Horizon

Runtime Efficient Event Scheduling in Multi-threaded Network Simulation

Georg Kunz, Mirko Stoffers, James Gross, Klaus Wehrle

http://www.comsys.rwth-aachen.de/

OMNeT++ Workshop, SimuTools, March 2011

Motivation

- **Need for Complex Network Simulation Models**
  - Detailed channel and PHY characteristics
  - Large scale P2P and Internet backbone models
  - ⇨ High processing and runtime demand

- **Proliferation of Multi-processor Systems**
  - Desktop: 4-8 cores, servers: 24 cores
  - “Desktop Cluster”
  - ⇨ Cheap, powerful commodity hardware

Motivation

- **Need for Complex Network Simulation Models**
  - Detailed channel and PHY characteristics
  - Large scale P2P and Internet backbone models
  - ⇨ High processing and runtime demand

- **Proliferation of Multi-processor Systems**
  - Desktop: 4-8 cores, servers: 24 cores
  - “Desktop Cluster”
  - ⇨ Cheap, powerful commodity hardware

⇨ Utilize Parallelization to Cut Runtimes?

Motivation: Downside of Parallelization

- **Parallelization Introduces Overhead**
  - Thread synchronization, management of shared data
  - Increased management overhead per event
  - Negative impact on events of low complexity

Motivation: Downside of Parallelization

- **Parallelization Introduces Overhead**
  - Thread synchronization, management of shared data
  - Increased management overhead per event
  - Negative impact on events of low complexity

- **Dilemma / Tradeoff**

Motivation: Downside of Parallelization

- **Parallelization Introduces Overhead**
  - Thread synchronization, management of shared data
  - Increased management overhead per event
  - Negative impact on events of low complexity

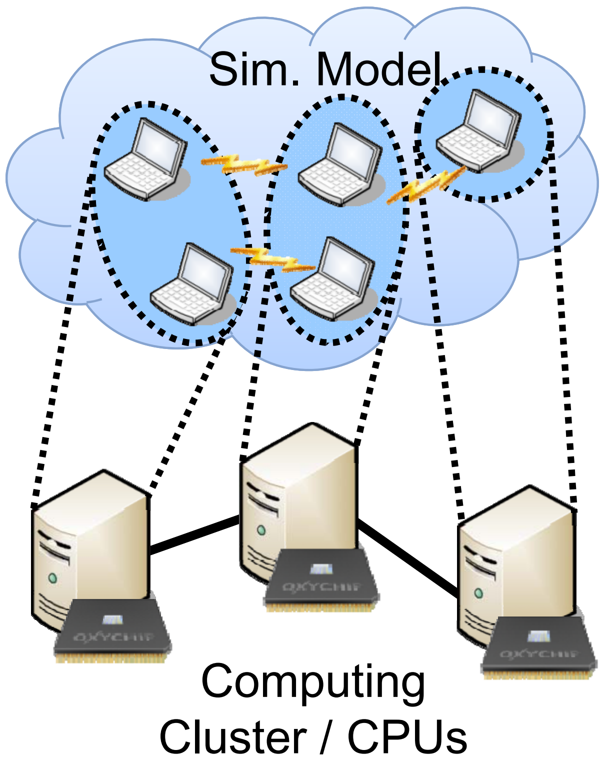
- **Dilemma / Tradeoff**

⇨ Minimize Parallelization Overhead

Horizon: Approach

- **Horizon**
  - Focus on multi-processor systems
  - Centralized architecture
  - Conservative synchronization
    - Determine independent events

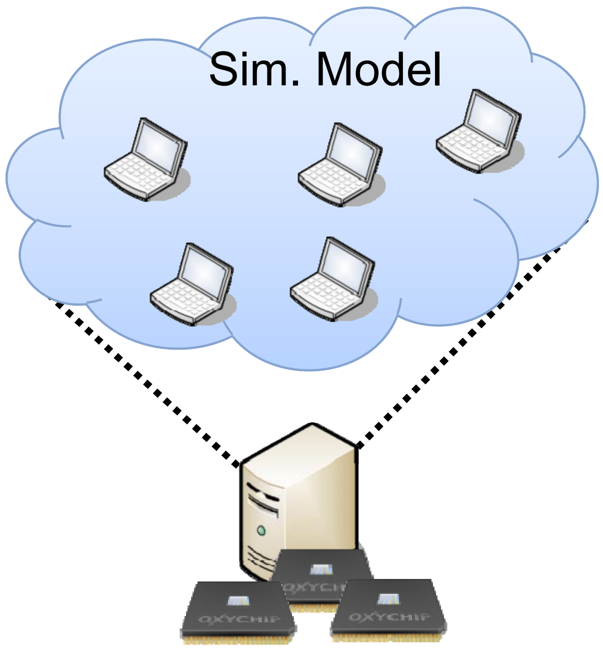
- **Expanded Events**
  - Modeling paradigm
  - Per event lookahead
  - ⇨ Identify independent events



Horizon: Approach

- **Horizon**
  - Focus on multi-processor systems
  - Centralized architecture
  - Conservative synchronization
    - Determine independent events

- **Expanded Events**
  - Modeling paradigm
  - Per event lookahead
  - ⇨ Identify independent events

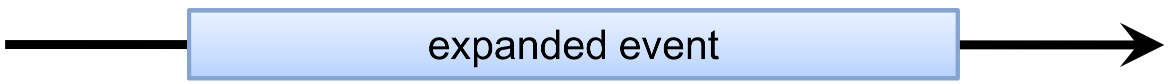


Horizon: Expanded Events

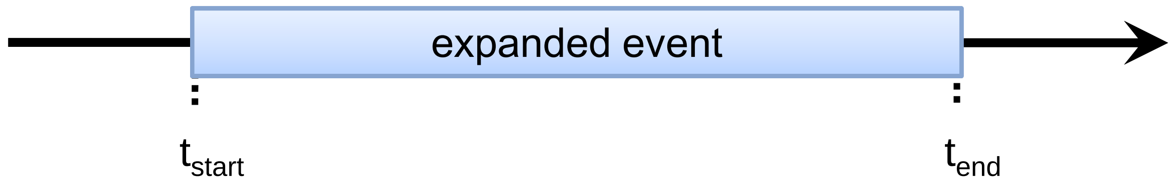
- **Expanded Events**
  - Model processes that span period of time
  - Augment discrete events with durations
  - ⇨ Discrete events span period of **simulated** time

Horizon: Expanded Events

- **Expanded Events**
  - Model processes that span period of time
  - Augment discrete events with durations
  - ⇨ Discrete events span period of **simulated** time

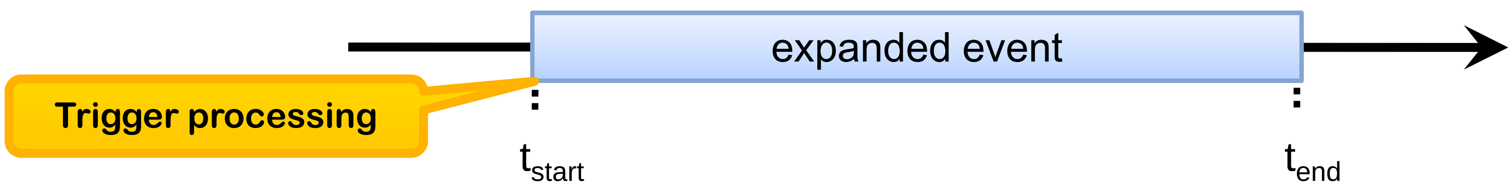


- **Expanded Events**
  - Model processes that span period of time
  - Augment discrete events with durations
  - ⇨ Discrete events span period of **simulated** time



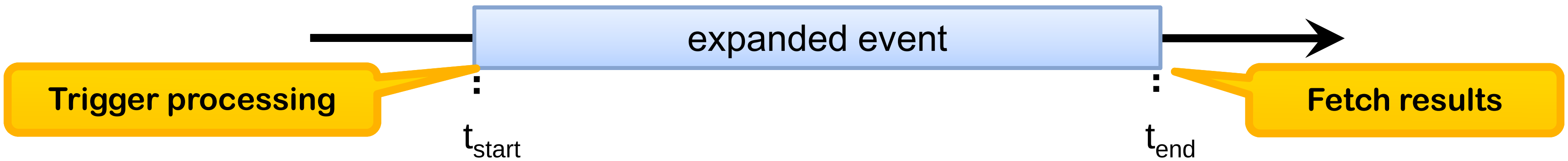
Horizon: Expanded Events

- **Expanded Events**
  - Model processes that span period of time
  - Augment discrete events with durations
  - ⇨ Discrete events span period of **simulated** time



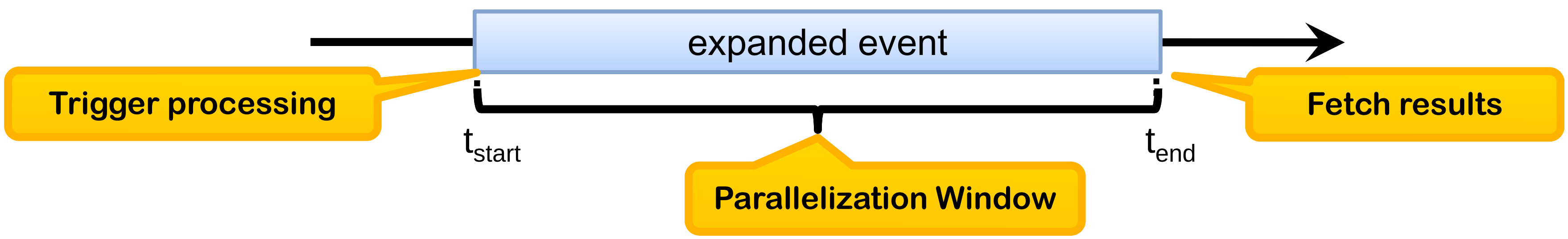
Horizon: Expanded Events

- **Expanded Events**
  - Model processes that span period of time
  - Augment discrete events with durations
  - ⇨ Discrete events span period of **simulated** time



Horizon: Expanded Events

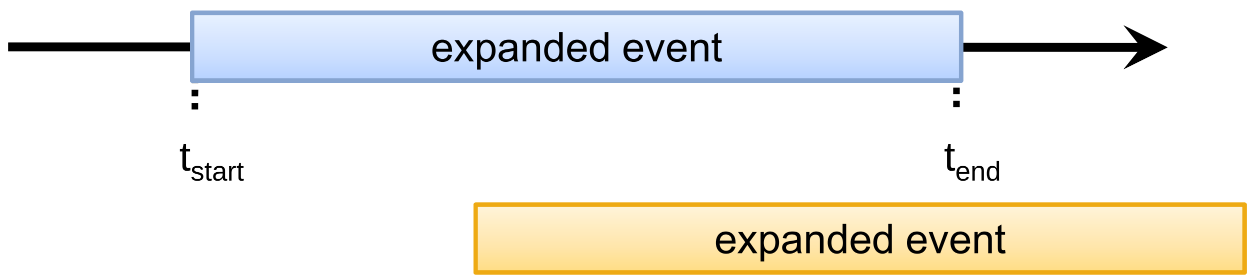
- **Expanded Events**
  - Model processes that span period of time
  - Augment discrete events with durations
  - ⇨ Discrete events span period of **simulated** time



- **Independent Events**
  - Events starting between t_{start} and t_{end}
  - Do not depend on results generated by overlapping event
  - Modeling paradigm

Horizon: Expanded Events

- **Expanded Events**
  - Model processes that span period of time
  - Augment discrete events with durations
  - ⇨ Discrete events span period of **simulated** time



- **Independent Events**
  - Events starting between t_{start} and t_{end}
  - Do not depend on results generated by overlapping event
  - Modeling paradigm

Challenges

How to reduce parallelization overhead?

Challenges and Solutions

- **We Address Two Challenges**

Thread Synchronization Overhead

Event Scheduling Overhead

Challenges and Solutions

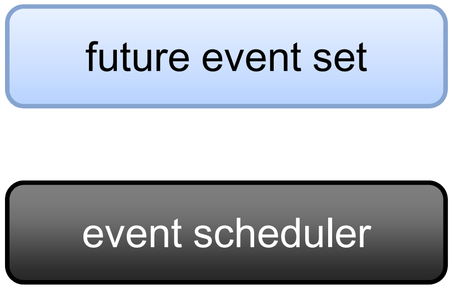
- **We Address Two Challenges**

Thread Synchronization Overhead

Event Scheduling Overhead

Thread Synchronization Overhead: Challenge

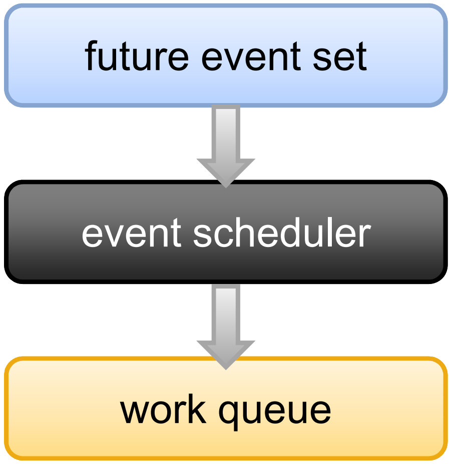
- **Master/Worker Architecture**
  - Master coordinates simulation progress
  - Workers do actual processing
  - Synchronization involves
    - Workers wait for incoming jobs
    - Access to shared data structures



- **Straightforward Implementation**
  - Locks, condition variables
  - Workers **pull** jobs from work queue
  - If lock occupied or no job available
    - Suspend thread
    - Free-up CPU resources

Thread Synchronization Overhead: Challenge

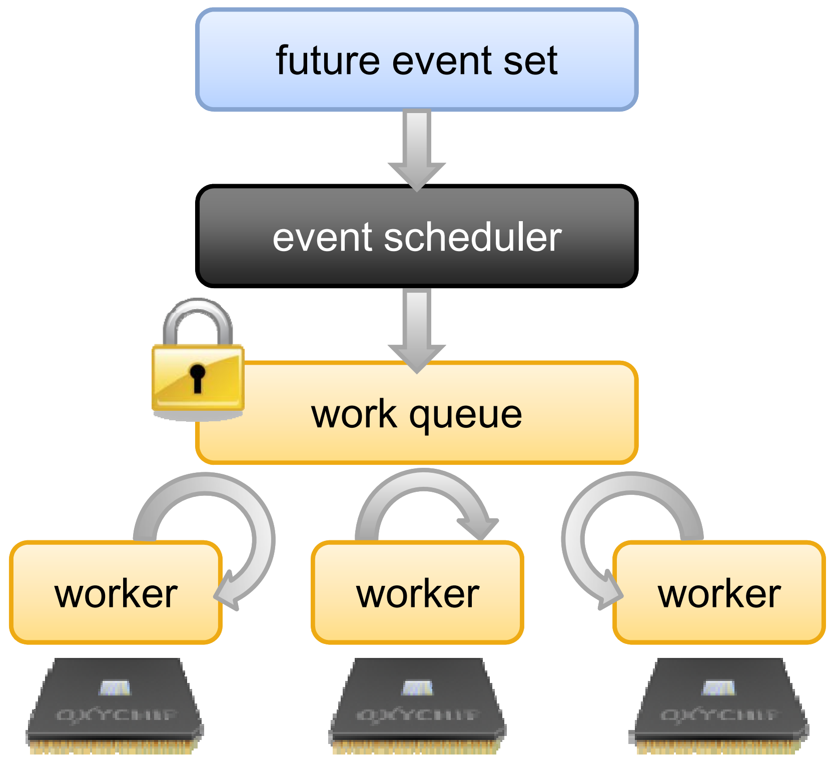
- **Master/Worker Architecture**
  - Master coordinates simulation progress
  - Workers do actual processing
  - Synchronization involves
    - Workers wait for incoming jobs
    - Access to shared data structures



- **Straightforward Implementation**
  - Locks, condition variables
  - Workers **pull** jobs from work queue
  - If lock occupied or no job available
    - Suspend thread
    - Free-up CPU resources

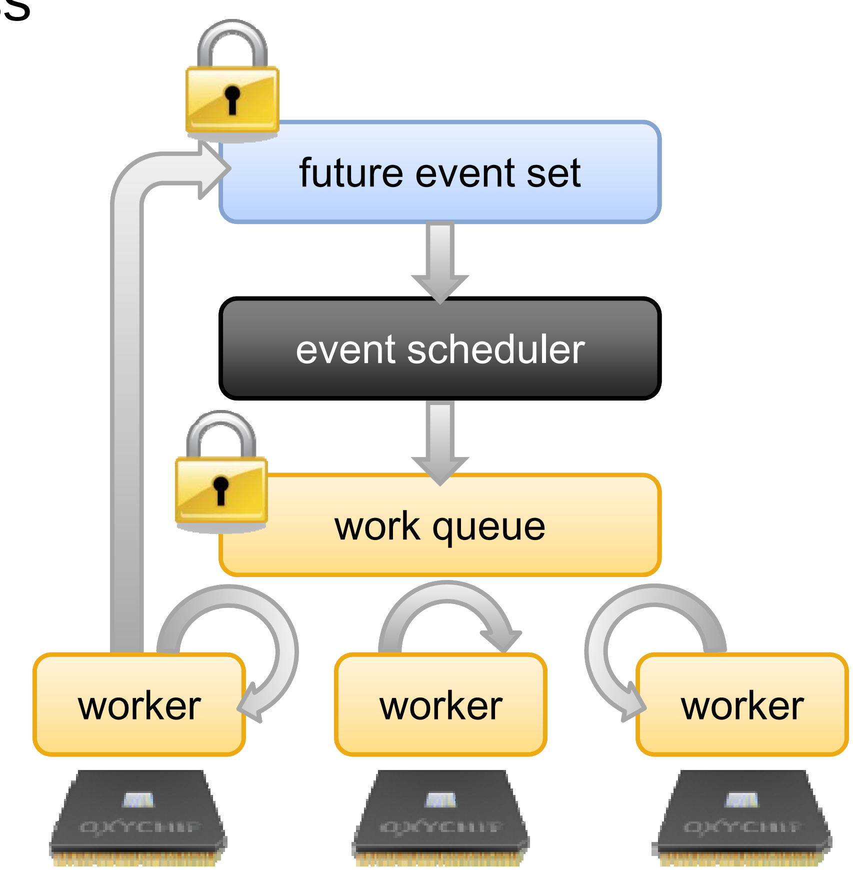
Thread Synchronization Overhead: Challenge

- **Master/Worker Architecture**
  - Master coordinates simulation progress
  - Workers do actual processing
  - Synchronization involves
    - Workers wait for incoming jobs
    - Access to shared data structures
- **Straightforward Implementation**
  - Locks, condition variables
  - Workers **pull** jobs from work queue
  - If lock occupied or no job available
    - Suspend thread
    - Free-up CPU resources



Thread Synchronization Overhead: Challenge

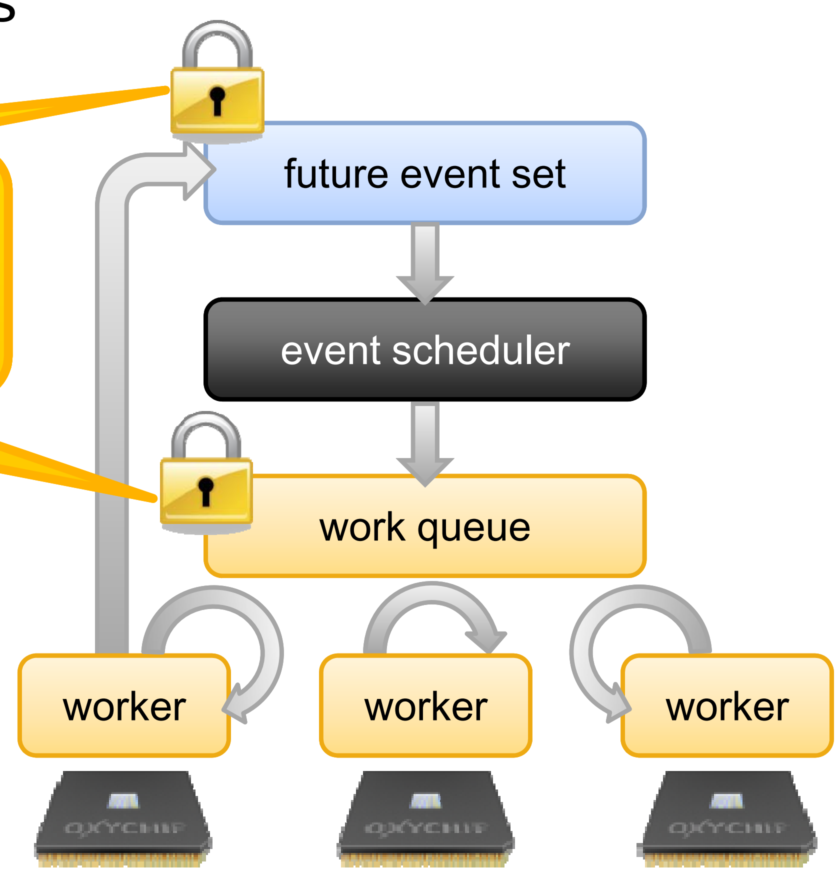
- **Master/Worker Architecture**
  - Master coordinates simulation progress
  - Workers do actual processing
  - Synchronization involves
    - Workers wait for incoming jobs
    - Access to shared data structures
- **Straightforward Implementation**
  - Locks, condition variables
  - Workers **pull** jobs from work queue
  - If lock occupied or no job available
    - Suspend thread
    - Free-up CPU resources



Thread Synchronization Overhead: Challenge

- **Master/Worker Architecture**
  - Master coordinates simulation progress
  - Workers do actual processing
  - Sync
    - V
    - A

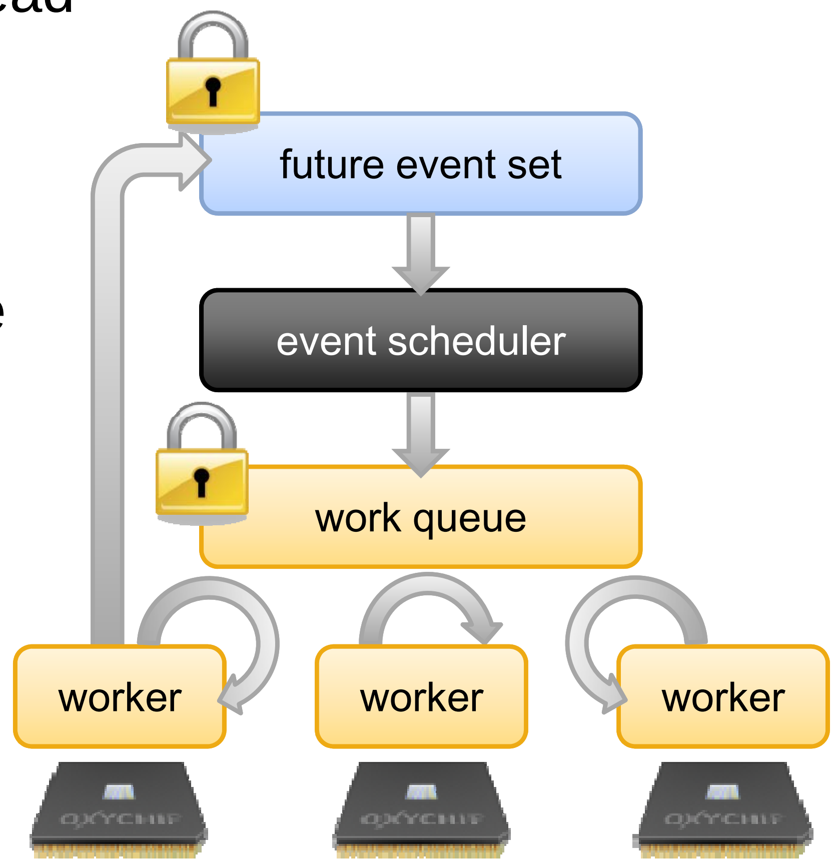
Increases Threading Overhead
- **Sys-calls into kernel**
- **Context switches**



- **Straightforward Implementation**
  - Locks, condition variables
  - Workers **pull** jobs from work queue
  - If lock occupied or no job available
    - Suspend thread
    - Free-up CPU resources

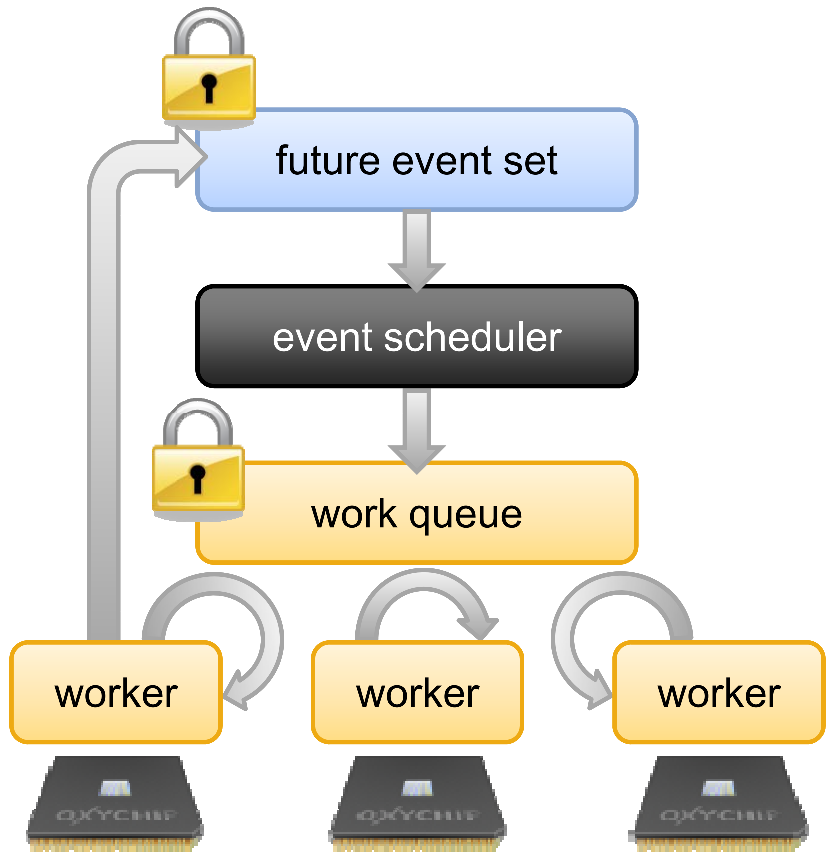
Thread Synchronization Overhead: Approach

- **Challenge**
  - Suspending Threads Increases Overhead
- **Observation**
  - Simulations run on dedicated hardware
  - Freeing-up CPUs is needless
  - Crucial to minimize offloading delay
- **Approach**
  - Use busy waiting for synchronization
  - Master actively **pushes** jobs to workers



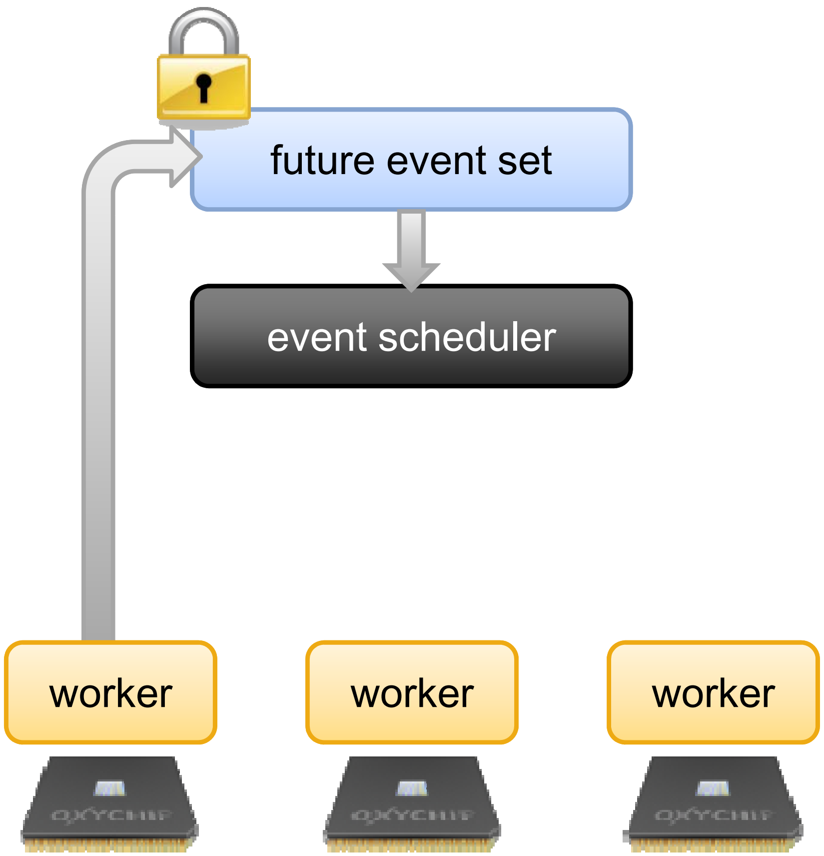
Thread Synchronization Overhead: Solution

- **Push-based Event Offloading**
  - ▶ Eliminate shared work queue
  - ▶ Introduce local synch. buffer per thread
  - ▶ Spinlock for future event set
- **Synchronization Buffer**
  - ▶ Master assigns jobs to empty buffer
  - ▶ Workers spin on empty buffer
- **Additional Benefit**
  - ▶ Master can identify busy threads
  - ▶ Master handles event instead of worker
  - ⇨ Make use of scheduler CPU



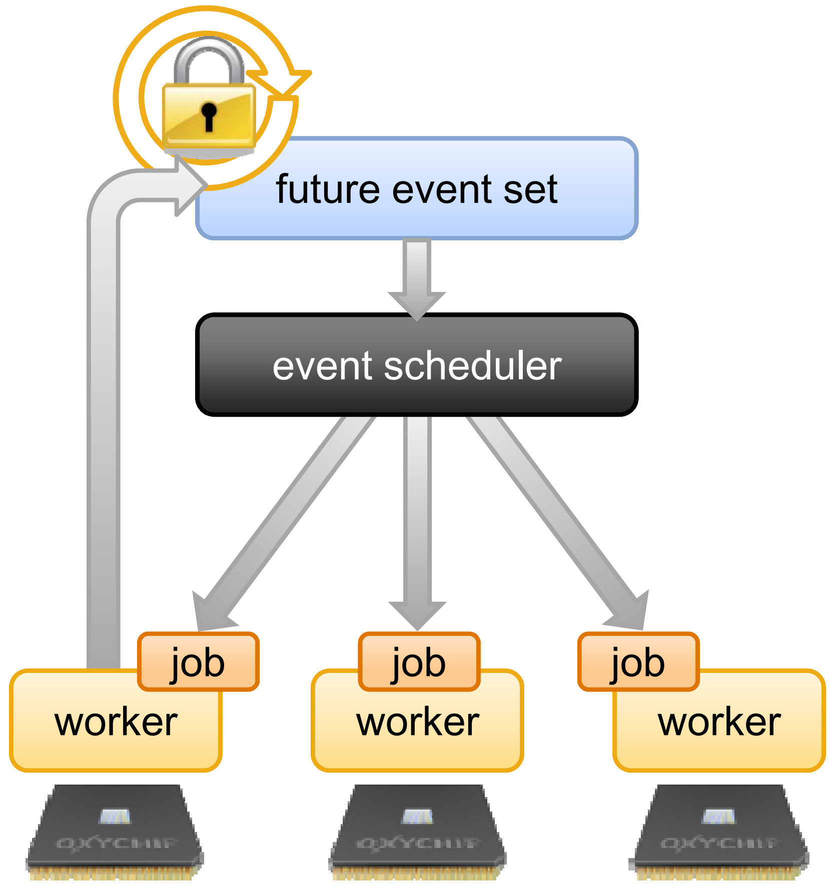
Thread Synchronization Overhead: Solution

- **Push-based Event Offloading**
  - Eliminate shared work queue
  - Introduce local synch. buffer per thread
  - Spinlock for future event set
- **Synchronization Buffer**
  - Master assigns jobs to empty buffer
  - Workers spin on empty buffer
- **Additional Benefit**
  - Master can identify busy threads
  - Master handles event instead of worker
  - ⇨ Make use of scheduler CPU



Thread Synchronization Overhead: Solution

- **Push-based Event Offloading**
  - Eliminate shared work queue
  - Introduce local synch. buffer per thread
  - Spinlock for future event set
- **Synchronization Buffer**
  - Master assigns jobs to empty buffer
  - Workers spin on empty buffer
- **Additional Benefit**
  - Master can identify busy threads
  - Master handles event instead of worker
  - ⇨ Make use of scheduler CPU



Challenges and Solutions

- **We Address Two Challenges**

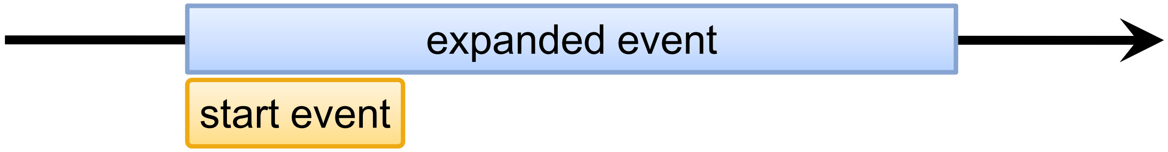
Thread synchronization Overhead

Event Scheduling Overhead

Event Scheduling Overhead: Challenge

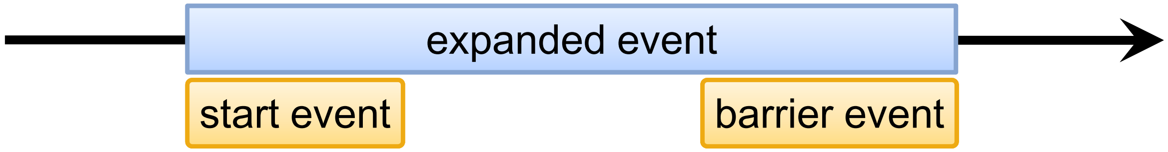
- **Integrate Expanded Events**
  - One discrete event marks start
  - Another discrete event marks end
  - Overlapping events: Start before barrier event
- **Straightforward Implementation**
  - Insert barrier event upon offloading
  - Wait at barrier event till execution finished

Event Scheduling Overhead: Challenge



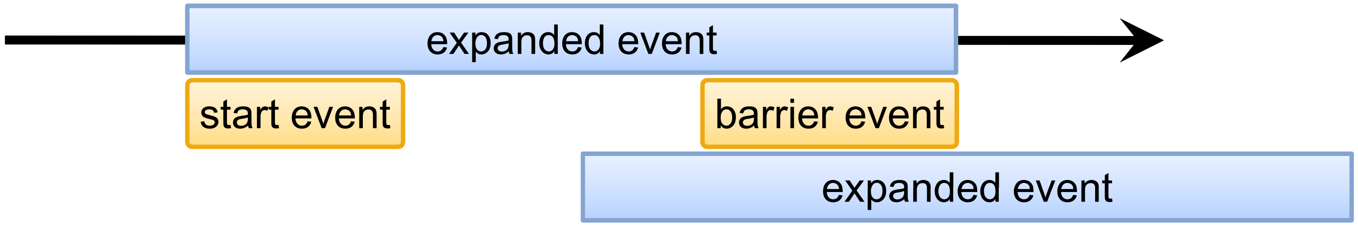
- **Integrate Expanded Events**
  - One discrete event marks start
  - Another discrete event marks end
  - Overlapping events: Start before barrier event
- **Straightforward Implementation**
  - Insert barrier event upon offloading
  - Wait at barrier event till execution finished

Event Scheduling Overhead: Challenge



- **Integrate Expanded Events**
  - One discrete event marks start
  - Another discrete event marks end
  - Overlapping events: Start before barrier event
- **Straightforward Implementation**
  - Insert barrier event upon offloading
  - Wait at barrier event till execution finished

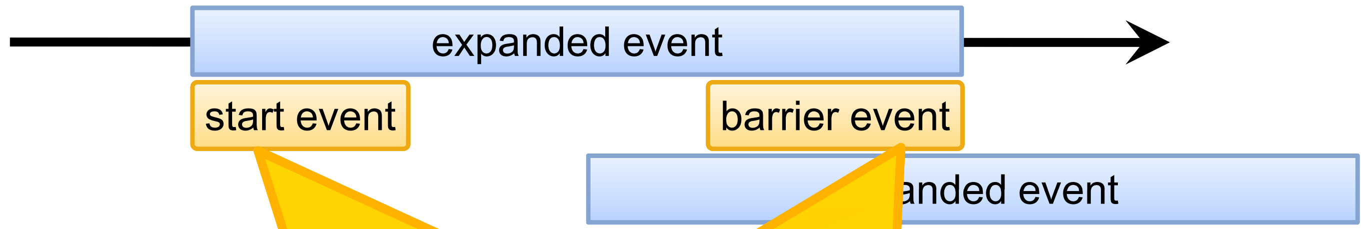
Event Scheduling Overhead: Challenge



- **Integrate Expanded Events**
  - One discrete event marks start
  - Another discrete event marks end
  - Overlapping events: Start before barrier event

- **Straightforward Implementation**
  - Insert barrier event upon offloading
  - Wait at barrier event till execution finished

Event Scheduling Overhead: Challenge



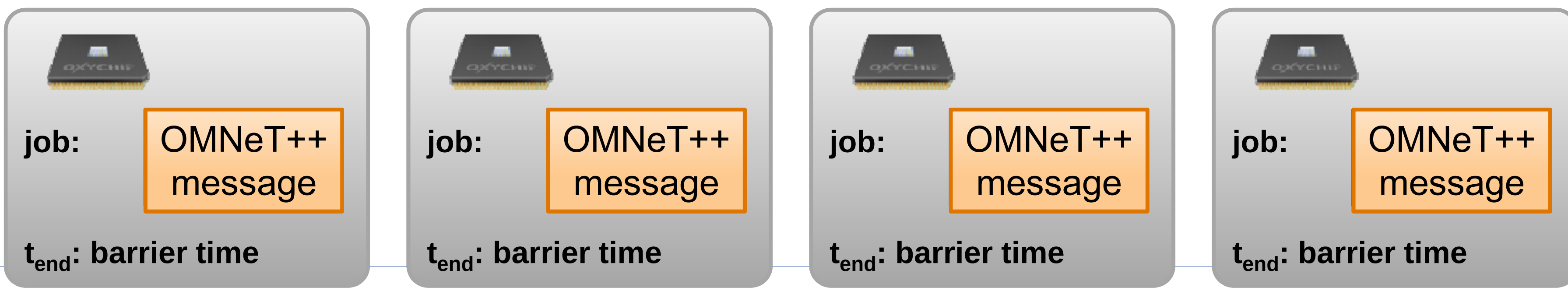
Doubles Overhead per Event
- **Creation, deletion of events**
- **Insertion, removal from FES**

- **Integrate Ex**
  - One discrete event marks start
  - Another discrete event marks end
  - Overlapping events: Start before barrier event

- **Straightforward Implementation**
  - Insert barrier event upon offloading
  - Wait at barrier event till execution finished

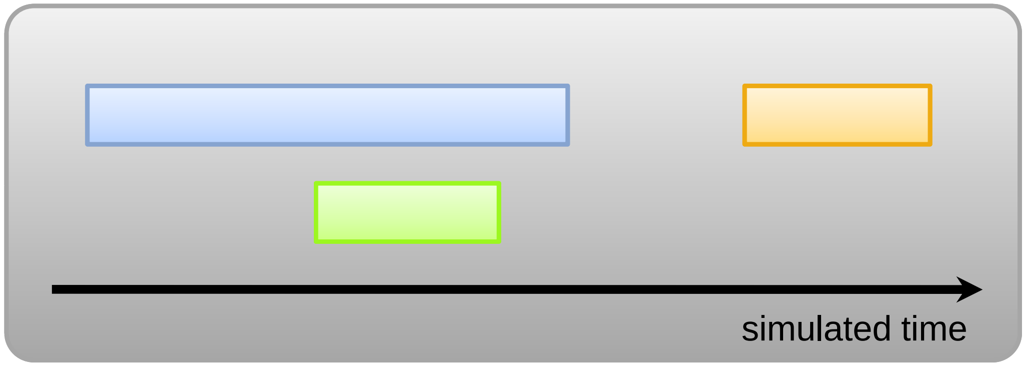
Event Scheduling Overhead: Approach

- **Observations**
  - Push-based synchronization
    - Upper bound for simultaneously offloaded events: #CPUs
    - ⇨ Upper bound for simultaneously existing barriers: #CPUs
- **Approach**
  - Avoid insertion into locked(!) message queue
  - Each thread maintains barrier time of current event
  - Pointer to earliest barrier enables fast lookup

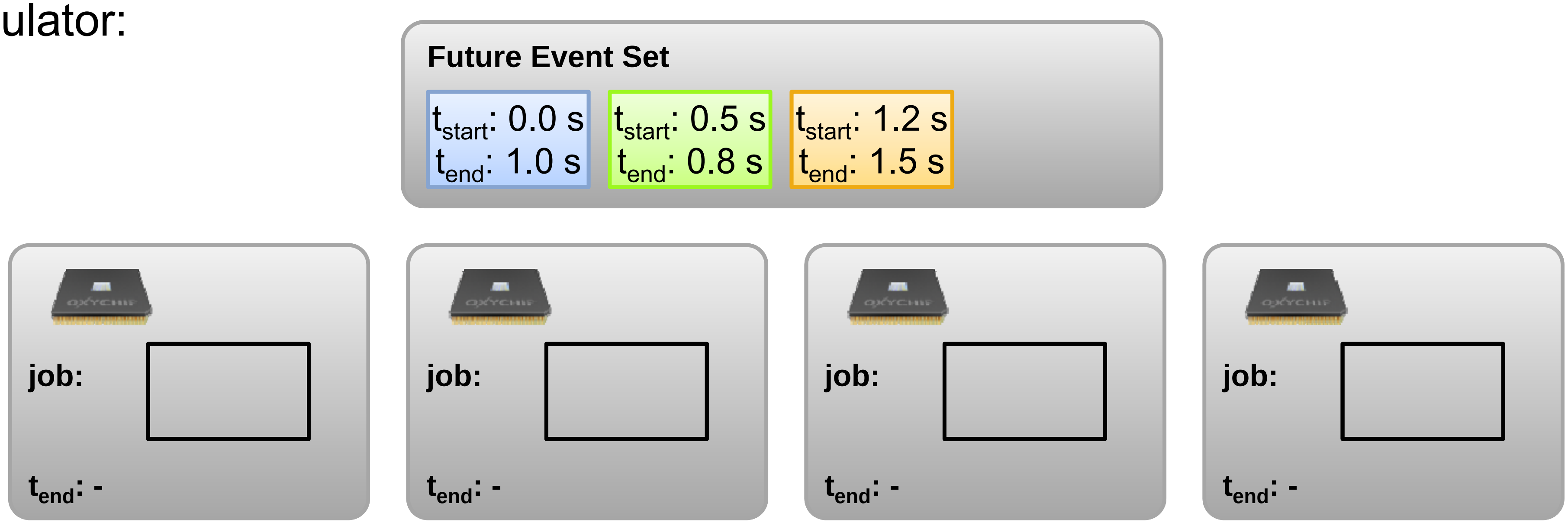


Event Scheduling Overhead: Solution

Example Schedule:

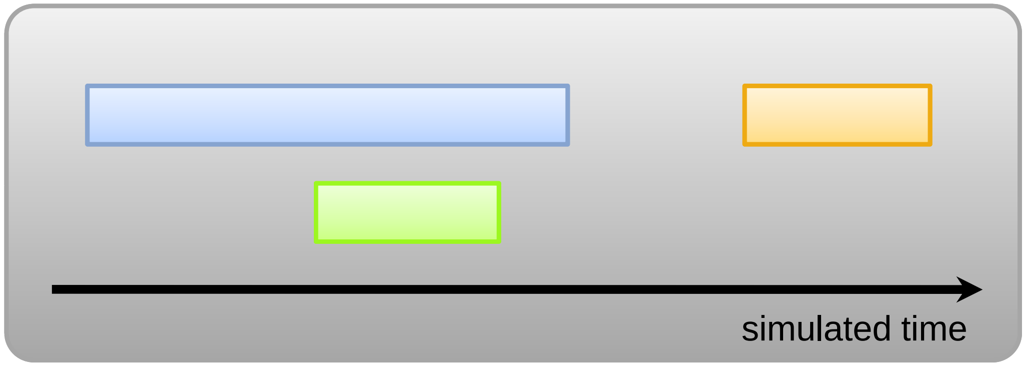


Simulator:

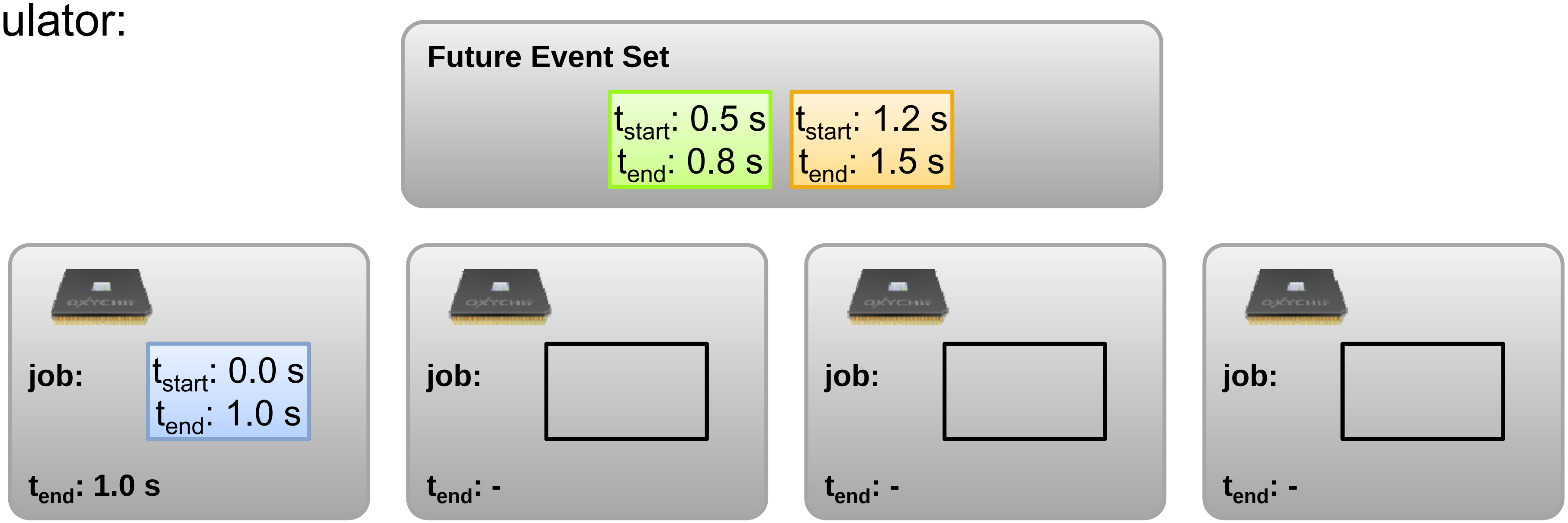


Event Scheduling Overhead: Solution

Example Schedule:

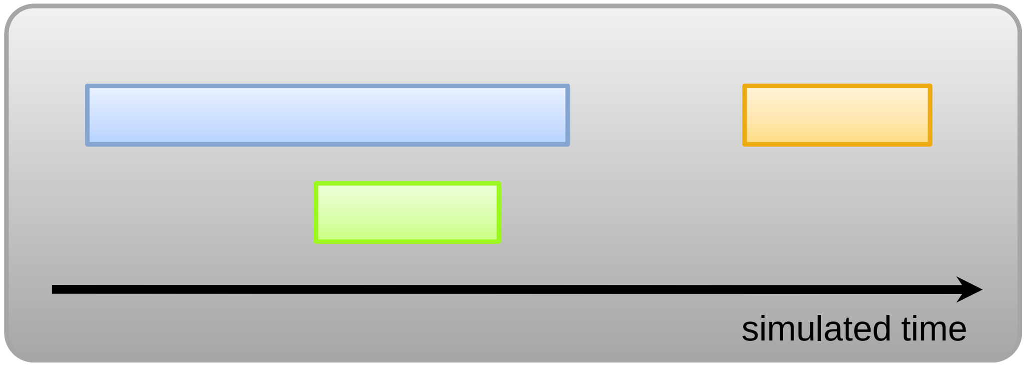


Simulator:

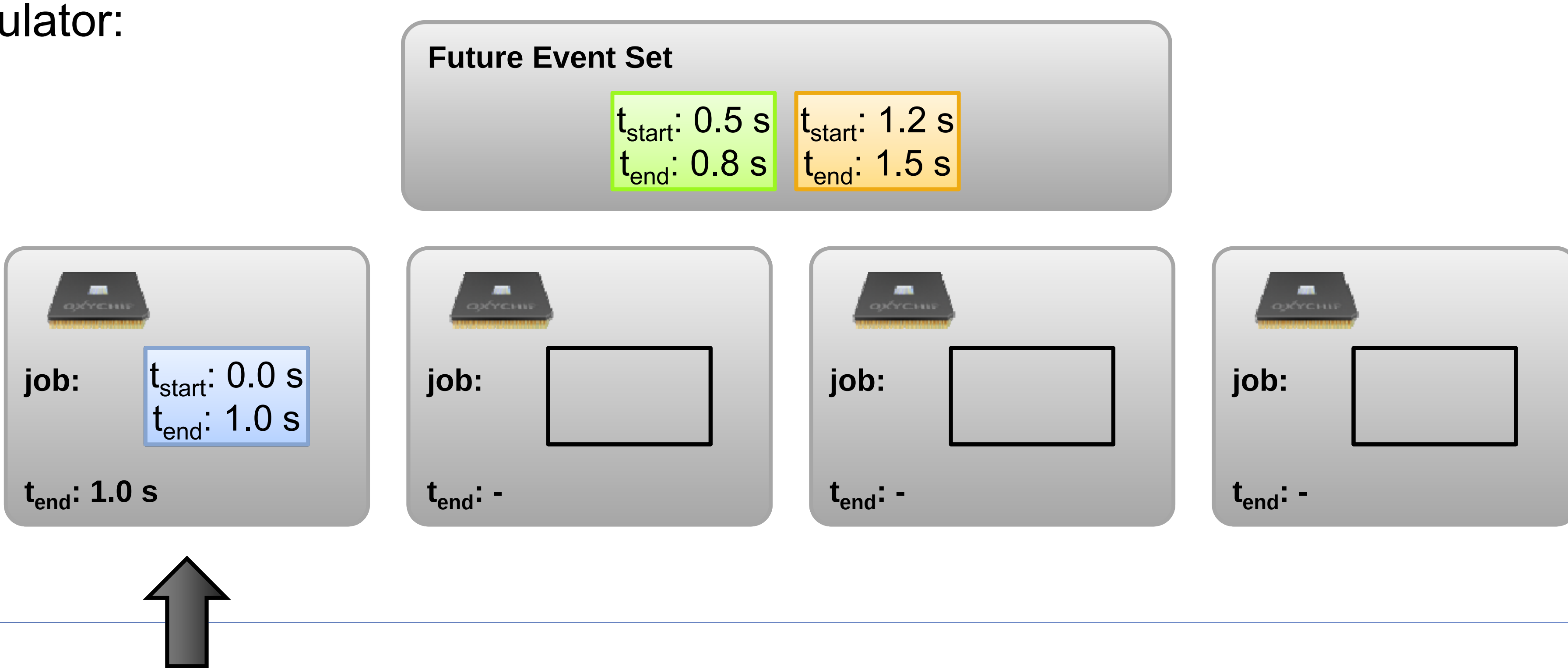


Event Scheduling Overhead: Solution

Example Schedule:

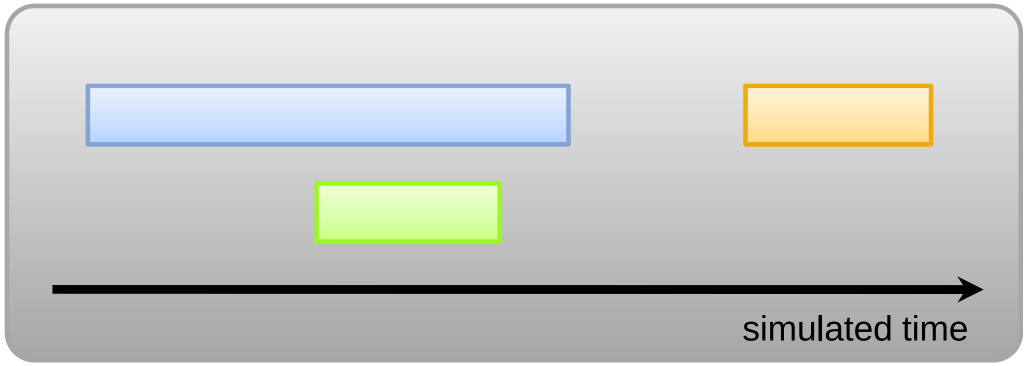


Simulator:

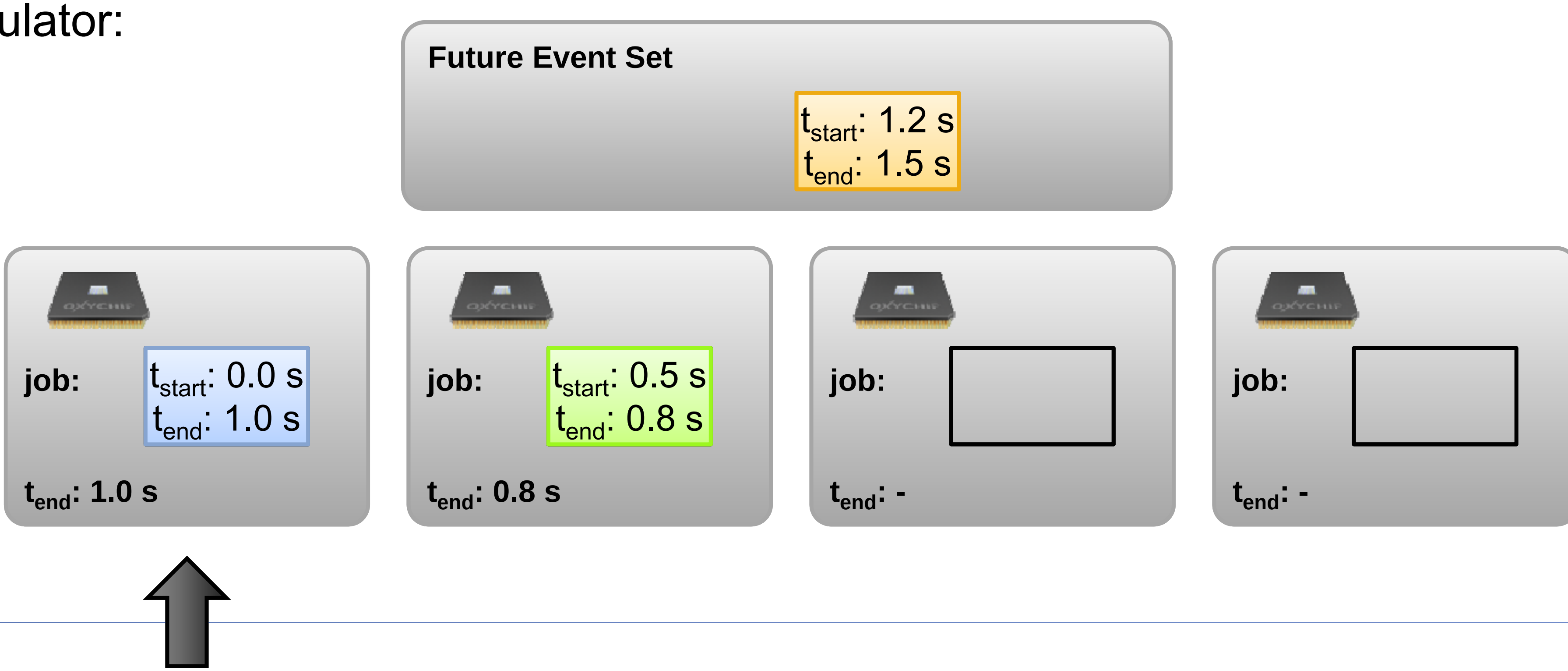


Event Scheduling Overhead: Solution

Example Schedule:

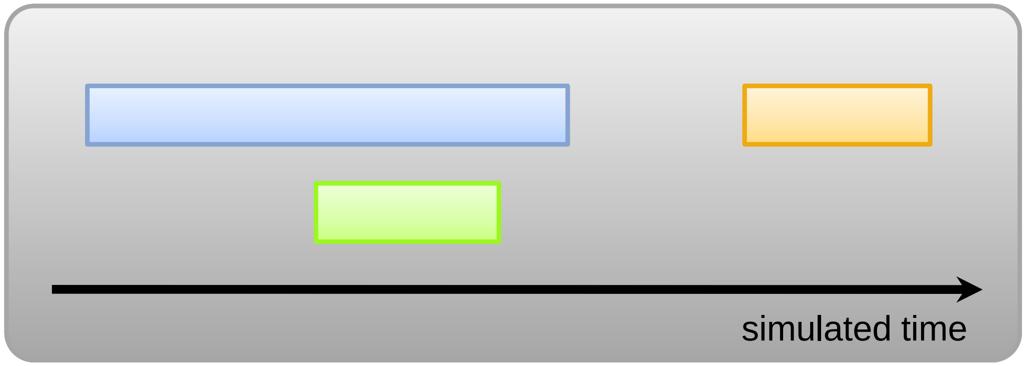


Simulator:

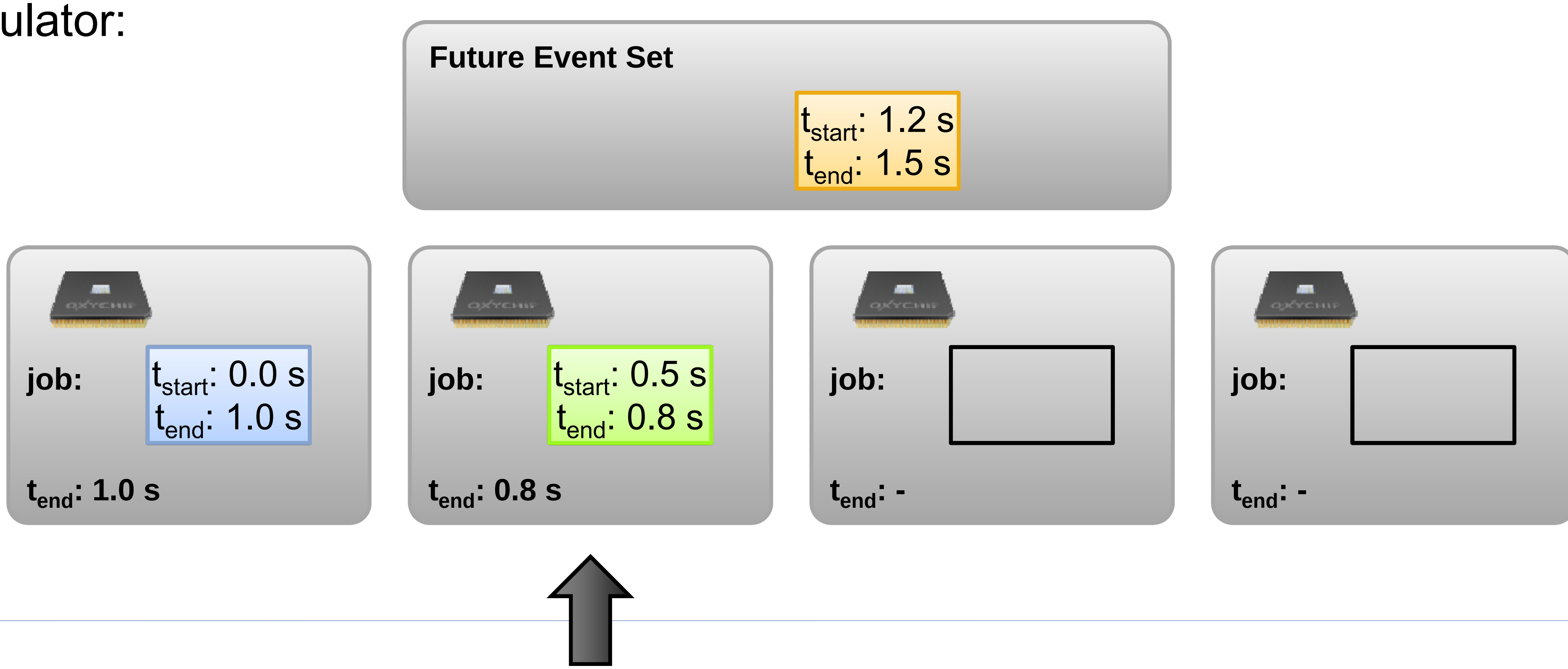


Event Scheduling Overhead: Solution

Example Schedule:

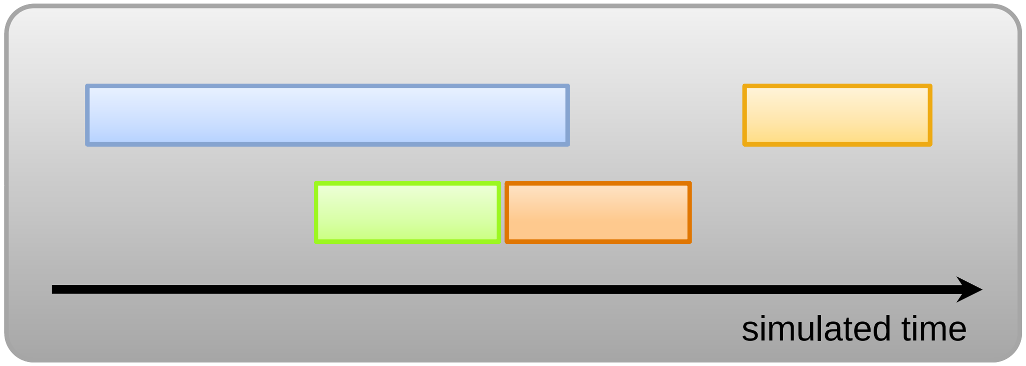


Simulator:

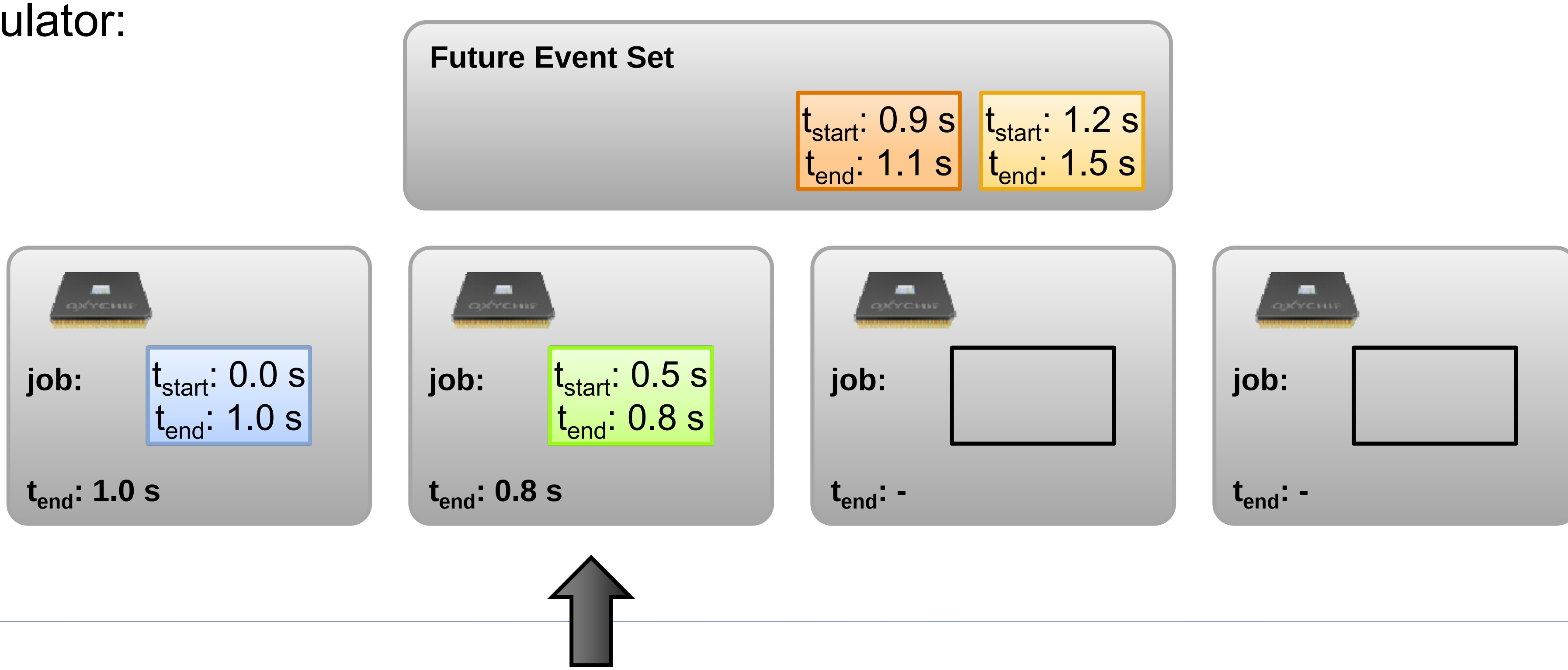


Event Scheduling Overhead: Solution

Example Schedule:

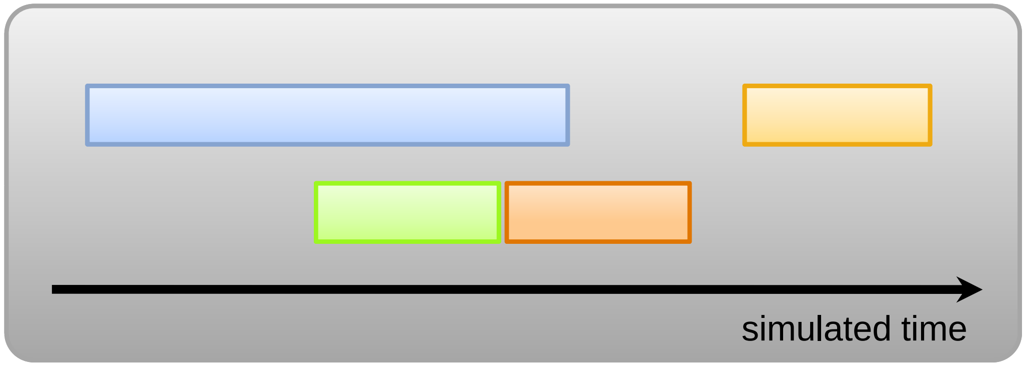


Simulator:

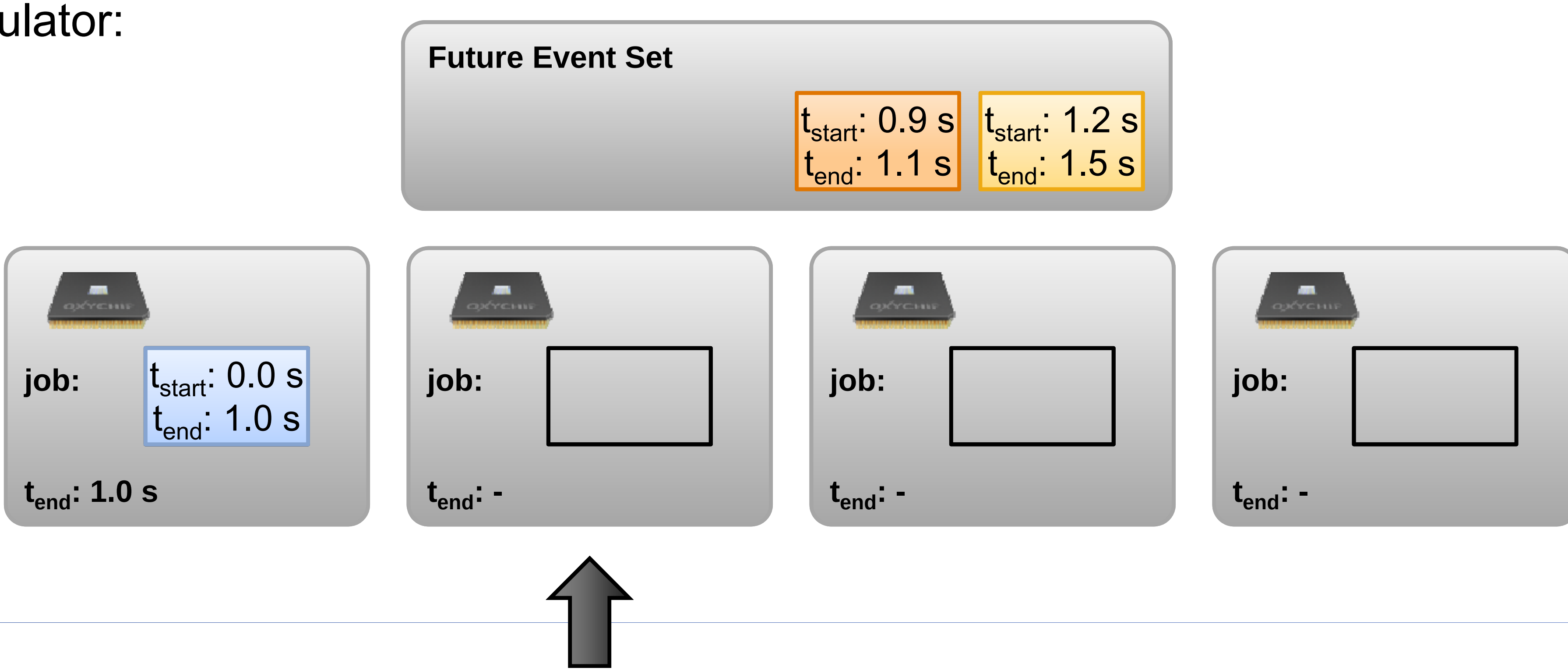


Event Scheduling Overhead: Solution

Example Schedule:

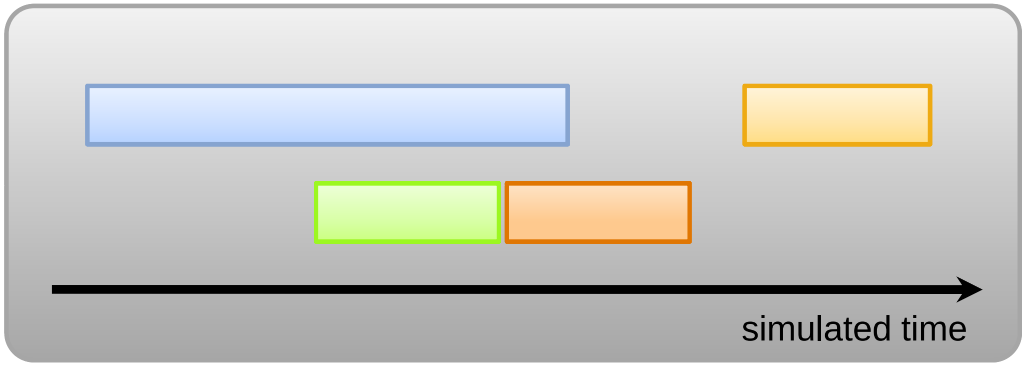


Simulator:

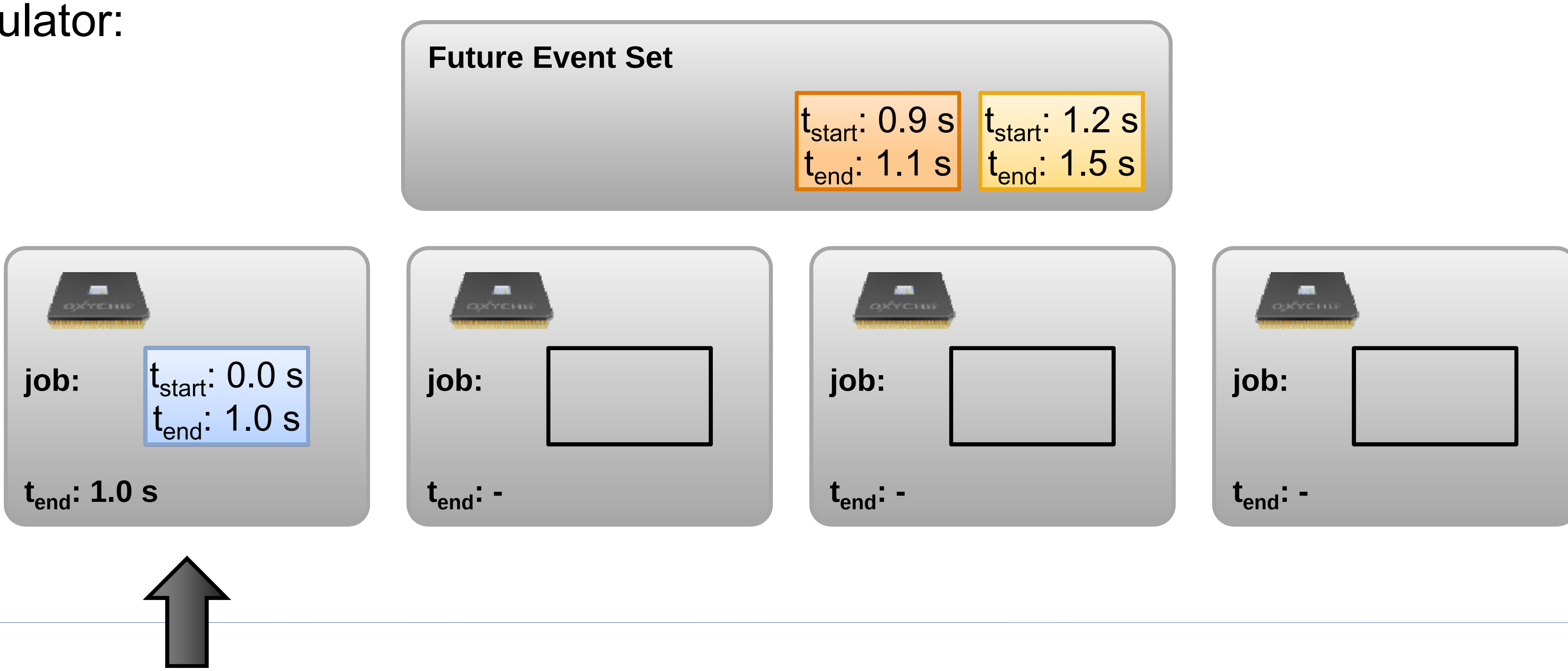


Event Scheduling Overhead: Solution

Example Schedule:

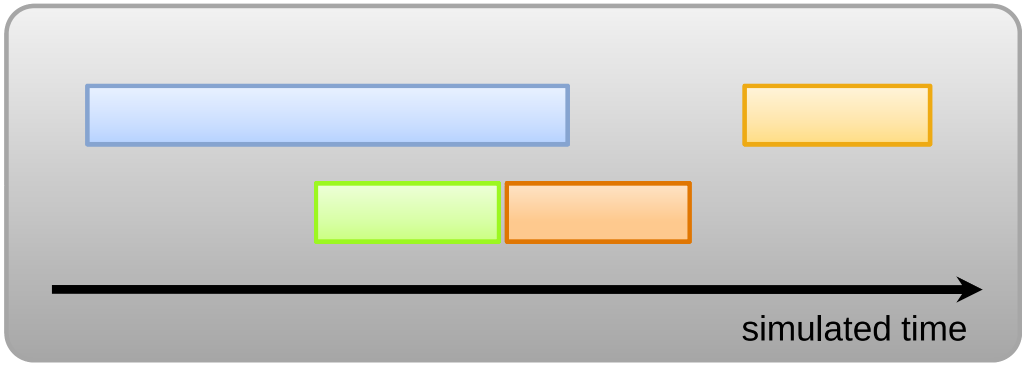


Simulator:

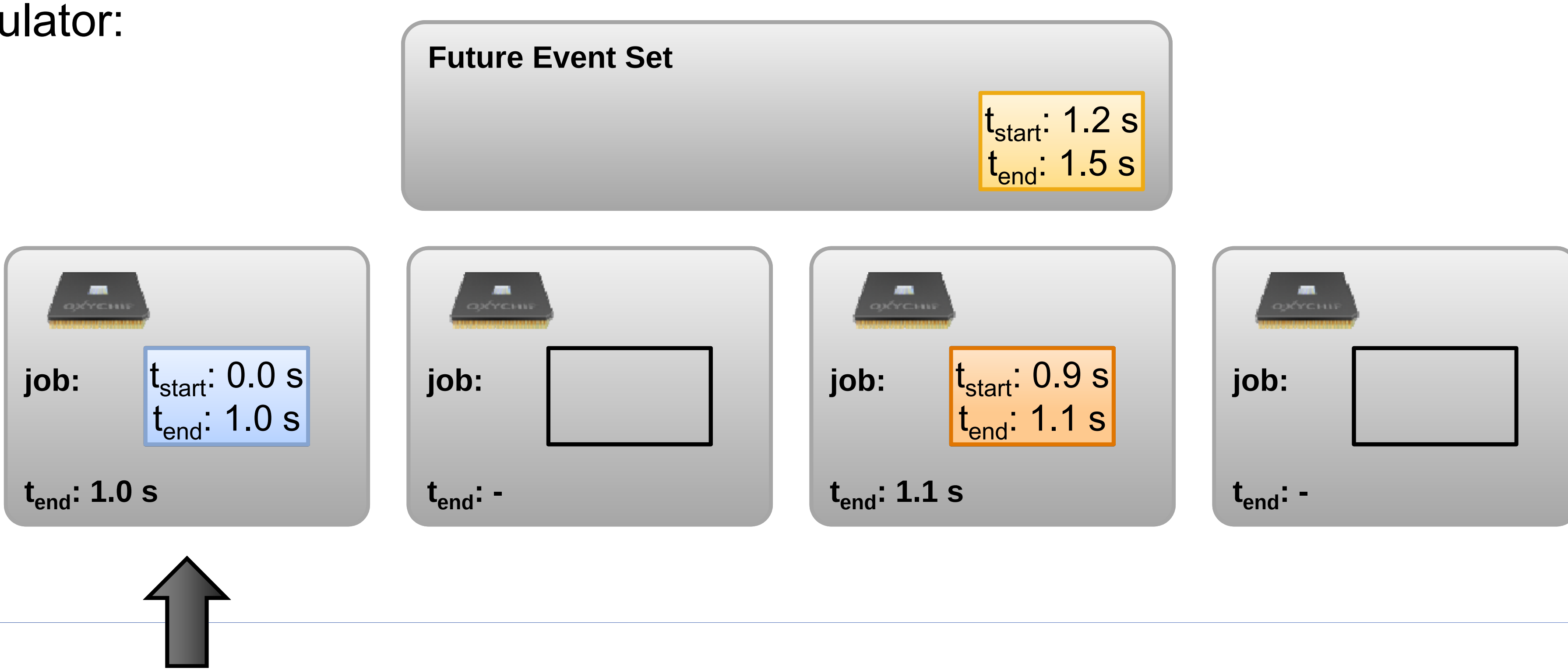


Event Scheduling Overhead: Solution

Example Schedule:

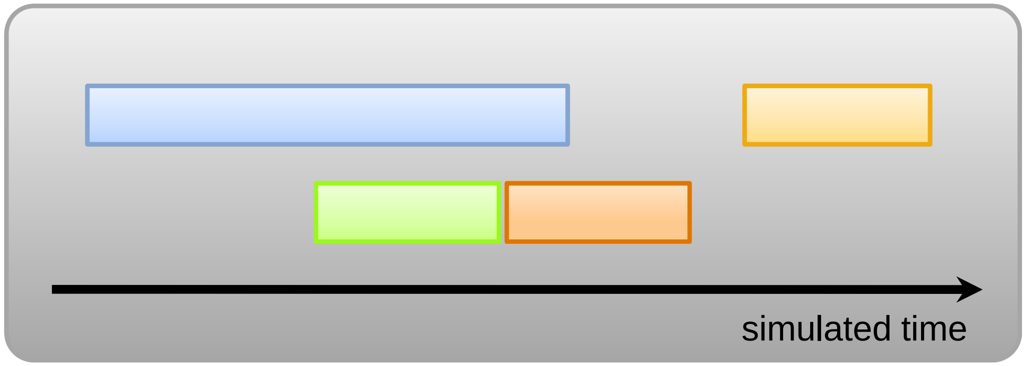


Simulator:

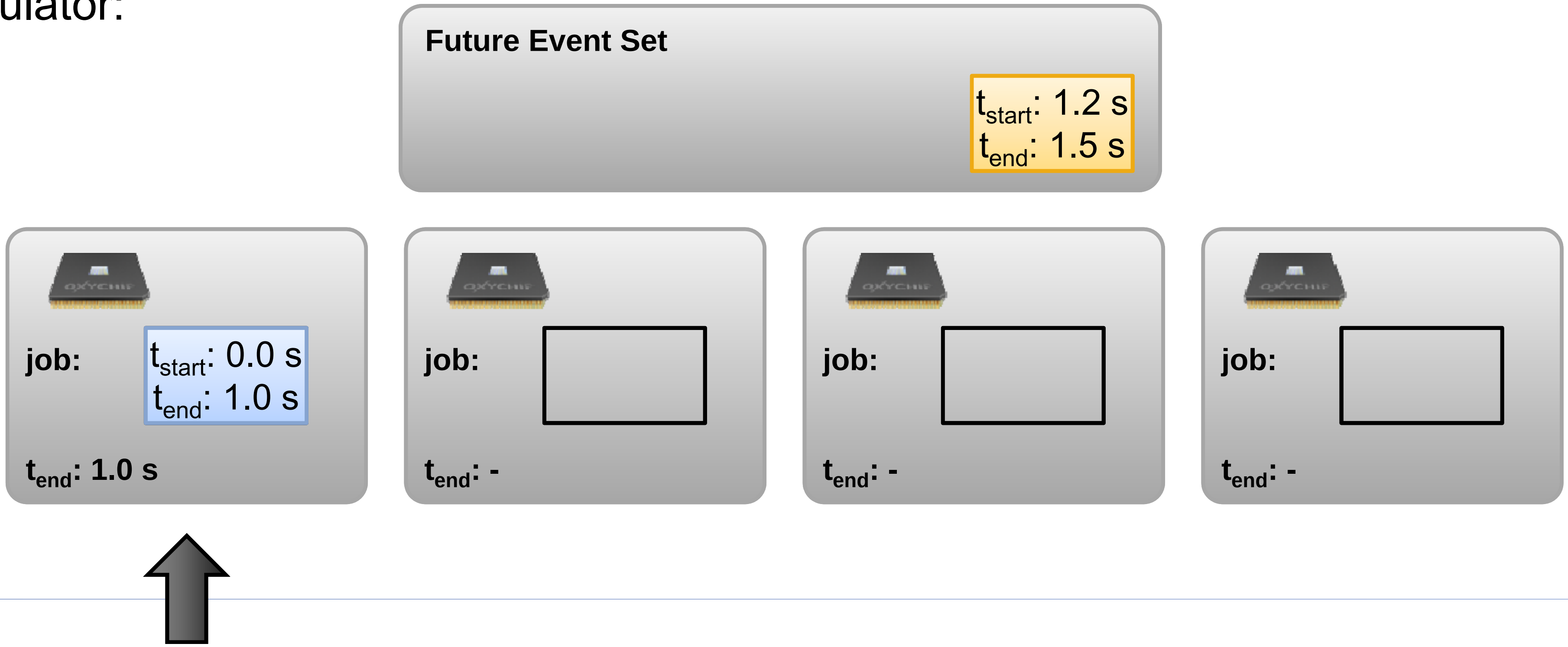


Event Scheduling Overhead: Solution

Example Schedule:

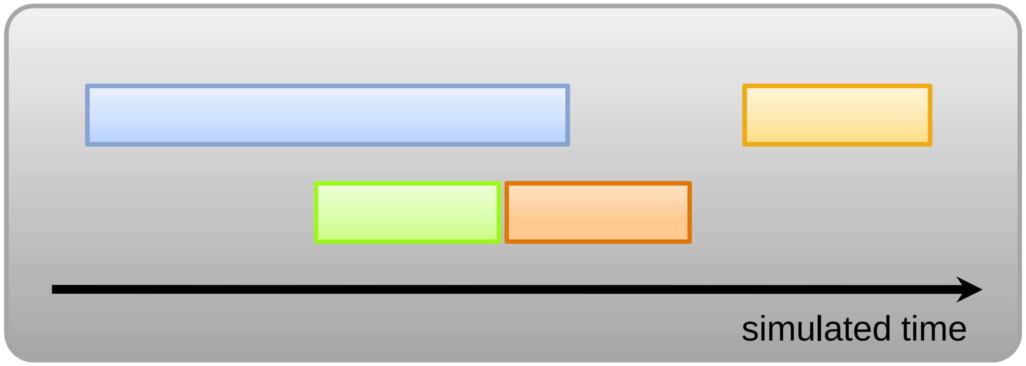


Simulator:

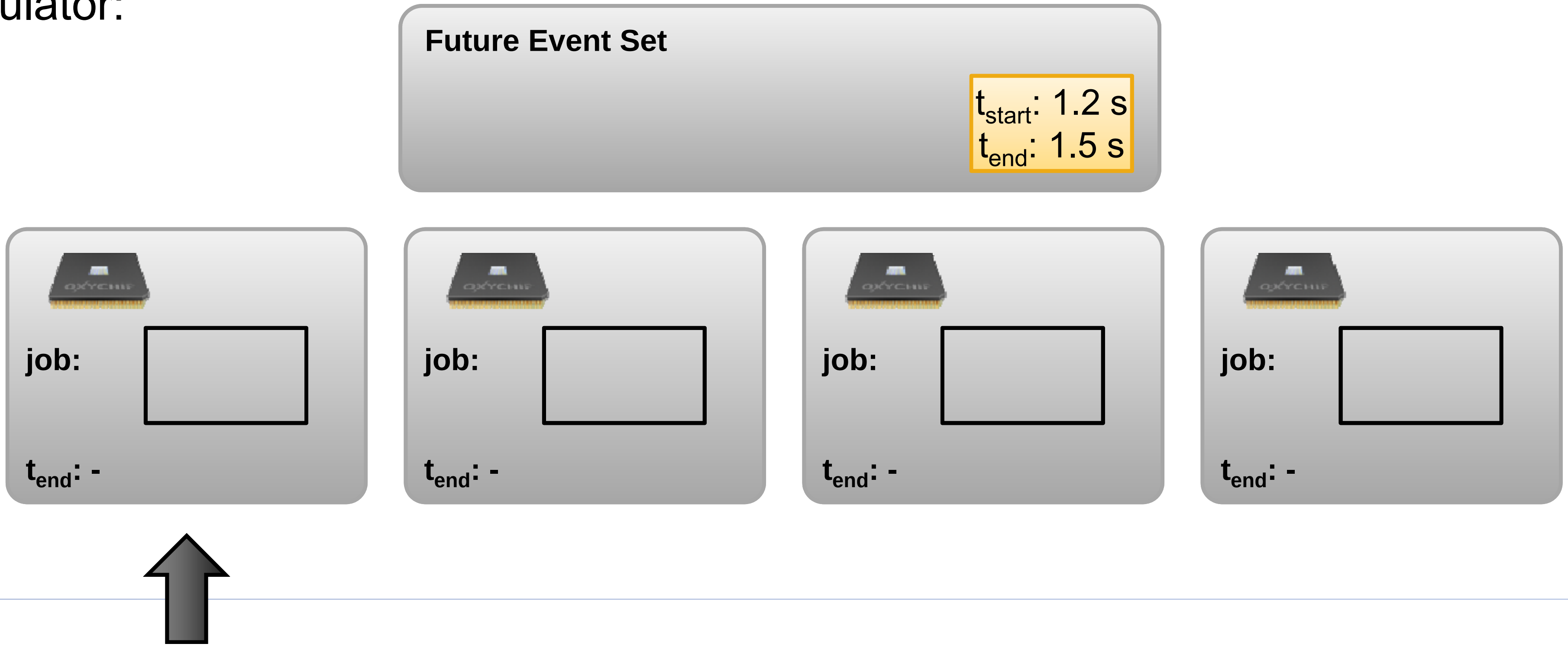


Event Scheduling Overhead: Solution

Example Schedule:

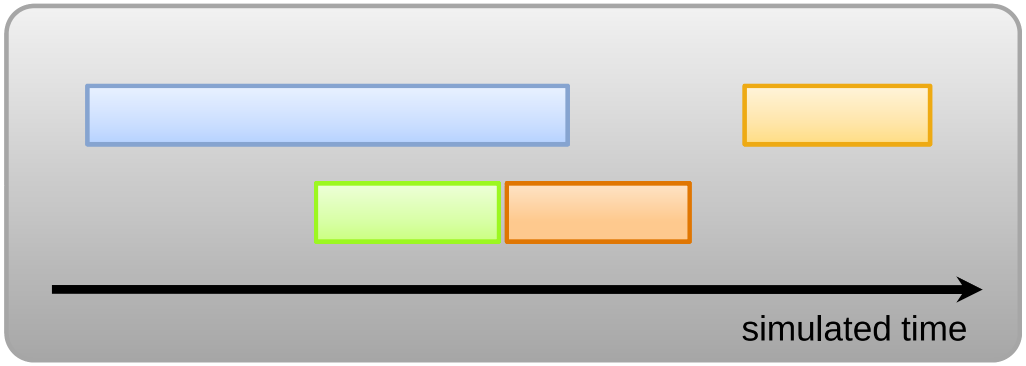


Simulator:

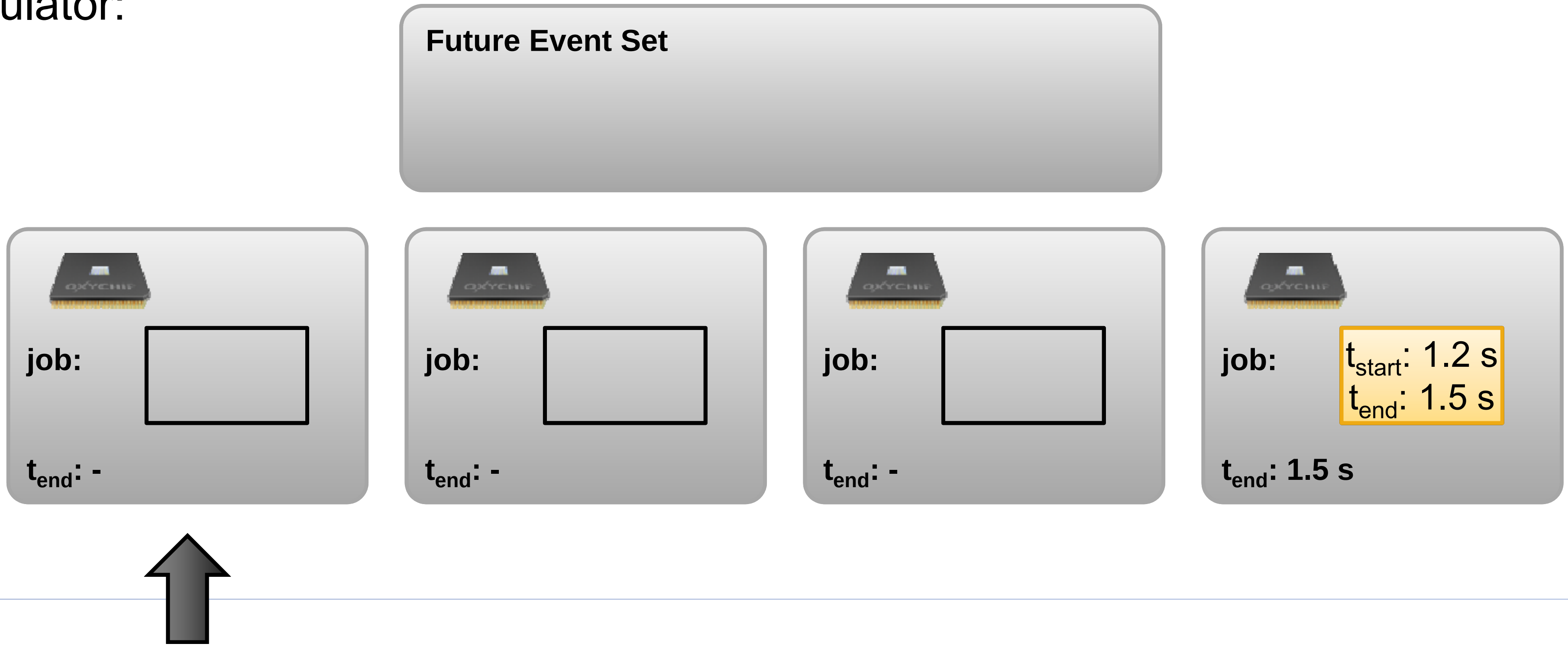


Event Scheduling Overhead: Solution

Example Schedule:

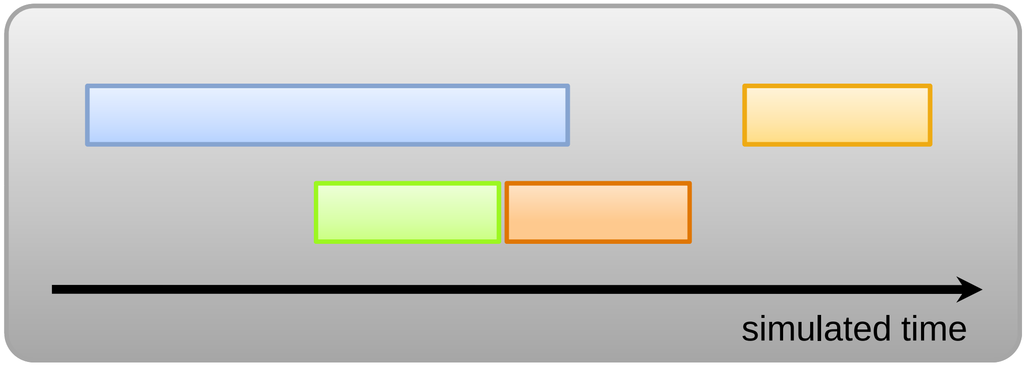


Simulator:

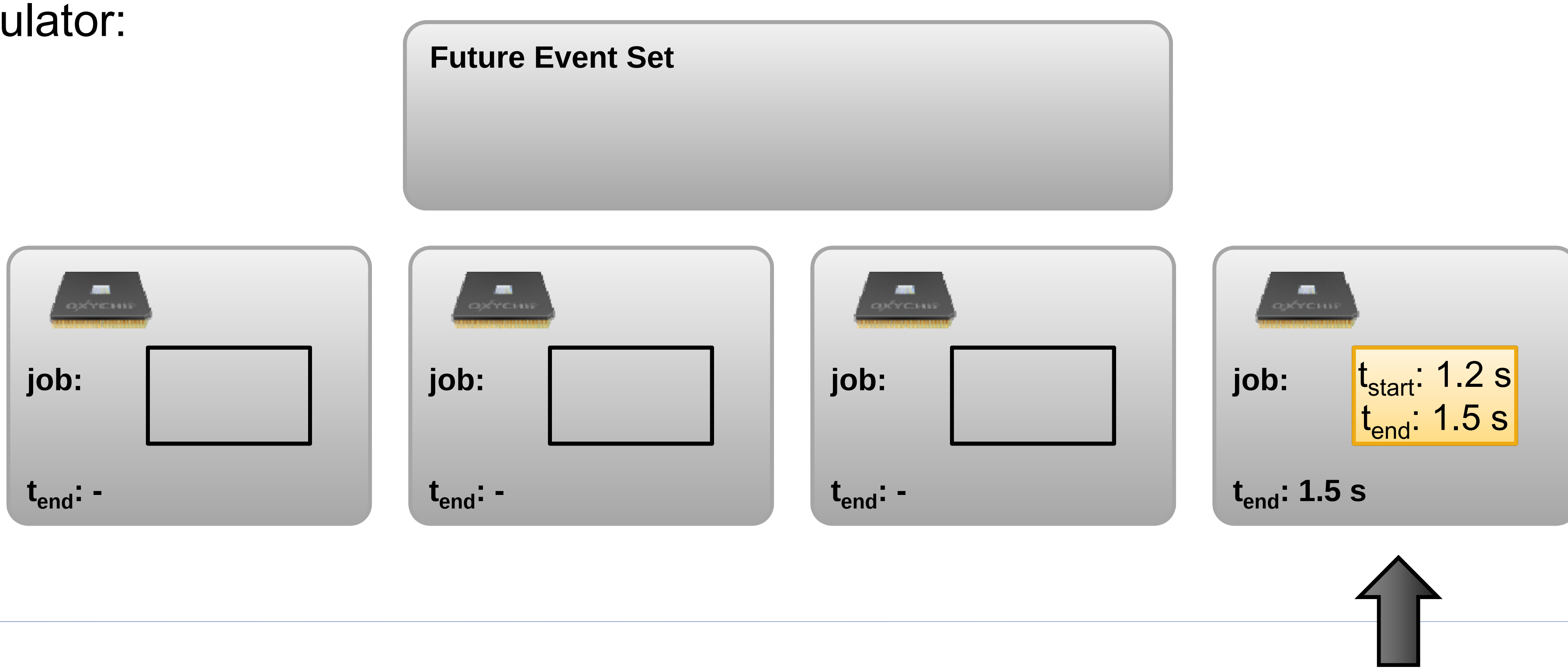


Event Scheduling Overhead: Solution

Example Schedule:



Simulator:



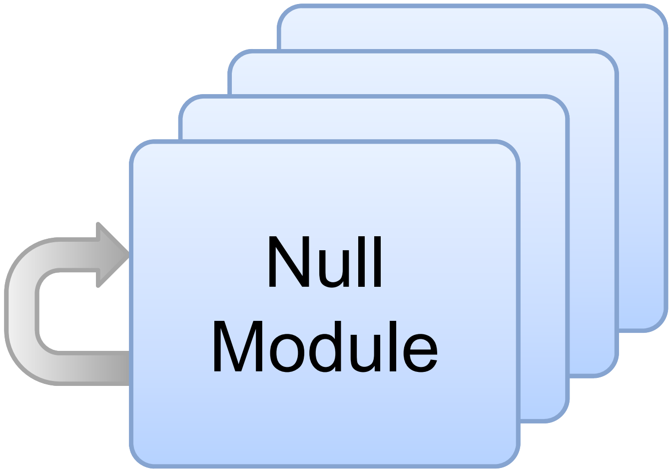
Evaluation

How does it perform?

Evaluation: Model

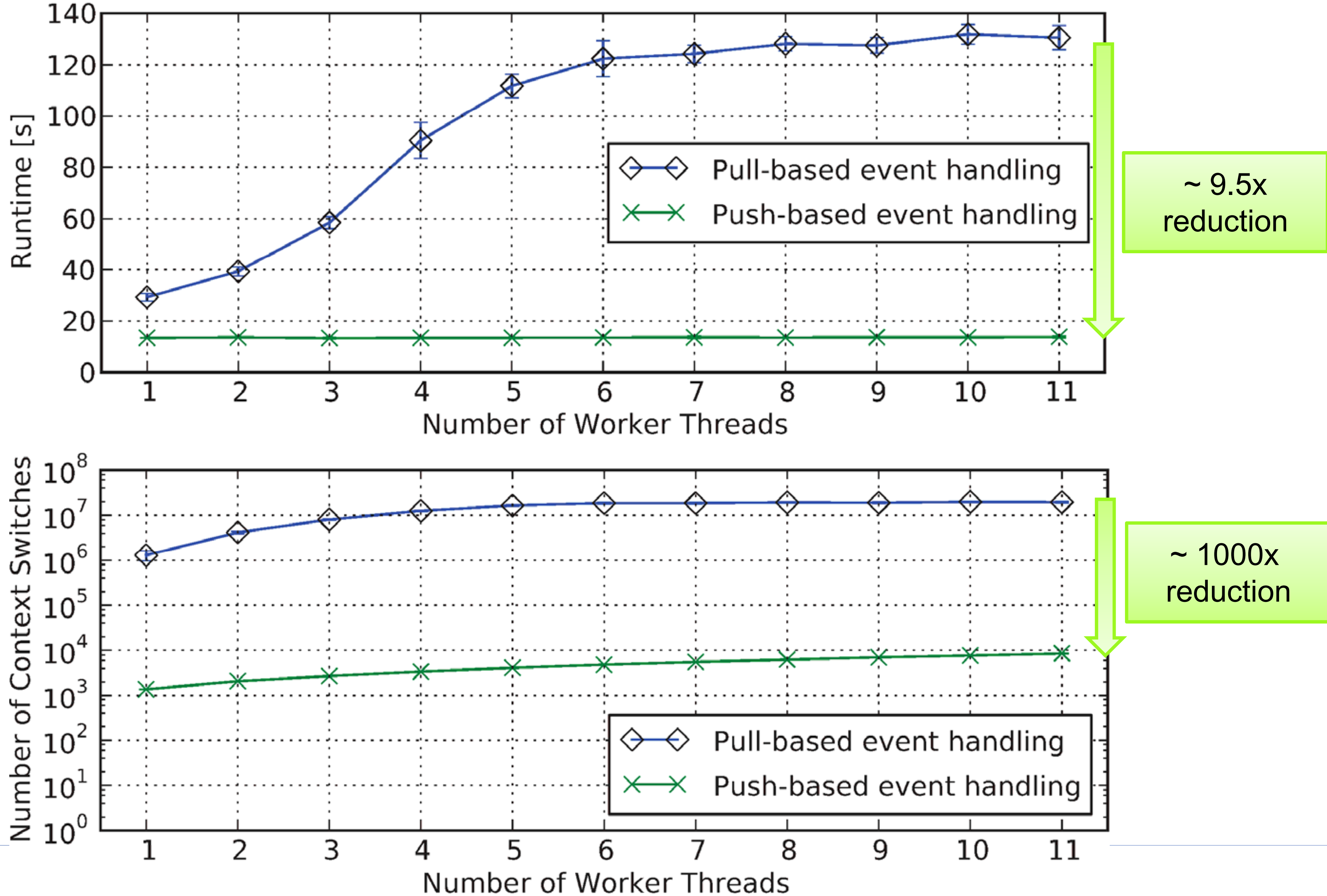
- **Design Goal**
  - Measure event handling overhead

- **"Null" Simulation Model**
  - 110 independent modules
  - Null module
    - Only re-schedules self-messages
    - No other computations
  - Execute 5.5 Million Events

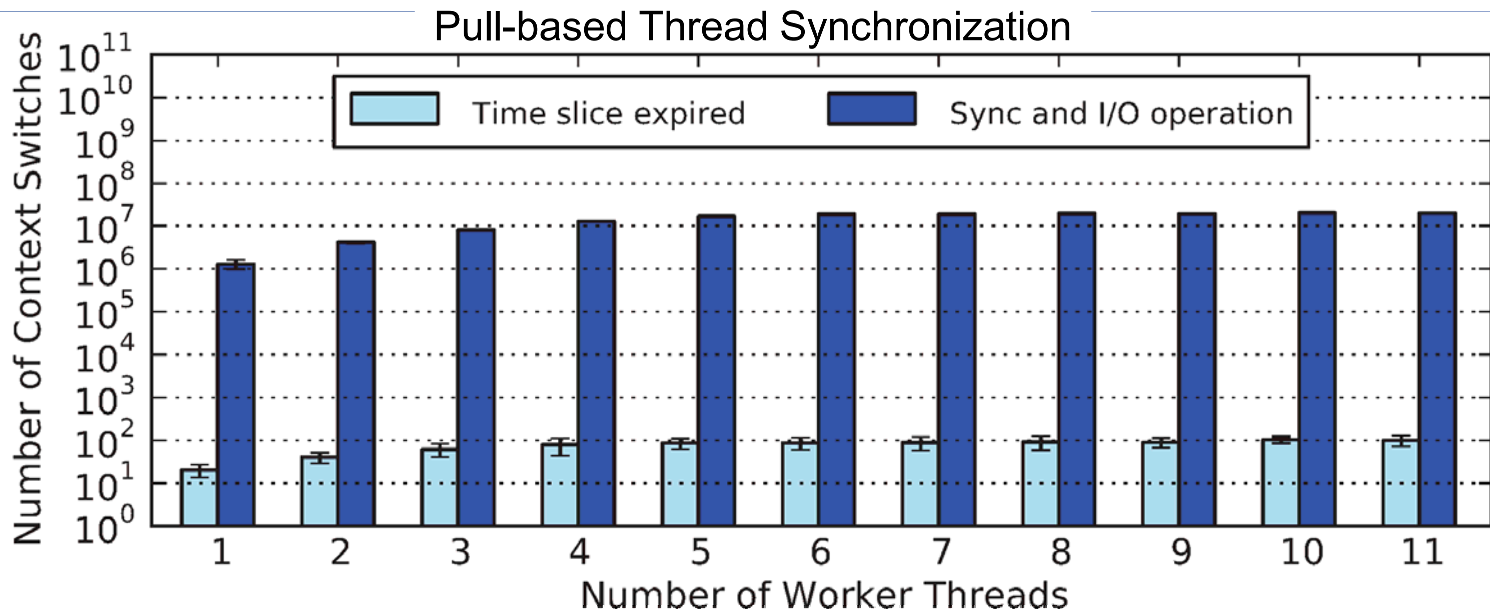


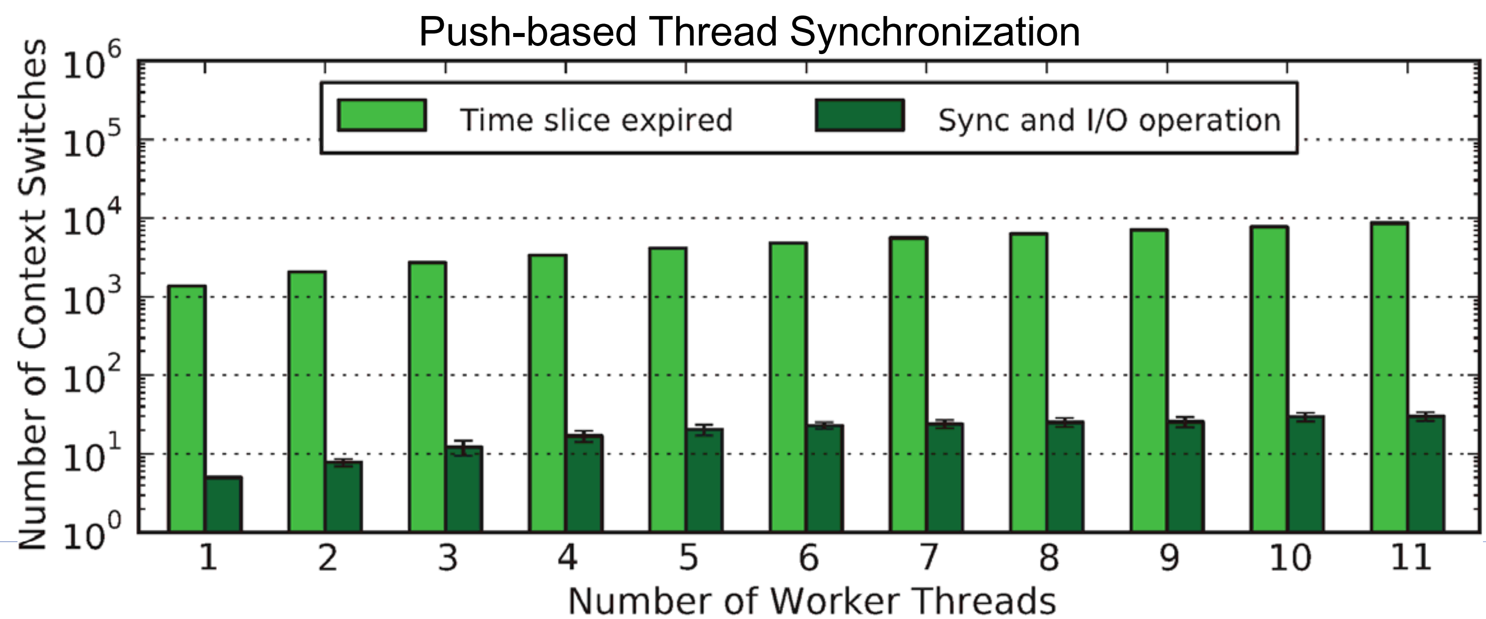
⇨ **Execution Time == Overhead**

Evaluation: Thread Synchronization Overhead

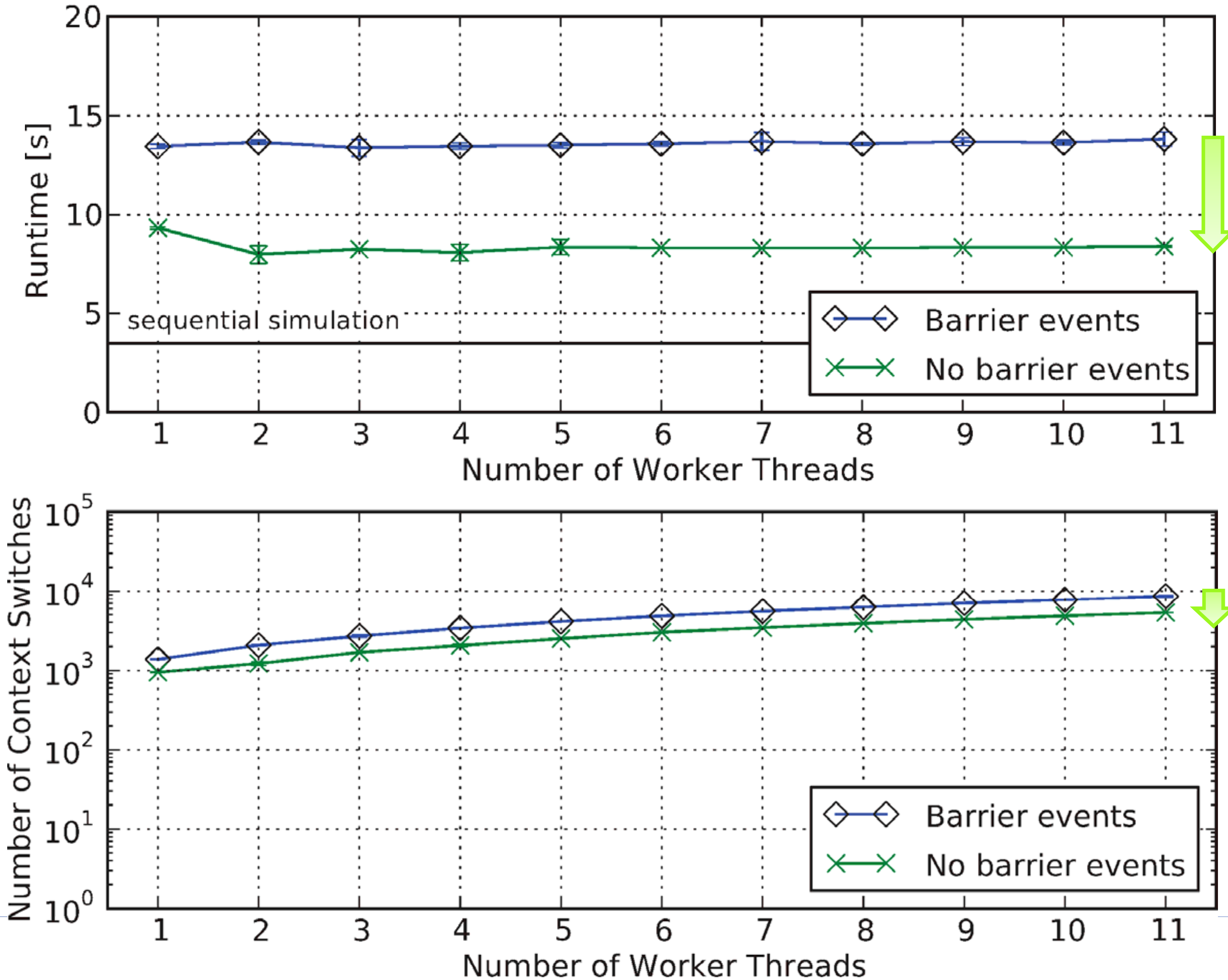


Evaluation: Analysis of Context Switches





Evaluation: Event Scheduling Overhead



~ 1.5x reduction

~ 1.5x reduction

Conclusions

The take away (barrier) message…

Conclusions

- **Parallelization Increases Overhead**
  - Thread synchronization
  - Event scheduling

- **Two Approaches to Mitigate Overhead**
  - **Push-based** thread synchronization minimizes context switches
  - **Local barrier information** replaces barrier messages

- **Overhead Reduction**
  - Push-based synchronization: ~9.5x reduction
  - Barrier algorithm: ~1.5x reduction
  - Combined: ~ 14x reduction

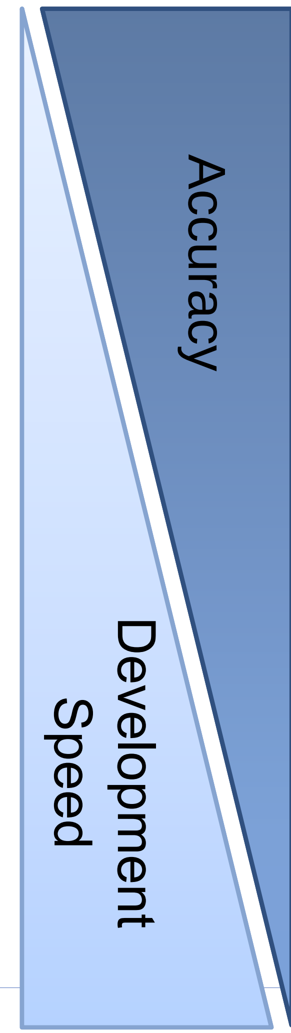
Thank You

Questions?

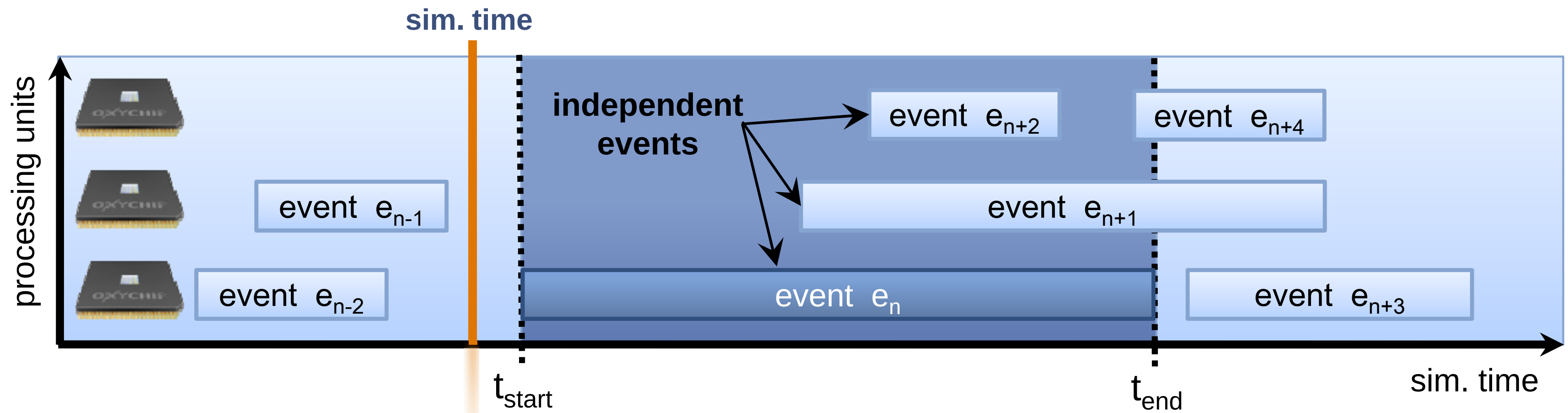
Backup Slides

Just in case someone asks…

- **How to Obtain Accurate Timing Information?**
  - Utilize existing techniques
- **Emulation**
  - Accurate profiling on emulated hardware
- **Automatic Simulation Calibration**
  - Applicable to simple hardware platforms only
- **Protocol Specifications**
  - Independent of hardware platform
- **Expert Knowledge**
  - Requires experience and careful judgment



- **Parallel Scheduling**
  - Offload independent events to worker CPUs



- **Causal Correctness**
  - Increasing timestamp order among dependent events
- **Data Integrity**
  - Compose model of self-contained functional units
  - Functional units correspond to concept of logical processes