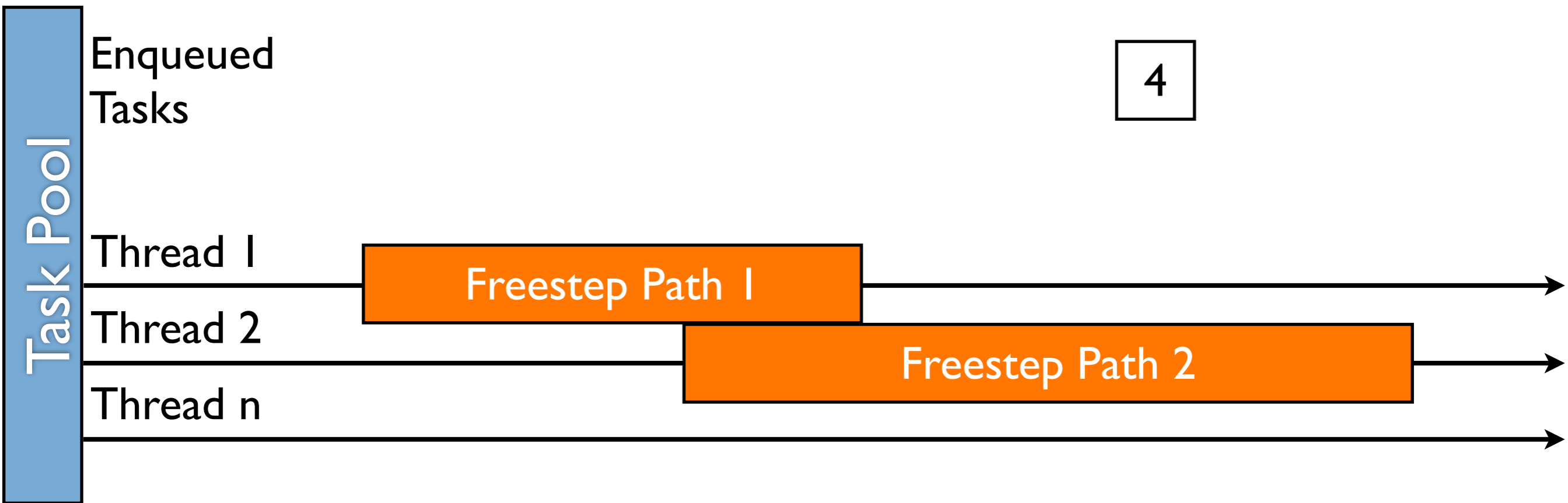
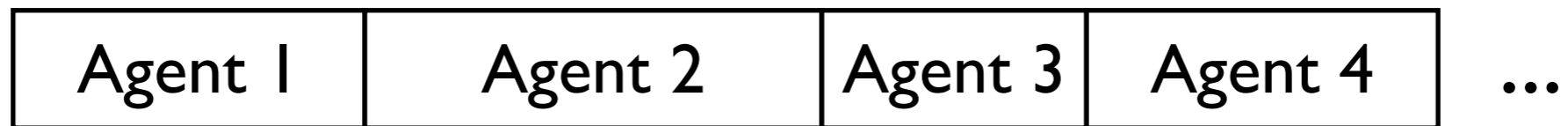
# Frames



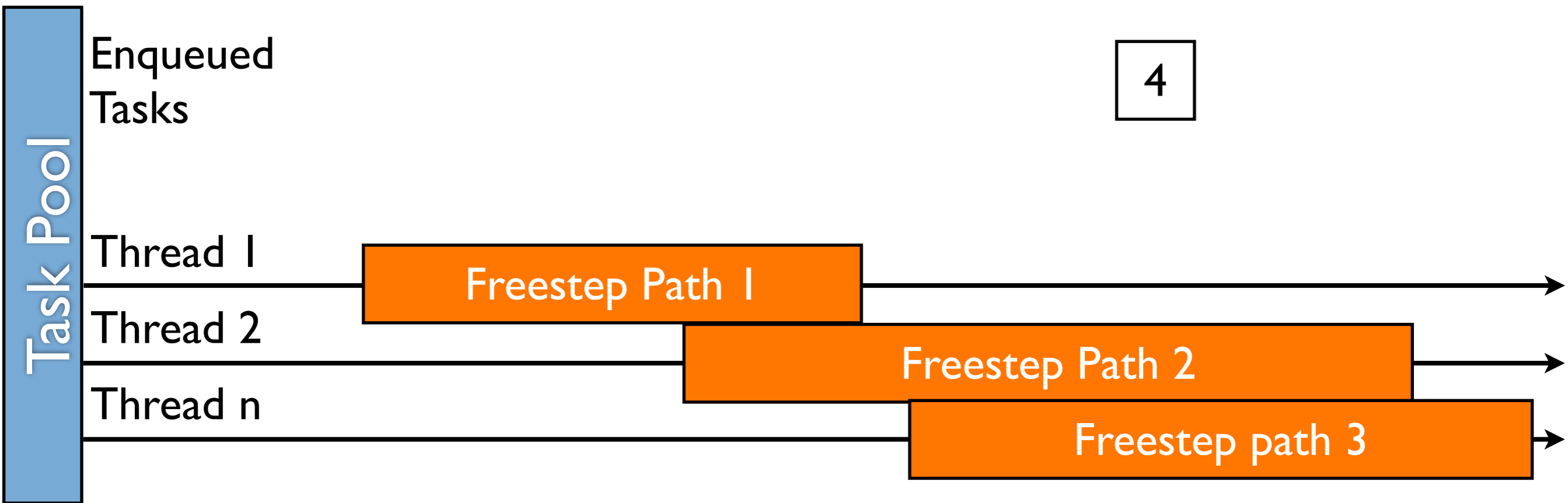
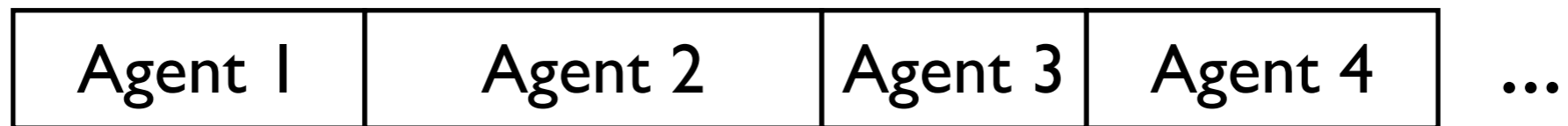
# Frames



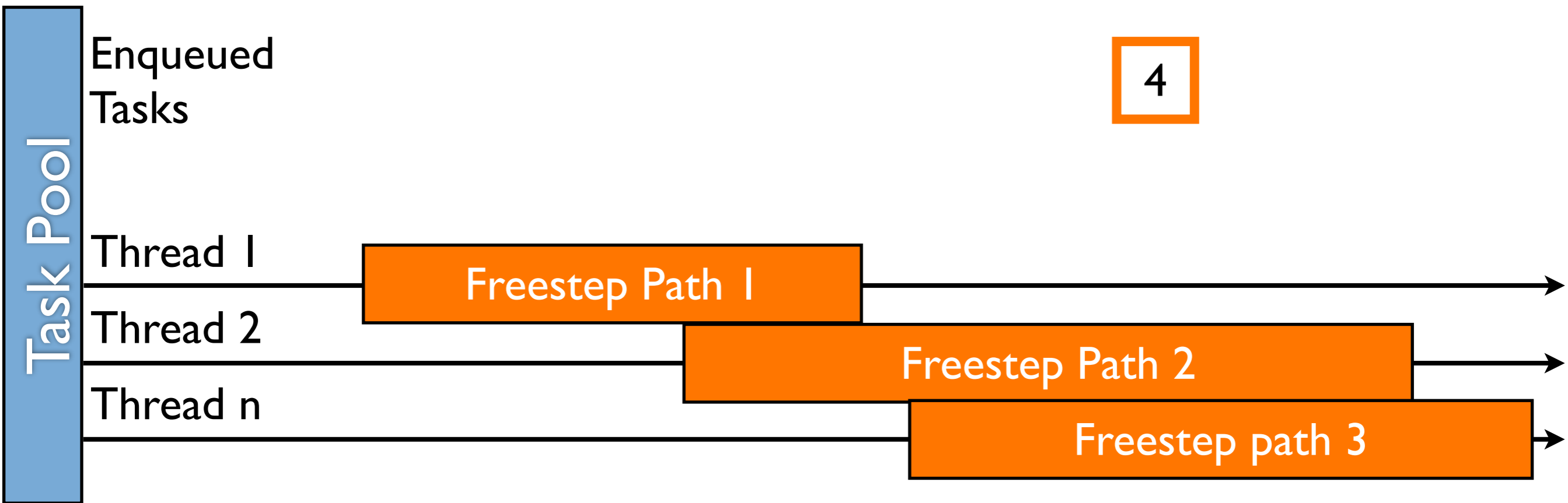
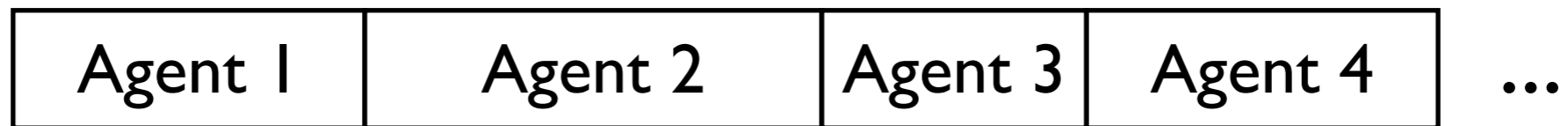
# Frames



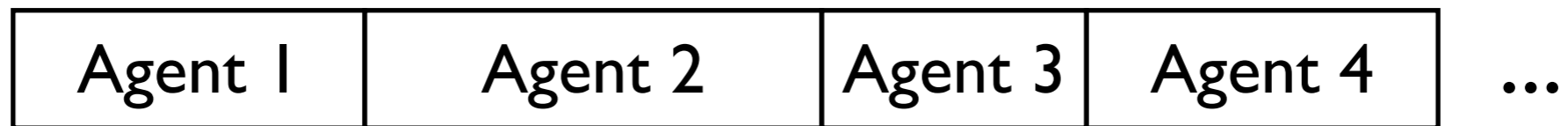
# Frames



# Frames



# Frames



## Task Pool

Enqueued  
Tasks

Thread 1



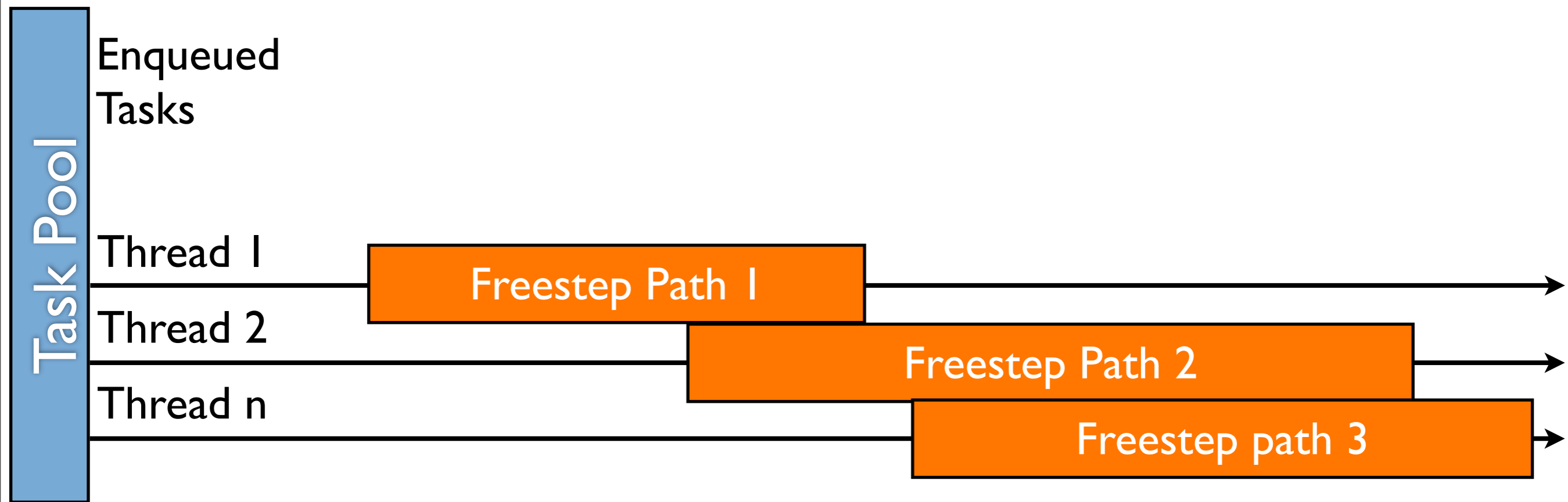
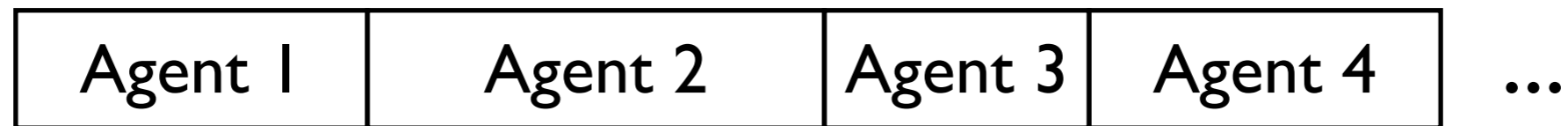
Thread 2



Thread n



# Frames

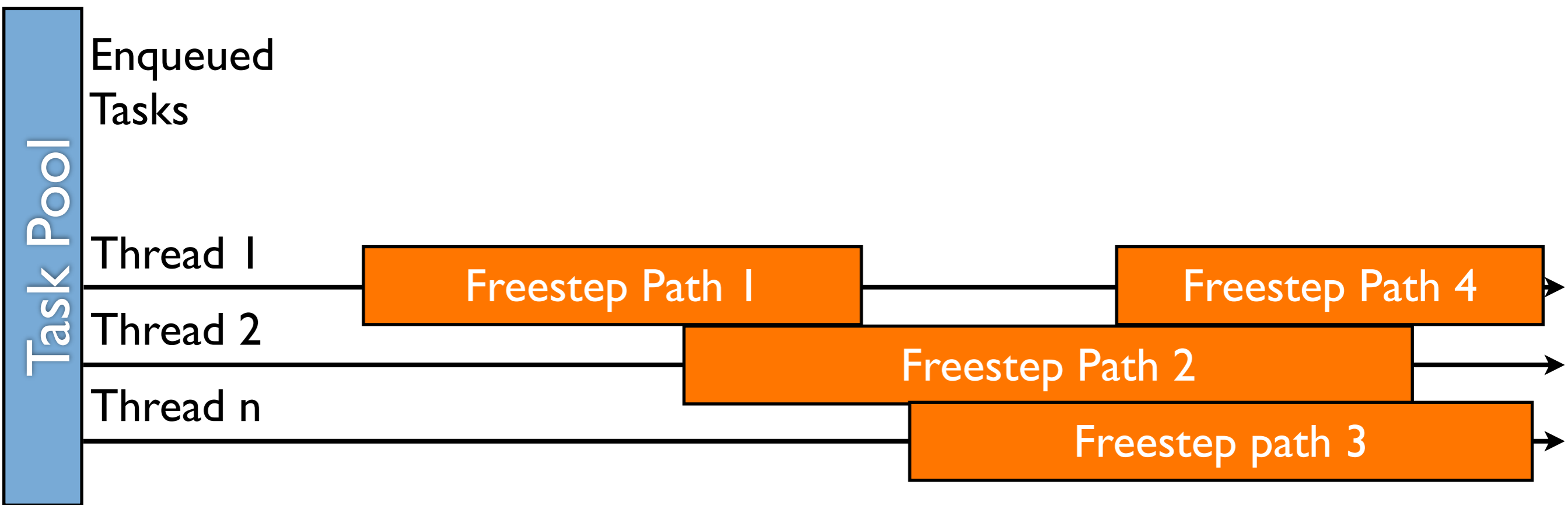
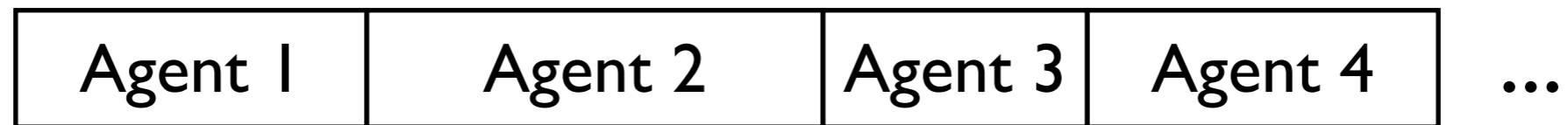


Scaling + automatic load balancing!!!

Still not deterministic -> lockstep approach with pathfinding iteration steps -> better bandwidth control with lockstep and systems

Task pools from now on -> or whatever helps to automatize SPU or GPU usage.

# Frames



Scaling + automatic load balancing!!!

Still not deterministic -> lockstep approach with pathfinding iteration steps -> better bandwidth control with lockstep and systems

Task pools from now on -> or whatever helps to automatize SPU or GPU usage.

# Frames

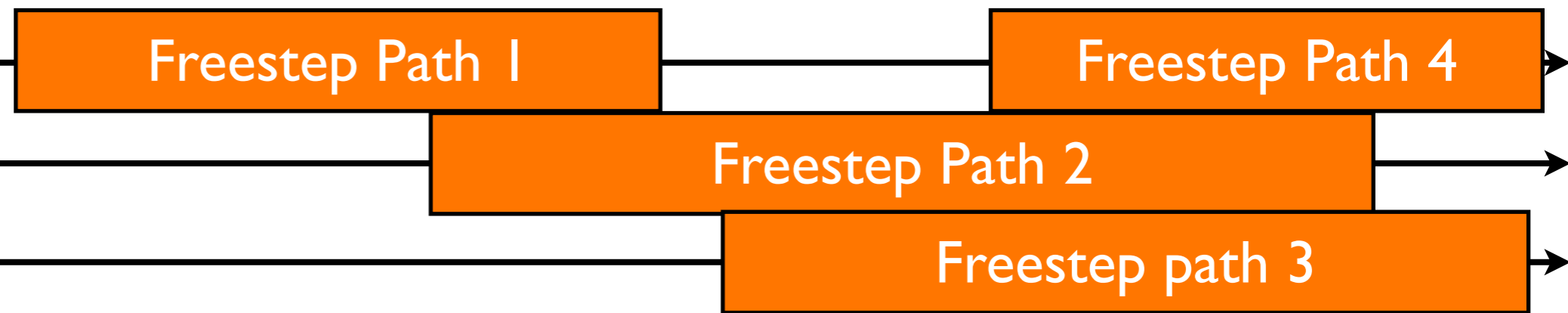


No raw threads  
anymore  
from now on!

## Task Pool

Enqueued  
Tasks

Thread 1  
Thread 2  
Thread n



Scaling + automatic load balancing!!!  
Still not deterministic -> lockstep approach with pathfinding iteration steps -> better bandwidth control with lockstep and systems  
Task pools from now on -> or whatever helps to automatize SPU or GPU usage.

# Synchronization

Futures  
Events / messages  
Polling of a pathfinding system  
etc.

# Future

# Future



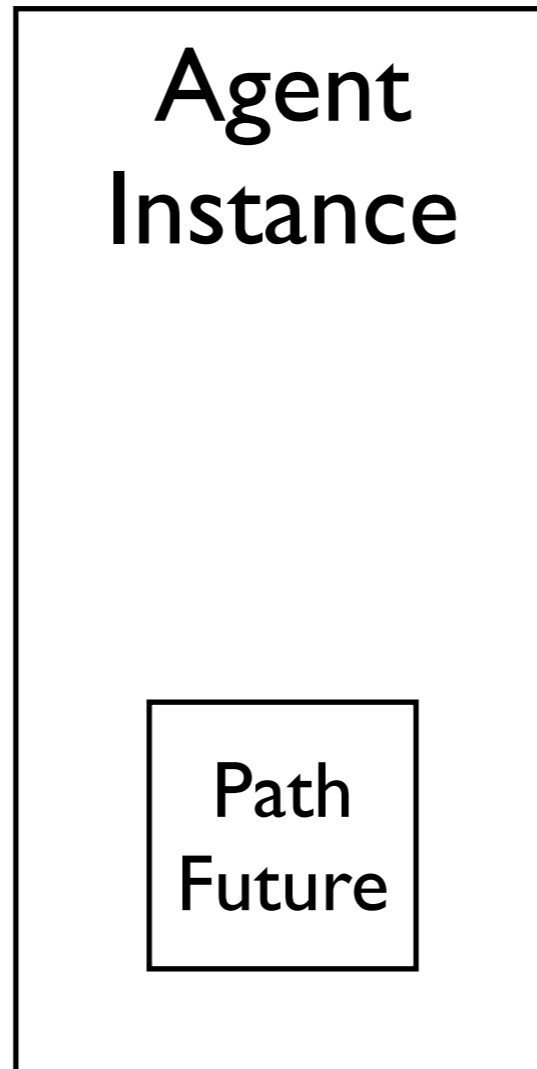
Agent  
Instance

# Future

Agent  
Instance

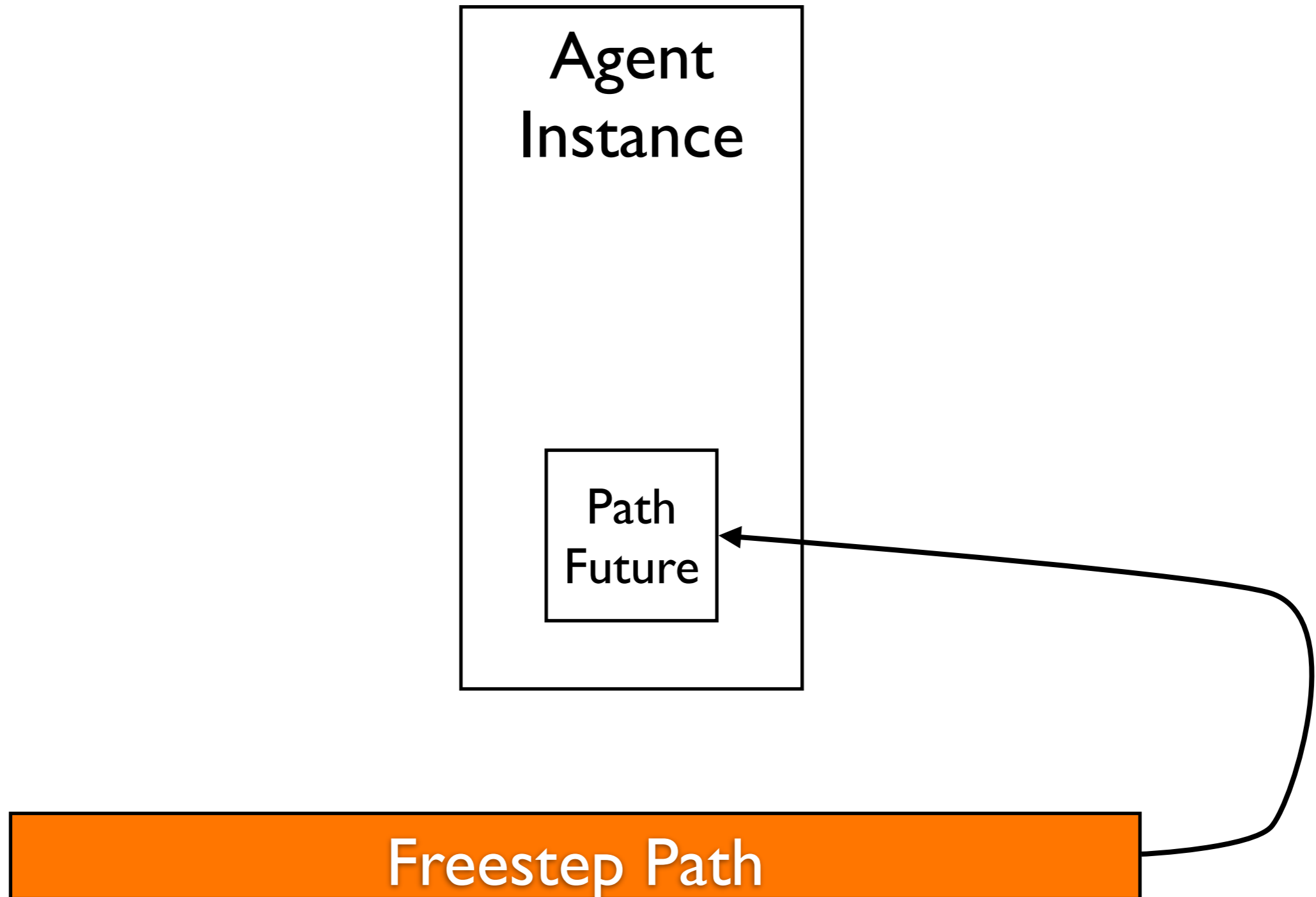
Path  
Future

# Future

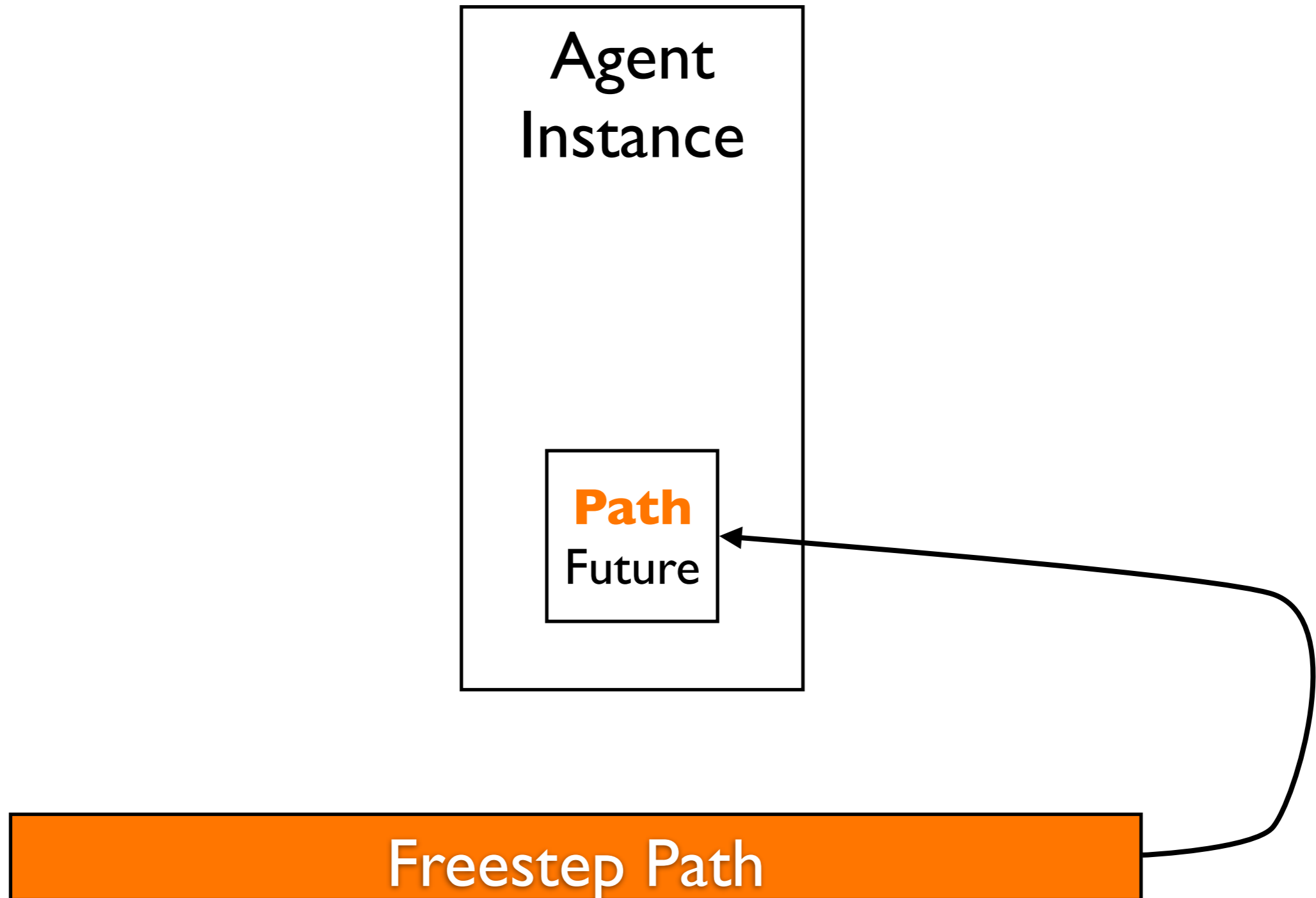


Freestep Path

# Future



# Future



# Event / Message

React to messages in the next frame!!

With lockstep system deterministic

Or polling

Enqueue path result or only message to pull path result from inside the path system

# Event / Message



Agent  
Instance

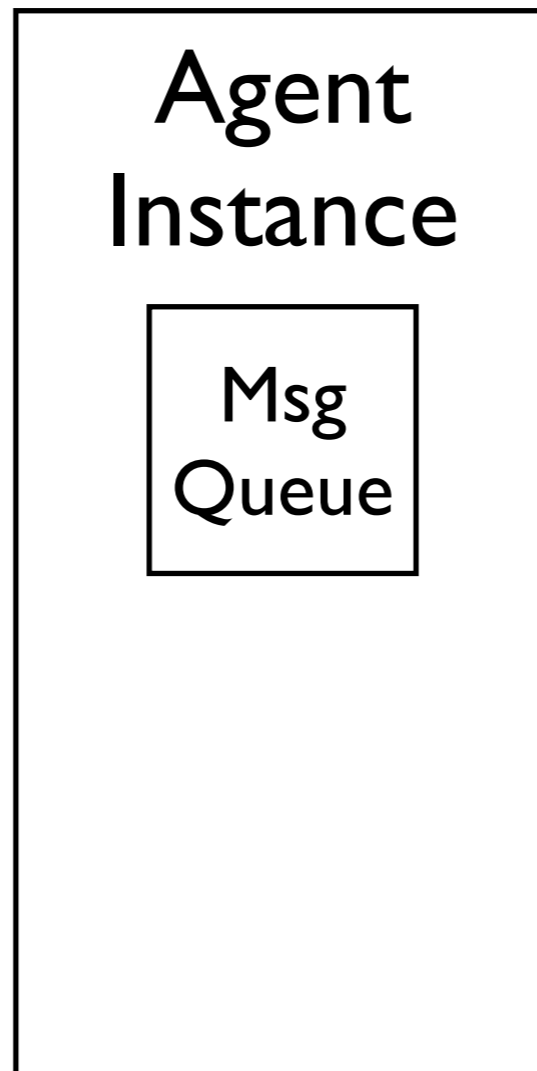
React to messages in the next frame!!

With lockstep system deterministic

Or polling

Enqueue path result or only message to pull path result from inside the path system

# Event / Message



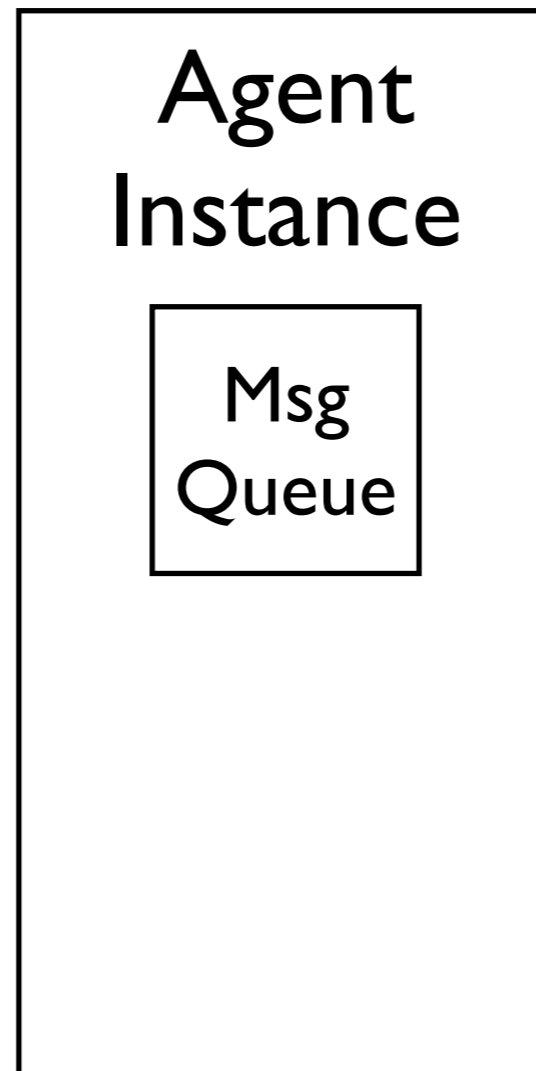
React to messages in the next frame!!

With lockstep system deterministic

Or polling

Enqueue path result or only message to pull path result from inside the path system

# Event / Message



Freestep Path

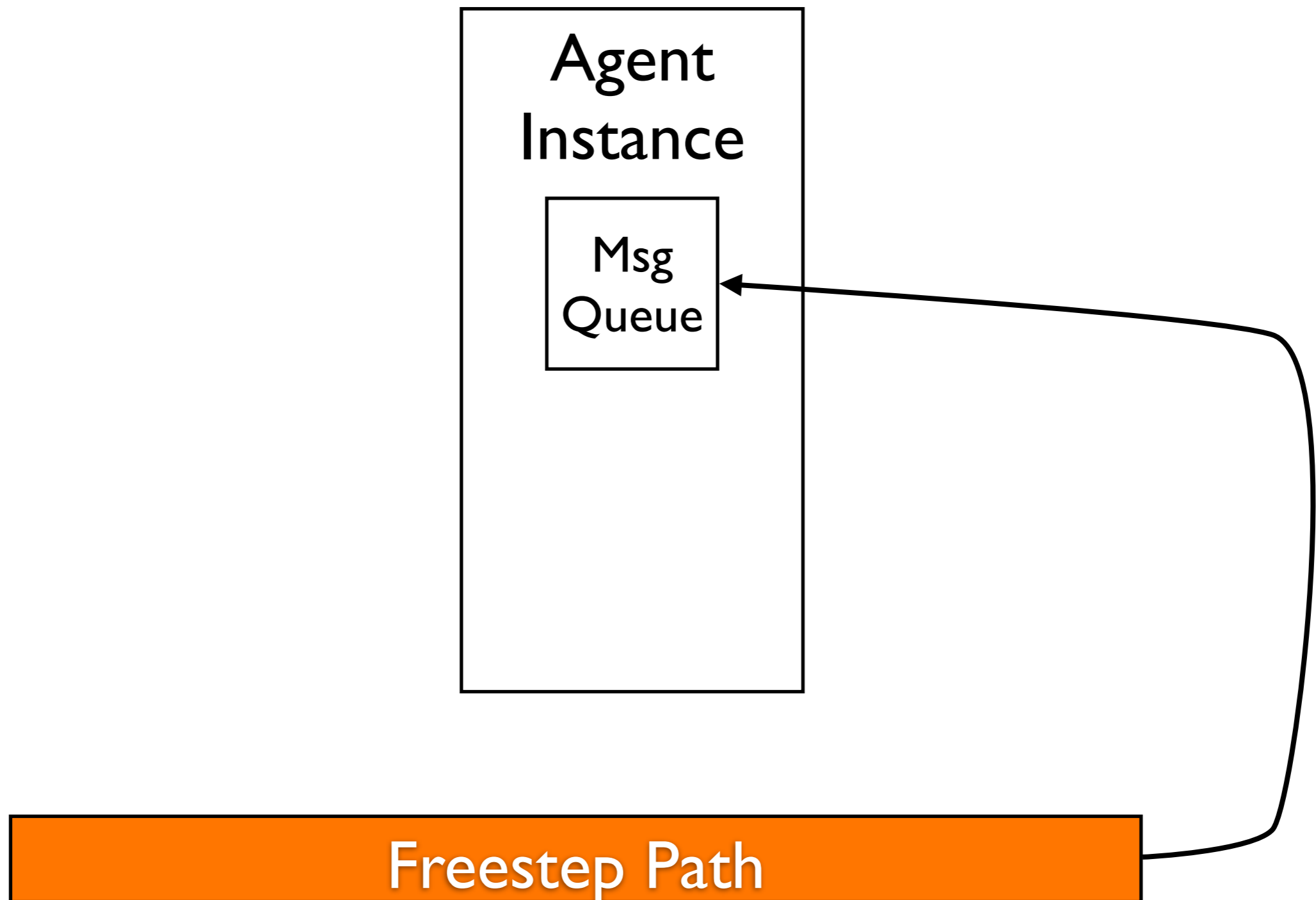
React to messages in the next frame!!

With lockstep system deterministic

Or polling

Enqueue path result or only message to pull path result from inside the path system

# Event / Message



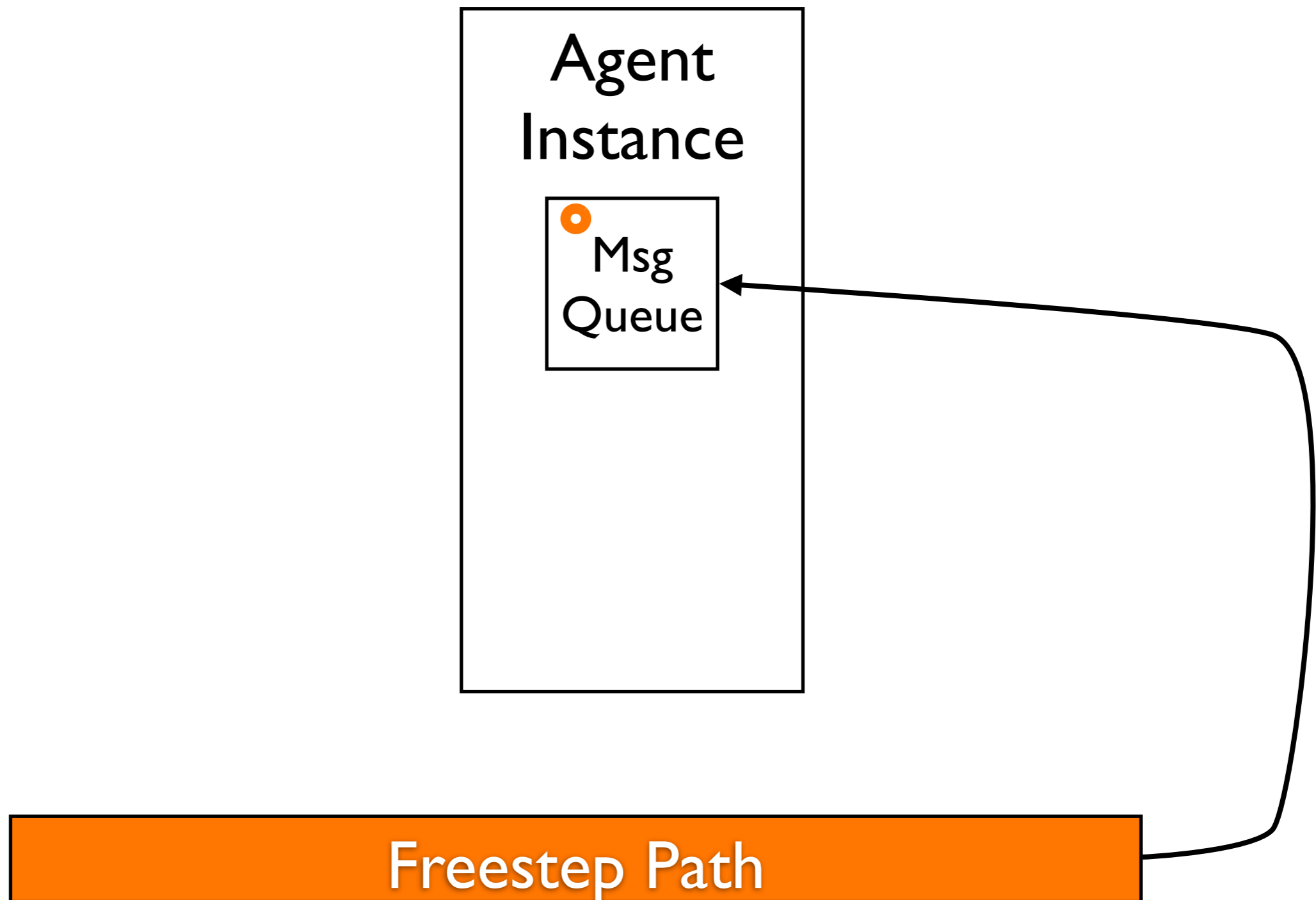
React to messages in the next frame!!

With lockstep system deterministic

Or polling

Enqueue path result or only message to pull path result from inside the path system

# Event / Message



React to messages in the next frame!!

With lockstep system deterministic

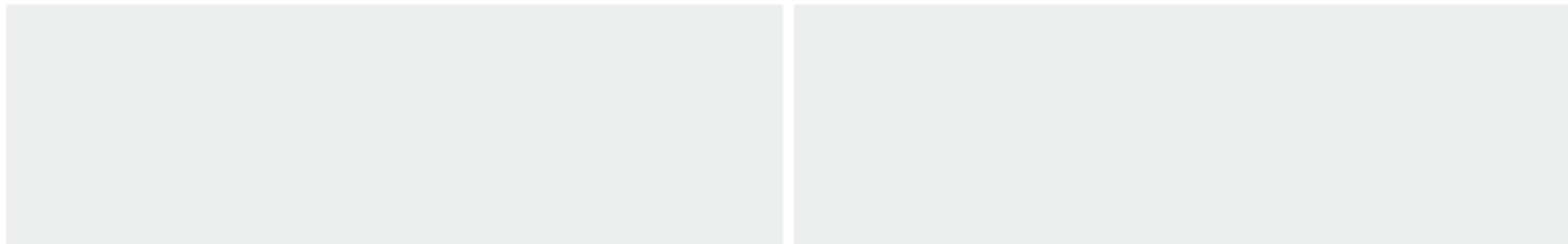
Or polling

Enqueue path result or only message to pull path result from inside the path system

# Asynchronous Calls

Pros

Cons



# Asynchronous Calls

Pros

Cons

Scaling (with Task Pool)

Potentially better  
memory locality

# Asynchronous Calls

## Pros

Scaling (with Task Pool)

Potentially better  
memory locality

## Cons

Determinism needs  
care

Get syncing right

Beware of side effects

# Asynchronous Calls

Avoid raw threads!

Pros

Cons

Scaling (with Task Pool)

Determinism needs care

Potentially better memory locality

Get syncing right

Beware of side effects

What if seq. updates take too long?

We still have lots of untapped parallelism left (cores to exploit)

# 2.

# Parallel Agents

What if seq. updates take too long?

We still have lots of untapped parallelism left (cores to exploit)

Look into one agent update step

We process agents in a sequential loop -> opportunity for parallelism!!!!

Agents as tasks

# Frames



Look into one agent update step

We process agents in a sequential loop -> opportunity for parallelism!!!!

Agents as tasks

# Frames

|



Update  
per time  
step

Look into one agent update step

We process agents in a sequential loop -> opportunity for parallelism!!!!

Agents as tasks

# Frames

|



Update  
per time  
step



Look into one agent update step

We process agents in a sequential loop -> opportunity for parallelism!!!!

Agents as tasks

Frames

|



Agent instances

Update per time step



Look into one agent update step

We process agents in a sequential loop -> opportunity for parallelism!!!!

Agents as tasks

# Frames

1



Agent instances



Update per time step

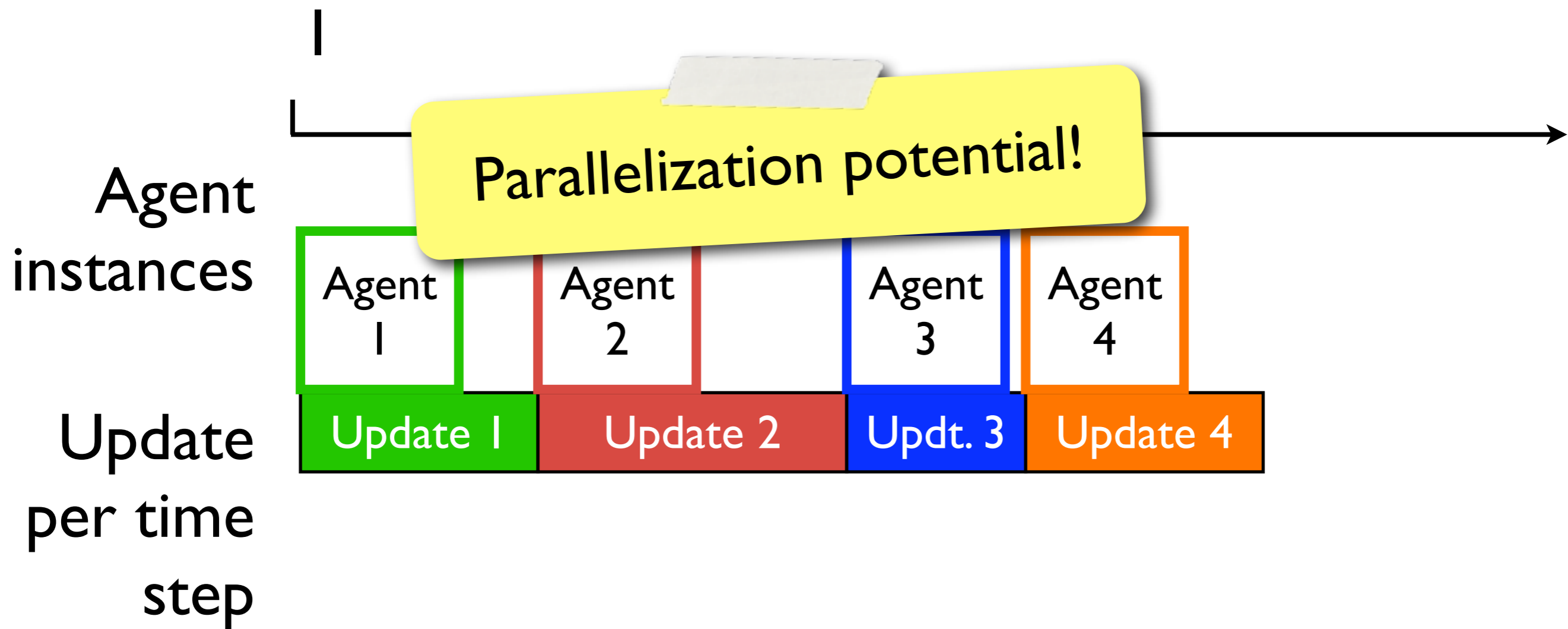


Look into one agent update step

We process agents in a sequential loop -> opportunity for parallelism!!!!

Agents as tasks

# Frames

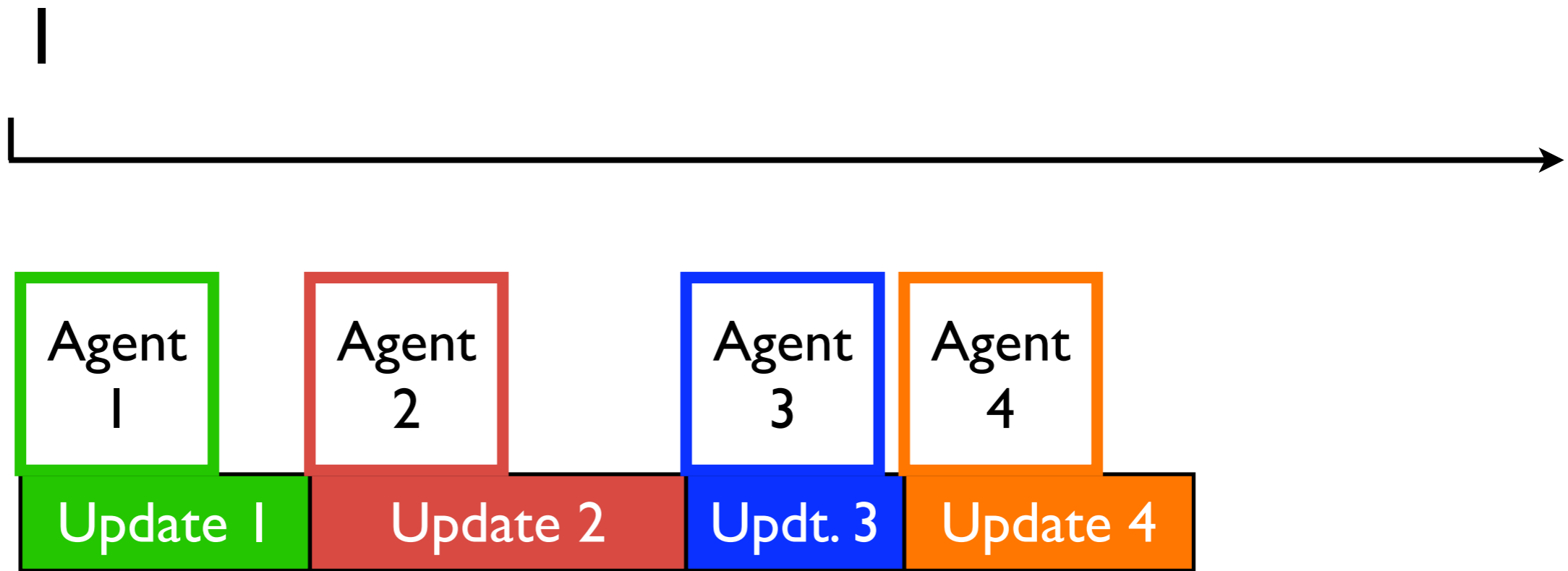


Look into one agent update step

We process agents in a sequential loop -> opportunity for parallelism!!!!

Agents as tasks

# Frames

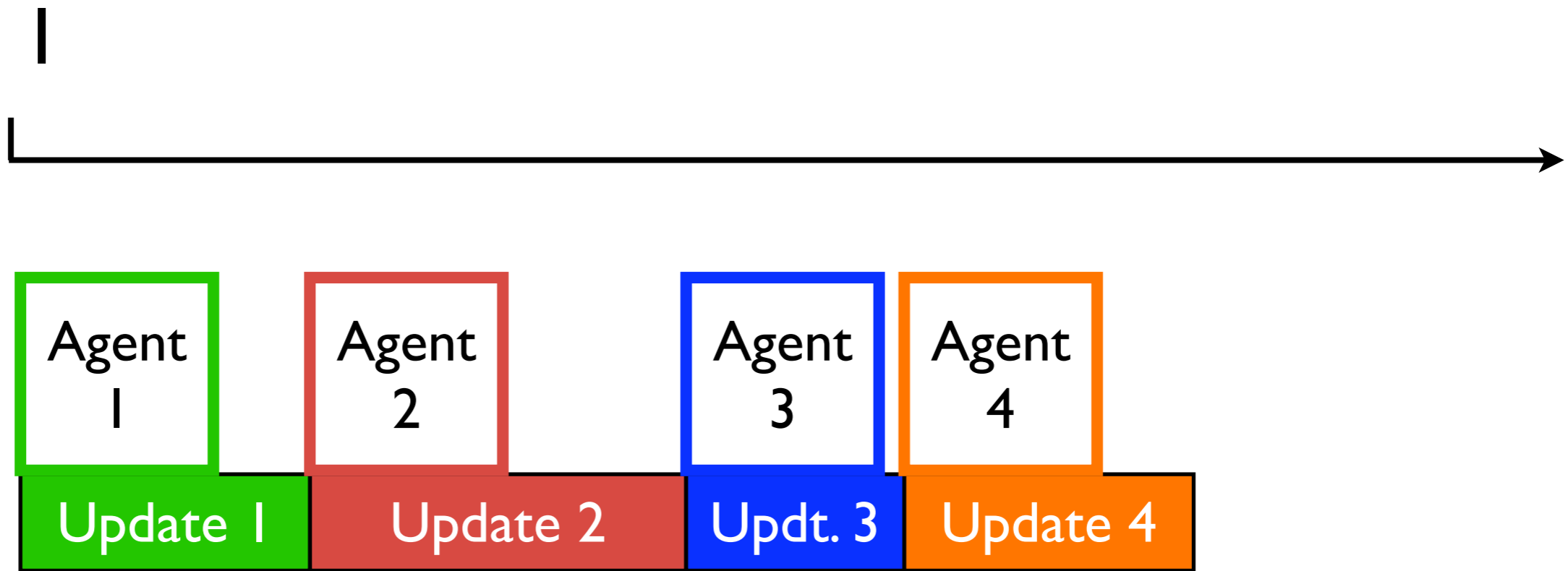


Think message passing + no shared mem to steer thinking!!!!!!!!!!!!!!

Task + message passing

Task pool threads or on SPU or GPU or whatever -> scaling + automatic load balance

# Frames



Task Pool Thread n

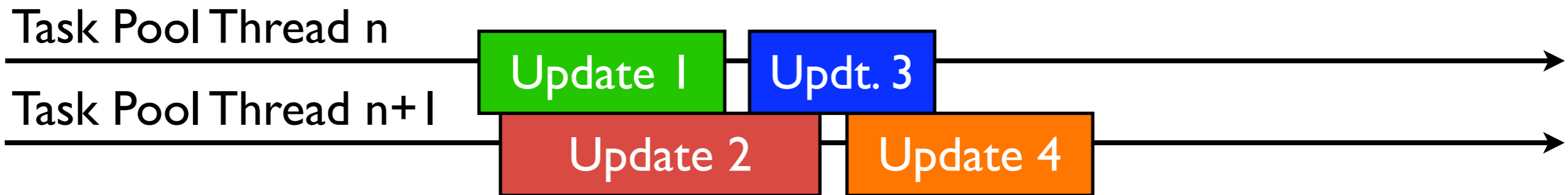
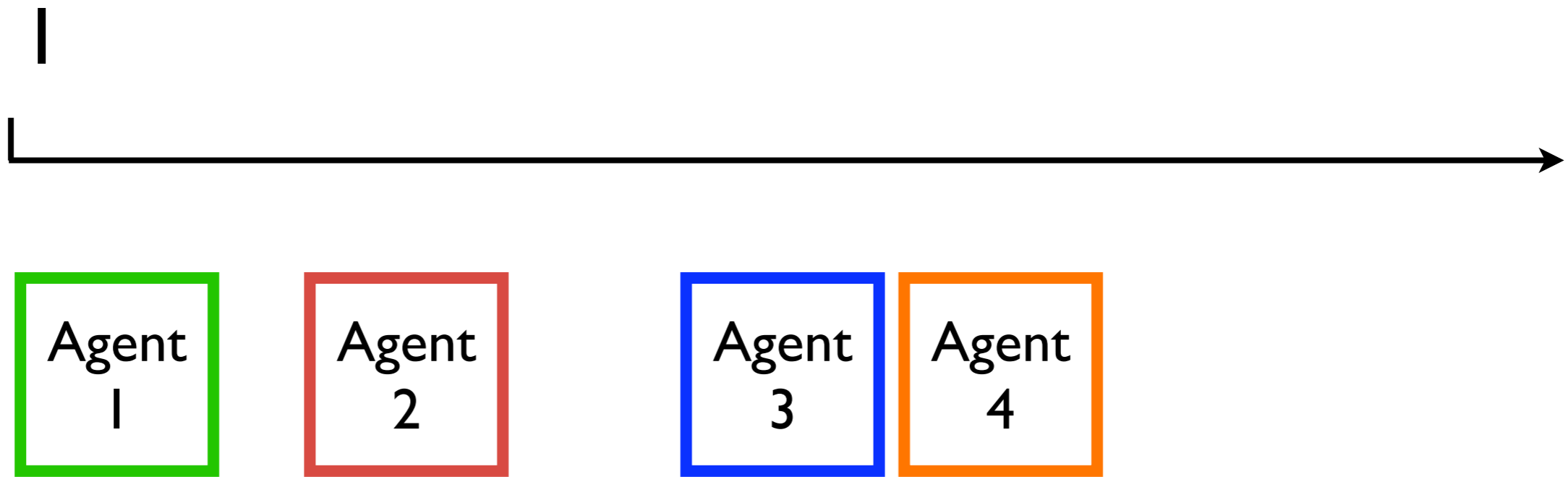
Task Pool Thread n+1

Think message passing + no shared mem to steer thinking!!!!!!!!!!!!!!

Task + message passing

Task pool threads or on SPU or GPU or whatever -> scaling + automatic load balance

# Frames



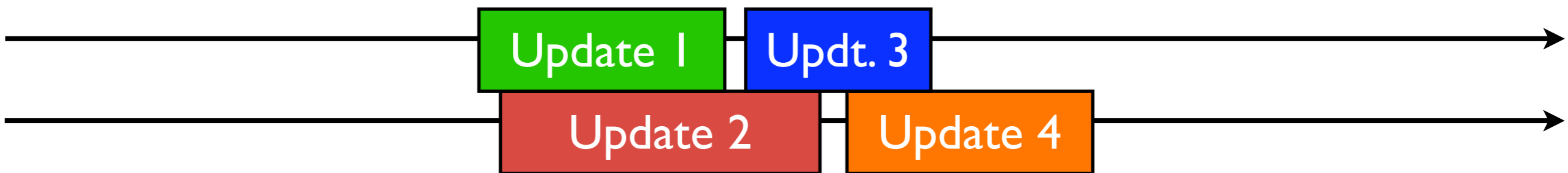
Think message passing + no shared mem to steer thinking!!!!!!!!!!!!!!

Task + message passing

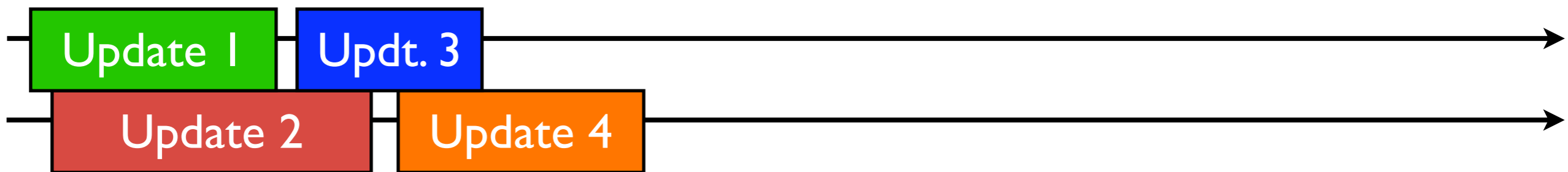
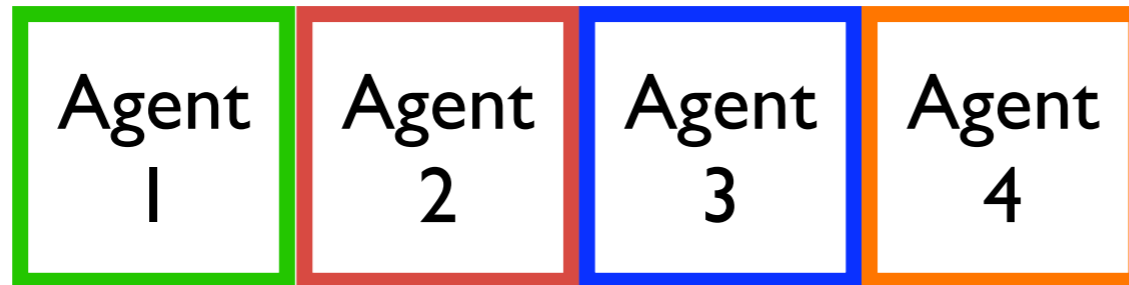
Task pool threads or on SPU or GPU or whatever -> scaling + automatic load balance

# Frames

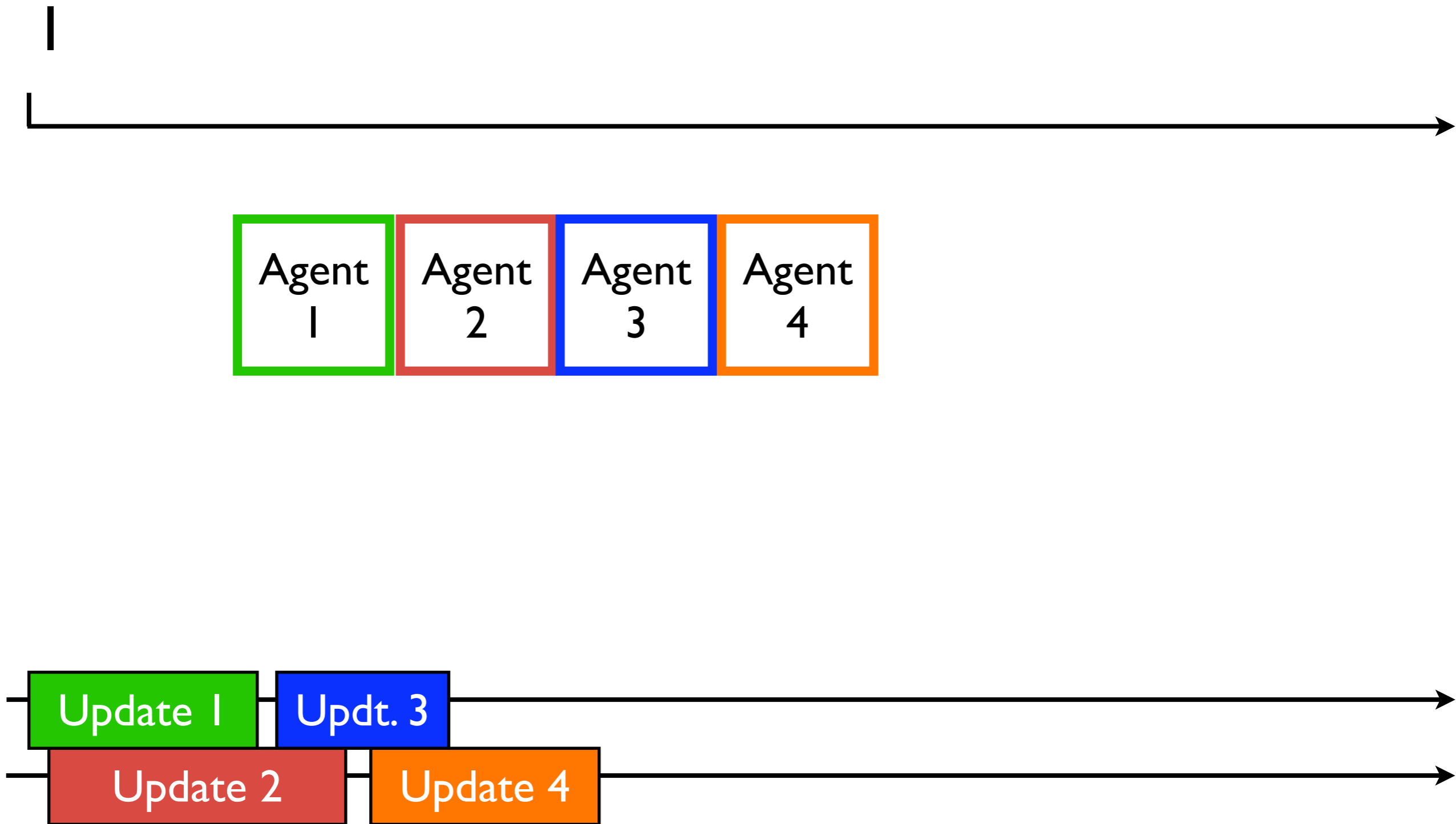
|



# Frames

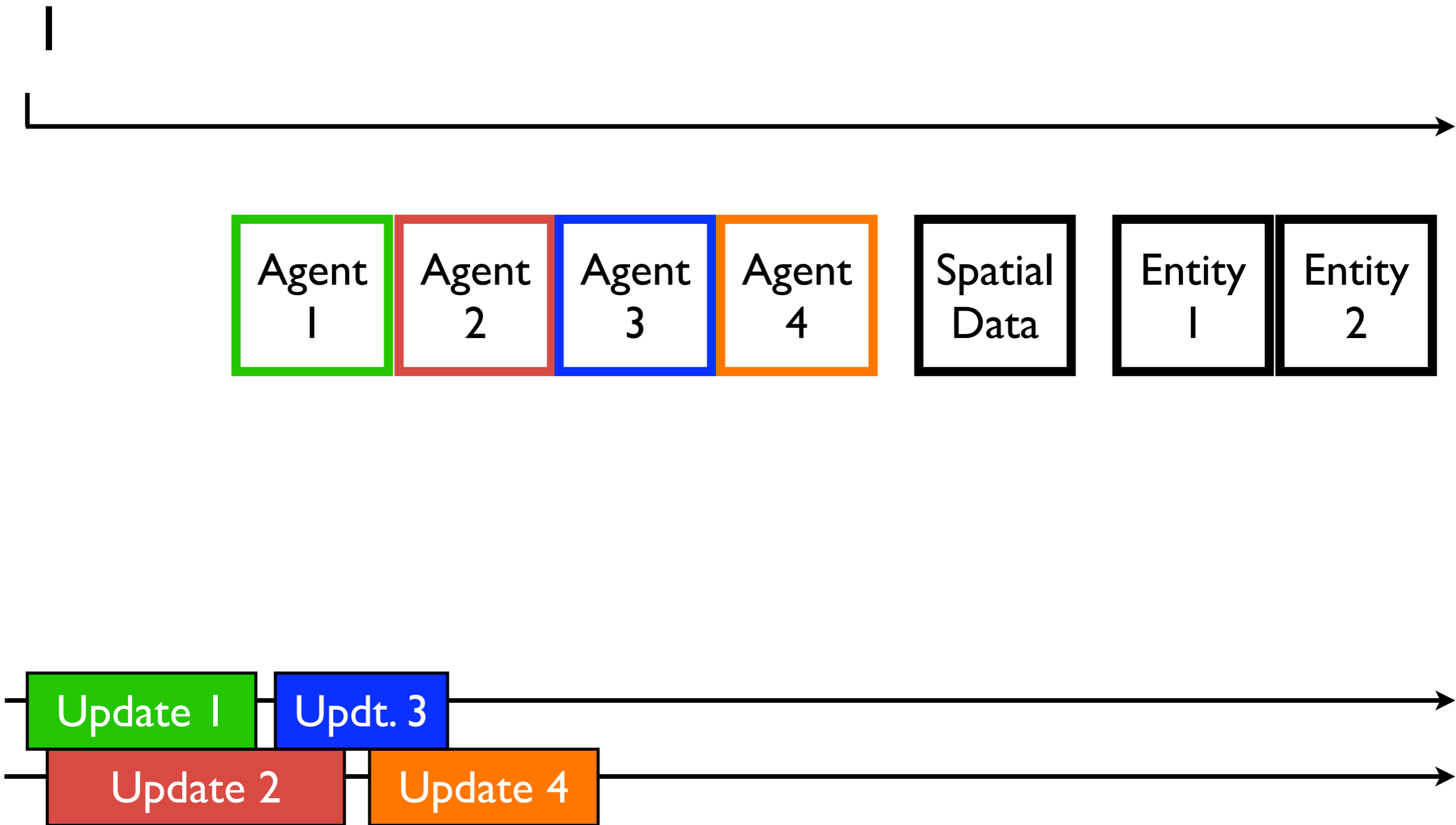


# Frames



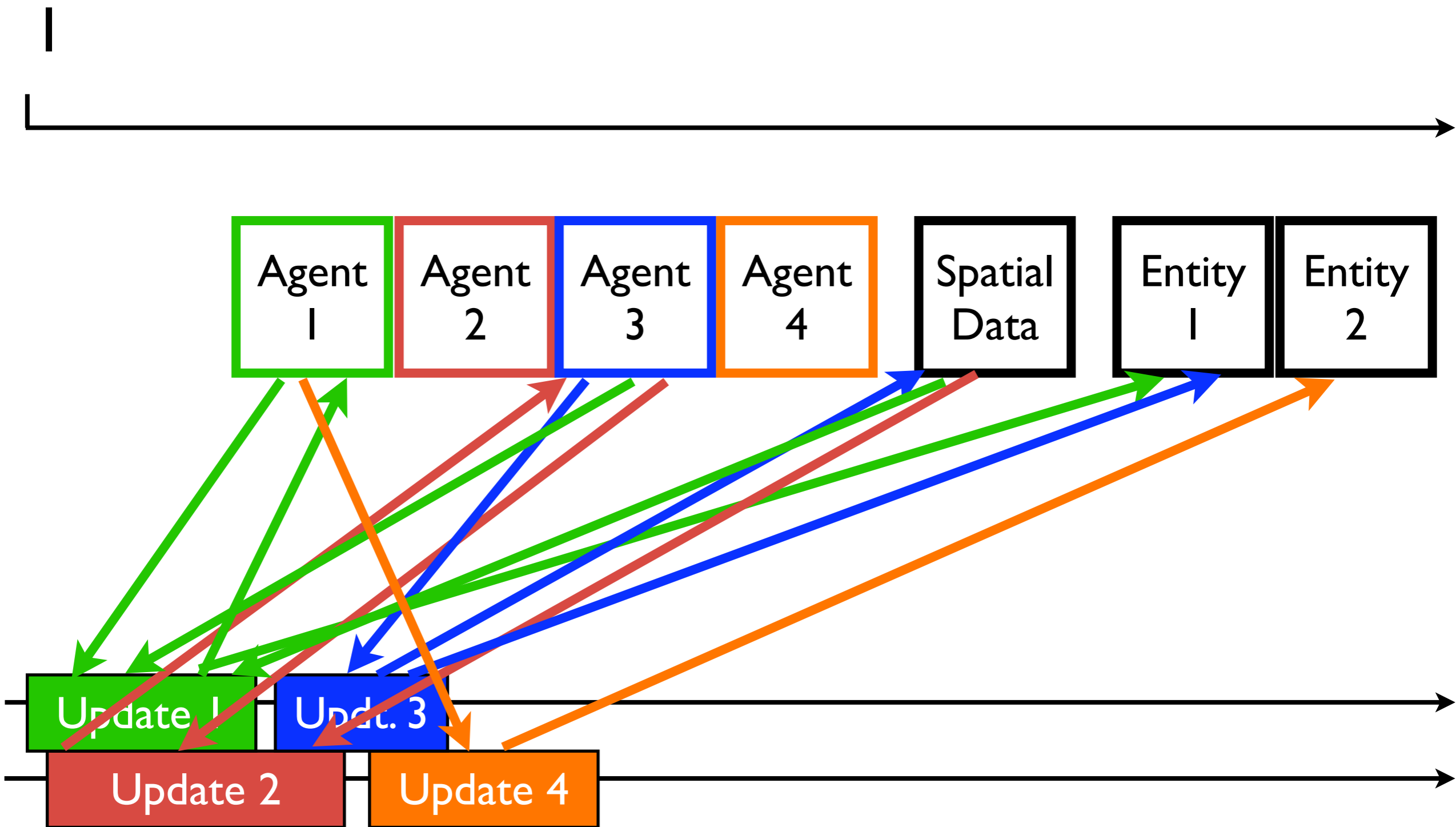
Shared data structures like spatial data and accessing / using other game world entities  
-> unorganized - bad for **bandwidth** + **locality** bad and not **deterministic**  
-> syncing needed -> convoying -> **contention** -> serialization -> blocking bad for task pools  
... if sequential solution was order dependent, then we are non-deterministic and need refactoring

# Frames



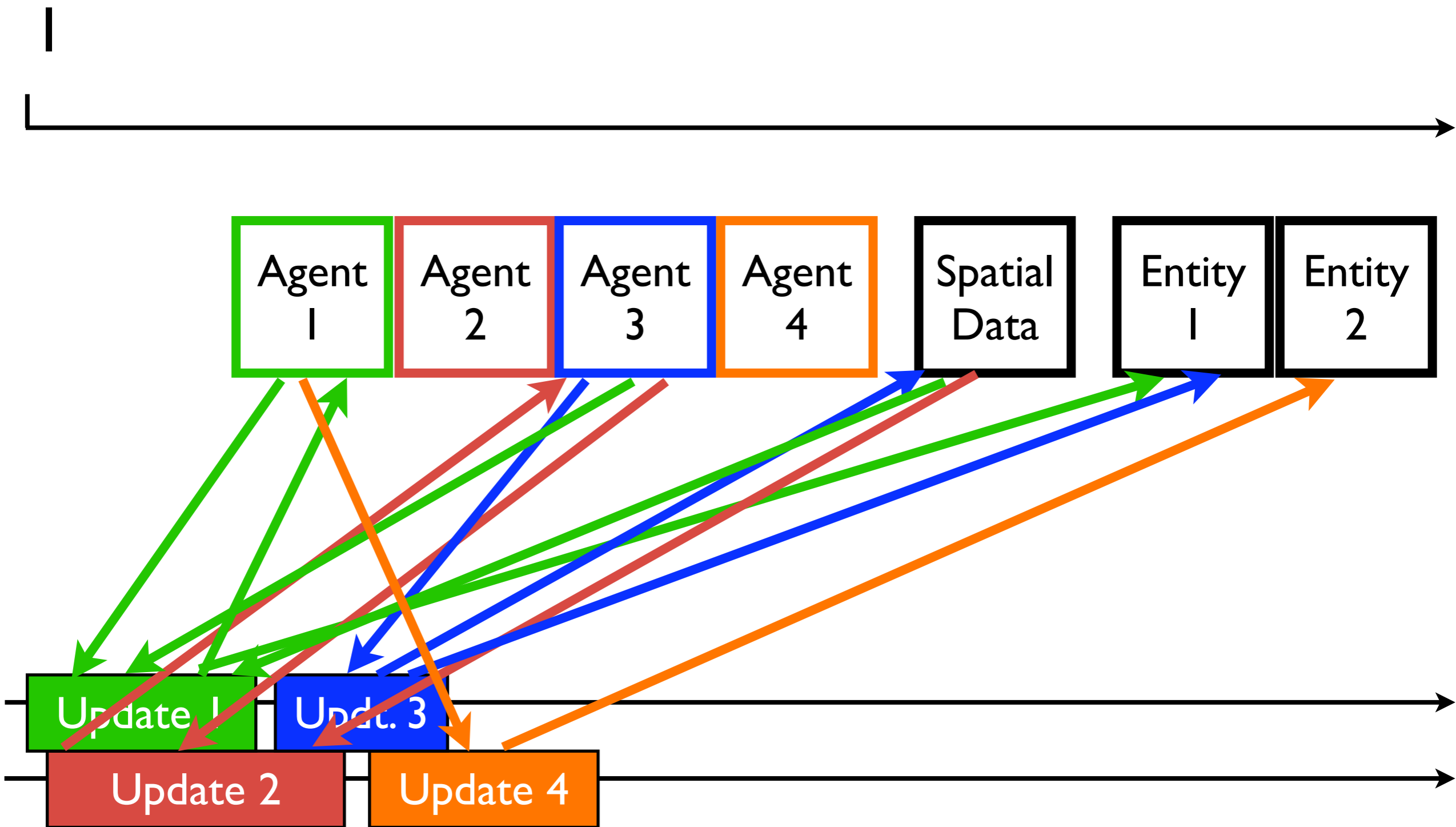
Shared data structures like spatial data and accessing / using other game world entities  
-> unorganized - bad for **bandwidth** + **locality** bad and not **deterministic**  
-> syncing needed -> convoying -> **contention** -> serialization -> blocking bad for task pools  
... if sequential solution was order dependent, then we are non-deterministic and need refactoring

# Frames



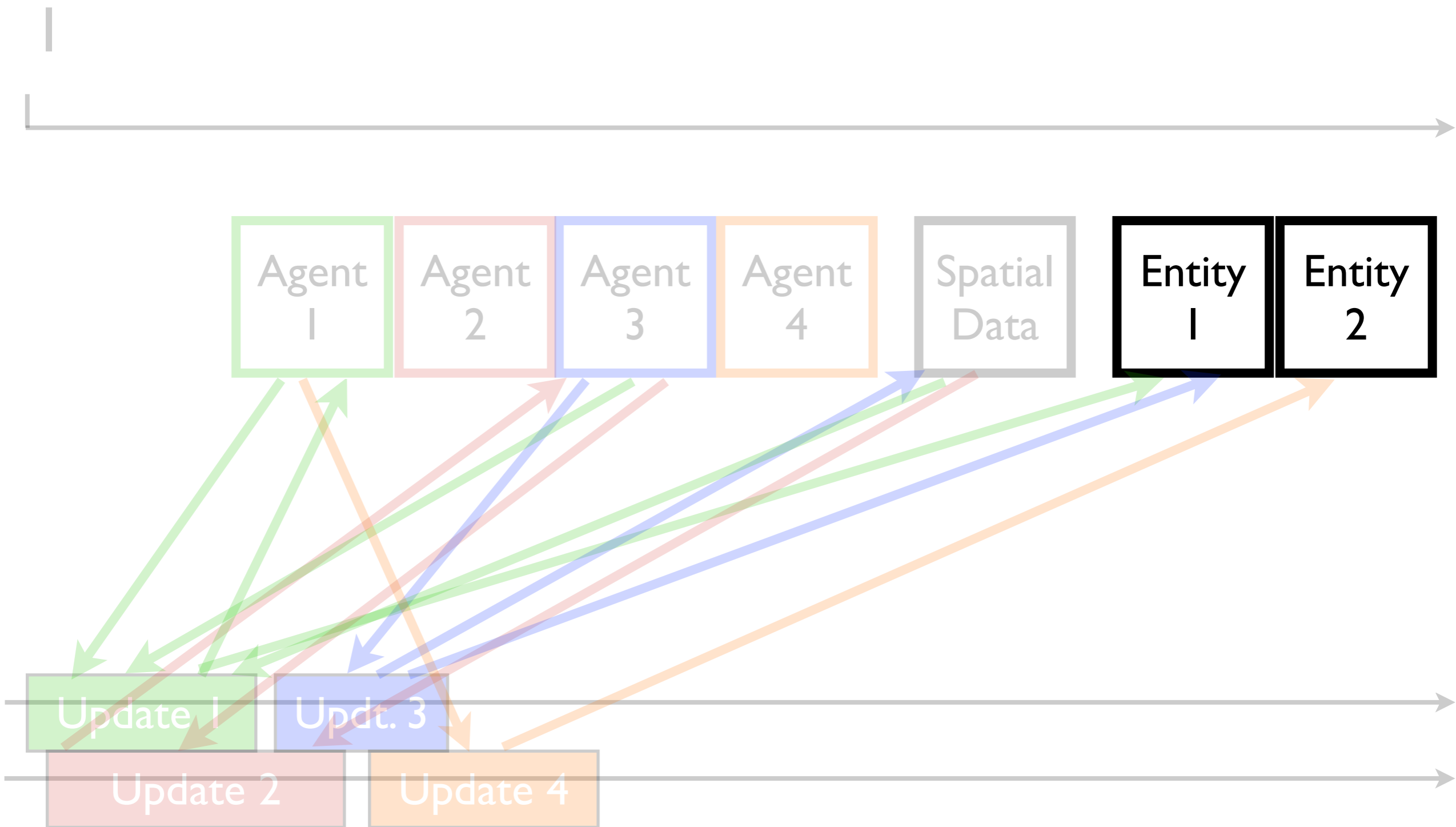
Shared data structures like spatial data and accessing / using other game world entities  
-> unorganized - bad for **bandwidth** + **locality** bad and not **deterministic**  
-> syncing needed -> convoying -> **contention** -> serialization -> blocking bad for task pools  
... if sequential solution was order dependent, then we are non-deterministic and need refactoring

# Frames



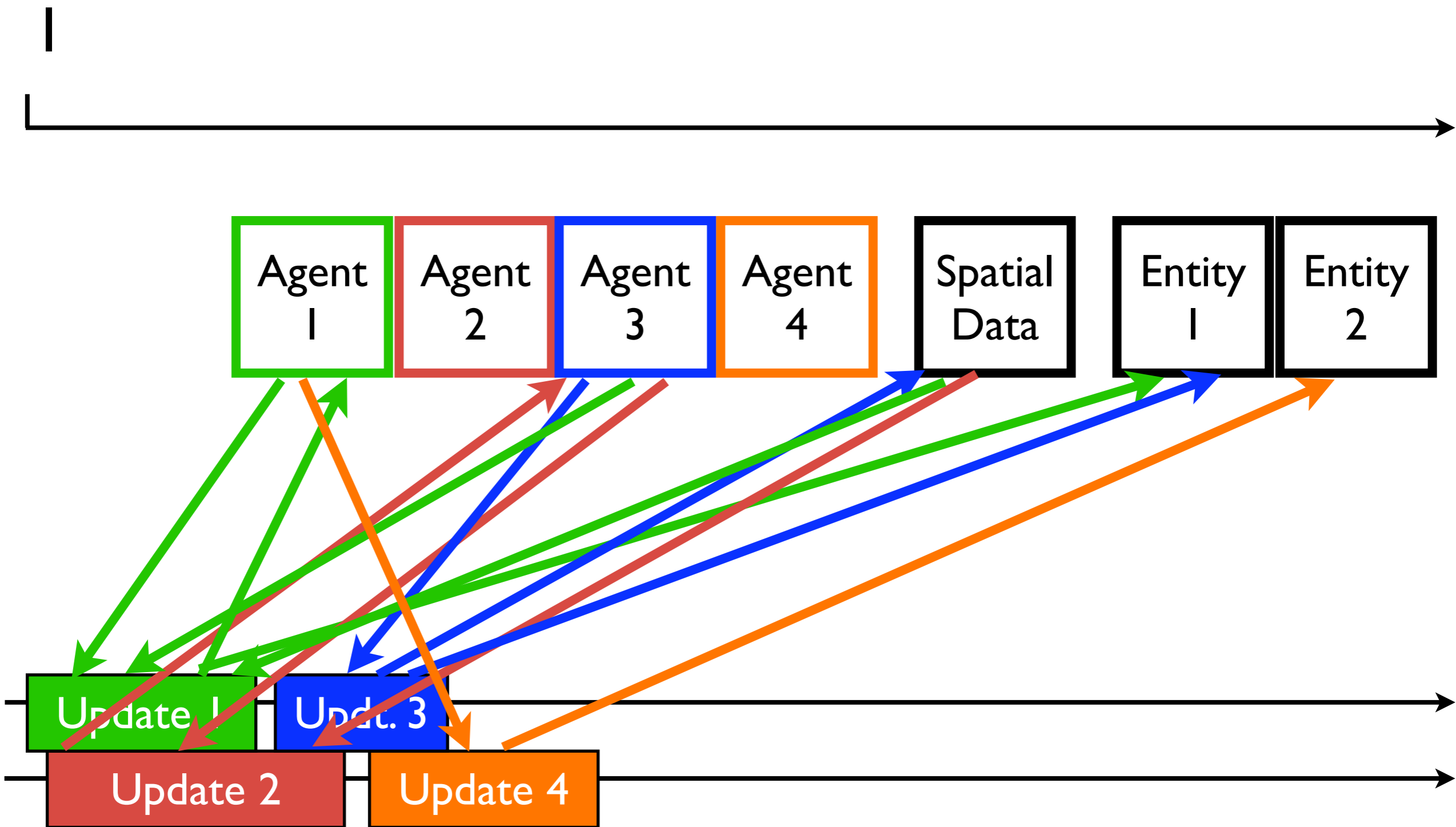
Entity management is ownership management -> network and distributed programming  
Suggestion: **defer anything but read accesses**, enqueue ownership/action requests and...  
... work on them after all agents are updated, agents react to results in next update.  
Detect conflict islands and resolve them deterministically. Compare with actor model ;-)

# Frames



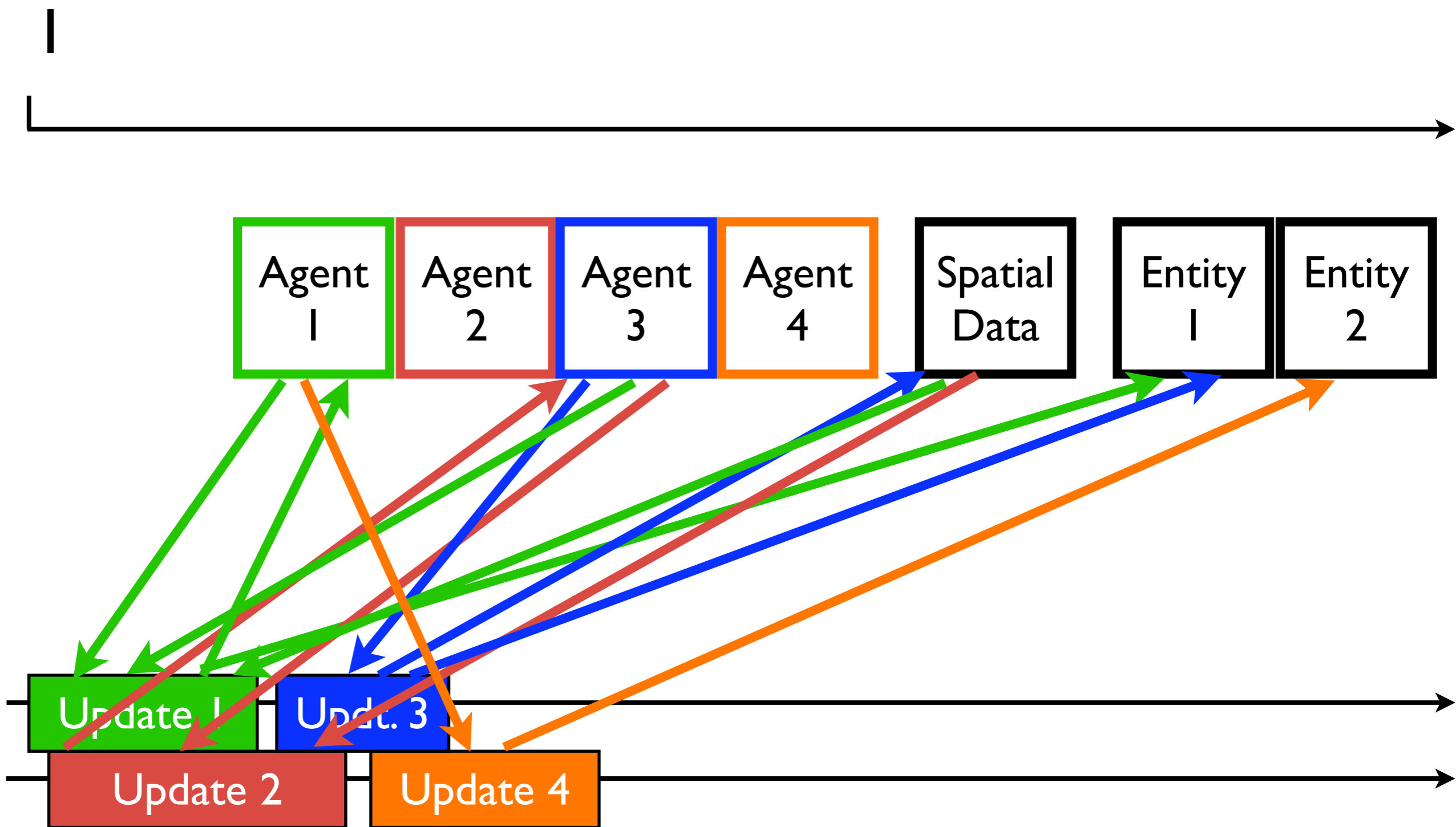
Entity management is ownership management -> network and distributed programming  
Suggestion: **defer anything but read accesses**, enqueue ownership/action requests and...  
... work on them after all agents are updated, agents react to results in next update.  
Detect conflict islands and resolve them deterministically. Compare with actor model ;-)

# Frames



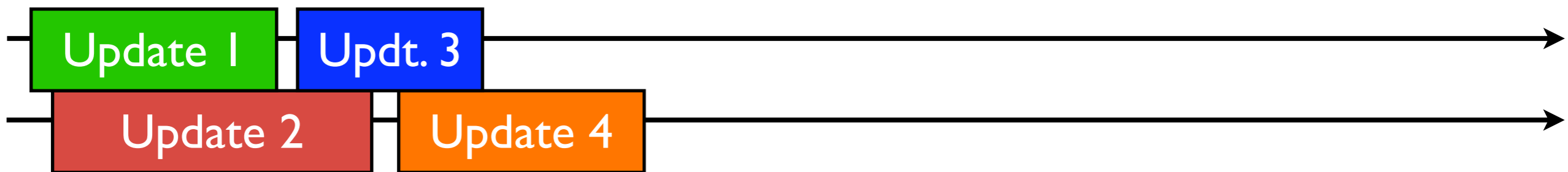
Entity management is ownership management -> network and distributed programming  
Suggestion: **defer anything but read accesses**, enqueue ownership/action requests and...  
... work on them after all agents are updated, agents react to results in next update.  
Detect conflict islands and resolve them deterministically. Compare with actor model ;-)

# Frames



So, how to solve syncing, serialization, and determinism problem?  
First: identify dependencies, reading, writing, public data, private data

# Frames



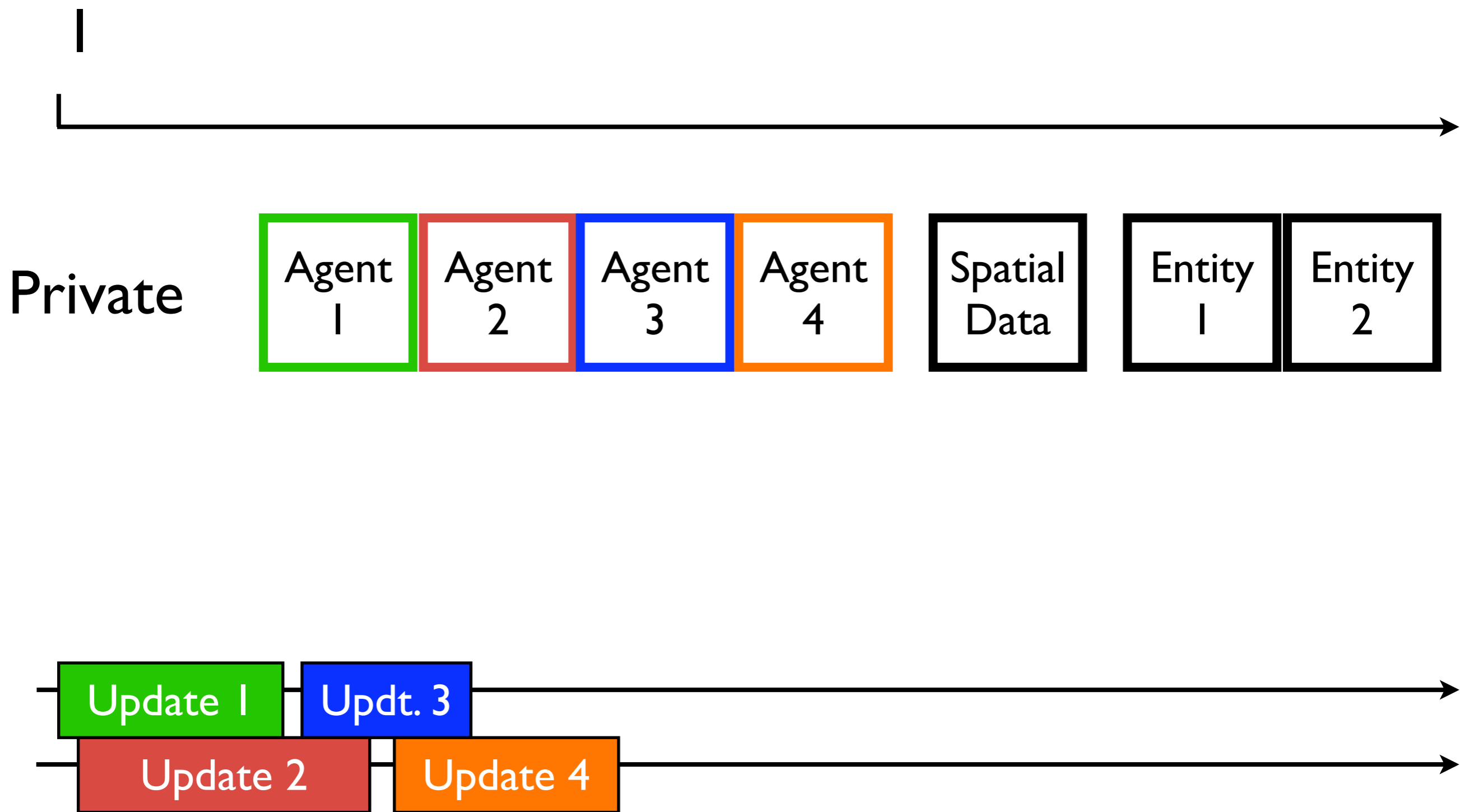
Split instance data into public data (read only) and private data (writable by instance owner)

Duplication of state

Public data represents state from last time step / frame

Annotation: add random number generator instances per agent private data

# Frames



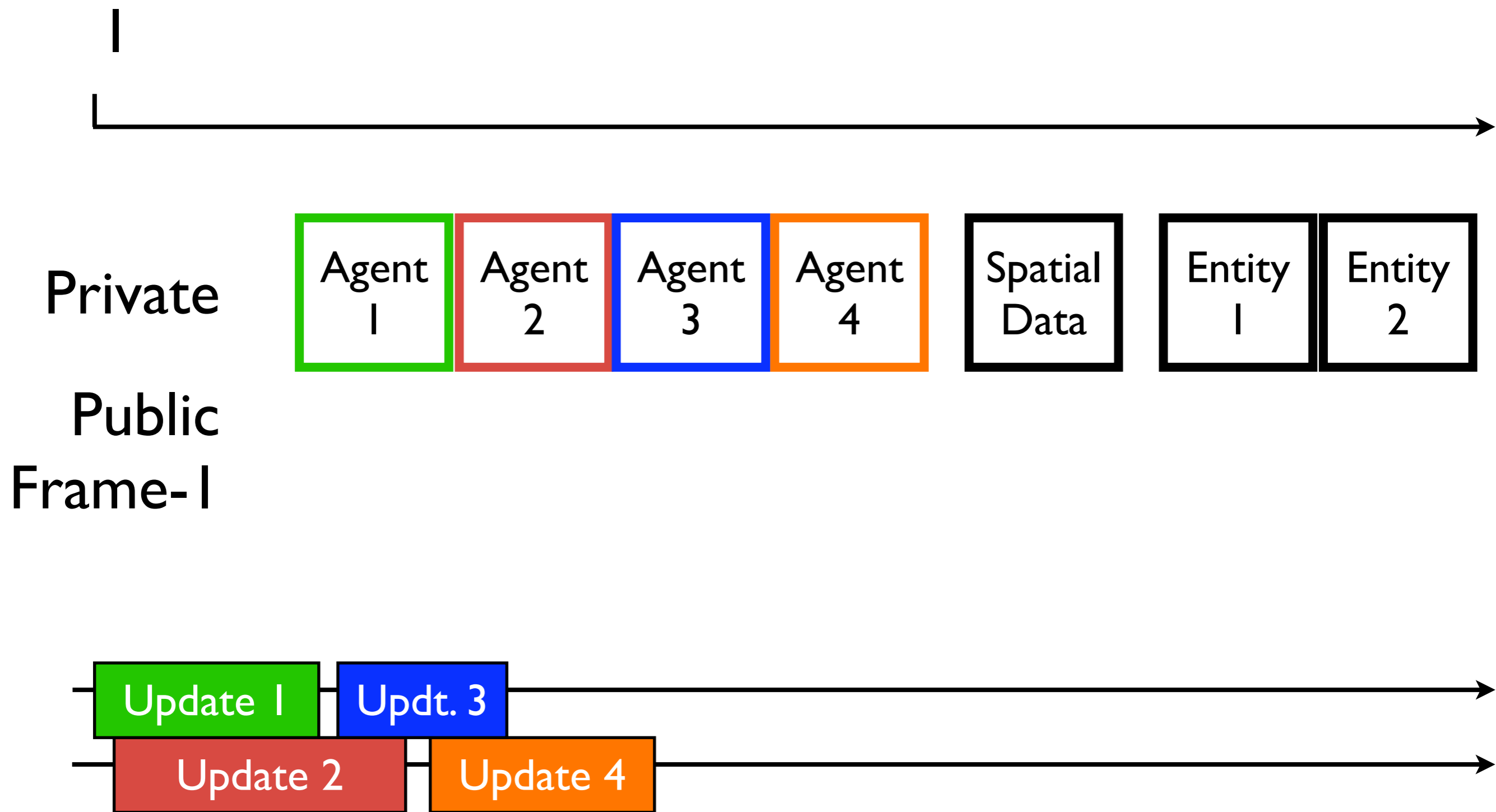
Split instance data into public data (read only) and private data (writable by instance owner)

Duplication of state

Public data represents state from last time step / frame

Annotation: add random number generator instances per agent private data

# Frames



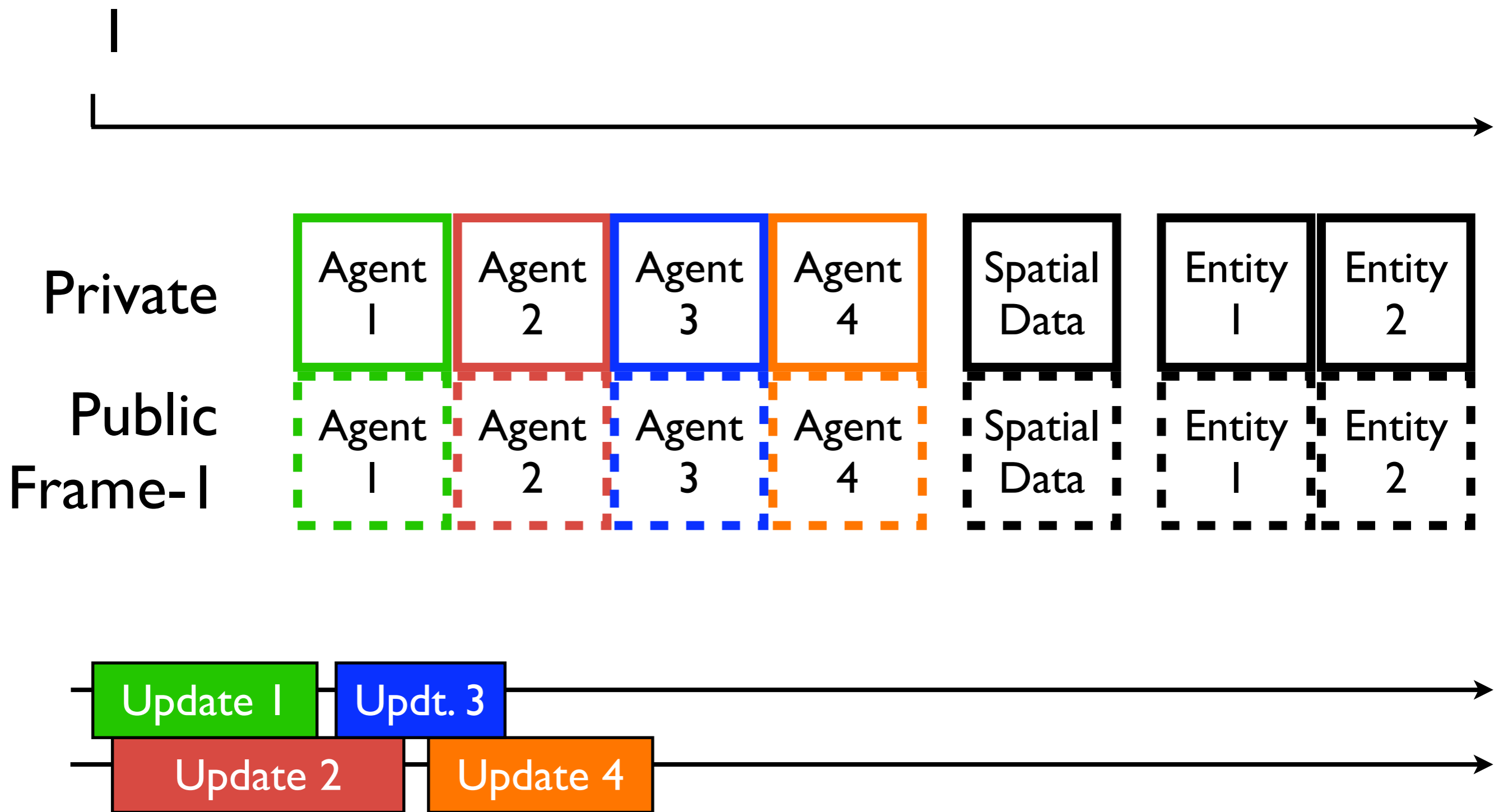
Split instance data into public data (read only) and private data (writable by instance owner)

Duplication of state

Public data represents state from last time step / frame

Annotation: add random number generator instances per agent private data

# Frames



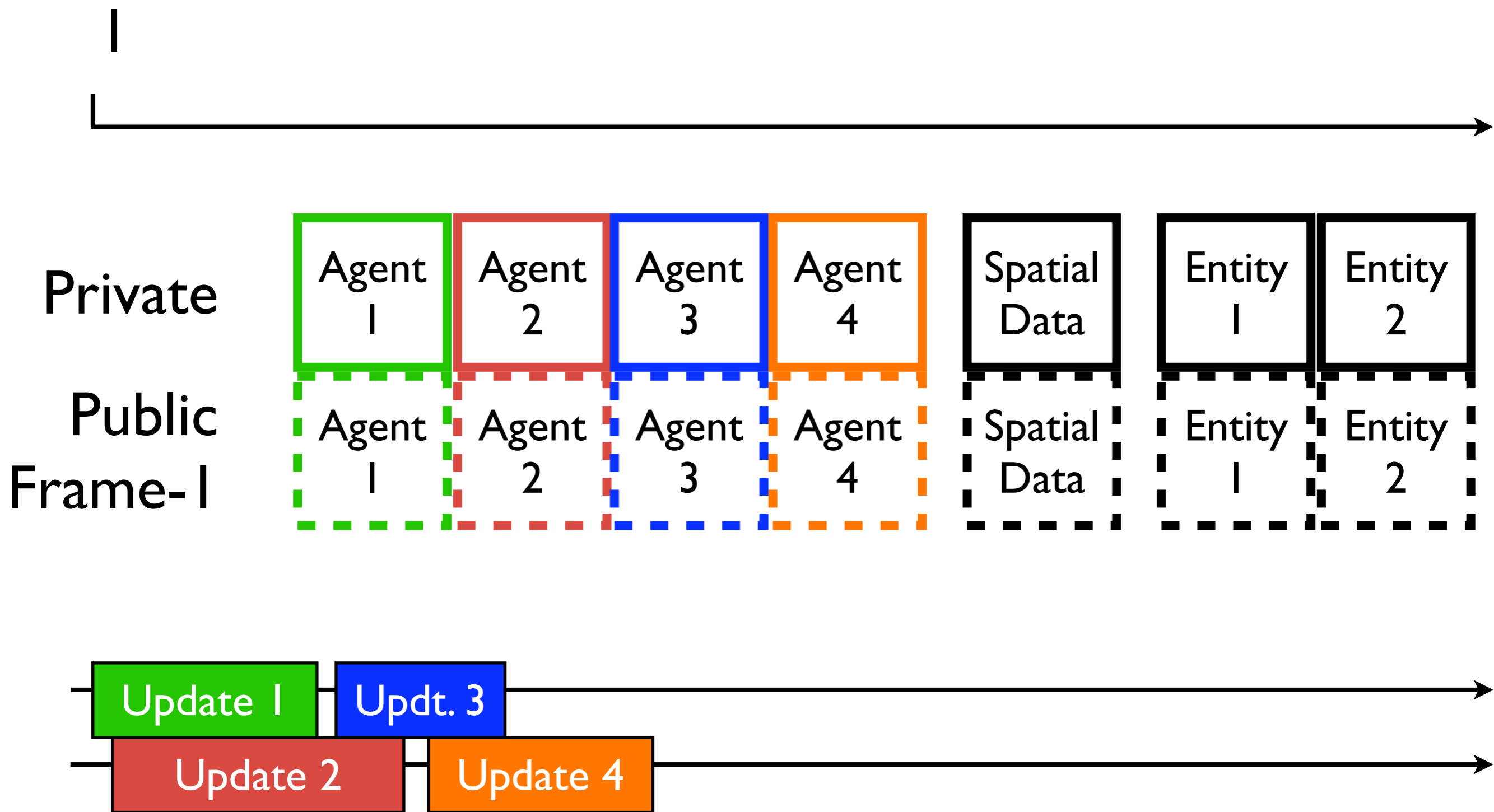
Split instance data into public data (read only) and private data (writable by instance owner)

Duplication of state

Public data represents state from last time step / frame

Annotation: add random number generator instances per agent private data

# Frames

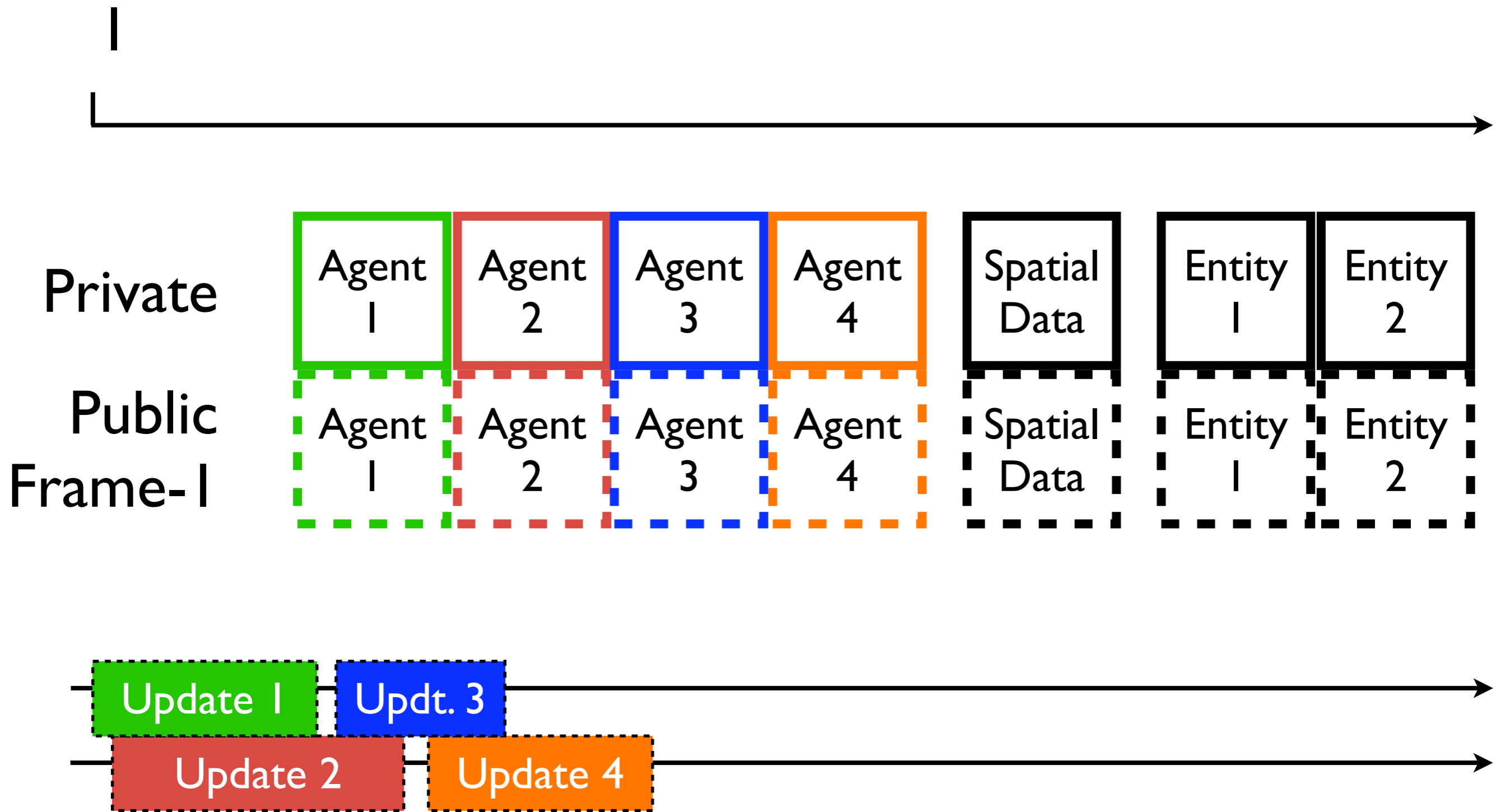


Reorganize update phase into two stages:

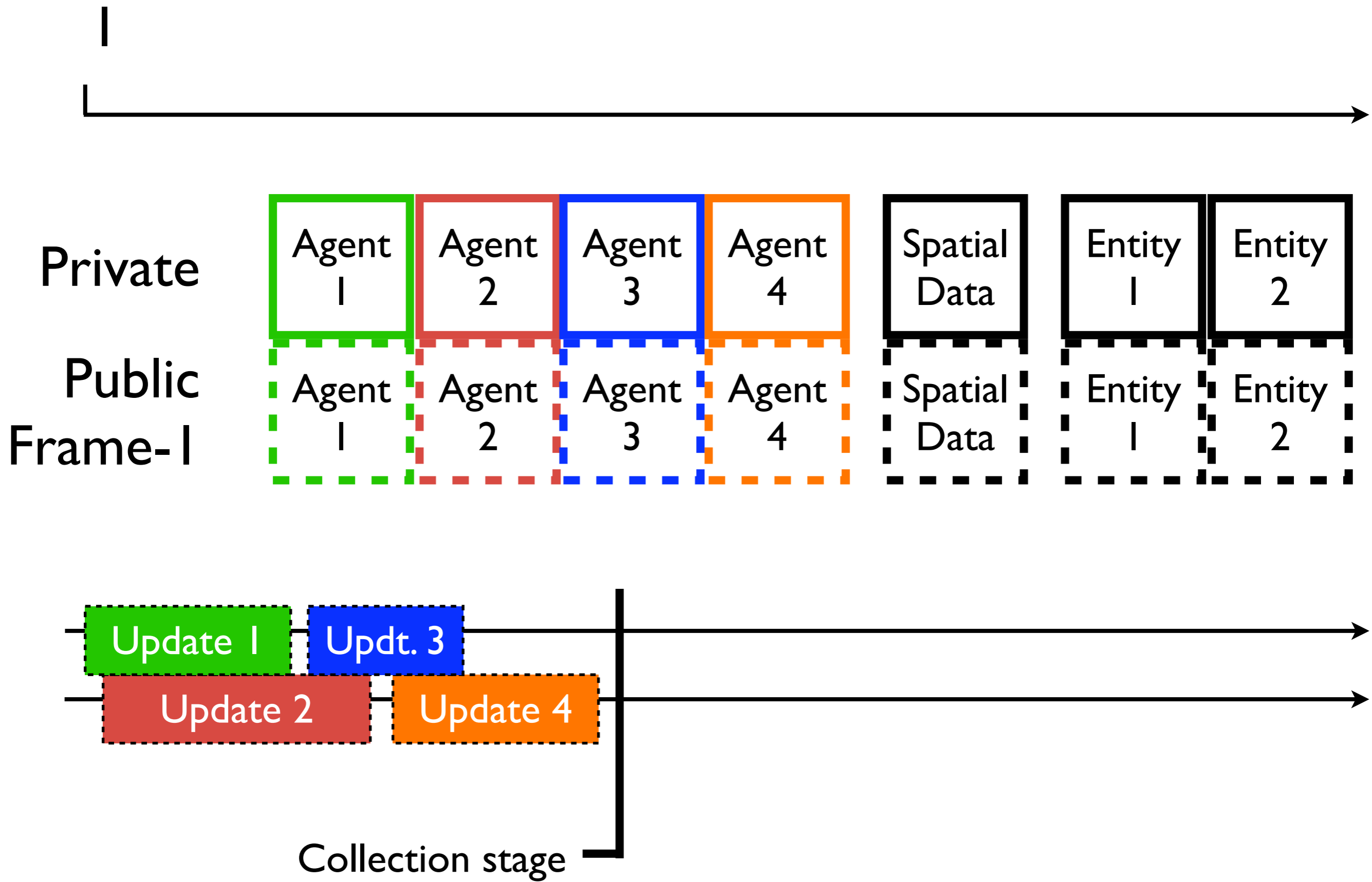
1. read public data and write own private data

-> reading public data must be without side effects!!!

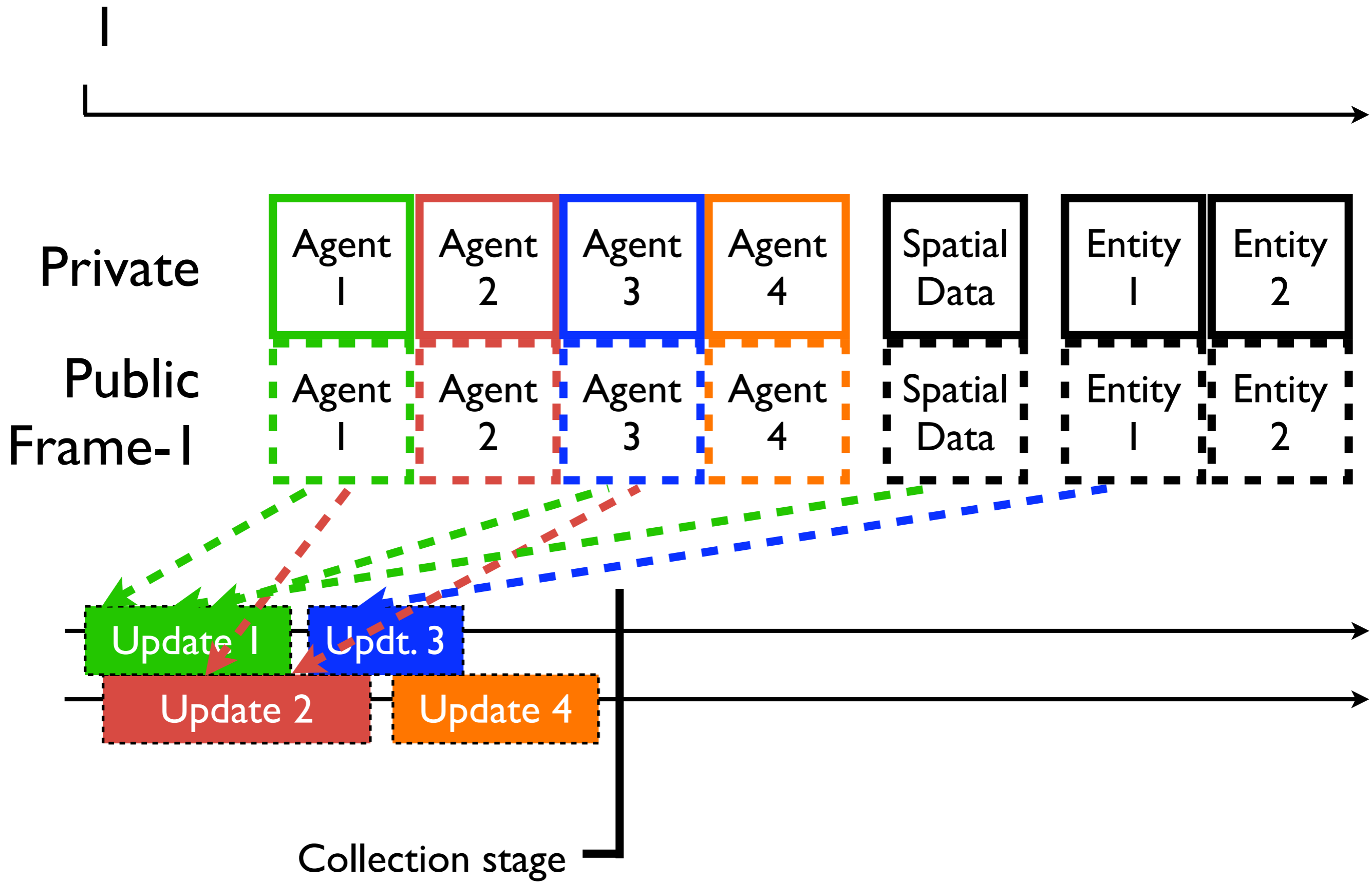
# Frames



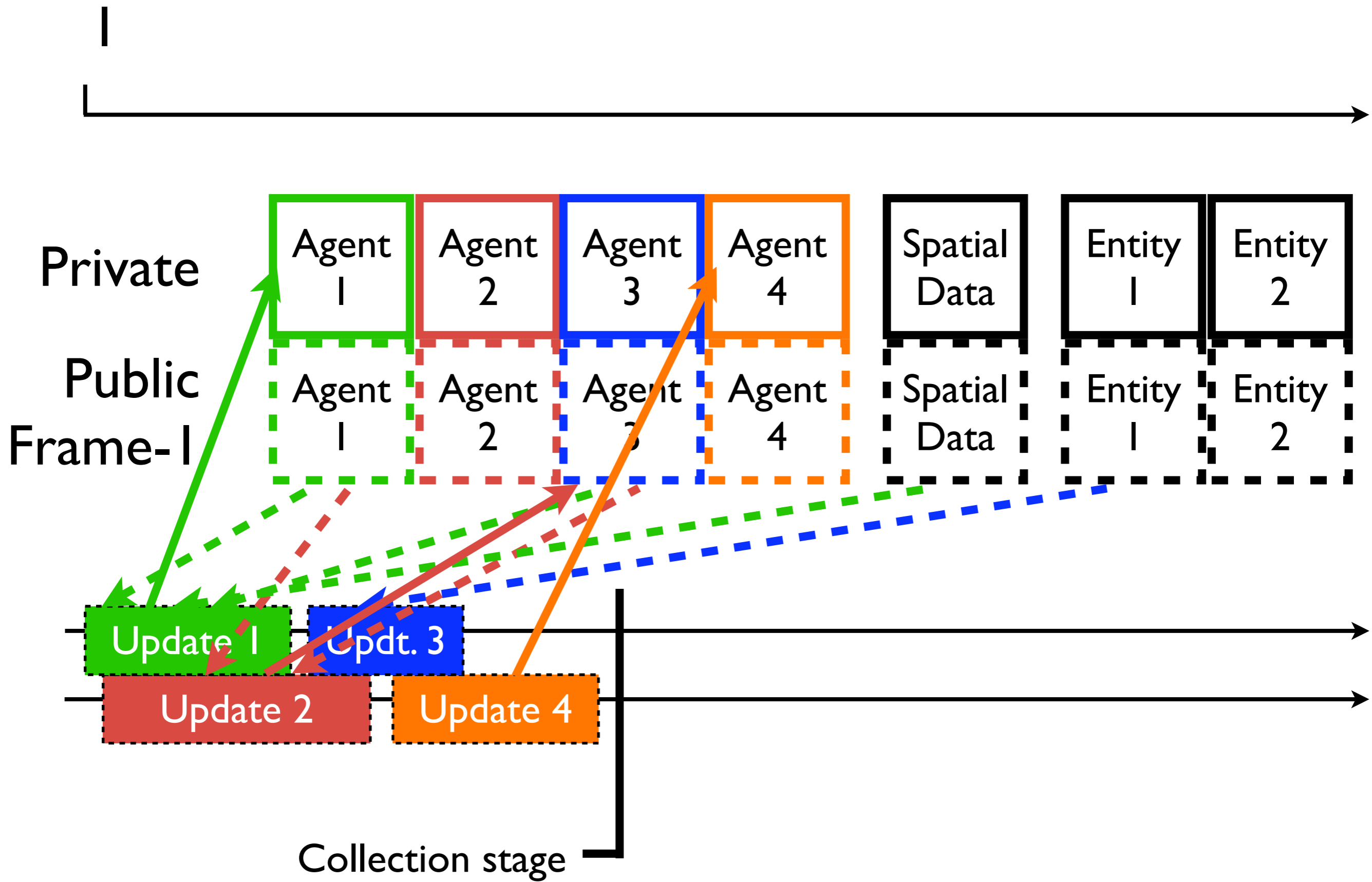
# Frames



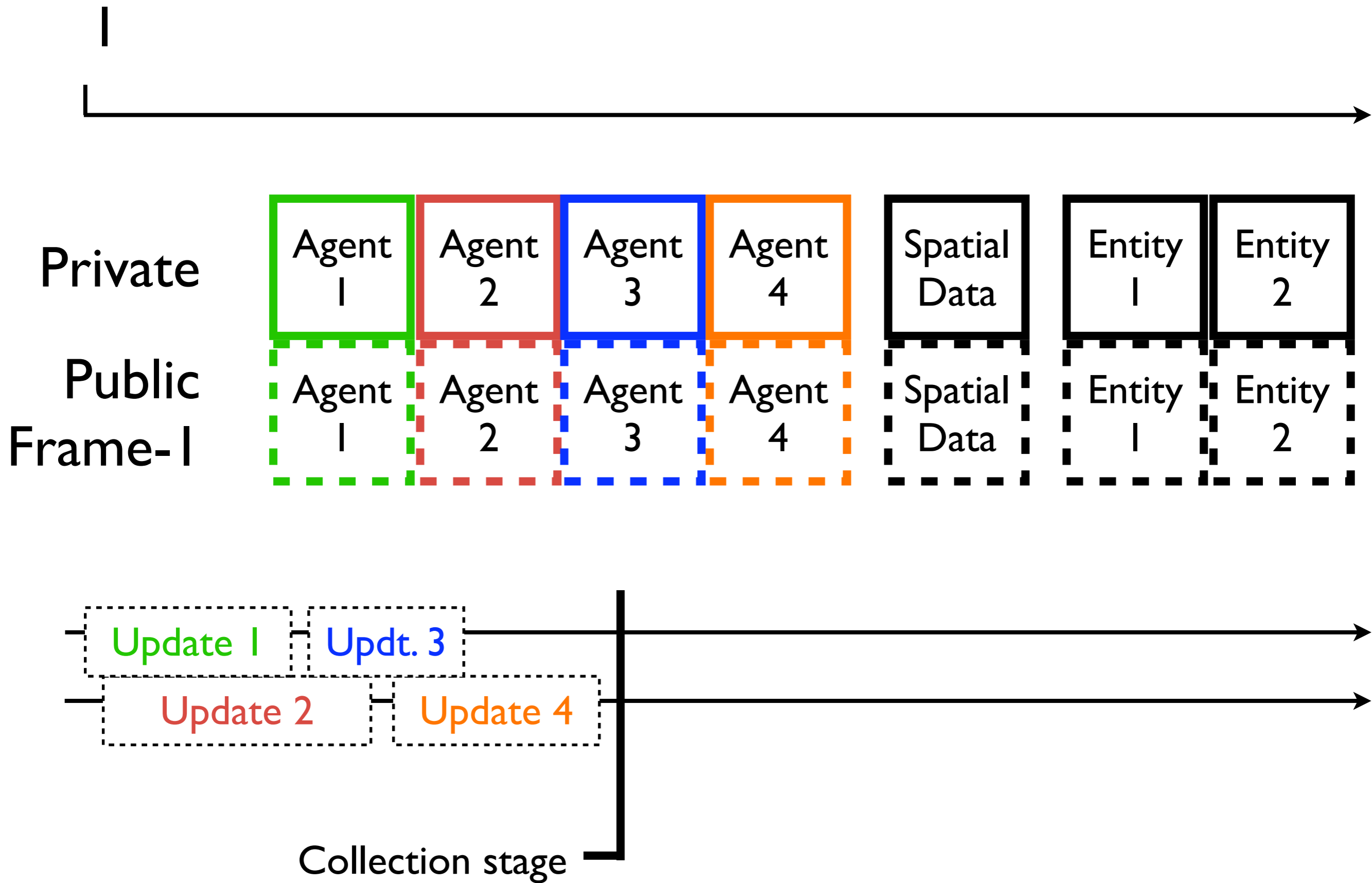
# Frames



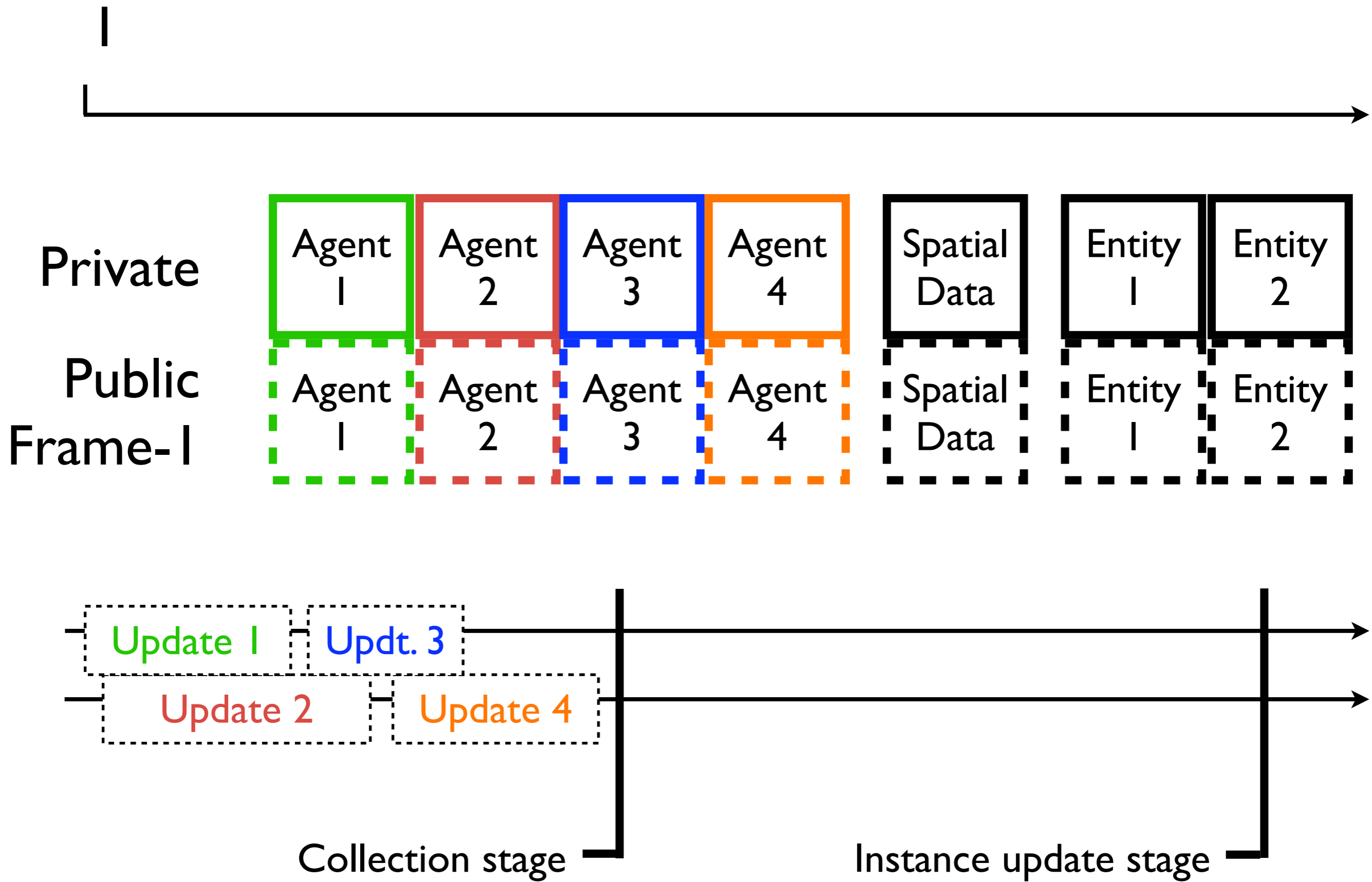
# Frames



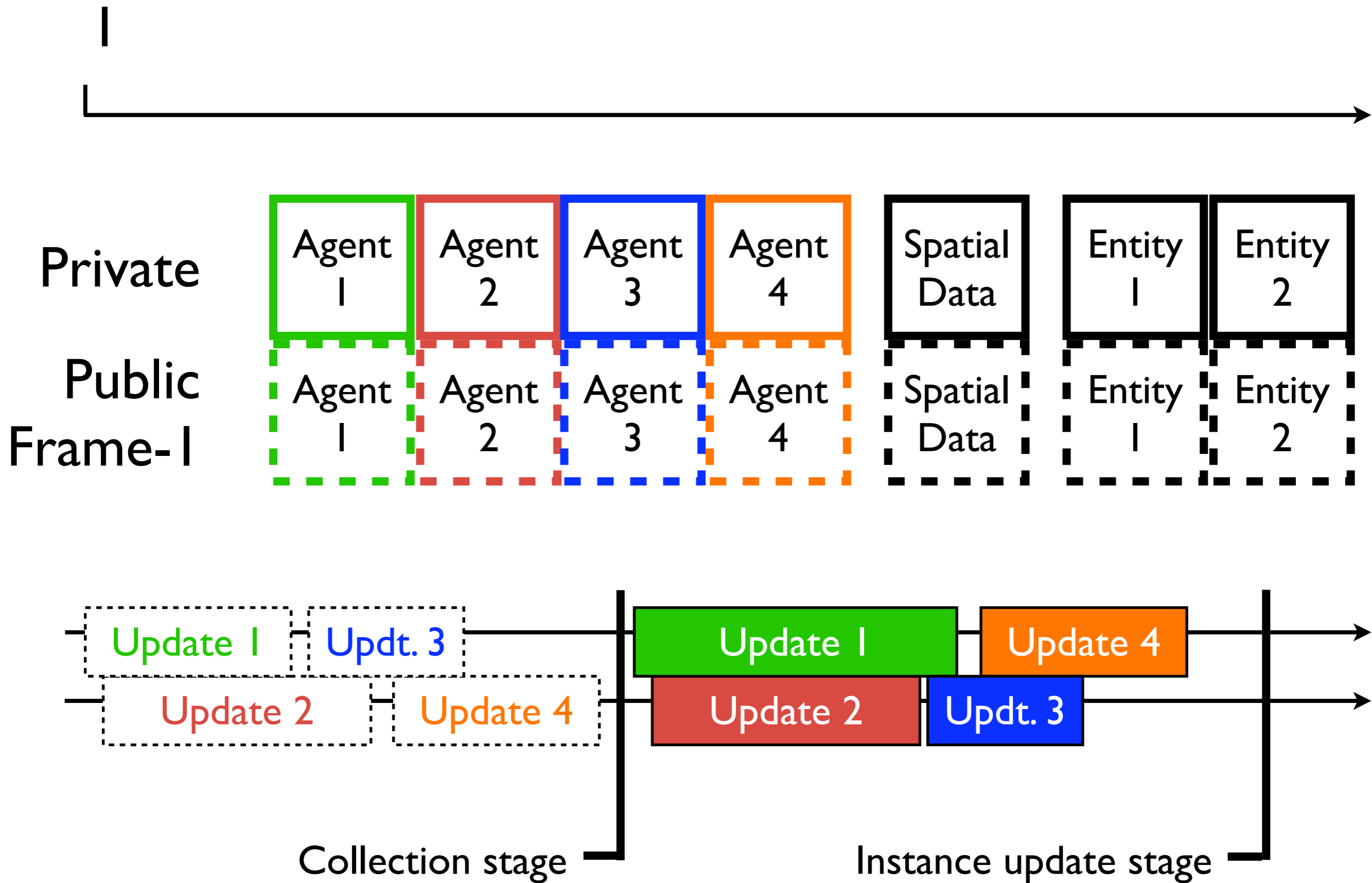
# Frames



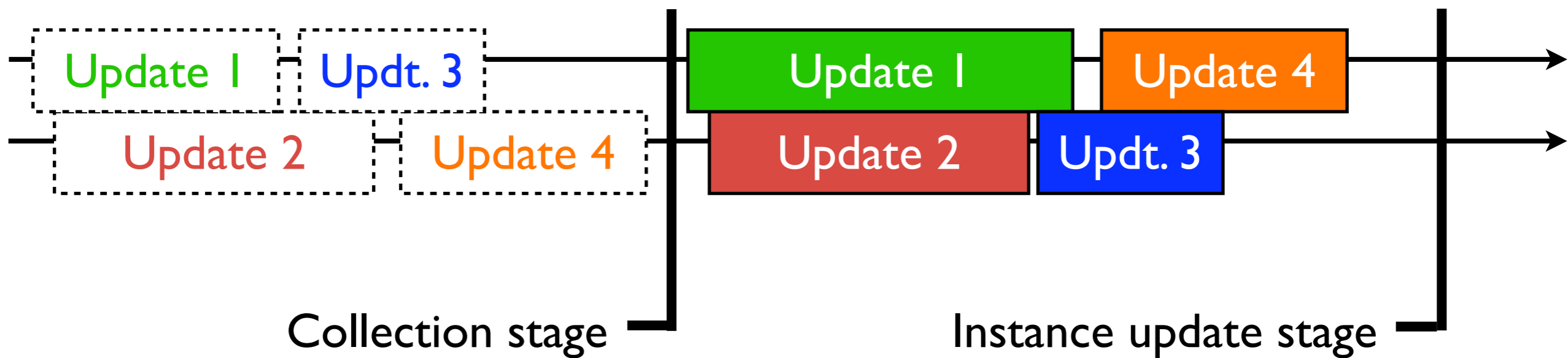
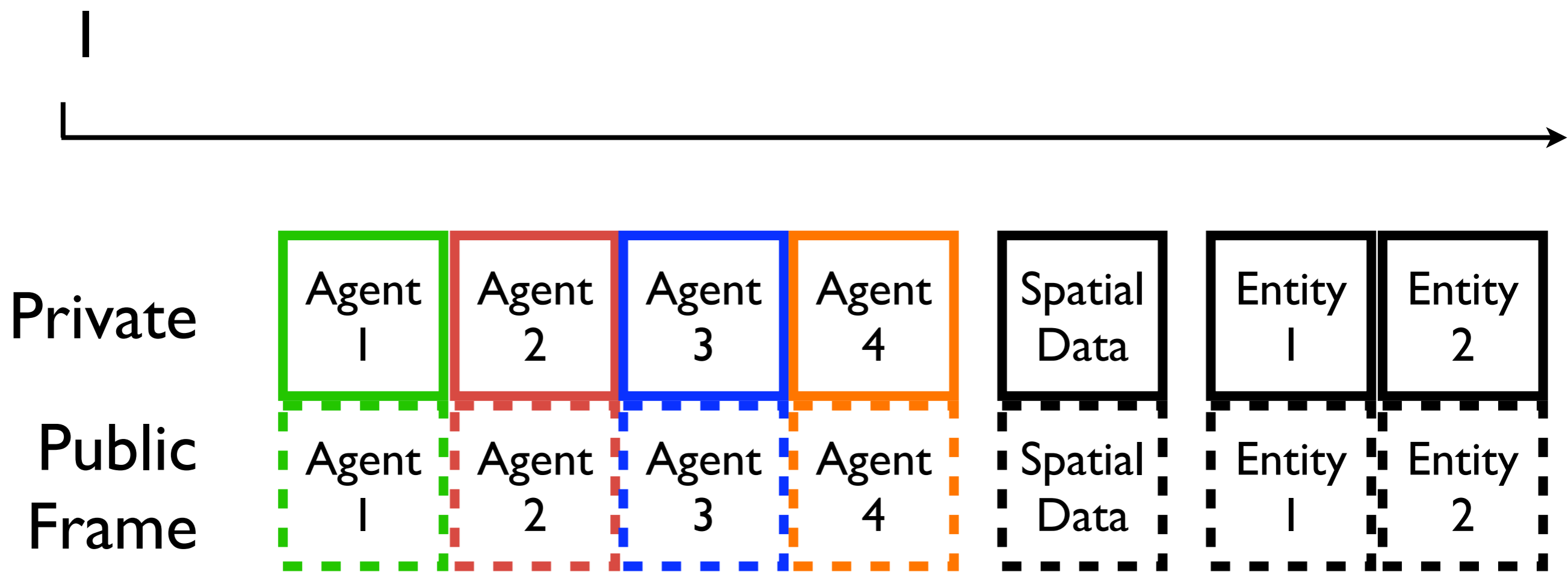
# Frames



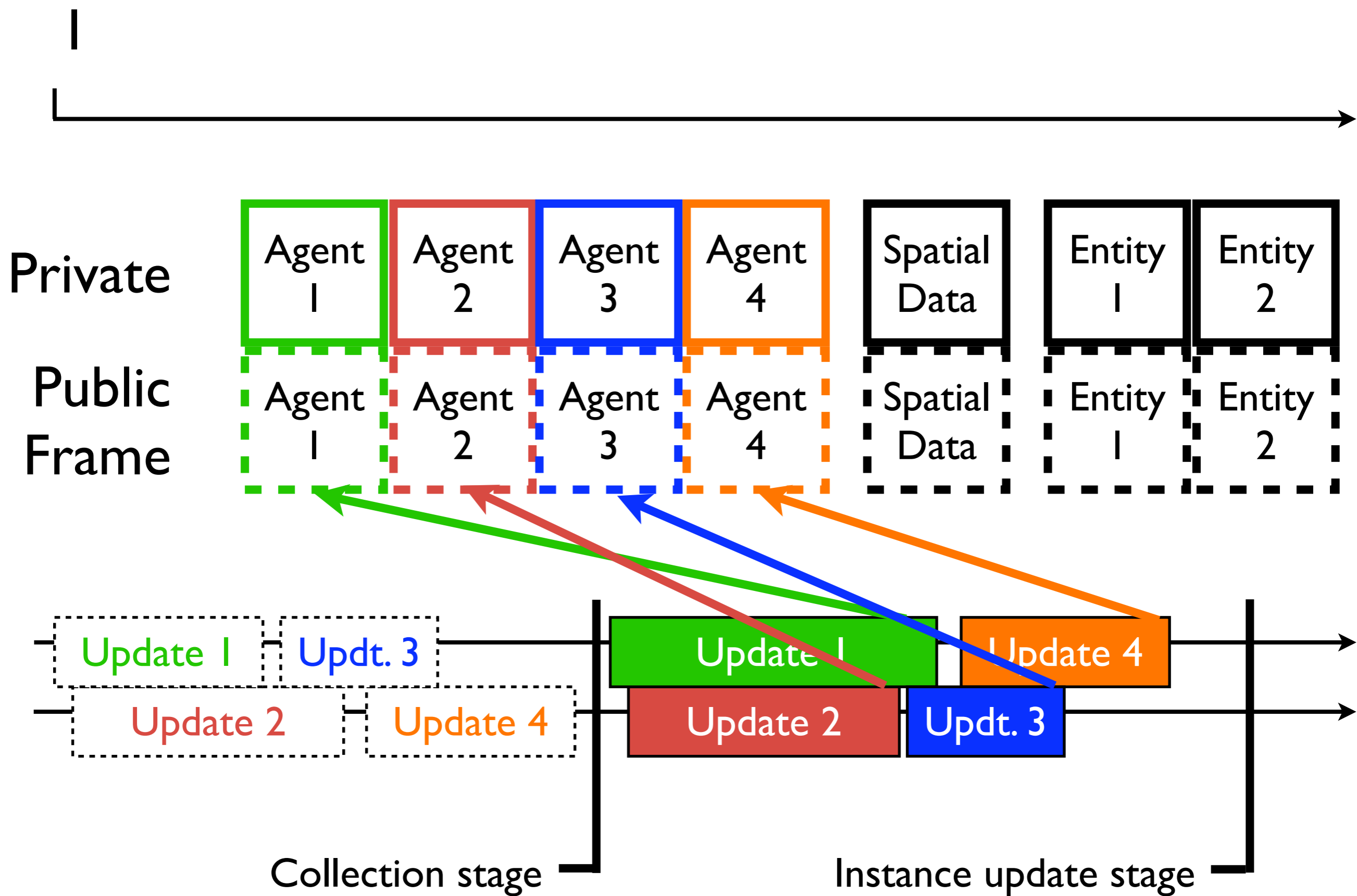
# Frames



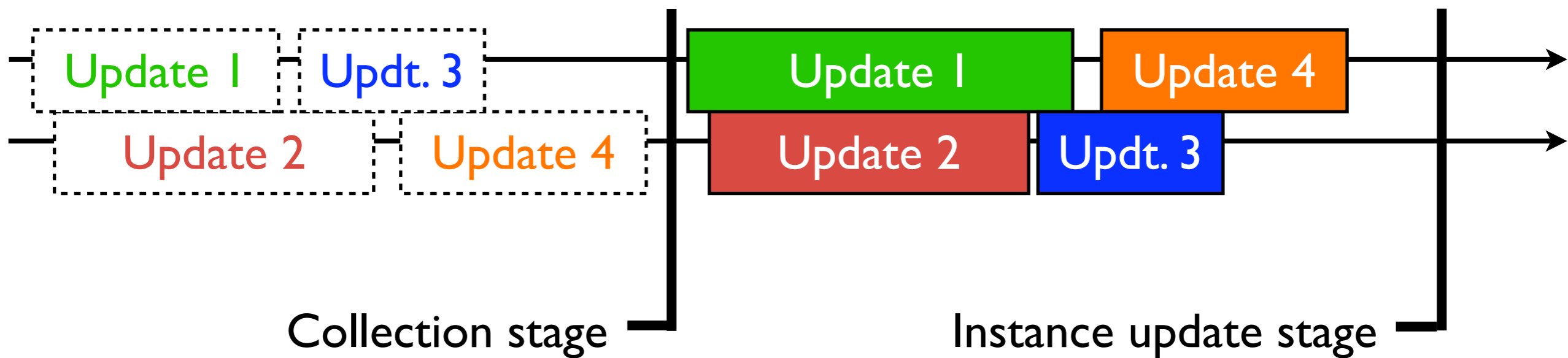
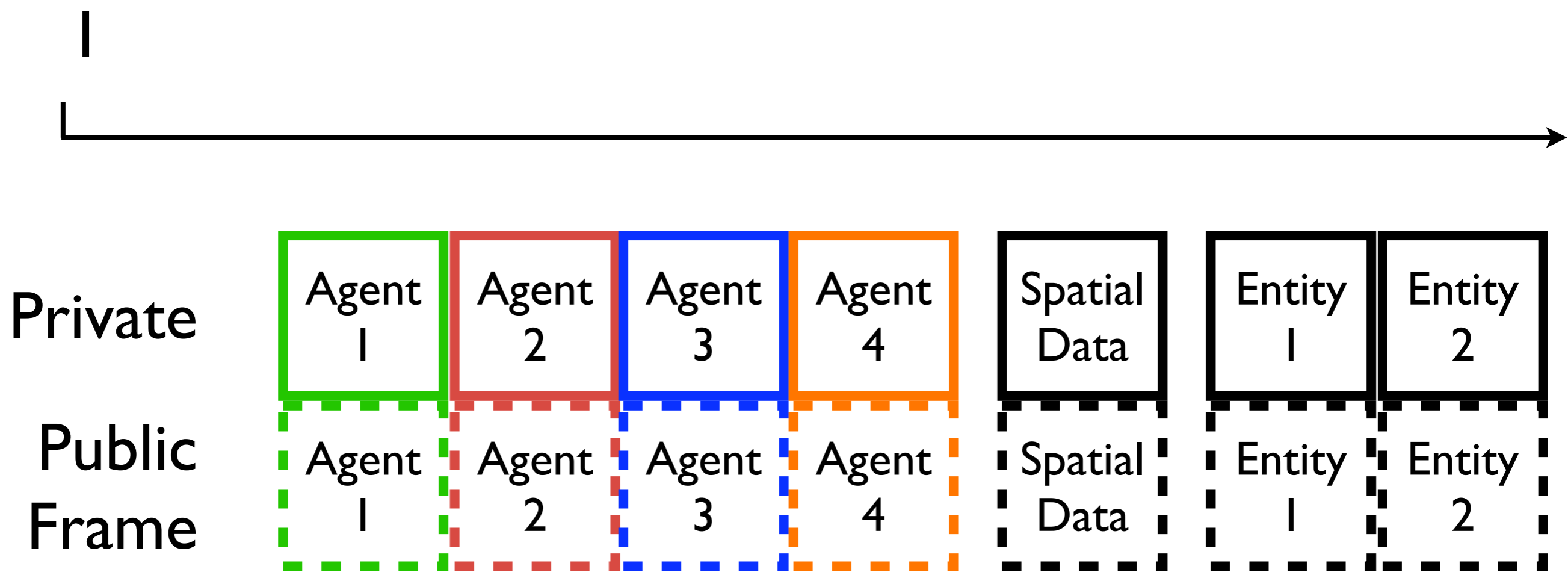
# Frames



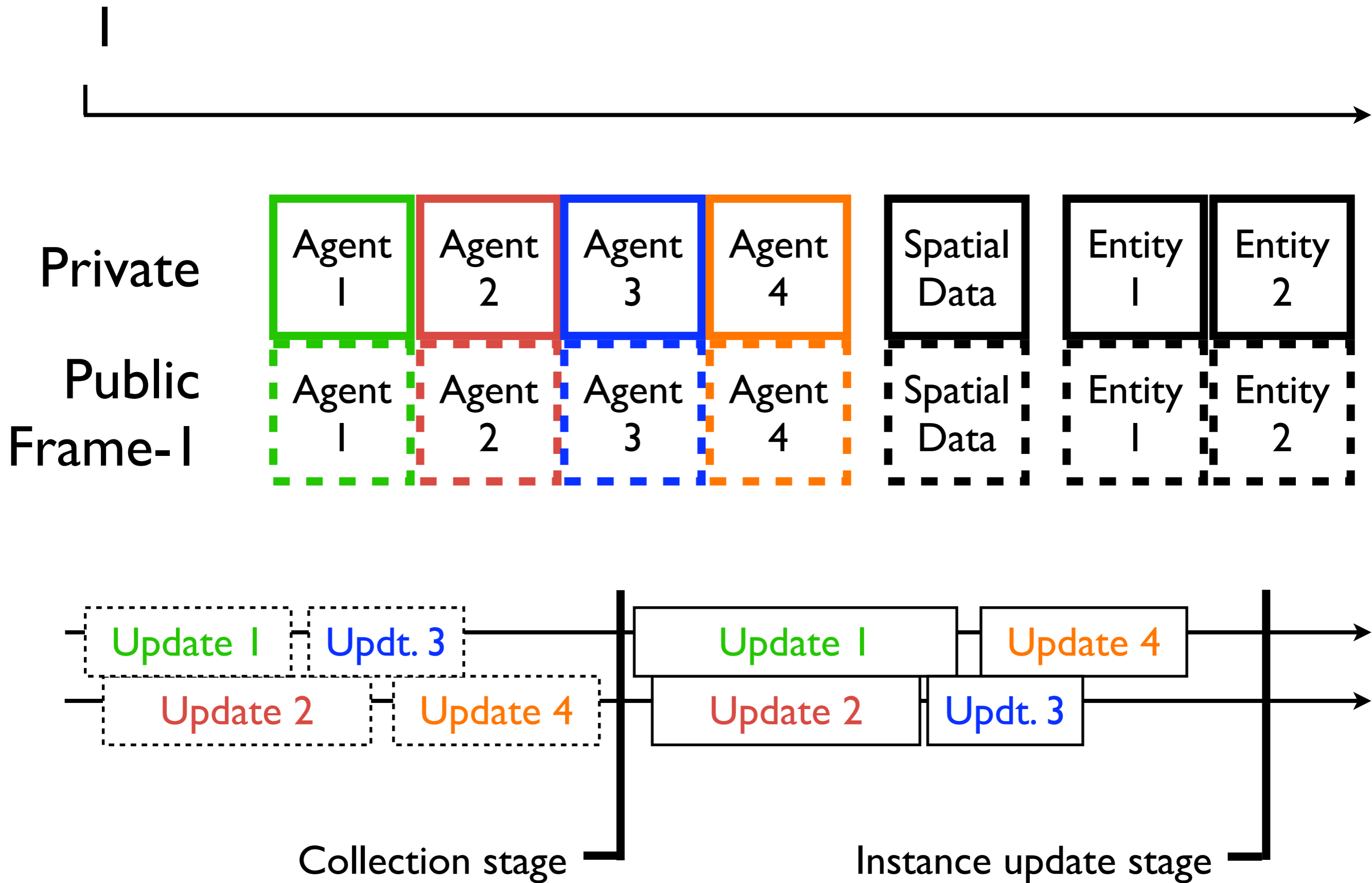
# Frames



# Frames



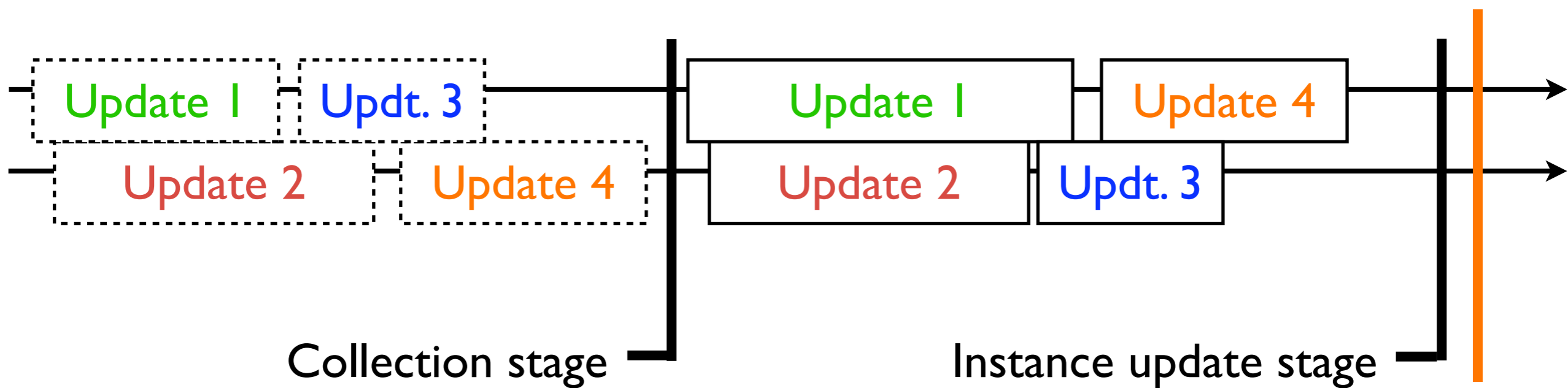
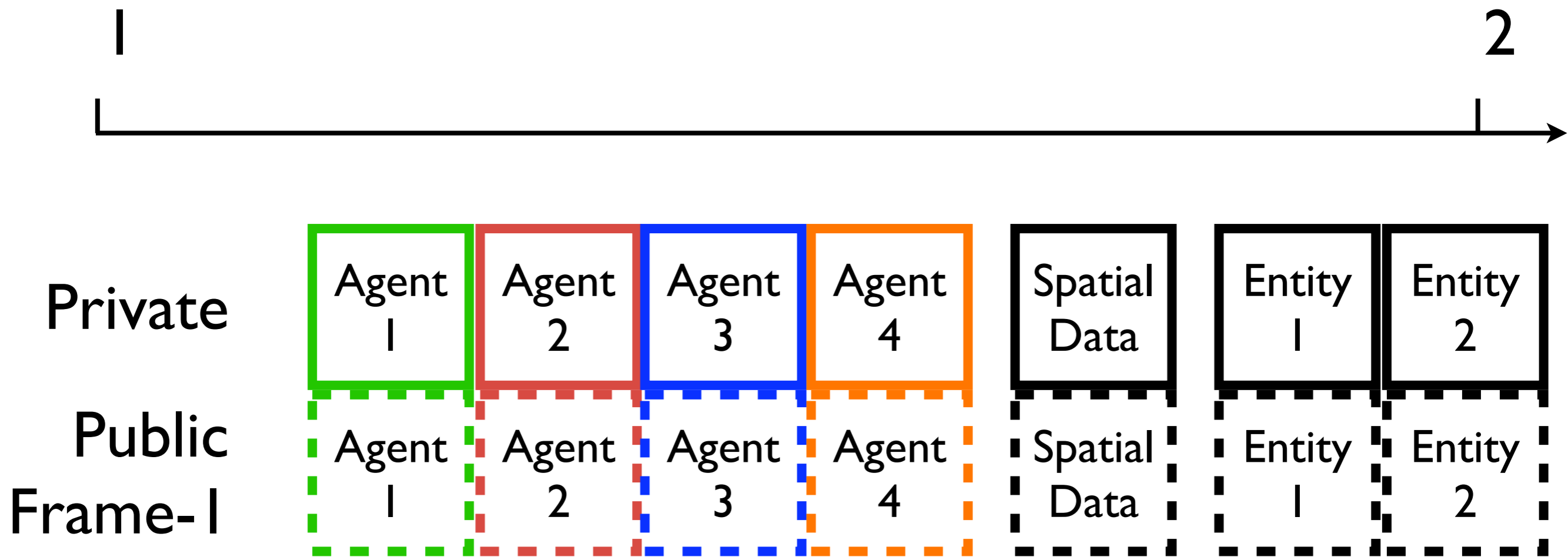
# Frames



Private and public data a bit like double buffering, could also be handled with double buffering...

On to the next time step

# Frames



Private and public data a bit like double buffering, could also be handled with double buffering...

On to the next time step

# Parallel Agents

Pros	Cons

Still danger of side effects and non-thread-safe func calls in agent...  
Really: implementation complex but no other way -> changes help even sequential exec!!!  
First step to get better locality and more implicit syncing

# Parallel Agents

Pros

Cons

Guides parallelization

Scaling possible

Implicit syncing

# Parallel Agents

## Pros

Guides parallelization

Scaling possible

Implicit syncing

## Cons

Implementation effort

Get determinism right

Get syncing right

Avoid side effects

Bad locality

# Parallel Agents

Implementation effort  
but **enables** even  
**more parallelism**

Guides parallelism implementation effort

Scaling possible

Get determinism right

Implicit syncing

Get syncing right

Avoid side effects

Bad locality

# 3.

# Data Parallel Systems

Unsolved problem from last approach: bad locality and bandwidth usage because of random/unordered access to data structures -> lets improve this

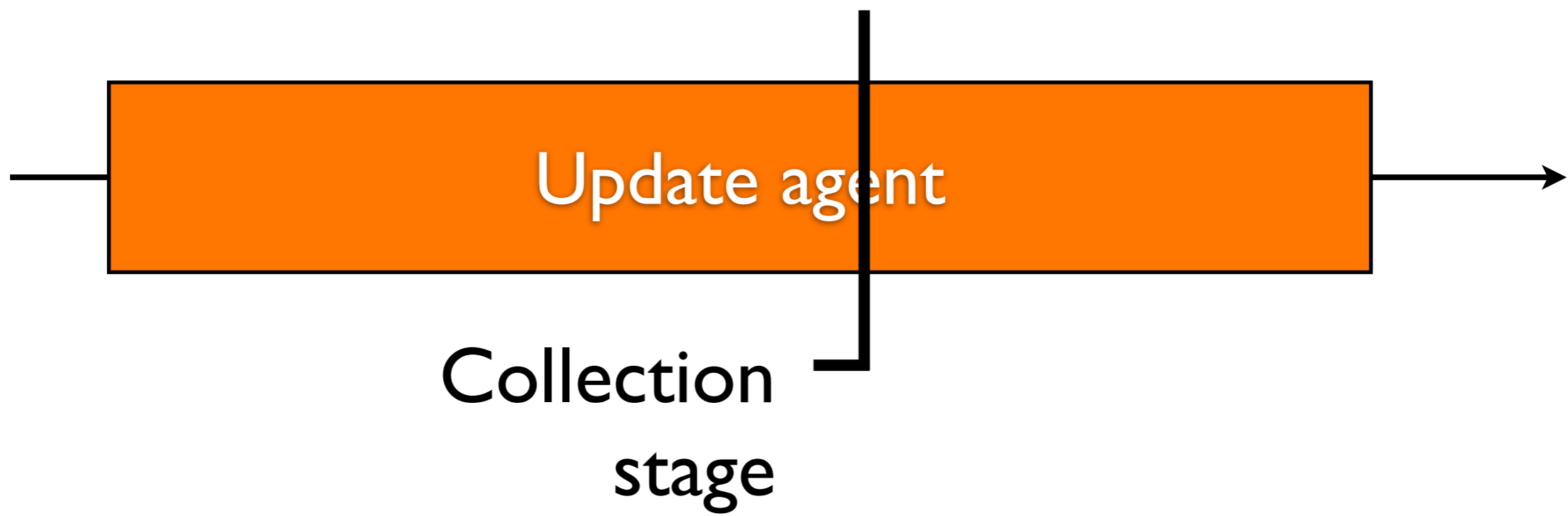
Approach builds upon last approach

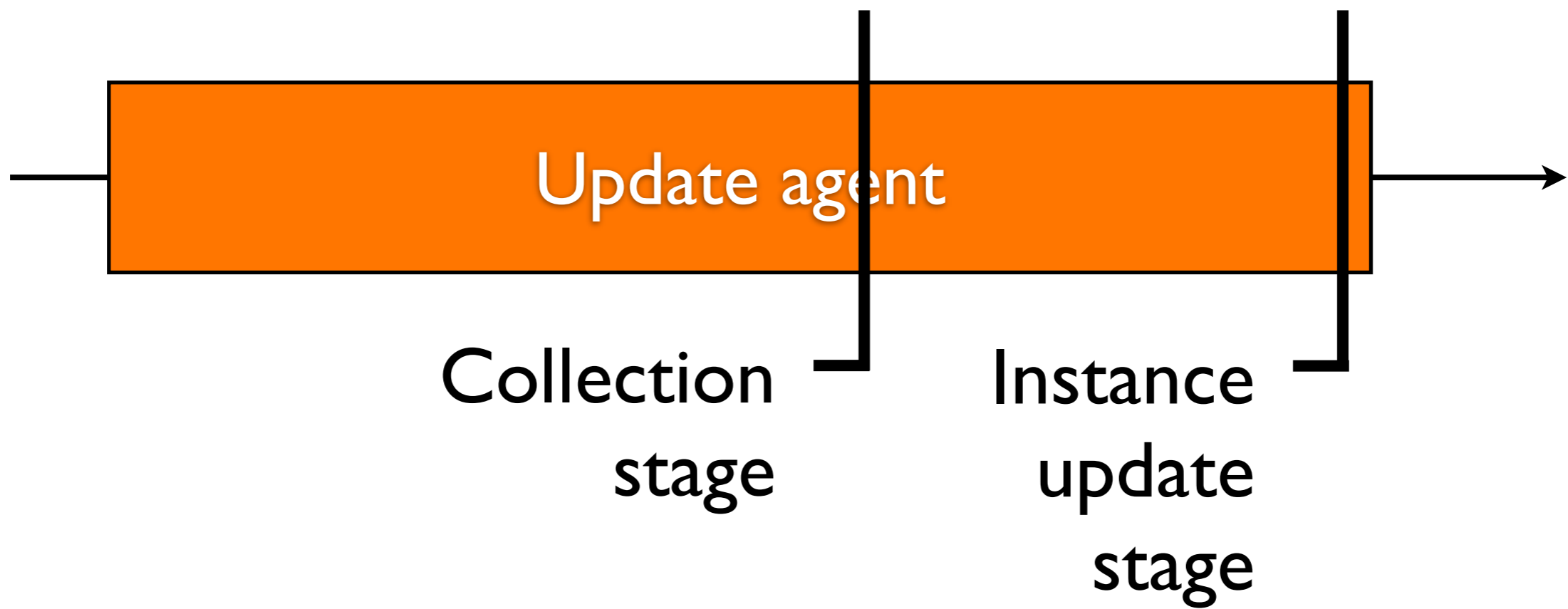
Approach supported by Valve's Source Engine, used by Insomniac and Guerilla Games for Killzone2

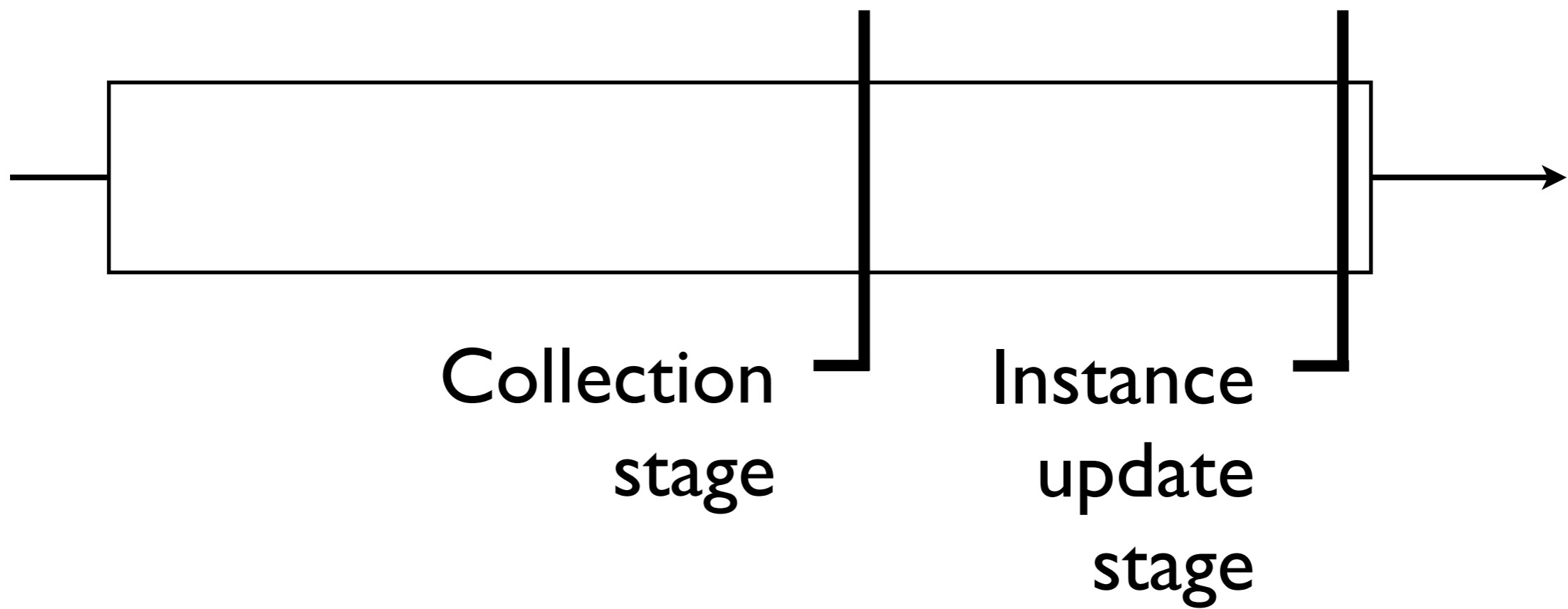
... task pool thread or SPU/GPU instruction exec

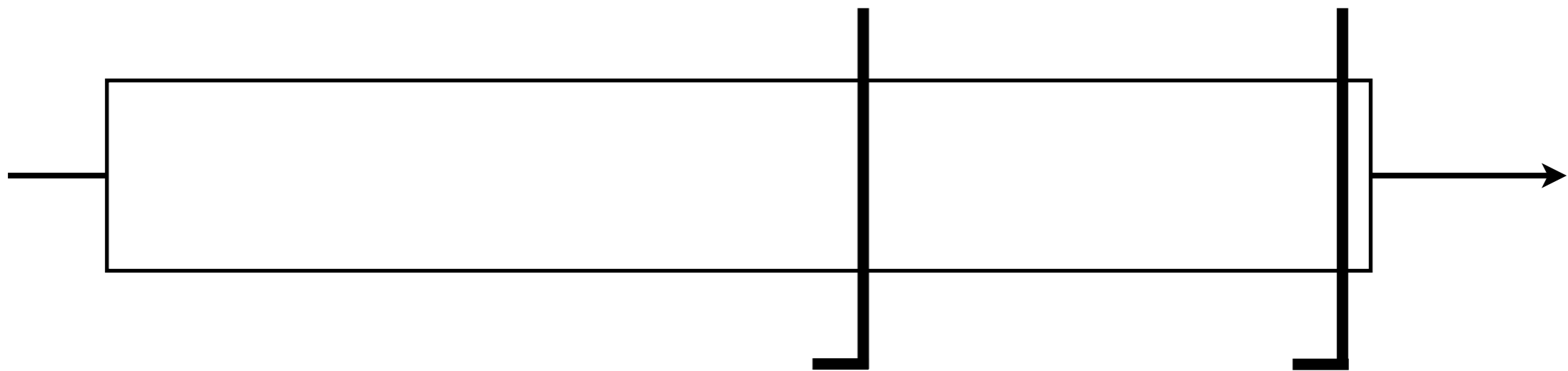


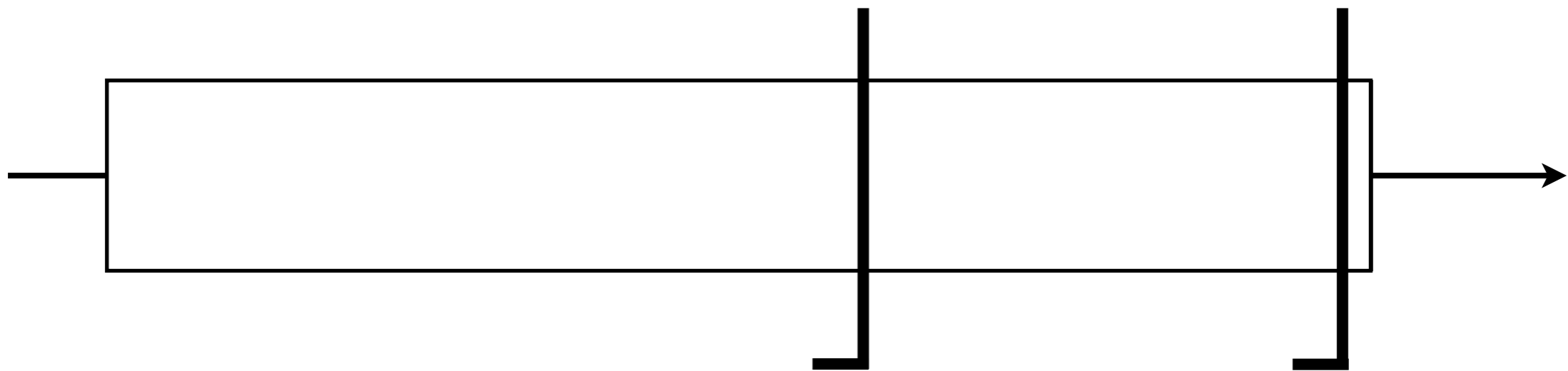












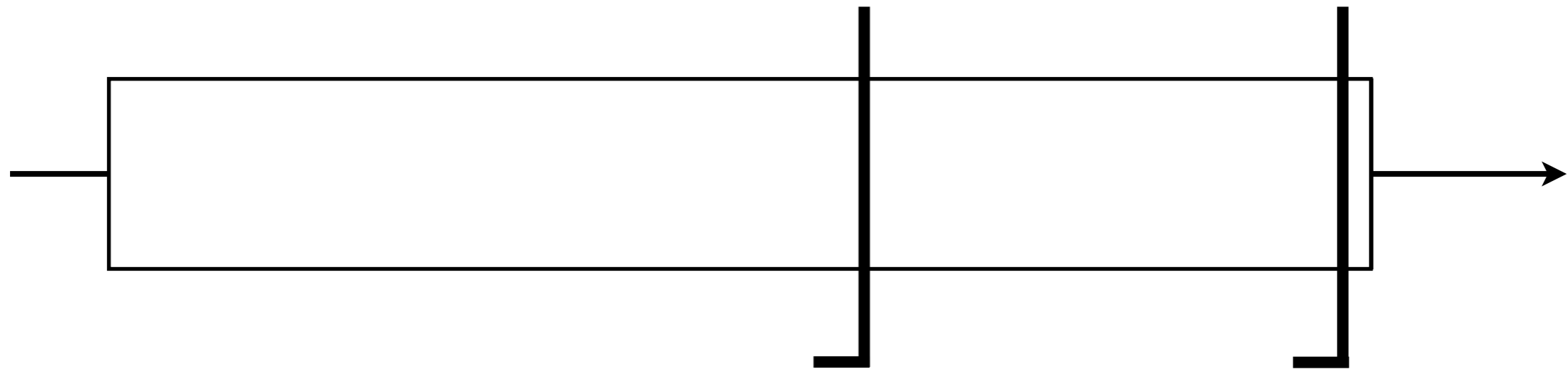
... looks different for every agent (order, timing, branches)  
- agents run in parallel on many cores -> random mem accesses, no coordination  
-> bandwidth?? memory locality??

Agent  
inst.

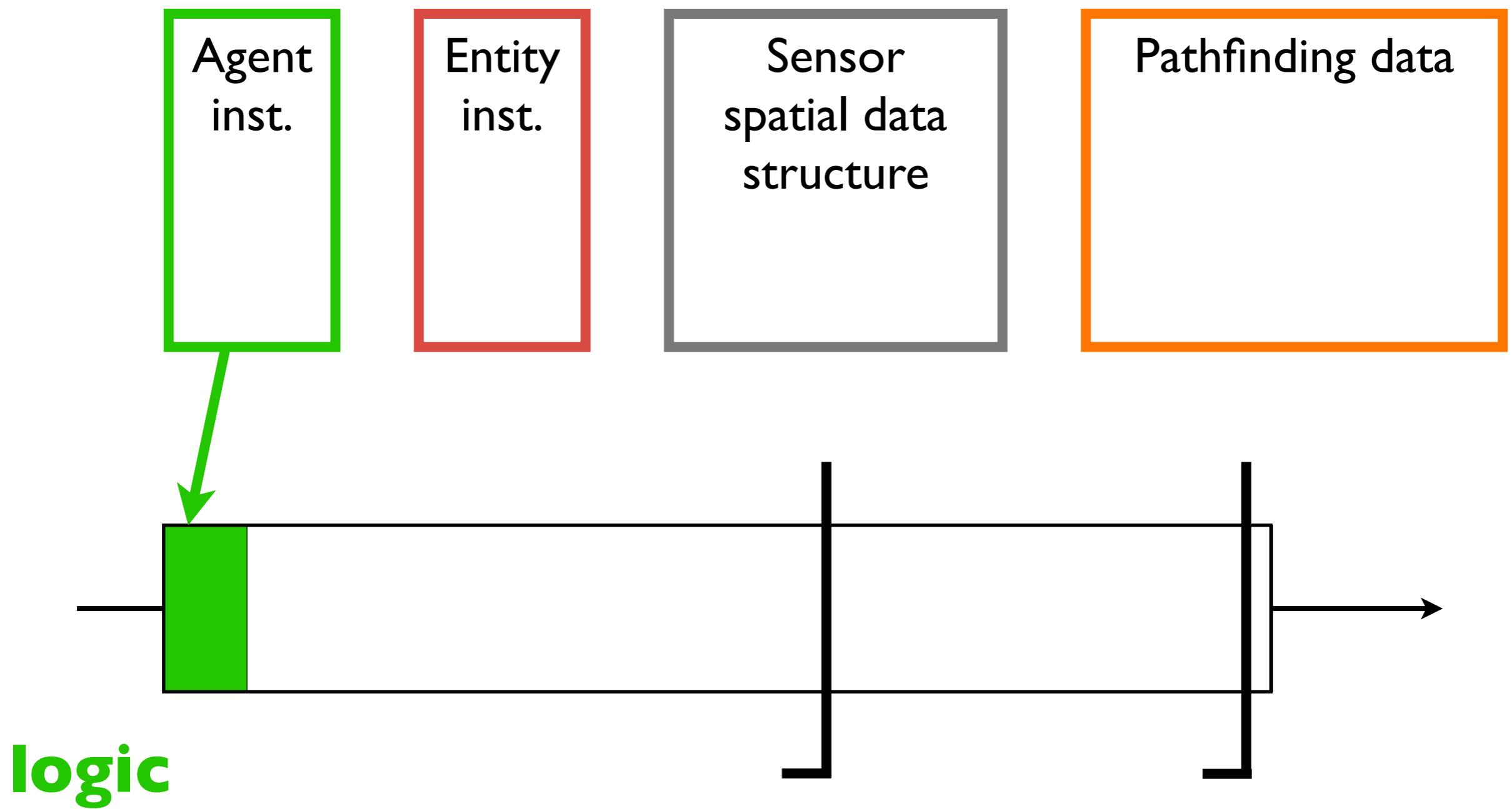
Entity  
inst.

Sensor  
spatial data  
structure

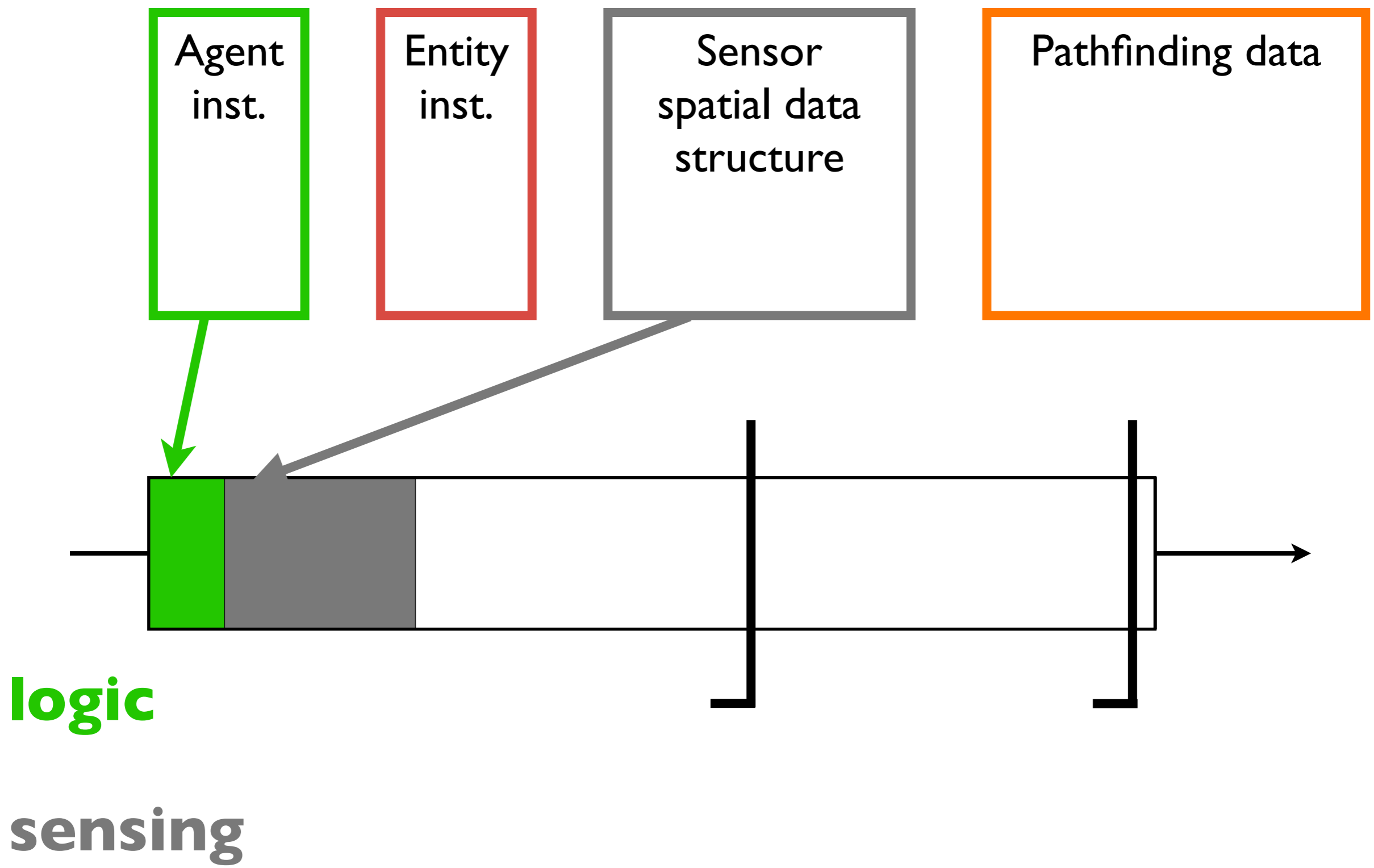
Pathfinding data



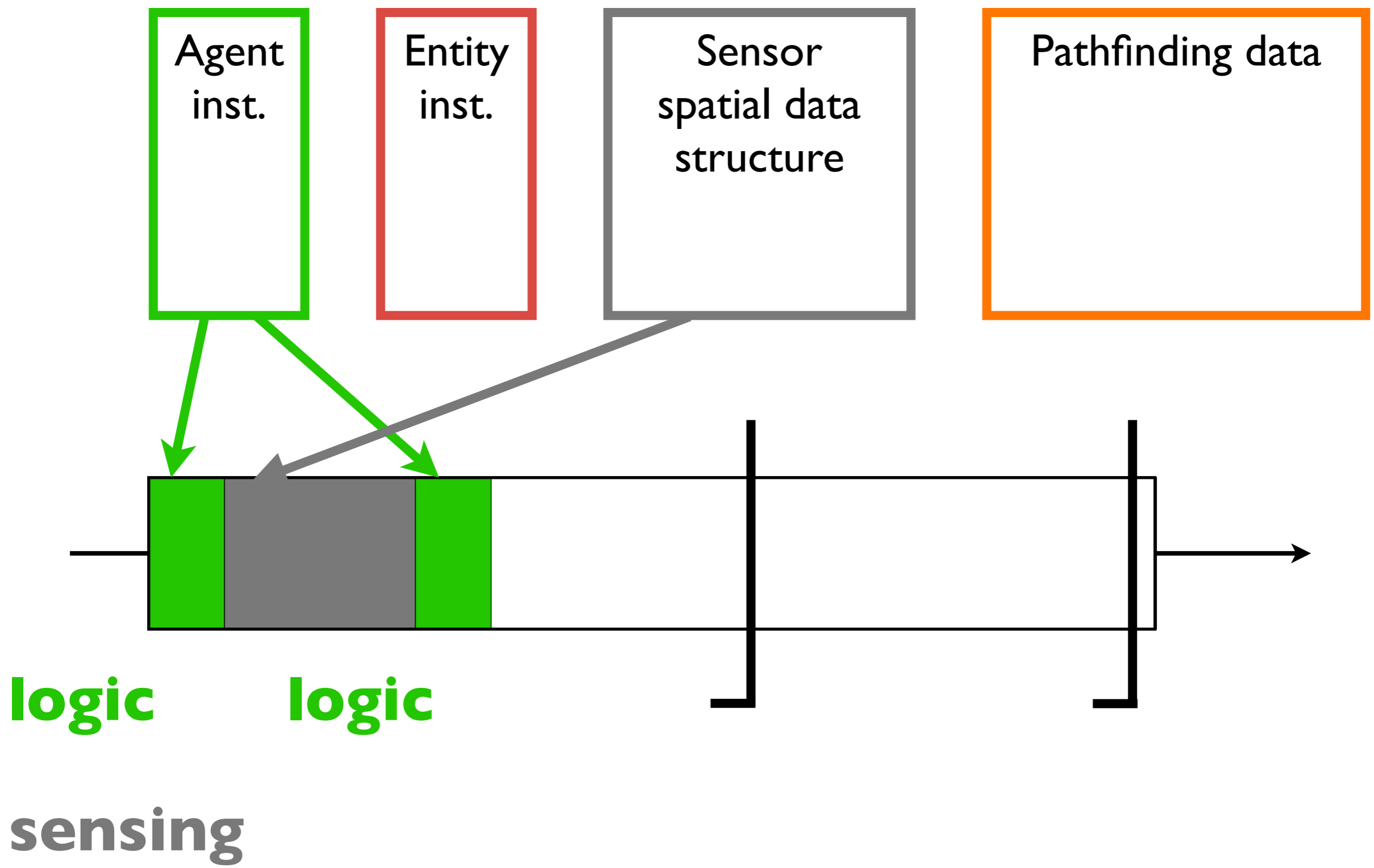
... looks different for every agent (order, timing, branches)  
- agents run in parallel on many cores -> random mem accesses, no coordination  
-> bandwidth?? memory locality??



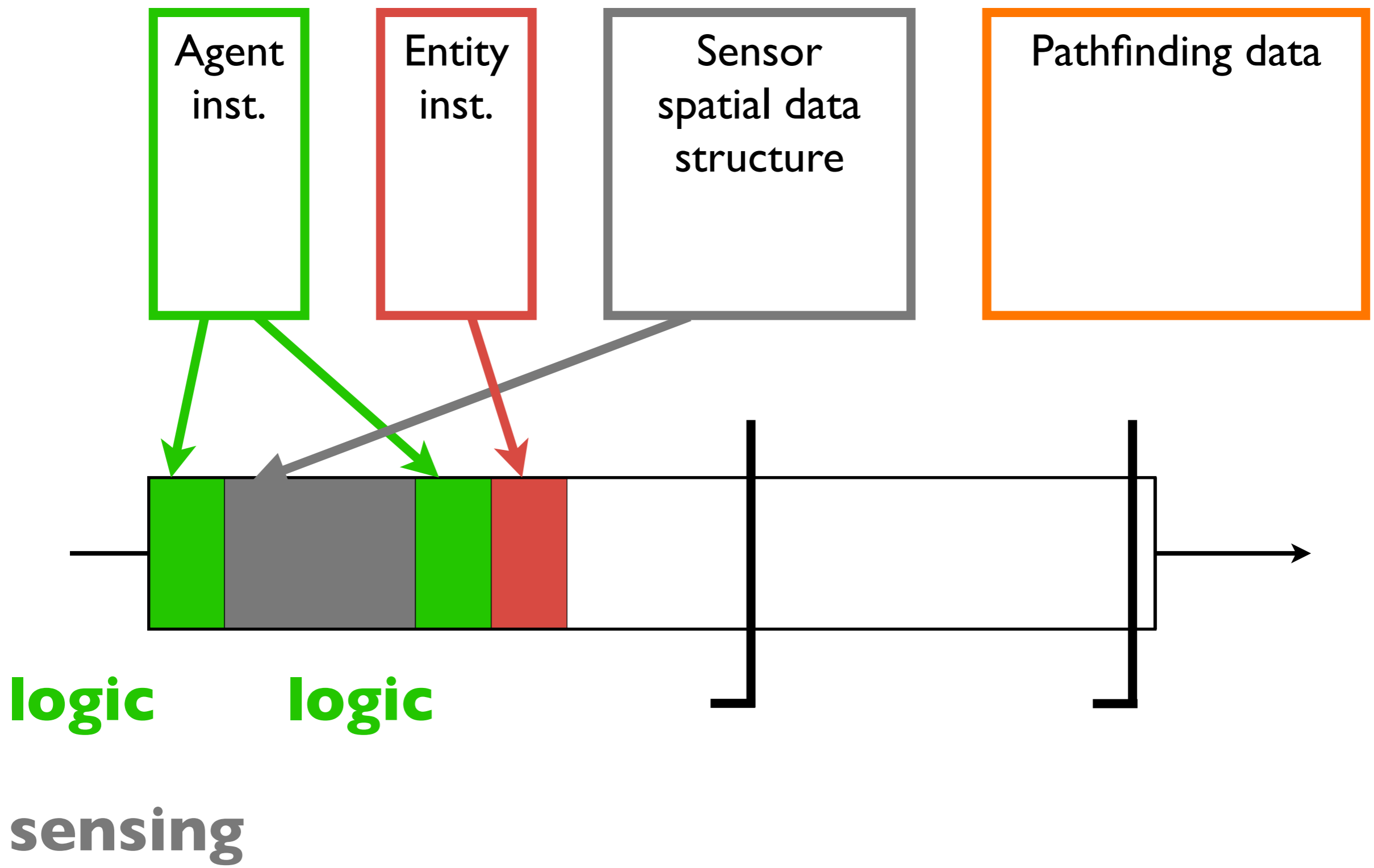
- ... looks different for every agent (order, timing, branches)
- agents run in parallel on many cores -> random mem accesses, no coordination
- > bandwidth?? memory locality??



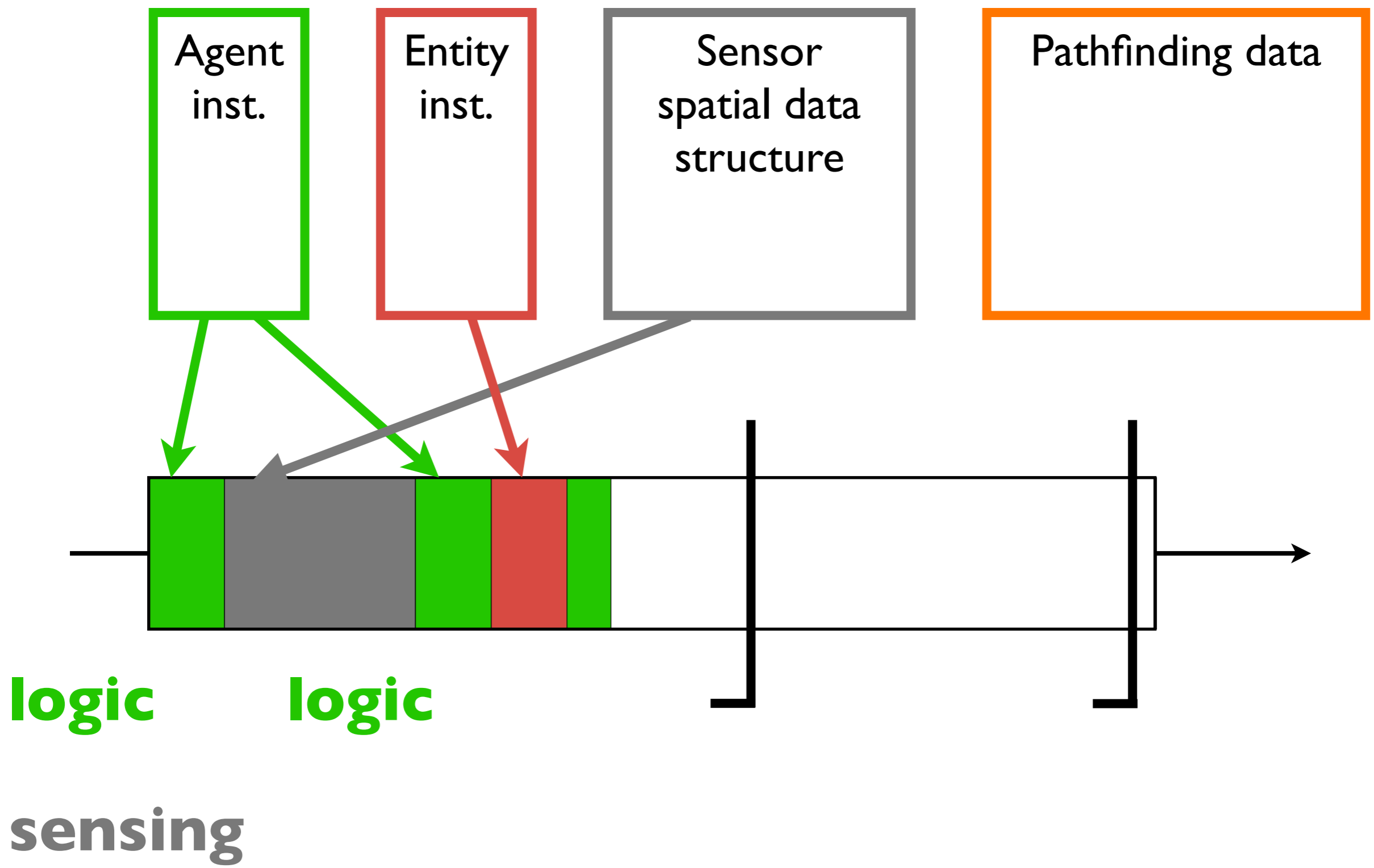
... looks different for every agent (order, timing, branches)  
 - agents run in parallel on many cores -> random mem accesses, no coordination  
 -> bandwidth?? memory locality??



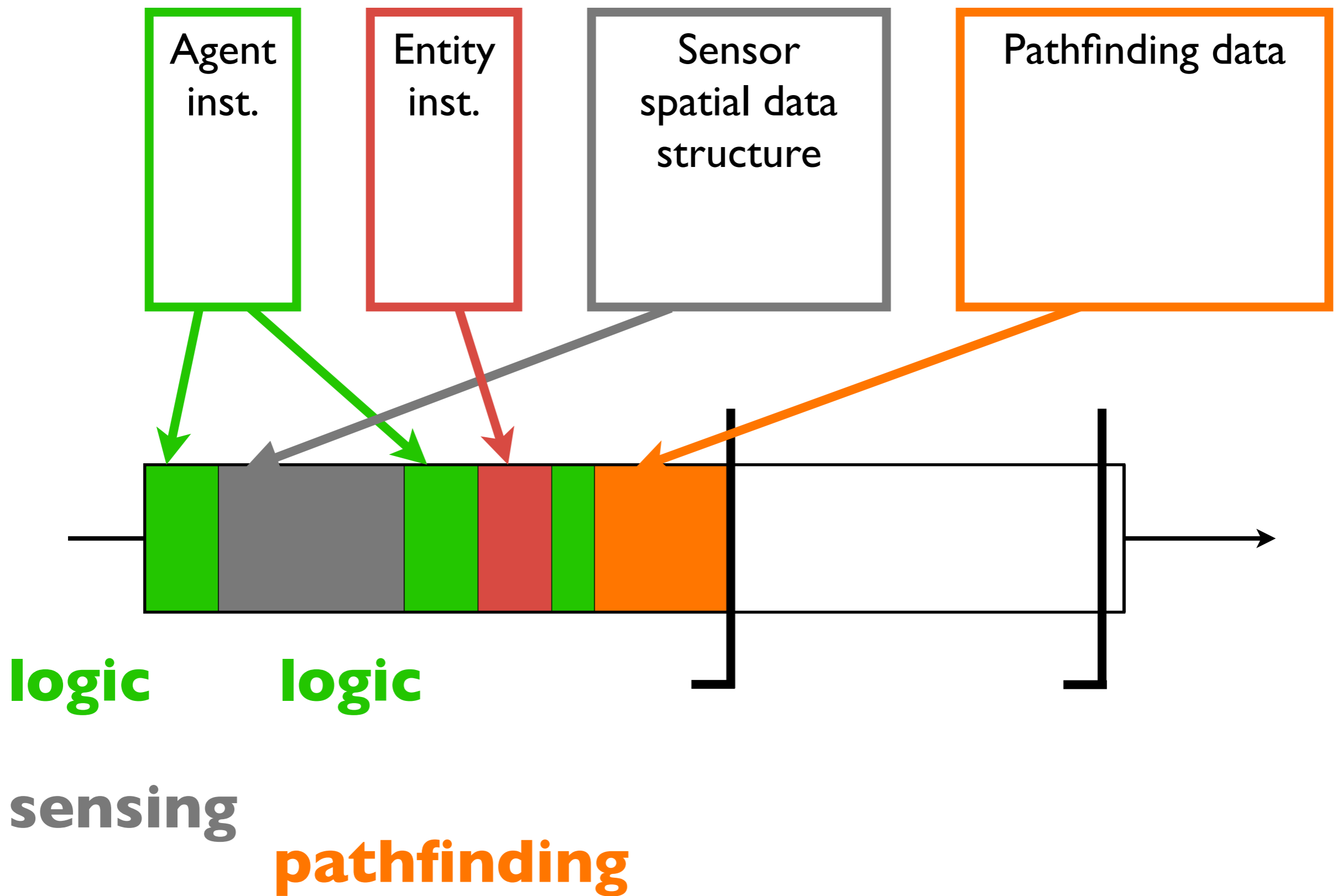
... looks different for every agent (order, timing, branches)  
 - agents run in parallel on many cores -> random mem accesses, no coordination  
 -> bandwidth?? memory locality??



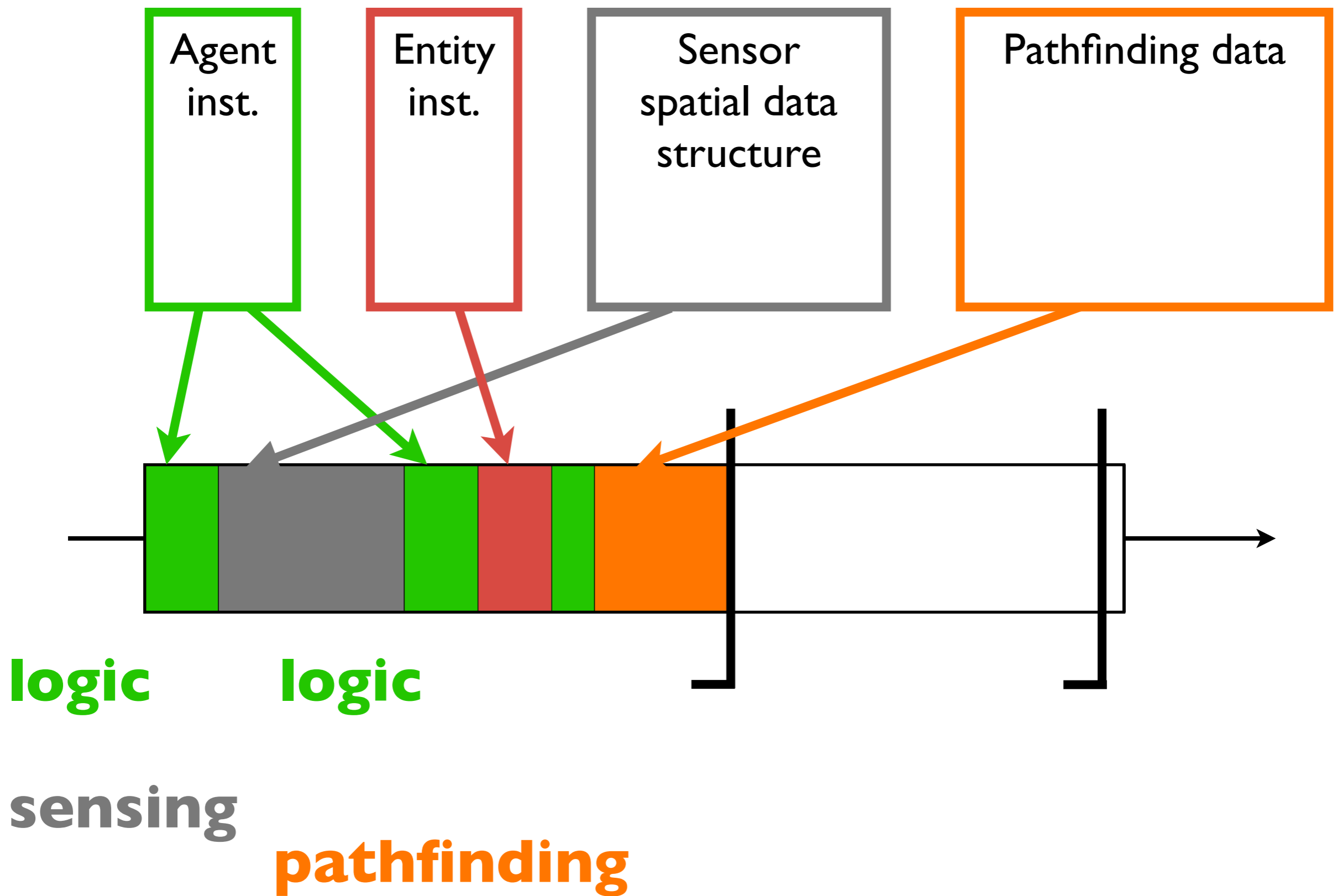
... looks different for every agent (order, timing, branches)  
 - agents run in parallel on many cores -> random mem accesses, no coordination  
 -> bandwidth?? memory locality??



... looks different for every agent (order, timing, branches)  
 - agents run in parallel on many cores -> random mem accesses, no coordination  
 -> bandwidth?? memory locality??



... looks different for every agent (order, timing, branches)  
 - agents run in parallel on many cores -> random mem accesses, no coordination  
 -> bandwidth?? memory locality??



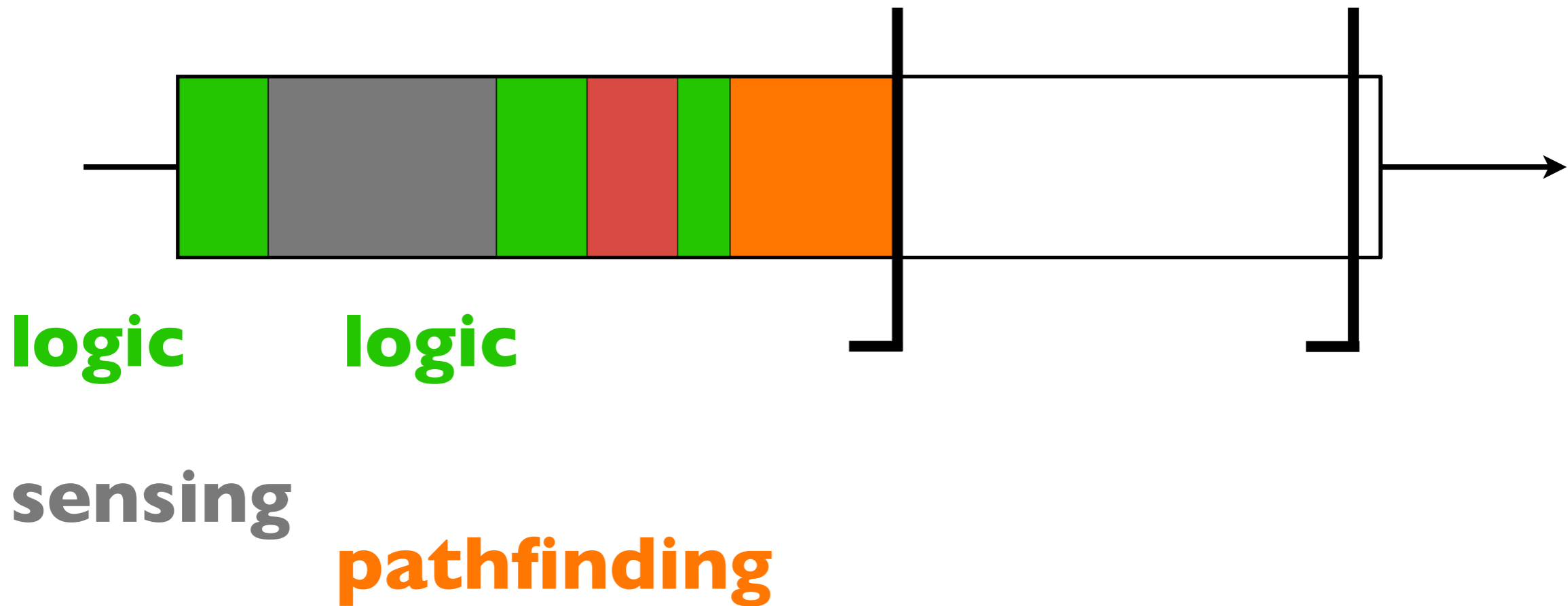
Agent logic, sensing, pathfinding, terrain analysis, planning  
 -> all different aspects (of the behavior) of an agent

Agent  
inst.

Entity  
inst.

Sensor  
spatial data  
structure

Pathfinding data

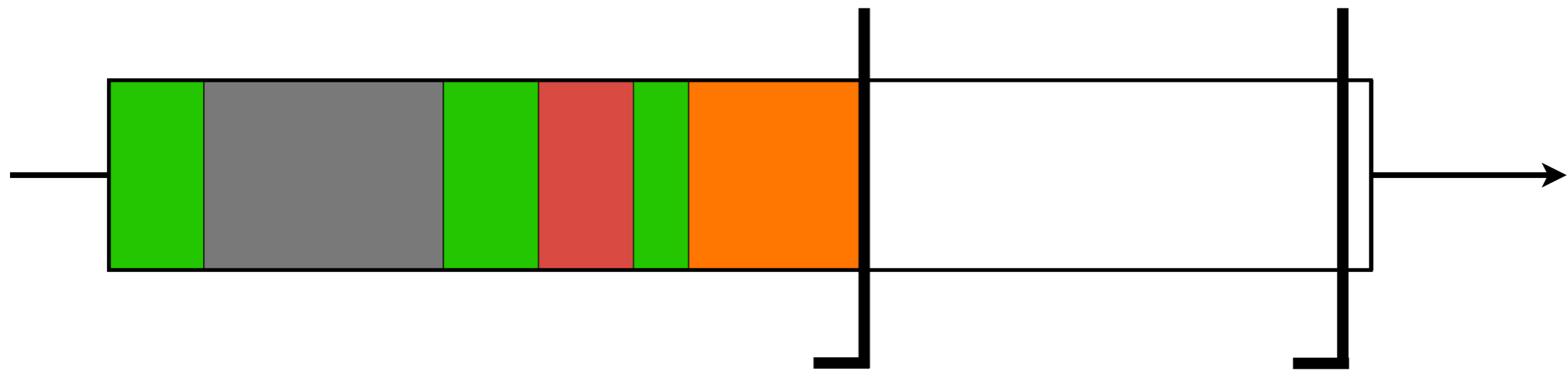


Agent  
inst.

Entity  
inst.

Sensor  
spatial data  
structure

Pathfinding data



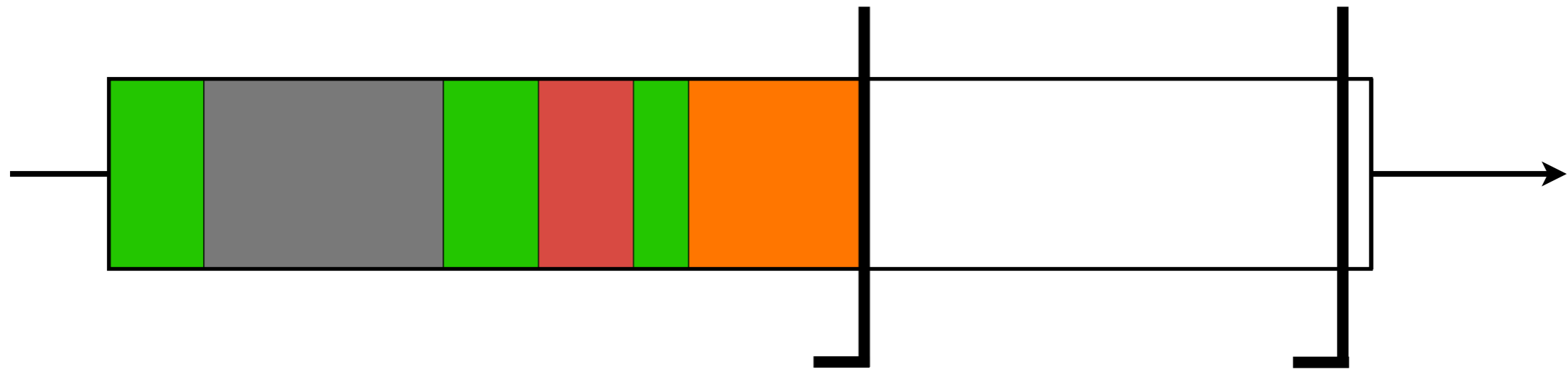
**pathfinding** **sensing** **logic**  
**logic**

Agent  
inst.

Entity  
inst.

Sensor  
spatial data  
structure

Pathfinding data



**pathfinding** sensing logic  
logic

# Aspects

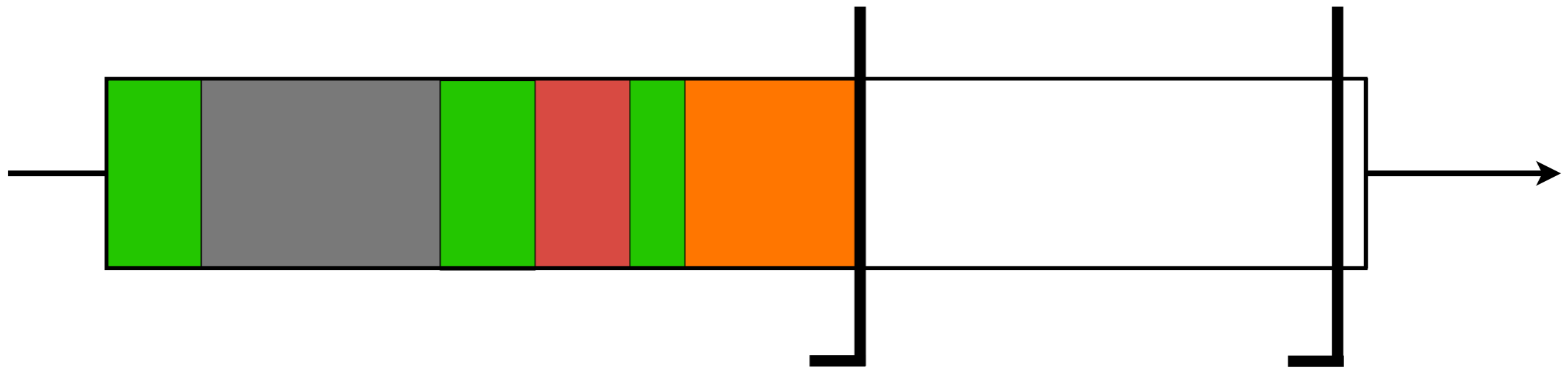
Agent logic, sensing, pathfinding, terrain analysis, planning  
-> all different aspects (of the behavior) of an agent

Agent  
inst.

Entity  
inst.

Sensor  
spatial data  
structure

Pathfinding data



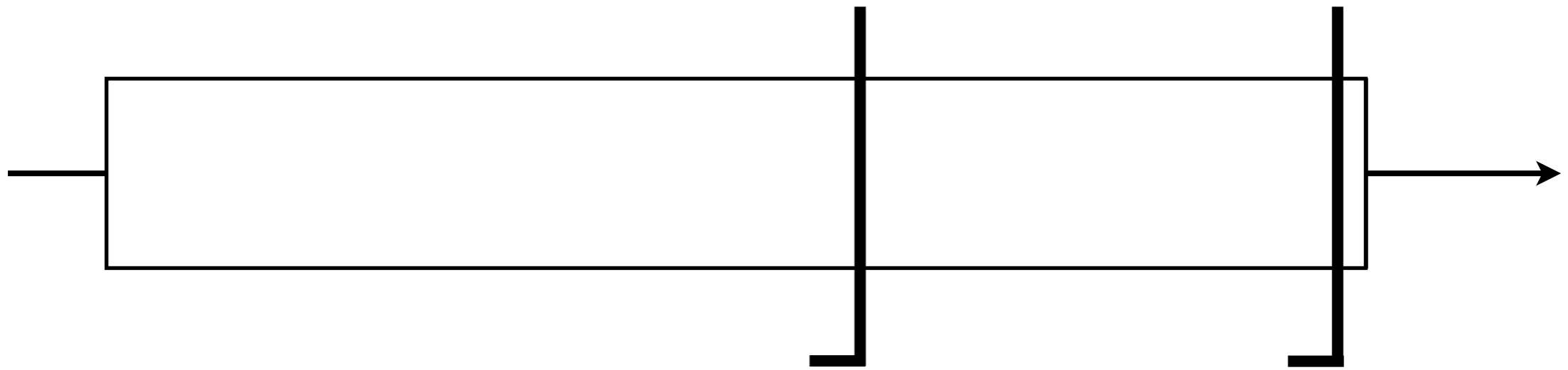
**pathfinding** sensing logic  
logic

Agent  
inst.

Entity  
inst.

Sensor  
spatial data  
structure

Pathfinding data

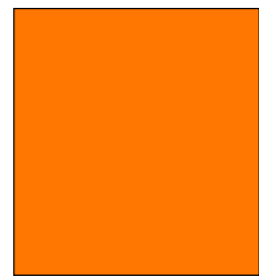


**pathfinding**

**sensing**

**logic**

**logic**

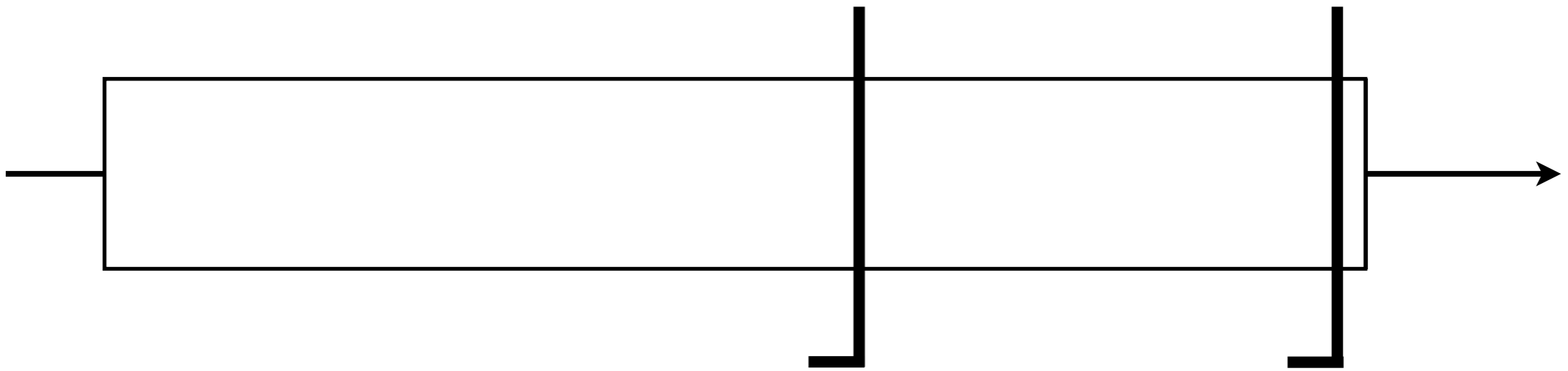


Agent  
inst.

Entity  
inst.

Sensor  
spatial data  
structure

Pathfinding data

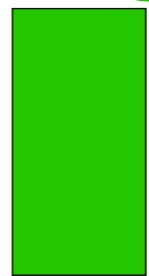
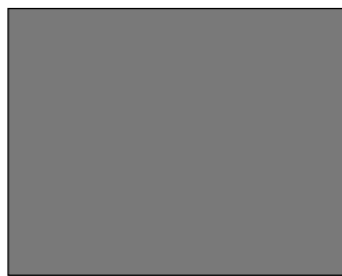
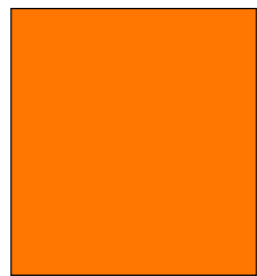


**pathfinding**

**sensing**

**logic**

**logic**

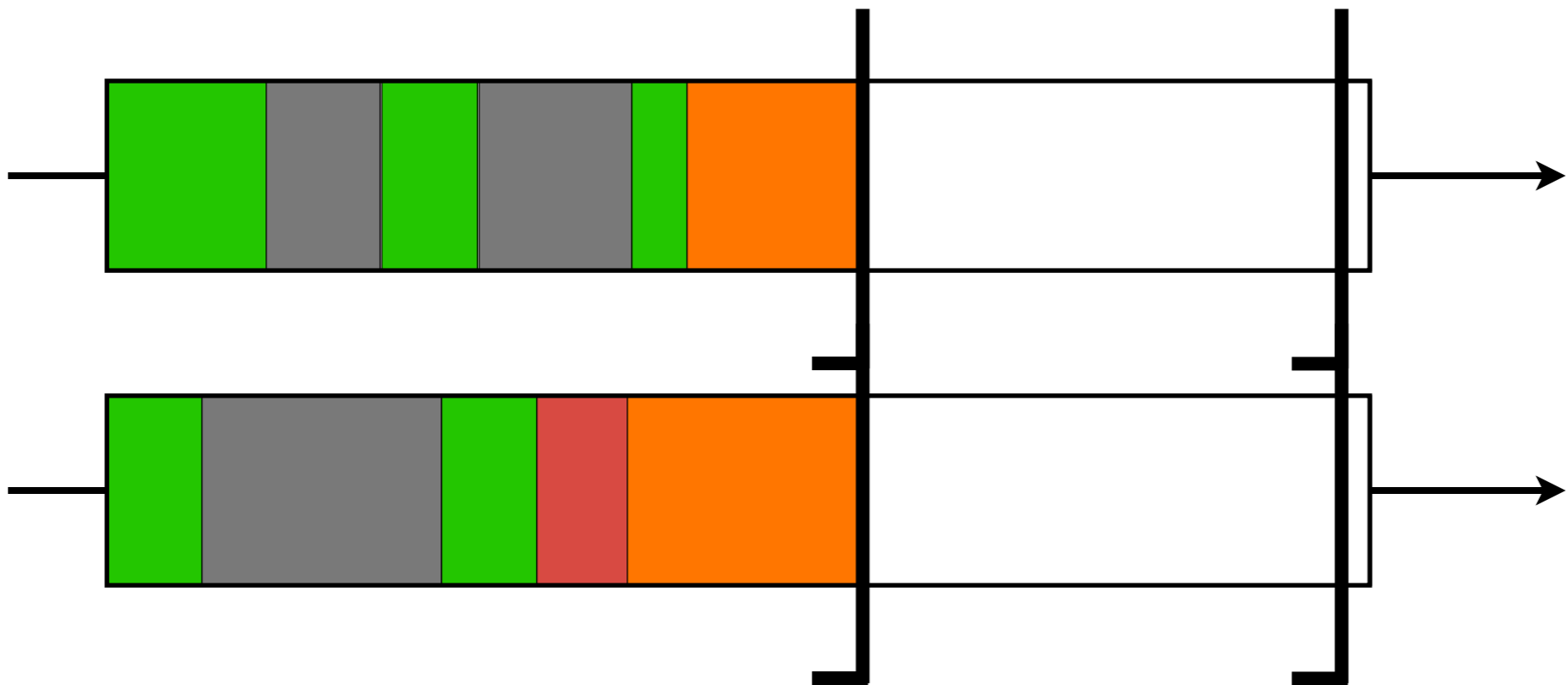


Agent  
inst.

Entity  
inst.

Sensor  
spatial data  
structure

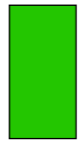
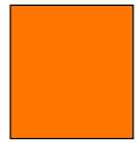
Pathfinding data



**pathfinding**

**sensing**

**logic**  
**logic**

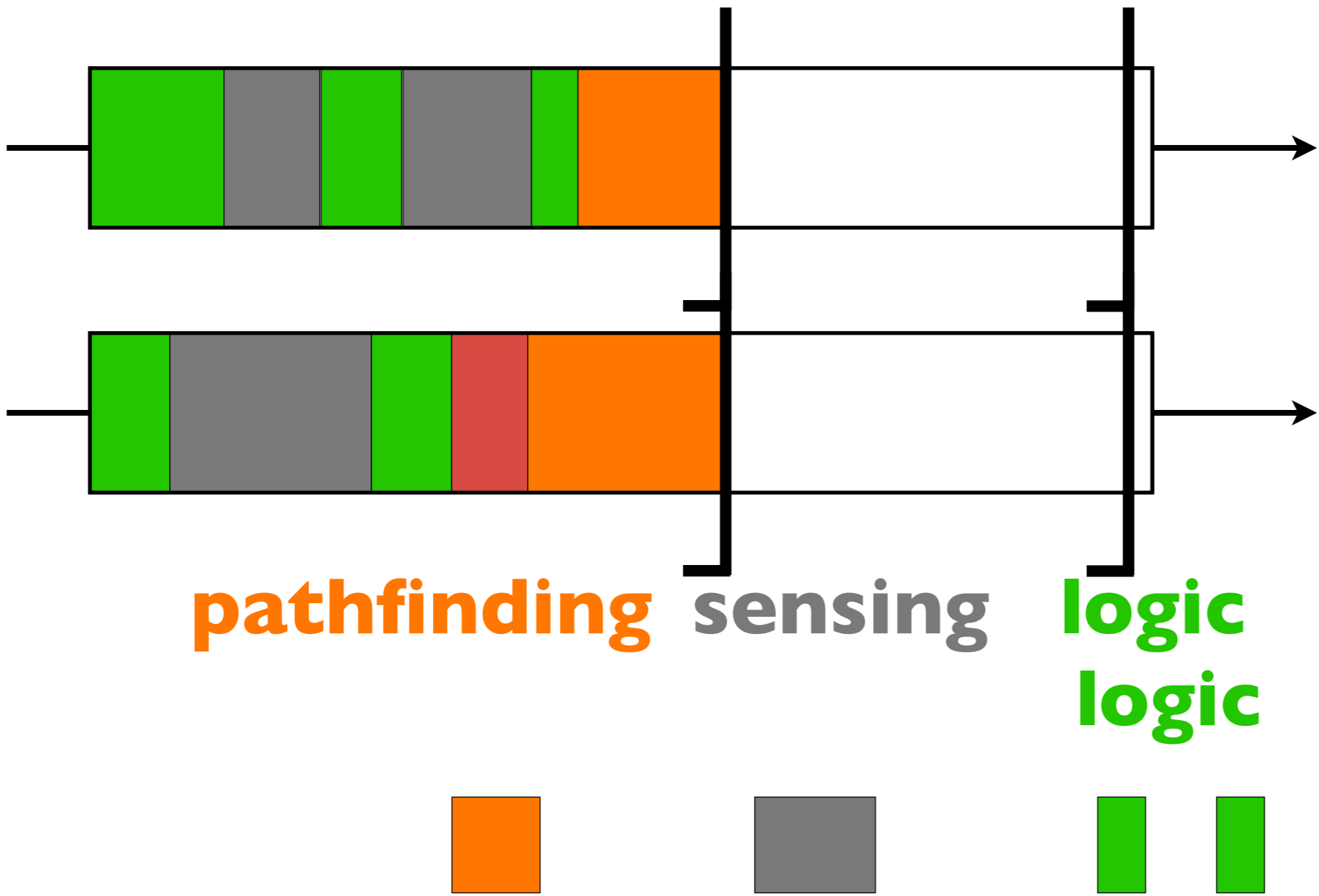


Agent  
inst.

Entity  
inst.

Sensor  
spatial data  
structure

Pathfinding data

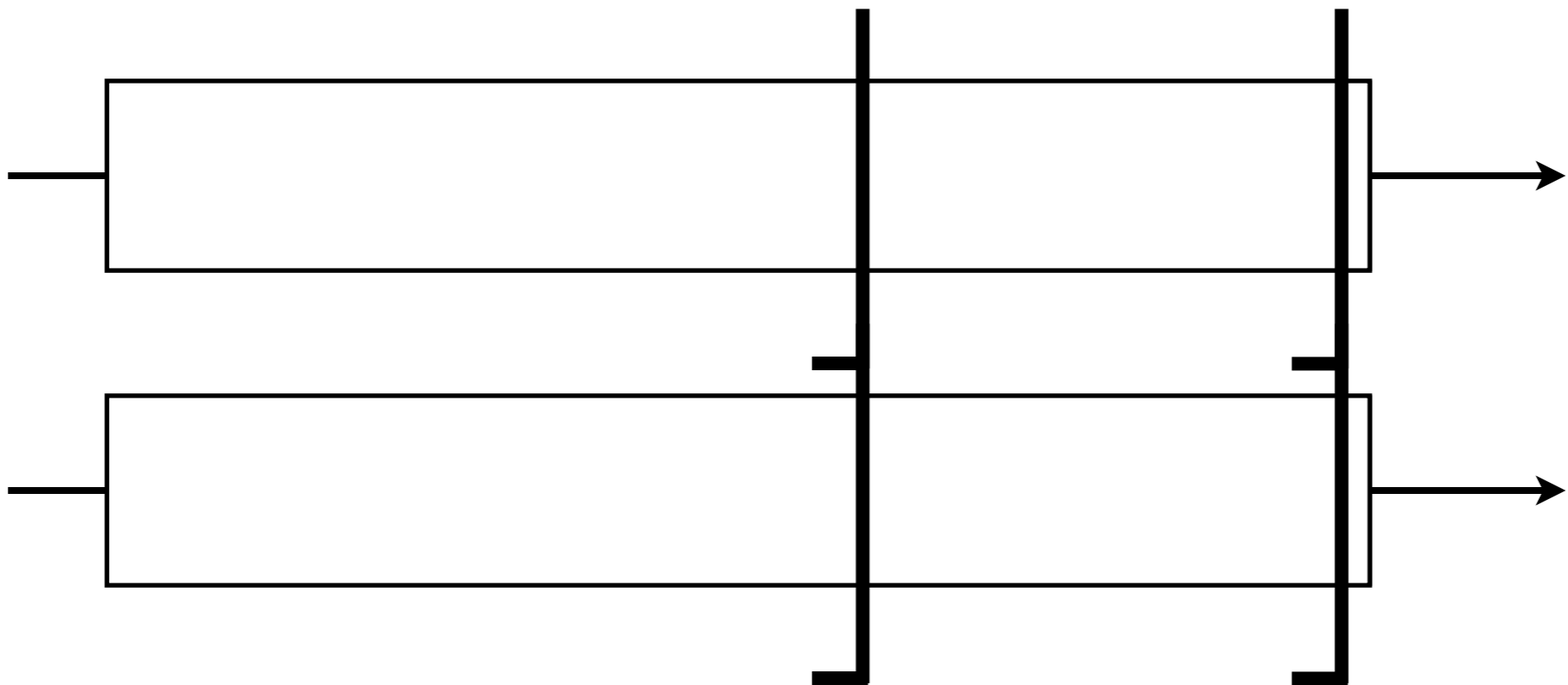


Agent  
inst.

Entity  
inst.

Sensor  
spatial data  
structure

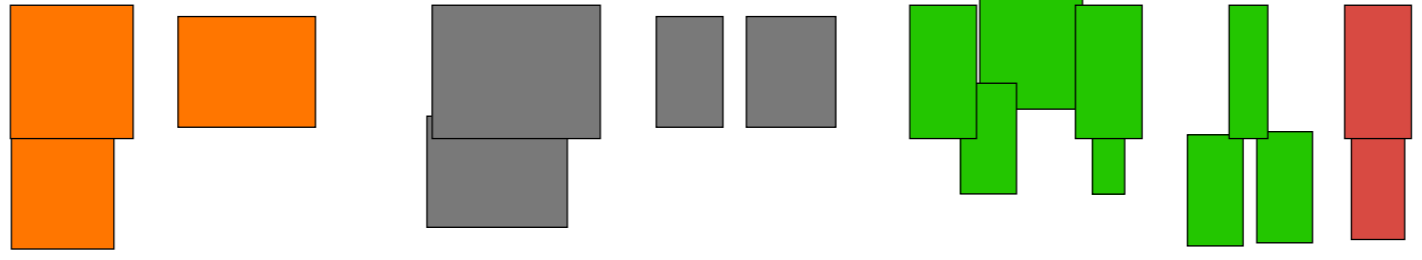
Pathfinding data



pathfinding

sensing

logic  
logic

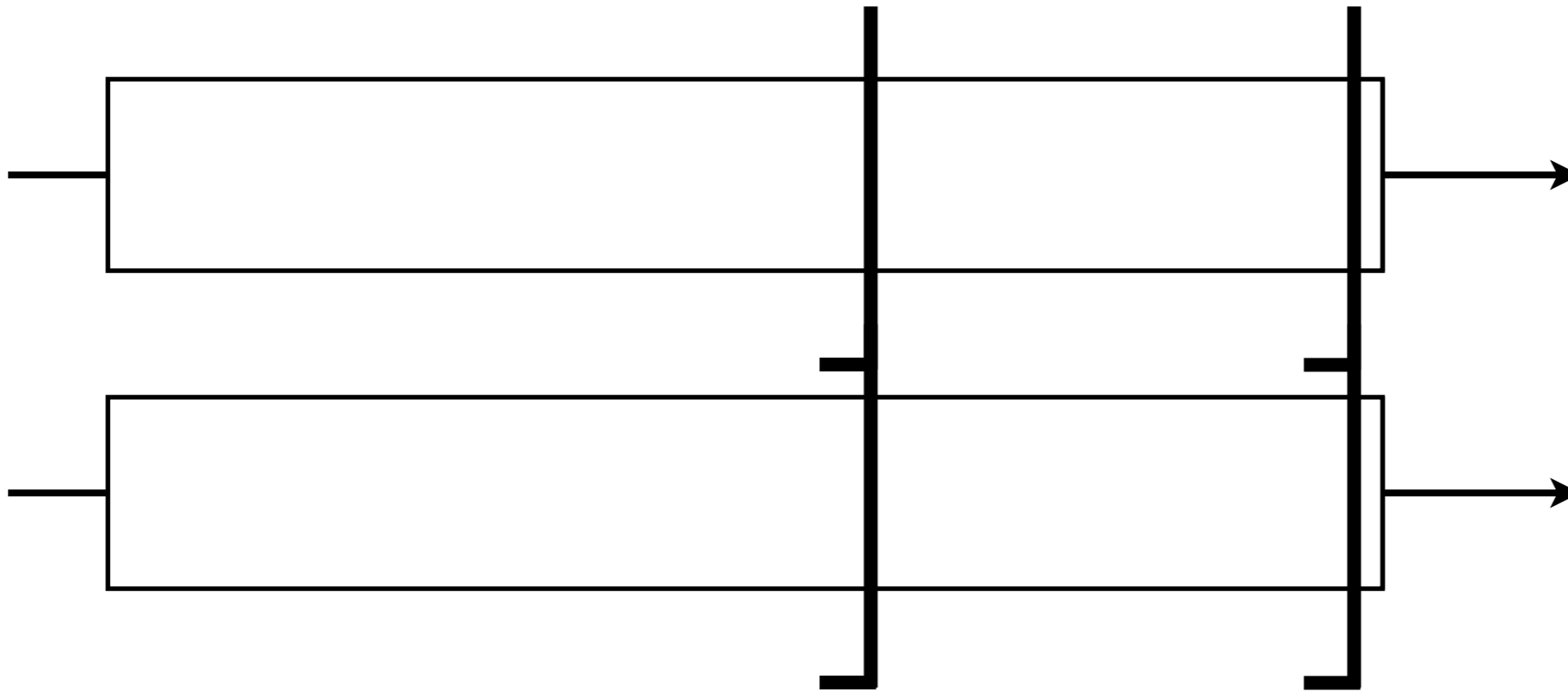


Agent  
inst.

Entity  
inst.

Sensor  
spatial data  
structure

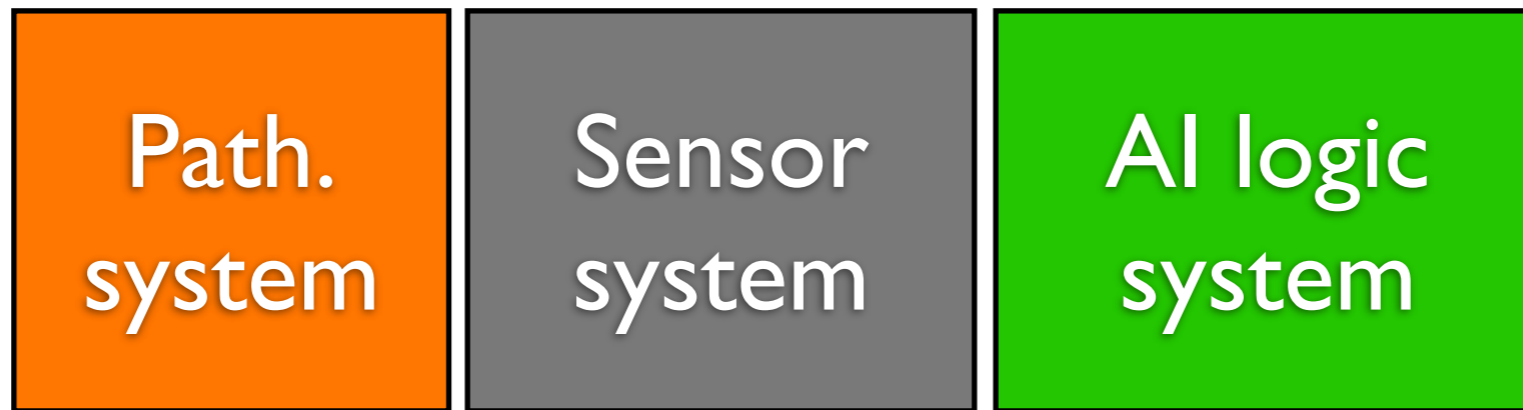
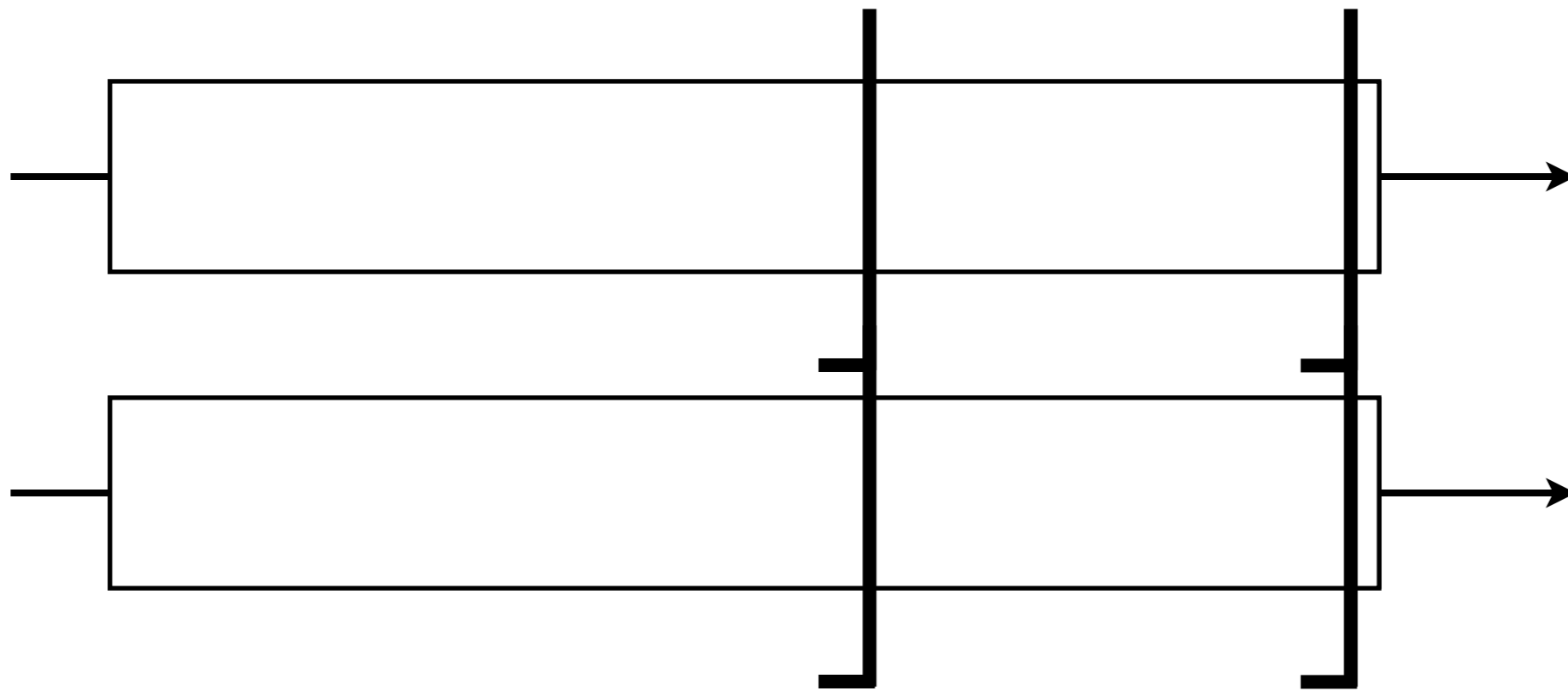
Pathfinding data

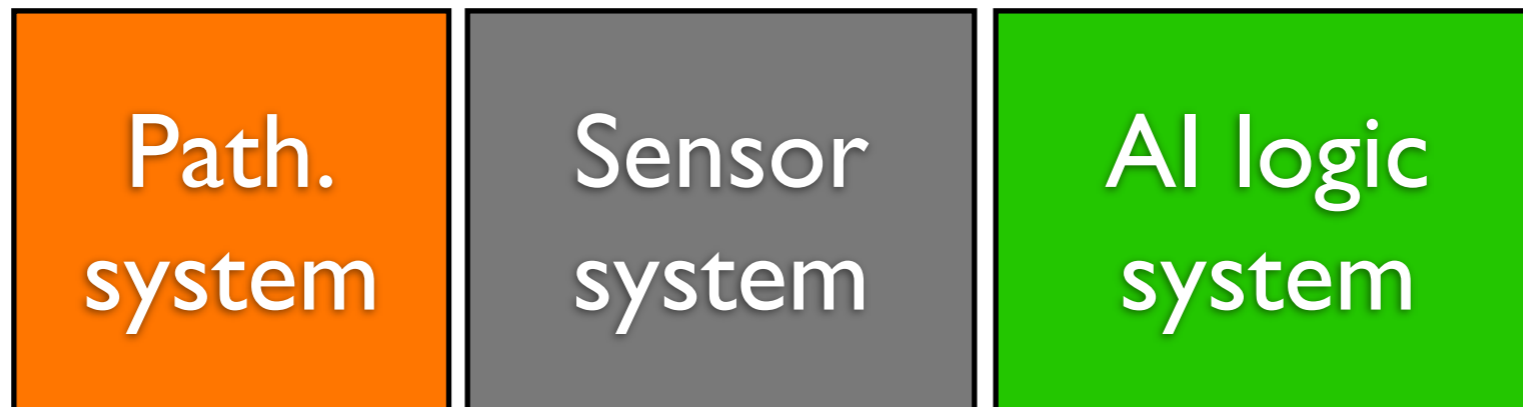
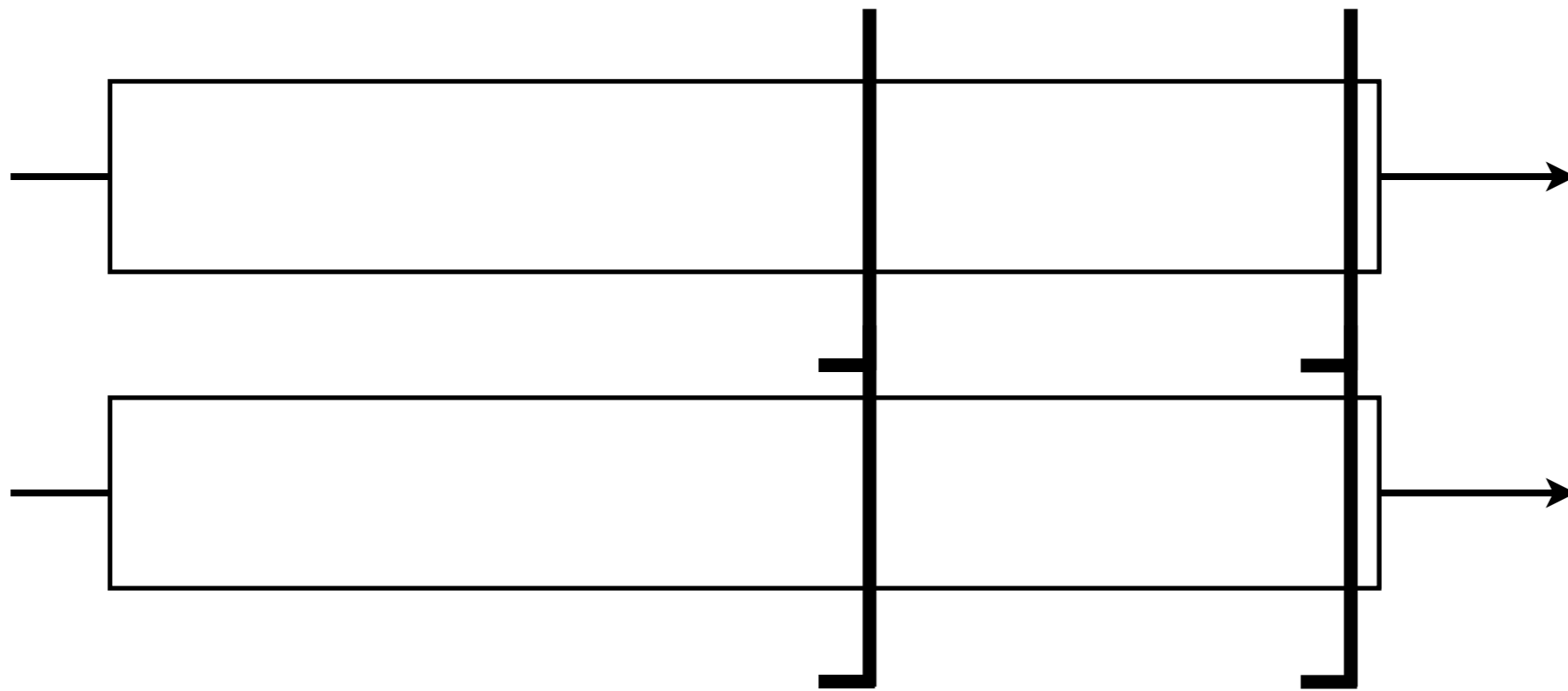


Path.  
system

Sensor  
system

AI logic  
system

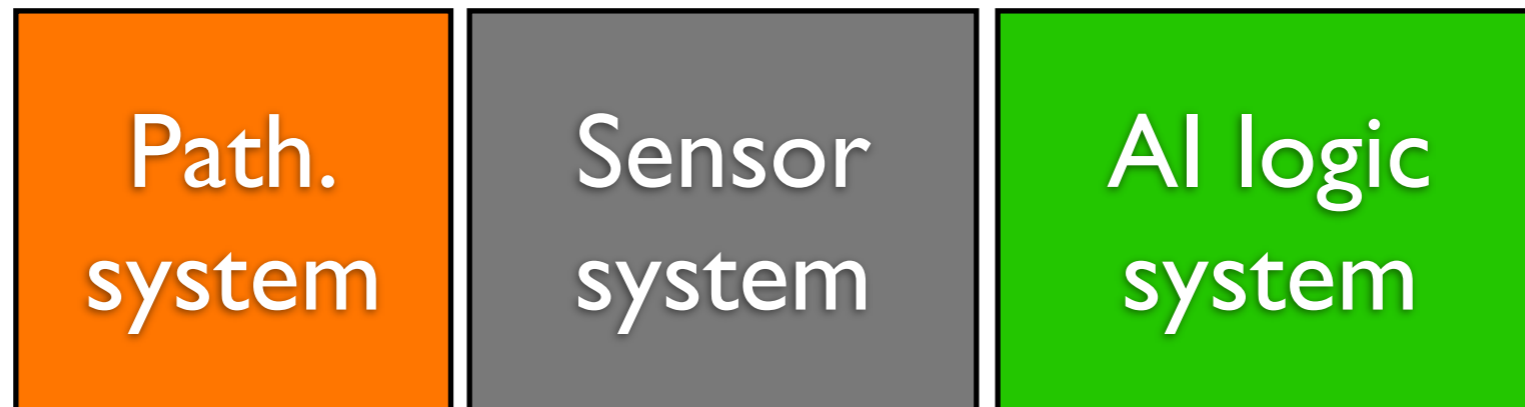




Path.  
system

Sensor  
system

AI logic  
system



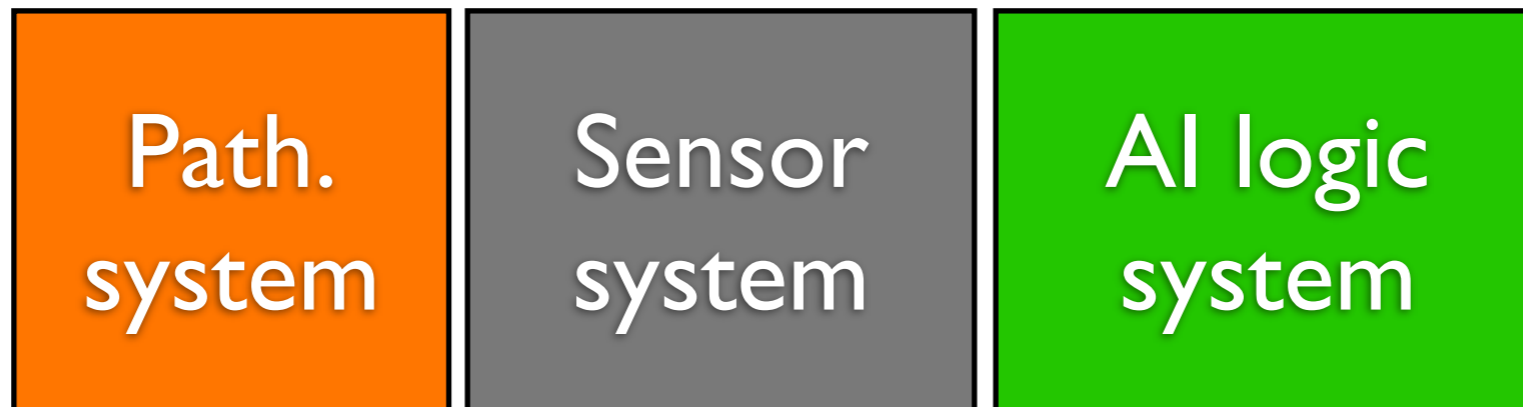
**No single update func per agent anymore...**

System have dependencies -> independent systems could run in parallel, asynchronously

Stages of our AI (pipeline)

Aspects needed every step always computed, other deferred upon request -> async calls

Manage parallel resources if multiple systems run in parallel!



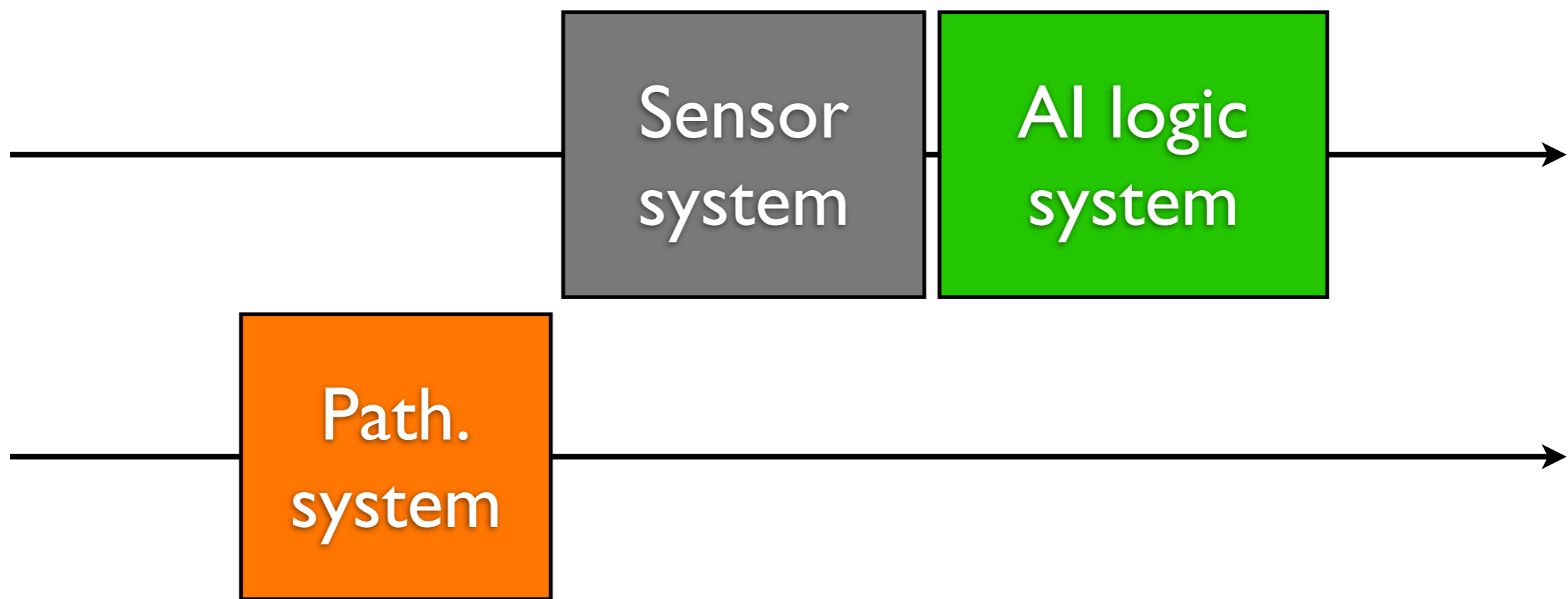
### **No single update func per agent anymore...**

System have dependencies -> independent systems could run in parallel, asynchronously

Stages of our AI (pipeline)

Aspects needed every step always computed, other deferred upon request -> async calls

Manage parallel resources if multiple systems run in parallel!



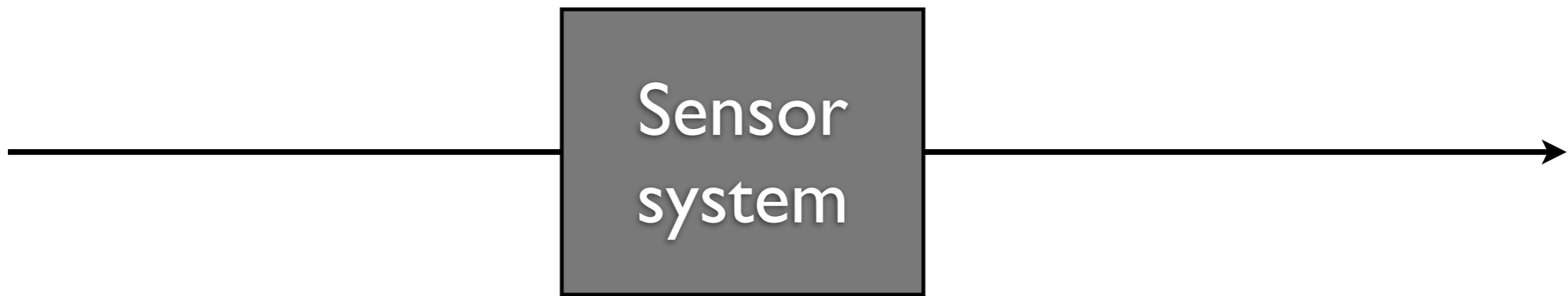
## **No single update func per agent anymore...**

System have dependencies -> independent systems could run in parallel, asynchronously

Stages of our AI (pipeline)

Aspects needed every step always computed, other deferred upon request -> async calls

Manage parallel resources if multiple systems run in parallel!



Sensor system data

Sensor system execution

Sensor system data

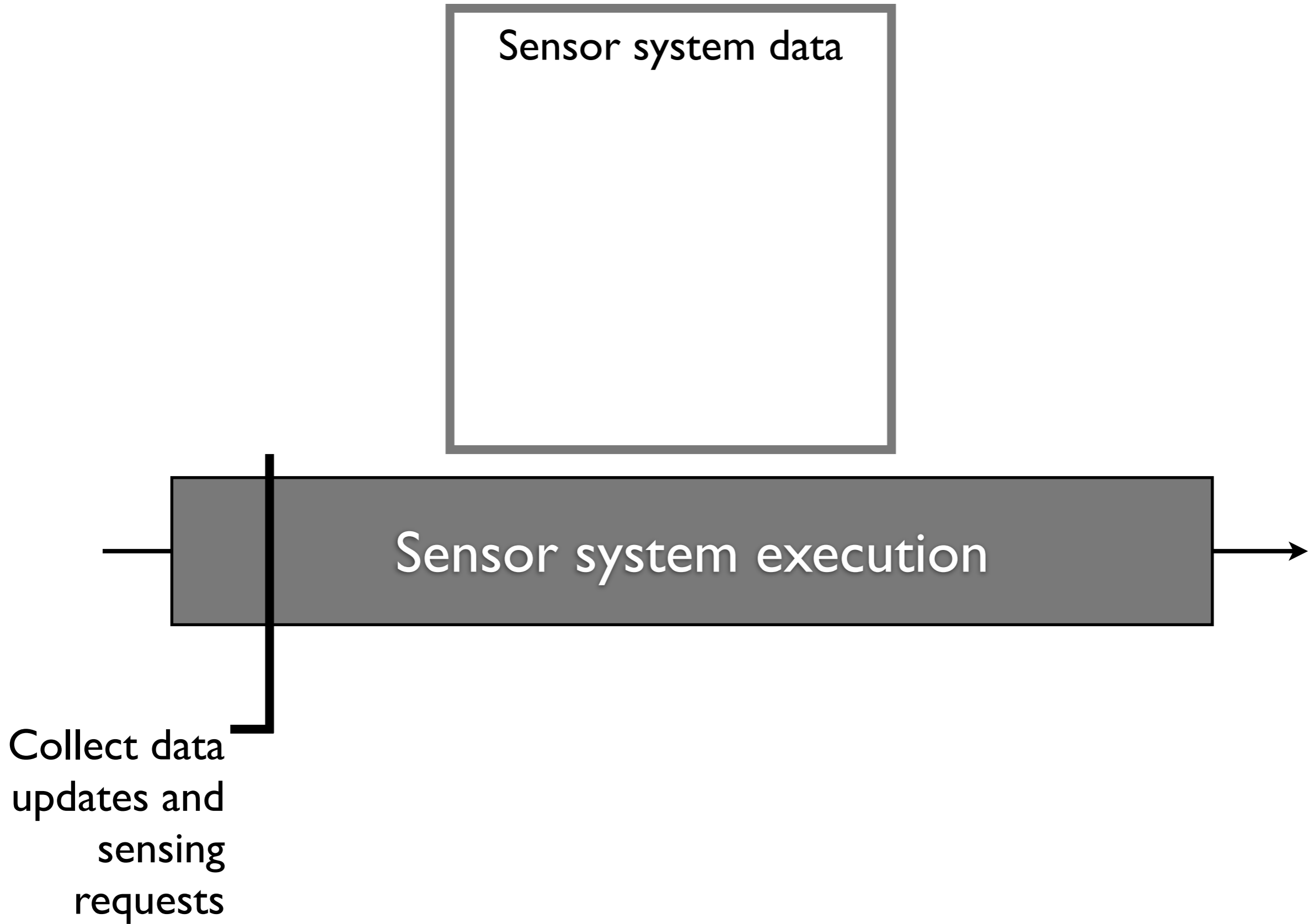


Sensor system execution

Sensor system execution runs through several stages

-> implicit syncing of stages (barriers)

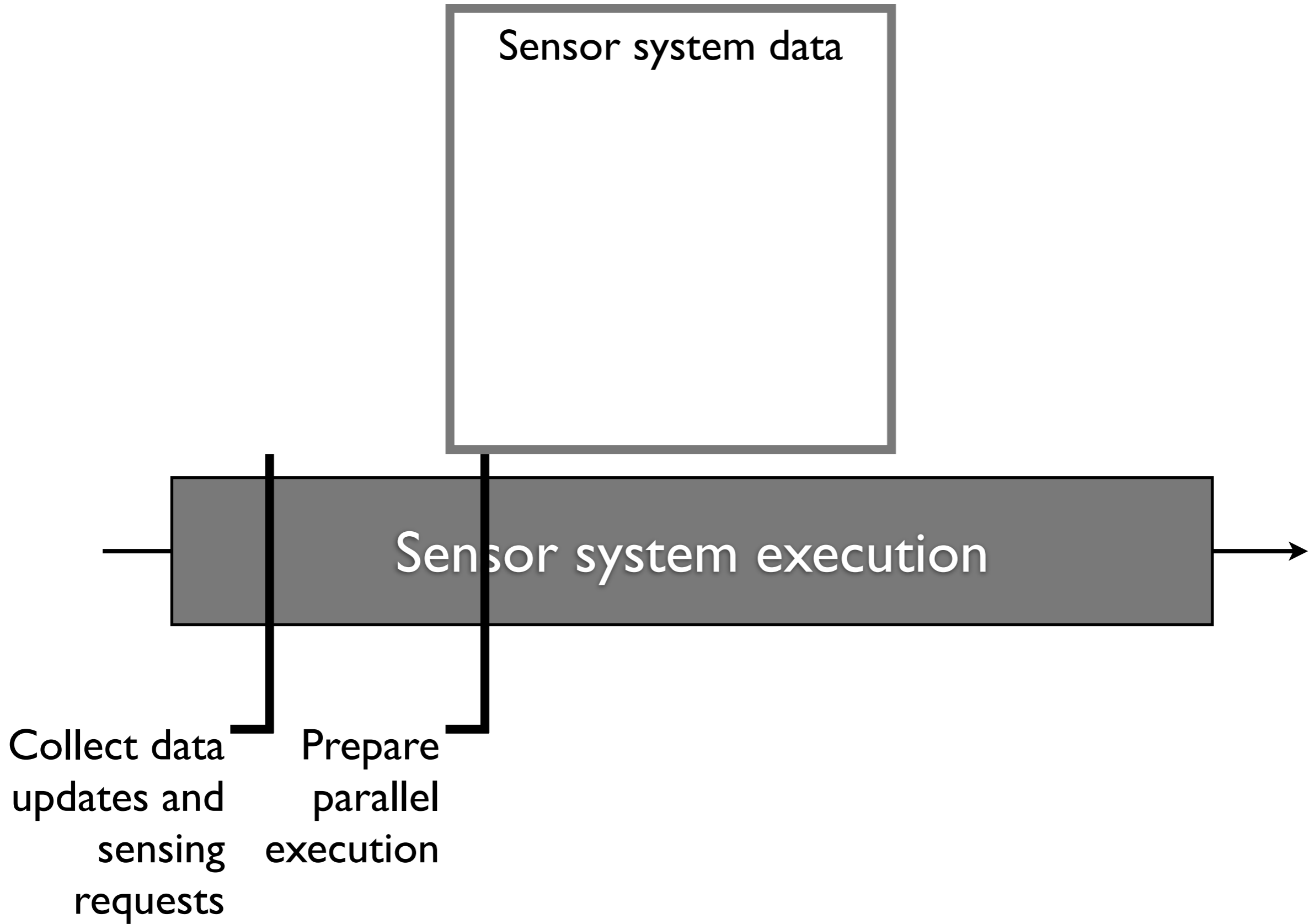
-> **first stage always active to collect input for next time step**



Sensor system execution runs through several stages

-> implicit syncing of stages (barriers)

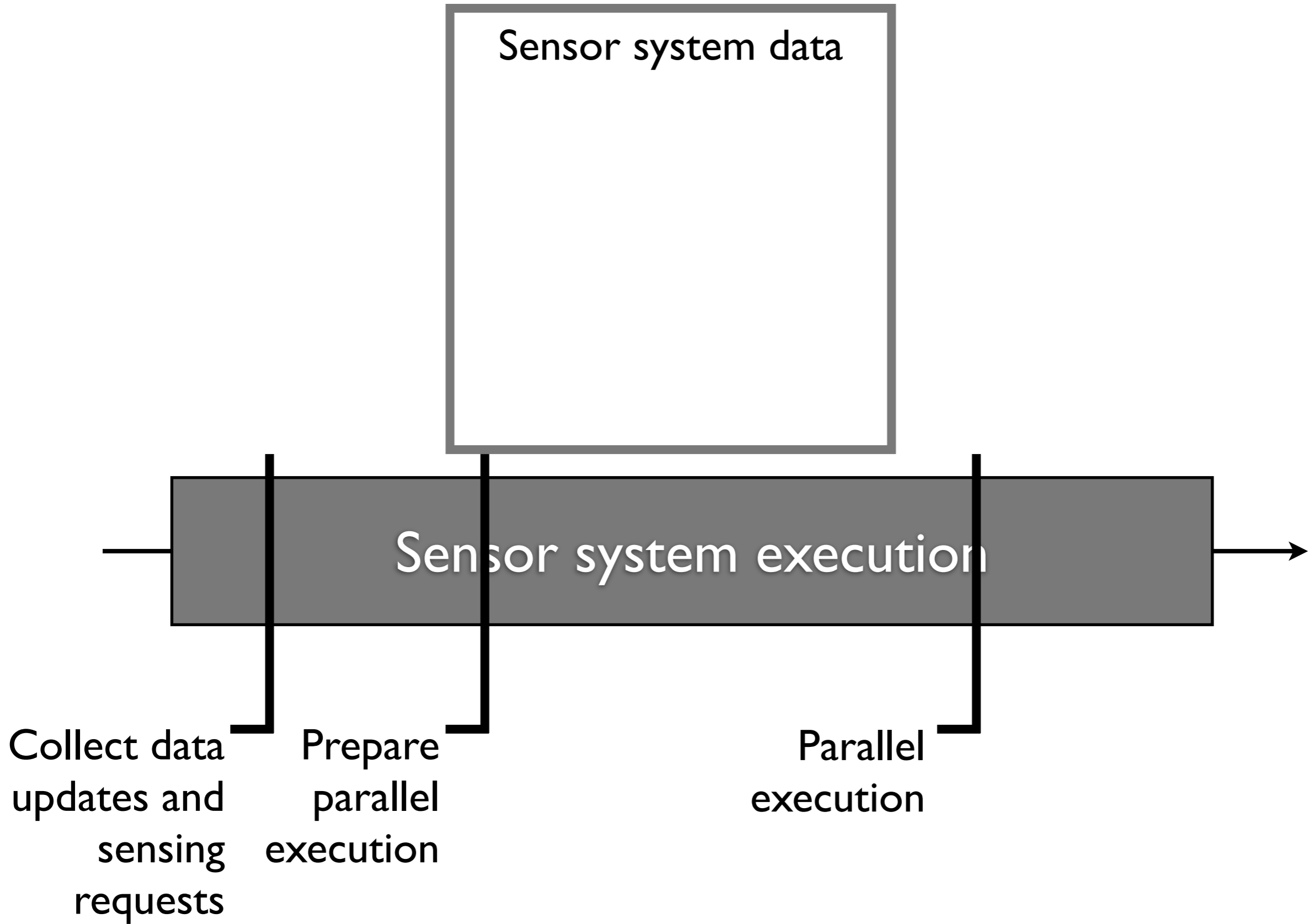
-> **first stage always active to collect input for next time step**



Sensor system execution runs through several stages

-> implicit syncing of stages (barriers)

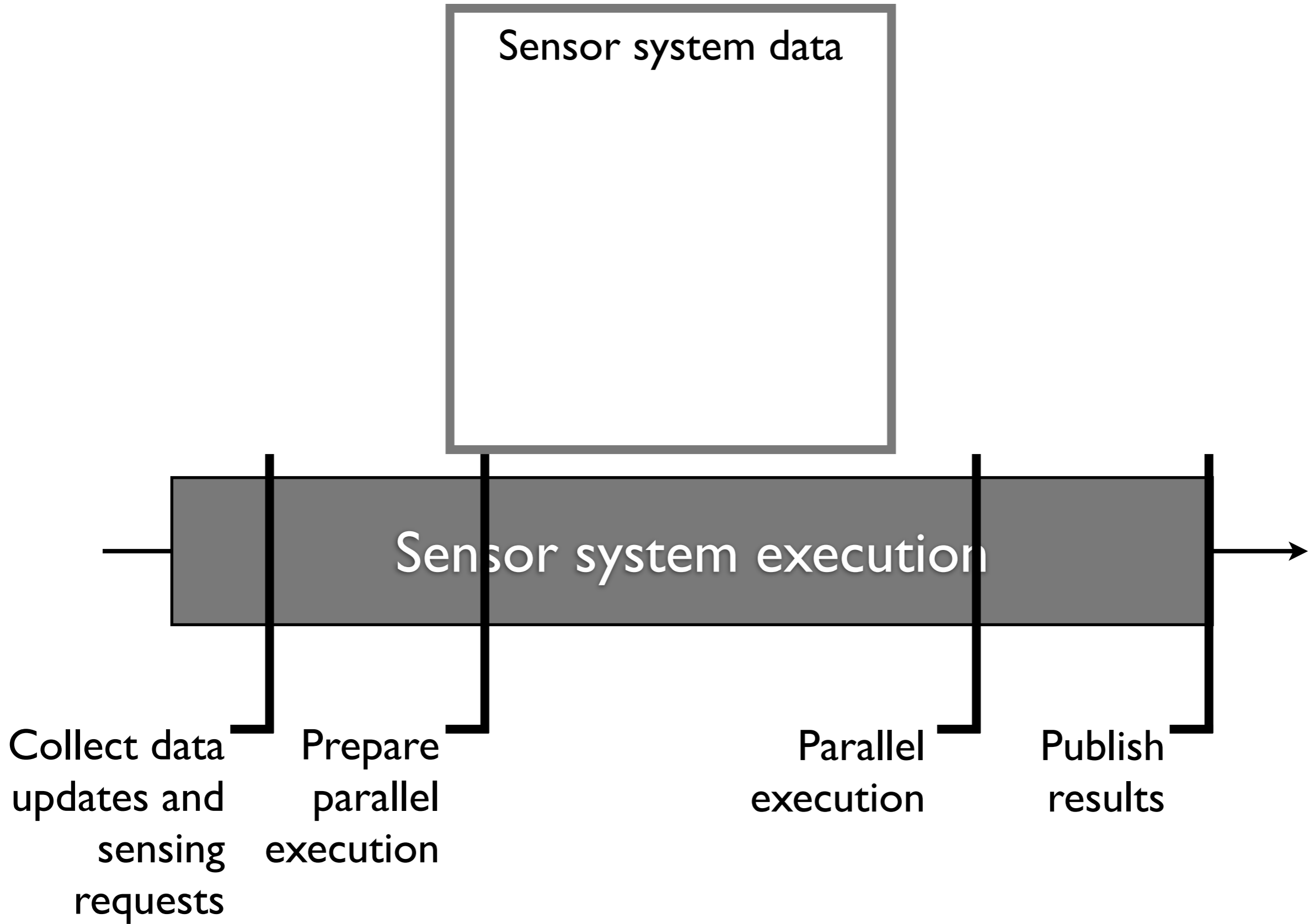
-> **first stage always active to collect input for next time step**



Sensor system execution runs through several stages

-> implicit syncing of stages (barriers)

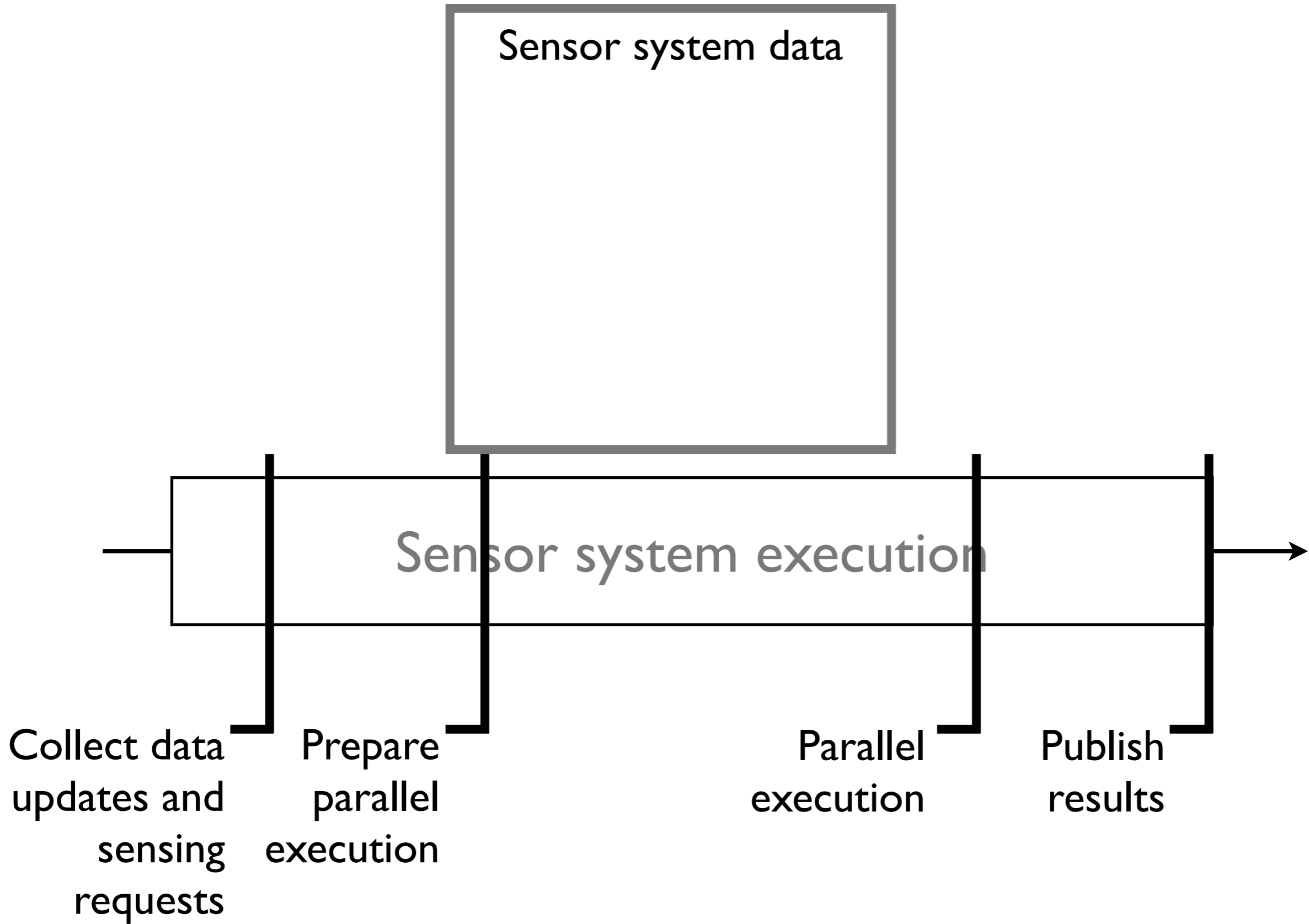
-> **first stage always active to collect input for next time step**



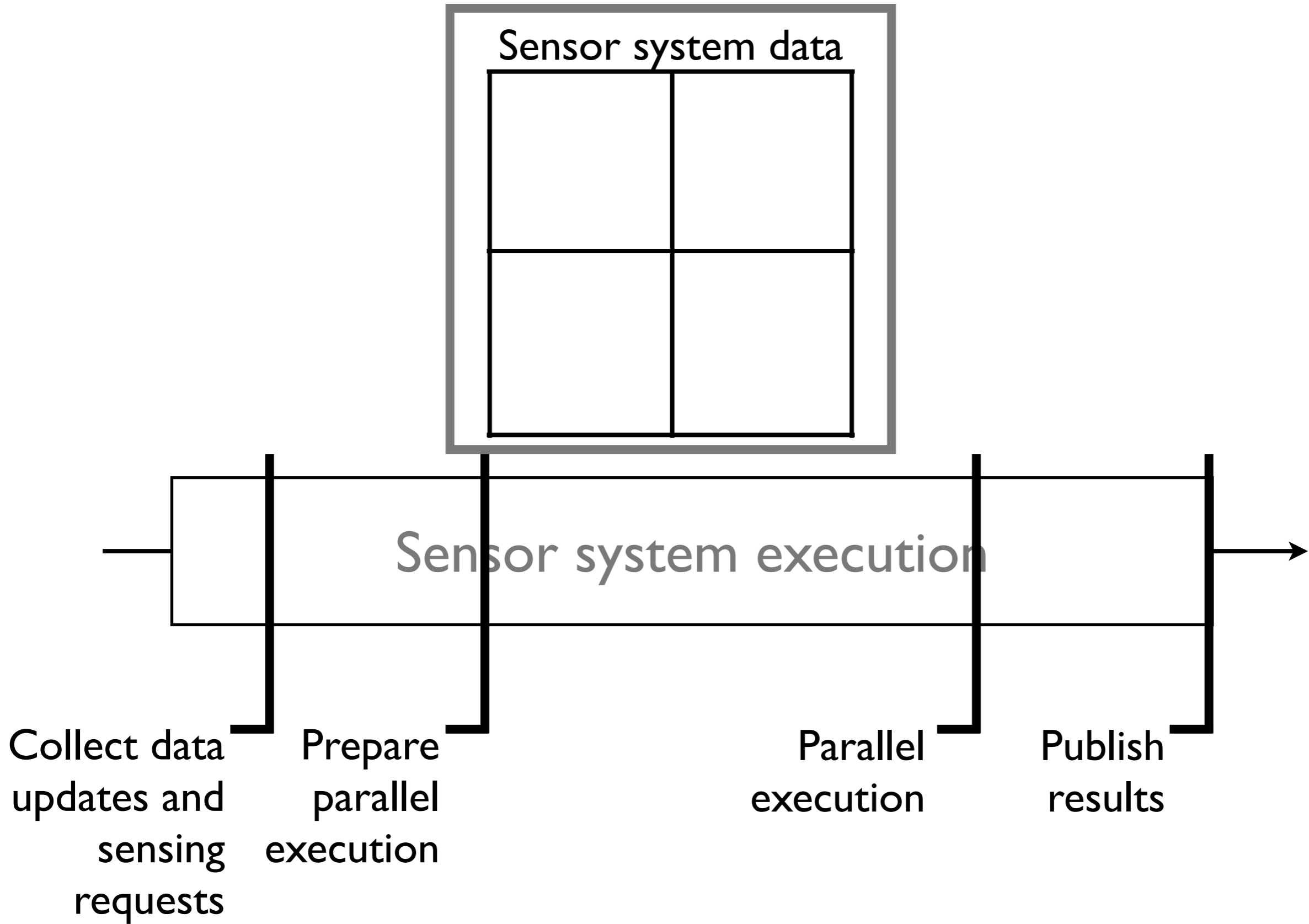
Sensor system execution runs through several stages

-> implicit syncing of stages (barriers)

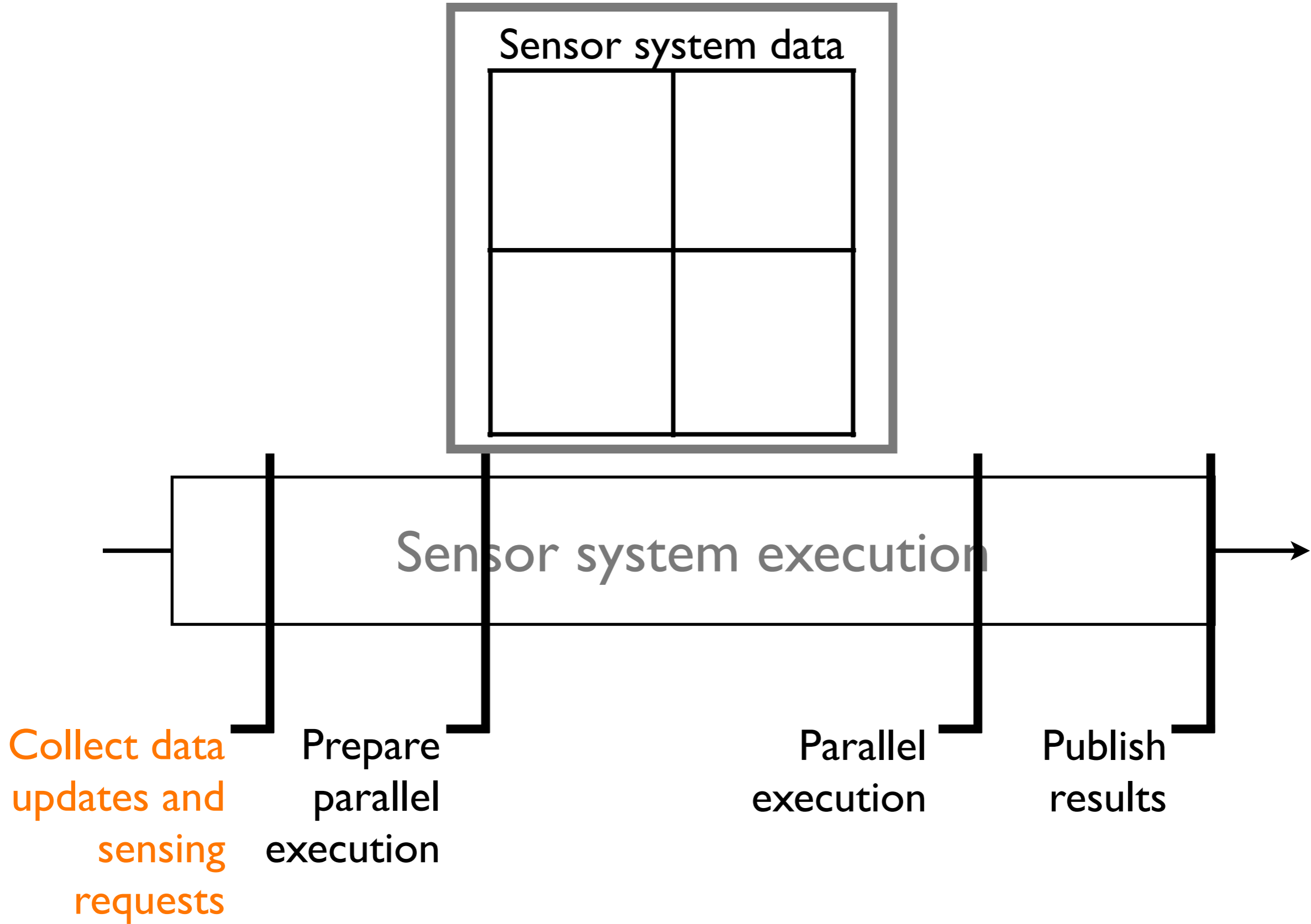
-> **first stage always active to collect input for next time step**



Example: 2d grid to store sense-able entities and sensors  
 Data decomposition -> data parallelism  
 Lean data, locality possible!



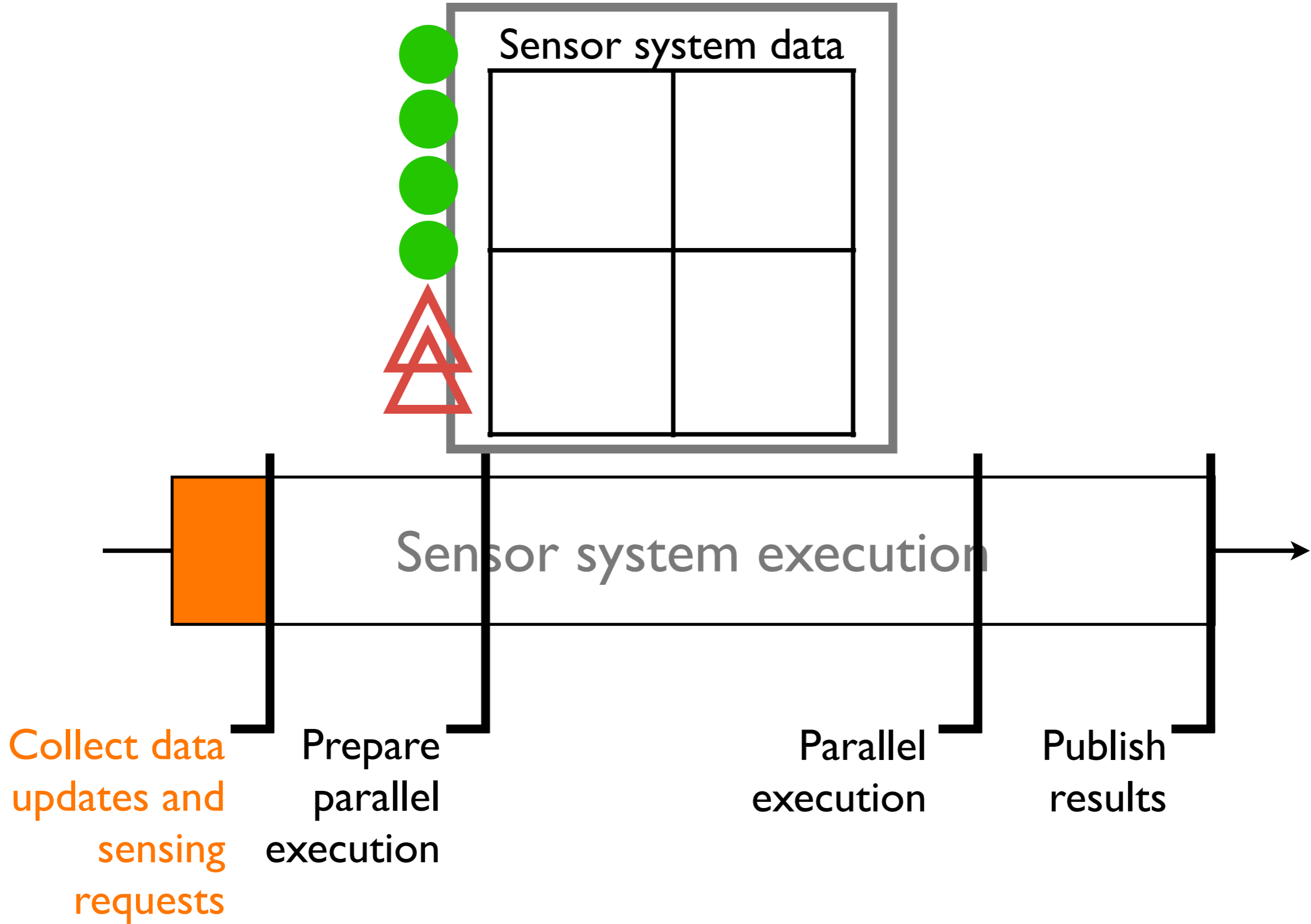
Example: 2d grid to store sense-able entities and sensors  
 Data decomposition -> data parallelism  
 Lean data, locality possible!



Collect (message queue?)

... data or update commands for data (points = entities)

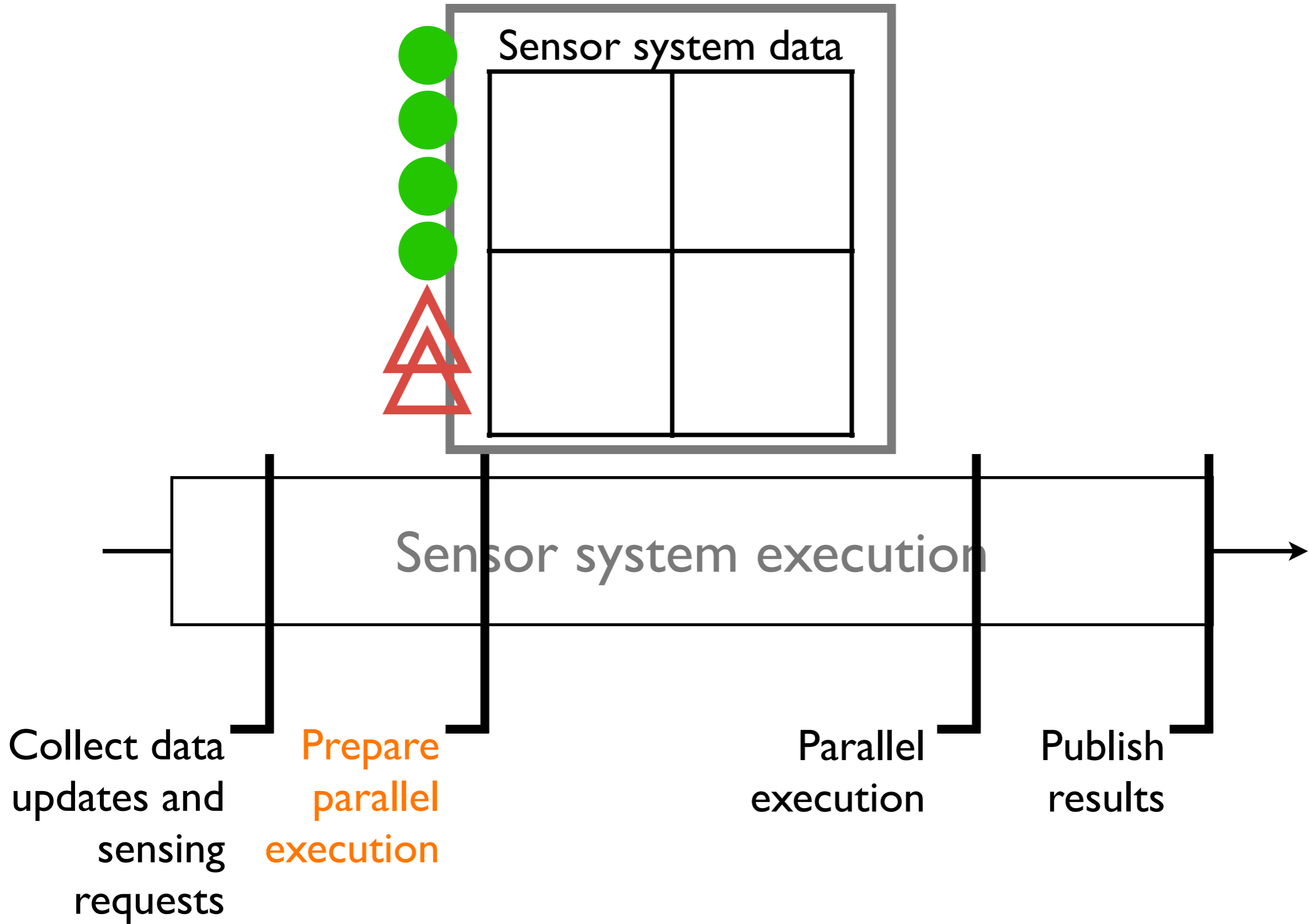
... and sensing requests (triangle = field of view)



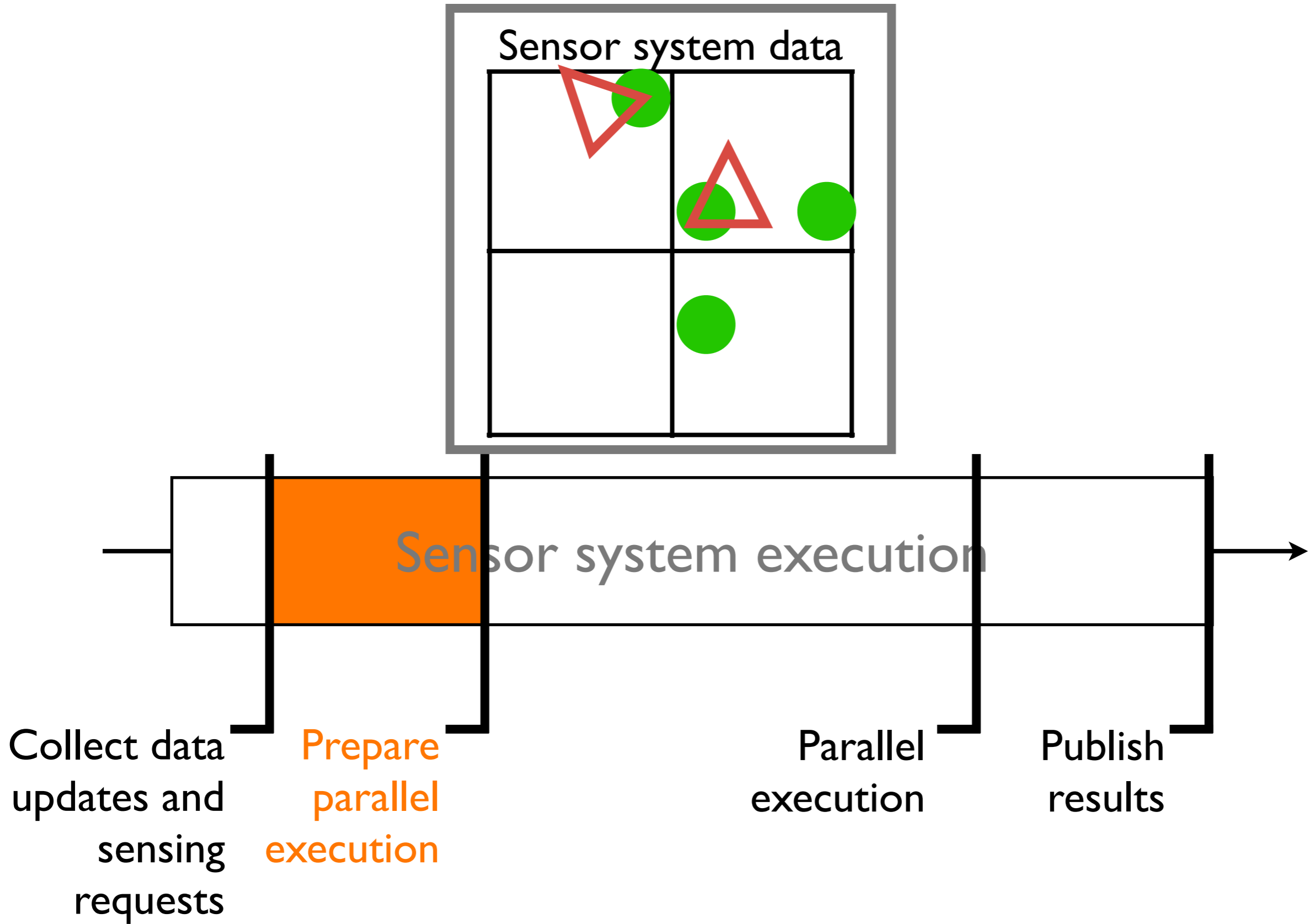
Collect (message queue?)

... data or update commands for data (points = entities)

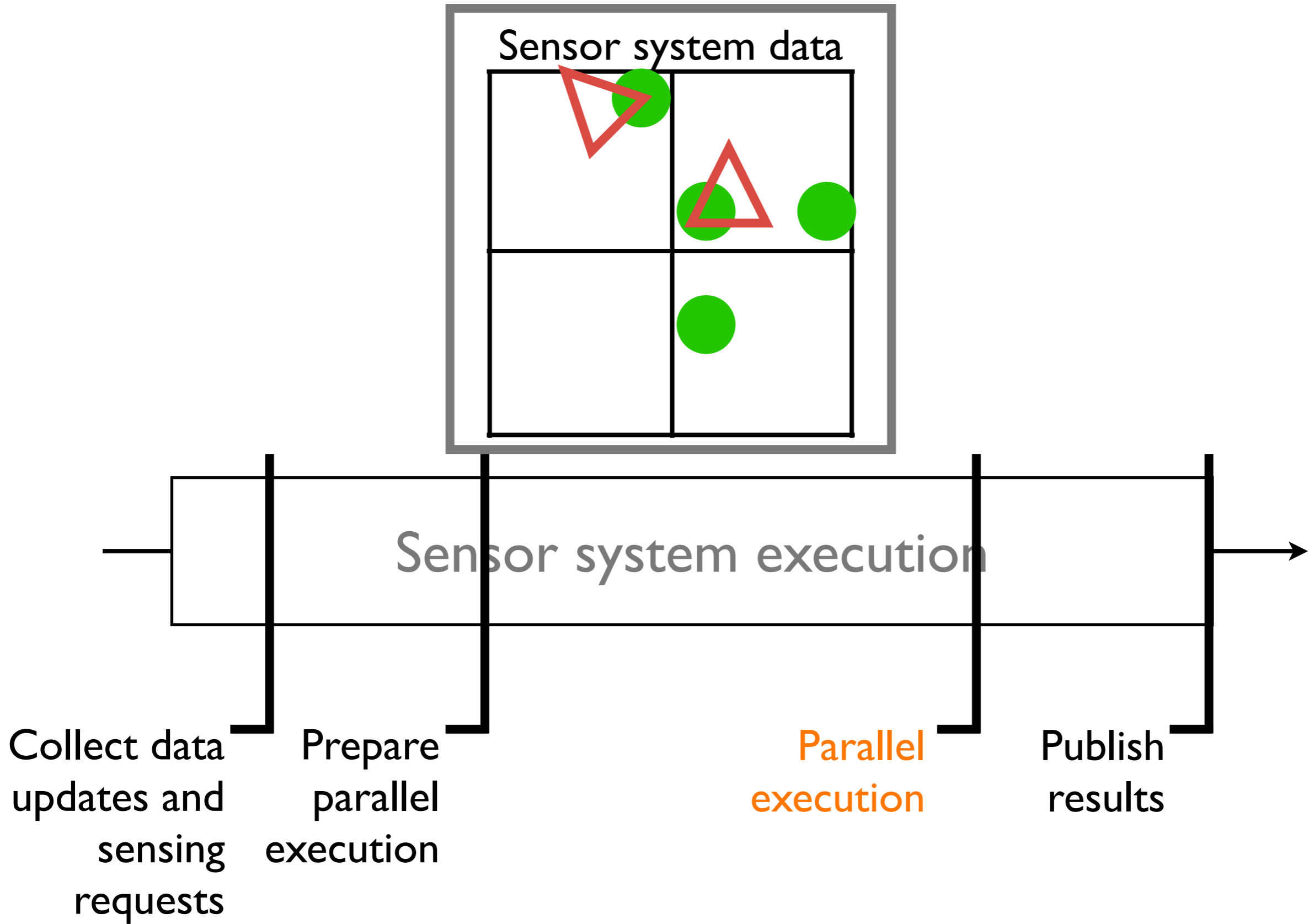
... and sensing requests (triangle = field of view)



Prepare data if necessary, possibly in parallel  
 Associate entities with grid cells or change their transformations  
 Add or enable/disable sensor view frustums, and associate them with grid cells  
 Sort if necessary...



Prepare data if necessary, possibly in parallel  
 Associate entities with grid cells or change their transformations  
 Add or enable/disable sensor view frustums, and associate them with grid cells  
 Sort if necessary...

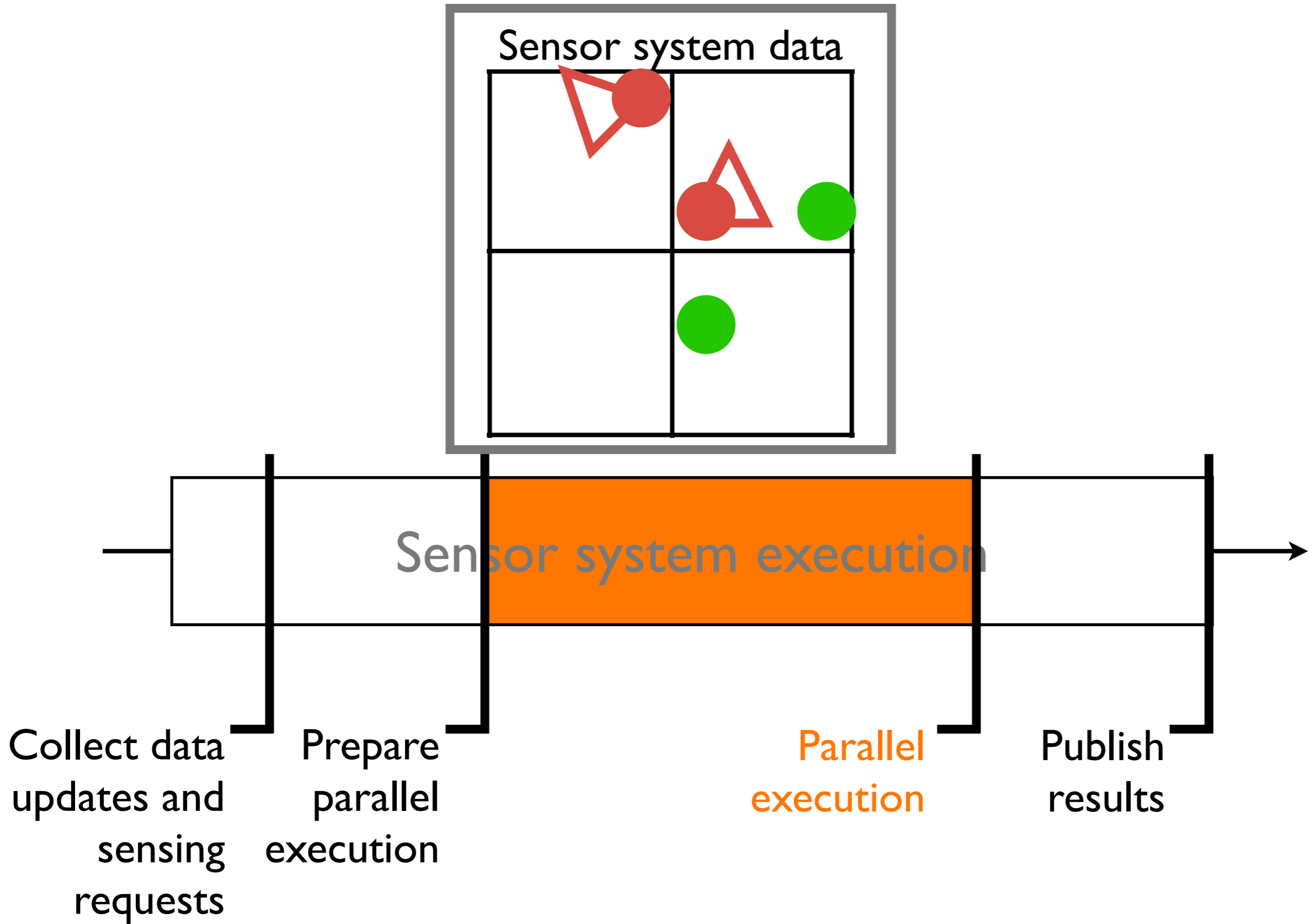


**Execute stages parallel:**

... to maximize locality run sensing per grid cell and in parallel

-> control: iterations per timestep and

... lockstep possible -> determinism if order independent parallel computation

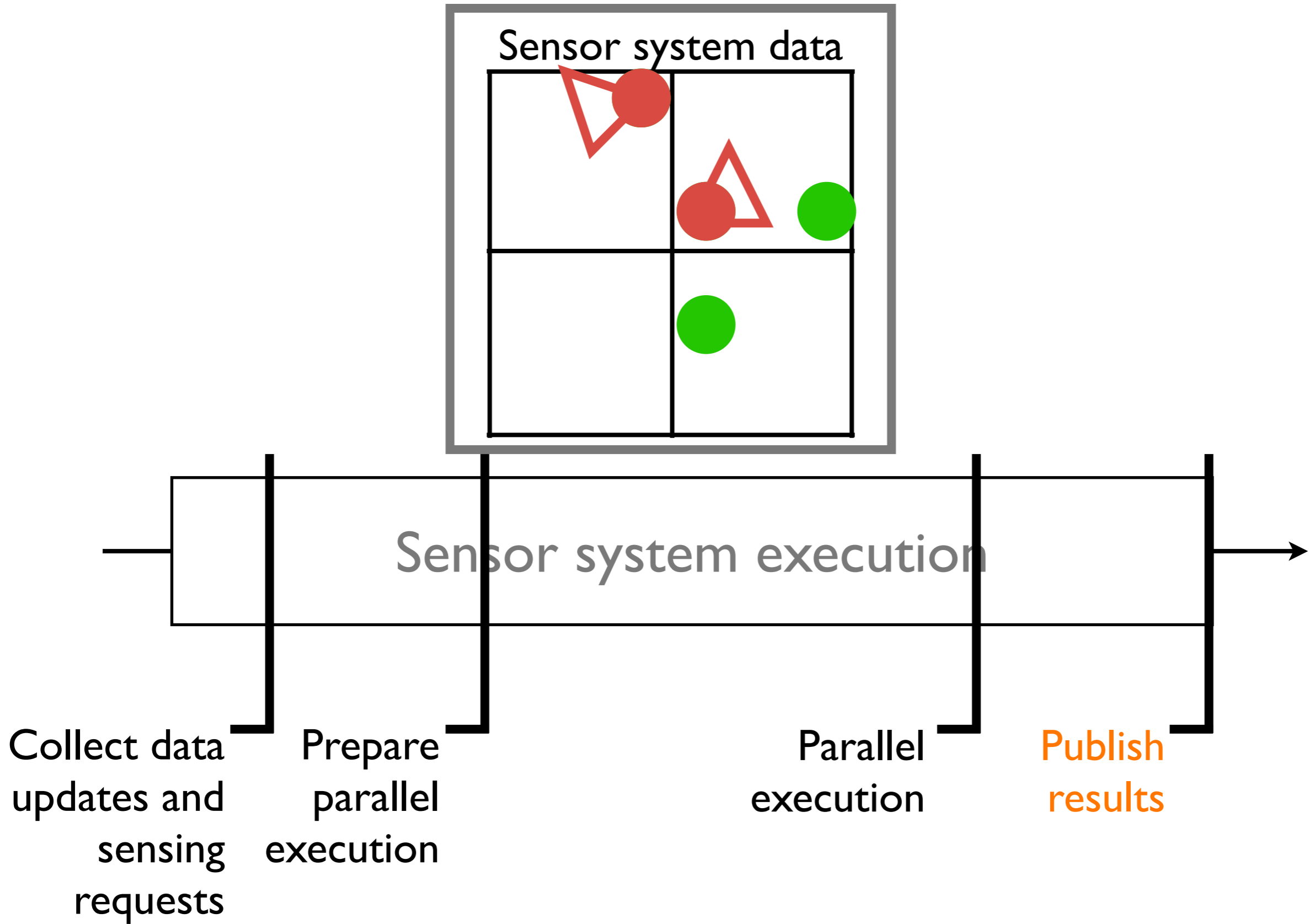


**Execute stages parallel:**

... to maximize locality run sensing per grid cell and in parallel

-> control: iterations per timestep and

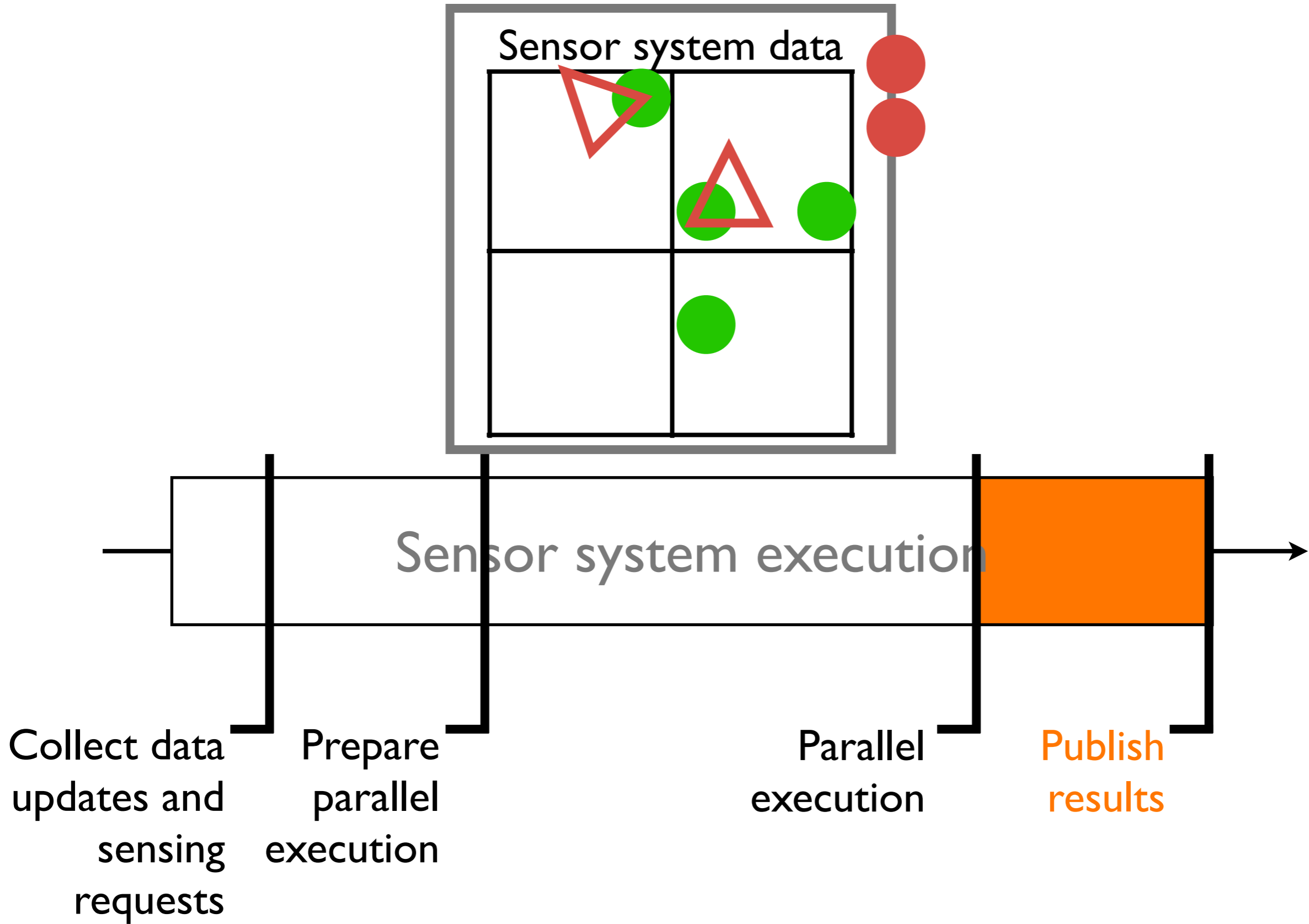
... lockstep possible -> determinism if order independent parallel computation



Publish the computational results, via observer (Intel Smoke Framework), or MVC pattern (planned for AiGameDev.com Sandbox)

... decide: buffer results till pulled or send them out

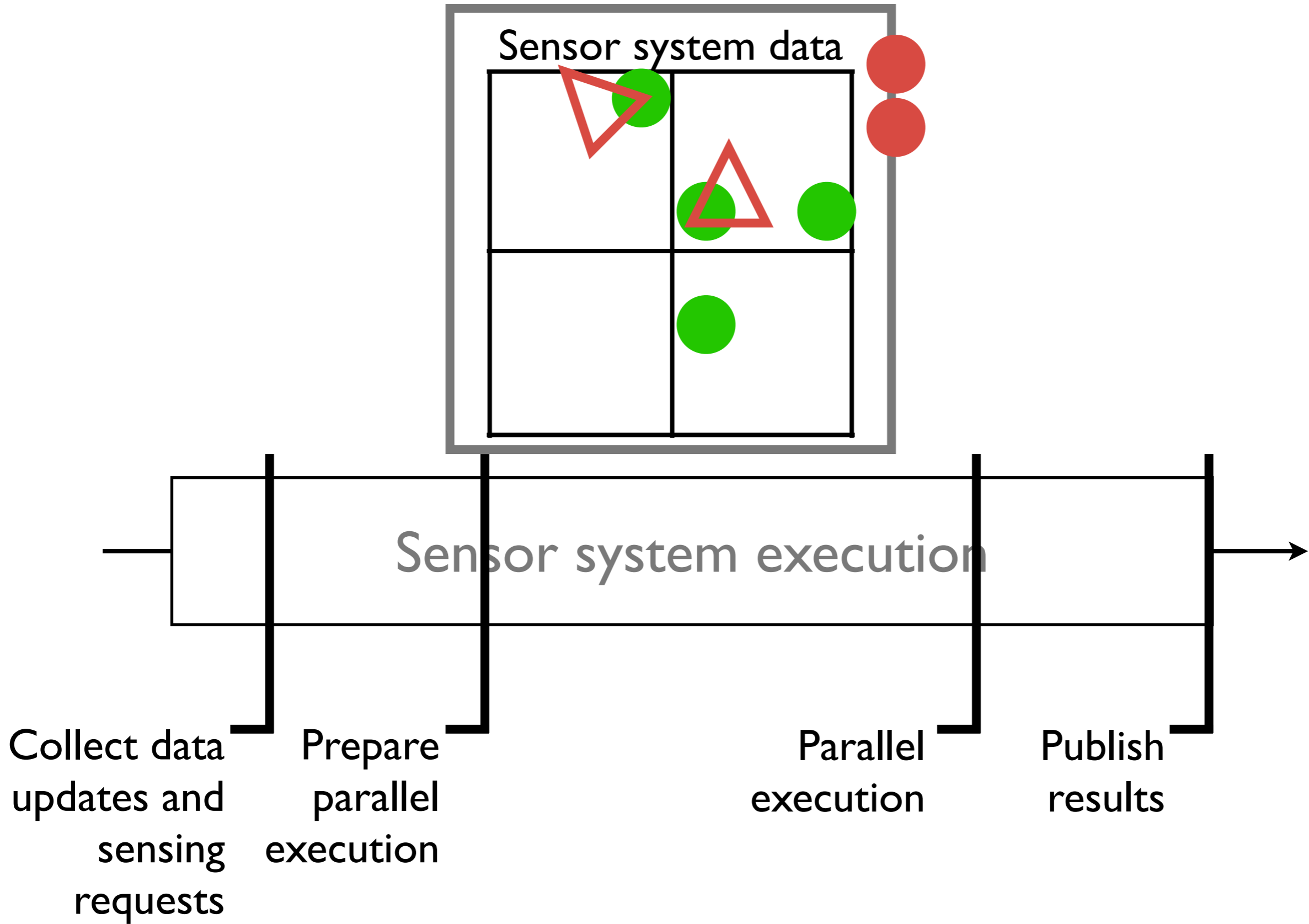
... decide: events, futures -> think determinism, look at async service call approach



Publish the computational results, via observer (Intel Smoke Framework), or MVC pattern (planned for AiGameDev.com Sandbox)

... decide: buffer results till pulled or send them out

... decide: events, futures -> think determinism, look at async service call approach



Exploit internal data access pattern by allow injection of Insomniac "SPU" shaders to generate other data that might be useful.

Full control of context for system and "SPU" shader -> no side effects

-> easy to exploit parallelism offered by system developer

# Data Parallel Systems

Pros	Cons

Data exchange between systems -> walk through system outputs and the inputs needed from last system to first and design the systems and their data accordingly (Insomniac's advice).

Porting system for system possible

# Data Parallel Systems

Pros	Cons
Lean Data Structures	
Locality	
Scaling	
Deterministic	
Implicit Synchronization	
Explicit Context	
Possibilities: “Shaders”	

# Data Parallel Systems

Pros	Cons
Lean Data Structures	Implementation effort
Locality	Intern. low-level parallel
Scaling	Get syncing right
Deterministic	Deferring necessary
Implicit Synchronization	
Explicit Context	
Possibilities: “Shaders”	

# Stream Processing

Input flows through kernels into output buffers

Solution space constraint

Automatic parallelization

Cross-platform capable

Look into:

Emergen't Floodgate tech

OpenCL

# Conclusion

Mittwoch, 17. Juni 2009

89

- minimize possibility for errors first
- then maximize performance
- in your domain: no low-level parallel programming, create frameworks that structure parallelism, allow you to express while framework manages parallelism and uses implicit syncing
- think tasks and workloads and avoid side effects
- use double buffering and message passing to decouple calculations and allow deferring of computations
- use ownership management and instanced random number generators and lockstep mode in systems to enable determinism - be order of computation independent
- differentiate reading and writing, input and output to make dependencies explicit and to identify aspects of your computation and collect aspects in staged systems
- optimize for throughput and balance max. locality and min. sharing to exploit the hardware
- no rocket science, just software engineering
- use parallelism to push your AI further

# Understand Parallel Hardware

Prevent errors – than care for performance

Know the hardware: optimize throughput, balance max locality with min sharing

# Error-freeness first

Prevent errors – than care for performance

Know the hardware: optimize throughput, balance max locality with min sharing

**Error-freeness first**

**Performance afterwards**

# Parallelization Approaches as Guides



**KISS**

**Keep It Short and Simple**

... thats it



Email: [bknafla@member.igda.org](mailto:bknafla@member.igda.org)

Email: [bknafla@member.igda.org](mailto:bknafla@member.igda.org)

Twitter: [@bjoernknafla](https://twitter.com/bjoernknafla)

**Thank You!**

# Three Great References

- Joe Valenzuela (Insomniac) “SPU gameplay” presentation from GDC '09 - <http://www.insomniacgames.com/tech/techpage.php>
- OpenCL or Nvidia’s CUDA documentation
- James Reinders “Intel Threading Building Blocks”, O’Reilly, 2007