



Jet Propulsion Laboratory
California Institute of Technology

Using Rescheduling and Flexible Execution to Address Uncertainty in Execution Duration for a Planetary Rover

Jagriti Agrawal, Wayne Chi, Steve Chien

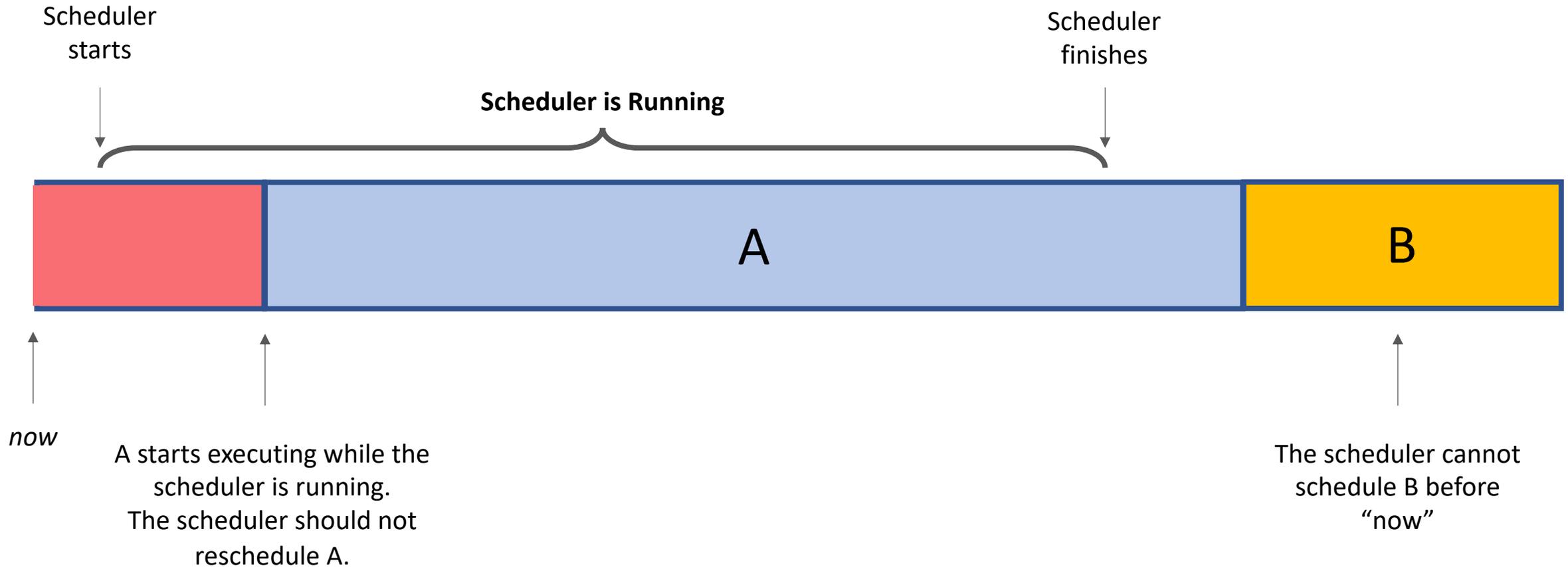
Challenge- How to Address Execution Uncertainty Given Non-zero Runtime Scheduler

- Reality (execution) consistently differs from our scheduling models
 - Activities may use fewer resources (time, energy, data volume) than expected
 - Activities may use more resources than expected, risking failure to perform all tasks
 - We consider effects of changes in specifically activity duration
 - Energy and data volume modeled as rates so they are implicitly affected
- Can respond to changes in activity duration by
 - 1) rescheduling to incorporate execution feedback
 - 2) allow generated schedule to adapt to changes during execution (Flexible Execution)
- The scheduler takes non-zero time to (re) schedule
 - Activities could start executing while the scheduler is running.
 - New changes may occur while the scheduler is running.
 - Scheduler runtime (T_{sc}) cannot be predicted exactly (non-determinism).

Goals and Assumptions

- Goal of scheduler
 - 1) schedule all activities
 - 2) schedule activities such that schedule has shortest possible *makespan*-difference between latest time an activity is scheduled to end and earliest time an activity is scheduled to start
- Assumptions
 - All activities fit in the initial schedule (the schedule is not oversubscribed).
 - Schedule activities form an approximately single serial path (only minor parallelism)
 - Focus on the M2020 Onboard Scheduler (Rabideau and Benowitz 2017)

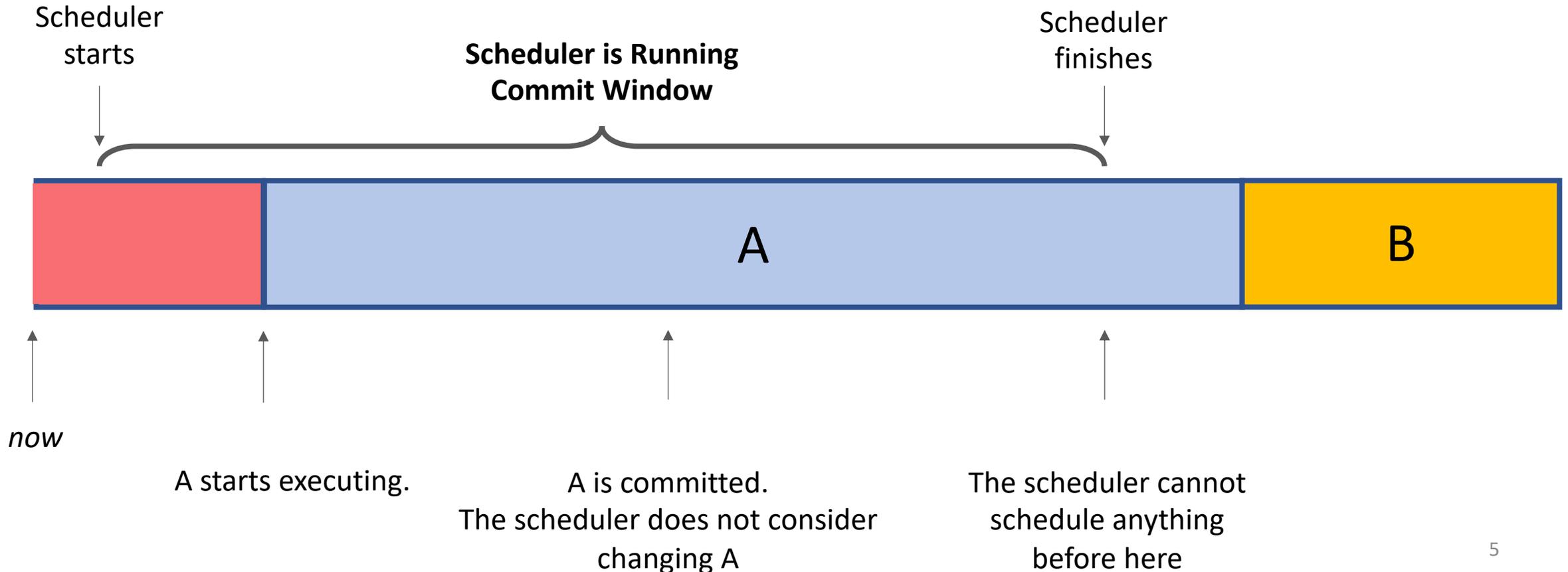
Non-zero runtime scheduler



Problem- What to execute while scheduler is running?

Commit Window Approach

- Activities that are scheduled to start during the commit window are committed to execution and cannot be rescheduled.
- Activities that are not scheduled to start during the commit window cannot be rescheduled to start in the commit window.

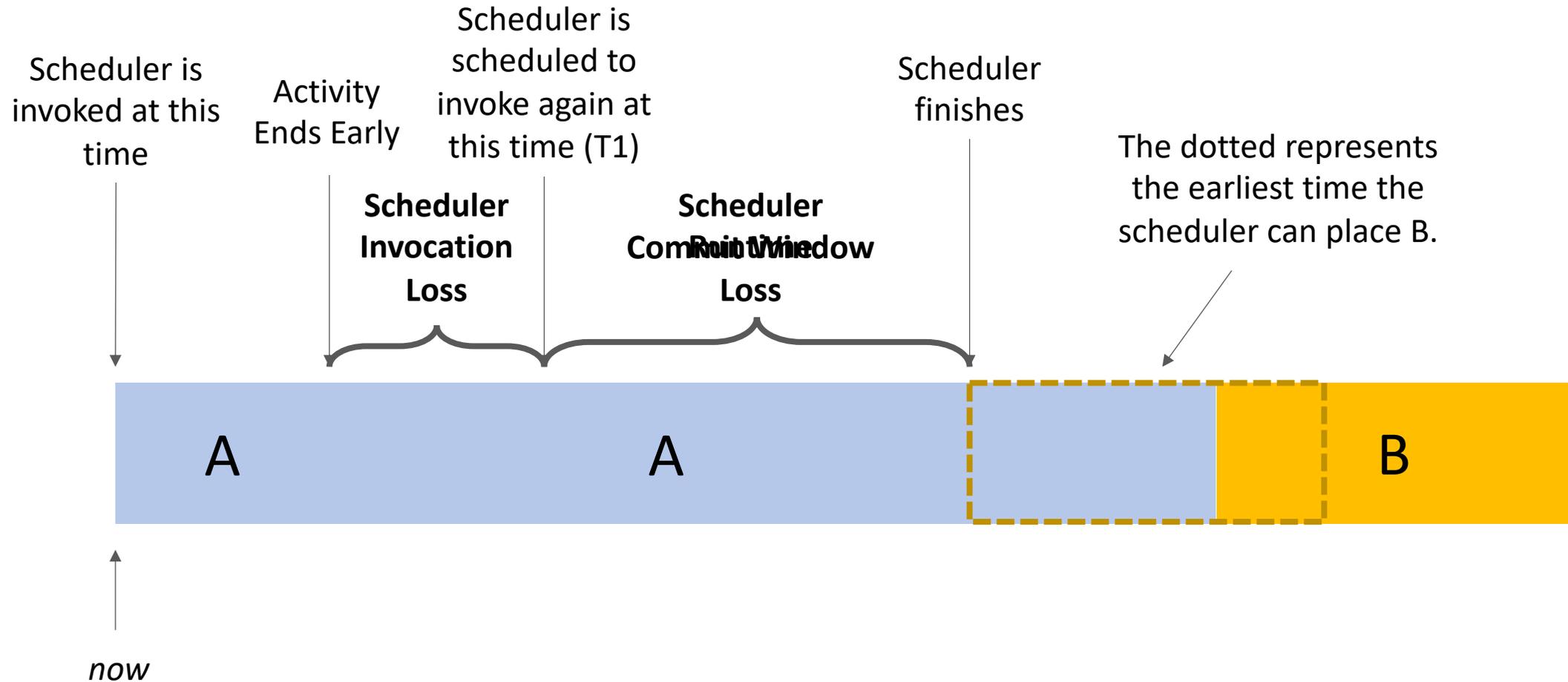


Framework for Analysis:

Scheduler Runtime Loss and Scheduler Invocation Loss

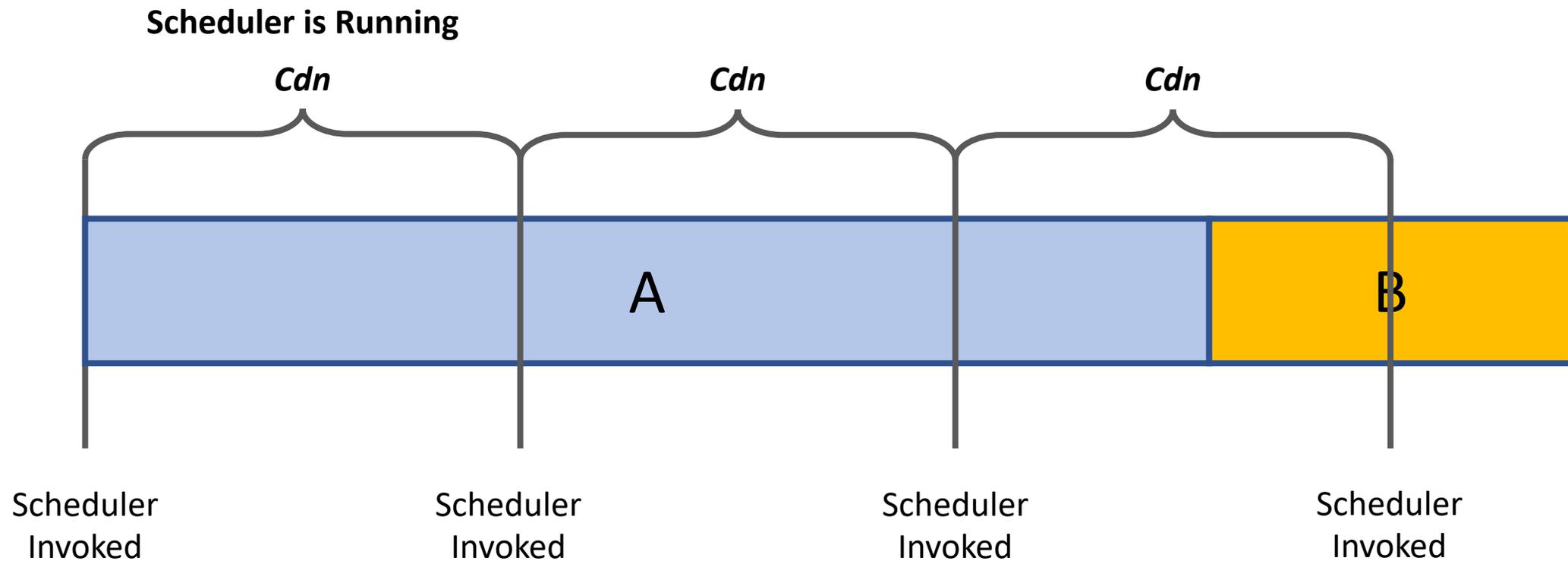
- Must quantify effectiveness of rescheduling and Flexible Execution techniques
- Quantify how techniques are able to reduce makespan by reducing losses: Scheduler runtime loss and scheduler invocation loss
- Scheduler runtime loss- the time that the scheduler is unable to recoup while the scheduler is predicted to be running
- Scheduler invocation loss- time lost due to waiting to reinvoke the scheduler.

Scheduler Runtime Loss and Scheduler Invocation Loss



Fixed Cadence Scheduling

- We assume that scheduler runtime (T_{sc}) = commit window.
- $Cdn \geq T_{sc}$ to ensure that the scheduler finishes before the scheduler is scheduled to invoke again.
 - For now, $Cdn = T_{sc}$.



- Large scheduler invocation loss and scheduler runtime loss

Event Driven Scheduling

- In order to prevent scheduler invocation loss, only invoke the scheduler when an “event” occurs.
 - An activity ends by more than Δ minutes

Activity ends
early and
Scheduler is
invoked

Scheduler
finishes here

(T_{sc})
Scheduler
Run Time
Scheduler is Running



↑
now

Event Driven Scheduling

- Invoke the scheduler only when an event occurs.
 - An activity ends by more than Δ minutes
- Scheduler Invocation Loss is mostly removed from Total Loss when the activity triggers Event Driven Scheduling.
 - If an activity ends by less than Δ minutes, it is considered scheduler invocation loss
- There will likely be fewer overall calls to the scheduler.
 - Fewer overall calls to the scheduler means more CPU can be allocated to the scheduler without starving lower priority CPU tasks
- Scheduler will not be reinvoked if event occurs while scheduler is running
 - Solving Transition Independent Decentralized Markov Decision Processes. (Becker, et. al. 2004)
- Back to back invocations could risk overconsumption of CPU
 - → Maximum reschedule limit
- No events result in scheduler never invoking
 - → Max time between rescheduling

Flexible Execution (FE)

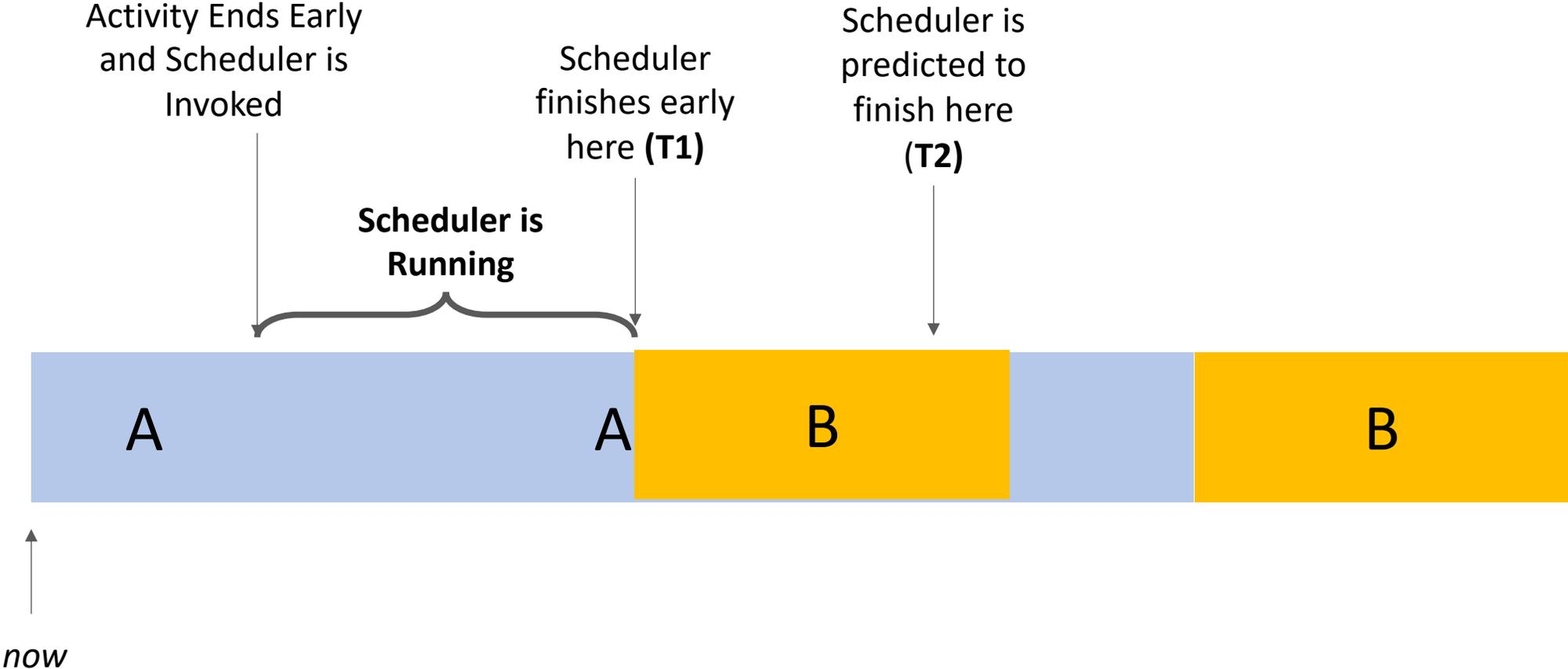
- Problem - previous methods are limited in response time by scheduler run time (T_{sc}) = commit window size
 - Takes limited advantage of activities ending early and does not significantly handle activities running long
- FE provides ability to change start times of activities in commit window
- Separate from scheduler and runs at faster frequency
- Prior work
 - Remote Agent Experiment (Muscettola et al. 1998; Pell et al. 1997, Muscettola 2002)
 - IDEA: Planning at the Core of Autonomous Reactive Agents (Gregory et al. 2002)
 - CASPER (Chien et al. 2000; Knight et al. 2001)
 - Generating Robust Schedules Through Temporal Flexibility (Policella et al. 2004)
 - Solve-and-Robustify (Policella et al. 2009)

Flexible Execution (FE) cont.

- Dispatch Window- Fixed amount of time after *Now*
 - Fe is only able to modify start times of activities within dispatch window
- Predecessor-Successor Relationship
 - Relative ordering between activities that share the same unit resources or share dependencies
 - E.g) A must complete successfully before B starts → A is a predecessor of B and B is successor of A
- FE allows execution according to a directed acyclic graph based on predecessor-successor relationships of activities in dispatch window
 - Called at a frequency of 1 Hz
- Two variations of FE algorithm
 - 1) Extended Veto
 - 2) Extended Push
 - Both perform same when activities end early; behavior differs when activities run late

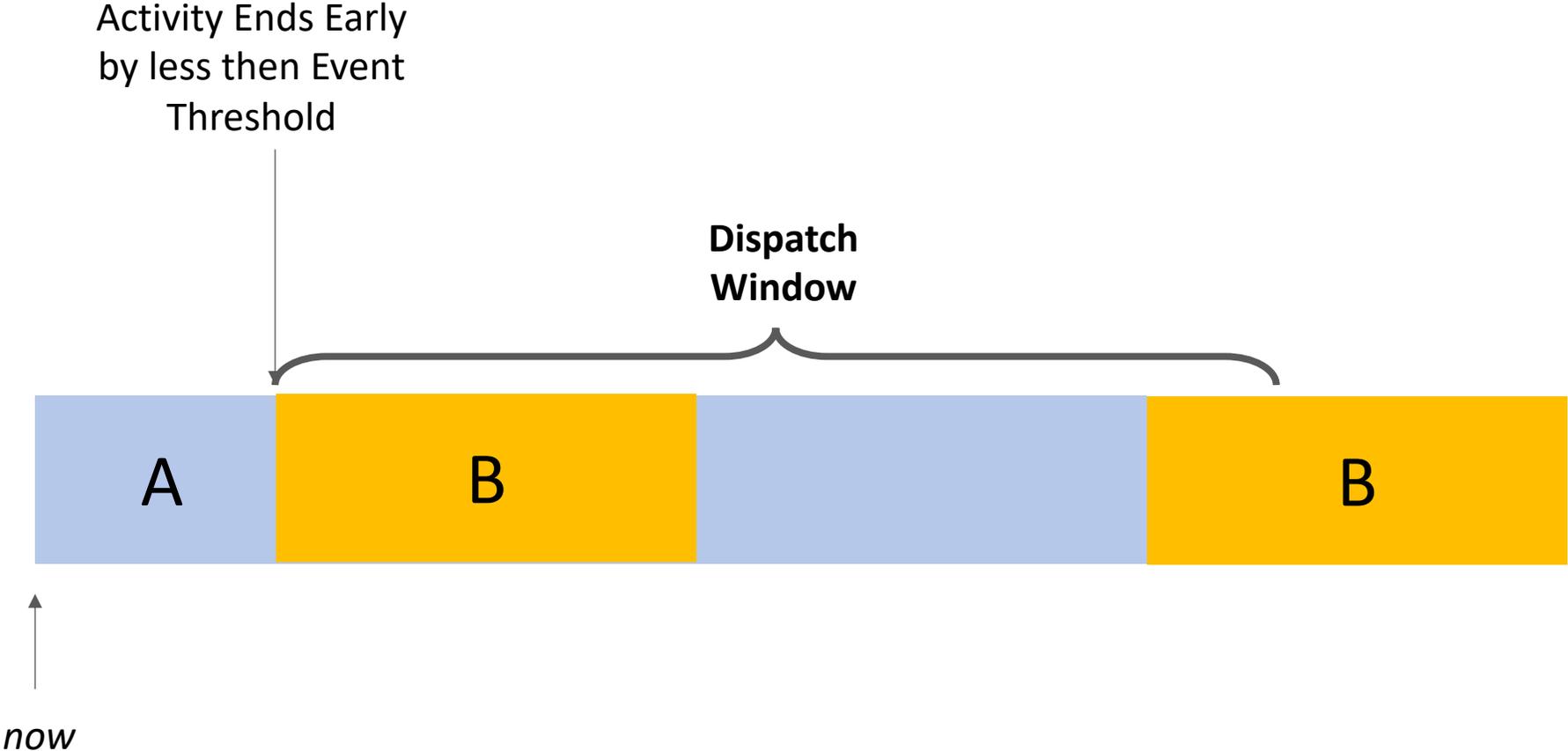
FE- Scheduler Ends Early

- FE can take advantage of the scheduler taking less time than predicted to run



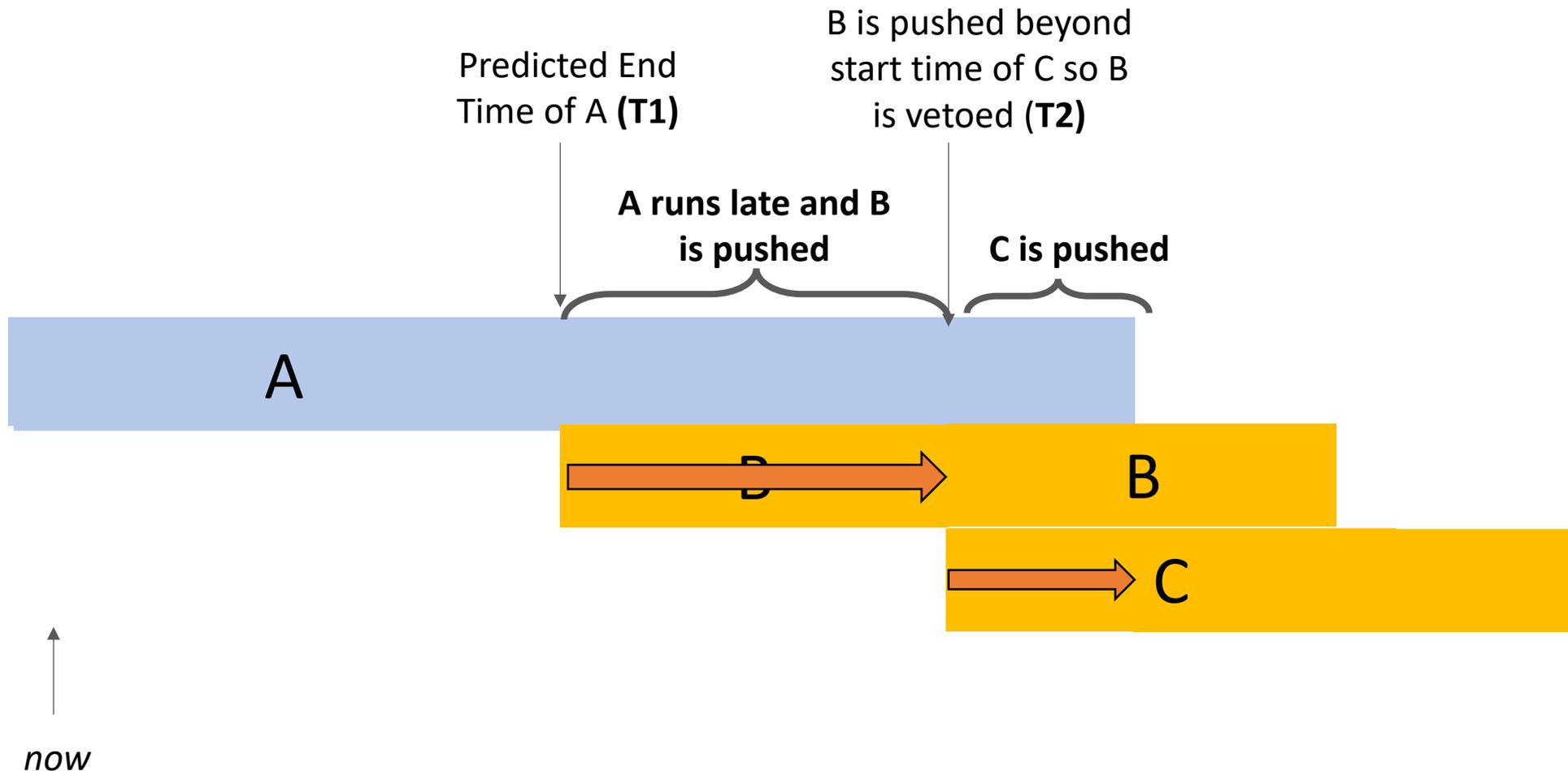
FE- Activity Ends Early

- FE can pull successor activities forward if predecessor ends early by less than event threshold



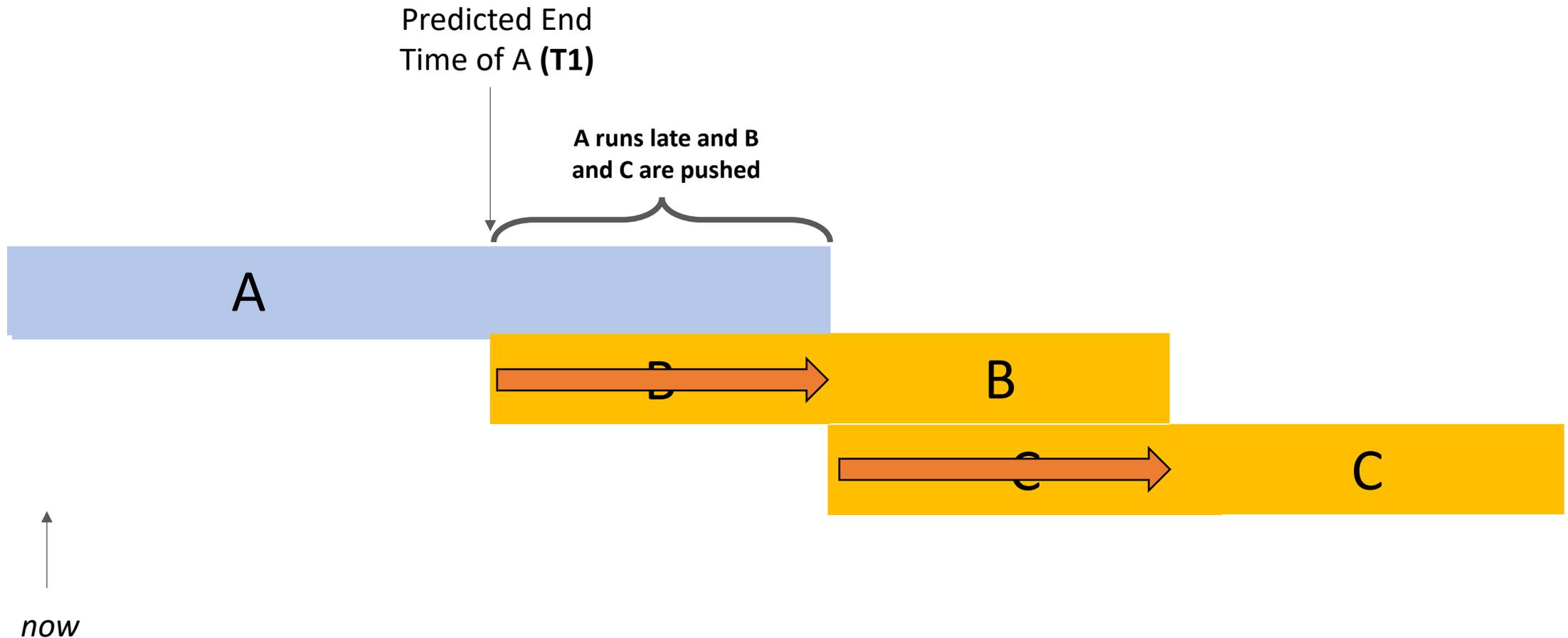
Activity Ends Late- FE Extended Veto

- If activity is pushed beyond some limit, L , it will be vetoed
- If activity being pushed exceeds start time of another scheduled activity, it will be vetoed



Activity Ends Late- FE Extended Push

- Activities continue to be pushed if a predecessor runs late (ripple push)



Complications with Flexible Execution

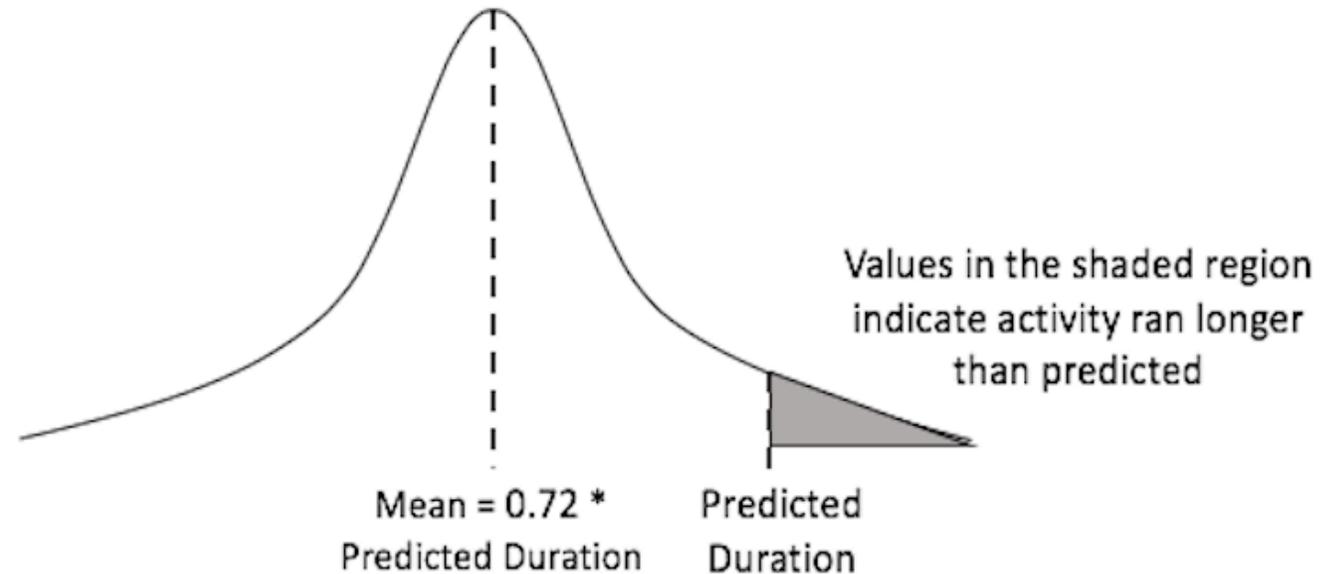
- Changes in non-depletable resources (power) or depletable resources (energy/data volume) do not cause activities to start earlier or later
- Allowed to run while scheduler is scheduling → actual execution may be different between when scheduler started running and when it ended → inconsistent schedule
- If activity runs long, activity can be pushed outside of commit window → possible that activity will not be scheduled in current invocation

Empirical Results- Inputs

- Use Sol Types
 - **Sol Types**- currently best available data on expected M2020 rover operations
 - Not always completely serial, contain execution, dependency constraints, and setup activities (e.g. preheats)
 - Each sol type contains 20-40 activities
 - Each sol type has a different objective (e.g. driving, more drilling, etc.)
 - Medium Drive, Short Drive, Abraded Proximity Science, Natural Proximity Science, Workspace
- 40 runs of simulation of execution on each of 8 sol types for each scheduling method per independent variable value
 - Mars 2020 surrogate scheduler- an implementation of same algorithm as Mars 2020 onboard scheduler but intended for a Linux workstation environment
 - Activity durations are varied in each run based on probabilistic model using data from the Mars Science Laboratory (MSL) Mission

Model to Vary Activity Durations

- Data from Mars Science Laboratory Mission (Gaines et al. 2016) indicates activities completed on average 28% early
- Use normal distribution to determine activity execution durations
 - Mean- 72% of nominal activity durations
 - Stdev determines percentage of activities that will take longer than predicted duration

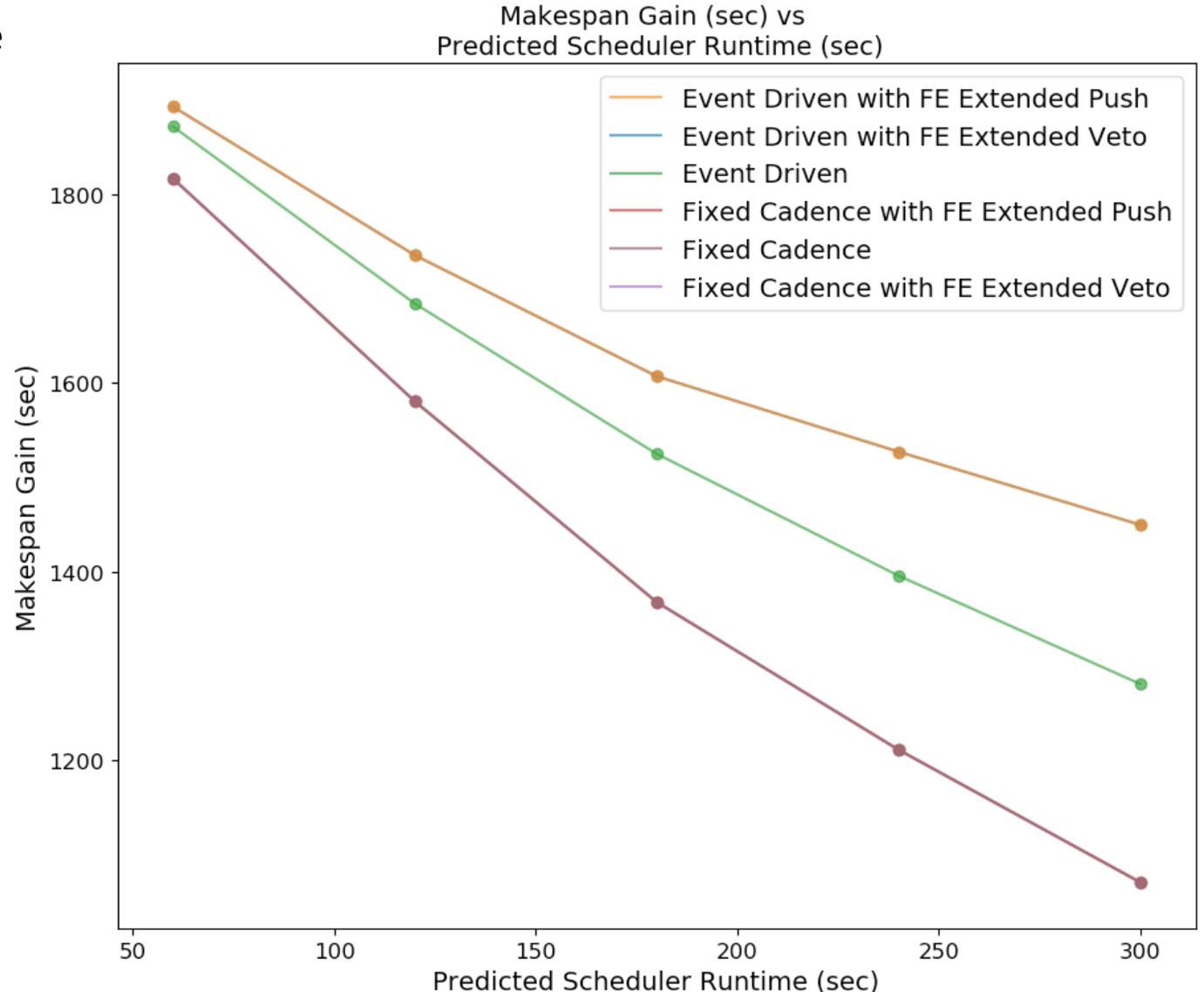


Makespan Gain vs Predicted Scheduler Runtime

- Commit window = predicted scheduler runtime dispatch window
- Activities **only end earlier than expected**
- **Scheduler always takes predicted runtime**

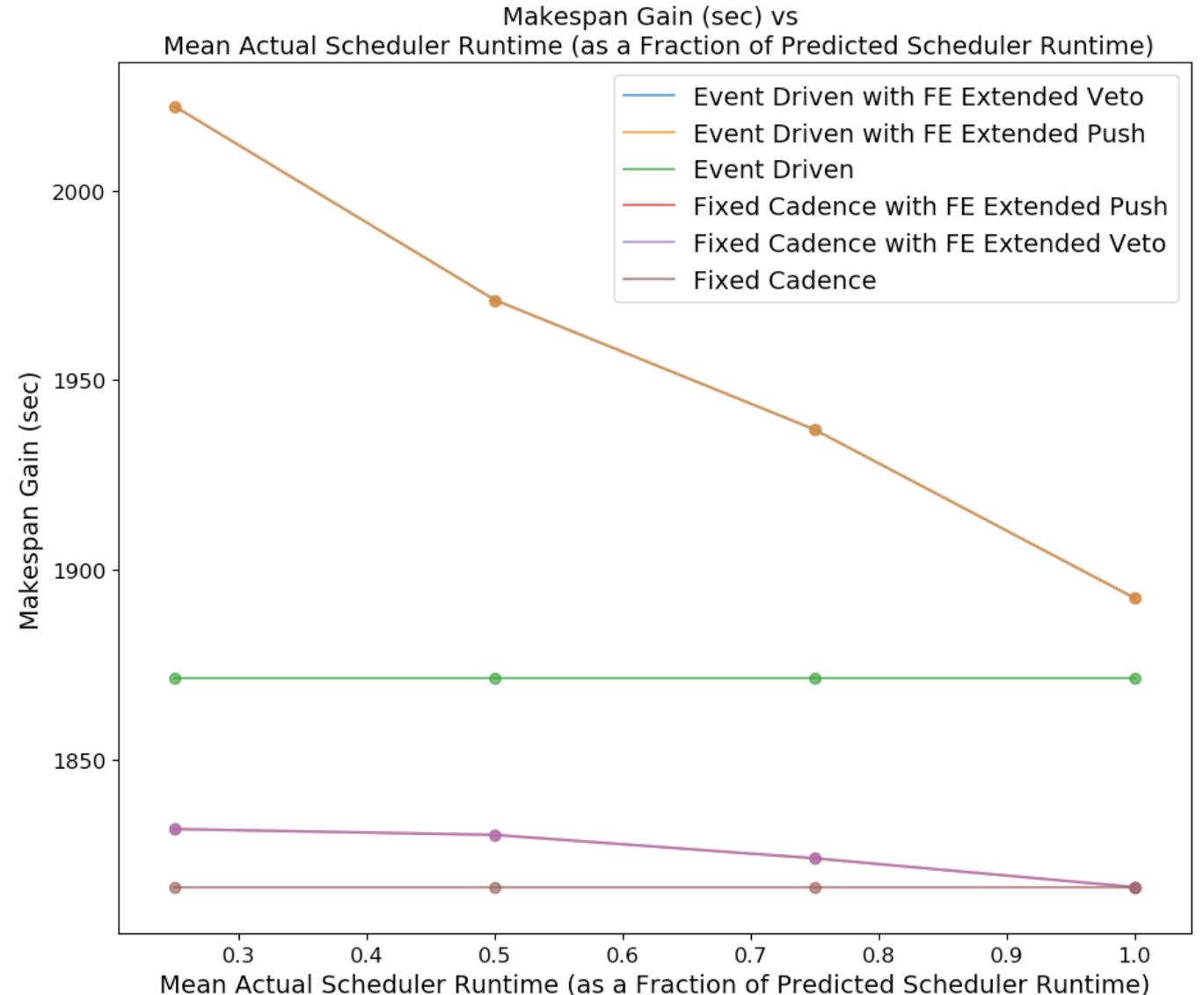
- **Makespan:** difference between the latest time an activity is scheduled to end and the earliest time an activity is scheduled to start
- **Makespan gain:** difference between the makespan of the initial schedule and the final executed schedule.

- Lowest Curve: Fixed Cadence, Fixed Cadence with Extended Push FE, Fixed Cadence with Extended Veto (FE does not help with Fixed Cadence and if scheduler never ends early, cadence = Tsc_p)
- Middle Curve: Event Driven with no FE
- **Highest Curve: Event Driven with either FE**



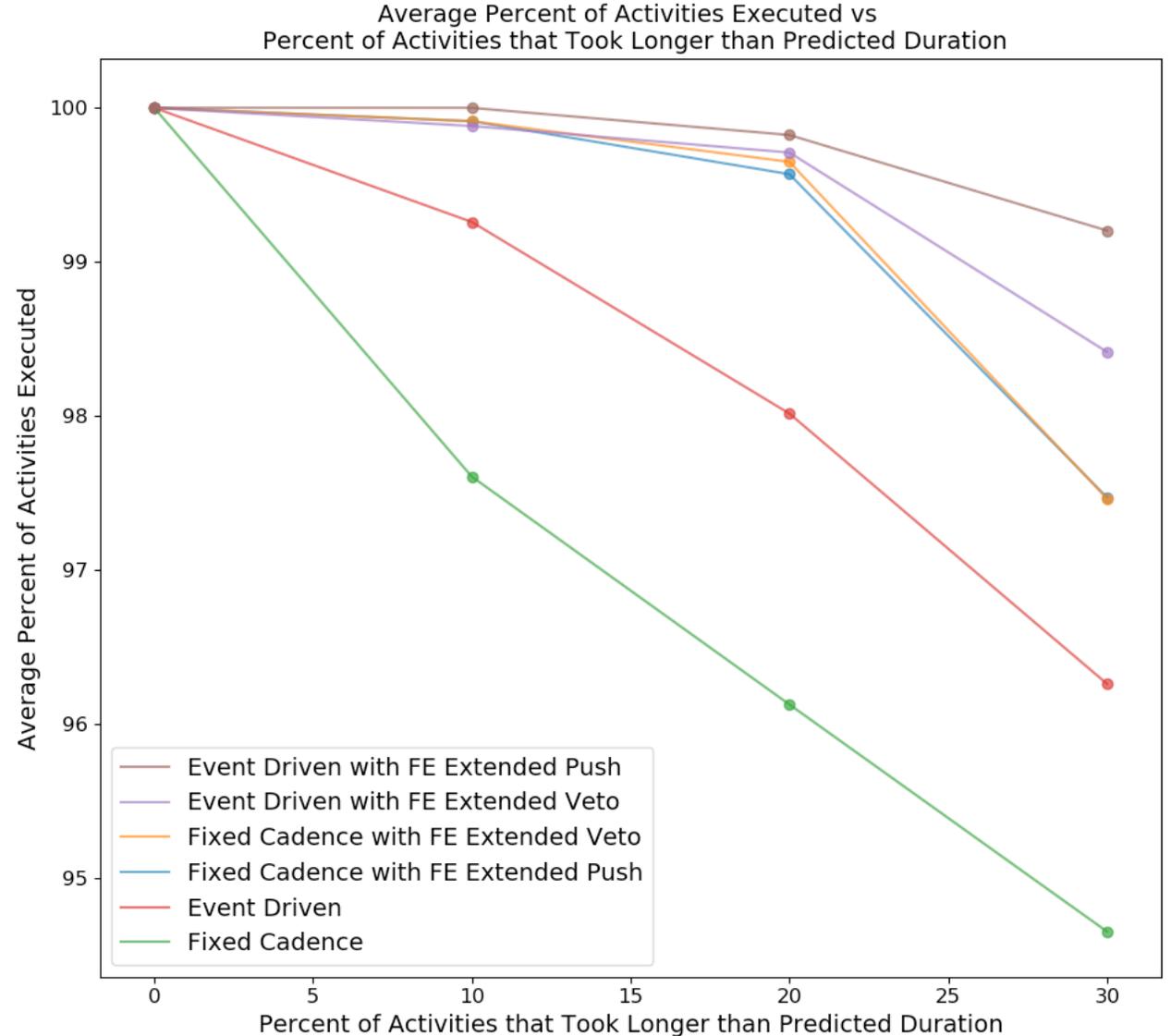
Makespan Gain vs Actual Scheduler Runtime (as a Fraction of Predicted Scheduler Runtime)

- Commit window = predicted scheduler runtime = dispatch window
- Activities **only end earlier than expected**
- **Scheduler can take less time than expected**
- Scheduler runtime determined via pseudo normal distribution where value is truncated if scheduler runtime is greater than T_{sc_p}
- Lowest Curve: Fixed Cadence, Event Driven
- Middle Curve: Fixed Cadence with either FE
- **Highest Curve: Event Driven with either FE**

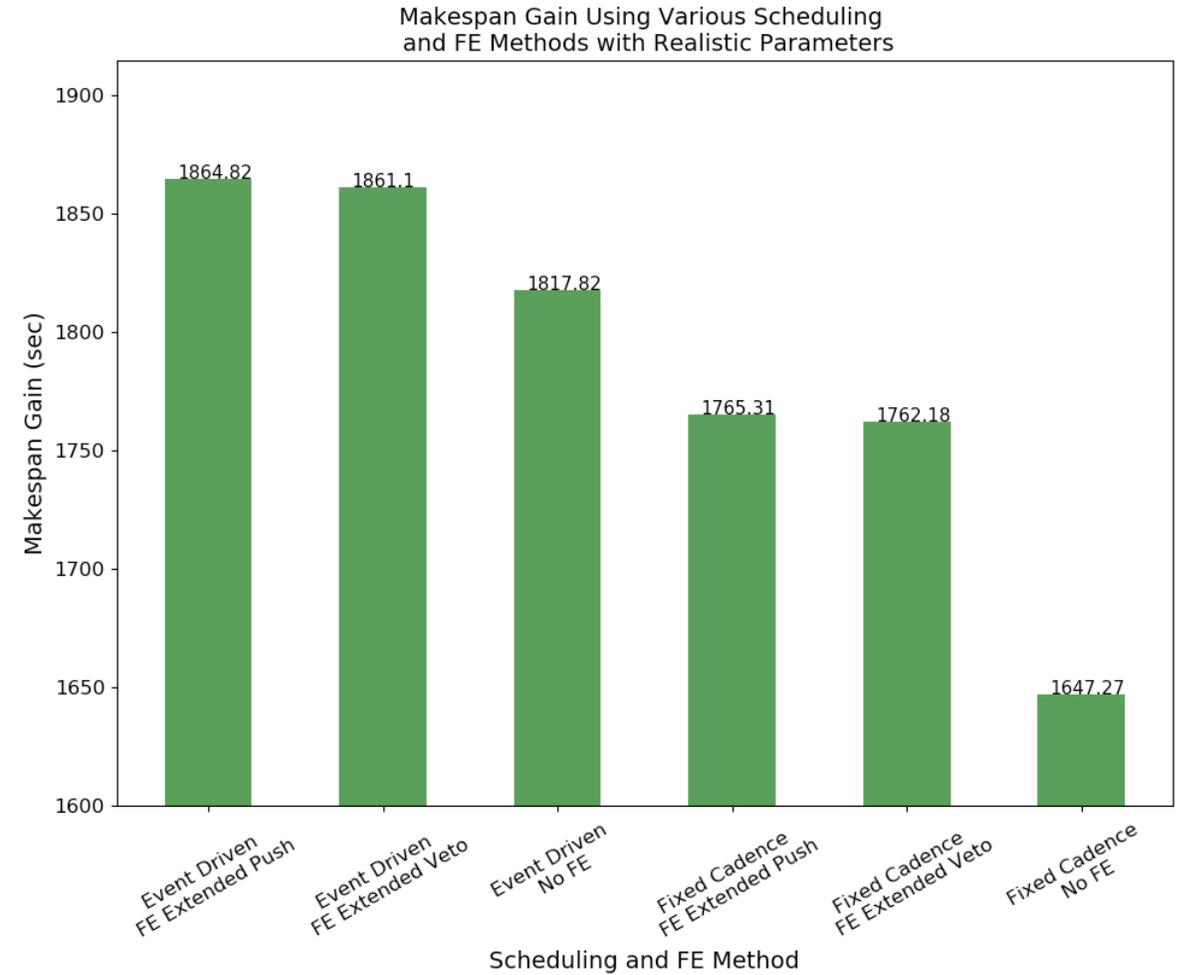
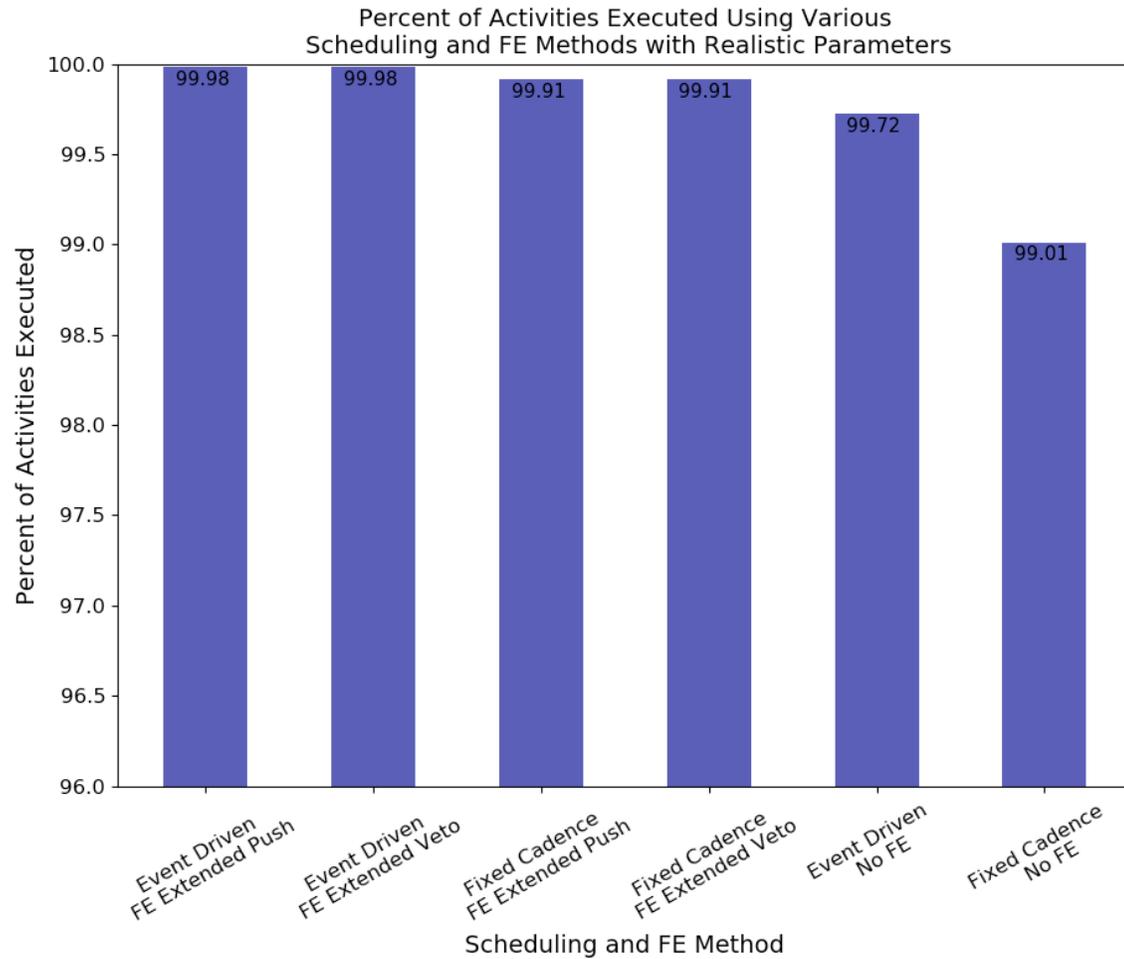


Percent of Activities Executed vs Percent of Activities that Ran Long

- Commit window = predicted scheduler runtime (Tsc_p) = dispatch window
- Scheduler takes as much time as expected to run
- **Activities run late by varying amounts**
- Event Driven with FE extended push then FE with extended veto perform best (fewest number of activities dropped)
- **Highest Curve: Event Driven with FE Extended Push**



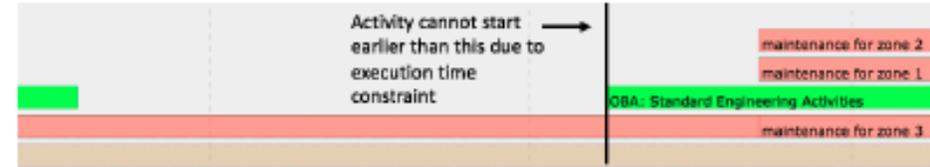
Results with Most Realistic Parameters



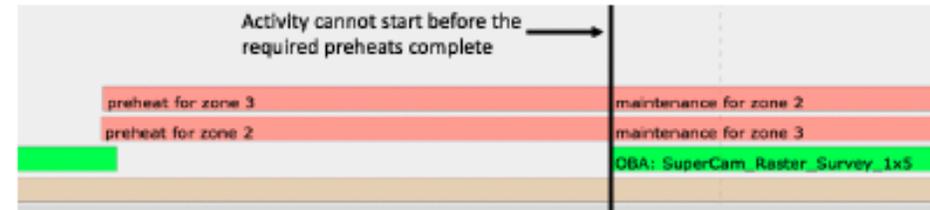
- The Predicted Scheduler Runtime, $Tsc_p = 60$ seconds
- On average, 5 percent of activities take longer than expected
- On average, activities take 72 percent of their original duration
- On average the scheduler takes half the predicted time to run. That is, Tsc_a (actual) = $0.5 \times Tsc_p$ (predicted)
- **Event Driven with extended push then extended veto perform best in terms of both number of activities executed and makespan gain**

Analysis of Computational Model for Loss

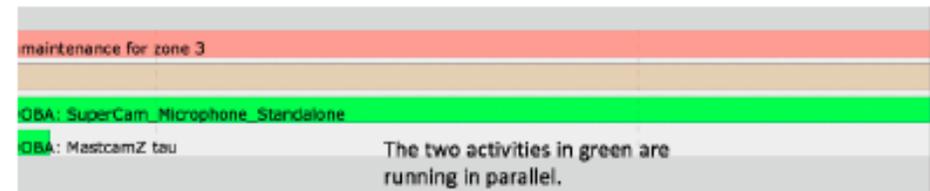
- An accurate model should result in:
 - **Theoretical Makespan Gain = Actual Makespan Gain + Scheduler Runtime Loss + Scheduler Invocation Loss**
 - Theoretical makespan gain is makespan gain using an instantaneous scheduler (0 sec commit window)
- Our model is inaccurate for following reasons:
 - 1) Execution Time Constraints
 - Any time before the earliest time an activity is allowed to start cannot be gained back
 - 2) Setup Activities
 - Activities may require setup activities such as preheats and are not allowed to start before such setup activities complete
 - 3) Parallelism
 - There may be activities not in critical path in a non-serial schedule
 - No matter how early they finish, they do not affect makespan



(a) In the Natural Proximity Science sol type, execution time constraints prevent activities (in green) from being able to start earlier.



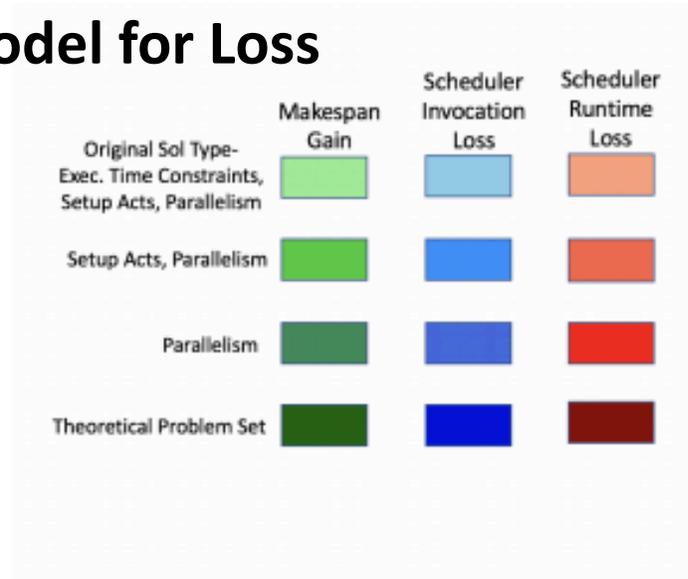
(b) In the Workspace sol type, preheats prevent activities (in green) from being able to start earlier.



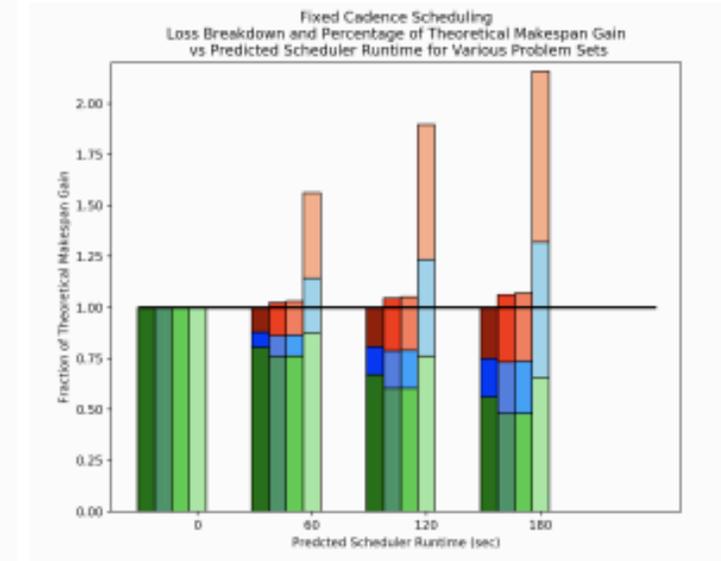
(c) The Abraded sol type is not fully serial and some activities (in green) run in parallel.

Results- Analysis of Computational Model for Loss

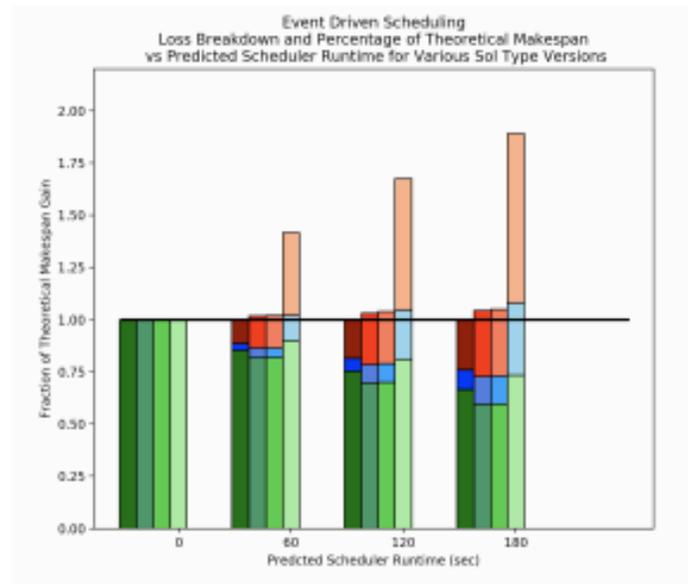
- **Scheduler runtime loss:** time that the scheduler is unable to recoup while the scheduler is predicted to be running
- **Scheduler invocation loss:** time lost due to waiting to reinvokethe scheduler
- Event Driven Scheduling, shown in Figure 16c, decreases the scheduler invocation and scheduler runtime loss compared to using Fixed Cadence Scheduling
- FE with Event Driven Scheduling, shown in Figure 16d further reduces the scheduler invocation loss and results in the highest makespan gain. The scheduler runtime loss increases slightly (FE may pull activities earlier and trigger events which would not have occurred otherwise)
- Theoretically, makespan gain + scheduler runtime loss + scheduler invocation loss = makespan gain with 0 sec commit window
 - Not true because of 1) exec. time constraint, 2) preheats/setup, 3) parallelism
- **Execution time constraints contribute most to model inaccuracy**



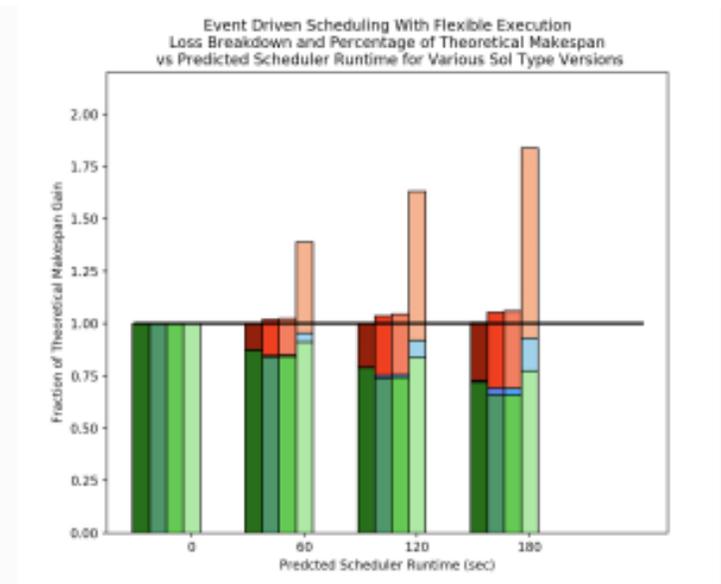
(a) Legend



(b) Fixed Cadence Scheduling



(c) Event Driven No FE



(d) Event Driven with FE

Future Work

- **Objective Function** – We focused on Makespan, but the goal is to maximize some utility function over executed activities.
 - May be range of activities in preference order
 - Execute most preferred activity if it does not cause any future activity to fail
 - Scheduling at a preferred time rather than earliest
 - Include energy/data volume in utility function
- Ensure scheduler isn't invoked too frequently (overconsumption of CPU) or infrequently (stale schedule)
 - Hybrid method- scheduler is invoked at least once every x minutes
 - Minimum time delay between invocations
- **Better Runtime Model** – Our simple probabilistic model varies only durations. Many other variables can be adjusted to more accurately depict runtime variations

Conclusion

- Event Driven scheduling outperforms Fixed Cadence scheduling in terms of decreasing loss
- Event Driven scheduling greatly decreases scheduler invocation loss
- FE decreases loss, and is more effective with higher scheduler runtimes
- Event Driven Scheduling with Extended Push results in highest average percentage of activities executed when activities take longer than expected
- If activities do not run long, Event Driven scheduling with either FE method results in highest makespan gain over varying values for predicted scheduler runtime
- For our specific inputs, execution time constraints, setup activities, and parallelism contribute to the imperfection of our computational model for loss and removing these factors from inputs results in an accurate model