Rave II

Developed by
Jeppesen Crew Academy

for version 22 of Crew Pairing, Crew Rostering and Tail Assignment

Practical Details

Restrooms

Breaks

Phones

Wifi

Lunch arrangements

Quiz

Survey

Participants presentation

- –Name, company
- –Role
- –Experience
- –Your expectations
- –<Other>

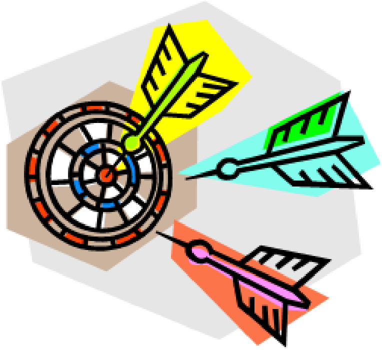
Course goals

This course will teach you:

- external tables and rules, more functionality
- rule exceptions
- how to use modules
- how to write cost functions
- constraints
- how caching works
- to avoid illegal sub-chain problems
- how to do performance analysis
- how to use accumulators.

You will know how to:

- update and maintain existing code
- implement new functionality with Rave
- consider performance issues**.**



Prerequisites

- Rave I course
- Min 6 months of real Rave programing experience
- Knowledge about the airline/rail business side.

Please –

Don't be afraid to ask questions if anything is unclear!

Course material

Course slides

Course manual

Online documentation

Code standards

Agenda Day 1

09:00 - 10:15	Recap of Rave I Tables
10:15 - 10:30	**Coffee break**
10:30 - 12:30	Modules Inheritance
12:30 - 13:30	**Lunch**
13:30 - 15:00	Contexts Iterators
15:00 - 15:15	**Coffee break**
15:15 - 17:00	Transforms Constraints

All times are approximate – changes may/will occur

Short breaks every ~40 minutes or so

Agenda Day 2

09:00 - 10:15	Performance
10:15 - 10:30	**Coffee break**
10:30 - 12:30	Caching
12:30 - 13:30	**Lunch**
13:30 - 15:00	Costs
	Rules
15:00 - 15:15	**Coffee break**
15:15 - 17:00	Accumulators
	Summary
	Evaluation

All times are approximate – changes may/will occur

Short breaks every ~40 minutes or so

Chapter 1

Recap Rave I

Recap of Rave I course

Constants

```
%briefing% = 0:45;
```

Parameters

```
%debriefing% = parameter 0:30
    remark ”d3: Length of debriefing: ”;
```

Variables

```
%block_time% = arrival - departure;
```

Functions

```
%work_start_with_offset%(Reltime off_set) =
    %start_time% + off_set;
```

Built-in functions

```
%required_rest_after_leg% =
    nmax(%block_time%, %min_rest%);
```

Avoid using other characters than a-z and ‘ _ ’, as first letter since they cannot be used in Python

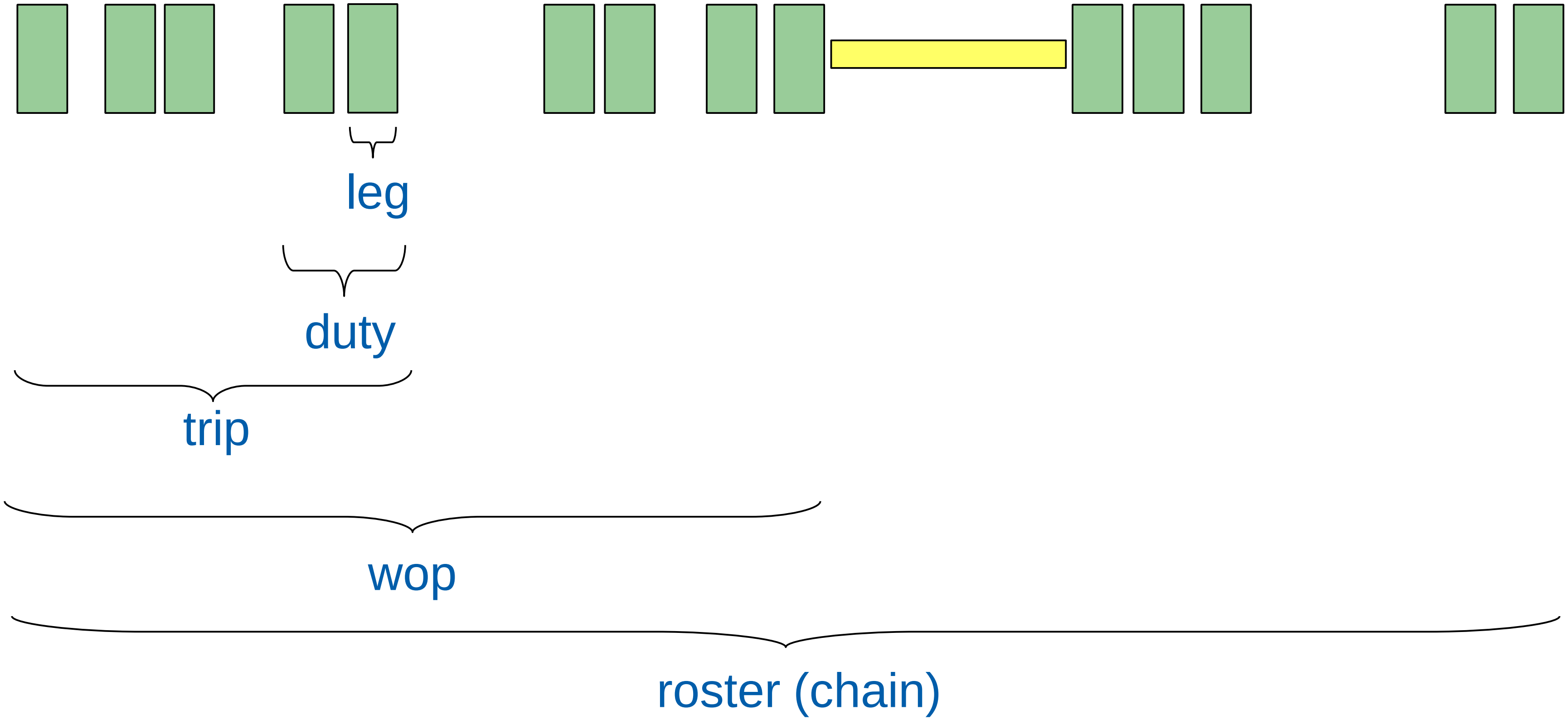
Recap of Rave I course

Levels

```
level trip =
    is_last( duty )
    when (%duty_arrival_name% = homebase);
end
```

Recap of Rave I course

Levels



Recap of Rave I course

Traversers & if-then-else

Traversers

```
%duty_block_time% =
    sum(leg(duty), %block_time%);
```

if-then-else

```
%cxn_time% =
    if is_last(leg(duty)) then
        0:00
    else
        next(leg(duty), departure)
        - arrival;
```

Recap of Rave I course

Tables (1)

```
%hotel_cost% =
    if %hotel% = “jerrys_inn” then
        250
    else if %hotel% = “plaza” then
        650
    else
        999;
```

```
%travel_time% =
    if %hotel% = “jerrys_inn” then
        0:30
    else if %hotel% = “plaza” then
        0:45
    else
        1:00;
```

Recap of Rave I course

Tables (2)

...could be simplified by using table lookups:

```
table hotel_costs_tab =
    %hotel%      -> %hotel_cost%, %travel_time%;
    ”jerrys_inn” -> 250,          0:30;
    ”plaza”      -> 650,          0:45;
     -           -> 999,          1:00;
end
```

Recap of Rave I course

External tables

Table definition:

```
table aircraft_family =
    aircraft_type -> String %aircraft_family%;
    external ”aircraft_family_file.etab”;
    ac_type -> ac_family;
    - -> ”no family”;
end
```

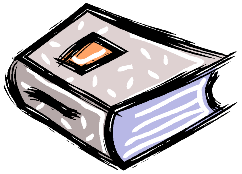
The “.etab” extension is optional; Rave will automatically append this to the table name if omitted. However, it is best practice to always include it.

Recap of Rave I course

External tables

Data file:

```
/*  Comment  */
2
Sac_type,
Sac_family,
"747", "747",
"74E", "747",
"72A", "727",
```



See
Help : Development > Rave Reference > Definitions > Tables
for more information about external tables

Recap of Rave I course

Sets and filters

Set

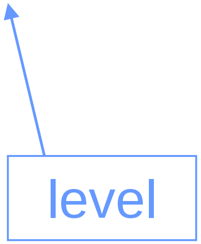
```
set asian_airports =
    [parameter] "BKK", " SIN", " HKG", " PEK", " NRT";
```

Set usage

```
%is_asian_airport% =
    arrival_airport_name in asian_airports;
```

Filters

```
filter active_legs = leg(not deadhead);
```



level

Recap of Rave I course

Rules

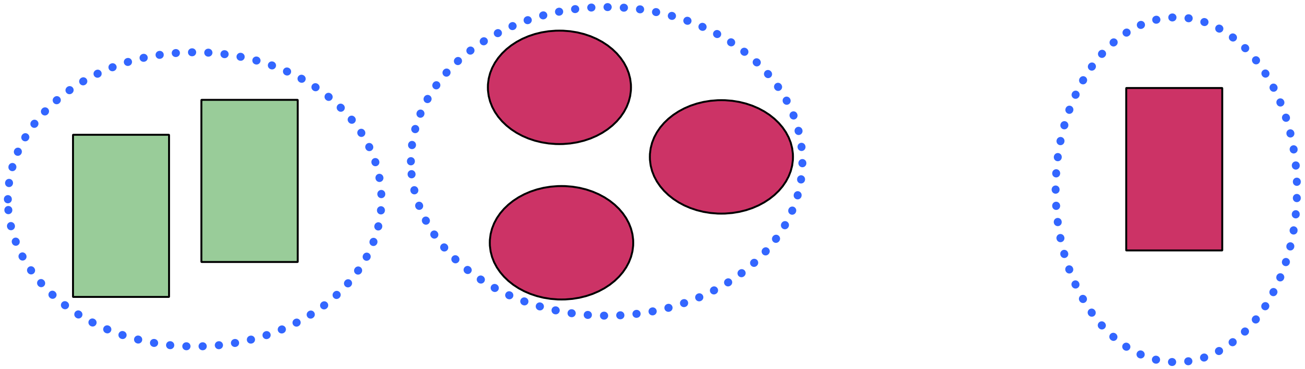
Rule

```
rule min_rest_in_trip =
    valid %shorthaul%;
    %trip_rest_time% >= %min_trip_rest_time%;
    remark "H3: Min rest in trip";
end
```

Recap of Rave I course

Iterators

- Divide groups of objects into smaller groups / bags
- Objects with identical values are put in the same bag



Recap of Rave I course

Iterators

```
/* Put each object in a separate bag */
iterator leg_set =
    partition(leg)
end
iterator trip_set =
    partition(trip)
end
```

Recap of Rave I course

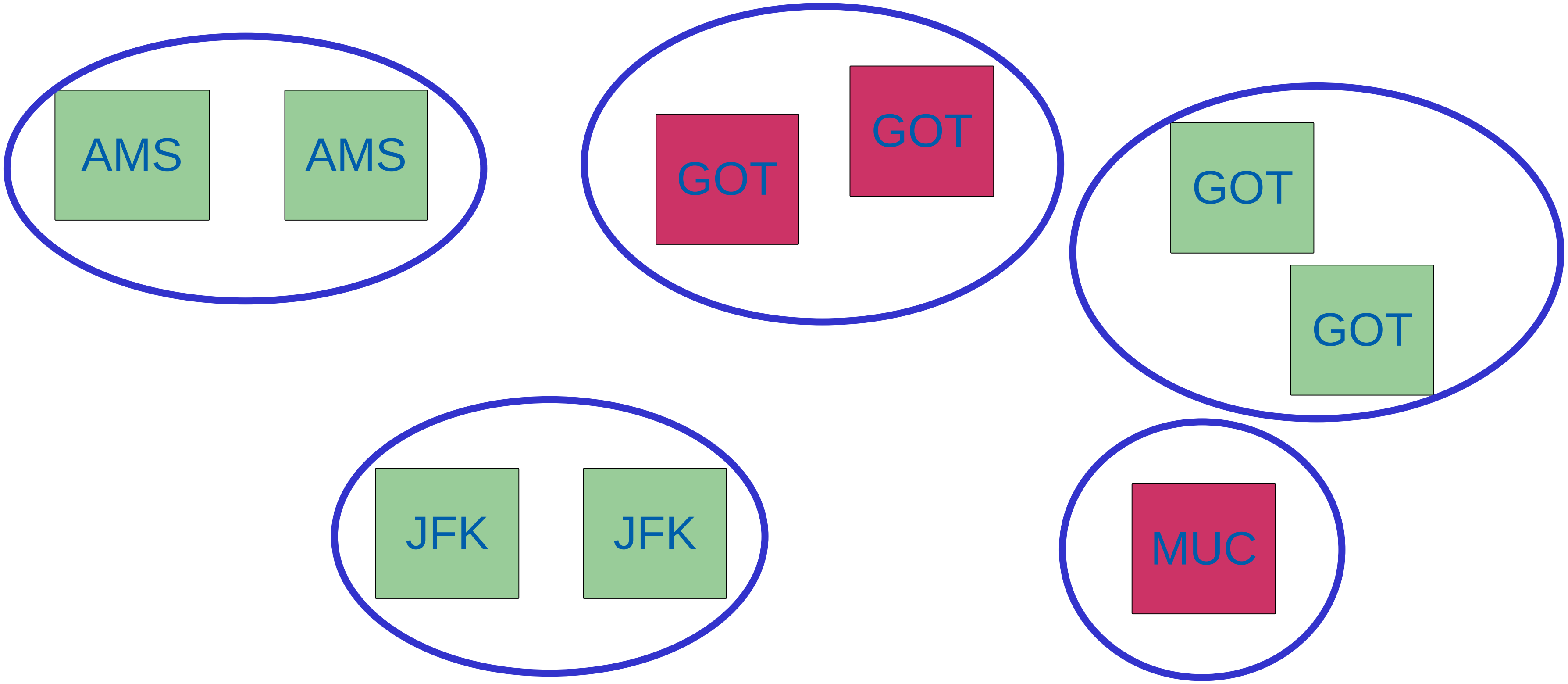
Iterator example

Iterator

```
iterator dep_port_set =
    partition(leg)
    by (deadhead, departure_airport_name);
end
iterator rave_expr_set =
    partition(leg)
    by(%any_rave_expression%);
end
```

Recap of Rave I course

Iterator example

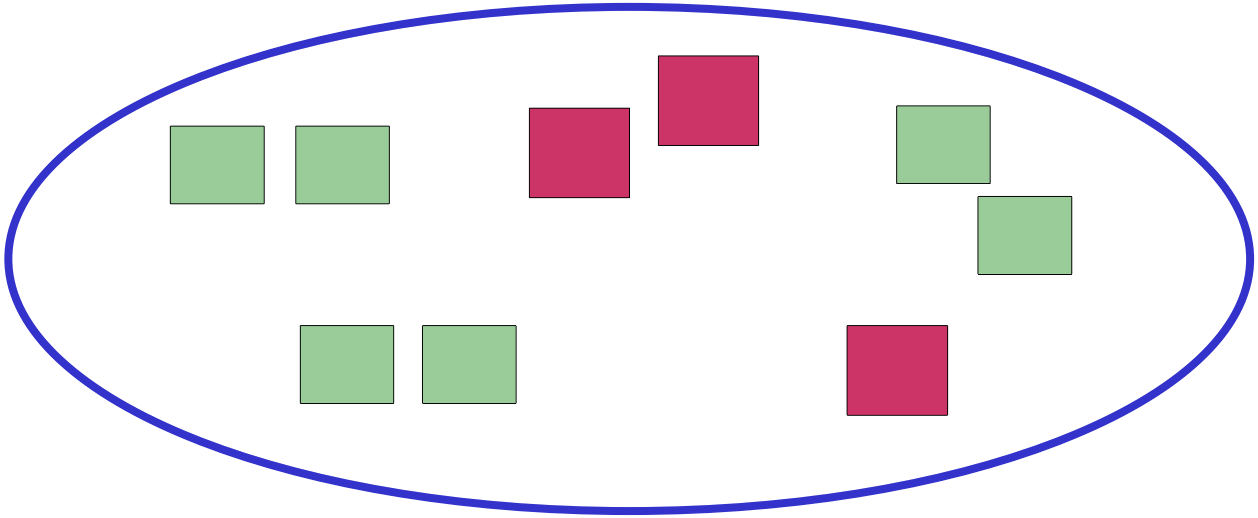


Recap of Rave I course

Iterator example

Iterator

```
iterator all_in_one =
    partition(leg)
    by(True);
end
```



Recap of Rave I course

Iterator example

Rave code

```
%nr_dep_ports% =
    count(dep_port_set) where (not deadhead);

%num_legs% = count(leg_set);
```

PRT code

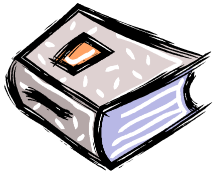
```
for leg_bag in ctx_bag.iterators.dep_port_set():
```

PDL code

```
Repeat foreach dep_port_set;
```

Built-in iterators

```
times
atom_set
```



See
Rave Reference > Expressions > Iterators and contexts
for more information about build in iterators

Chapter 2

Tables

Remarks

Tables

Internal tables

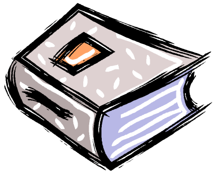
- Clusters

External tables

- Special field names
- Aggregation of values in result rows
- Intervals
- Examples

Performance

See *Rave Reference Manual, Definitions, Tables*



See
Rave Reference > Definitions> Tables
for more information about internal and external tables

Internal tables – clusters

Clusters:

- a basic internal table contains only one cluster
- within one cluster, the rows must be written in such a way that no condition value (data row) can cause two table rows to trigger
- the exception is the default row, which matches anything
- the order of table rows (within one cluster) is not important
- clusters are separated with the **&** (ampersand) character.

Internal tables – clusters

This is not allowed:

```
table ac_change_tab =
    deadhead, arrival -> %ac_change_ok%;
    y, “CPH”          -> false;
    y, -              -> true;
    n, “STO”          -> true;
    n, -              -> false;
end
```

This **is** allowed:

```
table ac_change_tab =
    deadhead, arrival -> %ac_change_ok%;
    y,  “CPH”         -> false;
    n,  “STO”         -> true;
    &
    y,   -            -> true;
    n,   -            -> false;
end
```

Tables

External tables – special field names

`row_number`

- A virtual column enumerating all rows
- May be used both as match- and result column
- e.g. Return the 3rd row of the table

`match_number`

- Enumerates the rows that match the current condition
- May only be used as a match column
- May not be used in a range or together with a relational operator
- e.g. Return the 5th row that matches condition

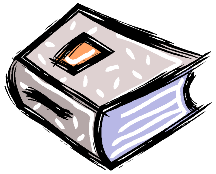
Tables

External tables – aggregation

- Aggregation of values in result rows
- You may use traversers such as `sum` or `last` if you want to produce an aggregated result from all rows that match a pattern
- No matching rows will result in the values `0` (zero) or `Void`.

Example: Summarize bid points and count number of bids

```
table bidcount_tab(String CrewId)=
    CrewId -> int %tot_bid_points%, int %bidcount%;
    external %bidfile%;
    crew_id -> sum(bid_points), count(bid_points);
end
```



See
Rave Reference > Definitions> Tables > Aggregation of values in result rows

Tables

External tables – intervals

```
table airp_layover=
    %layover%, duty.%end%, duty.%end% -> %allowed%;
    external %etable%;
    station, >= from_date, < to_date -> %is_allowed%;
end
```

Is equal to:

```
table airp_layover=
    %layover%, duty.%end% -> %allowed%;
    external %etable%;
    station , (from_date,to_date( -> %is_allowed%;
end
```

Tables

External tables – examples

- Convert M8[A-Z]* to M81 and 73[A-Z]* to 737
- Keep all other strings intact

from	to	result
M8A	M8ZZZ	M81
73A	73ZZZ	737

```
table stringlookup(String s)=
    s -> string %convert%;
    external %etab%;
    ("from", "to") -> "result";
    - -> s;
end
```

Warning: May affect performance negatively, more about that later.

Tables

External tables – examples

Flight notes on days of the week

```
table flight_note_tab =
    %flight_key%, %mon_key%, %tue_key%, ... %sun_key%
    -> string %flight_note%;
    external ”FlightNote.etab”;
    FlightKey, <=mon, <=tue, ... <=sun
    -> FlightNote;
    -,-,-,-,-,-,-,- -> ”No Note”;
end
%mon_key% = if flight-on-monday then 1 else 0;

”CA123”, 1,1,1,0,0,0,0, ”Service Staff”;
”CA456”, 0,0,0,0,1,0,0, ”Casual Friday”;
”CA333”, 0,0,0,0,0,1,1, ”Weekend Special”;
```

Self Study

Tables

External tables – examples

’Wildcards’ in internal tables:

```
table internal =
    %integer%, %boolean% -> %string%;
    10, True -> ”A”;
    12, True -> ”B”;
    &
    -, True -> ”C”;
end
```

’Wildcards’ in external tables:

- Reserve -1 and treat differently
- Define a new lookup not considering `%integer%` at all
- Use ’from’ and ’to’ columns and interval matching...

Tables

External tables – examples

External: ’from’ and ’to’ columns

```
table external =
     %integer%, %boolean% -> String %string%;
     external ”file.etab”;
     (int_col_from, int_col_to), bool_col -> str_col;
end
```

File.etab:

```
4
Iint_col_from,
Iint_col_to,
Bbool_col,
Sstr_col,
10,10, True, ”A”;
12,12, True, ”B”;
0,99, True, ”C”;
```

Planners have full control over match order, and full responsibility.

Tables

External tables – database

- Database tables always have to have a well defined unique key
  - add sequence number if no other key is available
    ➔ rows cannot be sorted top-down as with E-tables

- Possible to define dependencies to other tables
  - e.g. Departure airport name needs to exist in airport table

- Empty strings cannot be stored in database tables
  - they will be treated as ’void’ values

Remarks

Remark

Planner remark

Failtext

…

Remark, planner remark

```
%p% =
    parameter 8:00
    remark ”This is what p will do!”,
    planner ”This is what will show up when F1
    is pressed in the parameter form. There is
    room for more then 40 characters here...”;
```

Failtext

```
rule max_trip_duty_days =
    trip.%days% <= %max_trip_days%;
    failtext(int lhs, int rhs) =
        concat(“Not allowed to have ”,
               format_int(trip.%days%, “%i ”),
               “days in one trip.”);
end
```

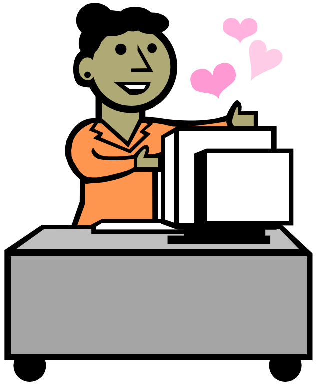
Note: Failtext can handle %l% >= %a% but
Rave Python API assumes %a% <= %l%

Exercise 2

~60 mins

Exercise 2 summary

- `count()` may only count integer column
  use `row_number`
- ascii sort order: `0..9 A..Z _ a..z`



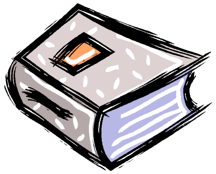
Chapter 3

Modules

Inheritance

Modules

- **Rave code structured logically**
- **Information hiding**
  - not all variables are visible everywhere
- **Abstraction**
  - other modules deal with sub-problem
- **Rule code shared between rule sets**



See
Rave Reference > Modules
for more information

Modules

Header

`module duty` A new module starts with the reserved word `module` followed by the file name: `module <module_name>`

`import leg` all other modules that are used in the current module have to be imported: `import <module_name>`

Modules

A basic module

```
file: $CARMUSR/crc/modules/duty


module duty
import leg;
import levels;
export %start% = first(leg(duty), leg.%start%);
export %end% = last(leg(duty), leg.%end%);
%length% = %end% - %start%;
%length_limit% = 10:00;
export %is_long% =
    %length% >= %length_limit%;
```

Variables

`%length% = …`

local variable can only be used in the current module

`export %start% = …`

can be used locally and in other modules that import this module. Accessed from other modules by `<module_name>.<variable_name>`, e.g. `leg.%start%`

`global export level duty = …`

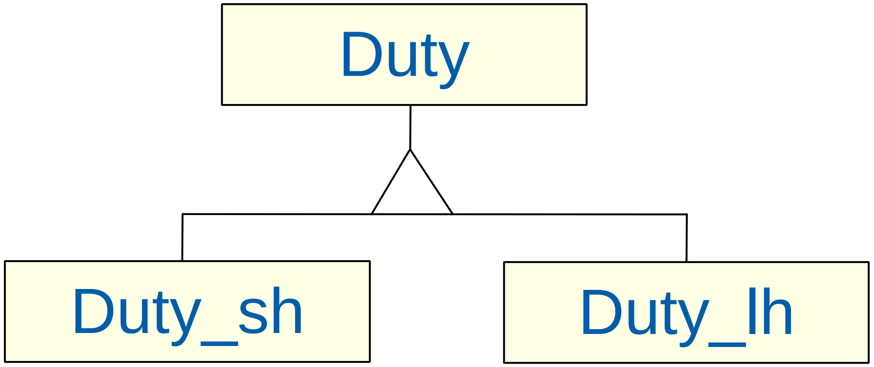
still needs import, but is used directly: `leg(duty)`. Compare with `levels.leg(levels.duty)`

> Note: (global) export mechanism only applies to Rave and not to reports, map variables or scripts.

Modules

Multiple implementations

A module may have several implementations:



Modules

Inheritance

- New module versions are defined based on existing ones
- The parent module must have a special syntax: **`root module <module_name>`**
- All definitions are inherited, but may be redefined
- A definition may be partially redefined
- A root definition may be incomplete.

Modules

Inheritance - redefinition

```
root module duty
%length_limit% = 10:00;
%is_long% = %block_time% > %length_limit%;
```

Child module

```
module duty_sh inherits duty
redefine %length_limit% = parameter 9:00;
/* This will affect %is_long% */
```

Modules

Inheritance – incomplete definition

```
root module duty
export %length_limit% = Reltime;
```

- The root module may have incomplete definitions, including only the data type
- All child modules must make a complete definition of `%length_limit%` with data type Reltime
- All redefinitions must always keep the original export status.

Modules

Inheritance

```
root module a_module
%a% = parameter 5 remark ”a comment”;
%b% = parameter 5 maxvalue 10;
table my_table =
    ac_type -> export %ac_family%
    ...
```

```
module child_module inherits a_module
redefine %a% = parameter _ remark ”New text”;
redefine %b% = parameter 20;
redefine export %ac_family% = ”SH”;
```

Will produce a compilation error

Note: %ac_family% is still a string.

Modules
Inheritance

Partial redefinition of a rule, examples:

1) Only change condition, keep everything else as is:

```
redefine rule inherited_rule =
    %x% < %y%;
end
```

2) Change valid and remark. Keep everything else as is:

```
redefine rule inherited_rule =
    valid leg.%is_long_haul%;
    _;
    remark ”LH version of rule”;
end
```

Modules
Rule set

A rule set:

- is defined by a top source file
- has the same name as the top file
- `use` statements define which modules should be included into the rule set.

Modules
`_topmodule`

- The top file may do `require`
- Code included in the top file by `require` statements will belong to the ‘virtual’ module: `_topmodule`

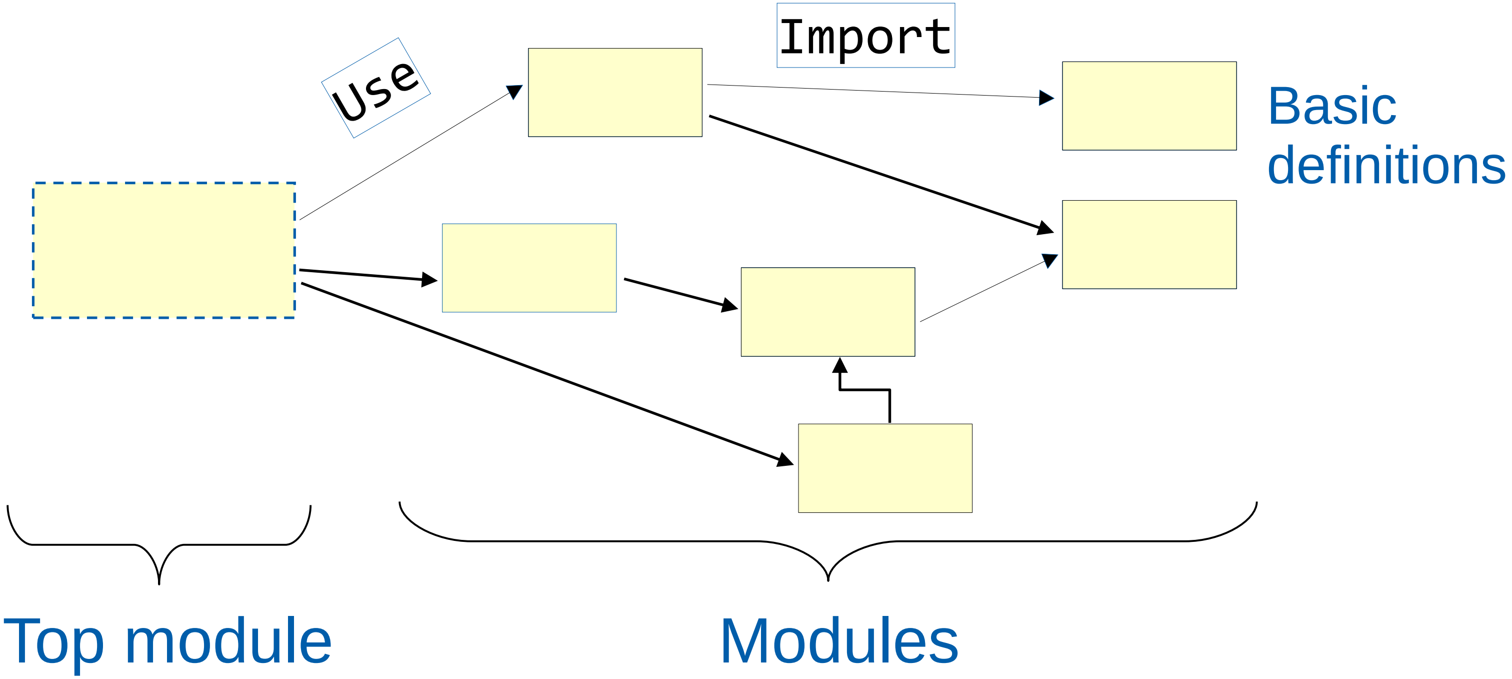
- An old rule set, with only `require`, will be one module called `_topmodule`
  - (Show Rule Values...)

- `_topmodule` may import other modules...
- ...*and* other modules may import the `_topmodule` (special case, not circular dependency)

Modules

Dependencies

- Circular dependencies are not allowed.
- The structure may be viewed as an acyclic graph.



Modules

Rule set – import or use in the top file

- `use <module>` is used to include a module into the rule set when it is not imported by any other module
- `use <child_module>` is used when we need to specify which child module that should be used (in case of multiple implementations)
- If an inherited module is imported, we still need to do `use` on the correct module version
- `import <module>` is used in the top file if there are definitions in the top file that refer to definitions in `<module>`
  Note: this is rare and should be avoided.

Modules

Rule set – example

Top file:

```
import duty;

use leg
use duty_lh
#if product(Studio)
    use report_check_legality;
end

%my_top_var% = duty.%variable%;
```

Usually not used

Let’s look at some examples

Modules

Example

- In `modules/my_modified_file:`

```
module my_modified_file inherits file    /*inherit*/

redefine export %aaa% = %ddd% / 2;      /*redefine*/
redefine %ddd% =
    parameter _ remark “Remark to ddd”;
                         /*only add/change remark*/
```

- In `source/top_file:`

```
use my_modified_file;    /*specify implementation*/
```

- In other files:

```
import file;      /*only specify root module name*/
%var% = file.%aaa%;
```

Modules

Finally about modules

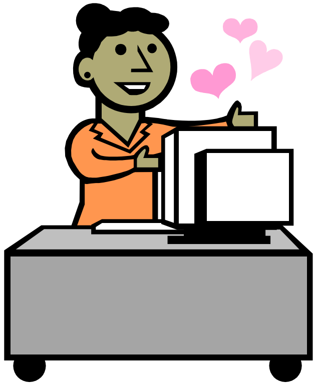
- Compilation is fast, only changed modules are recompiled
- All variables (even local) may be used in reports and scripts (always with the module reference)
- Global export is supported
  - you do not have to specify the module name in importing modules – only a Rave syntax feature
- There are modules for all optimization parameters in CARMSYS
- require is needed to define map variables for Studio.

Exercise 3

~45 mins

Exercise 3 summary

- `use` and `import` (discuss in class)



Chapter 4

Contexts

Iterators

Contexts

A certain context represents a certain group of objects:

- all objects in the window
- all trips in the sub-plan
- all rotations
- ...

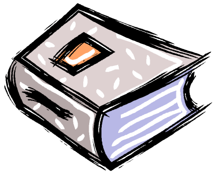
Use of **contexts** should normally be **avoided** in **Rave code** used by opt. Large risk for very **bad performance**.

You use them in Python code (and constraints).

See
Rave Reference > Definitions > Contexts
for more information

Contexts

`ac_rotations`: all rotations
`sp_crew`: all rosters
`sp_crrs`: all free trips
`sp_crew_chains`: all chains – i.e. rosters, free trips and duties



See
Studio Help > Keywords etc. > Contexts
for more information on currently defined contexts

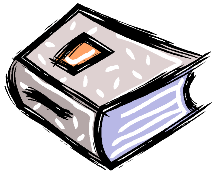
Contexts

- `default_context`: **what is in focus**
  - active chain (the one pointed at)
  - what is in a window
  - the whole plan.

- `current_context`: **the last explicitly named context**
  - very rarely needed.

Iterators

- Iterators divide a set of objects into smaller groups and sub-groups
- Sometimes you need to calculate values on the whole plan. For example, percentage of all flights leaving a certain airport. The iterator is then applied on a plan context
- Normally iterators are only used in reports and Python code. Rethink if you are on your way to use them in Rave code
- Normally you define the iterators you need, but there are also some built in ones.

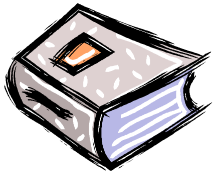


See
Studio Help > Keywords etc. > Iterators
for more information on currently defined iterators

Contexts
Example

Roster set iterator:

```
iterator roster_set =
    partition (roster);
end
```



See
Rave Reference > Definitions > Iterators
for more information on how to define iterators

Contexts
Example

Count all rosters in the plan:

```
%nr_assigned_crew% =
    context(sp_crew, count( roster_set ));
```

Contexts
Example

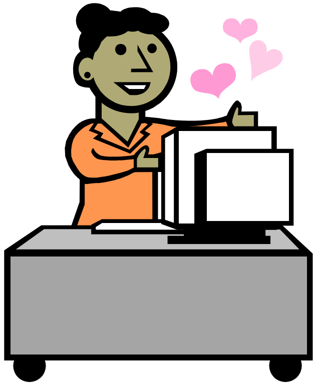
Total number of long trips in the sub-plan:

```
%nr_sp_long_trips% =
    context(sp_crrs,
        count(trip_set)
        where(%is_long_trip%));
```

Exercise 4

~60 mins

Exercise 4 summary



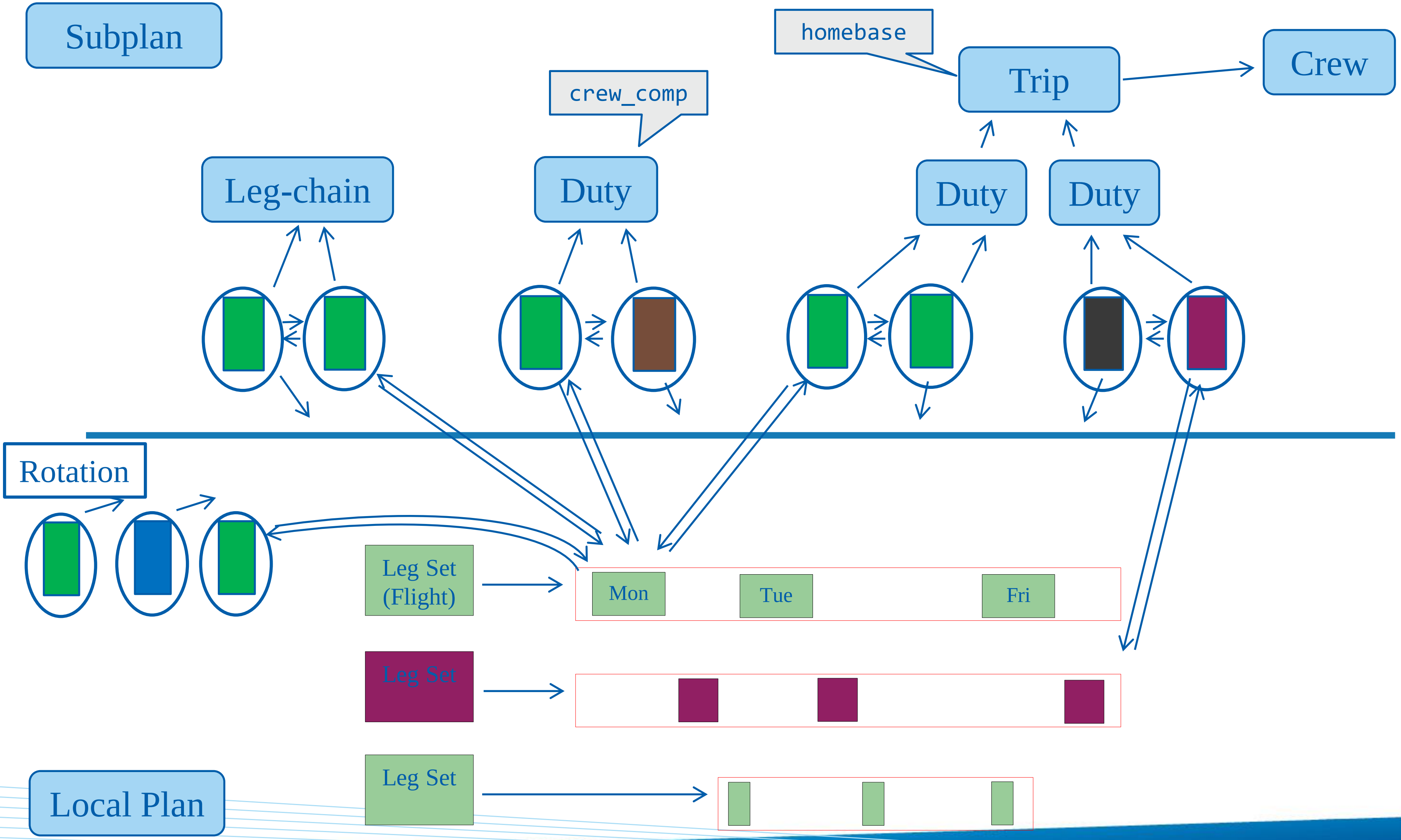
Data Model

Studio

Studio vs Rave

Introduction to transforms

Data model – Studio



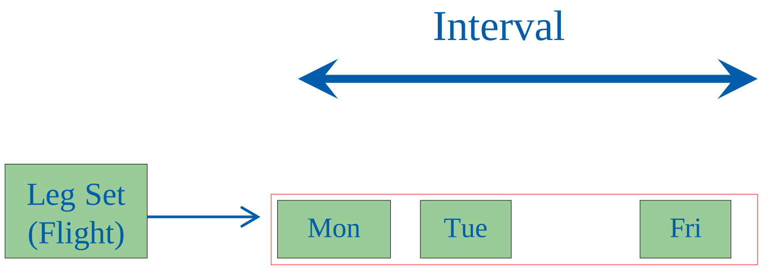
Introduction to transforms

Data model – Studio

Local Plan

Leg sets (Timetable legs)

- A set of legs that are equal in all aspects except for the dates (interval + frequency) when they are operating
- Contains all timetable data



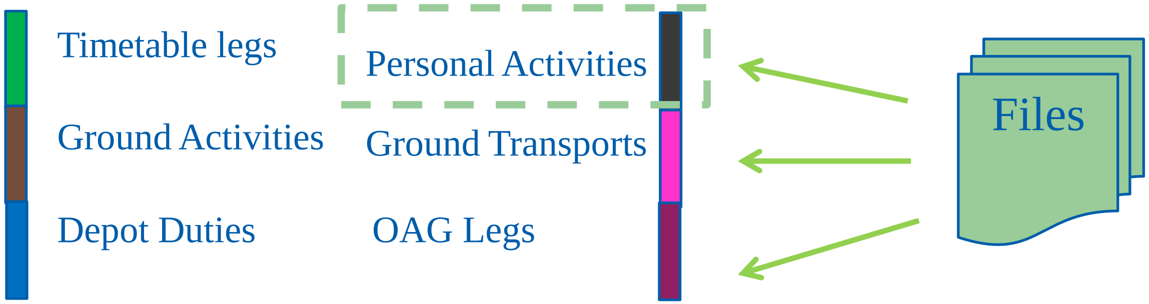
Interval

Leg Set (Flight)

Mon

Tue

Fri



Timetable legs

Ground Activities

Depot Duties

Personal Activities

Ground Transports

OAG Legs

Files

Rotations

- Chain associated with an aircraft or a vehicle
- One row per chain in the vehicle part of a local plan

Introduction to transforms

Data model – Studio

Subplan

Legs

One reference for each leg in the timetable that we want to plan

May be connected into leg chains

(Duties

One chain of sub-plan legs that are not part of any complete trip

Not a separate data object, mainly for displaying duties returned by APC)

Trips

One chain of sub-plan legs that make up a complete trip – homebase is set

Rosters

One chain for each crew, containing a sequence of trips

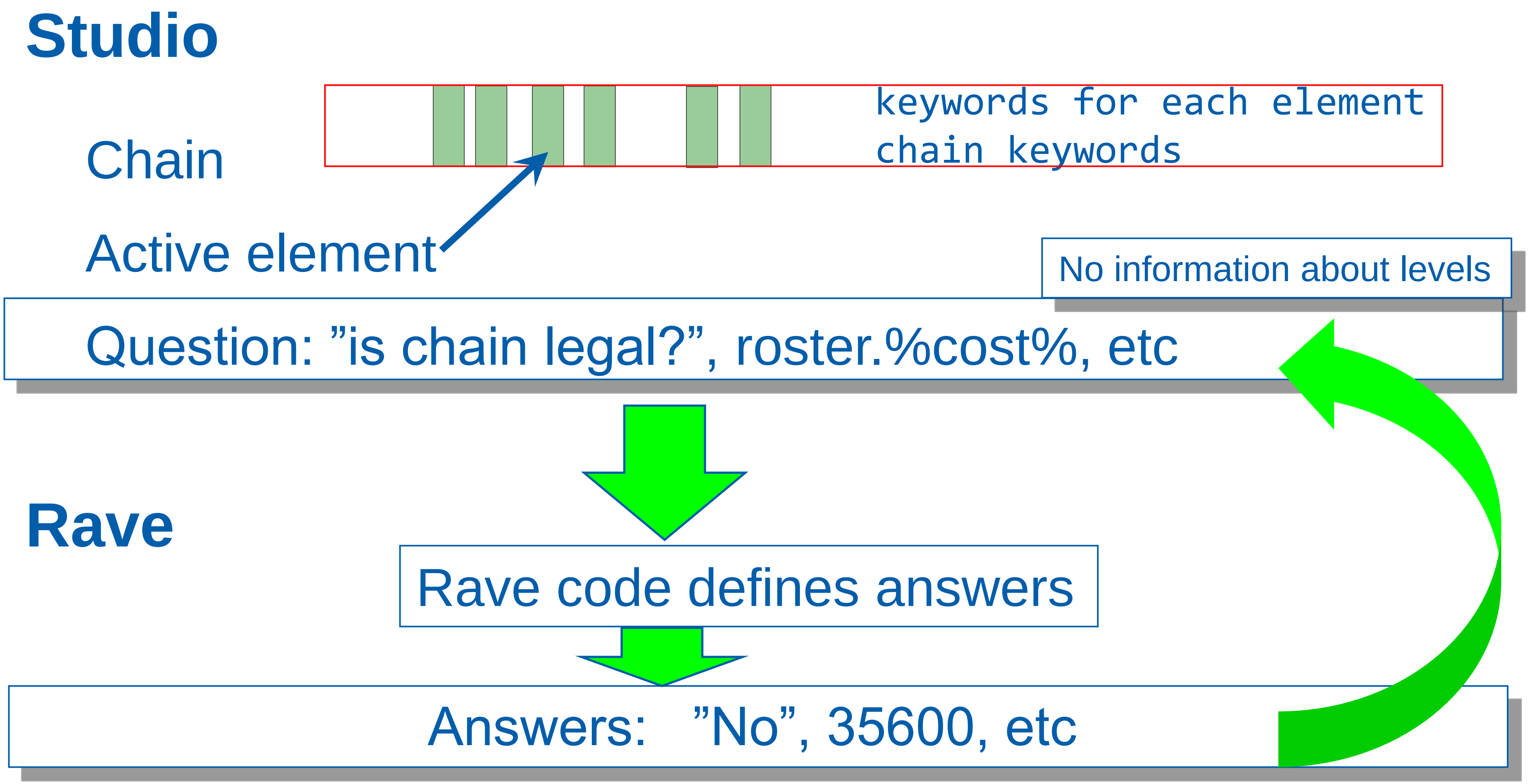
Introduction to transforms

Data model – Studio vs. Rave

- **Studio**
  - Trip, duty, leg etc. are determined by the contents in the different windows (trip, duty, leg-window etc)
  - They are predefined data model
- **Rave**
  - Trip, duty, leg etc. are defined by the level definitions in the Rave code
  - …and may be called something else or not defined at all.

Introduction to transforms

Input to Rave



Studio

Chain

keywords for each element
chain keywords

Active element

No information about levels

Question: ”is chain legal?”, roster.%cost%, etc

Rave

Rave code defines answers

Answers: ”No”, 35600, etc

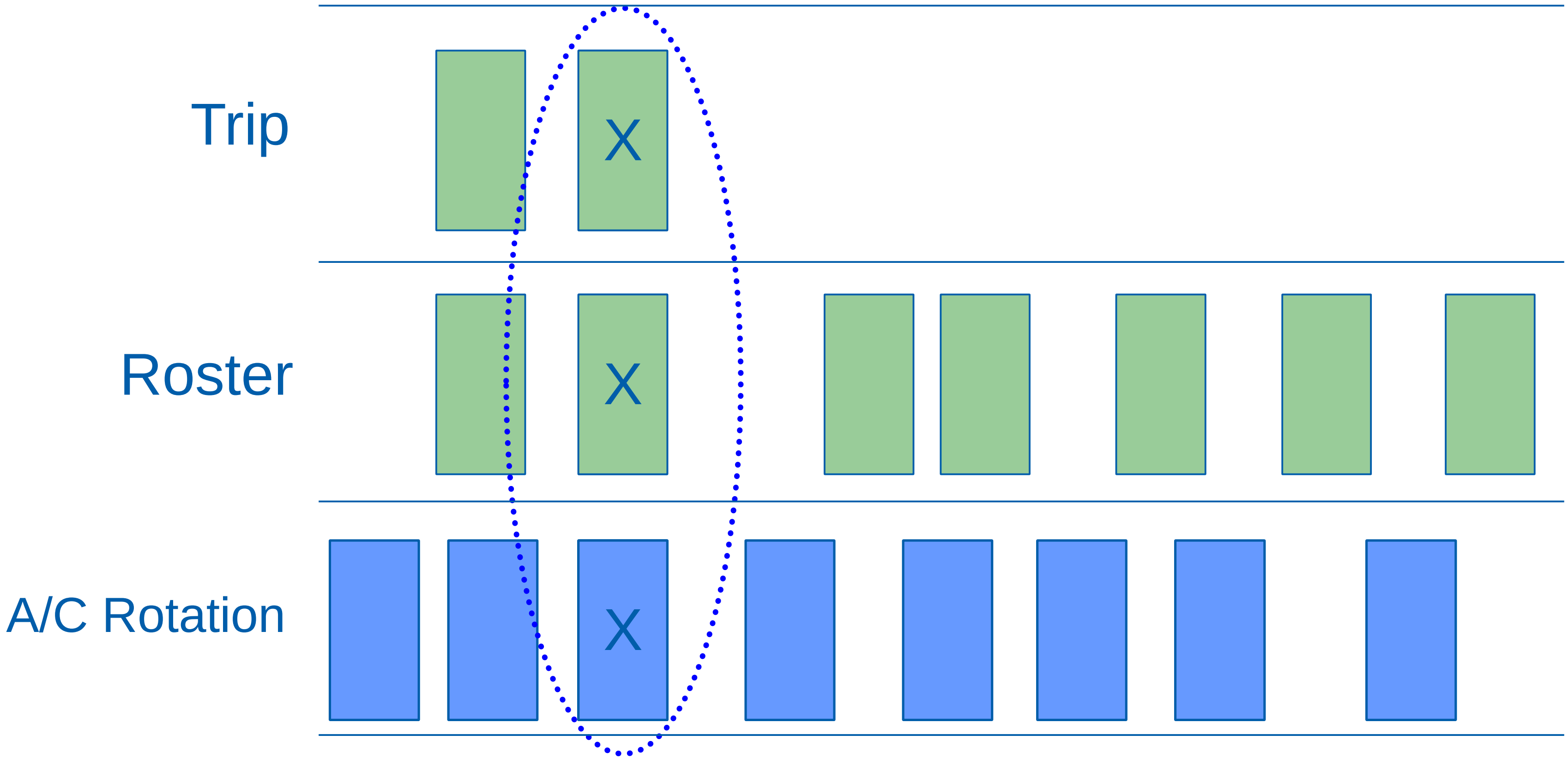
Transforms

Pairing

Rostering

Transforms

Transforms are used to find other representations of the same leg:

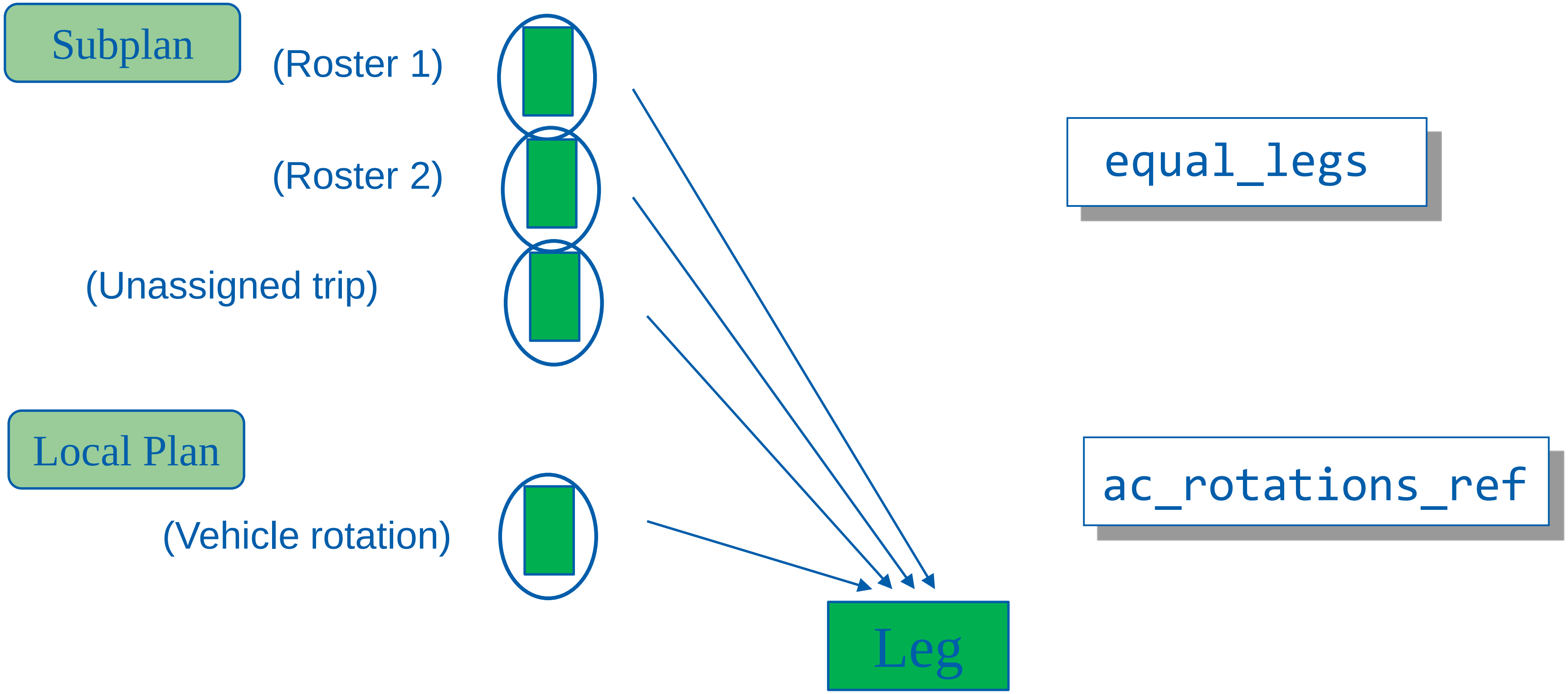


Transforms

The most used transforms

`equal_legs` All segments in the **sub-plan** pointing to the same leg as the current segment.

`ac_rotations_ref` All segments in the **rotations** pointing to the same leg as the current segment.



Transforms

The most important transforms:

- `equal_legs`
- `equal_trips`
- `ac_rotations_ref`

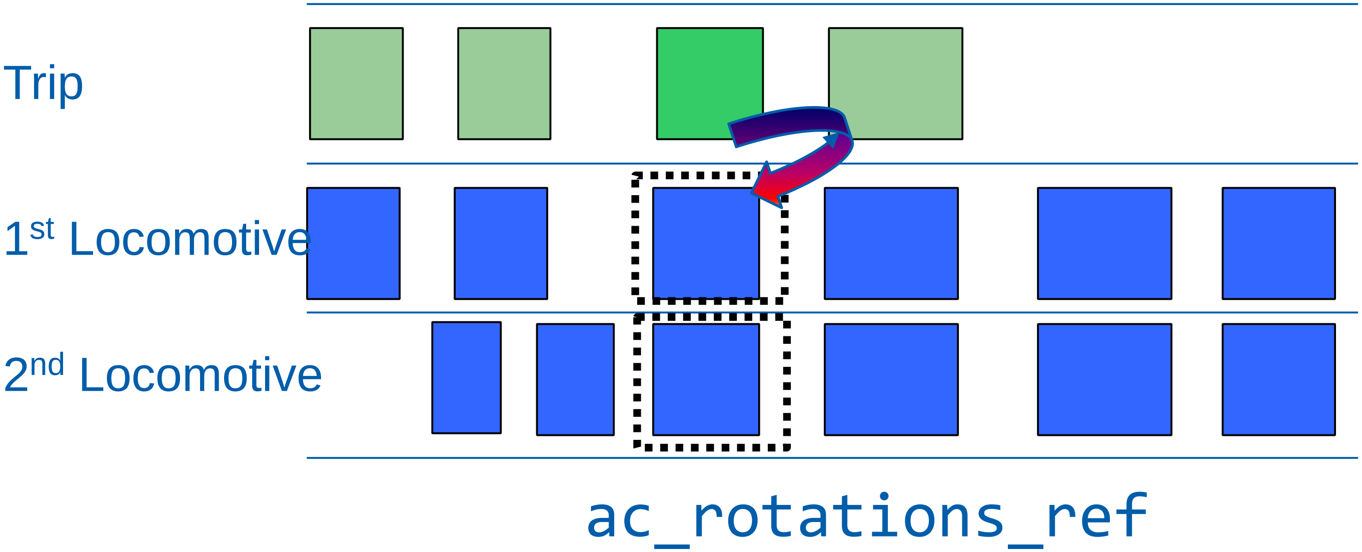
See
Studio Help > Keywords etc. > Transforms
for more information on currently defined transforms

See
Rave Reference > Expressions > Transforms

Transforms

We may use transforms to find references to the same leg:

```
%locomotives% = count(ac_rotations_ref);
```

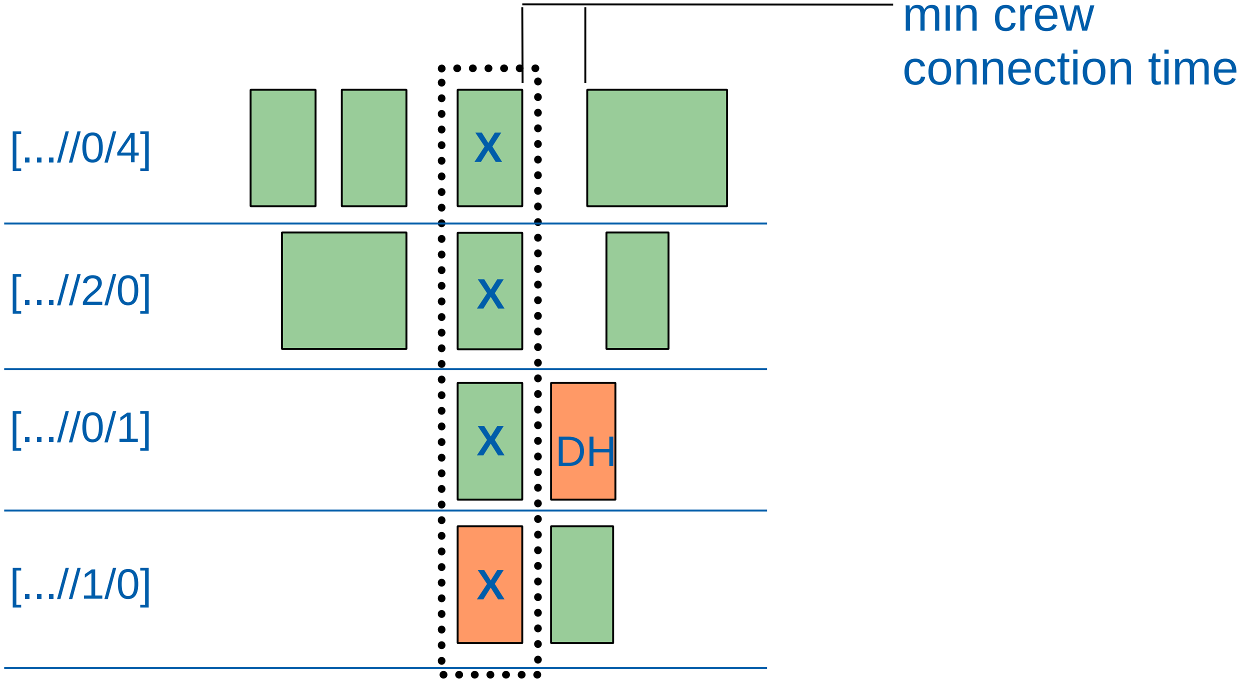


Transforms

Pairing – example

Calculate the minimum connection time for a flight

Do not count deadhead connections



equal_legs

Transforms

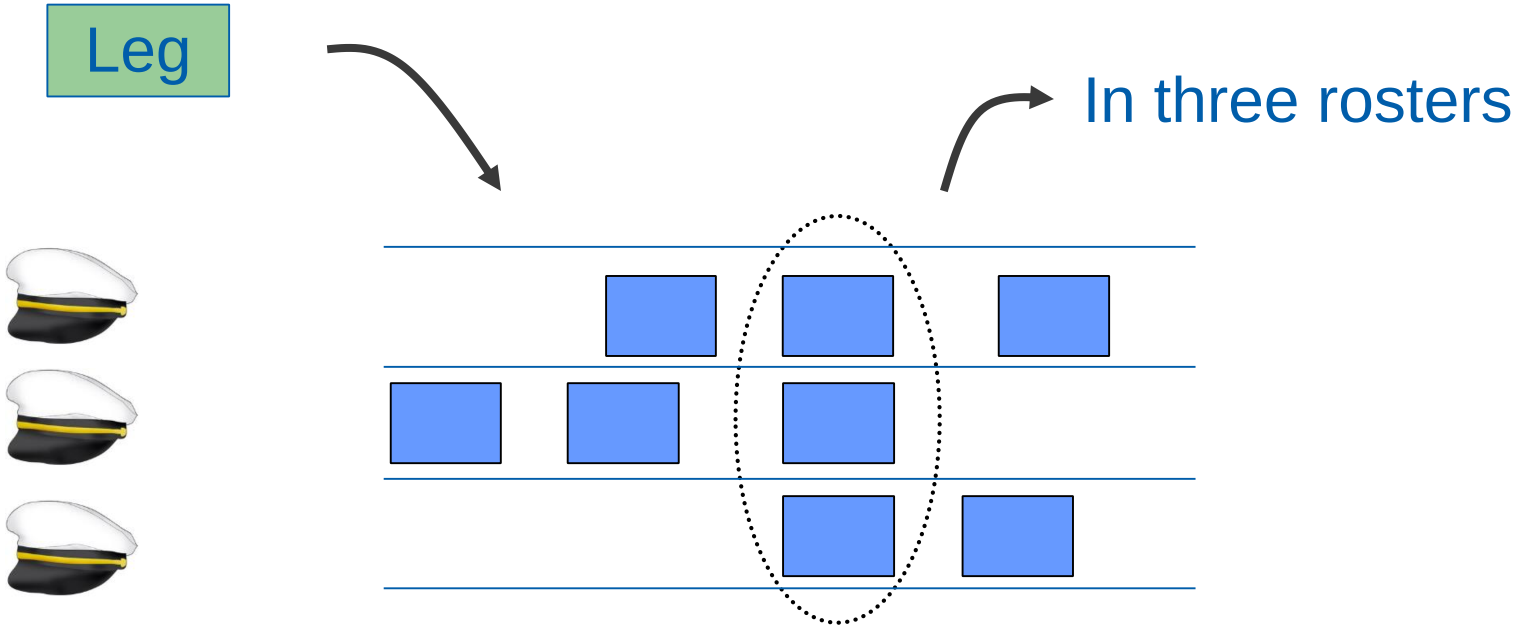
Pairing – example

```
%min_cnx_time%=
    min(equal_legs, leg.%connection_time%)
    where(%count_leg%);

%count_leg% =
    (not deadhead)
    and (not is_last(leg(duty)))
    and (next(leg(duty), not deadhead));
```

Transforms

Rostering – example

Number of assigned crew to the same flight:



Leg

In three rosters

Transforms

Rostering – example

```
%nr_assigned_crew% =
    count(equal_legs)
    where(not void(crr_crew_id));
```

`crr_crew_id` is void for any leg reference that is not part of a roster yet, i.e. an unassigned trip

A leg will have a reference from each crew member to which the leg is assigned, this is used by the transform

Rostering – example

- On many flights there is a need for cabin crew to have a certain language knowledge
- Write a rule to ensure that enough cabin crew have the language knowledge
- Assume that there is a leg variable saying whether the assigned crew member has language knowledge.

Transforms

Rostering – example

```
rule crew_has_language_knowledge =
    %nr_crew_with_language%
    >= %min_req_crew_with_language%;
end


%min_req_crew_with_language% =
    parameter 5
    remark ”Min crew with language: ”;
```

Transforms

Rostering – example

```
%nr_crew_with_language% =
    count(equal_legs)
    where(%has_language_knowledge%);
```

Disadvantages with this kind of rule:

- it will not work in optimizers
- it is illegal from the beginning
- how to change it for Studio?
- ... use constraints instead in optimizers!

Constraints

Constraints

Qualifications

Syntax

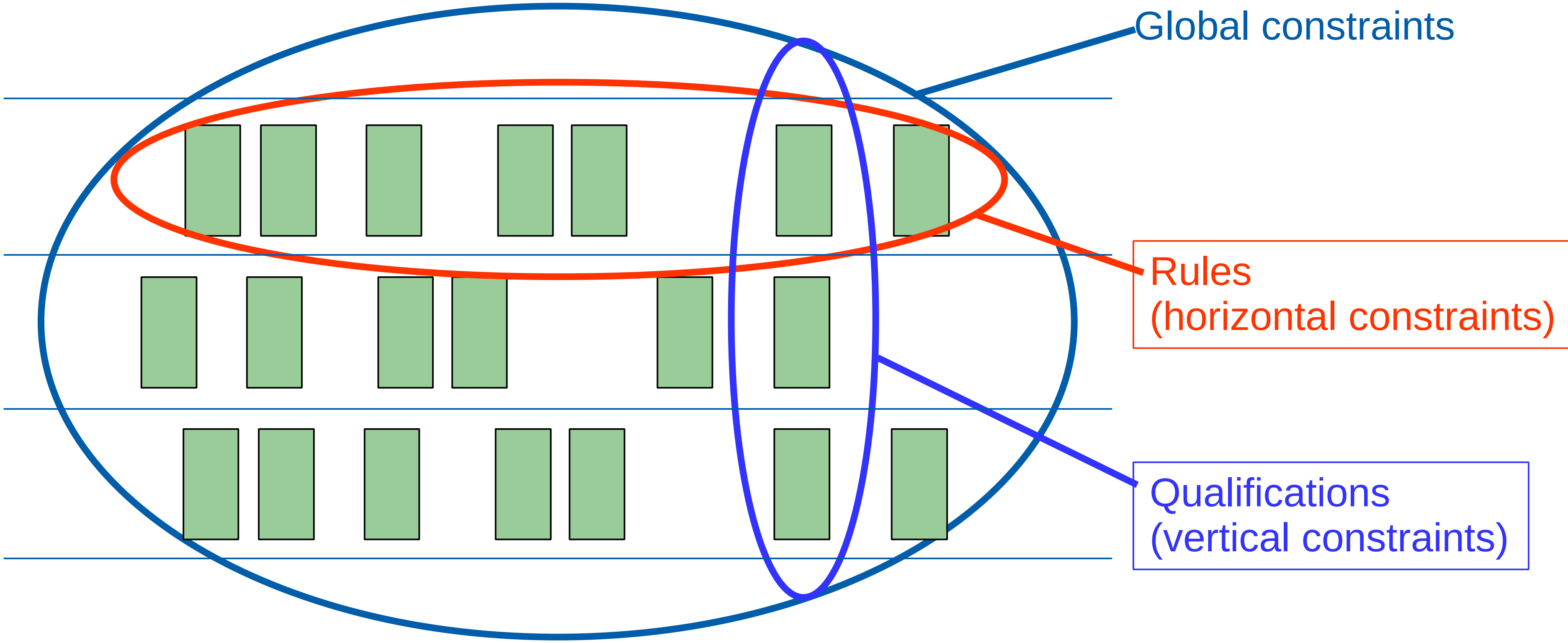
Pairing

Rostering

foreach

contexts and transforms

Constraints



Global constraints

Rules
(horizontal constraints)

Qualifications
(vertical constraints)

Optimization – restrictions

- Rules
  - access single trips/rosters

- Qualifications (vertical constraints)
  - access all identical legs/trips in the solution

- Global constraints
  - access the whole solution

- Optimizers can only see one chain when generating
- Qualifications and global constraints need special treatment.

Constraints

- A condition based on properties from several chains (trips or rosters)

- Terms:
  - global constraint is often used when the data in a constraint is specified by a context
  - vertical constraint is used when the data is specified by a transform.

Constraints – examples

“Limit the amount of duty days per base”
(global constraint)

“Limit the number of free days in a Crew Rostering plan.”
(global constraint)

“Limit the number of short rest trips to 100,
and set a cost of 1000 for exceeding ones:”
(global constraint)

”Not all pilots on a trip can be inexperienced”
(trip qualification)

“There must be enough language knowledge on a leg”
(leg qualification)

Base constraints

"Limit the amount of duty days per base"

Base constraints:

- are a special kind of global constraints
- in older versions efficiently implemented in Crew Pairing optimizer
- … but has now been removed
- … and only Rave global constrains are used.

Qualifications

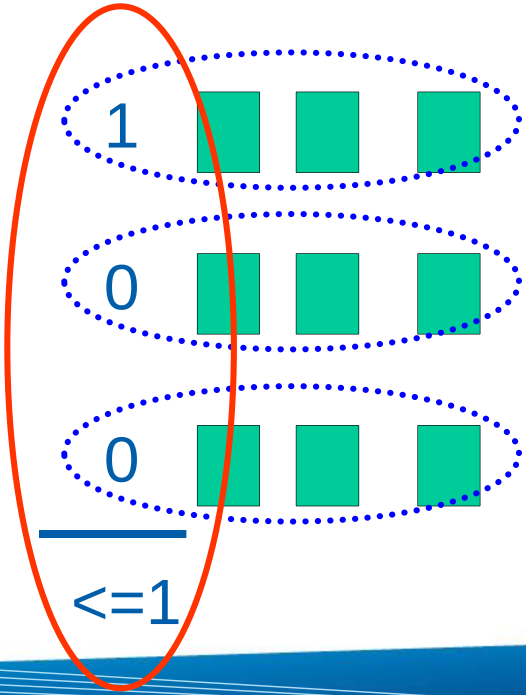
Vertical Constraints

- A qualification is a condition that identical legs/trips in the solution must follow
- Crew complement is a special kind of qualification that has been built into the system

Example:

Only one pilot on a trip may be inexperienced:

```
trip.%is_inexperienced% =
    if <inexperienced>
    then 1
    else 0;
```



Constraints

```
constraint_def ::=
gexport constraint [ponoff] constr_name =
    [valid ;]
    (foreach ;)*
    constraint_condition ;
    [nonadditive(formalargs) = expr ;]
    [cost [(expr)] = expr ;]
    [failtext ;]
    [remark "text" ;]
    [group_def]
end
```

Break

Constraints

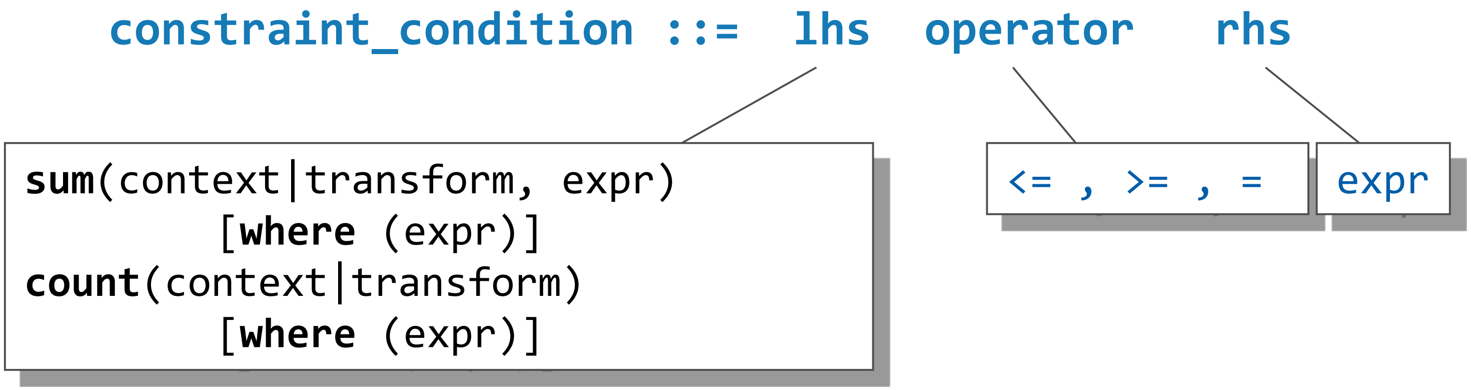
Syntax

```
constraint_def ::=
gexport constraint [ponoff] constr_name =
    [valid ;]
    (foreach ;)*
    constraint_condition ;
    [nonadditive(formalargs) = expr ;]
    [cost [(expr)] = expr ;]
    [failtext ;]
    [remark "text" ;]
    [group_def]
end
```

Constraint

Condition

The constraint condition is where the actual limitation is expressed, like “only one inexperienced crew”



```
constraint_condition ::=  lhs  operator   rhs
```

```
sum(context|transform, expr)
          [where (expr)]
count(context|transform)
          [where (expr)]
```

```
<= , >= , =
```

```
expr
```

The `sum()` or `count()` are followed by a relational operator, and a right-hand side expression. The traversers can optionally be limited with a `where` clause.

The data to be traversed by `sum()` or `count()`must be an application-defined context or transform.

Example

```
 count(equal_legs) where (crew.%is_inexperienced%) <= 1;
```

or

```
 sum(sp_crrs, %trip_is_long%) <= %max_long_trips%;
```

Constraints

Syntax

```
constraint_def ::=
gexport constraint [ponoff] constr_name =
    [valid ;]
    (foreach ;)*
    constraint_condition ;
    [nonadditive(formalargs) = expr ;]
    [cost [(expr)] = expr ;]
    [failtext ;]
    [remark "text" ;]
    [group_def]
end
```

Constraints

`cost`

- Specifies the cost for a global constraint when the left-hand side is out of range
- For Integer constraints, the deviation unit = 1
- For Reltime constraints the deviation unit = 0:01 (one minute)

```
cost = expr;
```

creates a linear cost
The total `cost = expr * deviation`
The deviation is the number of units outside the allowed range

Constraints
`cost`

`Rostering:`

- `cost (when max max_dev) = expr;`
  creates a *constant* cost while between limit and maximum deviation
  If deviation > `max_dev`, the constraint is illegal.
  This should be avoided

- `no cost`
  If there is no cost defined, it is illegal to be out of range. When used, make sure that the constraint cannot be broken by deassigning activities.
  This should be avoided.

Examples

Examples (Pairing)

Max 1000 hours of production (active block time) on trips with home base ARN. Give a penalty of 20 for each exceeding minute:

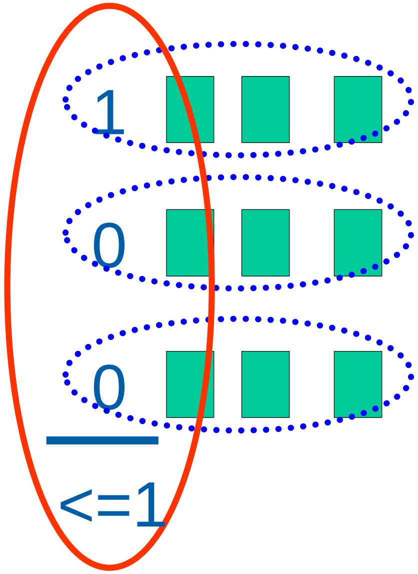
```
constraint ARN_base_time_limit =
    sum(sp_crrs, trip.%block_time%)
        where (homebase=“ARN”)
    <= 1000:00;
   cost = 20;
end
```

Qualifications

Example (Rostering)

Max one inexperienced crew member on the same trip:

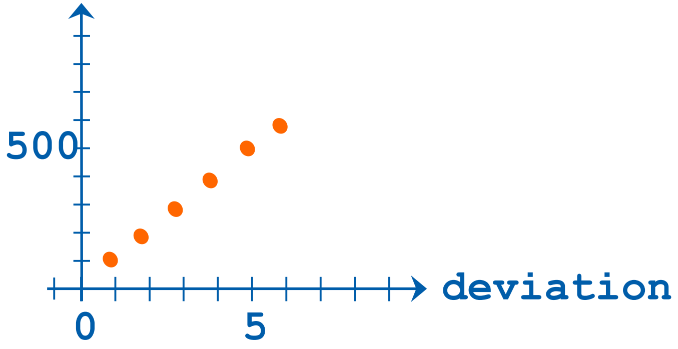
```
%max_inexp% = parameter 1;
constraint (on) trip_inexp =
    count(equal_trips)
        where (%crew_is_inexp%)
    <= %max_inexp%;
    cost = %thousand%;
end
```



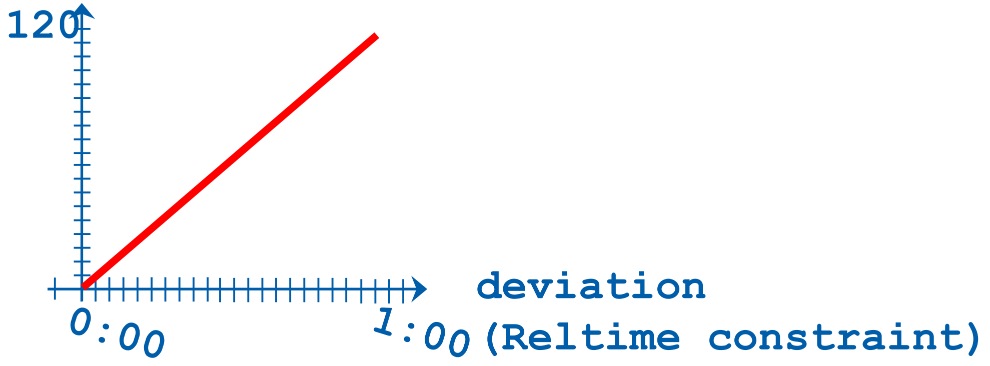
Constraints

Cost examples

`cost = 100;`



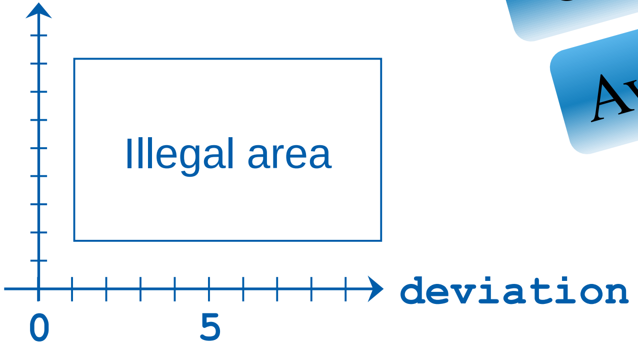
`cost = 2;`



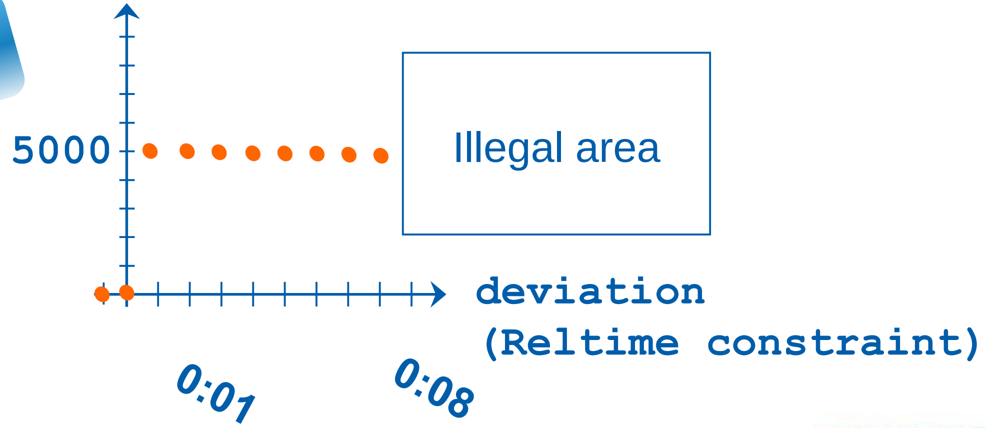
Only Rostering

Avoid!

`[no cost defined]`



`cost(when max 0:08)= 5000;`



Constraints
foreach

```
foreach lvar in foreach_set [alias] [where | while]
```

- creates a vector of constraints, one constraint for each element in `foreach_set`
- `lvar` variable can be used anywhere within the rest of the constraint expression
- `foreach_set` can be either a normal Rave set, an enum type or a range, e.g.
  `range_set = set(start, end [, step])`
- `alias` can be used to define output format of loop variable.

Examples

```
foreach day in set(1, %days_in_planning_period%)
    alias %day2datestring%(day);
foreach base in active_bases
    where %has_limit%(base);
```

Constraints
foreach example

A constraint that limits the number of 5-day trips that start on each day to 8.

`%start_date%` already defined as start date of the planning period,

and `%end_date%` as end date.

```
constraint (on) limit_5_day_trips_on_each_day=
    foreach day in set(%start_date%, %end_date%, 24:00);
    sum(default_context, %count_5_day_trip%(day)) <= 8;
    cost = 5000;
end

%count_5_day_trip%(Abstime day)=
    if %caldays% = 5
        and day <= trip.%start%
        and trip.%start% < (day + 24:00)
    then 1 else 0;
```

Constraints
`valid`

`valid expr;`
defines for which chains to test the constraint on

- Vertical constraints (qualifications):
  - `expr` may be chain dependent,
    ie which chains to test
- Global constraints:
  - `expr` must be constant,
    ie under what plan conditions to test

Example

Only apply vertical constraint on flight duty legs:

```
valid leg.%is_flight_duty%;
```

Constraints
`failtext`

Two ways to define text to print in Studio if/when the constraint is broken:

```
failtext expr;

failtext %my_own_expression%(base);
```

```
failtext(type lhs, type rhs)= … ;

failtext(int lhs, int rhs) =
    concat("Long trips at ",
           base,
           format_int(lhs, "%i"),
           format_int(rhs, "(%i)"));
```

Results in ” Long trips at STO 22 (20)”

Use remark in APC, name in Matador

Constraints
`remark`

```
remark
```

Defines a remark.
(Works the same as for parameters and rules)

Constraints

Contexts and transforms

Crew Rostering

A global constraint uses either of the contexts `sp_crew_chains` or `default_context`. Both give the same result in Crew Rostering Optimizer. But in Studio `sp_crew_chains` corresponds to all rosters and unassigned trips in the sub-plan, while `default_context` corresponds to the content in the current working window.
Use default_context!

Crew Pairing

A global constraint uses either of the contexts `default_context` or `sp_crrs`. Both give the same result in APC, but in Studio does `sp_crrs` correspond to all trips in the sub-plan, while `default_context` corresponds to the content in the current working window.
Use default_context!

Transforms

`equal_trips` can be used in Crew Rostering optimization,
and `equal_legs` in both Crew Rostering and Crew Pairing optimization.

Constraints

Contexts and transforms

	Pairing Opt.	Rostering Opt.	Studio
Contexts			
`sp_crew_chains`	■	All rosters and unassigned trips in the optimizer	All rosters and unassigned trips in plan
`default_context`	All trips in the optimizer		All chains in window
`sp_crrs`		■	All trips in plan
Transforms			
`equal_legs`	Supported	Supported	Supported
`equal_trips`	■	Supported	Supported

Constraints

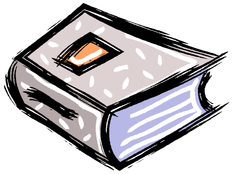
Additional elements

`group_def`

`nonadditive`

The `nonadditive` and `group_def` elements are features that provides more information about the constraint expression to the optimizers, helping them improve the performance.

In Pairing and Rostering they do not change the semantic meaning of the constraint in any way.



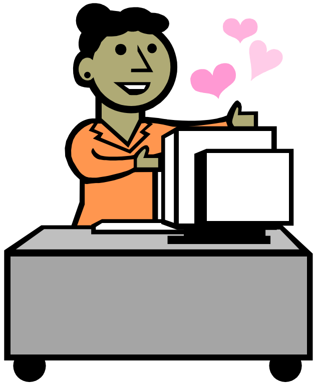
See
Crew Rostering Optimizer Reference Manual &
Crew Pairing Optimzer Reference Manual

Exercise 5

~45 mins

Exercise 5 summary

- Constraint limit to max 10%?



Recap

Recap Day 1:

- Rave I
- Tables
- Modules & Inheritance
- Contexts & Iterators
- Transforms & Constraints

Agenda

Day 2:

- Performance & Caching
- Costs & Rules
- Accumulators
- Summary and Evaluation

Chapter 6

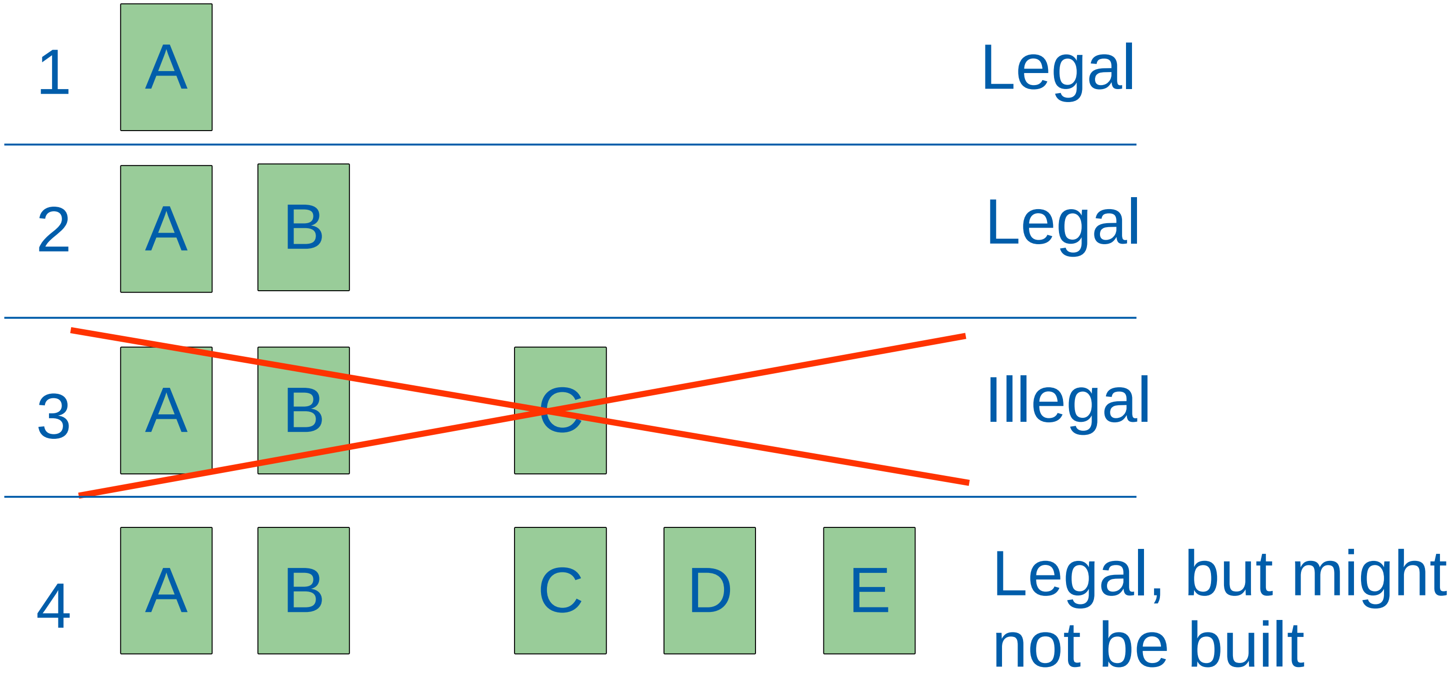
Illegal Subchains

Performance

Caching

Illegal sub-chains

Illegal sub-chains may prevent chains from being built:



Legal

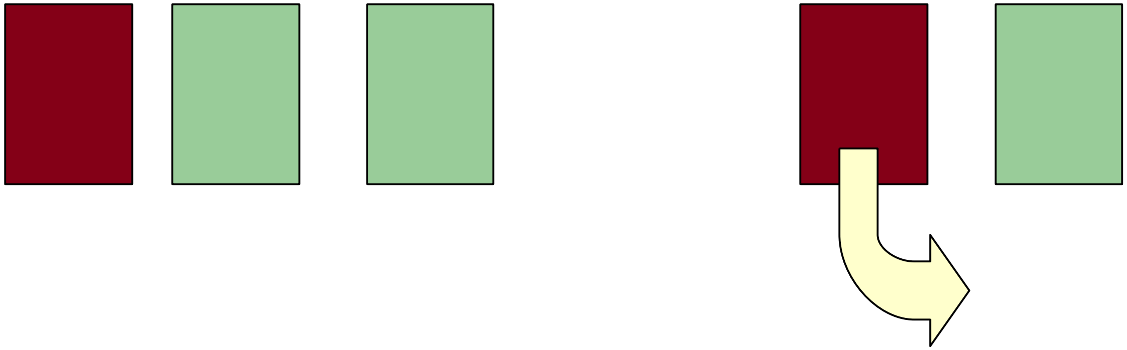
Legal

Illegal

Legal, but might not be built

Illegal sub-chains

Max two consecutive green duties:



sub-chains

Crew Pairing

A trip could become illegal when you remove legs from the beginning or end of the trip.

Example

A duty period may not have less than two hours of block time.

Illegal sub-chains

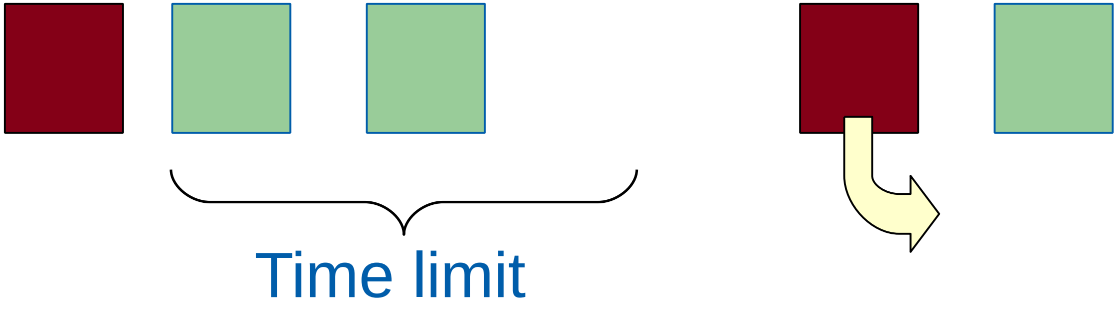
Hard to identify:

- APC warnings – status file
- Often not generated at all
- Run a complete solution again (and check status file)

Avoid illegal sub-chains

Loosen up the rule, restrict it more

- Add a time limit to the rule ‘max consecutive’
- Never have rules about ‘at least …’



Avoid illegal sub-chains

- Think about this when writing rules
- Replace with a penalty
  = allow, but to a high cost
- Use matador.map_roster_is_complete

This can be tricky,
learn more in the optimization courses.

Illegal sub-chains

Crew Pairing

Use the keywords `crr_is_closed_[first|last]`

Example

```
rule min_block_time =
    valid crr_is_closed_first
          and crr_is_closed_last;
    %block_time_trip%
    > %min_block_time_trip%;
    remark: “An example rule for block time”;
end
```

The keywords should be used with caution.

Learn more in the ‘Rave for Pairing Opt’ course.

Performance

Lazy evaluation

Tables

Levels

Range

Caching

Performance analysis

Performance

Lazy Evaluation

Logical expressions use lazy evaluation to improve performance. As soon as the outcome of an expression is fixed, the evaluation is stopped. E.g.

`%a% or %b%` If %a% is true, then we do not evaluate %b%

`%a% and %b%` If %a% is false, then we do not evaluate %b%

```
rule flight_deck_rule =
    valid* %fd% and %check_trip% and …
```

If this is not a flight deck trip, then the rest of the valid statement will not be evaluated (neither will the condition).

*) Note: Valid statements as such do not improve performance as the logic could be expressed in the condition

Performance

Tables

- Direct matching in an external table is handled very efficiently by hash keys

- Relational comparisons (and intervals) are matched by a linear search, and may therefore affect performance

- The statements above does not apply to internal tables. Internal tables are compiled and efficiently coded into c-switch statements.

Performance

Tables

- Look up several values from one table statement if possible
- When Rave calculates one value, the other values are cached (evaluated proactively) at the same time
- This is especially important for external tables using relational comparisons
- The table is only loaded once, less memory consumption

```
…
    -> %a%, %b%, %c%
…
```

Lookup of %a% will automatically cache %b% and %c%.

Levels

Leg, Duty, Trip, Wop, Roster

- only on important, basic objects
- not too complex definitions
- avoid while and where.

Avoid parallel level structures (day/week –duty/trip)

- use functions that return day/week given a time.

Performance

Dependency (attribute of)

The dependency determines what objects (levels) to associate the value with:

```
%block% = arrival – departure;


%check_out% =
    last(leg(duty), arrival + %debriefing%);
```

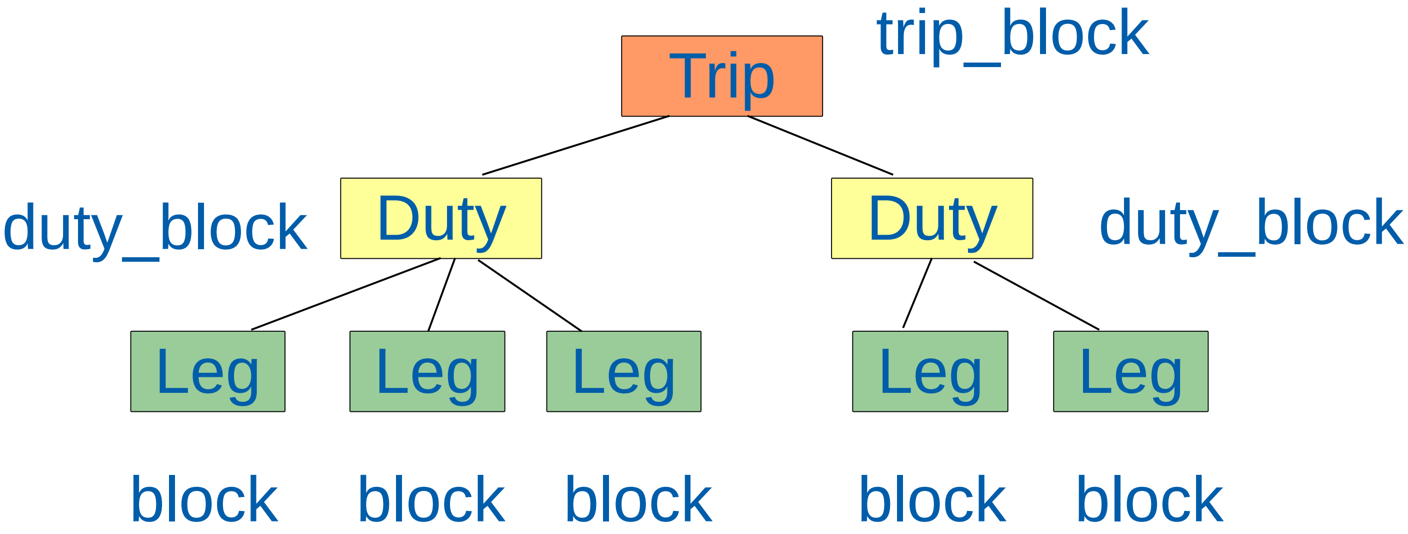
Dependency

```
%trip_days% =
    (round_up( %trip_end%, 24:00)
    - round_down(%trip_start%, 24:00))
    / 24:00;
```

Assuming `trip_start/end` are trip dependent, `trip_days` will also be trip dependent.

Performance

Dependency



Performance

Range

The range indicates how much of the chain the calculation uses:

```
    %block% = arrival - departure;
```

Dependency (attribute of): Leg

Range: Leg

This variable needs to be re-evaluated only when the leg changes.

Performance

Range

```
%leg_cxn_time% =
    next(leg(duty), departure) – arrival;
```

Dependency: Leg

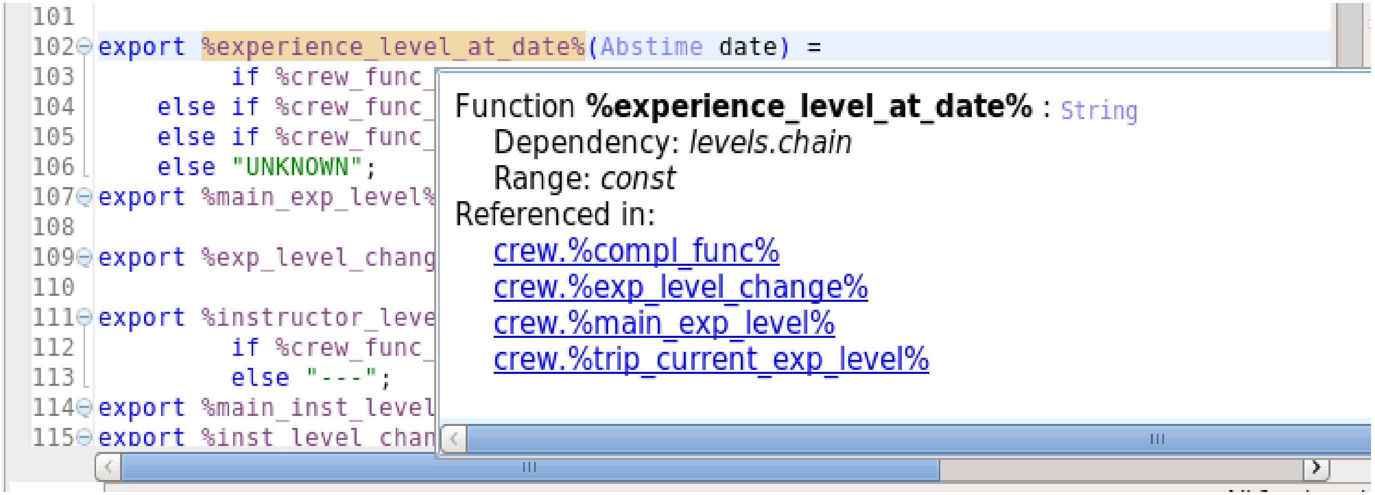
Range: Duty (formally Leg+Next leg)

Every time the duty changes, this variable needs to be re-evaluated.

Performance

Dependency, Range

Tooltip in DWS will display Dependency and Range information:



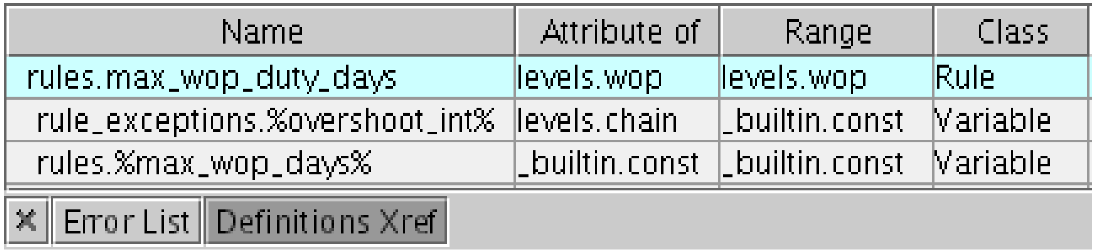
```
export %experience_level_at_date%(Abstime date) =
          if %crew_func_
     else if %crew_func_
     else if %crew_func_
     else "UNKNOWN";
export %main_exp_level%

export %exp_level_chang

export %instructor_leve
          if %crew_func_
          else "---";
export %main_inst_level
export %inst_level_chan
```

Function **%experience_level_at_date%** : String
Dependency: *levels.chain*
Range: *const*
Referenced in:
- crew.%compl_func%
- crew.%exp_level_change%
- crew.%main_exp_level%
- crew.%trip_current_exp_level%

Performance

Dependency, Range

Use Definitions Xref in Rave IDE to get detailed information about dependency and range in your rule set:



Name	Attribute of	Range	Class
rules.max_wop_duty_days	levels.wop	levels.wop	Rule
rule_exceptions.%overshoot_int%	levels.chain	_builtin.const	Variable
rules.%max_wop_days%	_builtin.const	_builtin.const	Variable

Performance

Caching

- The value of a rule or variable can be remembered (cached)

- A cached value is faster to look up than a new evaluation would be.

Performance

Caching

- Caching is fully automatic
- Only values for the current chain are cached
- Try to make sure that expensive calculations can be cached.

Caching – limitations

Limitations

- Iterators
- Some contexts/transforms
- Functions

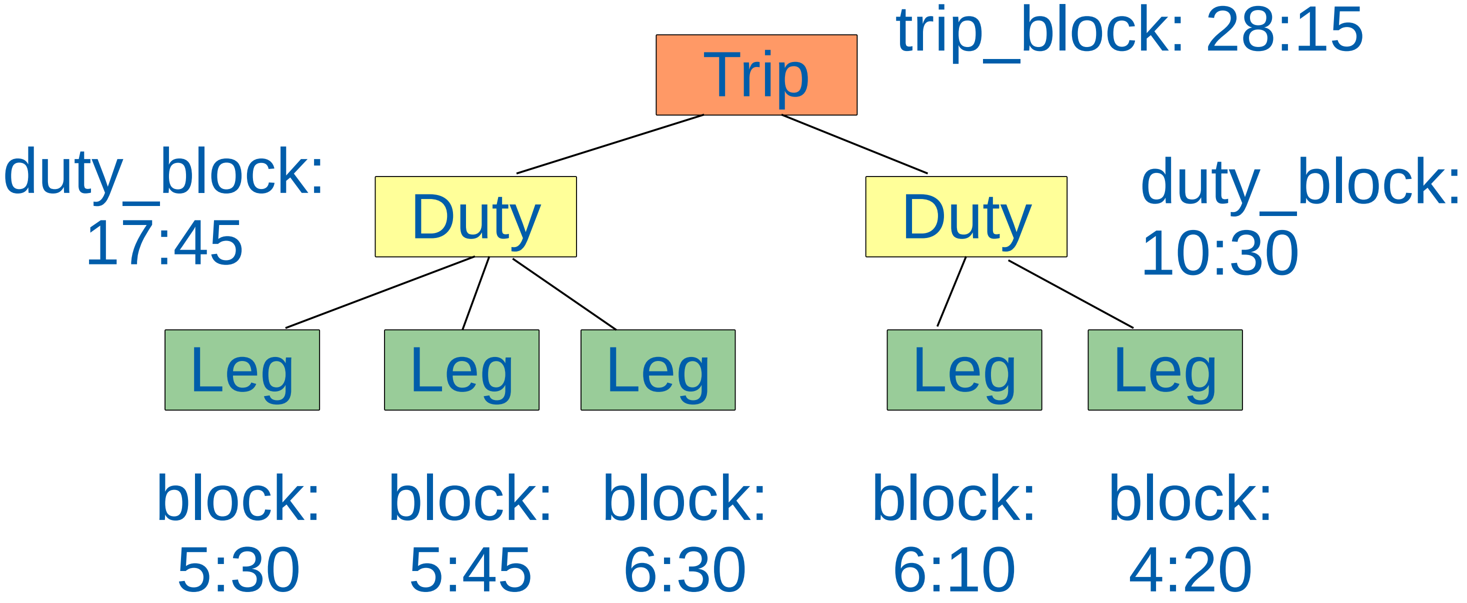
Trade-off between:

- speed
- overhead
- written code.

Performance

Caching

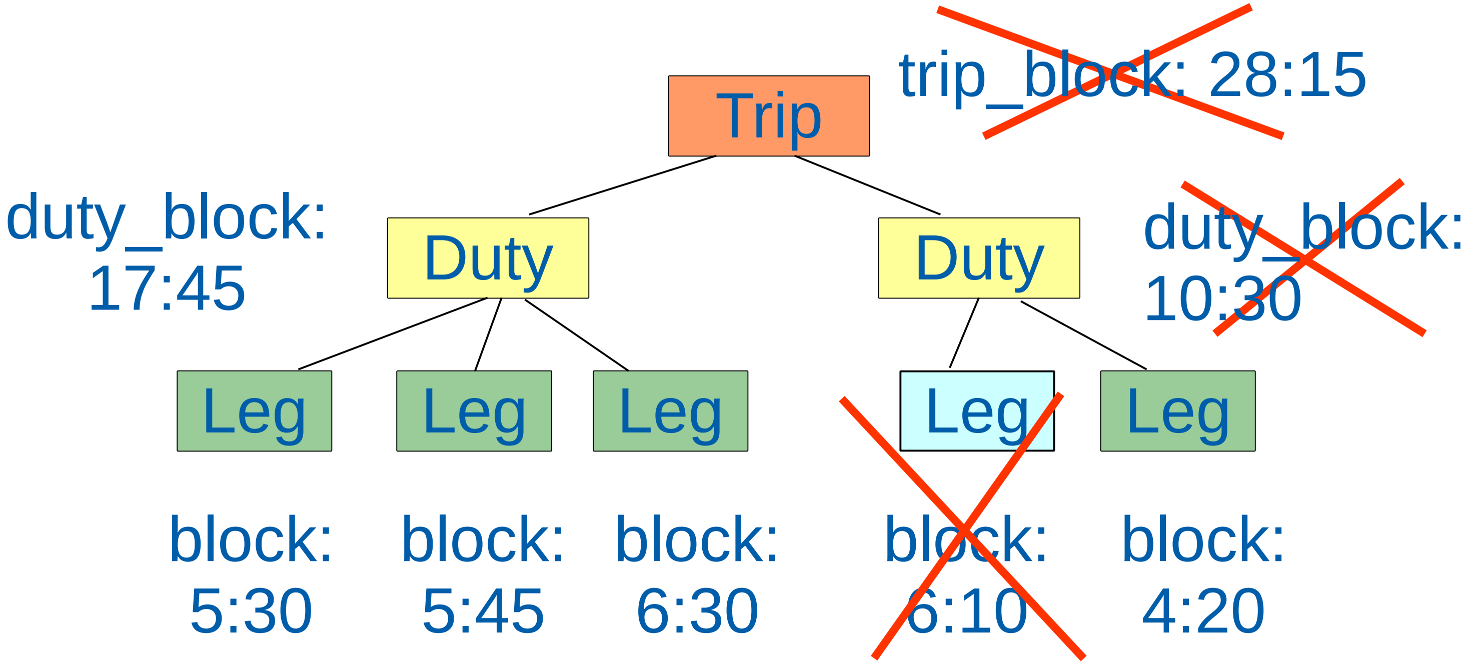
First call will evaluate (and cache) everything needed for the calculation:



Performance

Caching

When a leg changes, some values need to be re-evaluated, some will still be in the cache:



Trip

~~trip_block: 28:15~~

duty_block: 17:45

Duty

Duty

~~duty_block: 10:30~~

Leg

Leg

Leg

~~Leg~~

Leg

block: 5:30

block: 5:45

block: 6:30

~~block: 6:10~~

block: 4:20

Cache/Range

- When the range is affected, the cached value is invalid. Mainly important when chains are rebuilt often. Not so important in APC when column generation is used.

- Reuse variables that may be cached to improve Studio performance, i.e. don’t have 10 variables for the same thing.

Hint: Look at the compilation warnings in DWS

Caching

From C14, Rave is also able to (manually) cache functions:

```
%cache_function%(Int arg cache set(1,10)) = …
```

`set()` works like the foreach sets in constraints.

This is useful when you allow 40 bids, but crew normally only have 4 or 5:

```
%awarded_points%(Int bid_nr cache set(1,5)) =
    if %bid_fulfilled%(bid_nr)
    then %bid_points%(bid_nr)
    else 0;
```

Performance

Caching

Stepwise definitions:

```
leg.%block_time% = arrival – departure;
duty.%block_time% =
    sum(leg(duty), leg.%block_time%);
trip.%block_time% =
    sum(duty(trip), duty.%block_time%);
```

Performance
Range example

- We have a cost/penalty on leg connection time, dependent on a trip attribute

- How should a cost like this be written for efficiency?

Performance

Range example

As one trip cost, with stepwise definitions?

```
%trip_tot_penalty% =
    sum(duty(trip), %duty_tot_penalty%);

%duty_tot_penalty% =
    sum(leg(duty), %leg_penalty%);
```

Performance

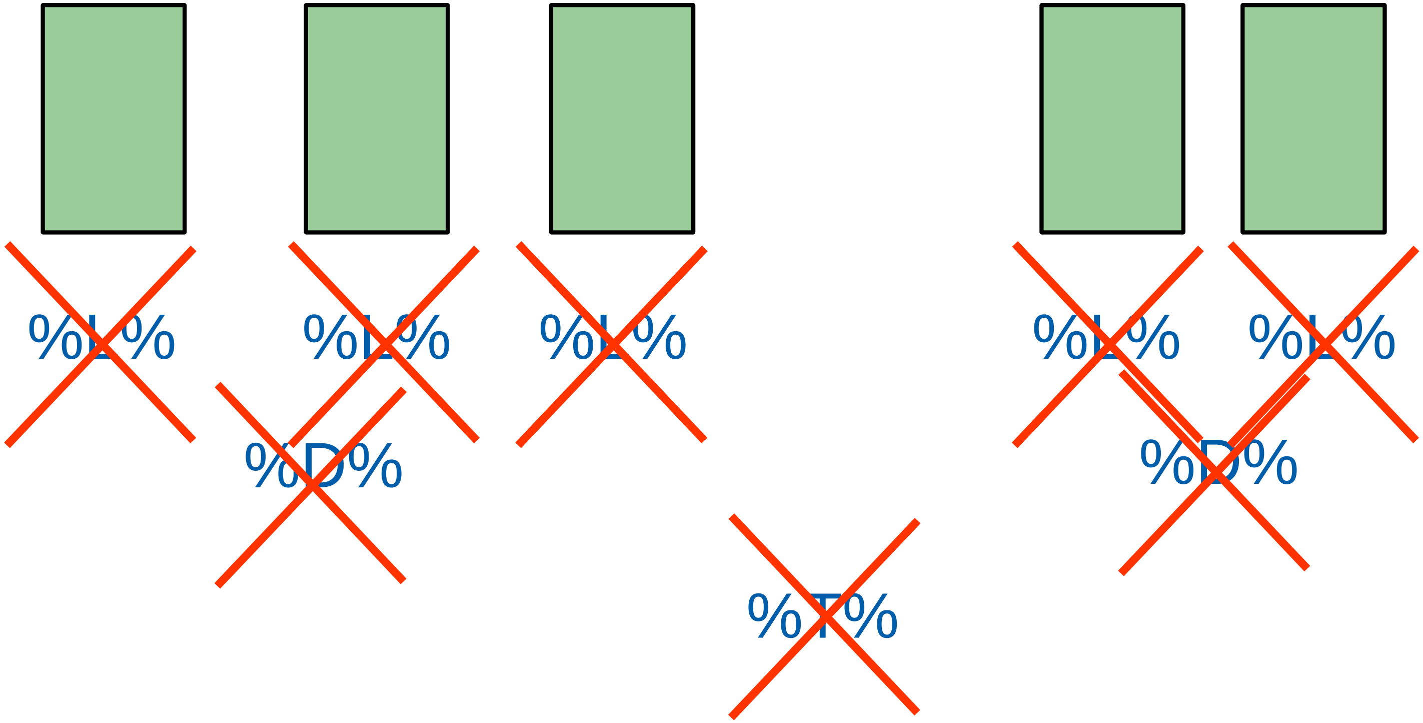
Range example

```
%leg_penalty% =
    if %trip_attribute%
    then cost1 * leg_attr
    else cost2 * leg_attr;
```

- The range of `%leg_penalty%` is trip
- Each time the trip changes, `%leg_penalty%` needs to be re-evaluated on every leg.

Performance

Range example



~~%L%~~ ~~%L%~~ ~~%L%~~ ~~%L%~~ ~~%L%~~

~~%D%~~ ~~%D%~~

~~%T%~~

Performance

Range example

Instead, define different `leg_`... and `duty_`... variables for each `trip_attr` and test at trip level:

```
%leg_penalty_cost1% = cost1 * leg_attr;
%duty_penalty_cost1% =
    sum( leg(duty), %leg_penalty_cost1% );
```

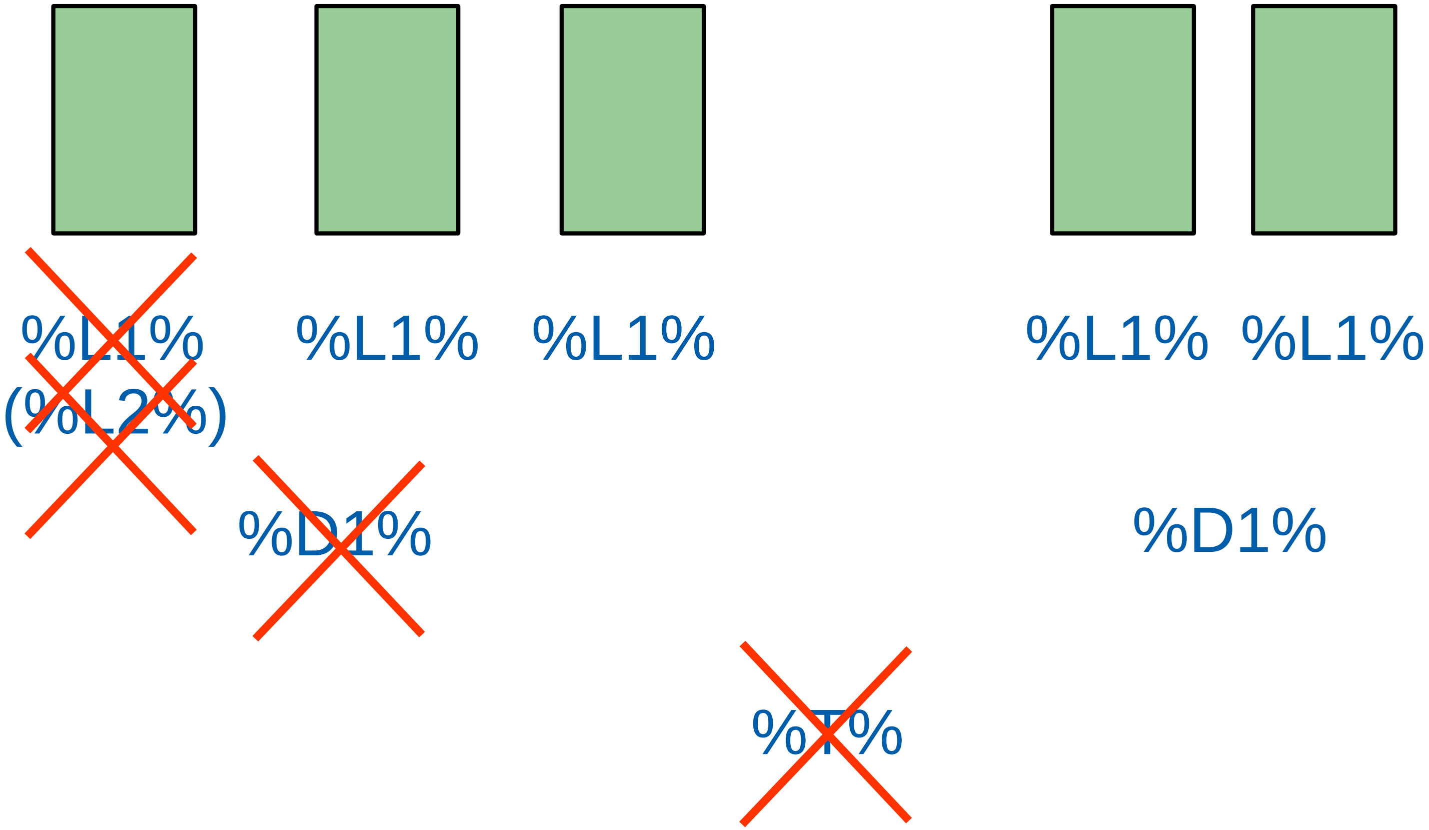
Performance

Range example

```
%trip_total_penalty% =
    if %trip_attribute% then
        sum(duty(trip), %duty_penalty_cost1%)
    else
        sum(duty(trip), %duty_penalty_cost2%);
```

Performance

Range example



~~%L1%~~

~~(%L2%)~~

%L1%

%L1%

%L1%

%L1%

~~%D1%~~

%D1%

~~%T%~~

Performance

Break

WALLY, WHAT ARE YOUR GOALS FOR THE COMING YEAR?
MY GOAL IS TO REPLACE MY SOUL WITH COFFEE AND BECOME IMMORTAL.
I MEAN SOME-THING ABOUT WORK.
OH, I THOUGHT YOU SAID MY GOALS.
scottadams@aol.com
www.dilbert.com
2-8-07 ©2007 Scott Adams, Inc./Dist. by UFS, Inc.

Performance
Analysis

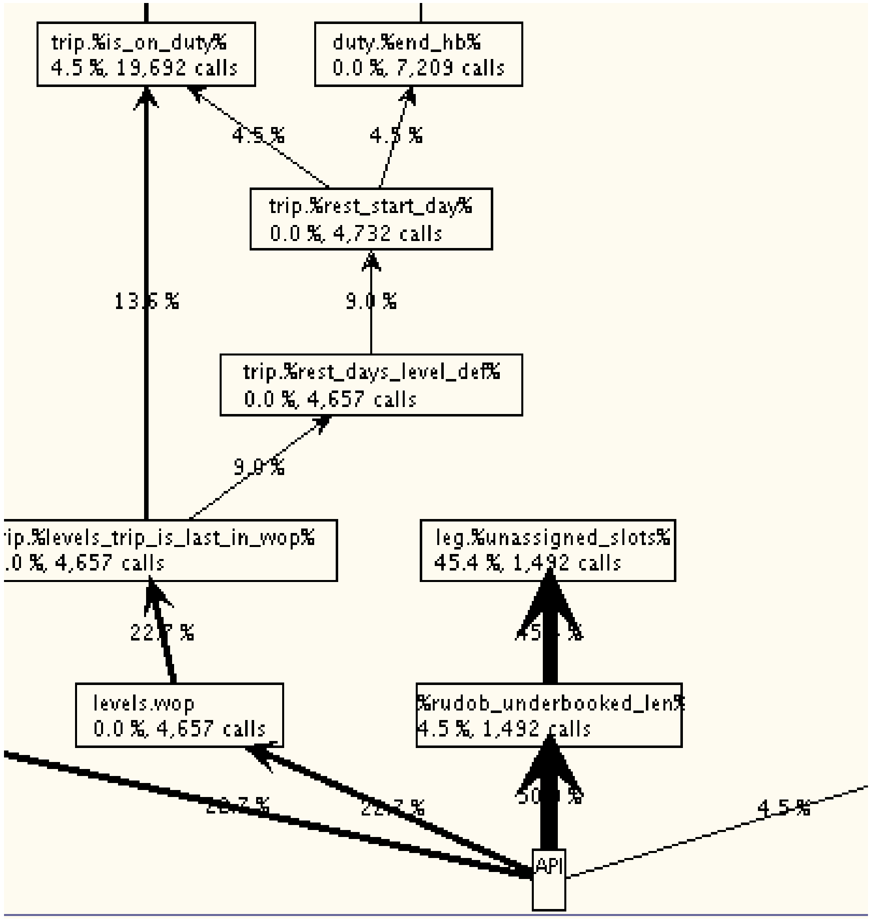
Pairing : `apc.crc_runtime_statistics`

- Time spent evaluating rules
- Rules that are never broken
- Rules shadowing each other
  - 1 week block_time < 20:00
  - 4 week block_time < 80:00

Performance

Analysis: Rave Profiler

Shows how time is spent evaluating Rave variables when a rule set is used



Performance

- Simplify rules – make them more general
- Easy and fast valid* statements
- Start with fast statements
  that cut the most obvious cases first
- Make use of lazy evaluation
  e.g. second operand to and, or only evaluated if needed:

```
A and B ≡ A  if A is false
A or B ≡ A   if A is true
```

Performance

Example 1

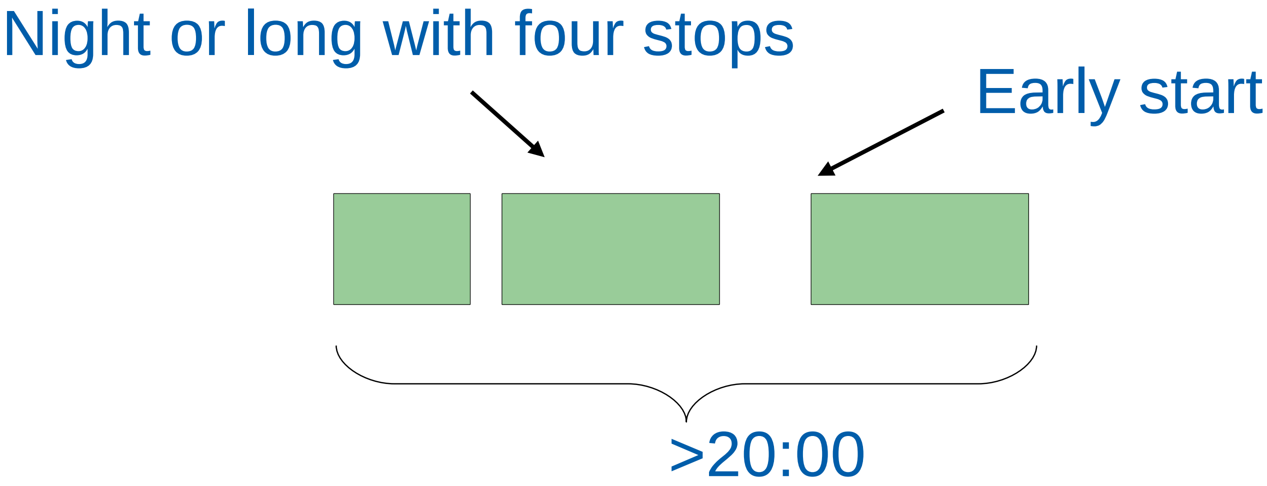
To make sure that pilots have enough time for night rest there is a rule for early starts:

“A duty may not start before 9:00 after a night duty or after long duties (>8:00) with four or more stops. There is no limitation on trips that are shorter than 20:00.”

Performance

Example 1

- First draw a picture of an illegal trip
- Decide which duty will be illegal
- …then the order in which the variables are checked.



Night or long with four stops

Early start

>20:00

Performance

Example 1

- It is more natural to make the last (second) duty illegal as the first will be legal on its own
- 20:00 trip length is easy to check
  typical valid statement
- Duty start after 9:00 is also easy to check
- Night duty involves an overlap calculation
  and four stops requires a count

Performance

Example 1

```
rule no_early_start_after_night =
    valid %trip_length% > 20:00
          and not is_first(duty(trip))
          and %check_in% < 9:00;
    not prev(duty(trip),
        %is_night_duty%
        or %long_with_4_stops%);
```

valid

condition
one variable

Performance – example 2

Real life example:

```
%block_time_per_24_hours% =
    let p_end = arrival,
        p_start = p_end - 24:00;
    if not %VALID% then
        0:00
    else
        sum(leg(roster),
            overlap(departure, arrival, p_start, p_end))
        where (%leg_flight_duty% and not deadhead
               and arrival >= p_start
               and departure < p_end);
```

Performance – example 2

Some performance considerations:

```
%block_time_per_24_hours% =
    let p_end = arrival,
        p_start = p_end - 24:00;
    if not %VALID% then
        0:00
    else
        sum(leg(roster),
           overlap(departure, arrival, p_start, p_end))
        where (%leg_flight_duty% and not deadhead
               and arrival >= p_start
               and departure < p_end);
```

Performance – example 2

Severe problem: bad range

```
%block_time_per_24_hours% =
    let p_end = arrival,
        p_start = p_end - 24:00;
    if not %VALID% then
        0:00
    else
       sum(leg(roster),
           overlap(departure, arrival, p_start, p_end))
       where (%leg_flight_duty% and not deadhead
              and arrival >= p_start
              and departure < p_end);
```

… we don’t use our levels.

Performance – example 2

Change to:

```
%block_time_per_24_hours% =
    let p_end = arrival,
        p_start = p_end - 24:00;
    if not %VALID% then
        0:00
    else
        sum(leg(wop),
            overlap(departure, arrival, p_start, p_end))
        from (current)
        backwards
        while (arrival > p_start)
        where (%leg_flight_duty%
               and not deadhead);
```

~50 times faster in Studio
even more in Matador!

Performance

Rules/Costs

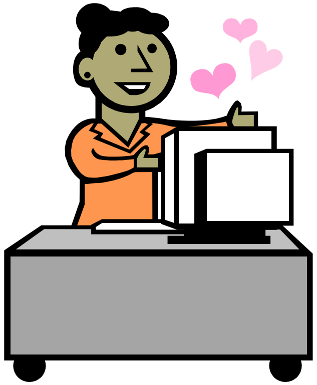
- Use rules to limit the generation (rostering)
- Usually hard rules are better than costs
- It is better to generate a few good rosters than to optimize many bad ones

- APC (Column Generator) works well with costs
- Rostering (Matador) is also making the shift to column generation.

Exercise 6

~75 mins

Exercise 6 summary



Chapter 7

Costs

Rules

Costs

Costs:

- are used by the optimizer to find the best solution
- set the objective for the optimizer:
  - productivity
  - fairness
  - robustness
  - PBS
  - quality of life
- …by making unwanted patterns less attractive.

Cost function examples

- Cost of trip/roster
  - Calculate a price for the current trip/roster
- Roster Initial cost
  - Special cost function used only by roster initial method
- Generation cost
  - Cost used to sort trips in Crew Rostering
- Overcover cost
  - Used in Crew Pairing when a leg is covered more than once (since we do not know which one(s) will become deadheads yet)

Cost of trip/roster

- Access the whole chain
- Should cover all cost aspects that can be calculated on a per-chain basis
- Low (no) cost on things you want, high on others
- Negative cost on granted bids.

Crew Pairing:

```
  apc_pac.%map_cost_of_crr% = “cost.of_trip”;
```

Crew Rostering:

```
  matador.%map_cost_of_roster% = “cost.of_roster”;
```

Roster Initial

- Is used when building the first optimization solution in rostering
- Include cost for important and long trips, broken/granted bids …

```
matador.%map_cas_initial_cost% =
     “cost.cas_initial”;
```

Cost of trip

Divide costs into groups and sub-groups

- legality
- quality
- bids
- layovers ...

```
cost.%of_trip% =
   %cost_of_deadheads%
   + %cost_of_layovers%
   + %cost_of_short_breaks%
   + %cost_of_working_day%
   + ...
```

Cost function

Make it possible to turn groups on/off:

```
%use_cost_of_layovers% =
    parameter true
    remark “Use layover cost: ”;
```

Cost function

Add cost for unassigned rosters:

```
cost.%of_roster% =
    if void(crr_crew_id)
    then %cost_of_unassigned%
    else %cost_of_assigned%;
```

Different cost functions:

linear

quadratic

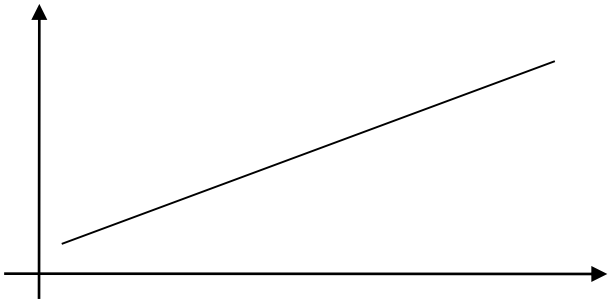
step

any function at all…

Cost function

Linear

- Easiest cost function
- Every extra item costs the same



Cost function

Linear

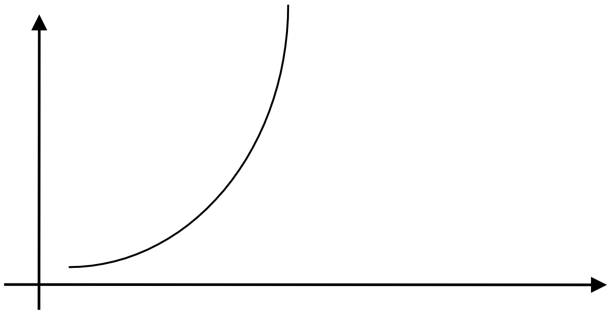
Used to limit the total number of occurrences in the solution:

```
%cost_of_duty_day% = 100;

%cost_of_unassigned_trip% = 10000;
```

Cost function

Quadratic

Each new occurrence costs much more than the previous one:



Cost function

Quadratic

Used to distribute the occurrences evenly between trips and crew.

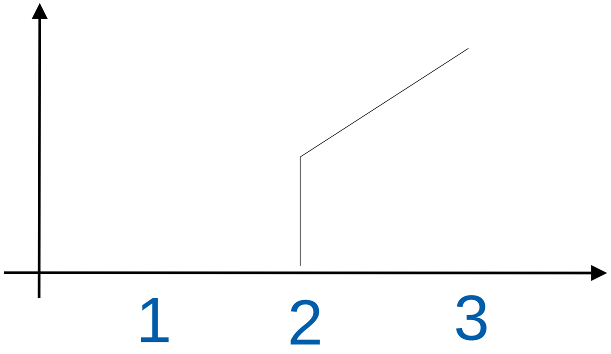
```
%cost_of_block_time% =
    let diff = %b_t_target% - %tot_b_t%;
    diff * diff;
```

```
99:00 / 99:00  ➔ 1*1 + 1*1 = 2
100:00 / 98:00 ➔ 0*0 + 2*2 = 4
```

Cost function
Step

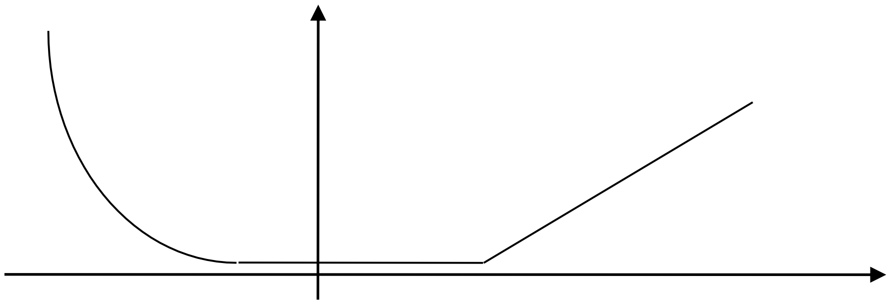
After a certain point, it starts costing more:

```
%cost_of_deadheads% =
    nmax(0, %num_dh% - 1) * 100;
```



Cost function
Any

A very complex cost function might take time to evaluate and cause loss in performance:



Costs

- When adding costs, look at the whole picture;
  - improving bid satisfaction
    usually increases production costs
- Look at the relation between costs
  - Do not start a costs race or inflation
  - Adding costs is like squeezing a balloon
- Watch out for overflow:
  `MAX_INT = 2^31`, the total cost should be far from that.

Note: To design costs and rules that work well with the optimizers is a difficult task.
Take the optimizer specific Rave courses before you do it!

Costs

Costs must be possible to understand for the planners.

Make them:

- **Transparent**

  Show all (active) cost elements as Custom KPIs and/or in report

- **Intuitive**

  All cost elements must be easy to understand

- **Consistent**

  Use consistent naming and consistent sorting in:

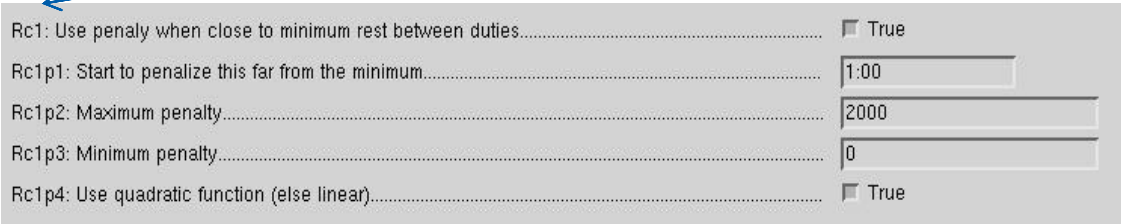
Rave parameter form

Reports

Custom KPI

Documentation

Identifiers can help



Rc1: Use penaly when close to minimum rest between duties........ □ True

Rc1p1: Start to penalize this far from the minimum........ 1:00

Rc1p2: Maximum penalty........ 2000

Rc1p3: Minimum penalty........ 0

Rc1p4: Use quadratic function (else linear)........ □ True

Rules

Recap

Rule hierarchies

The failure context

Rule exceptions

Rules

Recap

```
rule r =
    valid %v%;
    %a% < %l%;
end
```

Outcome of rule ”r”		%v%		
		TRUE	FALSE	VOID
%a% <%l%	TRUE	PASS	PASS	PASS
	FALSE	FAIL	PASS	PASS
	VOID	PASS	PASS	PASS

Use lazy evaluation, simple valid statements and actual value compared to a limit value. %v% and `%a% < %l%` should have the same dependency. This will be the dependency of the rule. The compiler will produce errors and warnings if the dependencies do not match.

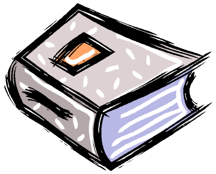
Rules

Rule hierarchies

Rules may be grouped by parent rules to indicate that they belong together.

Example:

```
rule (on) parent_LH =
    valid %is_LH%;
    sub_lh is legal;
    sub_fd is legal;
end
```



See
Rave Reference > Definitions > Rules
for more information on Rule hierarchies

Rules

Recap

Rule hierarchies

The failure context

Rule exceptions

Rules

`failure` iterator

For a given chain, the `failure` iterator provides specific information regarding the rules that are broken on that particular chain.

The `failure` iterator may be used either from a PDL report or from the Python Rave API (PRT). The `CheckLegality.py` report is an example where this information is used.

A dynamic `failtext` value can be defined in Rave, instead of the static remark, to give better support to planners:

```
failtext concat("Unknown trip code: ",
                trip.%code%);
```

Rules

failure iterator

The following values can be accessed from the Rave API.

The variables that can be used in PDL differ slightly.

index	Comment
rule	the rule itself
failtext	str
overshoot	int or RelTime, None if not applicable
actualvalue	Any Rave data type. None if not applicable.
limitvalue	Any Rave data type. None if not applicable.
startday	int, None if not defined
endday	int, None if not defined
startdate	AbsTime, None if not defined
enddate	AbsTime, None if not defined

Rules

`failure` iterator

The attributes `startday, endday, startdate` and `enddate` are explicitly defined in the rule by the Rave programmer.

They are primarily used when creating rule exceptions.

```
rule max_wop_duty_days =
    valid %v%;
    %actual_wop_value% <= %limit%;
    startdate = wop.%start%;
end
```

Rules

Recap

Rule hierarchies

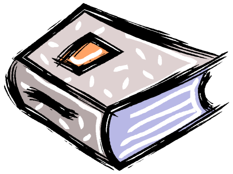
The failure context

Rule exceptions

Rule exceptions

Rule exceptions are used to relax or completely ignore a rule for a particular time period and crew:

- This crew, this rule at this time is legal
- This crew, this rule at this time has a relaxed limit value



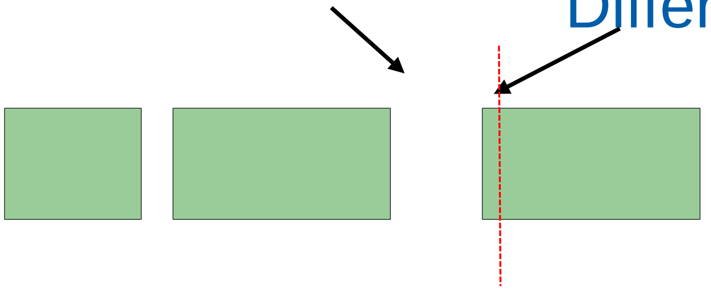
See
Rave Reference > Appendix: Modelling Information > Rule exception mechanism

Rules

Rule exceptions

A rule exception will add an accepted difference from the rule limit for a specific rule for a specific crew member:

Crew Id: 12345



Minimum rest: 2:00

Difference: 0:05

Etable: 12345, “Minimum rest”, 6jun2010 04:35, 0:05;

Rule exceptions

- The attributes from the `failure` iterator may be used to produce an external table with unique rows for each crew and rule failure which should be given an exception
- This information is then fed back to the rule definition which adjusts the rule for the cases when there should be an exception
- During a rule evaluation, the keyword `current_rule_id` returns the name of the rule that is being evaluated
  In this way, `current_rule_id` can be used to match the rule exceptions in the external table with the current rule.

Rules

Rule exceptions

Example:

```
rule min_rest_before_trip =
    /* Used when creating exception */
    startdate = trip.%start%;
    trip.%rest_time%
    >= %min_rest%
       - %exception_overshoot%(current_rule_id,
                               trip.%start%);
end


table rule_exceptions(String RuleId, Abstime SDate) =
    crew.%id%, RuleId, Sdate
        -> Reltime %exception_overshoot%;
    external %etab_name%;
    CrewId, RuleId, StartDate -> Overshoot;
    -,-,- ->  0:00;
end
```

Rule exceptions

Example continued:

1. Rule `min_rest_before_trip` on CrewID 1010 fails on a trip that starts on 01Mar2006
   Rest time is 2 hours too short

2. ‘Create Rule Exception’ script creates a new line in `RuleExceptions.etab`:

```
Crewid, RuleId ,                StartDate, Overshoot
1010,   “min_rest_before_trip”, 01Mar2006, 2:00
```

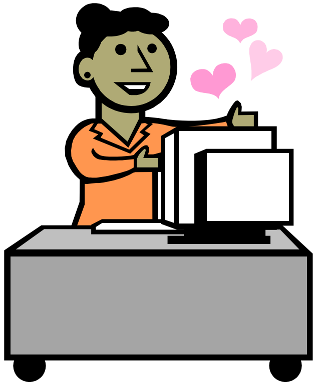
3. The next time the rule is checked, `%exception_overshoot%(…)` will return 2:00, making the rule legal

Exercise 7

~45 mins

Exercise 7 summary

- `cost.%roster%`



Chapter 8

Accumulators

Accumulators

- Accumulators make it possible to model rules spanning over long periods of time, both historical and future
- Accumulators can use stored data in order to calculate values outside the time span of the current plan
- Accumulators can use a projection function in order to calculate values that are later than the current plan

Accumulators

2 things can be done with an accumulator:

- **Evaluate**
  - Given a certain plan and stored (historic) data, evaluate the accumulator function for a particular interval

- **Accumulate data**
  - Given a certain plan and stored (historic) data, add the data for the current plan to the stored data.

Accumulators

First, let’s see how we would solve the following problem without an accumulator:

- Calculate the total block time between two time points

- If both time points are inside the plan interval we can use a straightforward rave function:

```
%acc_block_plan%(Abstime s, Abstime e) =
    sum(leg(roster), %leg.%block_time%)
    from(first where arrival > s))
    forwards while(arrival <= e);
```

Accumulators

For an interval entirely outside the plan we would need to use external table which contains the historical accumulated block time:

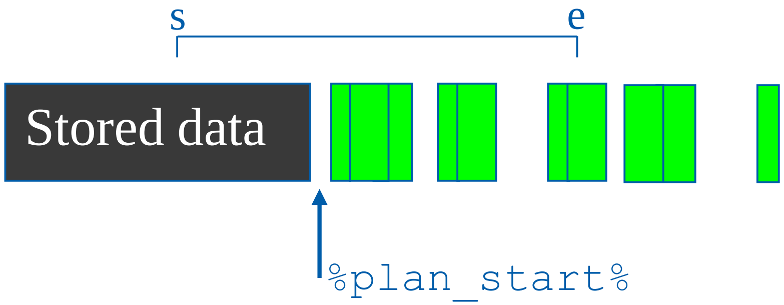
```
%acc_block_lookup%(Abstime s,Abstime e) =
    %acc_block_hist%(e) - %acc_block_hist%(s);

table acc_block_history(Abstime t) =
    %crr_crew_id%, t
        -> %acc_block_hist%;
    external ”acc_block.etab”;
    crew_id, time_point -> value;
    -,- -> void_abstime;
end
```

crew_id	time_point	value
12344	5Feb2007 0:00	166:00
12344	6Feb2007 0:00	168:00
12344	7Feb2007 0:00	171:30
17632	5Feb2007 0:00	227:40
17632	6Feb2007 0:00	232:10
17632	7Feb2007 0:00	232:10
...	...	...

Accumulators

For an interval spanning both over historical data and plan data, we need to combine the two expressions:

```
%acc_block%(Abstime s, Abstime e) =
    %acc_block_lookup%(s, %plan_start%)
    + %acc_block_plan%(%plan_start%, e);
```



For a general interval we would need lots of if-statements in order to handle all the different combinations...

... especially if we have stored data after the plan as well

... and we also need a python script to populate the etable every month.

Accumulators

We can define an accumulator, which handles everything for us:

```
accumulator  block_acc(Abstime s, Abstime e) =
    sum(leg(roster), leg.%block_time%)
    from(first where arrival > s))
    forwards while(arrival <= e);
    key = crr_crew_id;
    plan_start = pp.%start%;
    plan_end = pp.%end%;
    ...
end
```

Attributes `plan_start` and `plan_end` determine the range where plan data should be used for the evaluation

Attribute `key` is used to identify saved accumulated rows in the external table

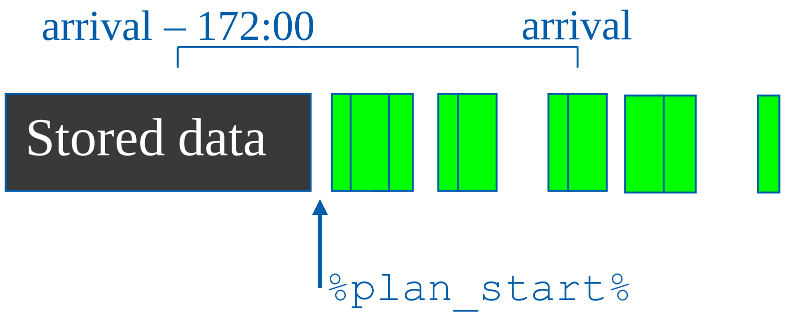
Accumulators

Evaluation

Usage:

```
%acc_172hrs% =
    block_acc(arrival - 172:00, arrival);
```

Data: (example)



will be evaluated by Rave as:

```
table lookup(arrival - 172:00, plan_start)
+ plan(plan_start, arrival)
```

Accumulators

Accumulation

- Accumulated data is stored in specific etables
  name of table can be defined with attribute `external`
- To accumulate the content of the chains in the window, use **`general-popup> Rave> Accumulate`**
- With python:
  `carmensystems.rave.utils.eval_accumulators()`
- The range that will be accumulated is defined by the accumulator attributes `acc_start` and `acc_end`
- The `acc_next` attribute determines the step size of the accumulation.

Accumulators

Accumulation

During accumulation, for each data point the following information is stored:

```
(accumulator_id, key, time_point, value)
```

There is one table for each data type of accumulator values: accumulator_int, accumulator_rel, accumulator_time (Abstime)

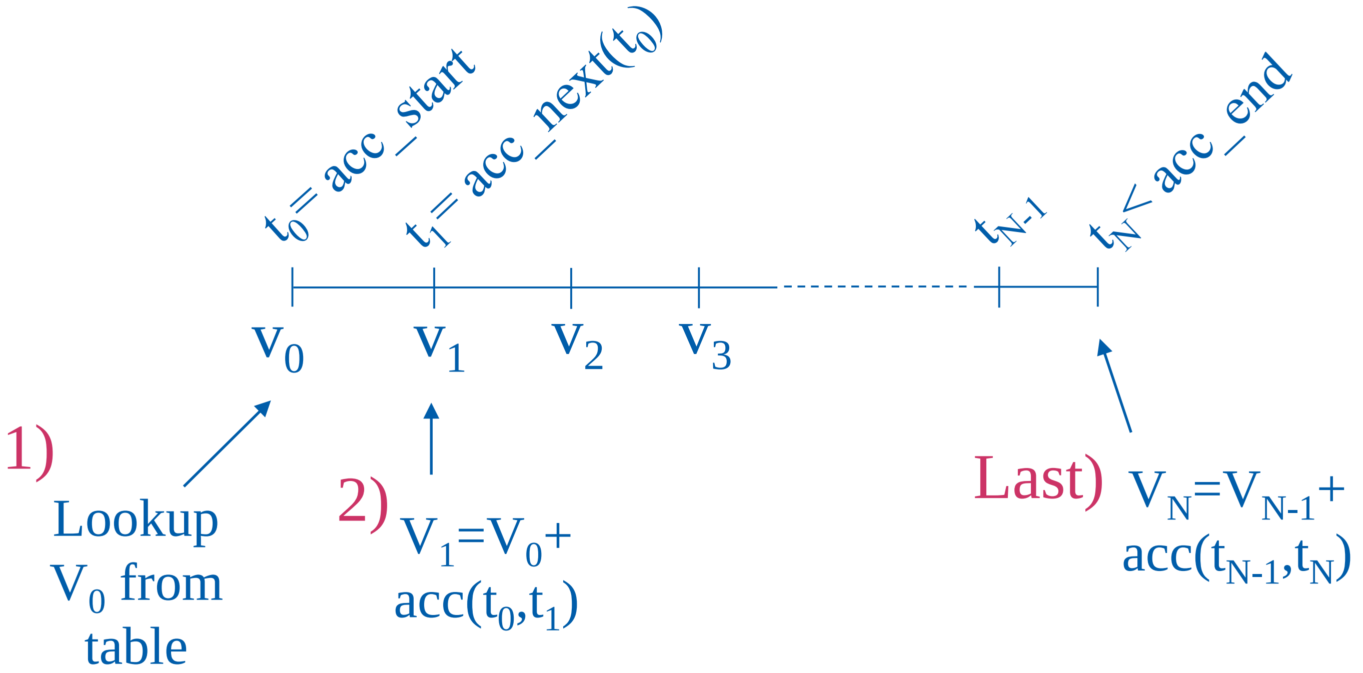
Accumulation attributes (cont.):

```
accumulator block_acc(Abstime s, Abstime e) =
    ...
    acc_start = pp.%start%;
    acc_end = pp.%end% + 0:01;
    acc_next(Abstime t) = t + 24:00;
end
```

Accumulators

Accumulation

The acumulation process:



For each i=1,...,N store (acc_id, key, t_i, V_i)

Accumulators

A full accumulator example:

```
pp.%start% = ”01[month][year]0:00”
pp.%end% = ”01[month+1][year]0:00”
```

```
accumulator block_acc(Abstime s, Abstime e) =
    sum(leg(roster), leg.%block_time%)
    from(first where arrival > s))
    forwards while(arrival <= e);
    key = crr_crew_id;
```

Evaluation

```
    plan_start = pp.%start%;
    plan_end = pp.%end%;
    proj_interval_func = (e-s)/5
```

Accumulation

```
    acc_start = pp.%start%;
    acc_end = pp.%end% + 0:01;
    acc_next(Abstime t) = t+24:00;
end
```

Accumulators

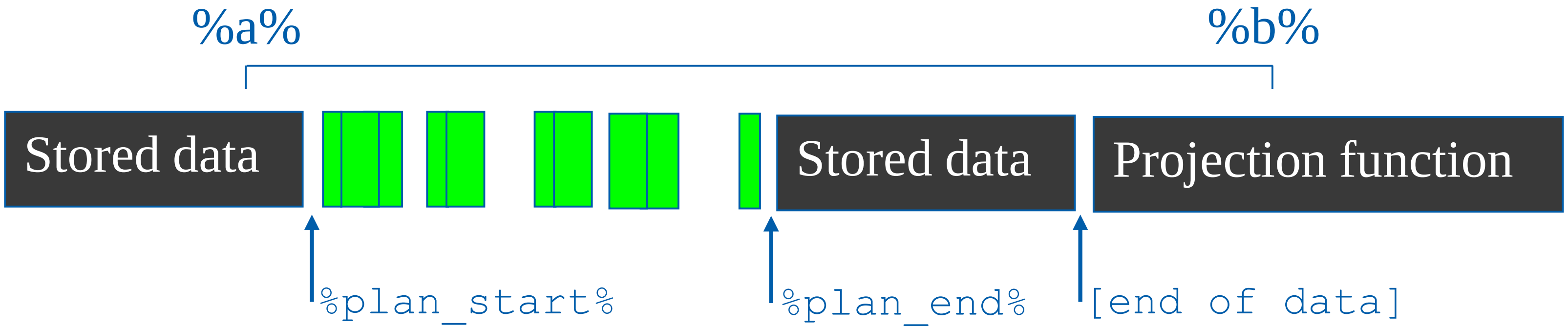
More about the evaluation:

`proj_interval_func` (optional)

- Used to predict the future (guess)
  after the plan and after any saved data

Accumulators

Another example:



`acc(%a%,%b%)` will be evaluated by Rave as:

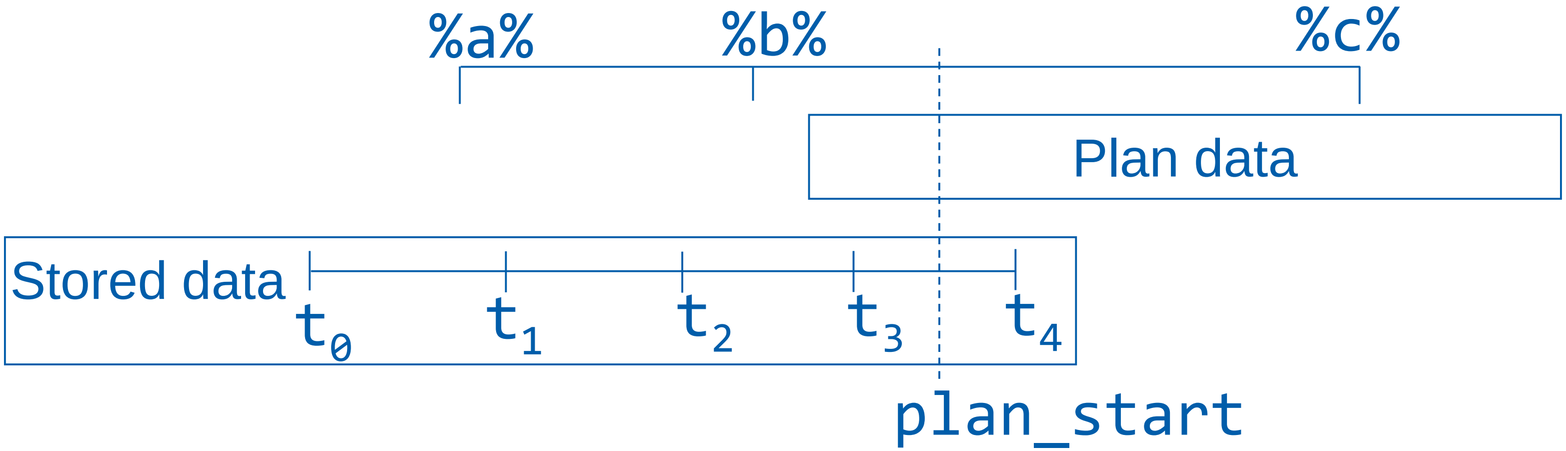
```
table lookup(%a%, plan_start)
+ plan(plan_start, plan_end)
+ table lookup(plan_end, [end_of_data])
+ proj_interval_func([end_of_data], %b%)
```

Break

Accumulators

More about the evaluation

When interval points are not exactly in the stored data:



Result: acc(%a%,%b%)-> lookup(t_1,t_2)

acc(%a%,%c%)-> lookup(t_1,t_3) + plan(t_3,%c%)

Accumulators
Performance

Evaluation performance:

- Hash keys match `accumulator_id` & `key`
- Linear search to find the right time point.

This means:

- Lots of crew and accumulators do not affect performance significantly
- Lots of history (many data points) may affect performance significantly.

Accumulators

The accumulator etables:

Name: accumulator_int, accumulator_rel, accumulator_time
or defined by attribute `external`

Empty tables must be created as a start

The tables must reside in one of the directories specified by the studio resource `default.default.CRC_ETAB_PATH`

(default value: ”`$(CARMUSR)/crc/etable`”)

Each table has the following columns:

	Accumulator Name	**Key**	**Time Point**	**Value** (Void for abstime tables)
Column name	**`name`**	**`acckey`**	**`tim`**	**`val`**
Datatype	String	String	Abstime	`accumulator_int`:Int `accumulator_rel`:Reltime

Accumulators

More about the accumulation:

Stored data after the plan will be adjusted if you accumulate the data in the plan.

	…	t_{55}	t_{56}	t_{57}	…	t_{101}	t_{102}	t_{103}	t_{104}
Original data		105	108	111	…	278	283	291	302
re-accumulated data (from plan)			108	114	…	286	288		
Adjusted data							Δ=+5	296	307
Final data		105	108	114	…	286	288	296	307

Accumulators

An abstime example:

```
accumulator last_dh(Abstime t, Bool forward)=
    if forward then
        first(duty(chain), duty.%end_date%)
        where (duty.%start% > t
               and duty.%is_deadhead%)
    else
        last(duty(chain), duty.%end_date%)
        where(duty.%start% < t
              and duty.%is_deadhead%);
    key = crr_crew_id;
    acc_start = %pp_start%;
    acc_end = %pp_end% + 0:01;
end
```

Accumulators

More on abstime accumulators:

- The attribute `acc_filter` can be used for Abstime accumulators
- It is used to populate the `filt` column in the accumulator_time table
- …with a value that can be used for filtering.

Accumulators

More on reltime and int accumulators:

- The `acc_force_store` attribute can be used for Int and Reltime accumulators
- It forces specific time points to be stored (for example January 1 each year).

Accumulators

A final usage example:

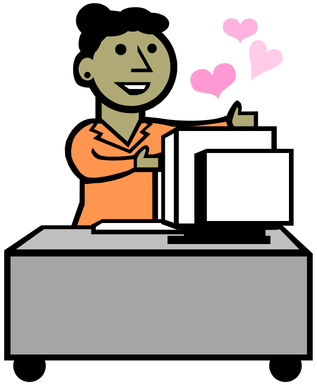
Rule: Max 48:00 hours block allowed within 172 hours

```
accumulator block_acc(Abstime s, Abstime e)=
    ...
end
rule max_acc_block_in_172_hrs =
    block_acc(arrival - 172:00, arrival)
    <= 48:00;
```

Exercise 8

~30 mins

Exercise 8 summary



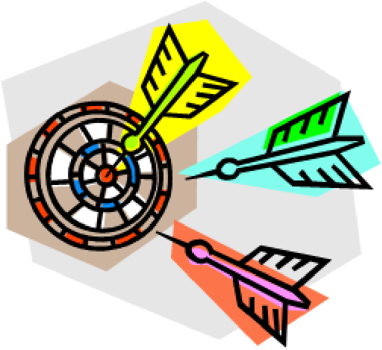
Chapter 9

Summary

Course summary

You have learned:

- advanced Rave features
- how to use modules
- how to write cost functions
- how caching works
- to avoid illegal sub-chain problems
- how to do performance analysis
- Rave accumulators.



Other Rave courses

- Rave Publisher I, II – Reports (PDL)
- PRT (Python Report Toolkit)
- Rave for pairing optimization
- Rave & Python for rostering opt.

Course Evaluation

Please take a few minutes to complete the evaluation form, it will help us improve the courses for you and your colleagues:

```
Special> Academy> Course Evaluation
```

Are the exercise definitions too vague (too real-life), would you like them to be more exact and straight forward?

Would you like to have even more info on slides (for self studying) or would you be stressed about the time constraint?

The end

This was Rave II

Welcome back to Jeppesen Crew Academy!