Time-Bounded Analysis of Real-Time Systems

Sagar Chaki[1], Arie Gurfinkel[1],
Ofer Strichman[2], Soonho Kong[1]

[1]Software Engineering Institute, CMU
[2]Technion, Israel Institute of Technology

Software Engineering Institute (SEI)

Department of Defense R&D Laboratory (FFRDC)

Created in 1984

Under contract to Carnegie Mellon University

Offices in Pittsburgh, PA; Washington, DC; and Frankfurt, Germany

SEI Mission: advance software engineering and related disciplines to ensure the development and operation of systems with predictable and improved cost, schedule, and quality.

Motivation: Real-Time Embedded Systems

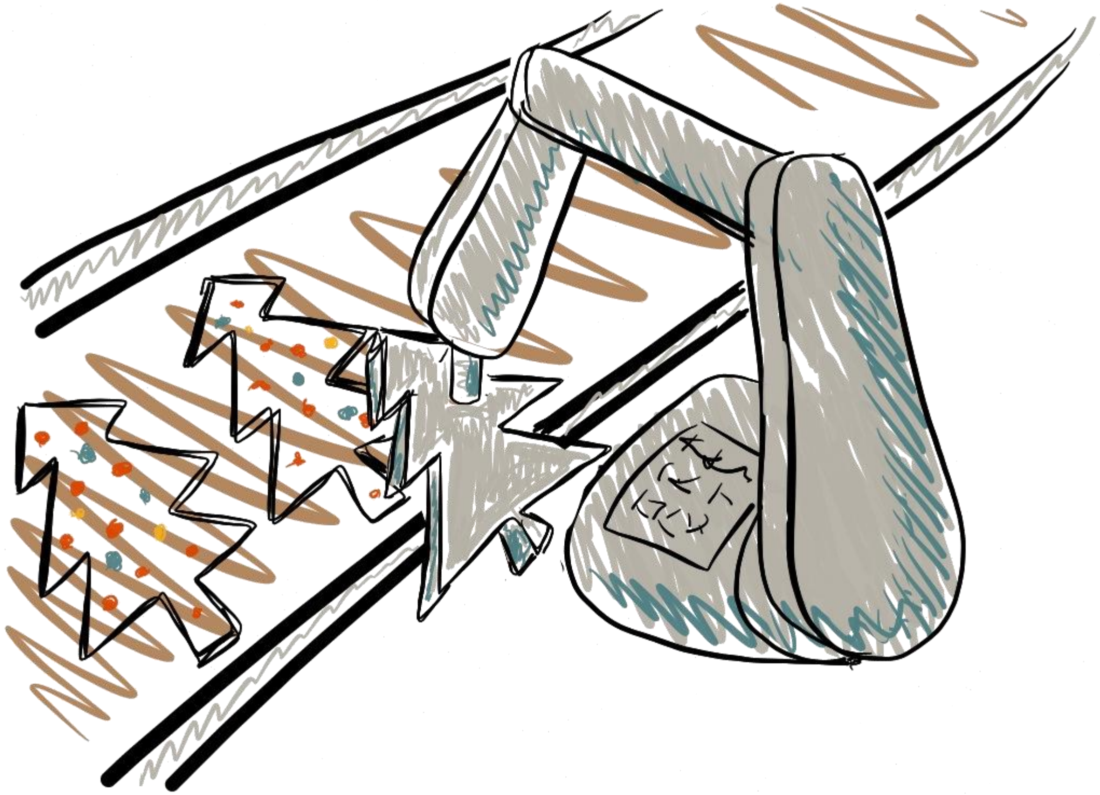
Avionics Mission System*

Rate Monotonic Scheduling (RMS)

Task	Period
weapon release	10ms
radar tracking	40ms
target tracking	40ms
aircraft flight data	50ms
display	50ms
steering	80ms

*Locke, Vogel, Lucas, and Goodenough. "Generic Avionics Software Specification". SEI/CMU Technical Report CMU/SEI-90-TR-8-ESD-TR-90-209, December, 1990

Case Study: A Metal Stamping Robot

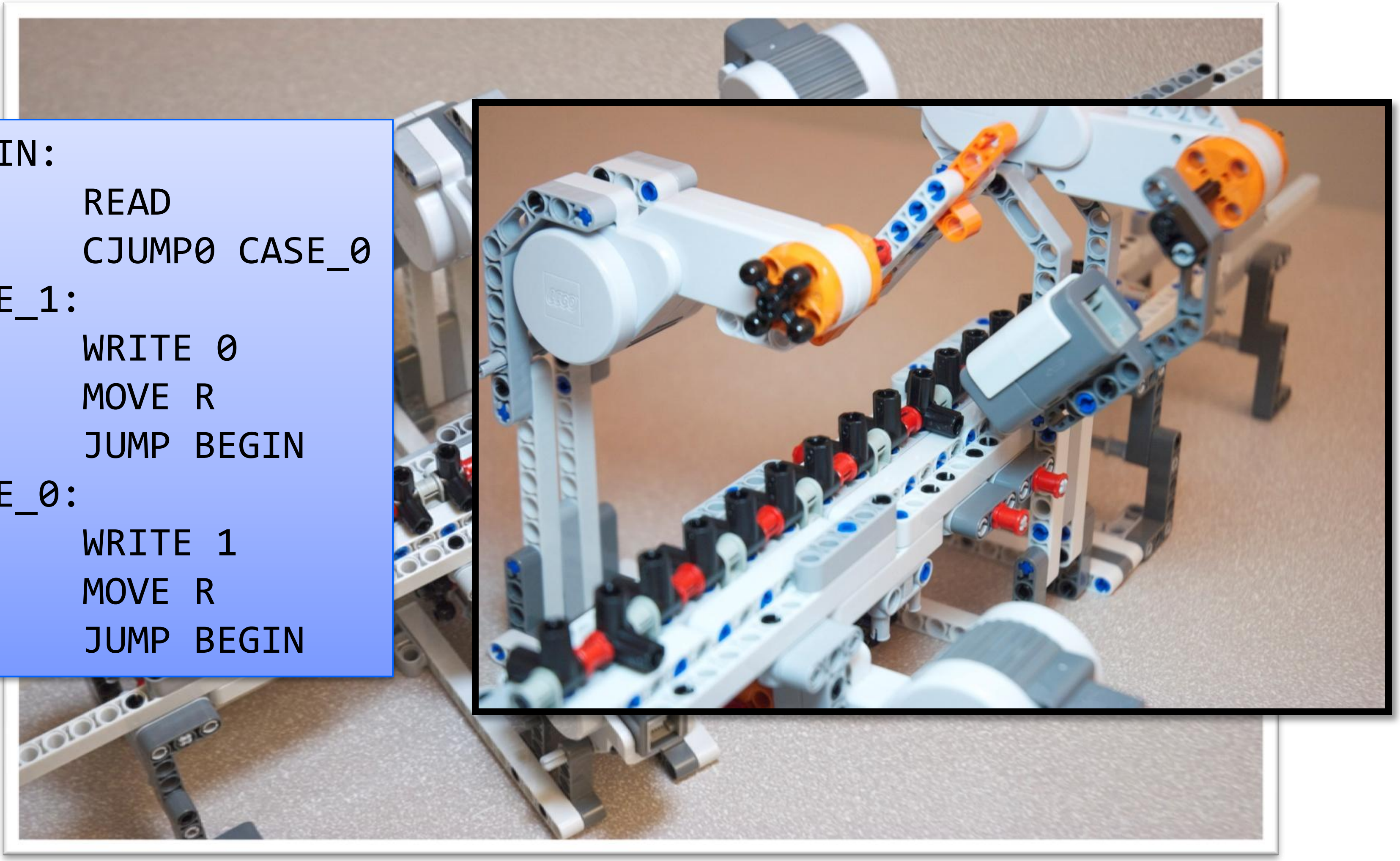


a.k.a. LEGO Turing Machine

Image courtesy of Taras Kowaliw

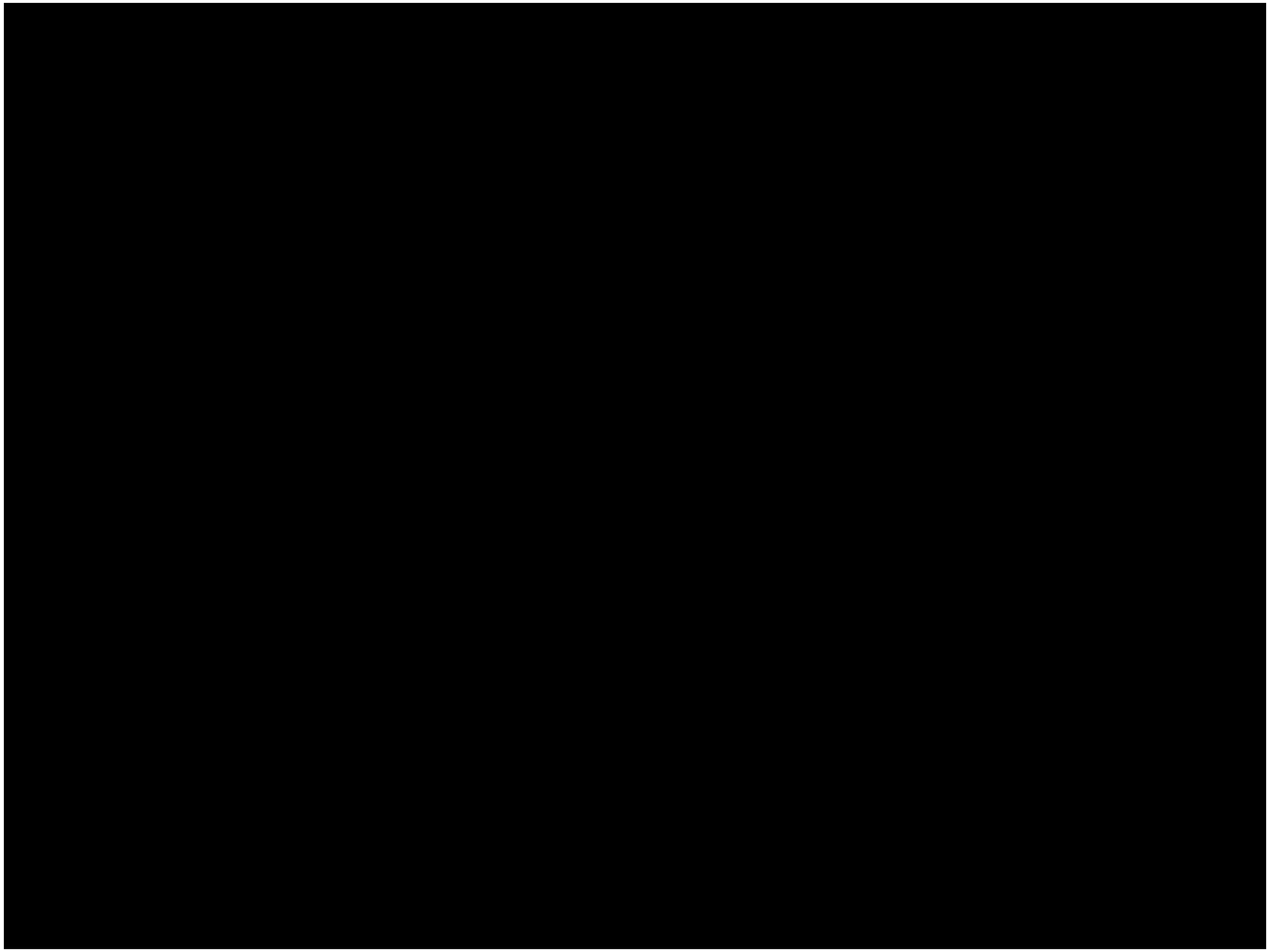
LEGO Turing Machine

```
BEGIN:
      READ
      CJUMP0 CASE_0
CASE_1:
      WRITE 0
      MOVE R
      JUMP BEGIN
CASE_0:
      WRITE 1
      MOVE R
      JUMP BEGIN
```



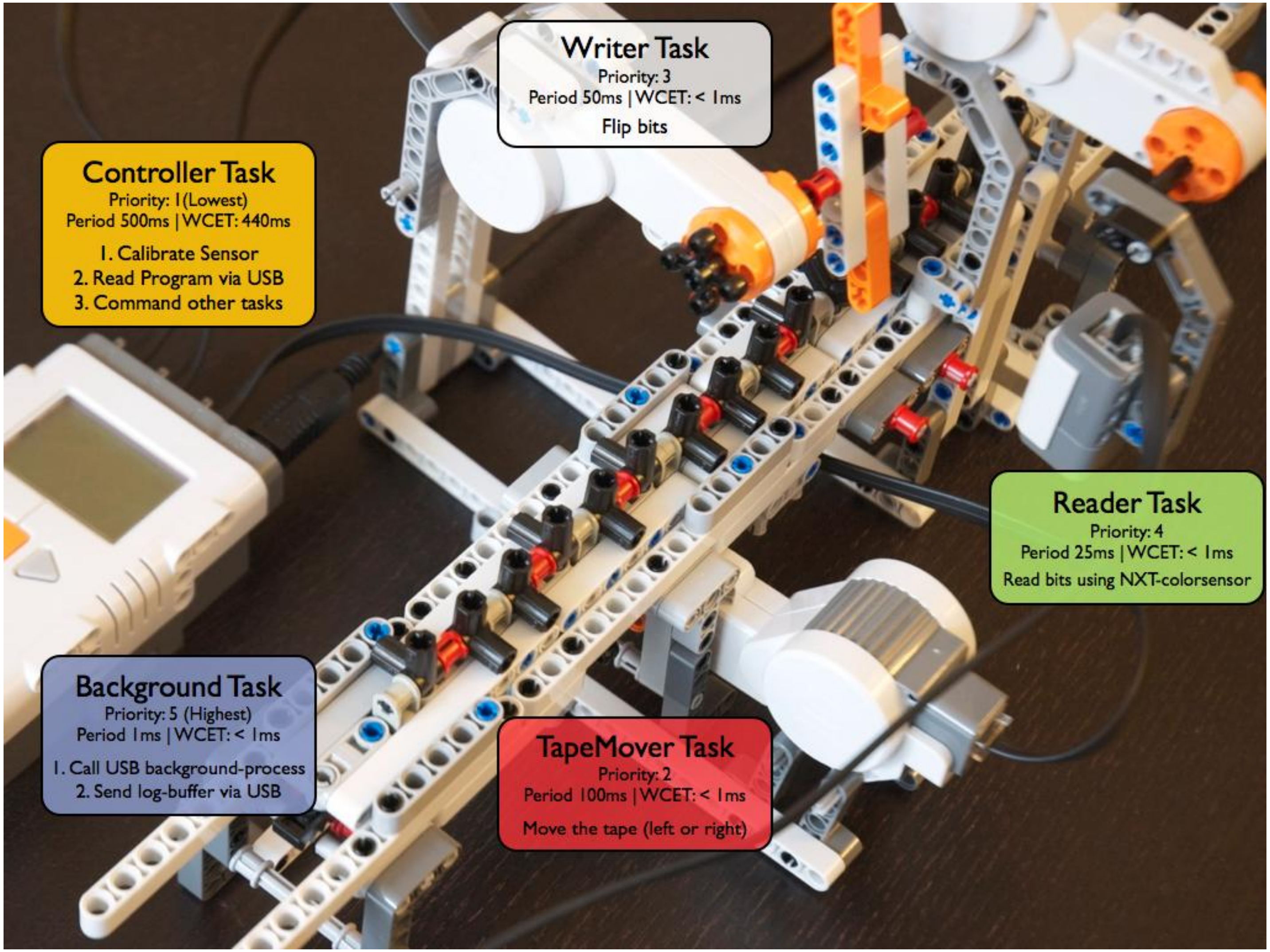
by Soonho Kong. See http://www.cs.cmu.edu/~soonhok for building instructions.

Turing Machine (Video)



http://www.youtube.com/watch?v=teDyd0d5M4o

Turing Machine: Task Structure



Turing Machine: Properties

Tape does not move when a bit is read or written

Read sensor and Write arm can move concurrently but must not interfere with one another

Read sensor's light is off when not in use

Read task WCET is less than 25ms

- reduced to checking API usage rules

No log messages are lost during USB communication

- each message is delivered to the server before a new one is produced

Time-Bounded Verification of Periodic Programs

Time-Bounded Verification

- Is an assertion A violated within X milliseconds of a system's execution from initial state I
  - A, X , I are user specified

Periodic Program

- Collection of periodic tasks
  - Execute concurrently with fixed-priority scheduling
  - Priorities respect RMS
  - Communicate through shared memory
  - Synchronize through preemption and priority ceiling locks

Assumptions

- System is schedulable
- WCET of each task is given

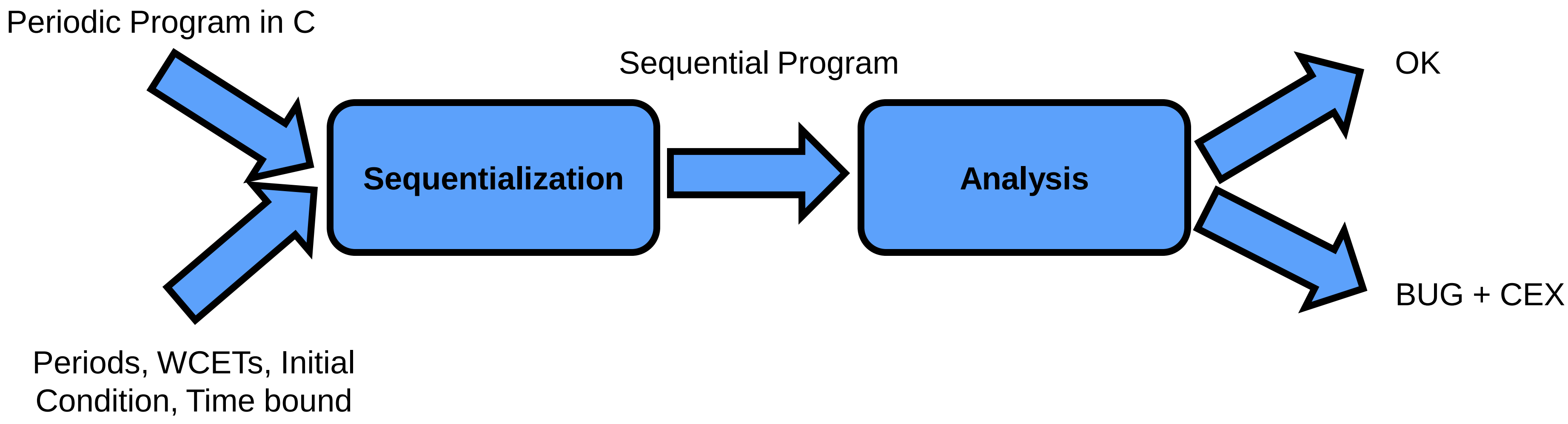
Time-Bounded Analysis of Real-Time Systems, Proc. of FMCAD 2011

Overall Approach

Supports C programs w/ tasks, priorities, priority ceiling protocol, shared variables

Works in two stages:

1. *Sequentialization* – reduction to sequential program w/ *prophecy* variables
2. *Bounded program analysis*: bounded C model checker (CBMC, HAVOC, …)



Periodic Program

An N-task periodic program PP is a set of tasks $\{\tau_1, \ldots, \tau_N\}$

A task τ is a tuple $\langle I, T, P, C, A \rangle$, where

- I is a task identifier
- T is a task body (i.e., code)
- P is a period
- C is the worst-case execution time
- A is the *release time:* the time at which task becomes first enabled

Semantics of PP is given by an asynchronous concurrent program:

parallel execution w/ priorities

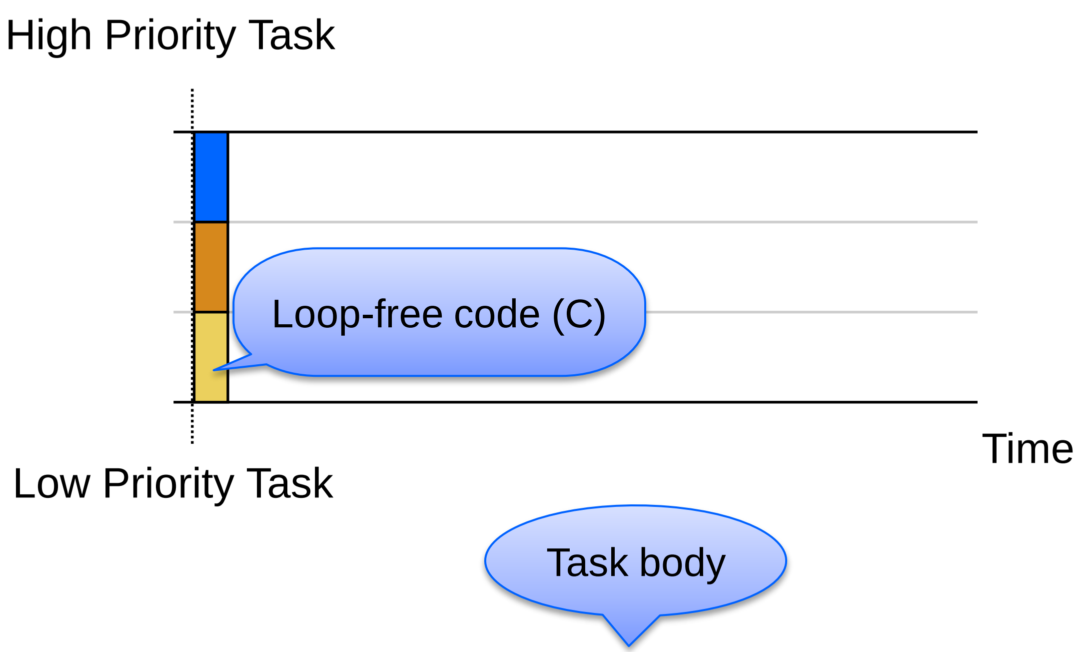
$$\Big\|_i \quad \begin{array}{l} k_i = 0; \\ \texttt{while (Wait(}\tau_i, k_i\texttt{))} \\ \quad T_i\ (); \\ \quad k_i = k_i + 1; \end{array}$$

blocks task *i* until next arrival time

Periodic Programs

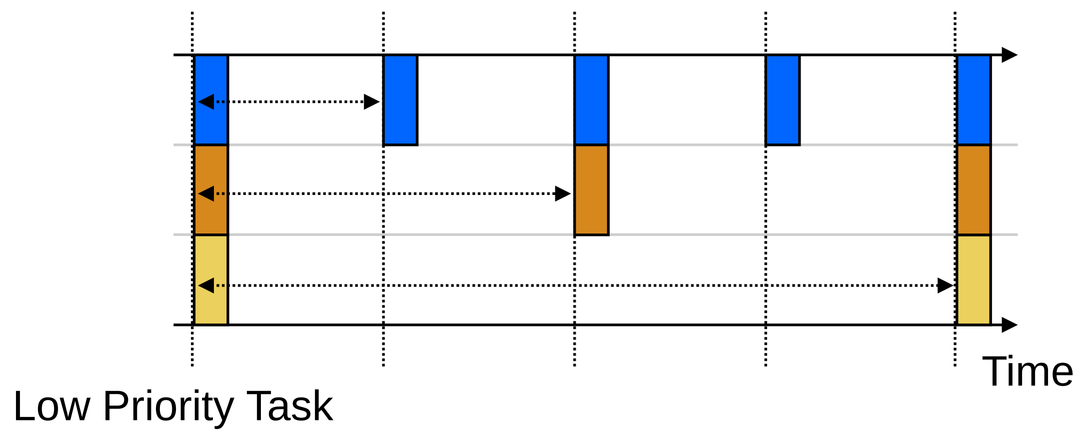
Task $\tau = (I, T, P, C, A)$

Periodic Programs



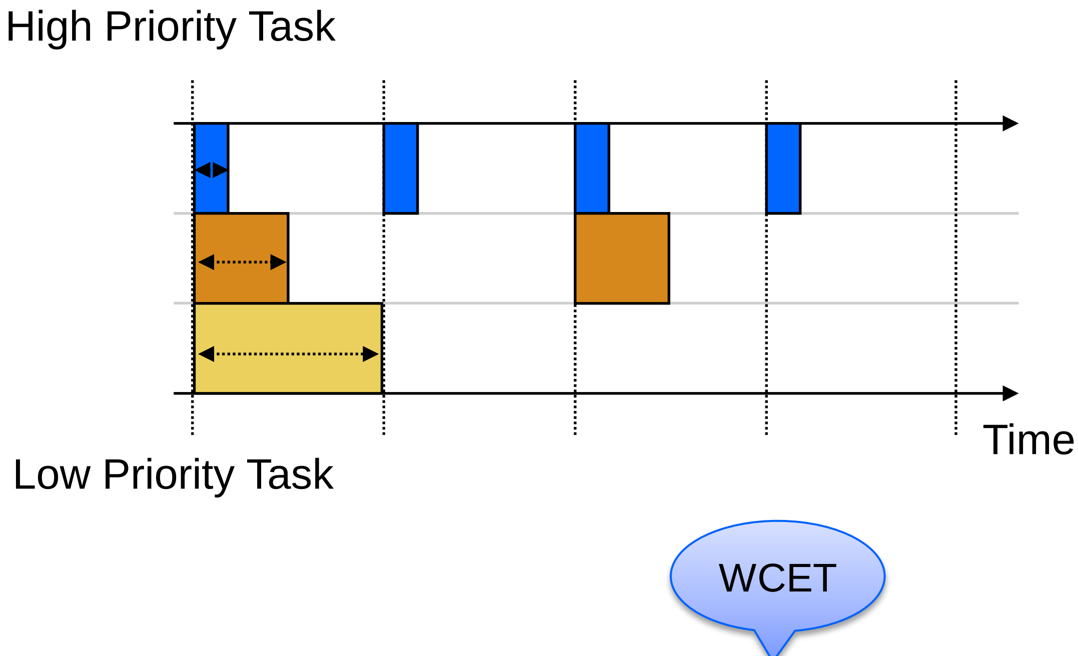
Task $\tau = (I, T, P, C, A)$

Periodic Programs



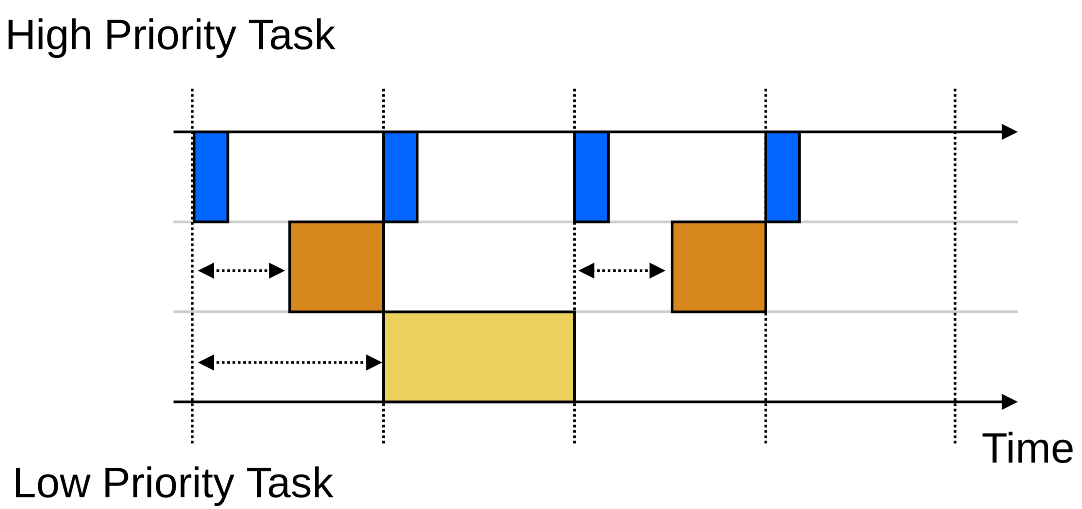
$$\text{Task}\ \tau = (I, T, P, C, A)$$

Periodic Programs



Task $\tau = (I, T, P, C, A)$

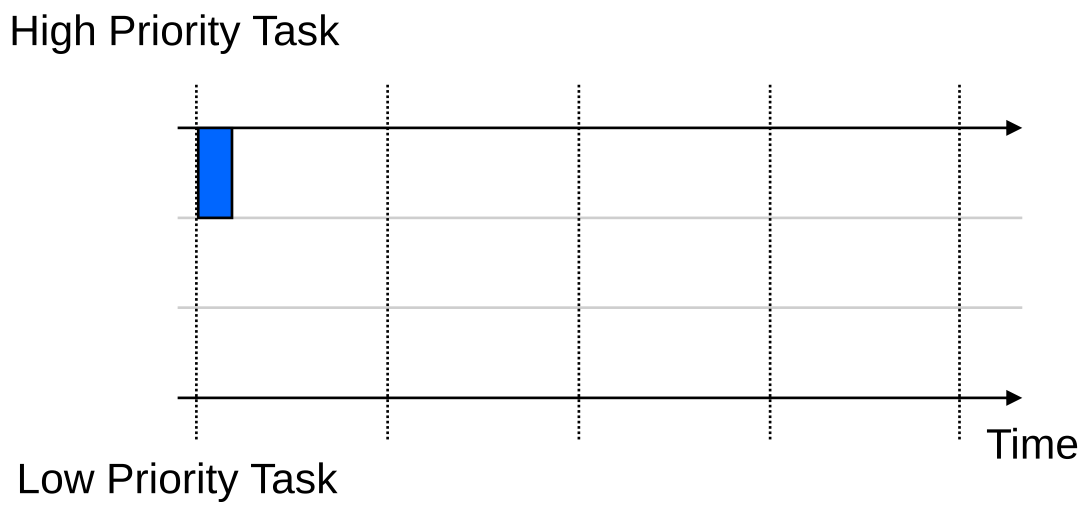
Periodic Programs



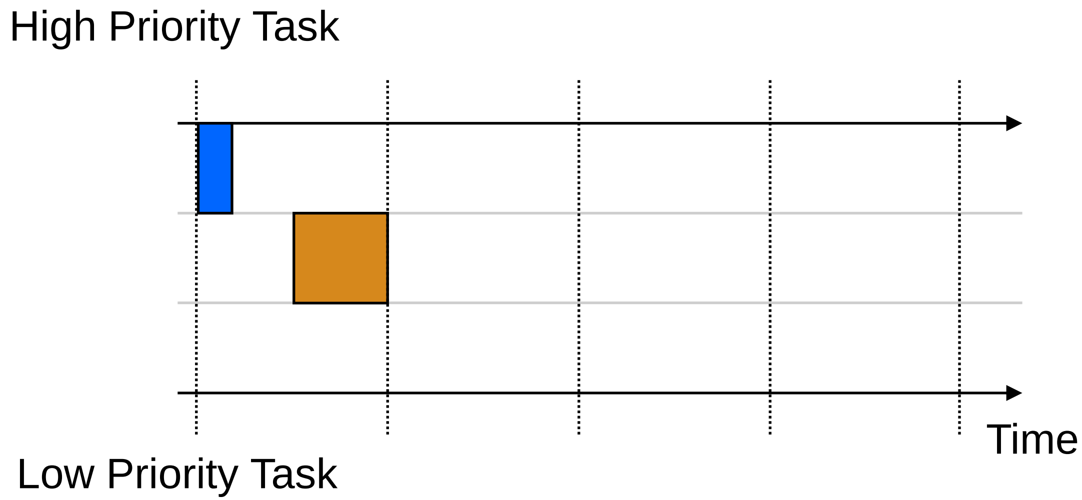
Arrival Time

Task $\tau = (I, T, P, C, A)$

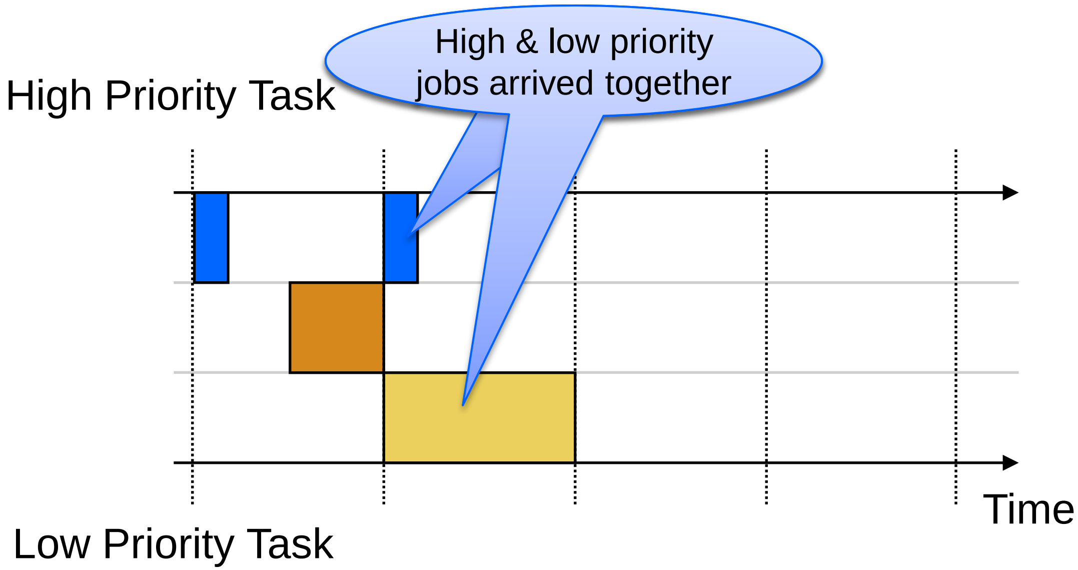
Preemptive Fixed Priority Scheduling



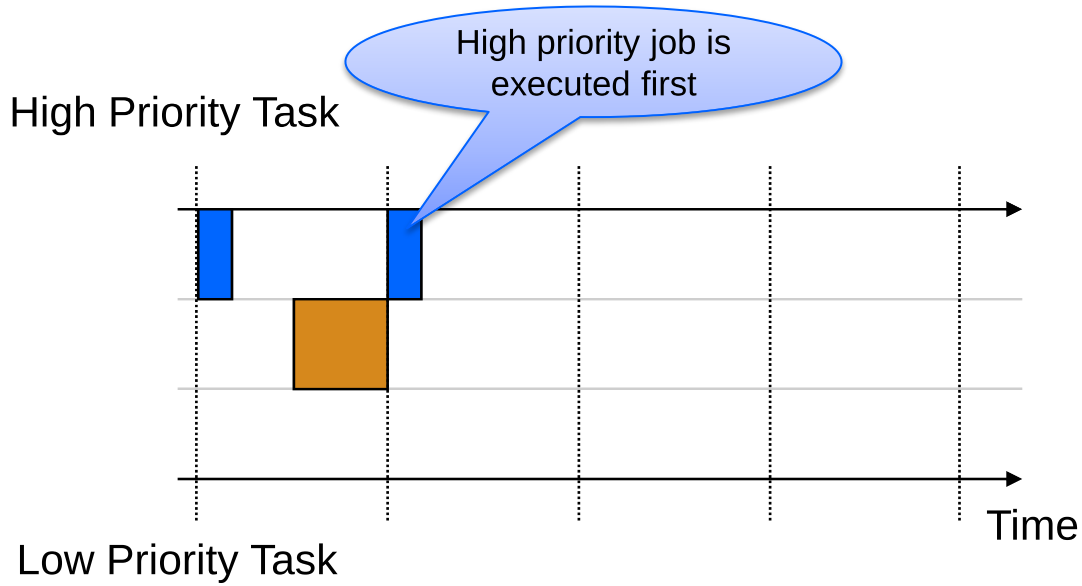
Preemptive Fixed Priority Scheduling



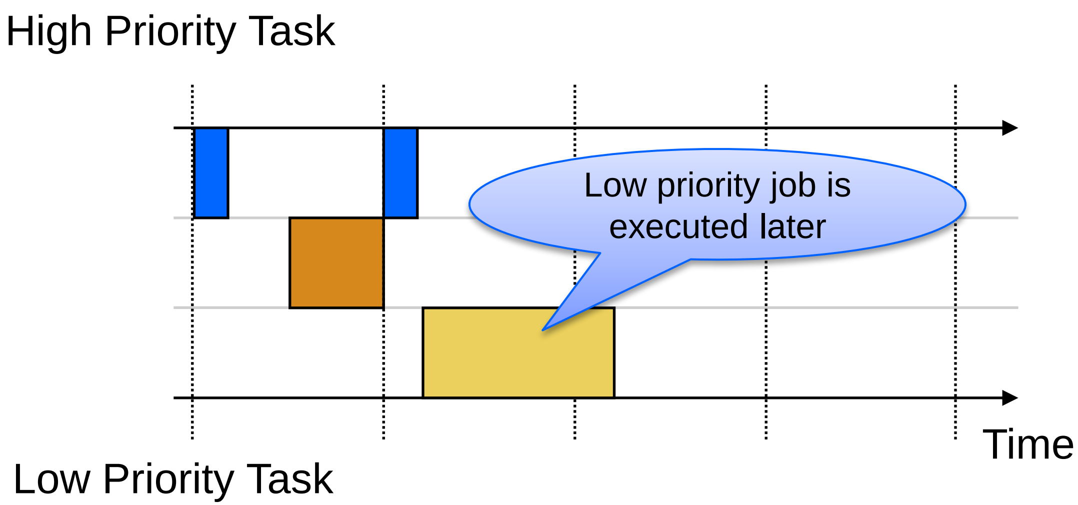
Preemptive Fixed Priority Scheduling



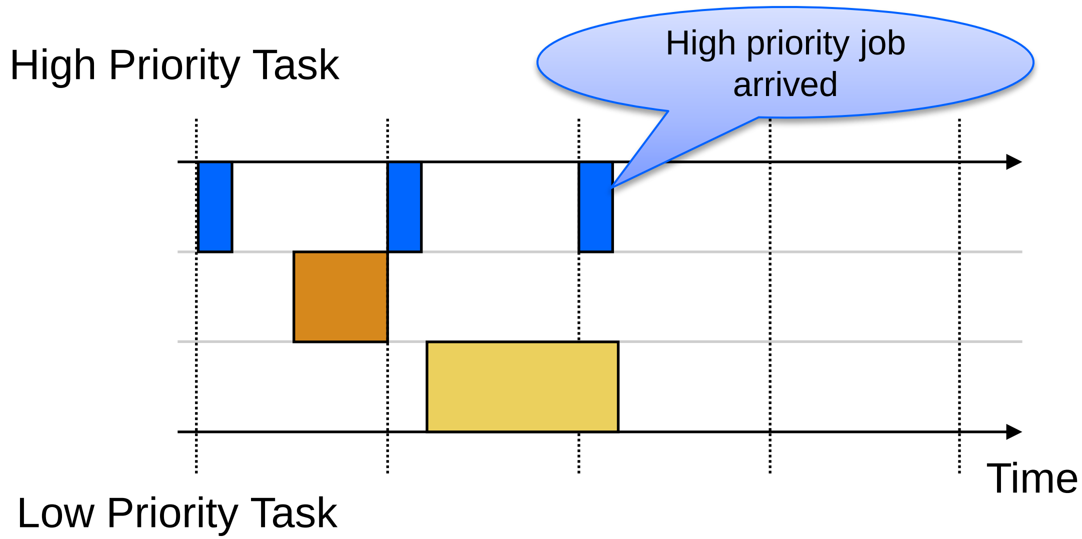
Preemptive Fixed Priority Scheduling



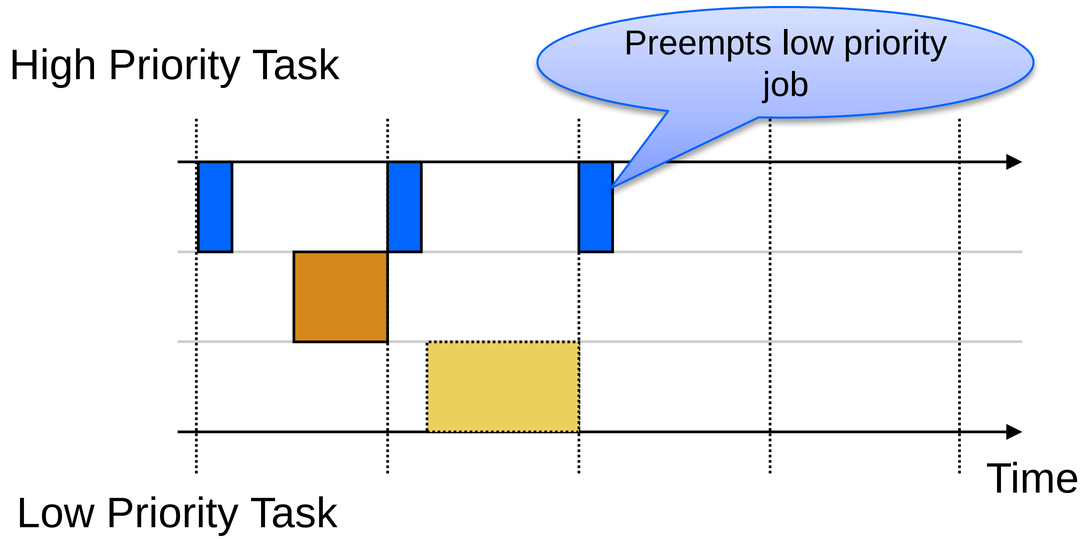
Preemptive Fixed Priority Scheduling



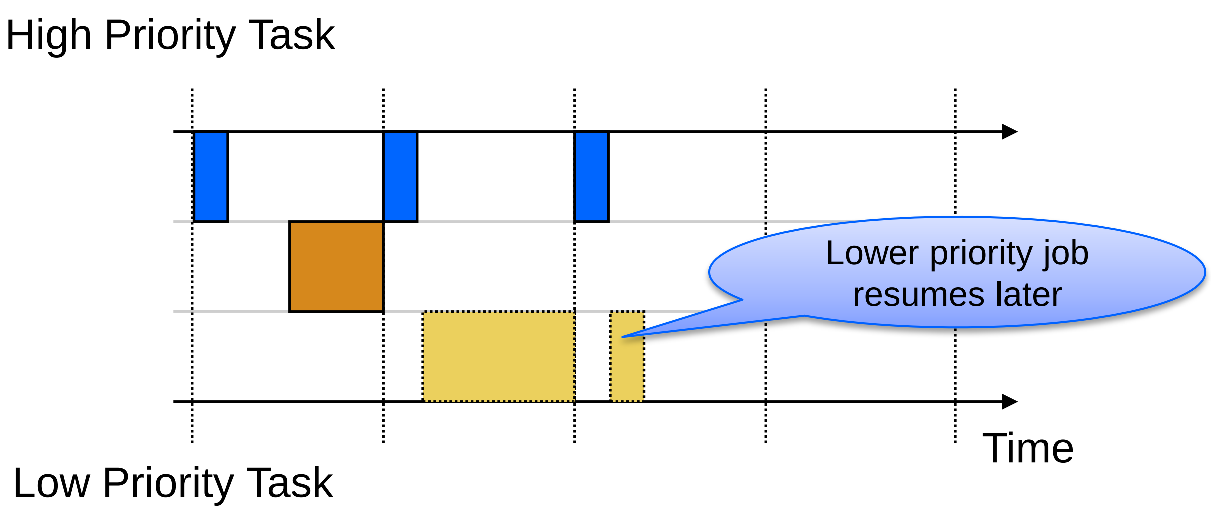
Preemptive Fixed Priority Scheduling



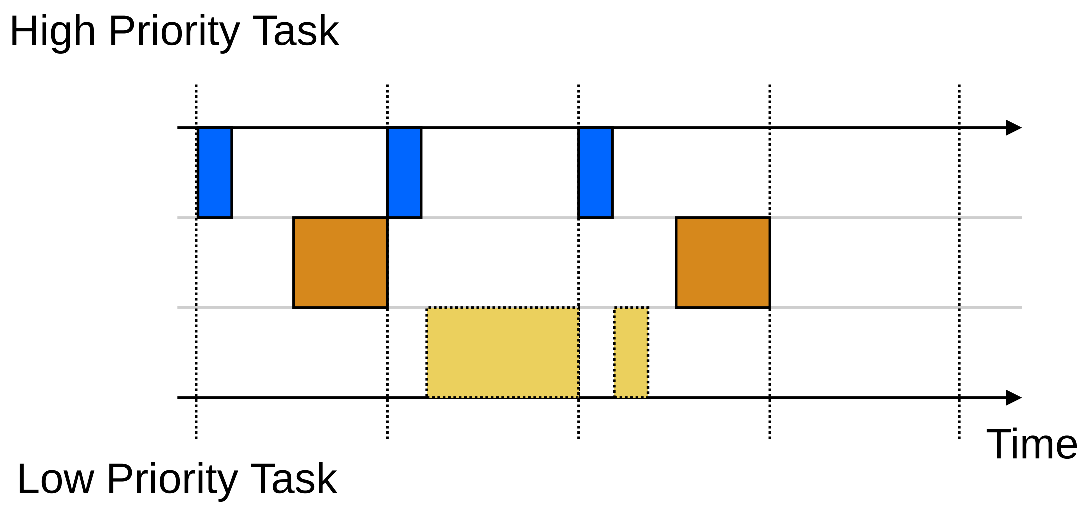
Preemptive Fixed Priority Scheduling



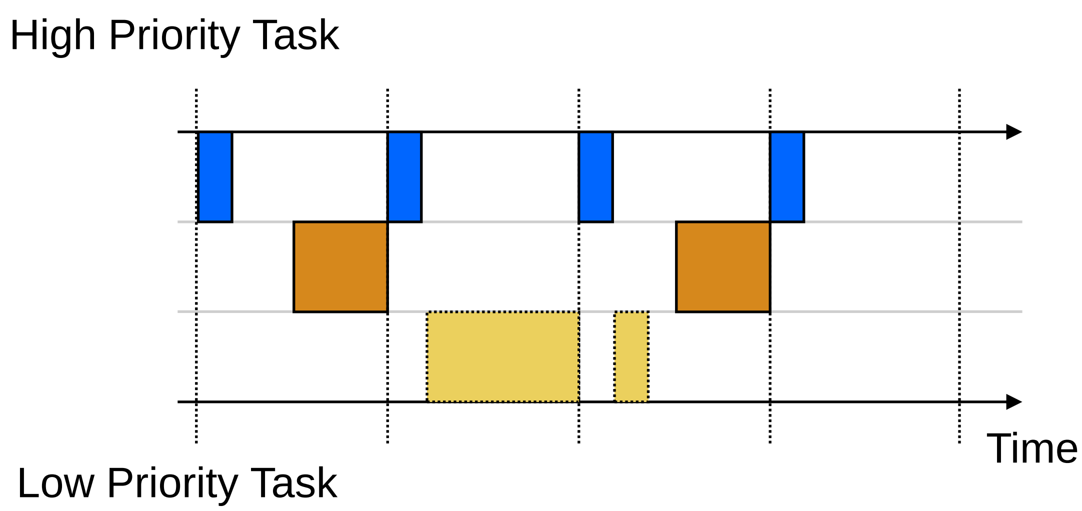
Preemptive Fixed Priority Scheduling



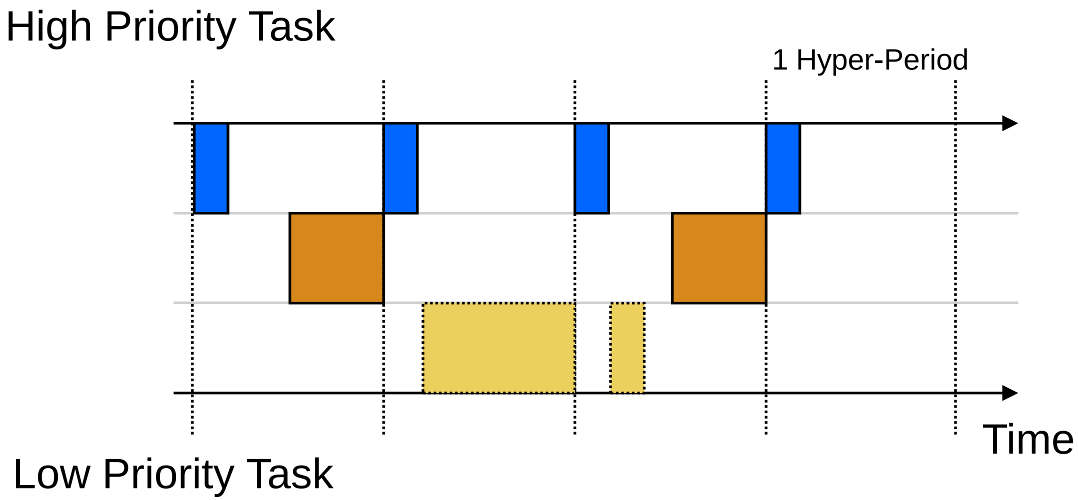
Preemptive Fixed Priority Scheduling



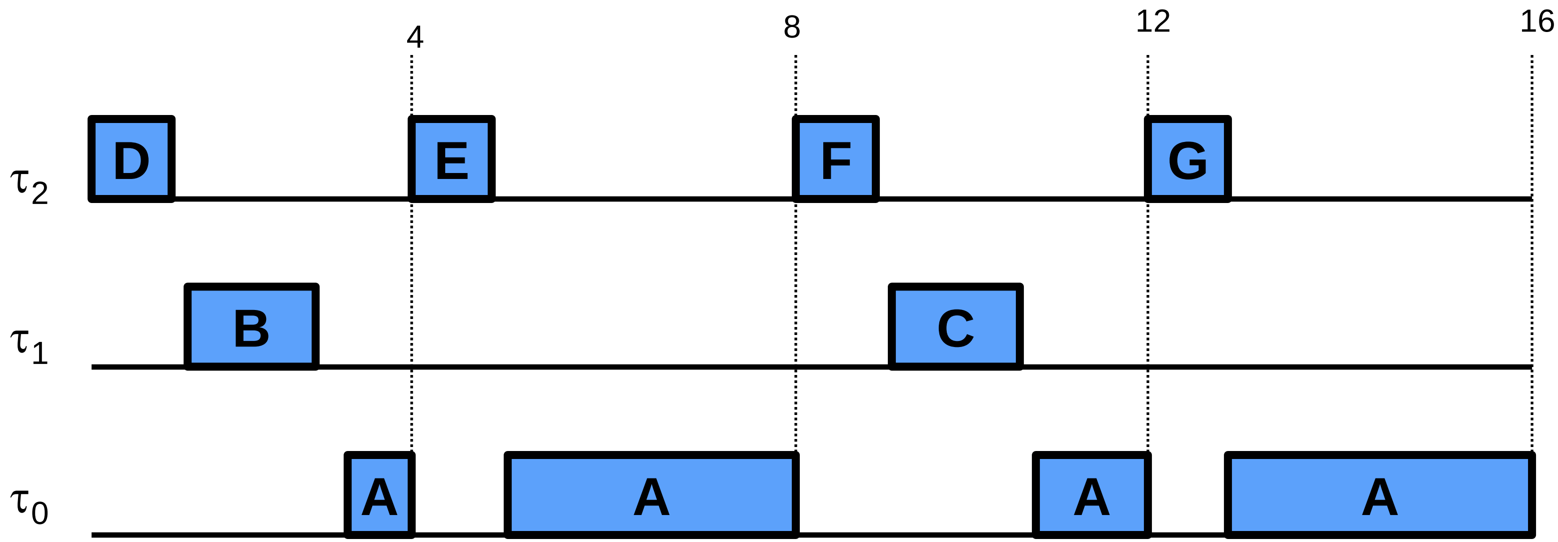
Preemptive Fixed Priority Scheduling



Preemptive Fixed Priority Scheduling



Example: Task Schedule



Task	WCET (C_i)	Period (P_i)	Arrival Time (A_i)	Response Time (RT_i)
τ_2	1	4	0	1
τ_1	2	8	0	3
τ_0	8	16	0	16

Maximum difference between arrival time and completion time of a job

Computed via Rate Monotonic Analysis

Time Bounded Semantics of Periodic Program

Assumptions

- Time window W is divisible by the period of each task (i.e., $W \mid P_i$)
- Each task arrives in time to complete in 1st period (i.e., $A_i + RT_i \le P_i$)

The time bound imposes a natural bound on # of jobs: $J_i = W / P_i$

Time-Bounded Semantics of PP is

$$\Big\|$$

```
k_i = 0;
while (k_i < J_i && Wait(τ_i, k_i))
  T_i ();
  k_i = k_i + 1;
```

Job-Bounded Abstraction

- Abstracts away time
- Approximates Wait() by a non-deterministic delay
- Preserves logical (time-independent) properties!

DEMO

C as a Modeling Language

Extend C programming language with 3 modeling features

Assertions

- assert(e) – aborts an execution when e is false, no-op otherwise

```
void assert (_Bool b) { if (!b)  exit(); }
```

Non-determinism

- nondet_int() – returns a non-deterministic integer value

```
int nondet_int () { int x; return x; }
```

Assumptions

- assume(e) – "ignores" execution when e is false, no-op otherwise

```
void assume (_Bool e) { while (!e) ;  }
```

Example of Using Assume/Nondet/Assert

```
int x, y;

void main (void)
{
  x = nondet_int ();

  assume (x > 10);
  y = x + 1;

  assert (y > x);

}
```

Example: Modeling with Prophecy Variables

```
int x, y, v;

void main (void)
{
  v = nondet_int ();
  x = v;

  x = x + 1;
  y = nondet_int ();
  assume (v == y);


}
```

v is a *prophecy* variable

it guesses the future value of y

syntactically: x is changed *before* y

semantically: x *depends on* y

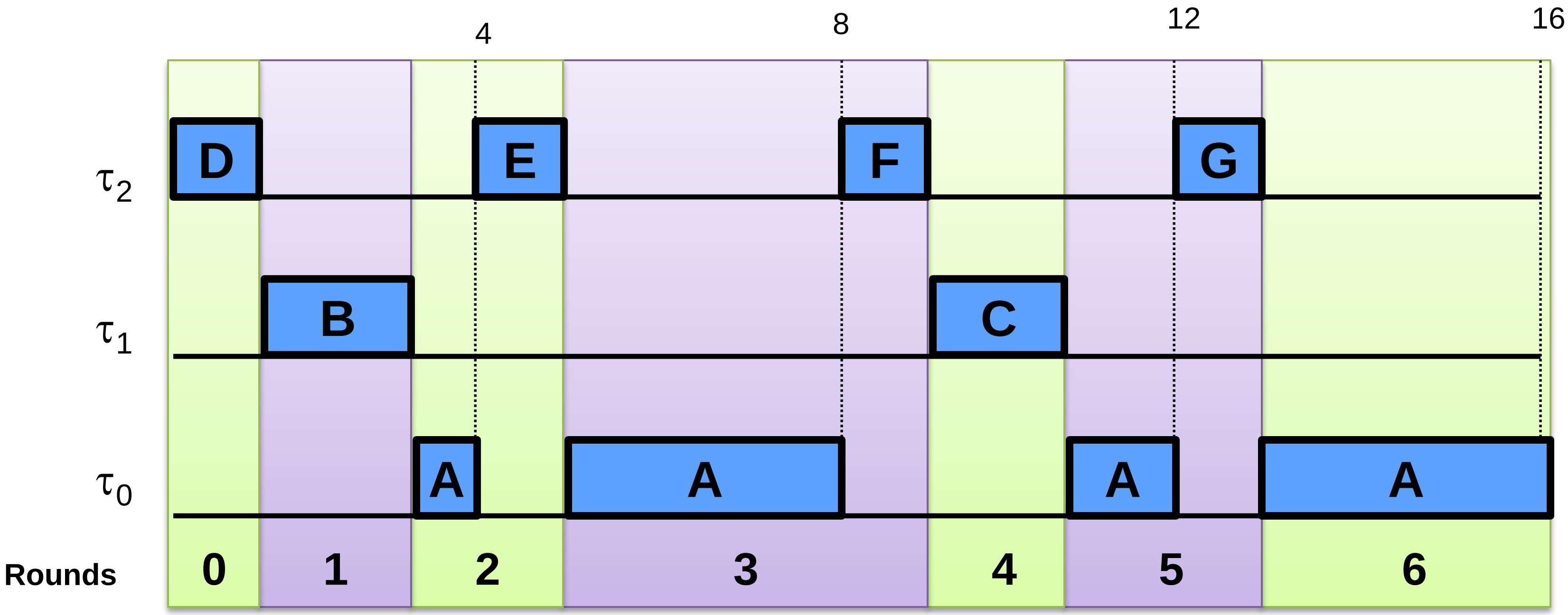
assume blocks executions with a wrong guess

Partition Execution into Rounds

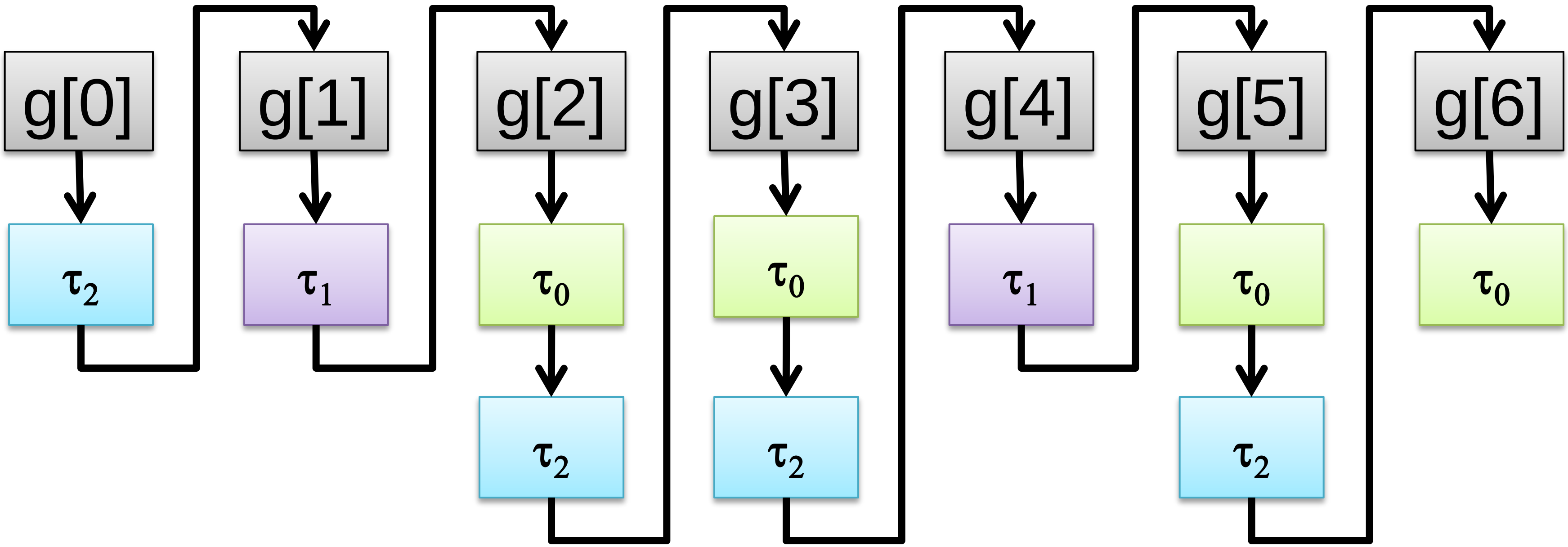
Execution starts in round 0

A round ends, and a new one begins, each time a job finishes

- # rounds == # of jobs



Sequentialization: Visually



Guess initial value of each global in each round (g[0] … g[6])

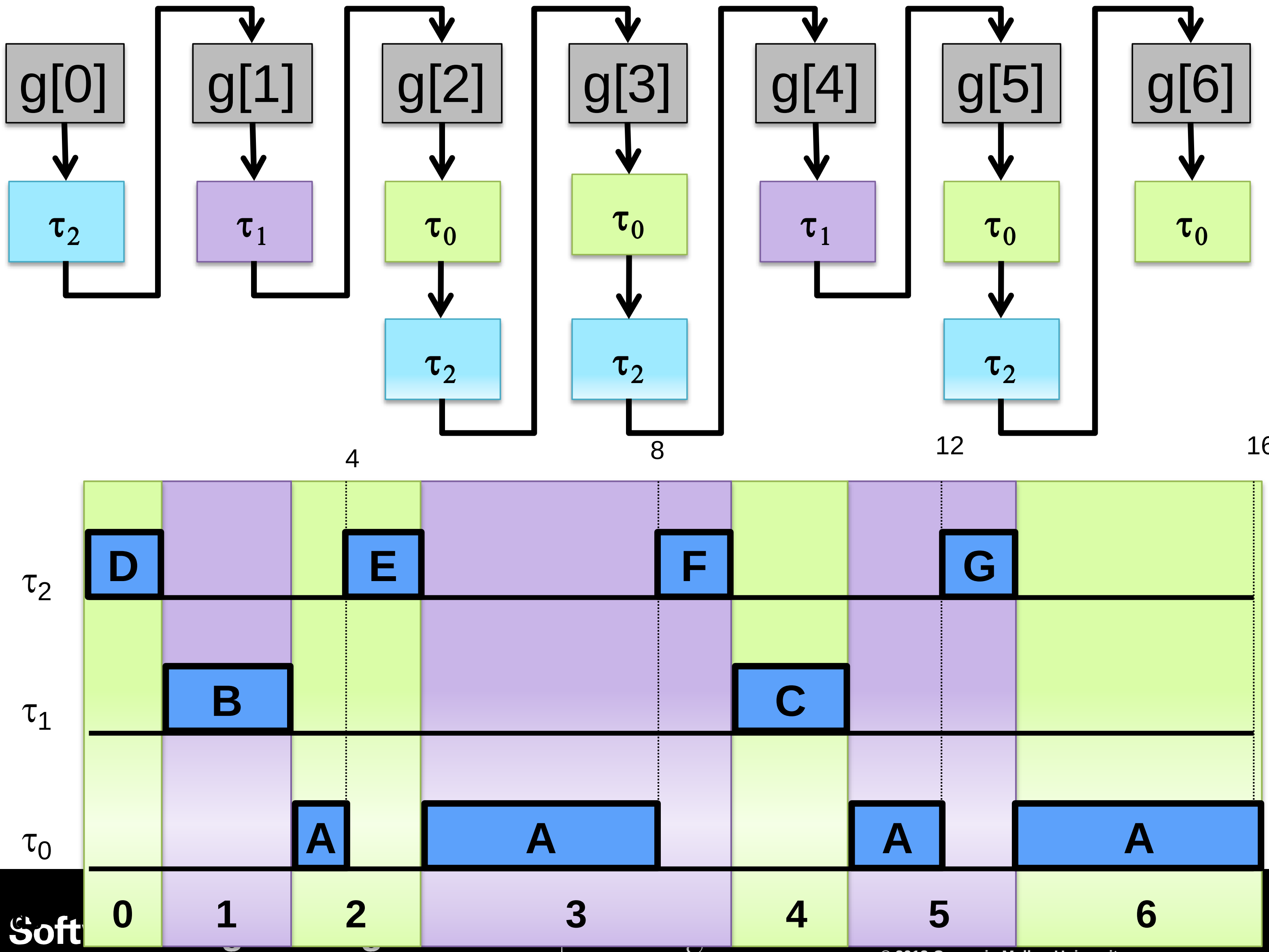
Remember initial values in prophecy variables

Execute task bodies

- τ_0
- τ_1
- τ_2

Check that final value of round i is the initial value of round i +1 (using the remembered prophecy values)

Sequentialization: Visually



Sequentialization: Overview

Sequential Program for execution of R rounds:

1. for each global variable g
   - let i_g[i] be the prophesied initial value of g in round i
     - Initialize i_g[i] with a non-deterministic value
   - let g[i] be the value of g in round i
     - Initialize g[i] to be equal to i_g[i]
2. non-deterministically choose for each task t and job j
   - start round: start[t][j]
   - end round: end[t][j]

   } must be well-nested
3. execute task bodies sequentially
   - in ascending order of priorities
   - for global variables, use g[i] instead of g when running in round i
   - non-deterministically decide where to context switch
   - at a context switch jump to a new round (cannot preempt a higher task)
4. check that initial value of round i+1 is the final value of round i
5. check user assertions

Sequentialization

```
var
  int round;                        // current round
  int job;                          // current job
  int endRound;                     // end round of the current job
  int g[R], i_g[R];                 // global and initial global
  int start[N][J], end[N][J];       // start/end round of every job
  Bool localAssert[N][J] = {1..1}; // local assertions
```

```
void main ()
  initShared();
```

```
initShared ()
 for each global g: g[0] = init_value (g);
```

User-supplied initial value of g

Sequentialization

```
var
  int round;                       // current round
  int job;                         // current job
  int endRound;                    // end round of the current job
  int g[R], i_g[R];                // global and initial global
  int start[N][J], end[N][J];      // start/end round of every job
  Bool localAssert[N][J] = {1..1}; // local assertions
```

```
void main ()
  initShared();
  initGlobals();
```

```
initGlobals ()
  for r in [1,R): //for each round
    for each global g: g[r] = i_g[r] = nondet();
```

Returns a non-deterministic value

Current Value of g at round r

Prophecied initial value of g at round r

Sequentialization

```
var
  int round;                         // current round
  int job;                           // current job
  int endRound;                      // end round of the current job
  int g[R], i_g[R];                  // global and initial global
  int start[N][J], end[N][J];        // start/end round of every job
  Bool localAssert[N][J] = {1..1}; // local assertions
```

```
void main ()
  initShared();
  initGlobals();
  scheduleJobs();
```

Will look at this in more details later, but it will essentially assign appropriate values to start[x][y] and end[x][y]

Sequentialization

```
var
  int round;                        // current round
  int job;                          // current job
  int endRound;                     // end round of the current job
  int g[R], i_g[R];                 // global and initial global
  int start[N][J], end[N][J];       // start/end round of every job
  Bool localAssert[N][J] = {1..1}; // local assertions
```

```
void main ()
  initShared();
  initGlobals();
  scheduleJobs();

  for t in [0,N) : // for each task
    for j in [0,J_t) : // for each job
      job = j;
      round = start[t][job];
      endRound = end[t][job];
      T'_t();
      assume (round == endRound);
```

Let's look at this is more detail

Sequentialization

```
void T'_t ()
  Same as T_t, but each statement 'st' is replaced with:
    contextSwitch (t); st[g ← g[round]];
  and each 'assert(e)' is replaced with:
    localAssert[t][job] = e;
```

```
void contextSwitch (task t)
  int oldRound;

  if (nondet ()) return;  // non-det do not context switch

  oldRound = round;
  round = nondet_int ();
  assume (oldRound < round <= endRound);

  // for each higher priority job, ensure that t does not preempt it
  for t1 in [t+1, N) :
    for j1 in [0,J_t1) :
      assume(round <= start[t1][j1] ||  round > end[t1][j1]);
```

Sequentialization

```
var
  int round;                        // current round
  int job;                          // current job
  int endRound;                     // end round of the current job
  int g[R], i_g[R];                 // global and initial global
  int start[N][J], end[N][J];       // start/end round of every job
  Bool localAssert[N][J] = {1..1}; // local assertions
```

```
void main ()
  scheduleJobs();
  initShared();
  initGlobals();

  for t in [0,N) : // for each thread
    for j in [0,J_t) : // for each job
      job = j;
      round = start[t][job];
      endRound = end[t][job];
      T'_t();
      assume (round == endRound);

  checkAssumptions ();
```

```
checkAssumtpions ()
  for r in [0,R-1):
    for each global g:
      assume (g[r] == i_g[r+1]);
```

Sequentialization

```
var
  int round;                         // current round
  int job;                           // current job
  int endRound;                      // end round of the current job
  int g[R], i_g[R];                  // global and initial global
  int start[N][J], end[N][J];        // start/end round of every job
  Bool localAssert[N][J] = {1..1};   // local assertions
```

```
void main ()
  scheduleJobs();
  initShared();
  initGlobals();

  for t in [0,N) : // for each thread
    for j in [0,J_t) : // for each job
      job = j;
      round = start[t][job];
      endRound = end[t][job];
      T'_t();
      assume (round == endRound);

  checkAssumptions ();
  checkAssertions ();
```

```
checkAssumtpions ()
  for r in [0,R-1):
    for each global g:
      assume (g[r] == i_g[r+1]);
```

```
checkAssertions ()
  for t in [0,N-1):
    for j in [0,J_t):
      assert (localAssert[t][j]);
```

Sequentialization

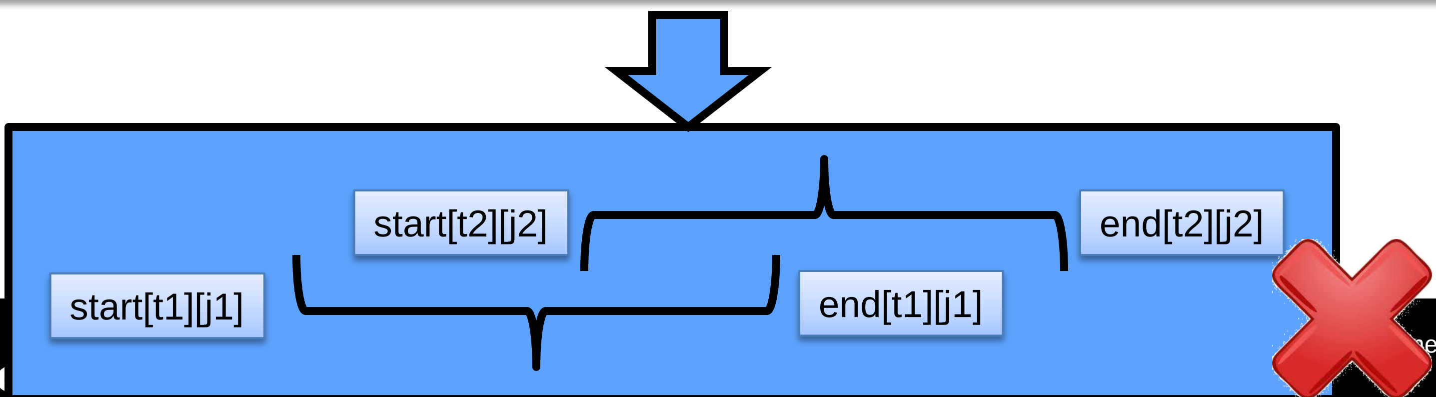
```
var
  int round;                          // current round
  int job;                            // current job
  int endRound;                       // end round of the current job
  int g[R], i_g[R];                   // global and initial global
  int start[N][J], end[N][J];         // start/end round of every job
  Bool localAssert[N][J] = {1…1}; // local assertions
```

```
void main ()
  scheduleJobs();
  initShared();
  initGlobals();

  for t in [0,N) : // for each task
    for j in [0,J_t) : // for each job
      job = j;
      round = start[t][job];
      endRound = end[t][job];
      T'_t();
      assume (round == endRound);

  checkAssumptions ();
  checkAssertions ();
```

Full Sequentialization

Sequentialization: Job Scheduling

```
void scheduleJobs ()
  for t in [0,N) :                                  // for each task
    for j in [0, J_t):                              // for each job
      start[t][j] = nondet_int ();
      end[t][j] = nondet_int ();
      assume (0 <= start[t][j]);                    // start in a legal round
      assume (end[t][j] <= R);                      // end in a legal round
      assume (start[t][j] <= end[t][j]);            // start before end
      assume (end[t][j] < start[t][j+1]);          // jobs are run in order

  // jobs are well-nested (low priority job does not preempt a high priority job)
  for t1 in [0,N-1): //  for each task
    for t2 in [t1 + 1,N): // for each task
      for j1 in [0, J_t1):   //for each job of t1
        for j2 in [0, J_t2): //for each job of t2
          if (start[t1][j1] <= end[t2][j2] && start[t2][j2] <= end[t1][j1])
             assume (start[t1][j1] <= start[t2][j2] <= end[t2][j2] <= end[t1][j1])
```



Additional Parts

Partial Order Reduction

- allow for context switches ONLY at statements that access shared variables
- ensure that read statements are preempted by write statements…

Preemption bounds

- we use RMA to compute an upper bound on the number of times one task can preempt another
- scheduleJobs() enforces this bound with additional constraints

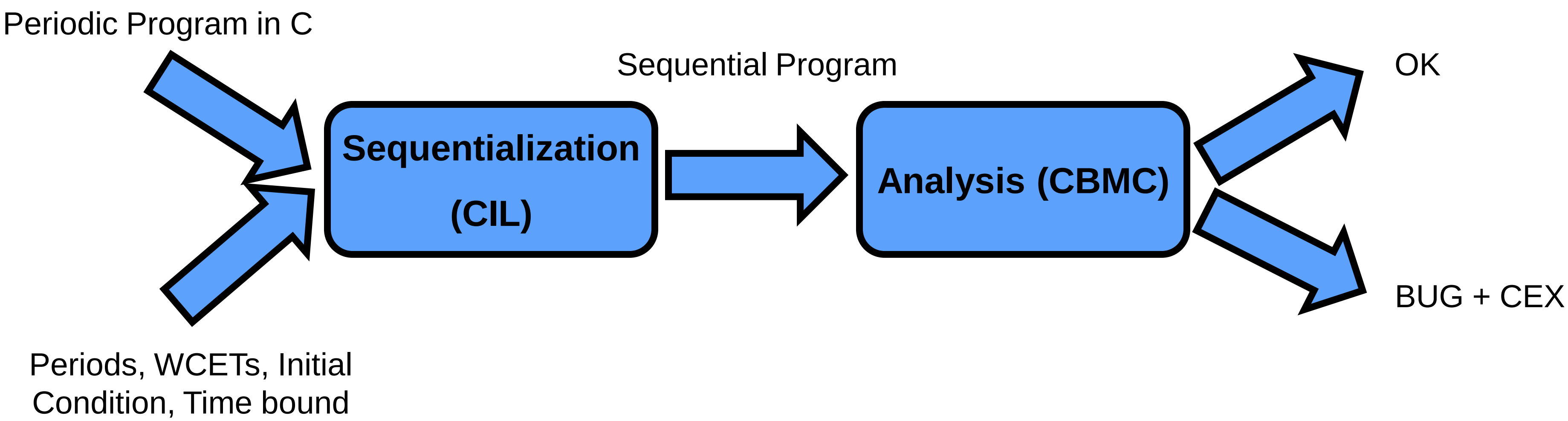
Locks

- preemption locks
  - do not allow context switch when a task holds a lock
- priority ceiling locks
  - extend the model with dynamic priorities (see details in the paper)

Assertions

- jump to the end of the execution as soon as a local assertion is violated

Implementation: REK



http://www.andrew.cmu.edu/~arieg/Rek

NXTway-GS: a 2 wheeled self-balancing robot

Original: nxt (2 tasks)

- *balancer* (4ms)
  - Keeps the robot upright and responds to BT commands
- *obstacle* (50ms)
  - monitors sonar sensor for obstacle and communicates with *balancer* to back up the robot

Ours: aso (3 tasks)

- *balancer* as above but no BT
- *obstacle* as above
- *bluetooth* (100ms)
  - responds to BT commands and communicates with the *balancer*

Verified consistency of communication between tasks

Experimental Results

Name	Program Size		SAT Size			Safe	Time(s)
	OL	SL	GL	Var	Clause		
nxt.ok1	377	2,265	6,541	136,944	426,686	Y	22.16
nxt.bug1	378	2,265	6,541	136,944	426,686	N	9.95
nxt.ok2	368	2,322	6,646	141,305	439,548	Y	13.92
nxt.bug2	385	2,497	7,398	144,800	451,517	N	17.48
nxt.ok3	385	2,497	7,386	144,234	449,585	Y	18.32
aso.bug1	401	2,680	7,835	178,579	572,153	N	16.32
aso.bug2	400	2,682	7,785	176,925	566,693	N	15.01
aso.ok1	398	2,684	7,771	175,221	560,992	Y	66.43
aso.bug3	426	3,263	10,387	373,426	1,187,155	N	59.66
aso.bug4	424	3,250	9,918	347,628	1,099,644	N	31.51
aso.ok2	421	3,251	9,932	348,252	1,101,784	Y	328.32

Time bound: 100ms
No partial order reduction

OL = #of original LOC
SL = #of seq LOC
GL = #of goto LOC
Var = #of SAT vars
Clause = #of SAT clauses
Safe = whether assert valid

Experimental Results: Partial Order Reduction

Lock-Free Reader-Writer protocols

Name	Program Size		SAT Size			Safe	Time(s)
	OL	SL	GL	Var	Clause		
RW1	190	3,428	5,860	42,441	125,150	Y	20.74
RW1-PO	190	5,021	7,626	45,493	134,818	Y	**14.71**
RW2	239	4,814	8,121	52,171	152,512	Y	165.89
RW2-PO	239	7,356	10,388	56,039	164,332	Y	**162.2**
RW3	285	7,338	21,163	139,542	419,737	Y	436.86
RW3-PO	285	12,002	26,283	153,826	467,105	Y	**199.13**
RW4	244	7,255	19,745	117,406	350,610	Y	321.25
RW4-PO	244	12,272	24,261	130,229	392,289	Y	**59.66**
RW5	188	3,198	5,208	41,371	119,037	Y	47.83
RW5-PO	188	4,791	7,138	45,321	131,701	Y	**20.35**
RW6	257	5,231	7,634	54,829	157,764	Y	165.33
RW6-PO	257	8,235	10,119	59,744	173,061	Y	**157.43**

OL = #of original LOC
SL = #of seq LOC
GL = #of goto LOC
Var = #of SAT vars
Clause = #of SAT clauses
Safe = whether assert valid

Related Work

Sequentialization of Concurrent Programs (Lal & Reps ‘08, and others)

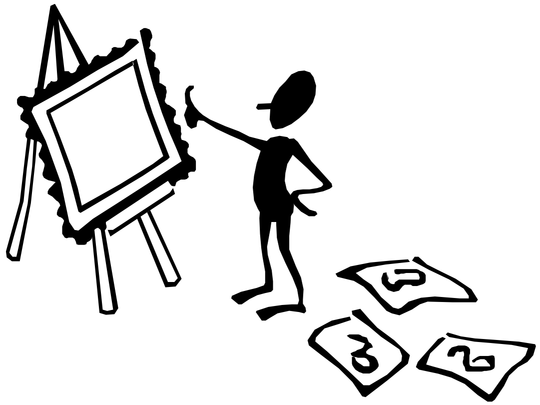
- Context Bounded Analysis of concurrent programs via sequentialization
- Arbitrary concurrent software
- Non-deterministic round robin scheduler
- Preserve executions with bounded number of thread preemptions
- Allow for arbitrary number of preemptions between tasks

Sequentialization of Periodic Programs (Kidd, Jagannathan, Vitek ’10)

- Same setting as this work
- Alternative sol’n: replace preemptions by non-deterministic function calls
- Additionally, supports recursion and inheritance locks
- No publicly available implementation – would be interesting to compare

Verification of Time Properties of (Models of) Real Time Embedded Systems

Conclusion



Past

- Time Bounded Verification of Periodic C Programs
- Small (but hard) toy programs
- Reader/Writer protocols (with locks and lock-free versions)
- A robot controller for LEGO MINDSTORM from nxtOSEK examples

Present

- Taking into account additional timing constraints for improved scheduling
  - arrival times, harmonicity, etc.
- A Lego Metal Stamping Robot (a.k.a. Turing Machine)
  - http://www.andrew.cmu.edu/~arieg/Rek (look for Turing Machine demo)

Future

- Verification without the time bound
- Abstraction / Refinement
- Additional communication and synchronization
  - Priority-inheritance locks, message passing
- Modeling physical aspects (i.e., environment) more faithfully
- More Case studies and model problems

QUESTIONS?

Contact Information

Presenter
Arie Gurfinkel
RTSS
Telephone: +1 412-268-5800
Email: arie@cmu.edu

U.S. mail:
Software Engineering Institute
Customer Relations
4500 Fifth Avenue
Pittsburgh, PA 15213-2612
USA

Web:
www.sei.cmu.edu
http://www.sei.cmu.edu/contact.cfm

Customer Relations
Email: info@sei.cmu.edu
Telephone: +1 412-268-5800
SEI Phone: +1 412-268-5800
SEI Fax: +1 412-268-6257