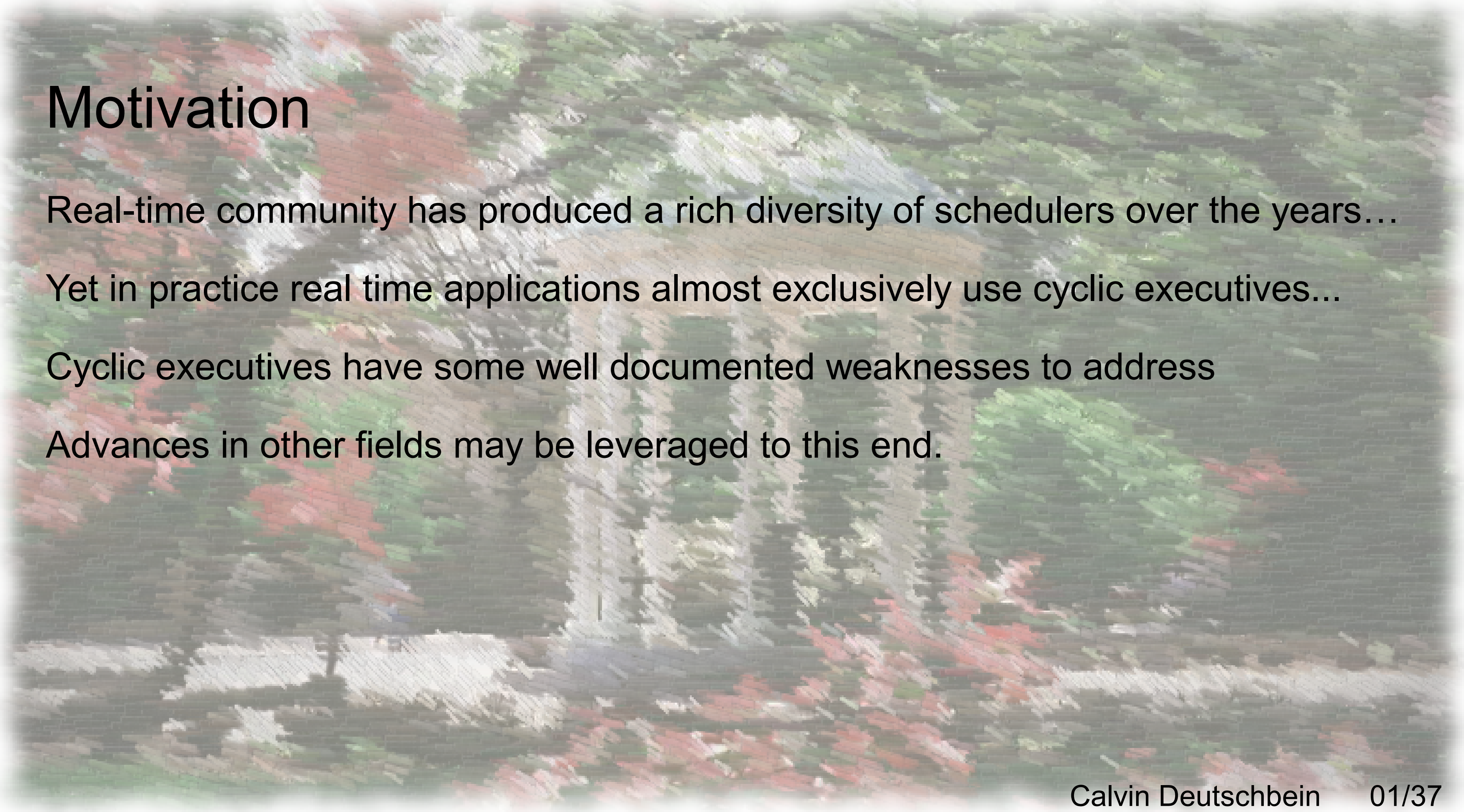
# Multi-core Cyclic Executives for Safety-Critical Systems

Calvin Deutschbein
University of North Carolina - Chapel Hill
cd@cs.unc.edu



## Motivation

Real-time community has produced a rich diversity of schedulers over the years…

Yet in practice real time applications almost exclusively use cyclic executives...

Cyclic executives have some well documented weaknesses to address

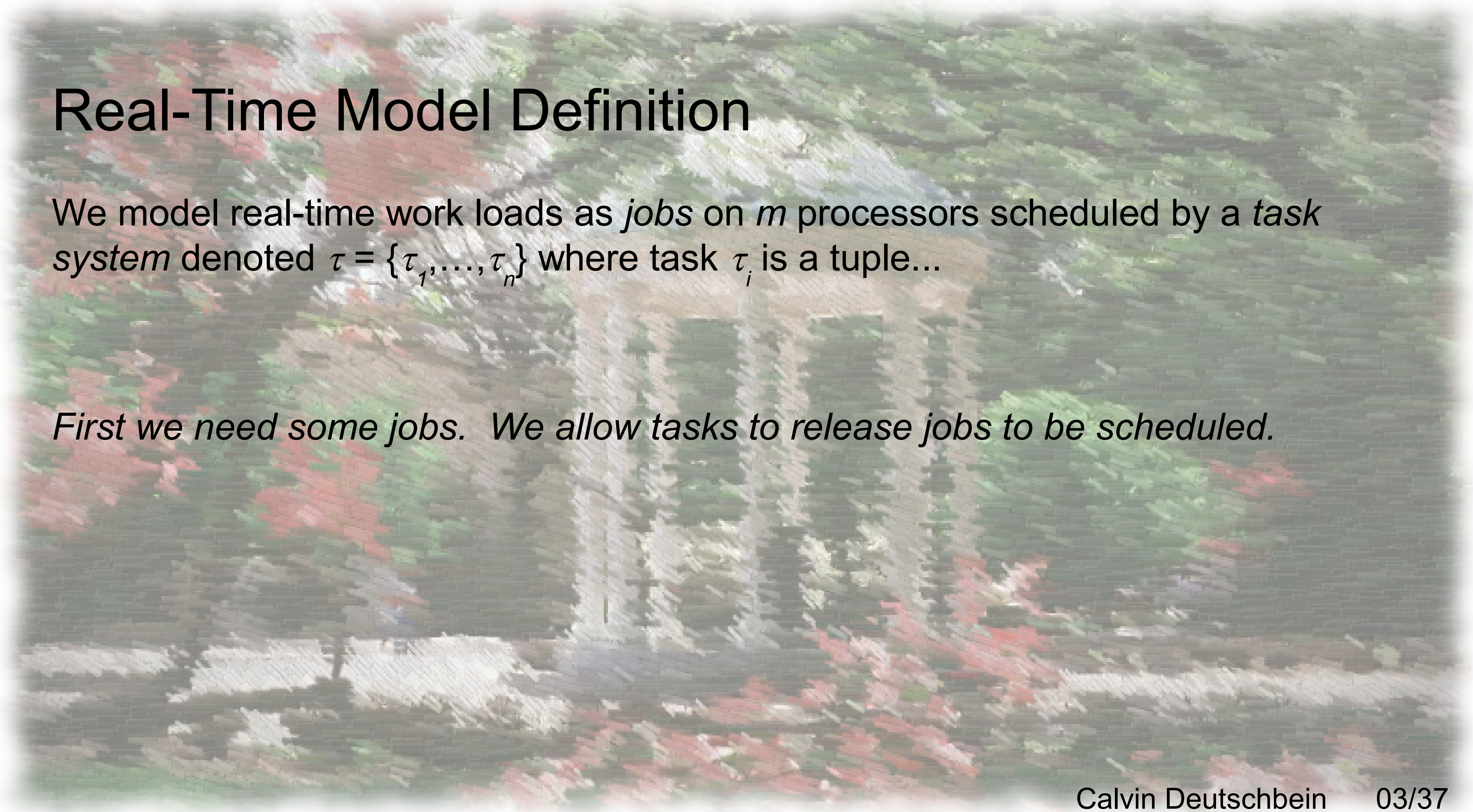
Advances in other fields may be leveraged to this end.

*This presentation will cover:*

Intro to Real-time Model

Intro to Cyclic Executive

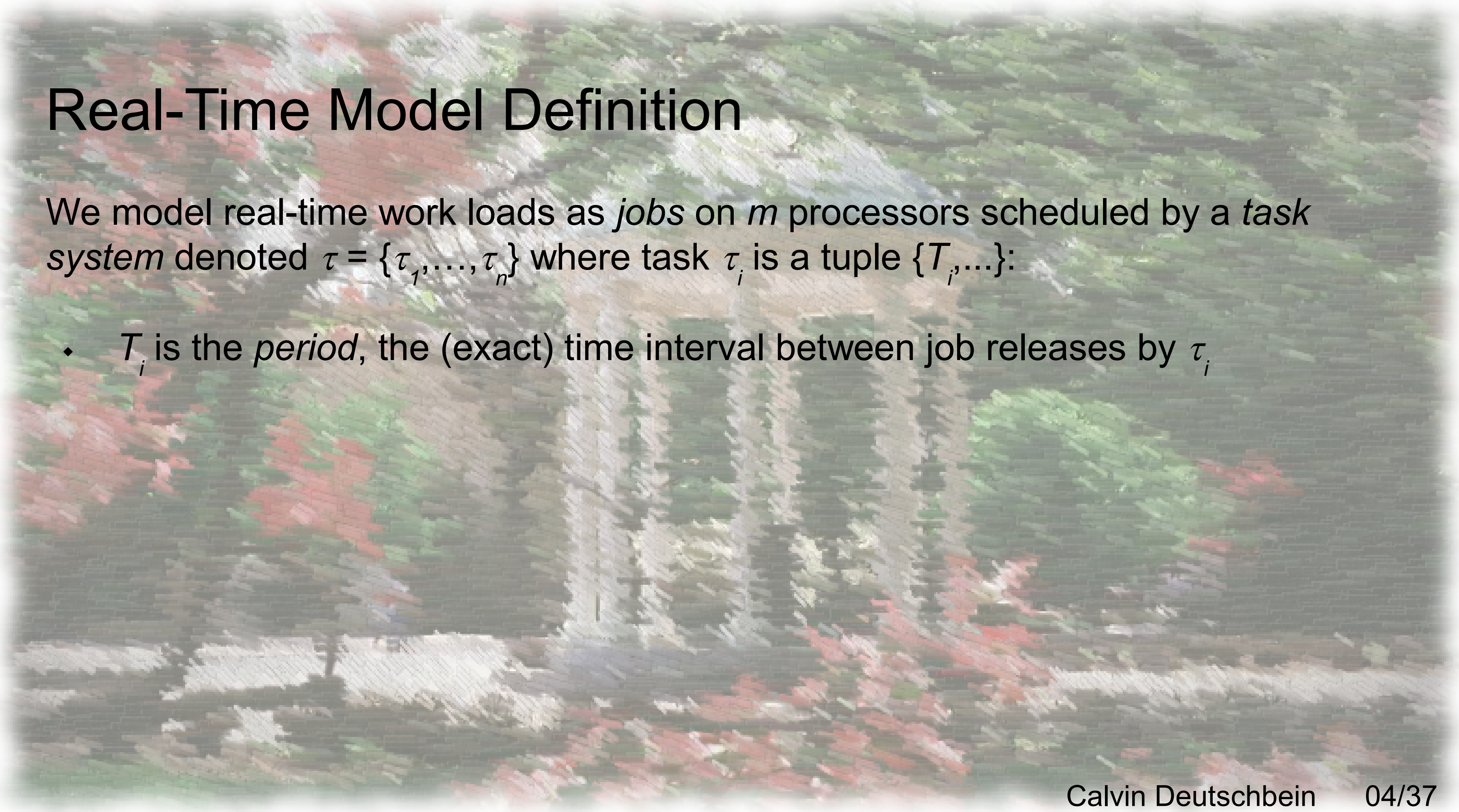
Generating Multi-core Cyclic Executives



## Real-Time Model Definition

We model real-time work loads as *jobs* on *m* processors scheduled by a *task system* denoted $\tau = \{\tau_1,\ldots,\tau_n\}$ where task $\tau_i$ is a tuple...

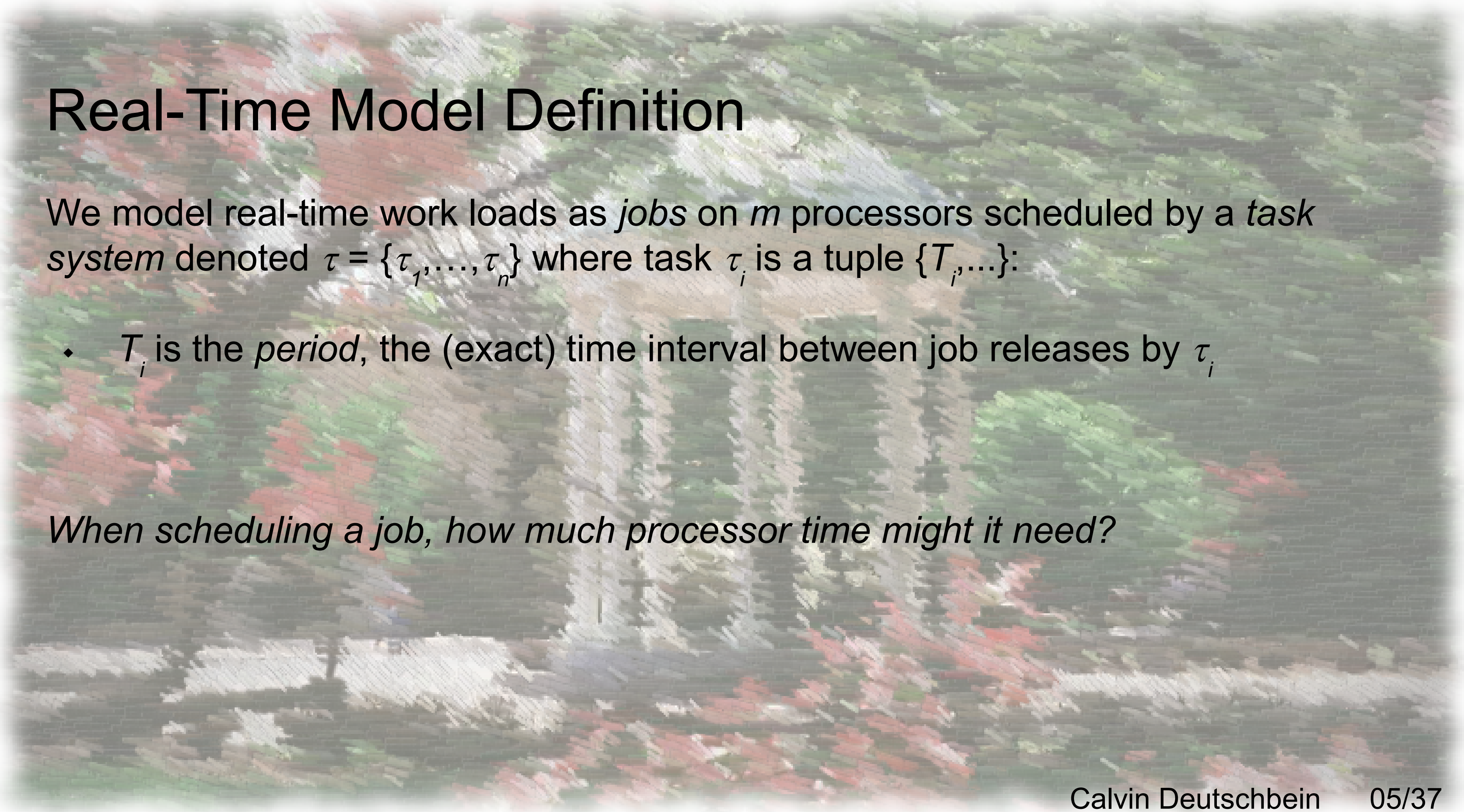
*First we need some jobs. We allow tasks to release jobs to be scheduled.*



## Real-Time Model Definition

We model real-time work loads as *jobs* on *m* processors scheduled by a *task system* denoted $\tau = \{\tau_1,\ldots,\tau_n\}$ where task $\tau_i$ is a tuple $\{T_i,\ldots\}$:

- $T_i$ is the *period*, the (exact) time interval between job releases by $\tau_i$



## Real-Time Model Definition

We model real-time work loads as *jobs* on *m* processors scheduled by a *task system* denoted $\tau = \{\tau_1,\ldots,\tau_n\}$ where task $\tau_i$ is a tuple $\{T_i,...\}$:

- $T_i$ is the *period*, the (exact) time interval between job releases by $\tau_i$

*When scheduling a job, how much processor time might it need?*

## Real-Time Model Definition

We model real-time work loads as *jobs* on *m* processors scheduled by a *task system* denoted $\tau = \{\tau_1,\ldots,\tau_n\}$ where task $\tau_i$ is a tuple $\{T_i,C_i\}$:

- $T_i$ is the *period*, the (exact) time interval between job releases by $\tau_i$
- $C_i$ is the *WCET* or worst case execution time of each job released by $\tau_i$

## Real-Time Model Definition

We model real-time work loads as *jobs* on *m* processors scheduled by a *task system* denoted $\tau = \{\tau_1,\ldots,\tau_n\}$ where task $\tau_i$ is a tuple $\{T_i,C_i\}$:

- $T_i$ is the *period*, the (exact) time interval between job releases by $\tau_i$
- $C_i$ is the *WCET* or worst case execution time of each job released by $\tau_i$

*How long after job release does it have to achieve temporal correctness?*

## Real-Time Model Definition

We model real-time work loads as *jobs* on *m* processors scheduled by a *task system* denoted $\tau = \{\tau_1,\ldots,\tau_n\}$ where task $\tau_i$ is a tuple $\{T_i,C_i\}$:

- $T_i$ is the *period*, the (exact) time interval between job releases by $\tau_i$
- $C_i$ is the *WCET* or worst case execution time of each job released by $\tau_i$
- $T_i$ also the *relative deadline* or how long after release the job has to complete

*This is the standard implicit-deadline periodic real-time model that cyclic executives schedule.*

# Real-Time Model Definition: Example

Well rested is sleeping 8 hours every 24 hours. $\{T_1 = 024, C_1 = 008\}$

Well fed is eating once every 8 hours. $\{T_2 = 008, C_2 = 001\}$

Full-time work is 40 hours every week. $\{T_3 = 168, C_3 = 040\}$

*Is this schedule feasible? How?*

## Intro to Cyclic Executive

A cyclic executive is a finite length predetermined schedule created off-line

At run-time, the schedule is executed repeatedly or cyclically

Each repetition is called a “major frame” or “major schedule”

A major frame is formed of a sequence of “minor frames” of a fixed time

Jobs are scheduled within the minor frames such that each job has sufficient time

*“common way to describe a schedule is to describe a complete major cycle as a sequence of different minor cycles, and to express each minor cycle as a sequence of actions”*

## Advantages of Cyclic Executives

Cyclic Executives are:

- Efficient and low overhead
- Highly Predictable
- Palatable to certification authorities
- Widely used in practice

## Disadvantages of Cyclic Executives

Cyclic Executives are:

- Frequently rely on schedules that are generated *ad hoc*
- Lack a firm theoretical foundation to expand into multi-core architecture
- Only suitable to certain types of workloads

## Cyclic Executive: Example

$\{T_1 = 024, C_1 = 008\}, \{T_2 = 008, C_2 = 001\}, \{T_3 = 168, C_3 = 040\}$

From $t$ = 00 + 24$n$ to $t$ = 01 + 24$n$: $\tau_2$

From $t$ = 01 + 24$n$ to $t$ = 09 + 24$n$: $\tau_1$

From $t$ = 09 + 24$n$ to $t$ = 10 + 24$n$: $\tau_2$

From $t$ = 10 + 24$n$ to $t$ = 18 + 24$n$: $\tau_3$ (or idle if complete)

From $t$ = 18 + 24$n$ to $t$ = 19 + 24$n$: $\tau_2$

Otherwise idle.

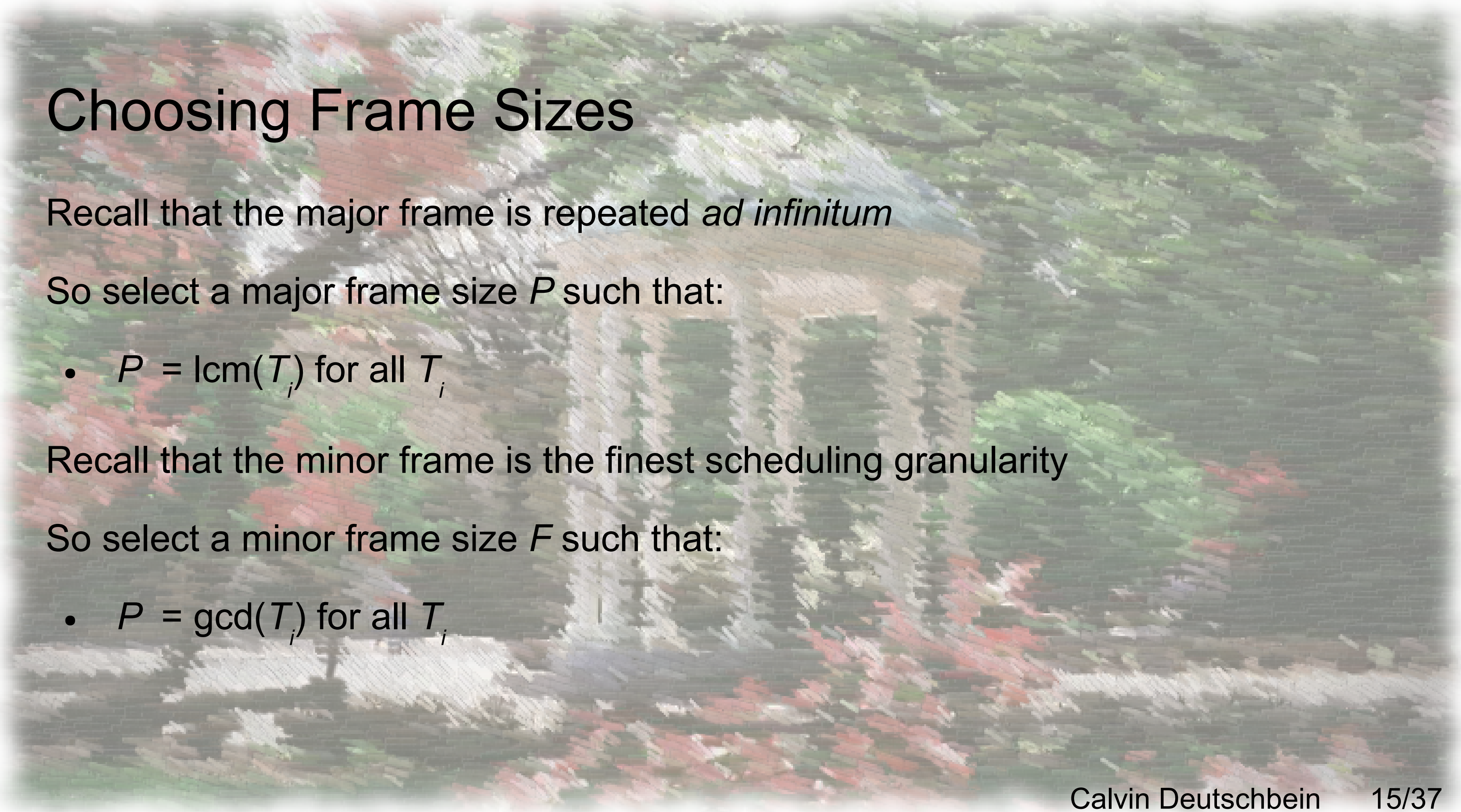
## Creating Cyclic Executives From Task Sets

Considering a *task system* $\tau = \{\tau_1, \ldots, \tau_n\}$ where task $\tau_i$ is a tuple $\{T_i, C_i\}$:

To generate a (multi-core) schedule so each $\tau_i$ is scheduled every $T_i$ units for $C_i$:

- Choose major and minor frame size
- Assign each job of each task time on a processor in an appropriate frame

*We will only consider identical processors for the sake of this presentation.*



## Choosing Frame Sizes

Recall that the major frame is repeated *ad infinitum*

So select a major frame size *P* such that:

- $P = \text{lcm}(T_i)$ for all $T_i$

Recall that the minor frame is the finest scheduling granularity

So select a minor frame size *F* such that:

- $P = \text{gcd}(T_i)$ for all $T_i$

## Choosing Frame Sizes

If *P* is larger, then the problem and schedule becomes unnecessarily large

If *P* is smaller, the schedule may not cycle appropriately

If *F* is larger, then some task will not necessarily have deadlines respected

If *F* is smaller, then the problem and schedule becomes unnecessarily large

## Creating the Cyclic Executive: Linear Program

Corresponding linear programs can create cyclic executives

Can solve both integer and linear programs – allows *non-preemptive* schedules

Linear programs take case polynomial time for preemptive schedules

Integer programs take worst case exponential time (NP-hard strong sense)

In practice, tools can often solve integer programs very quickly (eg Gurobi)

Schedule generation performed off-line.

## Creating the Cyclic Executive:  Linear Program

Recall standard form for linear programming has 3 main components:

**Objective Function:**

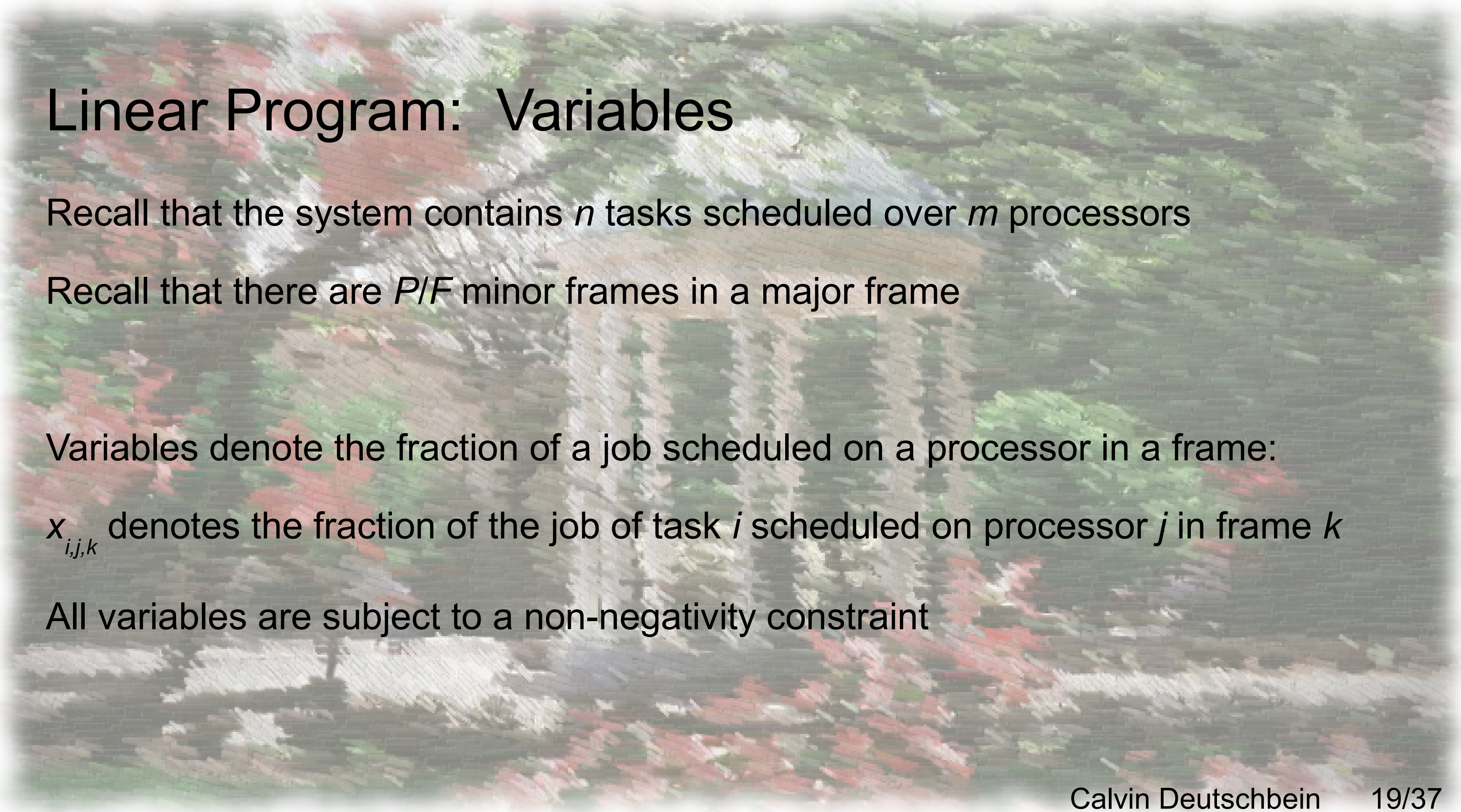
e.g. Minimize $c_i x_i + c_j x_j$

**Constraints:**

e.g. $x_i - x_j \leq n$

**Variables:**

e.g. $x_i \geq 0, x_j \geq 0$



## Linear Program: Variables

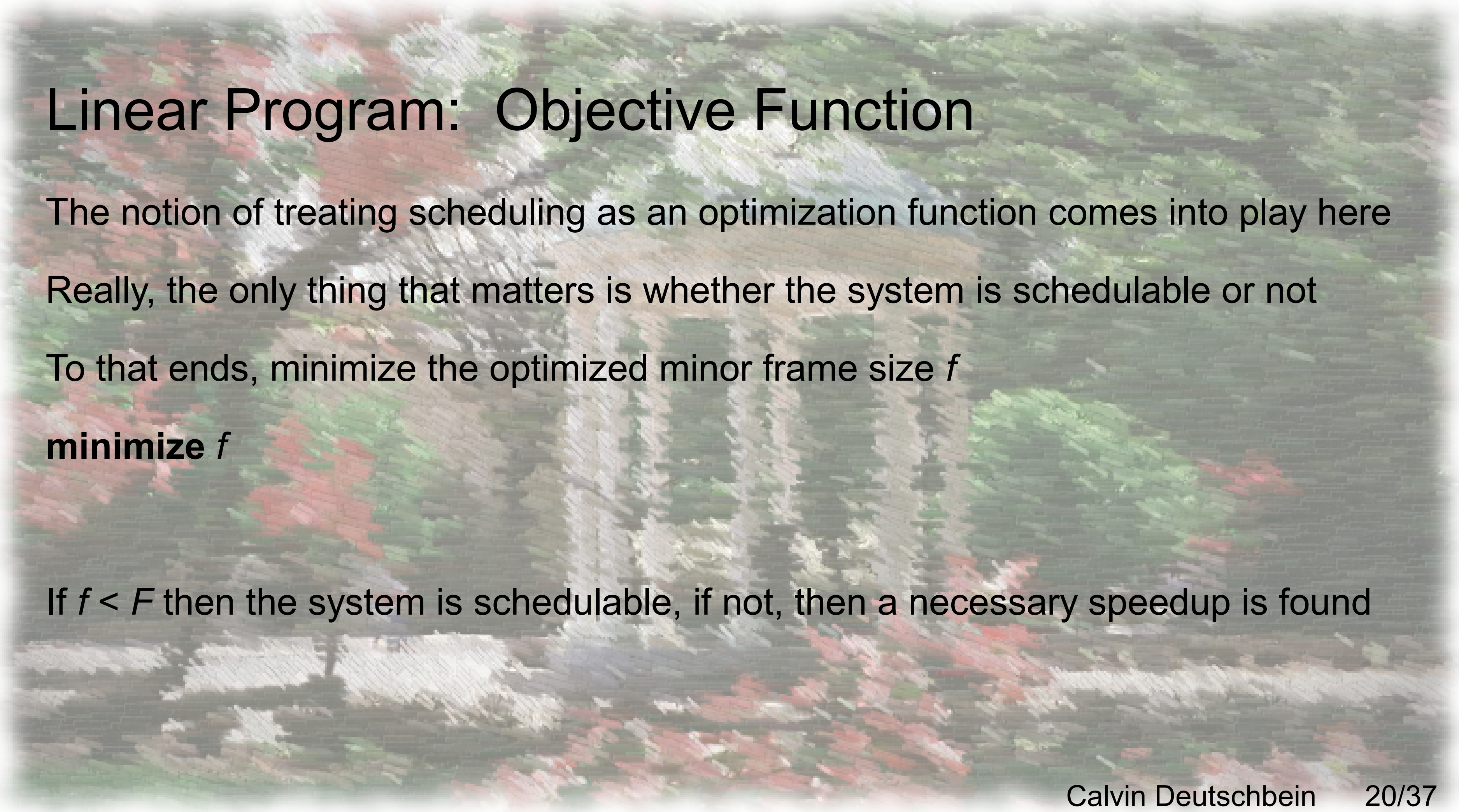
Recall that the system contains *n* tasks scheduled over *m* processors

Recall that there are *P*/*F* minor frames in a major frame

Variables denote the fraction of a job scheduled on a processor in a frame:

$x_{i,j,k}$ denotes the fraction of the job of task *i* scheduled on processor *j* in frame *k*

All variables are subject to a non-negativity constraint



## Linear Program: Objective Function

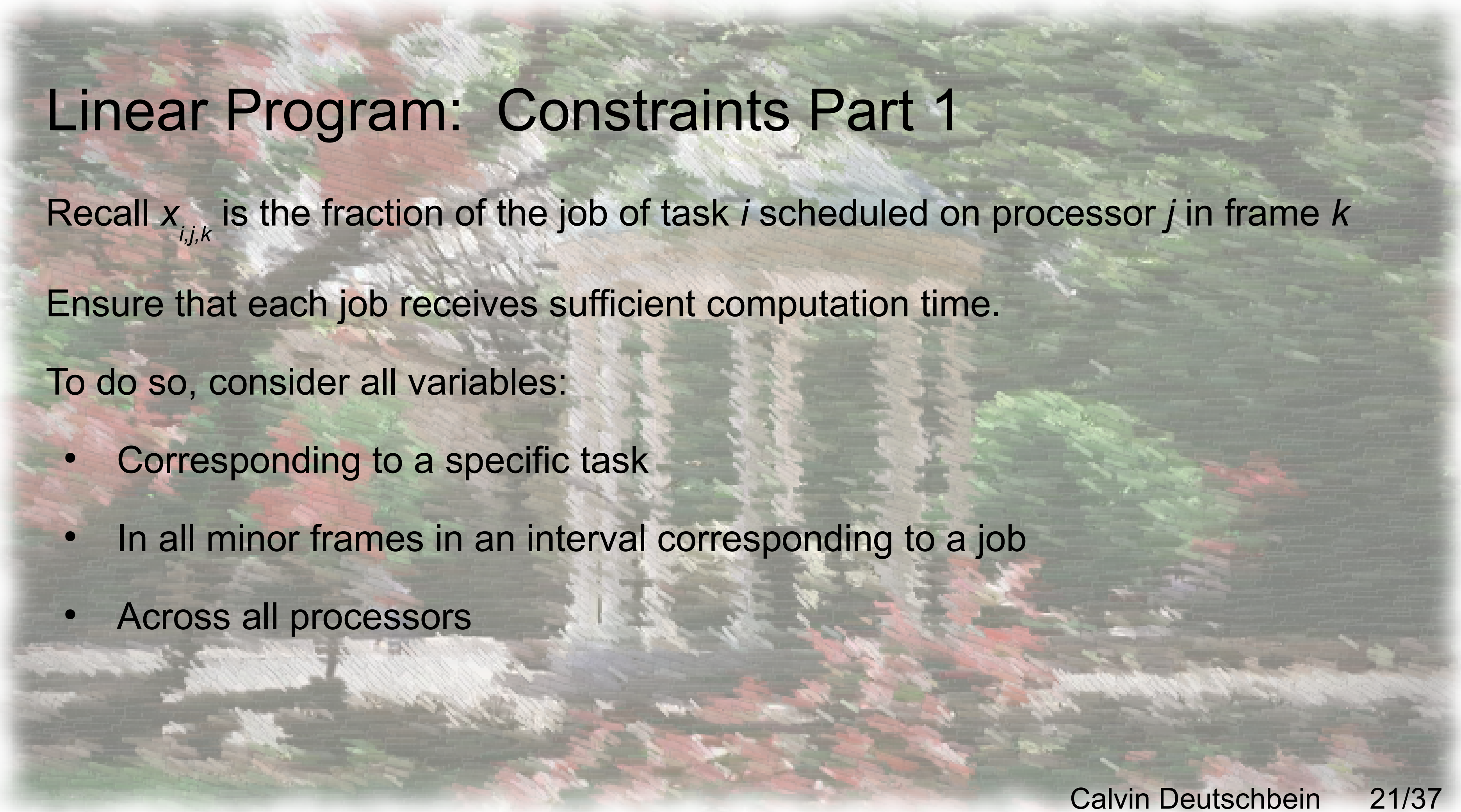
The notion of treating scheduling as an optimization function comes into play here

Really, the only thing that matters is whether the system is schedulable or not

To that ends, minimize the optimized minor frame size $f$

**minimize** $f$

If $f < F$ then the system is schedulable, if not, then a necessary speedup is found



## Linear Program: Constraints Part 1

Recall $x_{i,j,k}$ is the fraction of the job of task *i* scheduled on processor *j* in frame *k*

Ensure that each job receives sufficient computation time.

To do so, consider all variables:

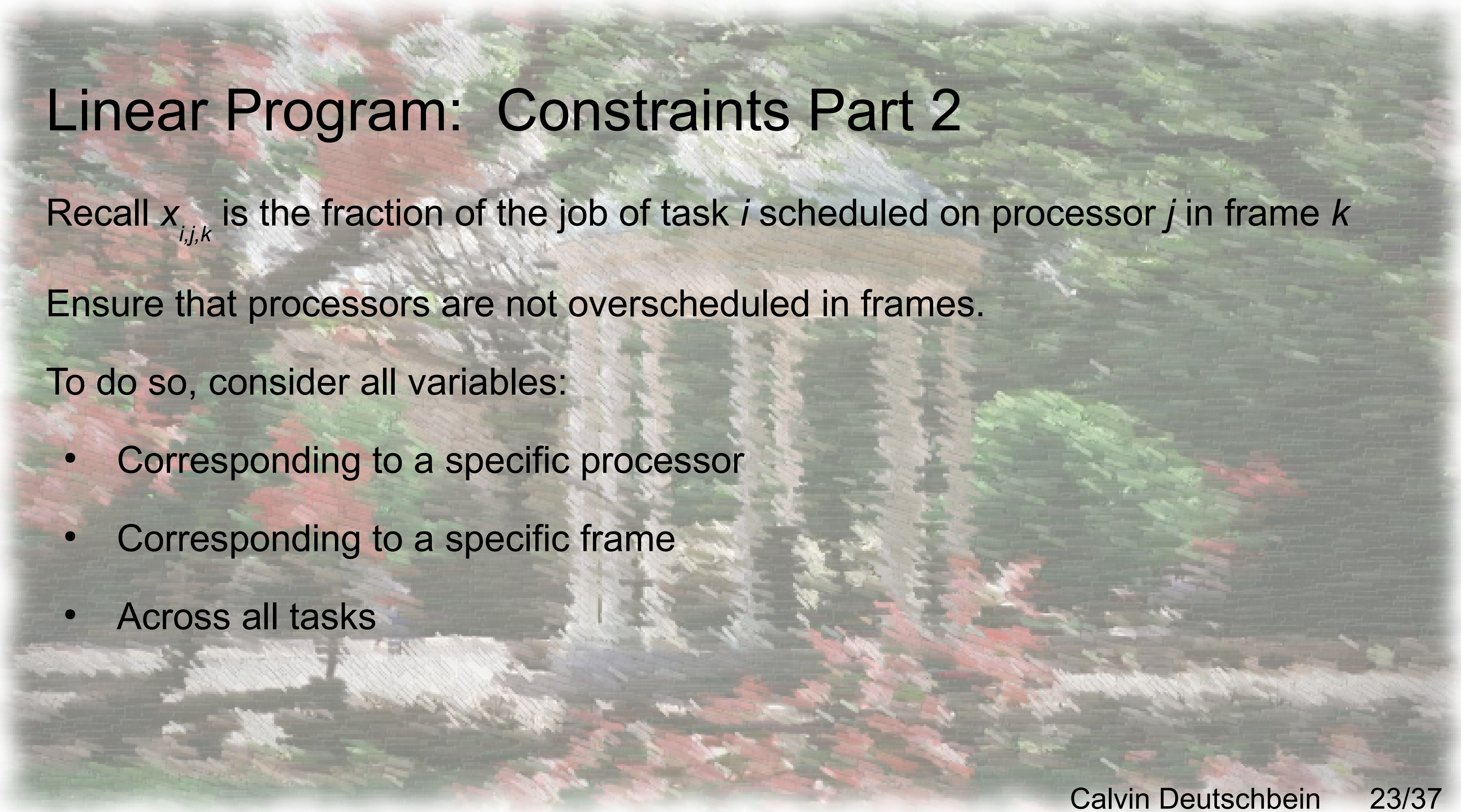
- Corresponding to a specific task
- In all minor frames in an interval corresponding to a job
- Across all processors

## Linear Program:  Constraints Part 1

Recall $x_{i,j,k}$ is the fraction of the job of task *i* scheduled on processor *j* in frame *k*

- Fix *i*
- Sum over *k* in [*c* * $T_i$/F, (*c* + 1) * $T_i$/F - 1] for each *c* in [0, *P*/$T_i$]
- Sum over all *j*

$$\forall i, \forall c \in [0, P/T_i], \sum_{j=1}^{m} \sum_{k=c*T_i/F}^{(c+1)*T_i/F-1} x_{i,j,k} = 1$$



## Linear Program:  Constraints Part 2

Recall $x_{i,j,k}$ is the fraction of the job of task *i* scheduled on processor *j* in frame *k*

Ensure that processors are not overscheduled in frames.

To do so, consider all variables:

- Corresponding to a specific processor
- Corresponding to a specific frame
- Across all tasks

## Linear Program:  Constraints Part 2

Recall $x_{i,j,k}$ is the fraction of the job of task *i* scheduled on processor *j* in frame *k*

- Fix *j*
- Fix *k*
- Sum over all *i*

$$\forall j, \forall k, \sum_{i=1}^{n} x_{i,j,k} * C_i \leq f$$

## Linear Program: Constraints Part 3

Recall $x_{i,j,k}$ is the fraction of the job of task *i* scheduled on processor *j* in frame *k*

Ensure that jobs are not concurrently scheduled on different processors.

To do so, consider all variables:

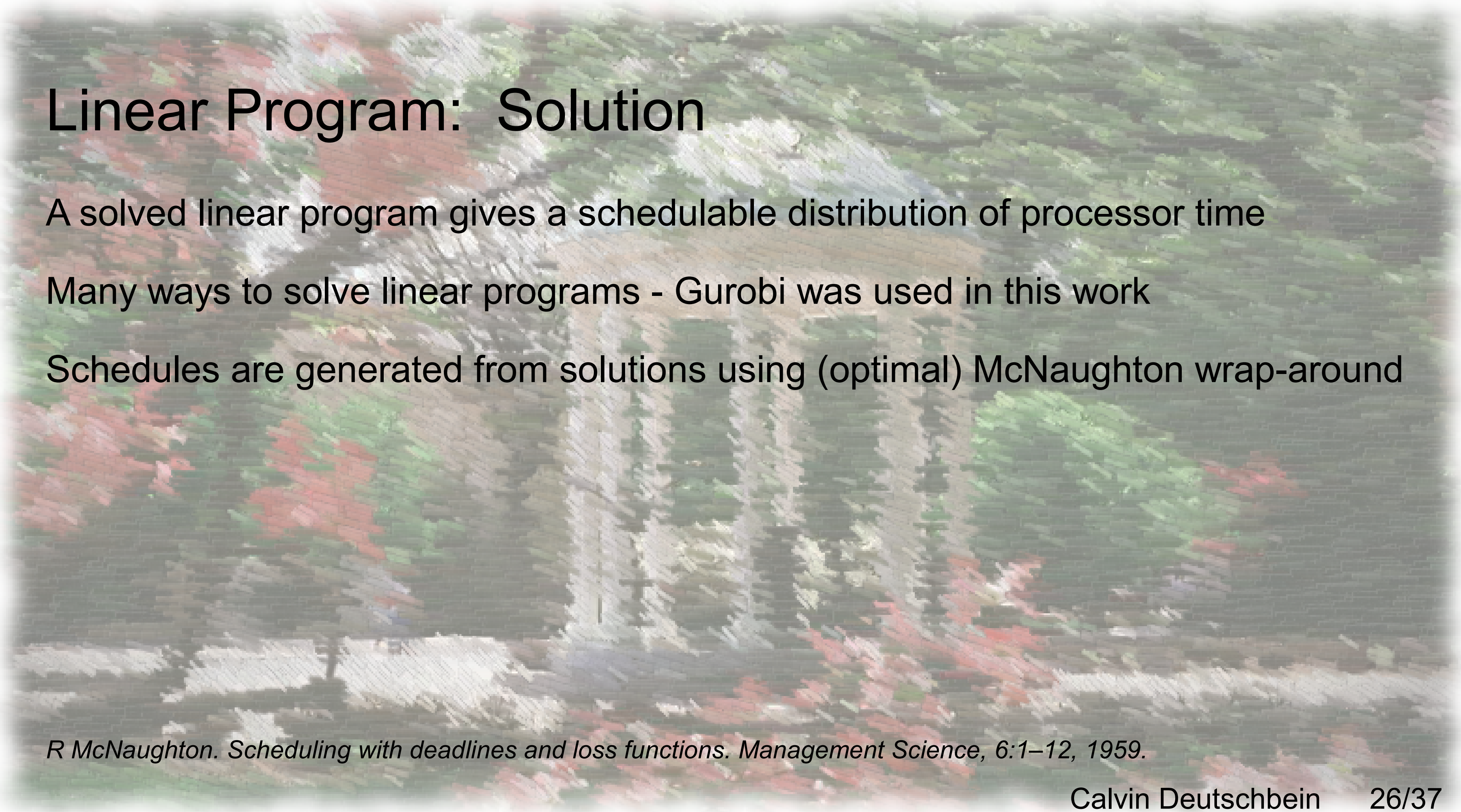
- Corresponding to a specific task
- Corresponding to a specific frame
- Across all processors

## Linear Program:  Constraints Part 3

Recall $x_{i,j,k}$ is the fraction of the job of task *i* scheduled on processor *j* in frame *k*

- Fix *i*
- Fix *k*
- Sum over all *j*

$$\forall i, \forall k, \sum_{j=1}^{m} x_{i,j,k} * C_i \leq f$$



## Linear Program: Solution

A solved linear program gives a schedulable distribution of processor time

Many ways to solve linear programs - Gurobi was used in this work

Schedules are generated from solutions using (optimal) McNaughton wrap-around

*R McNaughton. Scheduling with deadlines and loss functions. Management Science, 6:1–12, 1959.*

## Linear Program: Non-preemptive Solution

For non-preemptive solutions, jobs are not permitted to stop and restart.

This corresponds to a 0/1 integer solution to the program.

In most cases, tools are capable of determining these solutions quickly, but...

Finding such solutions is NP-Hard in the strong sense.

However, there is a polynomial time 2-approximation…

## Non-preemptive schedule: Overall Idea

Begin with the preemptive non-integer (extreme point) solution to the program.

Jobs assigned integrally are left as-is.

Partially scheduled jobs and processor-frames are placed in a bipartite graph.

A perfect matching is found (in polynomial time via Ford-Fulkerson).

Jobs are fully assigned to the frame with which they are matched.

Jobs are necessarily no longer than frame size, so…

Adding a job at most requires the frame to double in size.

*Jan-Karel Lenstra, David Shmoys, and Eva Tardos. Approximation algorithms for scheduling unrelated parallel machines. Mathematical Programming, 46:259–271, 1990.*

## Non-preemptive schedule:  Foundation

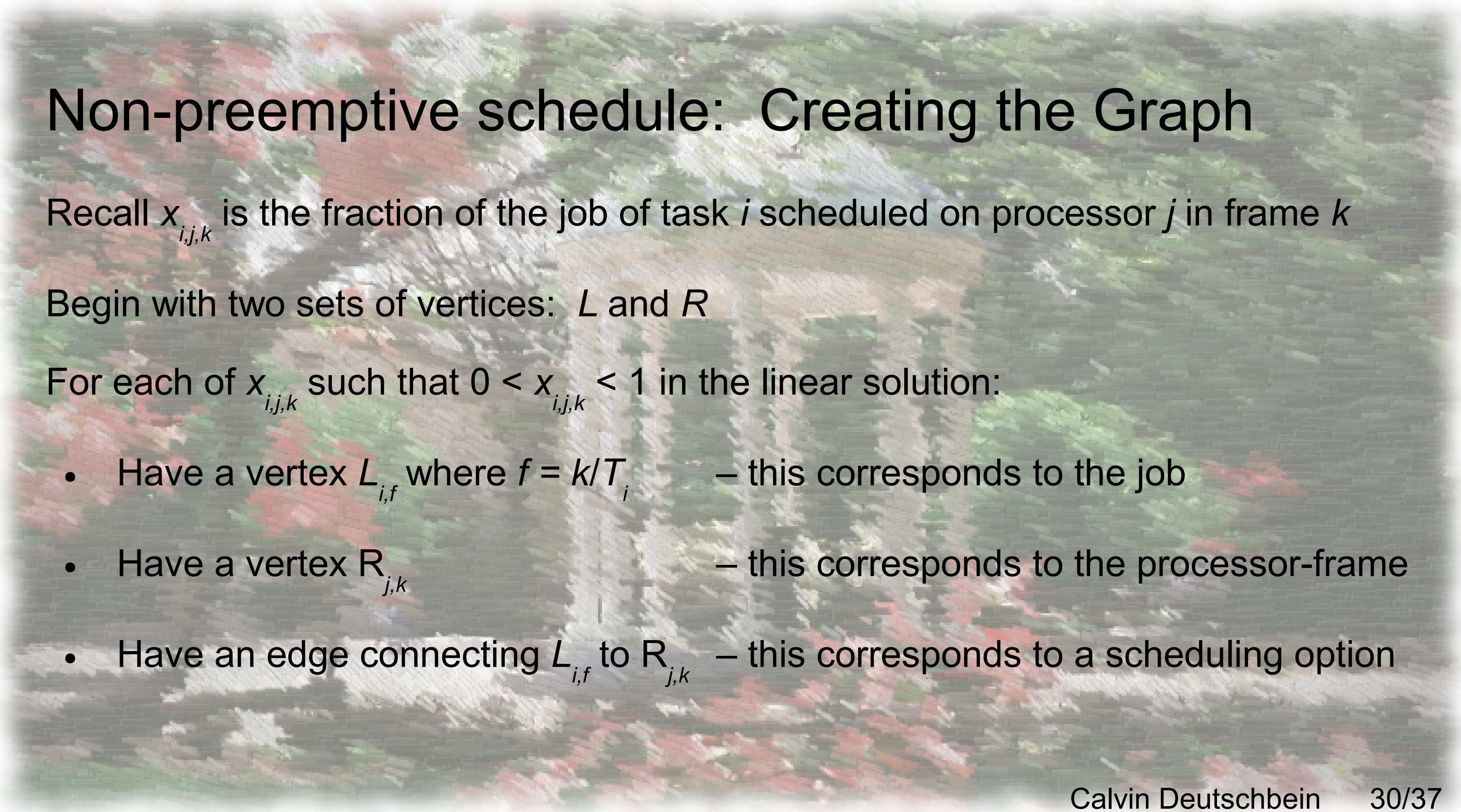
Ensure that the linear solution is *basic solution* (optimal solution at vertex point)

This can be found in polynomial time using, for example, barrier algorithm.

LP has ($n * m * P/F$) variables and ([number of jobs] + $m * P/F$) constraints, so…

At most  ([number of jobs] + $m * P/F$) take on non-zero values.

*Do note that the number of constraints is reduced by removing the requirement that jobs aren't scheduled for more than frame length.*



## Non-preemptive schedule: Creating the Graph

Recall $x_{i,j,k}$ is the fraction of the job of task *i* scheduled on processor *j* in frame *k*

Begin with two sets of vertices: *L* and *R*

For each of $x_{i,j,k}$ such that $0 < x_{i,j,k} < 1$ in the linear solution:

- Have a vertex $L_{i,f}$ where $f = k/T_i$ – this corresponds to the job
- Have a vertex $R_{j,k}$ – this corresponds to the processor-frame
- Have an edge connecting $L_{i,f}$ to $R_{j,k}$ – this corresponds to a scheduling option

## Non-preemptive schedule:  Creating the Graph

Create a *source* and *sink* vertex.

Add edges from *source* to all vertices in *L.*

Add edges from all vertices in *R* to *sink*.

Set all edge weights to 1.

Calculating max flow will give a perfect matching.

## Experimental Evaluation

Task utilization (that is, $C_i/T_i$) generated by unbiased UUniFast algorithm [4].

Task periods assigned uniformly and randomly over $F \times \{1, 2, 3, 4\}$ ($F$ = 25ms)

Task $C_i$ were determined as the product of utilization and period.

Systems with some $C_i$ greater than $F$ were discarded (as they are infeasible).

All the experiments assumed a four-processor platform ($m = 4$).

## Experimental Evaluation

Task utilization (that is, $C_i/T_i$) generated by unbiased UUniFast algorithm [4].

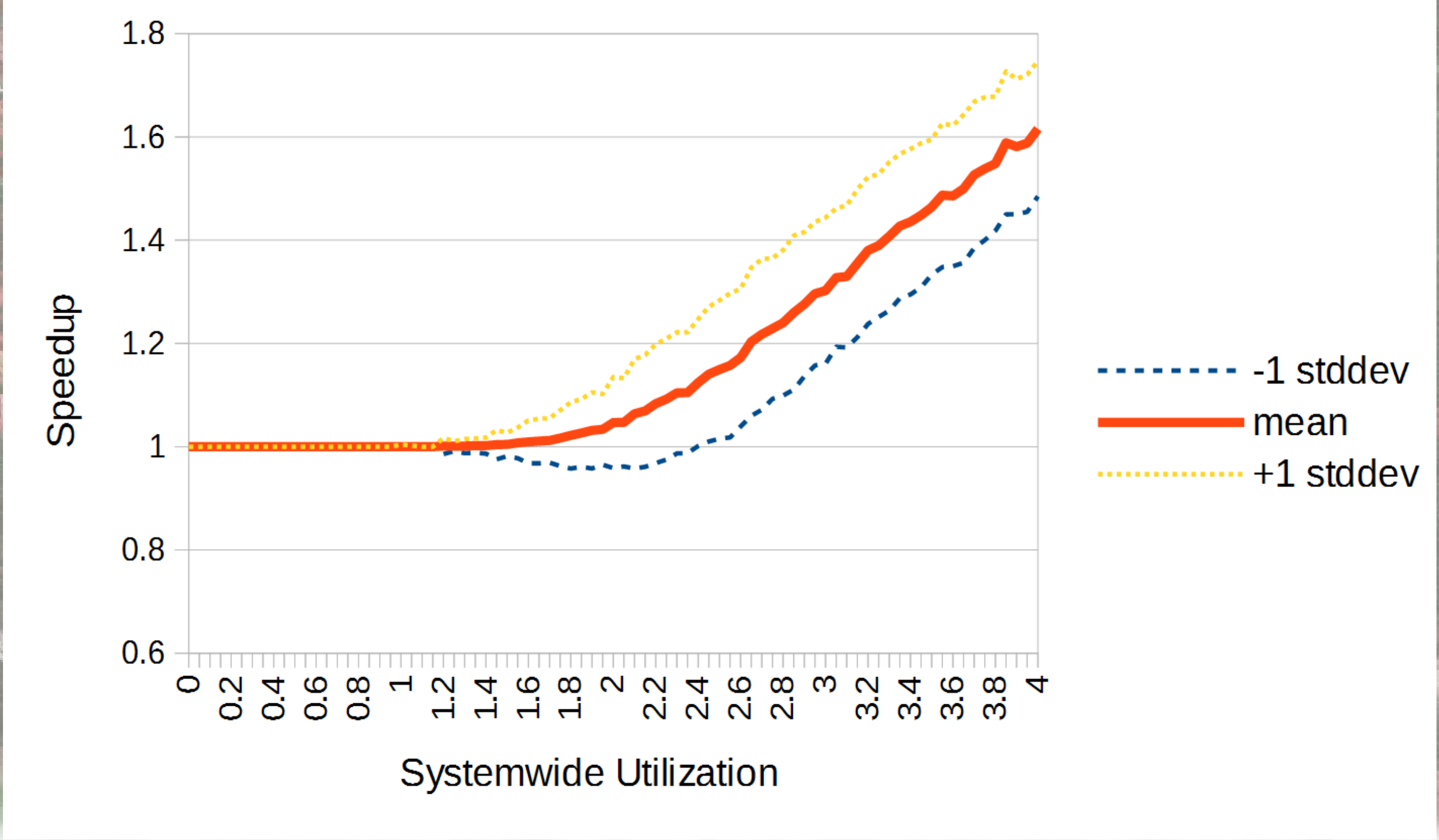
Task periods assigned uniformly and randomly over $F \times \{1, 2, 3, 4\}$ ($F$ = 25ms)

Task $C_i$ were determined as the product of utilization and period.

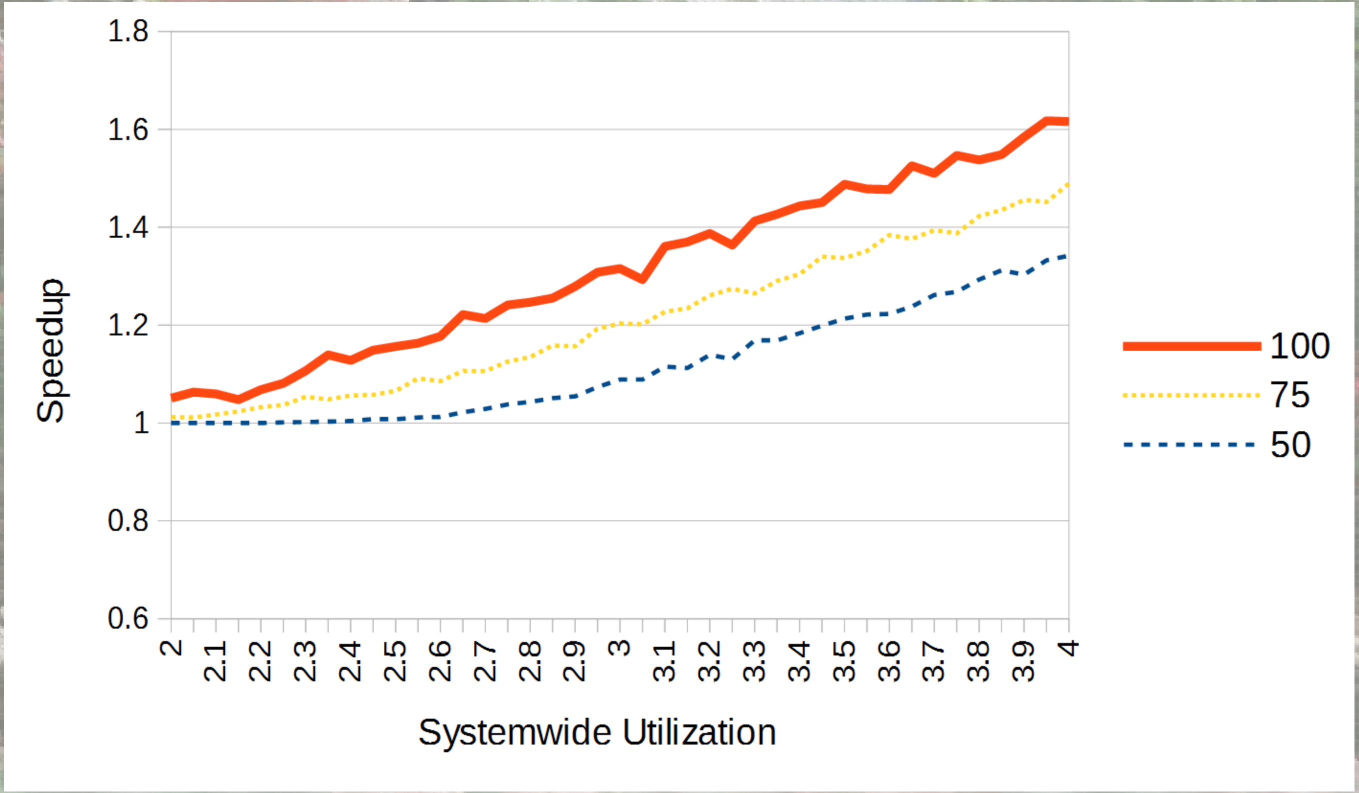
Systems with some $C_i$ greater than $F$ were discarded (as they are infeasible).

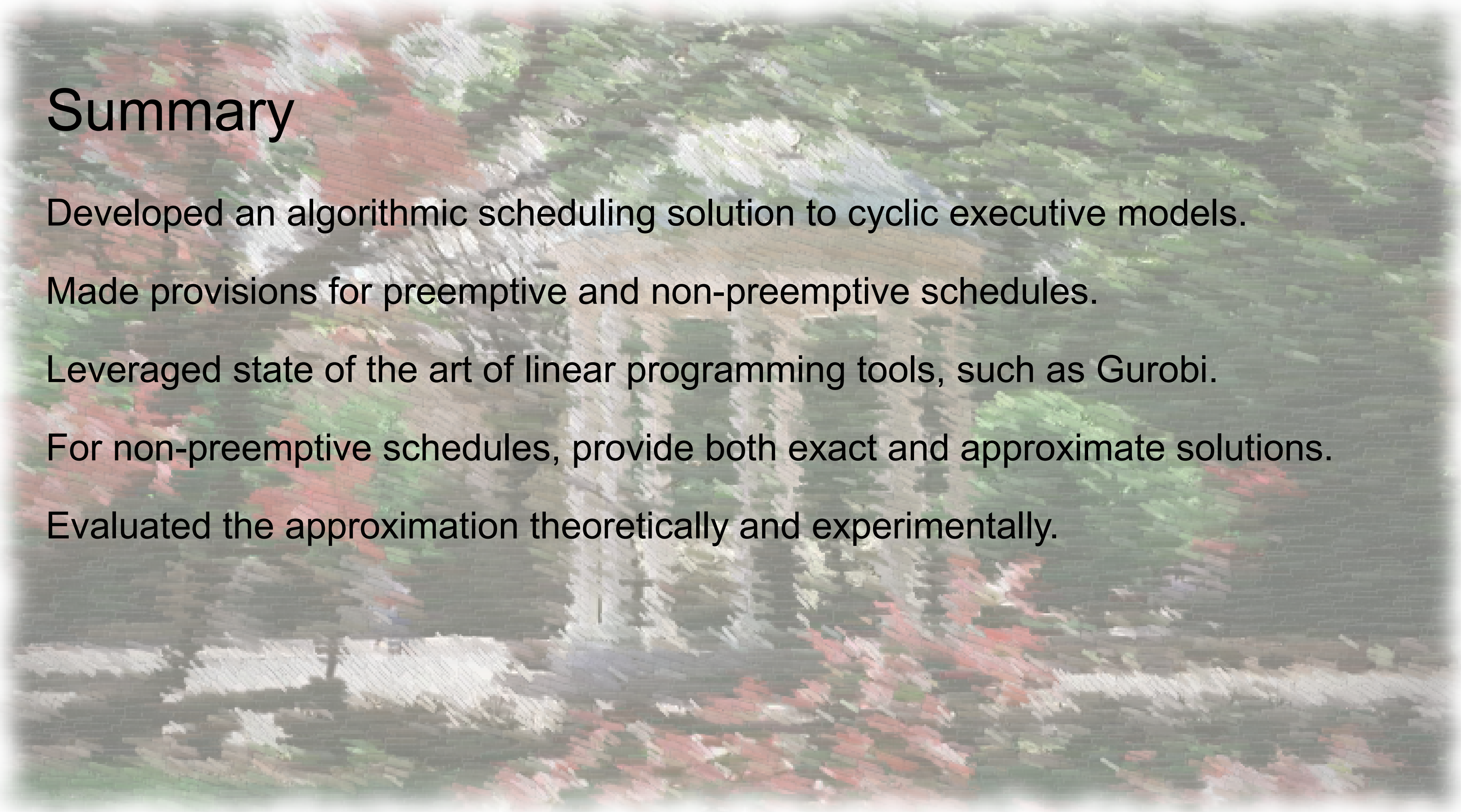
All the experiments assumed a four-processor platform ($m = 4$).

# Speedup Required for Approximation Schedule



## Mean Speedup with max $C_i$ bounded at %age of F





## Summary

Developed an algorithmic scheduling solution to cyclic executive models.

Made provisions for preemptive and non-preemptive schedules.

Leveraged state of the art of linear programming tools, such as Gurobi.

For non-preemptive schedules, provide both exact and approximate solutions.

Evaluated the approximation theoretically and experimentally.

## Questions?

Special thanks to my advisor Sanjoy Baruah, Real-Time Group, UNC, and NSF.