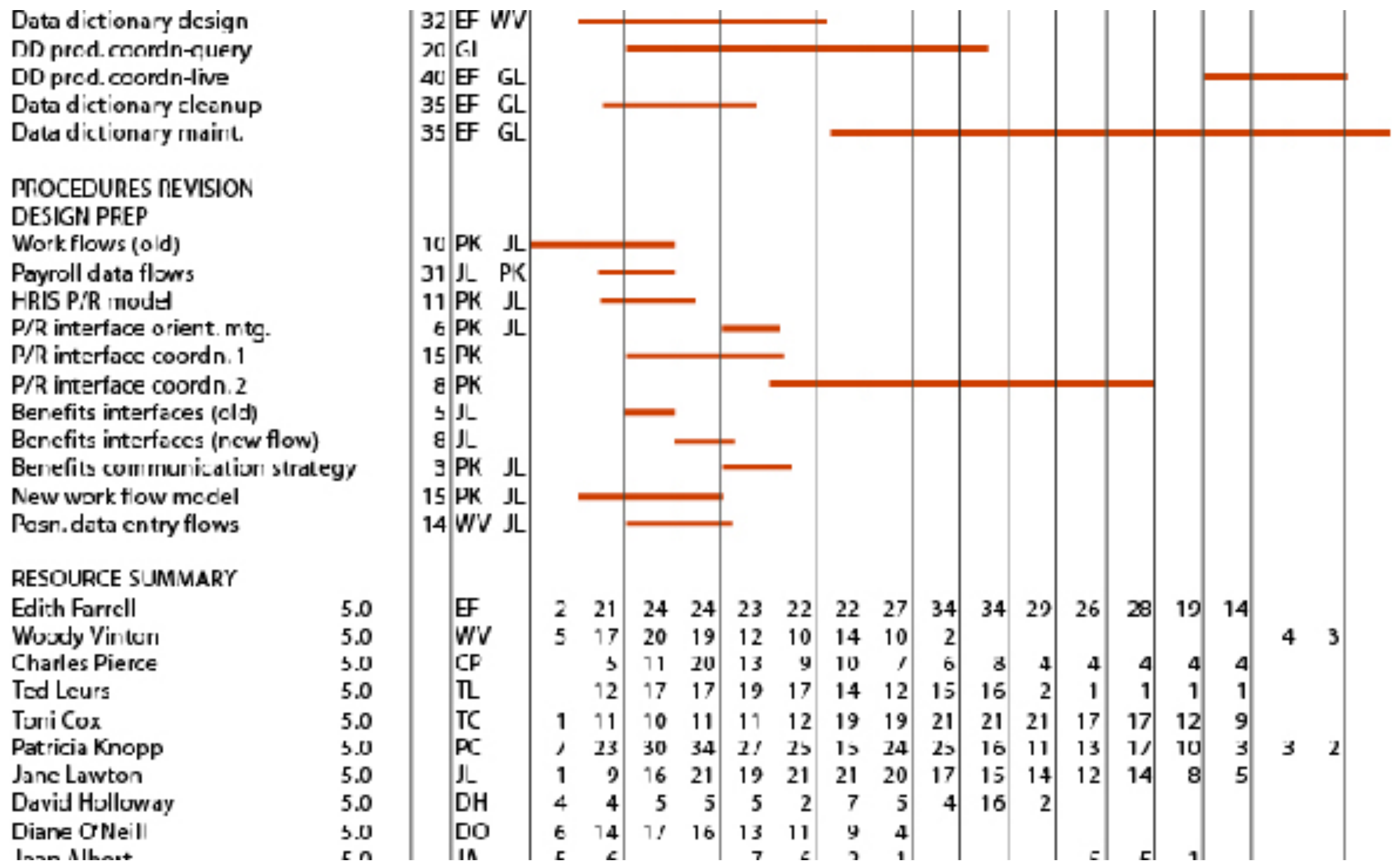
CO3091 - Computational Intelligence and Software Engineering

Lecture 10



Software Project Scheduling — Part I

Leandro L. Minku

Overview

- Part I:
  - What is the Software Project Scheduling Problem (SPSP)?
  - Why are automated methods important for the SPSP?
  - How to formulate the SPSP as an optimisation problem?
  - How to solve the SPSP using optimisation algorithms?
  - [Except constraints]
- Part II:
  - How to formulate the constraints of the SPSP?
  - How to deal with the constraints of the SPSP?

Announcements

- tail -1 logFile.tsv | cut -f 4 -d$'\t'
- tail -1 logFile.tsv | cut -f 4

Where Are We?

Optimisation Algorithms

Hill Climbing
Simulated Annealing
Evolutionary Algorithms

Understanding is important to be able to propose good designs for different applications

How to Apply for SE problems

Software Modularisation
Requirements Selection

Practice identifying optimisation problems and solving them using computational intelligence

Evaluation

Statistical Tests

Can help you to choose parameters and designs, and better understand their impact

Other Applications

Traveling Salesman Problem
Lorry Problem
Very Large Scale Integration
Wind Farm

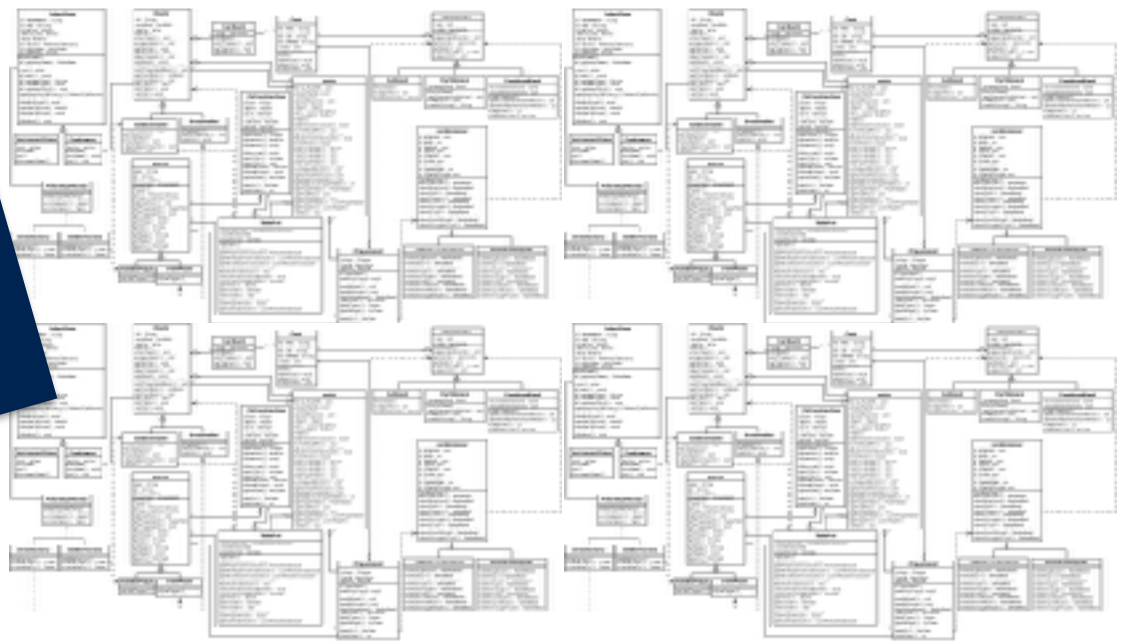
Practice identifying optimisation problems and solving them using computational intelligence

Where Are We Going?

- How to apply optimisation algorithms for:
  - Software project scheduling
  - Software energy optimisation
  - Software testing
- Other optimisation algorithms:
  - NSGA-II
  - Ant colony optimisation
- Machine learning
  - …

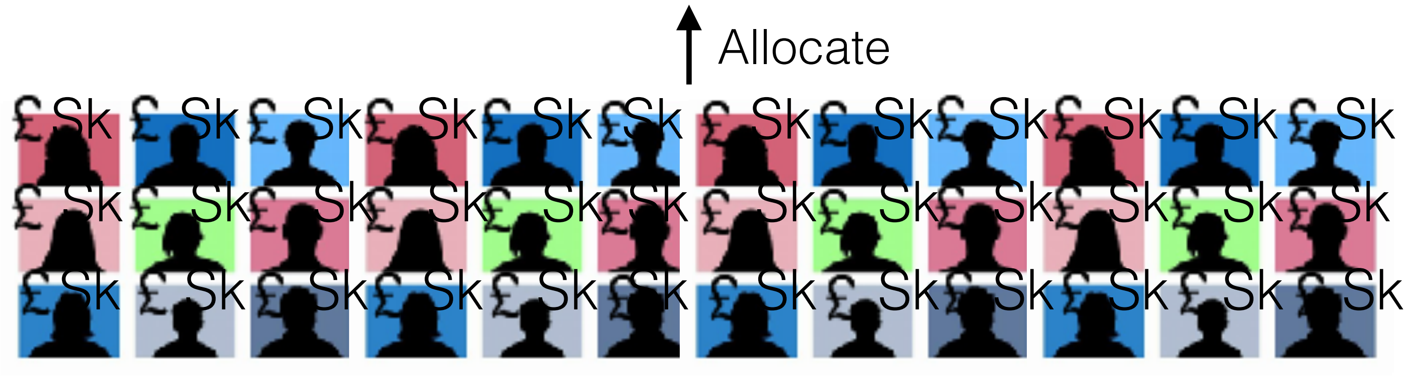
Software Project Scheduling Problem (SPSP)

Different allocations will lead to different project cost and duration.



Each task requires a set of skills and effort.

Tasks also have a precedence relation.



Software Project Scheduling Problem (SPSP)

SPSP: find a good allocation of employees to tasks in a software project so as to minimise its **cost** and **duration.**

It is very difficult to optimally assign employees to tasks manually.

- The space of possible allocations can be enormous.

We can use optimisation algorithms (e.g., EAs) to solve the SPSP!

Advantages of Optimisation Algorithms for the SPSP

- Insight into how to optimise objectives -- they may find solutions that no human has thought of.
- Speed up the task of allocating employees to tasks.
- Help software manager to find solutions that satisfy all constraints.
  - Team must have skills to perform a task.
  - No overwork is allowed.

Problem Formulation and Evolutionary Algorithm Design

Formulating the SPSP

Setting: assume we are given

- n employees $e_1, \ldots, e_n$ with salaries sal_i and sets of skills $skill_i$;
- m tasks $t_1, \ldots, t_m$ with required efforts $reqEff_j$ and sets of required skills $reqSk_j$;
- a task precedence graph (TPG).

Problem: allocate employees to tasks so as to:

- minimise cost (total salaries paid) and
- minimise duration (completion time).

Constraints:

- team must have required skills and
- no overwork.

For the purpose of today's lecture, let's pretend that there are no constraints.

Formulation — Design Variable

- We have *n* employees and *m* tasks.
- We need to assign employees to tasks.
- An employee can be assigned to more than one task.
- Design variable:
    - **x’** = Matrix of employees by tasks.
    - Entries $x'_{i,j}$ represent the dedication of employee i to task j.
    - Dedication = percentage of the employee’s time dedicated to the task.
    - Which values could each entry $x'_{i,j}$ assume?
    - Depends on the granularity *k* of the problem.

x’	**t_1**	**t_2**	**…**	**t_m**
e_1	$x'_{1,1}$	$x'_{1,2}$	…	$x'_{1,m}$
e_2	$x'_{2,1}$	$x'_{2,2}$	…	$x'_{2,m}$
…	…	…	…	…
e_n	$x'_{n,1}$	$x'_{n,2}$	…	$x'_{n,m}$

$$x'_{i,j} \in \left\{ \frac{0}{k}, \frac{1}{k}, \frac{2}{k}, \ldots, \frac{k}{k} \right\}$$

EA Design — Representation

- Representation: **x** = matrix of dedications of employees to tasks, where dedications are integers in {0,…,*k*}.

x	**t_1**	**t_2**	**…**	**t_m**
e_1	$x_{1,1}$	$x_{1,2}$	…	$x_{1,m}$
e_2	$x_{2,1}$	$x_{2,2}$	…	$x_{2,m}$
…	…	…	…	…
e_n	$x_{n,1}$	$x_{n,2}$	…	$x_{n,m}$

$x_{i,j} \in \{0,\dots,k\}$

EA Design — Mutation

Mutation of $x_{i,j}$ picks a new dedication uniformly at random from:

$$\{0,\ldots,k\} \setminus x_{i,j}$$

Can you propose a better mutation operator?

EA Design — Crossover

Exchange rows

x	t_1	t_2	t_3	t_4
e_1	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$
e_2	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$
e_3	$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,4}$

$\longrightarrow$

x	t_1	t_2	t_3	t_4
e_1	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$
e_2	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$
e_3	$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,4}$

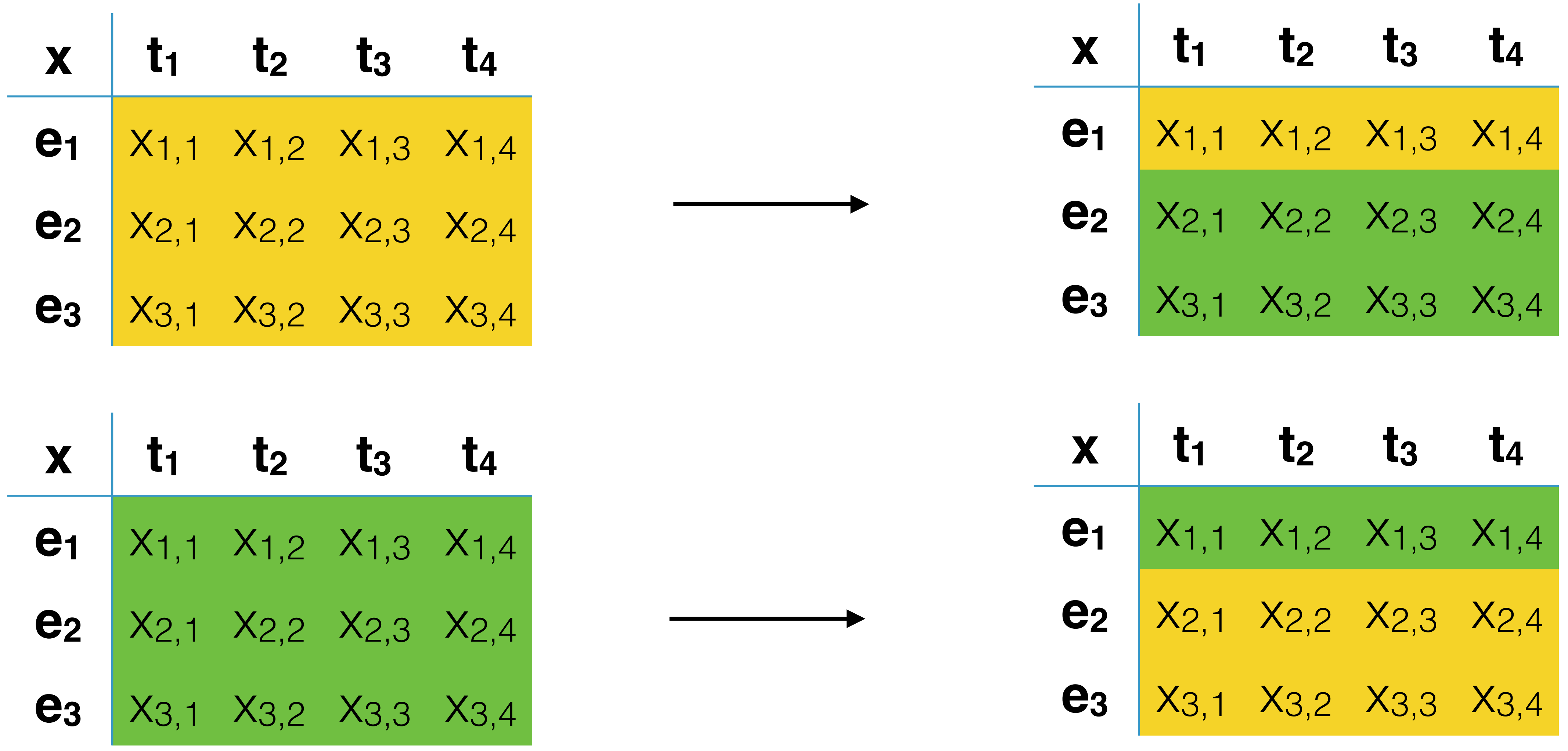
x	t_1	t_2	t_3	t_4
e_1	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$
e_2	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$
e_3	$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,4}$

$\longrightarrow$

x	t_1	t_2	t_3	t_4
e_1	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$
e_2	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$
e_3	$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,4}$

EA Design — Crossover

Exchange rows



x	t_1	t_2	t_3	t_4
e_1	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$
e_2	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$
e_3	$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,4}$

$\longrightarrow$

x	t_1	t_2	t_3	t_4
e_1	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$
e_2	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$
e_3	$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,4}$

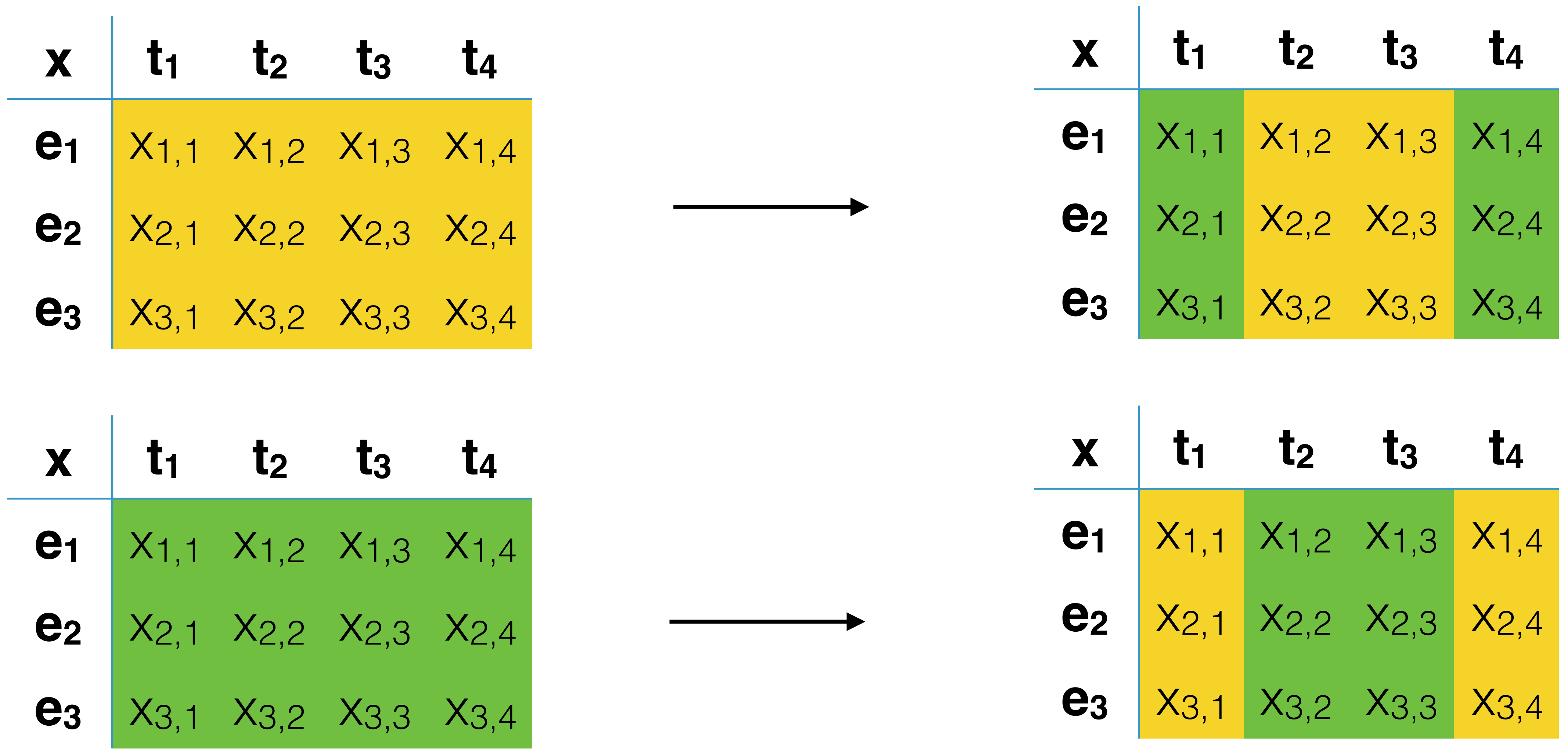
x	t_1	t_2	t_3	t_4
e_1	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$
e_2	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$
e_3	$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,4}$

$\longrightarrow$

x	t_1	t_2	t_3	t_4
e_1	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$
e_2	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$
e_3	$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,4}$

EA Design — Crossover

Exchange columns



x	t_1	t_2	t_3	t_4
e_1	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$
e_2	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$
e_3	$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,4}$

$\longrightarrow$

x	t_1	t_2	t_3	t_4
e_1	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$
e_2	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$
e_3	$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,4}$

x	t_1	t_2	t_3	t_4
e_1	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$
e_2	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$
e_3	$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,4}$

$\longrightarrow$

x	t_1	t_2	t_3	t_4
e_1	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$
e_2	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$
e_3	$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,4}$

EA Design — Crossover

- Which is better: rows or columns?
    - We don't know!
    - We can randomly decide which of them to apply whenever we need to perform crossover.

Formulation — Objective Functions

- Problem: allocate employees to tasks so as to:
  - minimise cost (total salaries paid) and
  - minimise duration (completion time).
- Objectives:
  - cost (total salaries paid) and
  - duration (completion time).

How to calculate cost and duration?

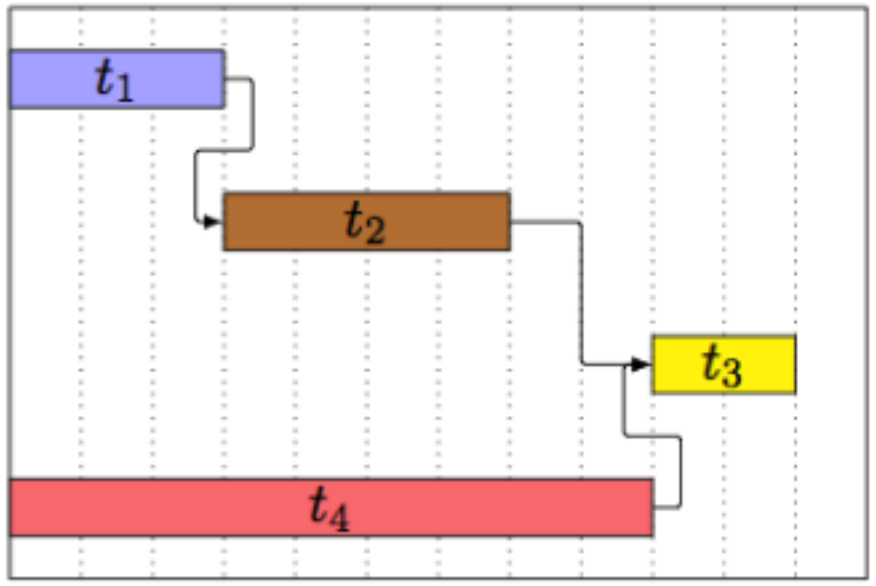
Calculating Cost and Duration

x	**t₁**	**t₂**	**t₃**	**t₄**
e₁	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$
e₂	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$
e₃	$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,4}$

$\xrightarrow{\text{decode}}$

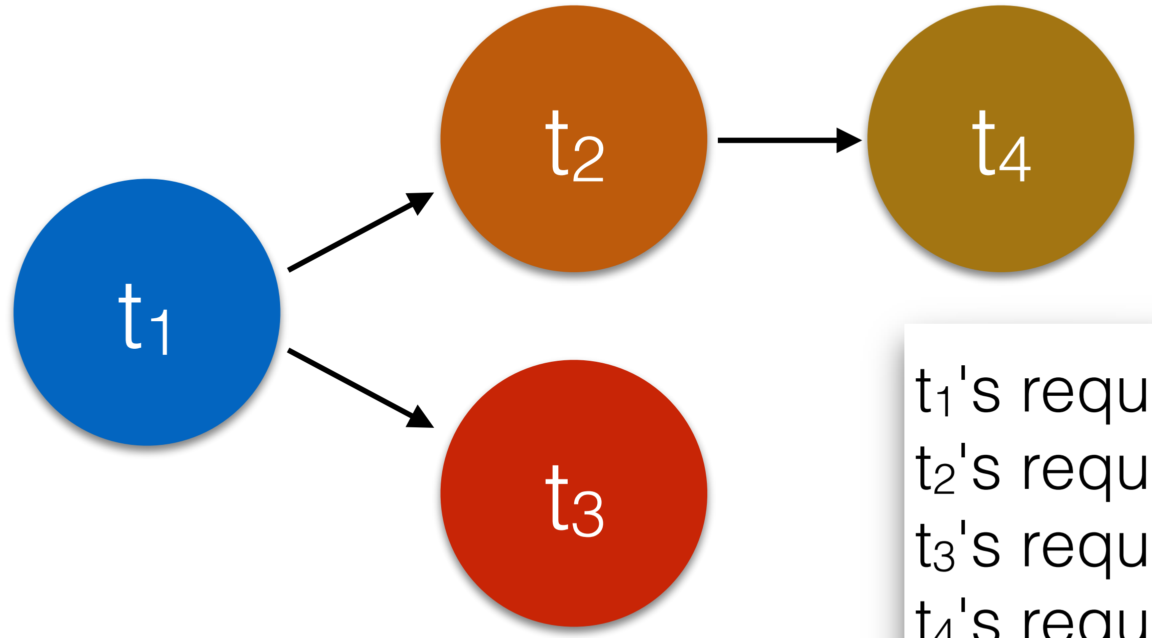
x'	**t₁**	**t₂**	**t₃**	**t₄**
e₁	$x'_{1,1}$	$x'_{1,2}$	$x'_{1,3}$	$x'_{1,4}$
e₂	$x'_{2,1}$	$x'_{2,2}$	$x'_{2,3}$	$x'_{2,4}$
e₃	$x'_{3,1}$	$x'_{3,2}$	$x'_{3,3}$	$x'_{3,4}$

$\downarrow$ + TPG, tasks required efforts



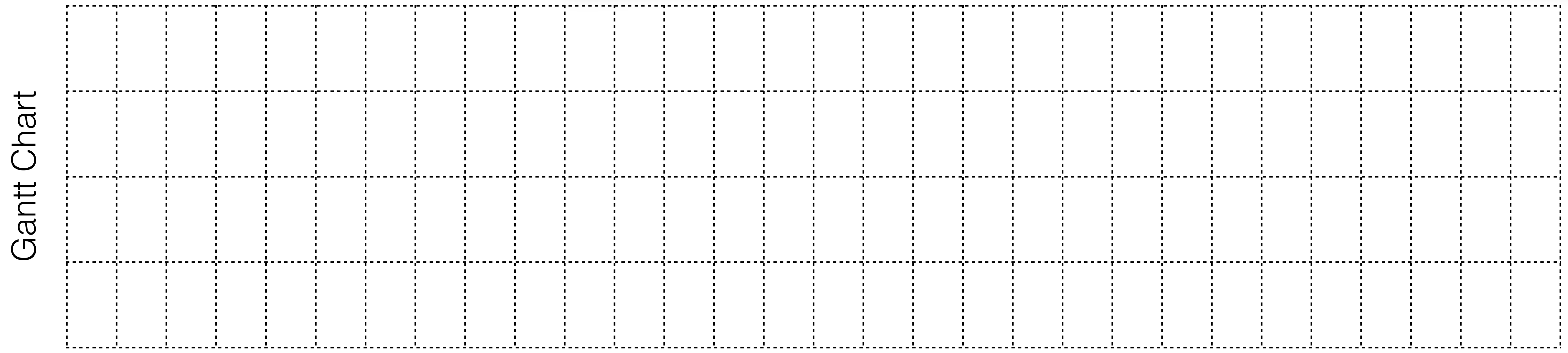
Cost and duration $\xleftarrow{\text{+ salaries}}$

Example for 1 Employee, 4 Tasks, k=2

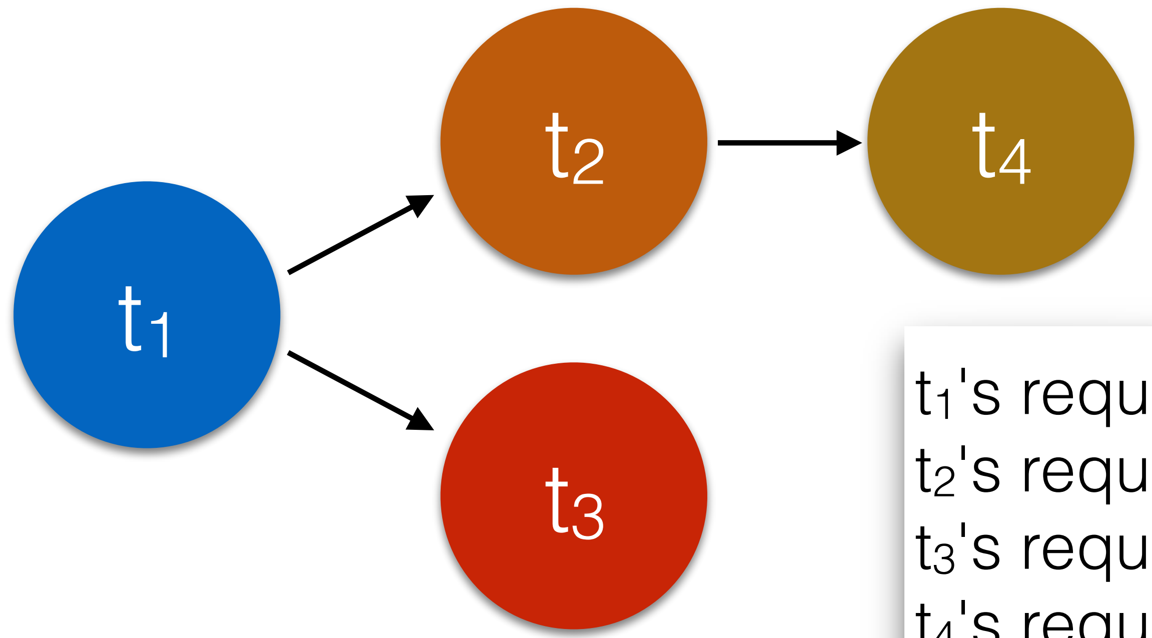


x	t_1	t_2	t_3	t_4
e_1	1	1	1	1

t_1's required effort and skills: 4 p-month, {sql, java}
t_2's required effort and skills: 4 p-month, {java}
t_3's required effort and skills: 8 p-month, {java}
t_4's required effort and skills: 2 p-month, {java}
e_1's salary and skills: $1000 per full time month, {sql, java}



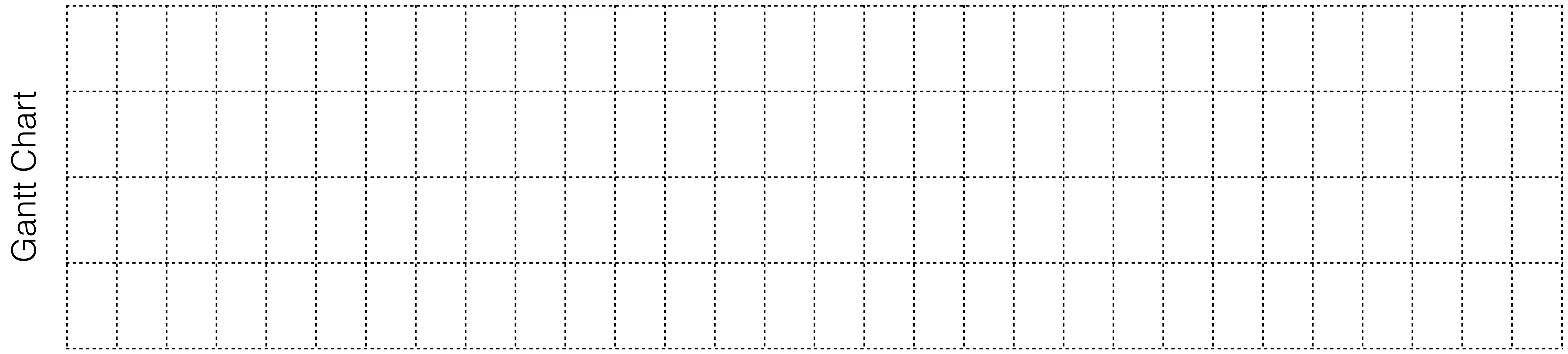
Example for 1 Employee, 4 Tasks, k=2



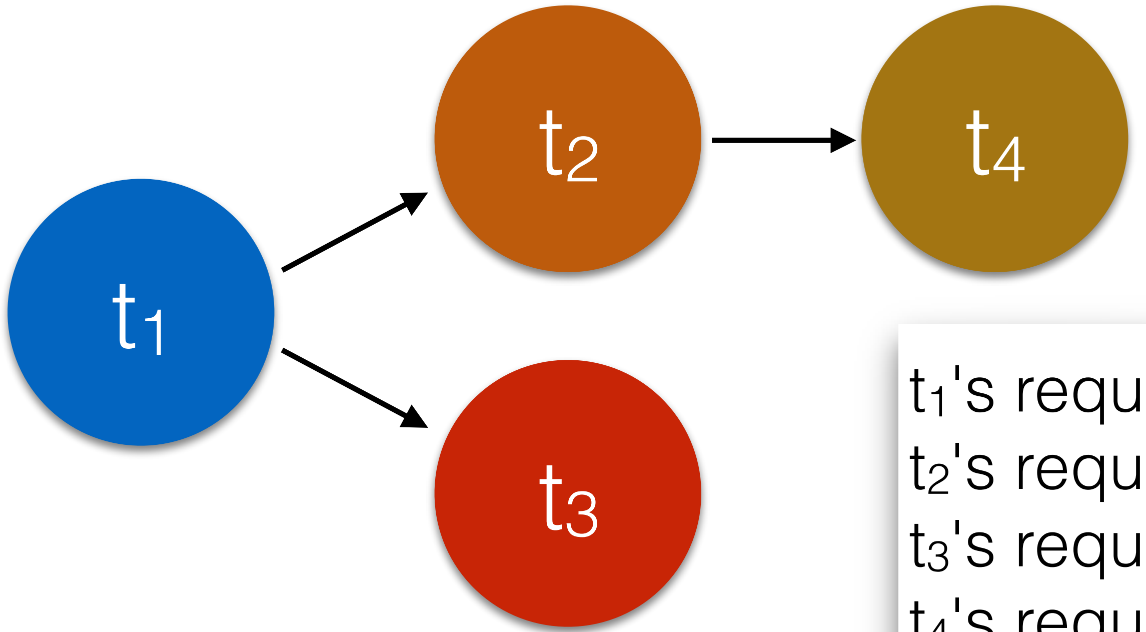
$$\begin{tikzcd} & t_2 \arrow[r] & t_4 \\ t_1 \arrow[ur] \arrow[dr] & & \\ & t_3 & \end{tikzcd}$$

x'	t_1	t_2	t_3	t_4
e_1	0.5	0.5	0.5	0.5

t_1's required effort and skills: 4 p-month, {sql, java}
t_2's required effort and skills: 4 p-month, {java}
t_3's required effort and skills: 8 p-month, {java}
t_4's required effort and skills: 2 p-month, {java}
e_1's salary and skills: \$1000 per full time month, {sql, java}

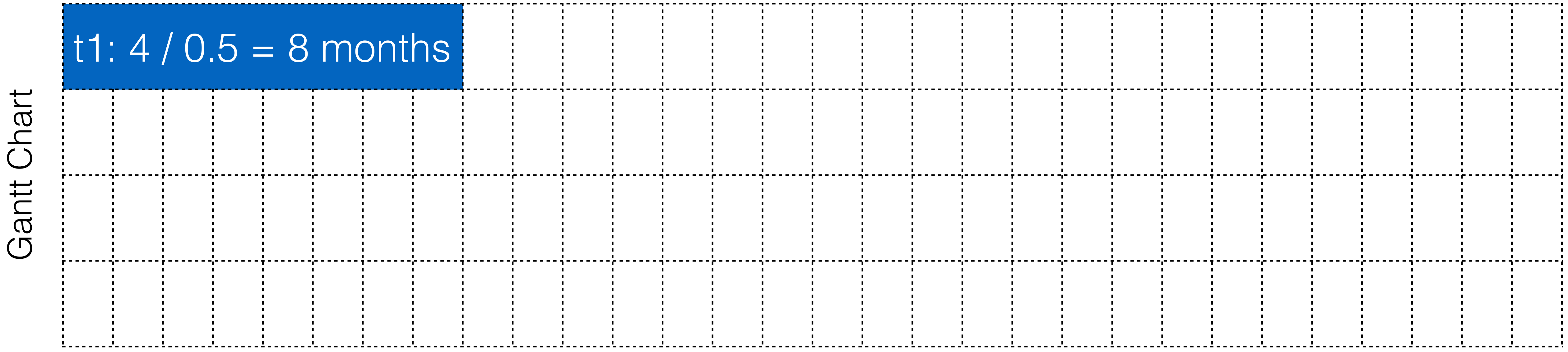


Example for 1 Employee, 4 Tasks, k=2

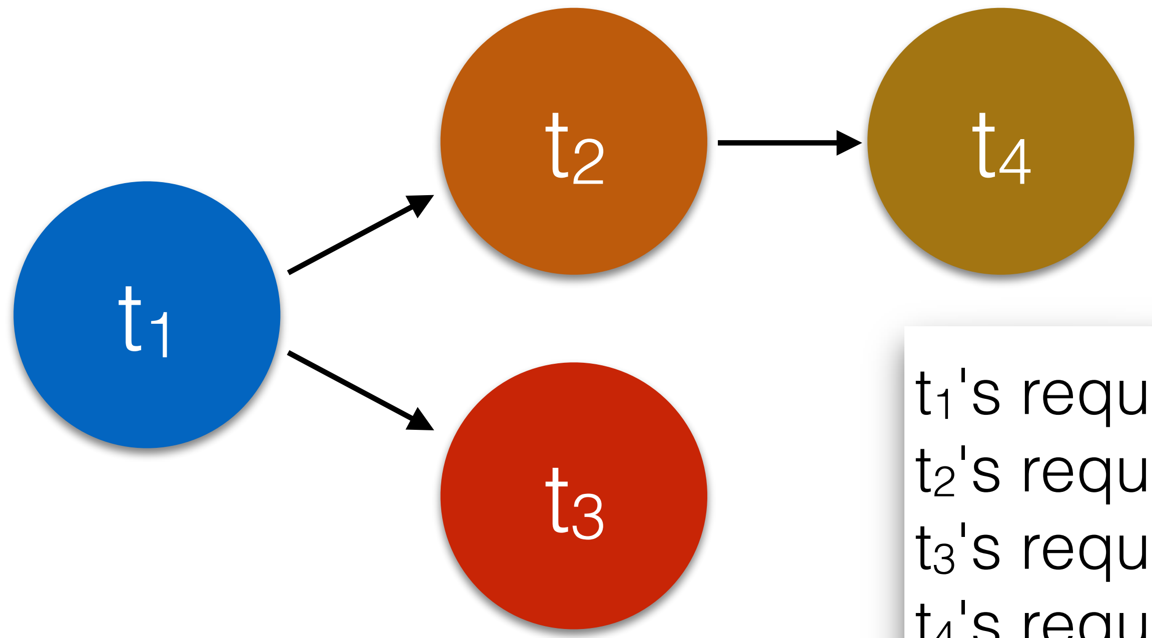

$$\begin{tikzcd} & t_2 \arrow[r] & t_4 \\ t_1 \arrow[ur] \arrow[dr] & & \\ & t_3 & \end{tikzcd}$$

x'	**t_1**	**t_2**	**t_3**	**t_4**
e_1	0.5	0.5	0.5	0.5

t_1's required effort and skills: 4 p-month, {sql, java}
t_2's required effort and skills: 4 p-month, {java}
t_3's required effort and skills: 8 p-month, {java}
t_4's required effort and skills: 2 p-month, {java}
e_1's salary and skills: $1000 per full time month, {sql, java}

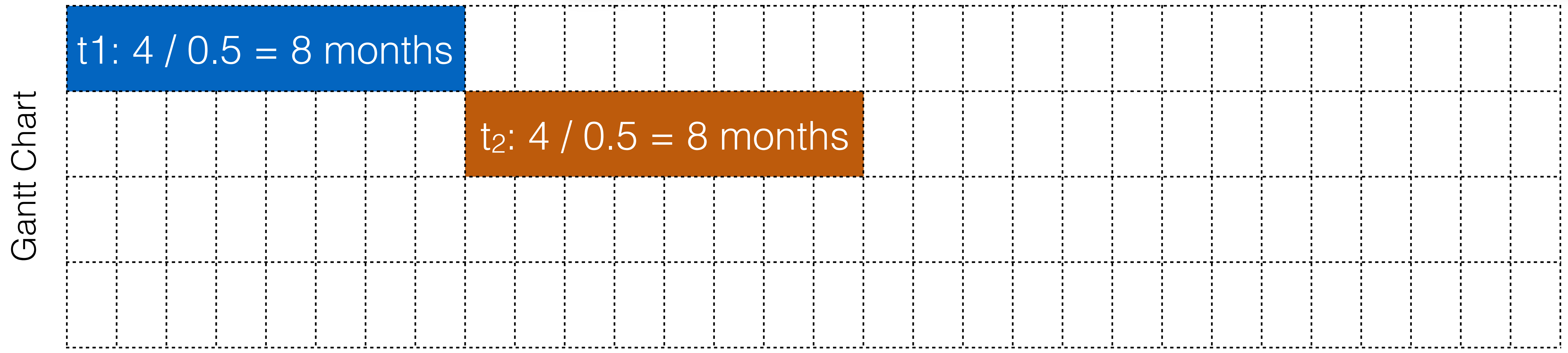


Example for 1 Employee, 4 Tasks, k=2

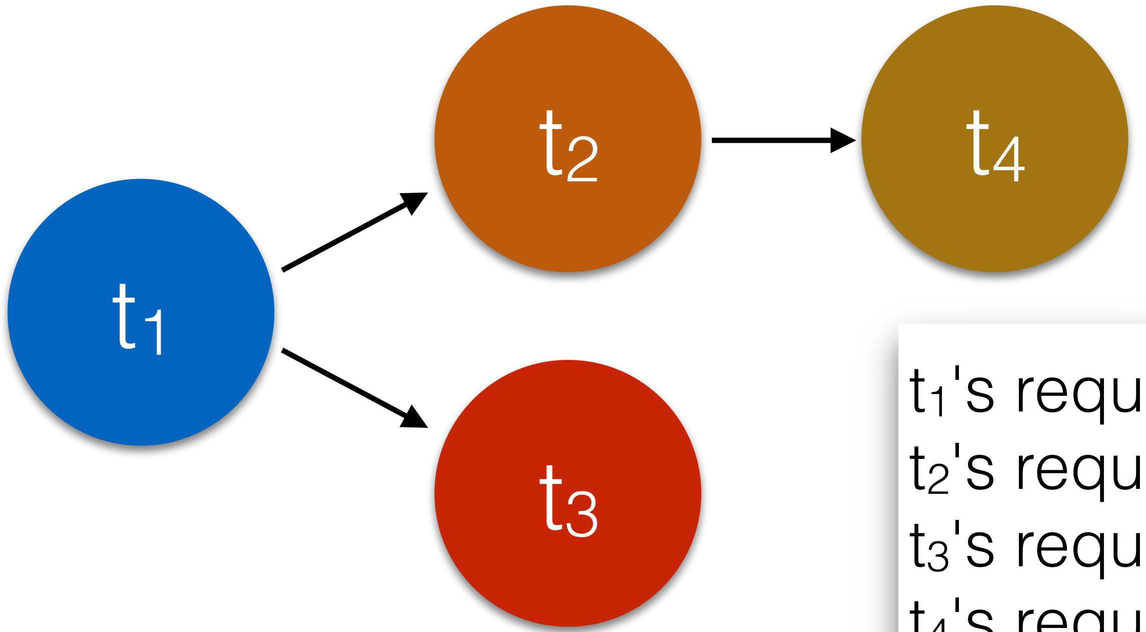


x'	t_1	t_2	t_3	t_4
e_1	0.5	0.5	0.5	0.5

t_1's required effort and skills: 4 p-month, {sql, java}
t_2's required effort and skills: 4 p-month, {java}
t_3's required effort and skills: 8 p-month, {java}
t_4's required effort and skills: 2 p-month, {java}
e_1's salary and skills: $1000 per full time month, {sql, java}

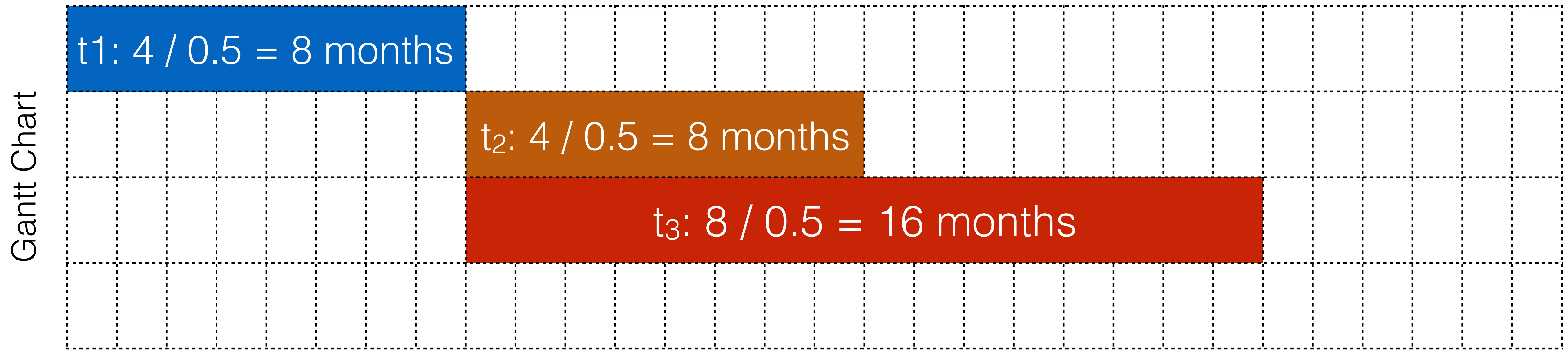


Example for 1 Employee, 4 Tasks, k=2

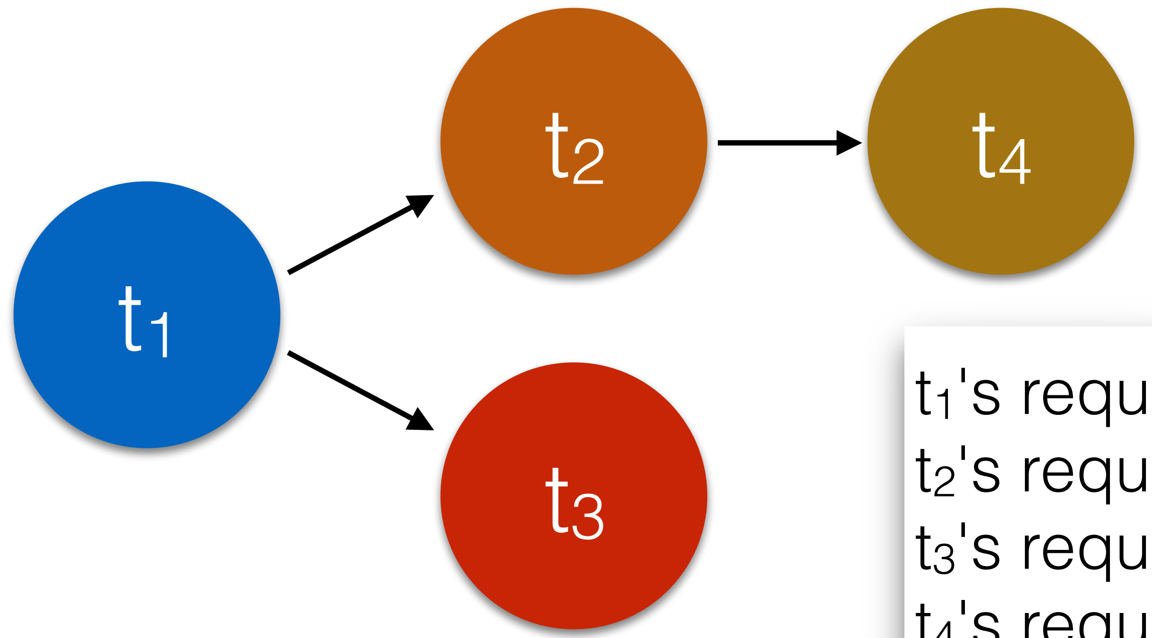


x'	t_1	t_2	t_3	t_4
e_1	0.5	0.5	0.5	0.5

t_1's required effort and skills: 4 p-month, {sql, java}
t_2's required effort and skills: 4 p-month, {java}
t_3's required effort and skills: 8 p-month, {java}
t_4's required effort and skills: 2 p-month, {java}
e_1's salary and skills: \$1000 per full time month, {sql, java}

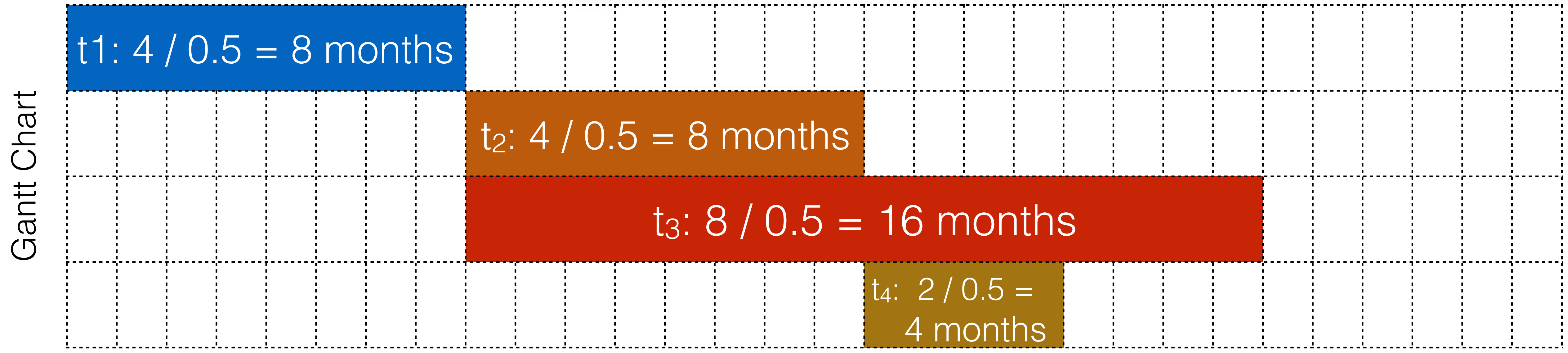


Example for 1 Employee, 4 Tasks, k=2



x'	**t_1**	**t_2**	**t_3**	**t_4**
e_1	0.5	0.5	0.5	0.5

t_1's required effort and skills: 4 p-month, {sql, java}
t_2's required effort and skills: 4 p-month, {java}
t_3's required effort and skills: 8 p-month, {java}
t_4's required effort and skills: 2 p-month, {java}
e_1's salary and skills: \$1000 per full time month, {sql, java}



EA Design — Fitness Function

$$\text{fitness}(\mathbf{x'}) = w_{cost} * \text{cost}(\mathbf{x'}) + w_{dur} * \text{duration}(\mathbf{x'})$$

(to be minimised)

w_{cost} and w_{dur} $\in [0,1]$

$w_{cost} + w_{dur} = 1$

EA Design — Parents and Survival Selection

- Parents selection: 2-Tournament Selection.
- Survivor selection: fitness-based delete-worst.
- Stoping criterion: maximum number of generations.

Summary

Summary of Problem Formulation

- Design variable: matrix of dedications of employees to tasks.
- Objectives: cost and duration (to be minimised).
- Constraints: ?

Summary of Evolutionary Algorithm Design

- Representation: integer matrix of employees by tasks.
- Fitness function: fitness(**x'**) = w_{cost} * cost(**x'**) + w_{dur} * duration(**x'**)
- Mutation: picks new dedication uniformly at random.
- Crossover: exchanges rows or columns.
- Parents selection: 2-Tournament selection.
- Survival selection: fitness-based delete-worst.
- Stoping criterion: maximum number of generations
- Constraints: ?

Further Reading

L. Minku, D. Sudholt and X. Yao. "Improved Evolutionary Algorithm Design for the Project Scheduling Problem Based on Runtime Analysis", IEEE Transactions on Software Engineering vol. 40, n. 1, p. 83-102, 2014. Read sections 1—3.

http://ieeexplore.ieee.org/document/6648326/