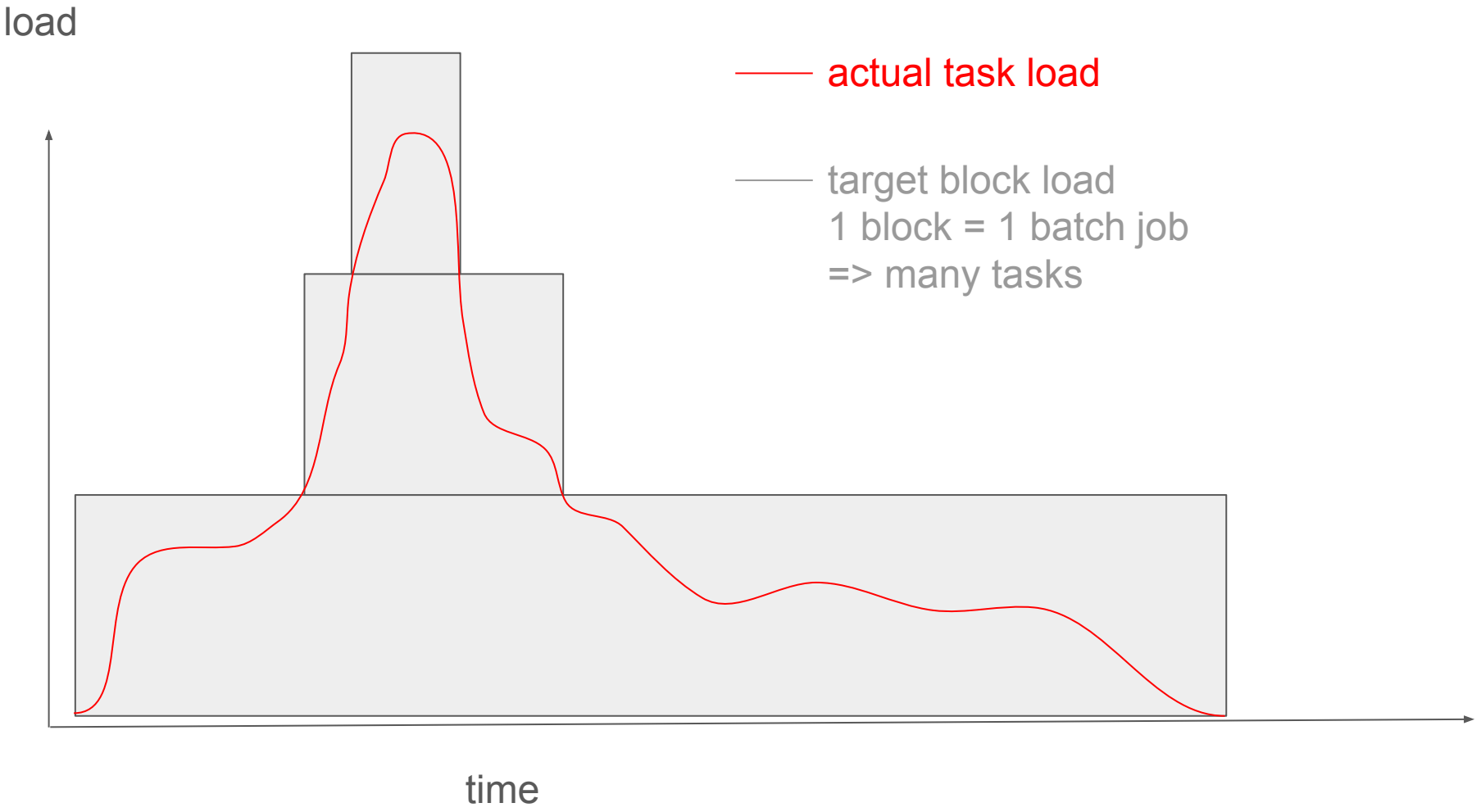
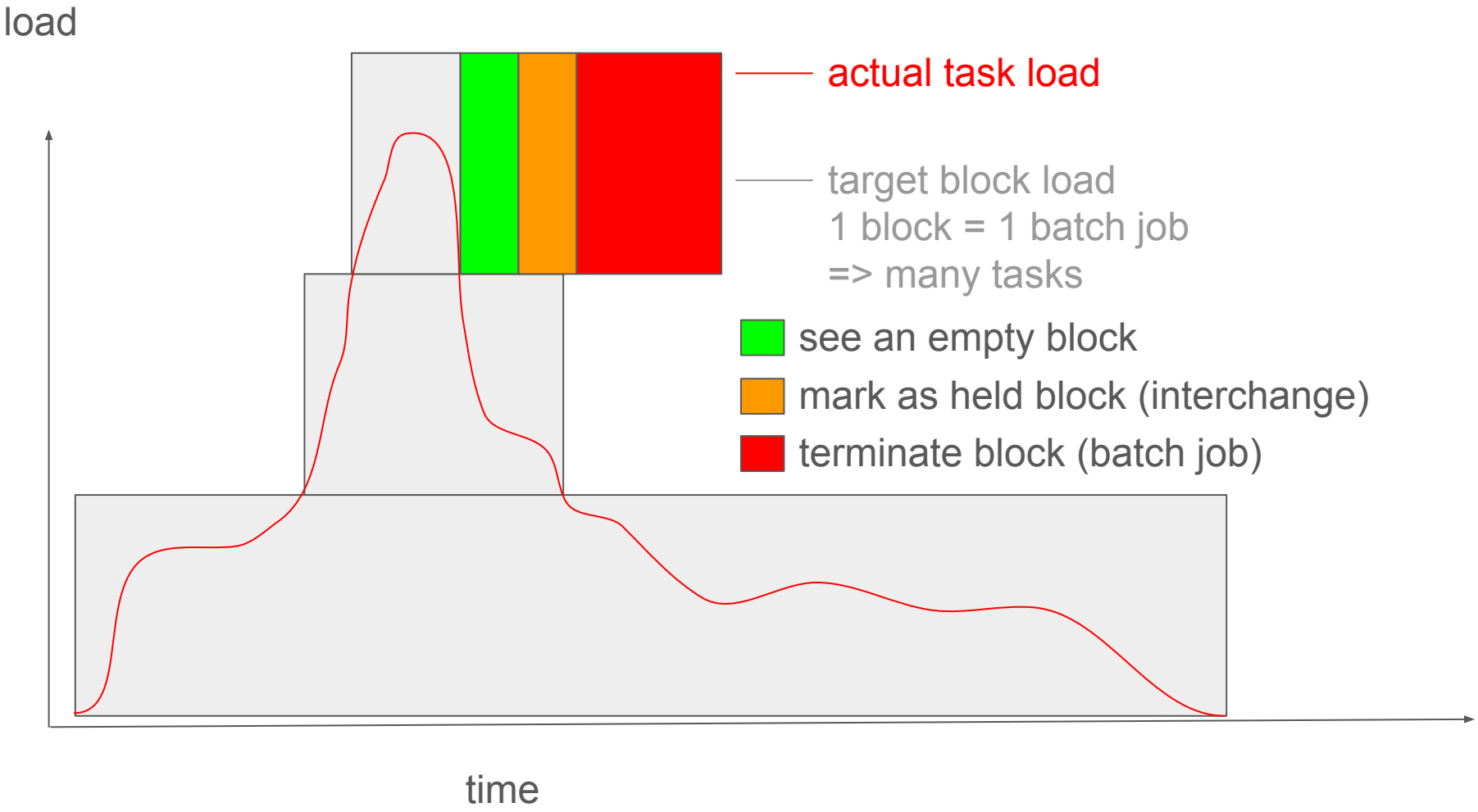
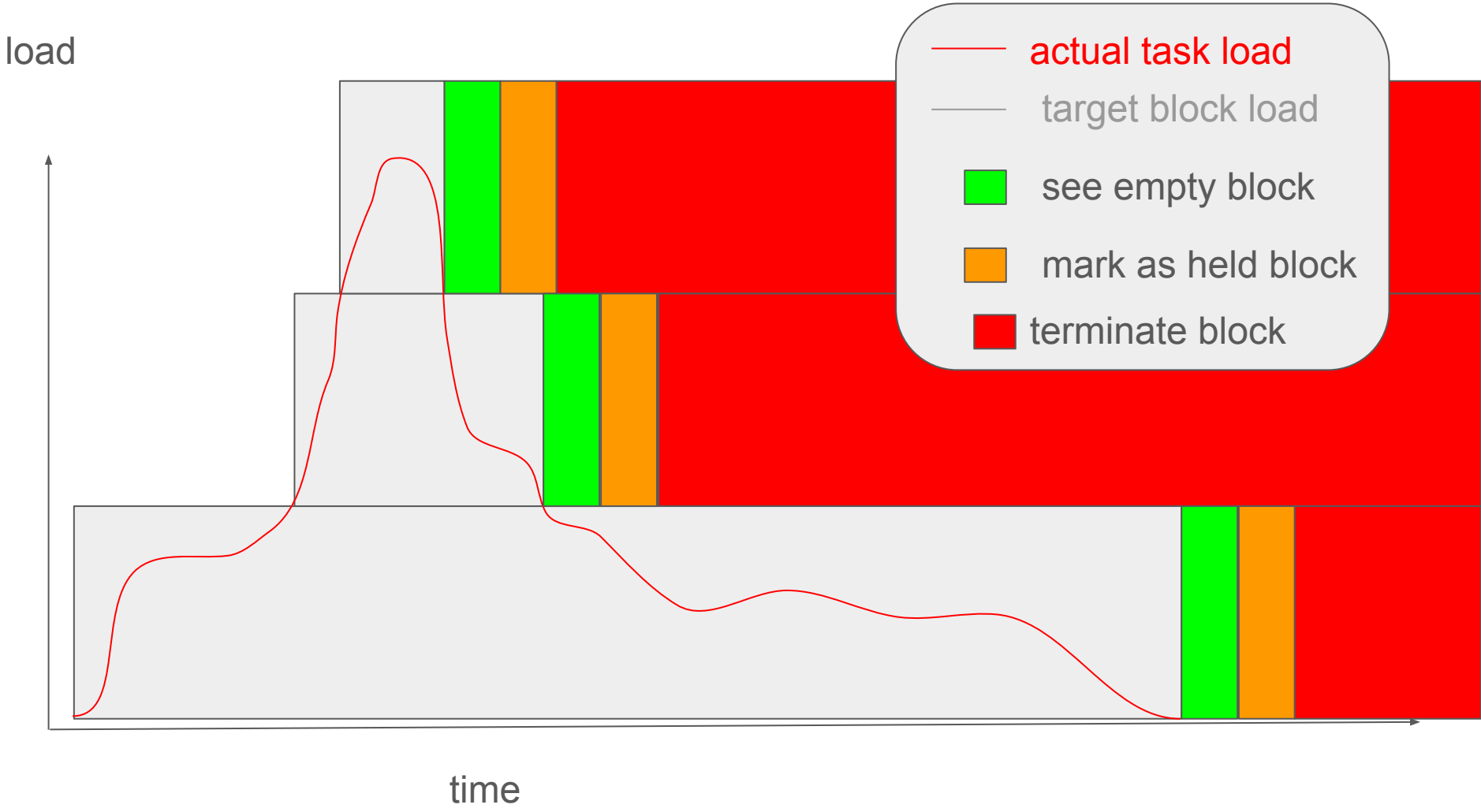
# Temporal Logic of Actions vs Lost Tasks vs Block Scaling

Ben Clifford <benc@hawaga.org.uk>
APeX 2026


load
actual task load
target block load
1 block = 1 batch job
=> many tasks
time


load
actual task load
target block load
1 block = 1 batch job
=> many tasks
see an empty block
mark as held block (interchange)
terminate block (batch job)
time


load
actual task load
target block load
see empty block
mark as held block
terminate block
time

# TLA+/PlusCal: task dispatch + scaling

```
fair process submit_side_submitter =
"SUBMIT_SIDE_SUBMITTER"
variables
 i = 1;
 t = 0;
begin
 SA:
 while i =< Len(InitialTasks)
 do
  t := InitialTasks[i];
  submitted_tasks := submitted_tasks + 1;
  executor_outstanding_tasks :=
               executor_outstanding_tasks + 1;
  TaskQueue := Append(TaskQueue, t);
 end while;
 SC:
 await Len(InitialTasks) = Len(CompletedTasks);
 SD:
 Shutdown := TRUE;
end process;
```

+ SUBMIT_SIDE_COMPLETOR
+ INTERCHANGE
+ MANAGER

```
fair process submit_side_scaler = "SUBMIT_SIDE_SCALER"
begin
 SCALEL:
 while ~Shutdown do
  if executor_outstanding_tasks > 0 then
   ManagerHeld := FALSE;
   ManagerKilled := FALSE;
  else
   SCALEKILLA:
   ManagerHeld := TRUE;
   SCALEKILLB:
   ManagerKilled := TRUE;
  end if;
 end while;
 SCALEK:
 ManagerKilled := TRUE;
end process;
```

```
TasksCompleted == <>(CompletedTasks = InitialTasks /\
executor_outstanding_tasks = 0)

ProcessesCompleted == <>(/\ pc["SUBMIT_SIDE_SUBMITTER"] = "Done"
/\ pc["SUBMIT_SIDE_COMPLETOR"] = "Done" /\
pc["SUBMIT_SIDE_SCALER"] = "Done" /\ pc["INTERCHANGE"] = "Done"
/\ pc["MANAGER"] = "Done")
```

```
Progress(45) at 2026-09-11 11:59:09: 11,889 states generated, 3,277 distinct states found, 0 states left on queue.
```

**Error: Temporal properties TasksCompleted and ProcessesCompleted were violated.**

```
Error: The following behavior constitutes a counter-example:

State 1: <Initial predicate>

/\ executor_outstanding_tasks = 0
/\ Shutdown = FALSE
/\ task = defaultInitValue
/\ ResultsManagerToInterchangeQueue = <<>>
/\ manager_task = defaultInitValue
/\ CompletedTasks = <<>>
/\ InitialTasks = <<100, 101, 102>>
/\ TasksSubmitToInterchangeQueue = <<>>
/\ TasksInterchangeToManagerQueue = <<>>
/\ x = 0
/\ ManagerKilled = TRUE
/\ ResultsInterchangeToSubmitQueue = <<>>
/\ pc = [SUBMIT_SIDE_SUBMITTER |-> "SUBMIT_LOOP", SUBMIT_SIDE_COMPLETOR |-> "COMPLETOR_LOOP", SUBMIT_SIDE_SCALER |->
"SCALER_LOOP", INTERCHANGE |-> "INTERCHANGE_LOOP", MANAGER |-> "MANAGER_START"]
/\ ManagerHeld = FALSE
/\ submitted_tasks = 1
```

```
State 2: <SUBMIT_LOOP>
/\ executor_outstanding_tasks = 1
/\ TasksSubmitToInterchangeQueue = <<100>>

State 3: <SUBMIT_LOOP>
/\ executor_outstanding_tasks = 2
/\ TasksSubmitToInterchangeQueue = <<100, 101>>

State 4: <MANAGER_START>
/\ ManagerKilled = TRUE

State 5: <SCALER_LOOP>
/\ executor_outstanding_tasks = 2
/\ ManagerKilled = FALSE

State 6: <INTERCHANGE_LOOP>
State 7: <INTERCHANGE_TASK_GET>
State 8: <INTERCHANGE_TASK_PUT>
/\ TasksSubmitToInterchangeQueue = <<101>>
/\ TasksInterchangeToManagerQueue = <<100>>

State 9: <MANAGER_UNSTARTED>
/\ ManagerKilled = FALSE

State 10: <MANAGER_TASK_GET>
/\ manager_task = 100

State 11: <MANAGER_RESULT_PUT>
State 12: <INTERCHANGE_RESULT_GET>
State 13: <INTERCHANGE_RESULT_PUT>
State 14: <INTERCHANGE_LOOP>
State 15: <INTERCHANGE_TASK_GET>
State 16: <INTERCHANGE_TASK_PUT>

State 17: <COMPLETOR_LOOP>
/\ executor_outstanding_tasks = 1
/\ CompletedTasks = <<100>>

State 18: <MANAGER_TASK_GET>
State 19: <MANAGER_RESULT_PUT>
State 20: <INTERCHANGE_RESULT_GET>
State 21: <INTERCHANGE_RESULT_PUT>
State 22: <INTERCHANGE_LOOP>

State 23: <COMPLETOR_LOOP>
/\ executor_outstanding_tasks = 0
/\ CompletedTasks = <<100, 101>>
```

```
State 23: <COMPLETOR_LOOP>
/\ executor_outstanding_tasks = 0
/\ CompletedTasks = <<100, 101>>

State 24: <SCALER_LOOP>
/\ executor_outstanding_tasks = 0

State 25: <SUBMIT_LOOP>
/\ executor_outstanding_tasks = 1
/\ TasksSubmitToInterchangeQueue = <<102>>

State 26: <SUBMIT_LOOP>
State 27: <INTERCHANGE_TASK_GET>

State 28: <SCALER_HOLD>
/\ executor_outstanding_tasks = 1
/\ ManagerHeld = TRUE
/\ ManagerKilled = FALSE
```

```
State 29: <INTERCHANGE_TASK_PUT>
State 30: <MANAGER_TASK_GET>

State 31: <SCALER_KILL>
/\ executor_outstanding_tasks = 1
/\ ManagerKilled = TRUE

State 32: <MANAGER_RESULT_PUT>
/\ ManagerKilled = TRUE

State 33: <SCALER_LOOP>
/\ executor_outstanding_tasks = 1
/\ ManagerKilled = FALSE

State 34: <INTERCHANGE_RESULT_GET>
State 35: <INTERCHANGE_LOOP>
Back to state 33: <INTERCHANGE_TASK_GET>
```

```
State 23: <COMPLETOR_LOOP>
/\ executor_outstanding_tasks = 0
/\ CompletedTasks = <<100, 101>>

State 24: <SCALER_LOOP>  SCALAR COMITTED TO KILL BLOCK
/\ executor_outstanding_tasks = 0    BY IF STATEMENT

State 25: <SUBMIT_LOOP>
/\ executor_outstanding_tasks = 1
/\ TasksSubmitToInterchangeQueue = <<102>>

State 26: <SUBMIT_LOOP>
State 27: <INTERCHANGE_TASK_GET>
  TASK DISPATCH COMMITTED TO BEFORE HOLD HAPPENS

State 28: <SCALER_HOLD>         HOLD HAPPENS TOO LATE
/\ executor_outstanding_tasks = 1
/\ ManagerHeld = TRUE
/\ ManagerKilled = FALSE
```

```
State 29: <INTERCHANGE_TASK_PUT>
State 30: <MANAGER_TASK_GET>
         MANAGER GETS THE TASK BUT ...

State 31: <SCALER_KILL>
/\ executor_outstanding_tasks = 1
/\ ManagerKilled = TRUE     GOODBYE MANAGER, GOODBYE TASK

State 32: <MANAGER_RESULT_PUT>
/\ ManagerKilled = TRUE

State 33: <SCALER_LOOP>
/\ executor_outstanding_tasks = 1
/\ ManagerKilled = FALSE

State 34: <INTERCHANGE_RESULT_GET>
State 35: <INTERCHANGE_LOOP>
Back to state 33: <INTERCHANGE_TASK_GET>
```

# A fix

- Interchange reports empty block
- HOLD_MANAGER commands => interchange
- **Wait until interchange reports DRAINED block, not empty block**
- Terminate block

Manager draining already exists - for walltime expiry avoidance (similar problem)
PR #3065

# Why is this more likely now?

- early-era parsl scaling in code:
  - only after entire executor idle for 2 minutes - unlikely a new task arrives at exact wrong moment

- HTEX scaling strategy - PR #1563
  - allows scaling in empty blocks when other blocks not empty
- BlockIdManagerSelector - PR #3560
  - helps blocks empty out faster

- more likely for task dispatch to be happening when a block is scaled in

# Why is this more likely in the immediate future?

- Control channel changes will change the timing of empty block info and HOLD_MANAGER commands
- Timing changes tickle race conditions

# Future

imagine: type-checking on temporal steroids

Plenty of concurrency in all of parsl/globus compute/academy

- plenty of scope for this kind of analysis

How can I be confident that the Python code implements the algorithms I analysed here? ideally in CI / automatically [spoiler: its very very hard]

How can I be confident that the algorithms/specification is actually what I want?

This is not a technical question?