COMPETENCE FOR BI, BIG DATA AND ESB PROJECTS

Job Design Best Practices

Job Design Best Practices

1. Parameters to steer the job
2. Naming Conventions
3. Transaction Handling
4. Error Handling
5. Understanding the job code structure
6. Typical pitfalls
7. Release Management
8. Deployment / External Resources
9. Documentation

Parameters to steer the job

- A job gets its context parameters from different possible sources and the last write action determines the actual value
  - Value set in job intern context settings
  - Value given from parent tRunJob via “Transmit whole context”
  - Value explicitly set in parent tRunJob context parameter settings
  - Value explicitly set in the TAC
  - Value read from implicit context load
- Variables can be transferred via a job which does not have explicitly defined these variables but only with the Transmit whole context function
- Context variables which steers a job or defines the work to do should be set as prompt with a description
- If a context variable points to a file set as type “File”. The file type is technical a String but shows in the prompt value dialog a file chooser dialog. Same applies to directories
- Main jobs should use implicit context load to allow external changes to the values without redeployment of the job and without Talend know-how
- Put only static variables into the context property file like database credentials but NEVER steering variables!

Job Design Best Practices

1. Parameters to steer the job
2. Naming Conventions
3. Transaction Handling
4. Error Handling
5. Understanding the job code structure
6. Typical pitfalls
7. Release Management
8. Deployment / External Resources
9. Documentation

Naming Convention

- Never build dedicated items (e.g. database connections) for deployment layers like (Test / Production).
  - Use context variables to switch the layer if necessary
  - Use semantic names addressing the business purpose
- Decide the project structure!
  - Reference projects are a bad idea – currently
  - Differentiate between operative and dispositive projects
- Use always a start folder to be able to merge projects later
- Use canonical names (depends on the project purpose):
  - Source or target
  - Sub-descriptions
  - Functional name parts like (_main, _worker, _trigger, _check)
- Use context variable names which are unique!
  - E.g. not “host” better “dwh_db_host”

Job Design Best Practices

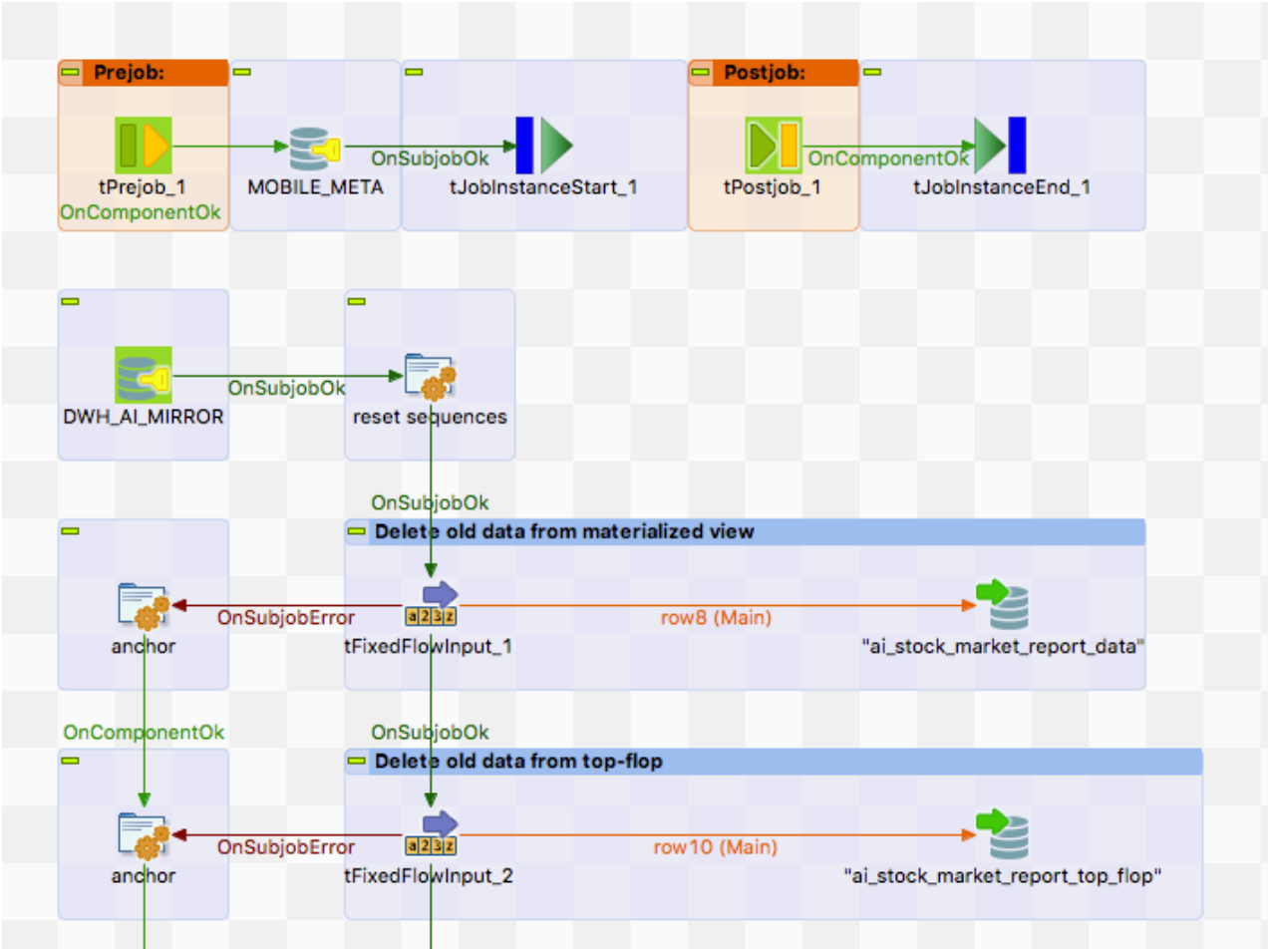
1. Parameters to steer the job
2. Naming Conventions
3. **Transaction Handling**
4. Error Handling
5. Understanding the job code structure
6. Typical pitfalls
7. Release Management
8. Deployment / External Resources
9. Documentation

Transaction Handling

- Always try to run DML statements within transactions
- Avoid creating static tables with Talend jobs, use dedicated scripts
- Register the work done only if the transaction has been finished successfully (Consider using the job instance framework)
- Check which component needs its own database connection. E.g. some database cannot cope with multiple streams at the same time (e.g. MySQL)
- Database components needing the same connection should reference a separate connection component. These connections can be shared over multiple jobs to allow a huge transaction spanning over multiple jobs – BUT it is not recommended to do that – avoid it where you can, this design is complex and error prone.
- Consider a common unified approach in your jobs

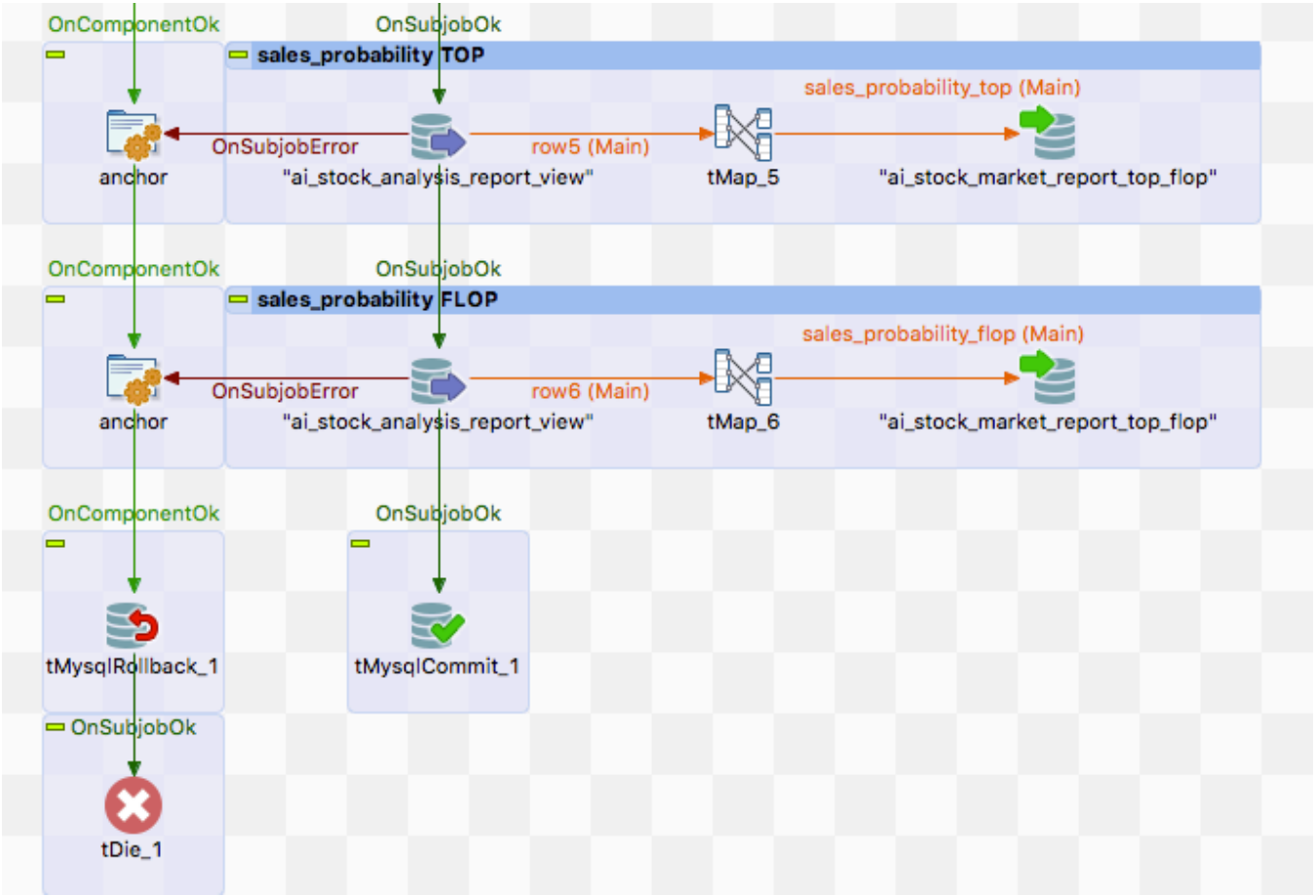
Transaction Handling – example – the start...

- Open a connection at the beginning
- Do your work within the transaction



Transaction Handling – example – at the end

- Open a connection at the beginning
- Commit or Rollback



Job Design Best Practices

1. Parameters to steer the job
2. Naming Conventions
3. Transactions
4. **Error Handling**
5. Understanding the job code structure
6. Typical pitfalls
7. Release Management
8. Deployment / External Resources
9. Documentation

Error Handling

- Register the work done only if the transaction has been finished successfully (Consider using the job instance framework)
- Define error codes:
  - Technical error (“Unknown error”)
  - Input data error (rejects or no data)
  - Resource error (Connections broken)
  - Output error (SQL constraints violated)
- For complex process structures consider a dedicated table which holds the status of all items to process and also the error state and the number of attempts.

Job Design Best Practices

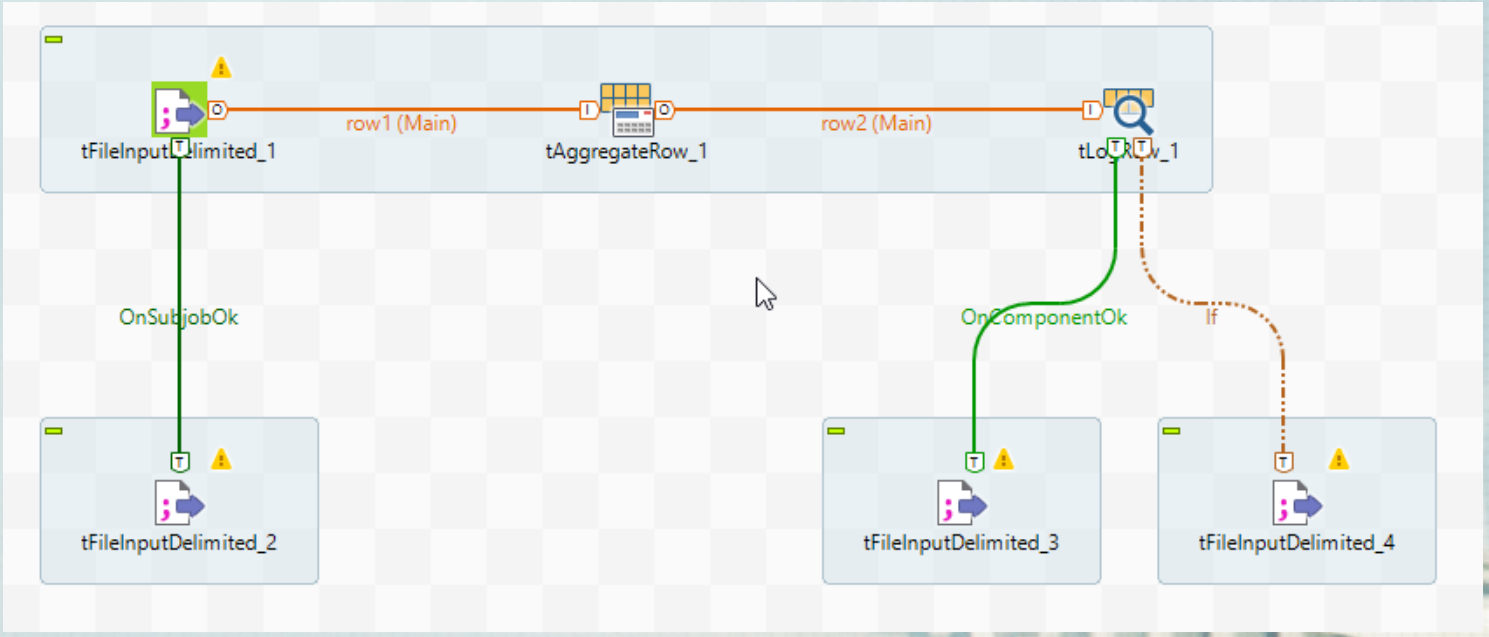
1. Parameters to steer the job
2. Naming Conventions
3. Transactions
4. Error Handling
5. Understanding the job code structure
6. Typical pitfalls
7. Release Management
8. Deployment / External Resources
9. Documentation

Understanding the job code structure

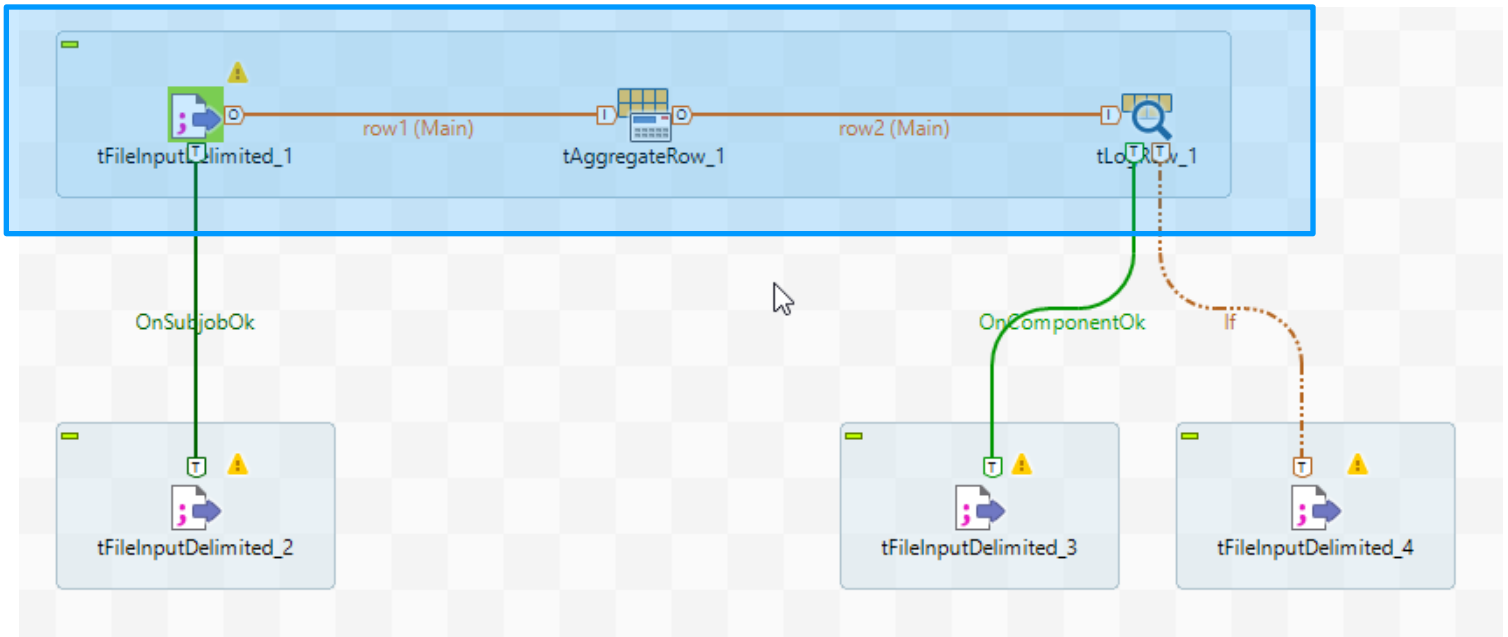
- Every component consists of **THREE** parts:
  1. BEGIN – initiate necessary resources and open a loop if needed
  2. MAIN – process data within a flow
  3. END – send final data and close resources

- Caution if the *iterate* function is used!
  - The functionality of *iterate* can be compared to a for each loop
  - The functionality connected to the *iterate* is executed once for every item found in the set
  - So keep in mind that the data flow is different from what it may seem at first sight e.g. the functionality in the blue box finishes last!

Understanding: OnSubjobOk, OnComponentOk, If



Understanding: OnSubjobOk, OnComponentOk, If



try {

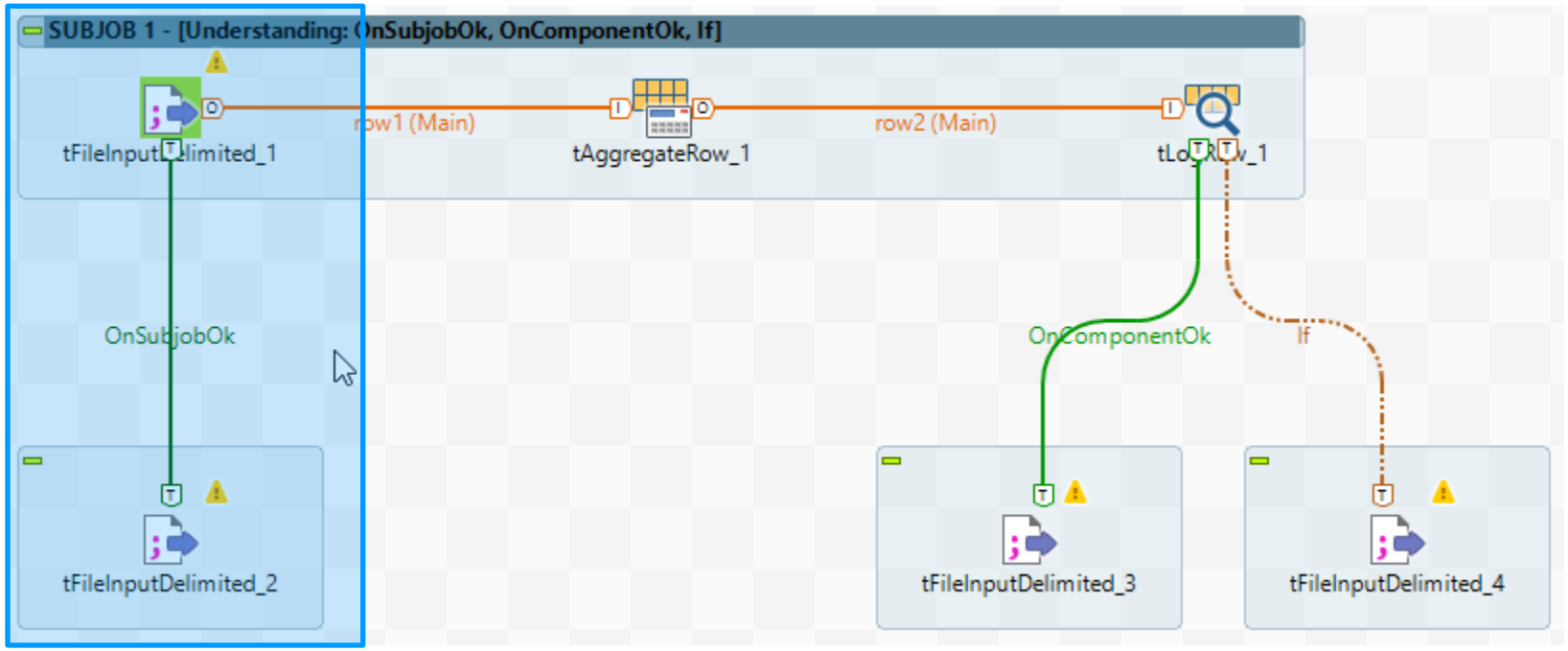
B3 **B2** **B1** **M1** **M2** **M3** **E1** **E2** **E3**

} catch {

}

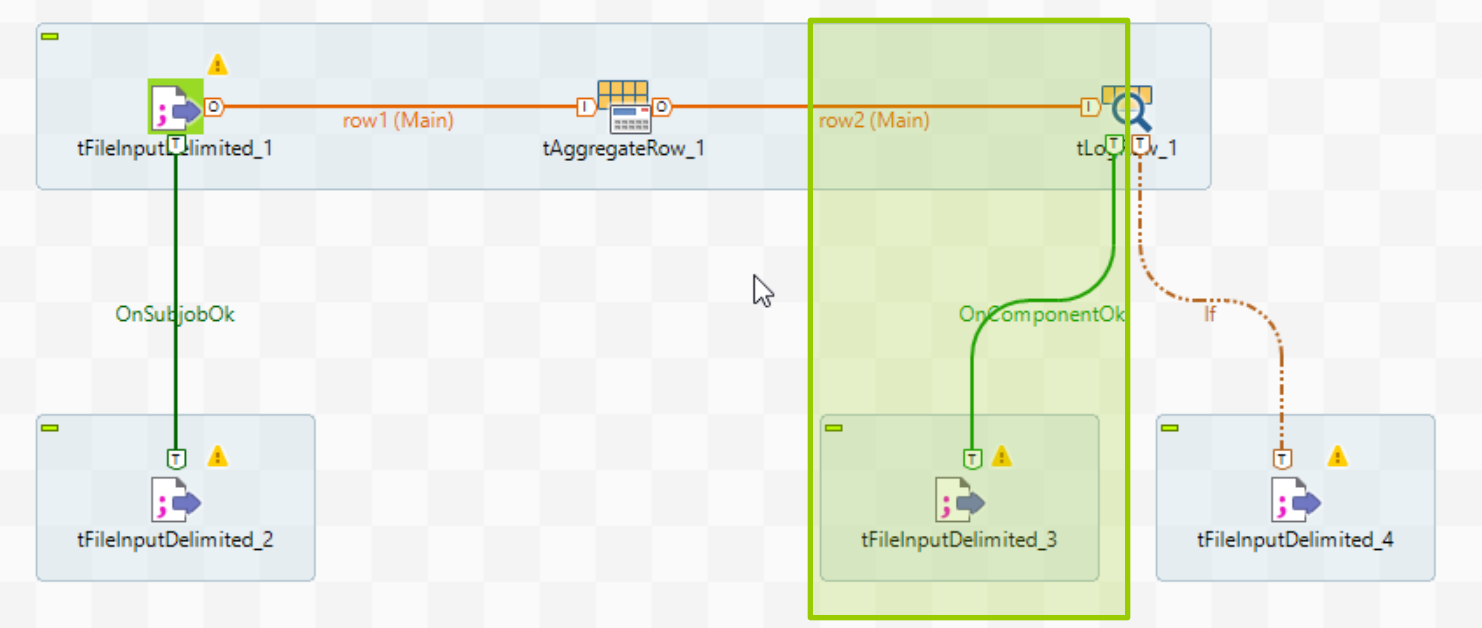
OnSubjobOk

Understanding: OnSubjobOk, OnComponentOk, If



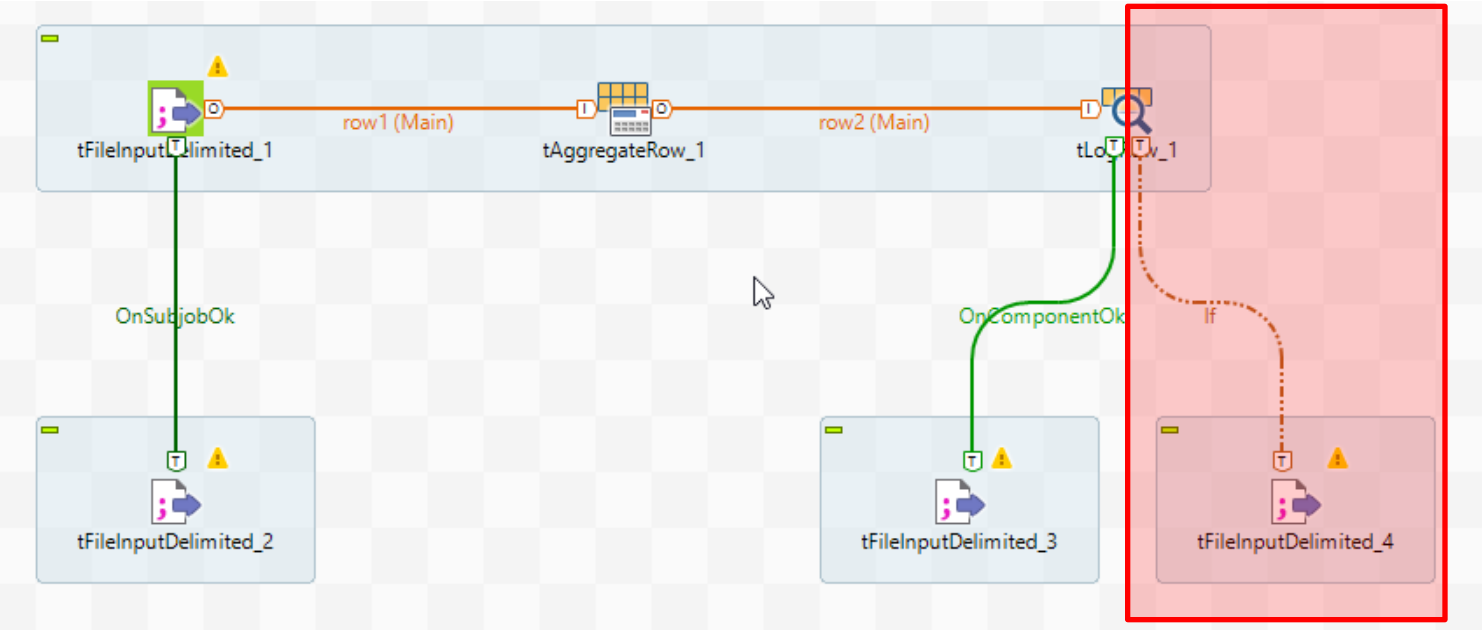
- The OnSubjobOk is only fired after the whole Subjob is finished (In this case: “SUBJOB 1”)
- The next slide shows the structure of the Subjob and at what point the trigger is fired

Understanding: OnSubjobOk, OnComponentOk, If



» Contrary to the OnSubjobOk trigger the OnComponentOk trigger is directly fired after the tLogRow component is finished (after the end-part of the component code)

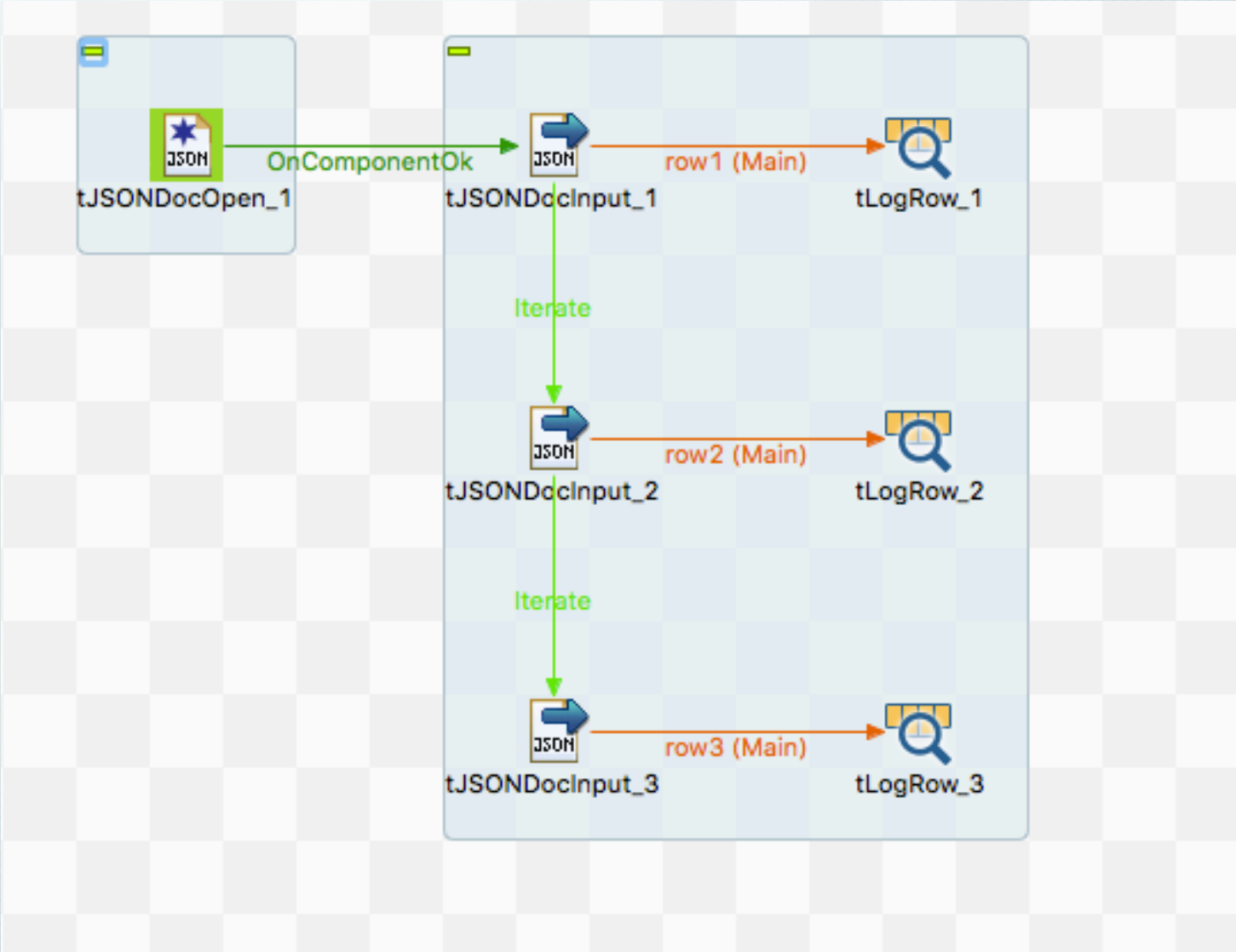
Understanding: OnSubjobOk, OnComponentOk, If



- Same as the OnComponentOk trigger the if trigger is fired directly after the component is finished.

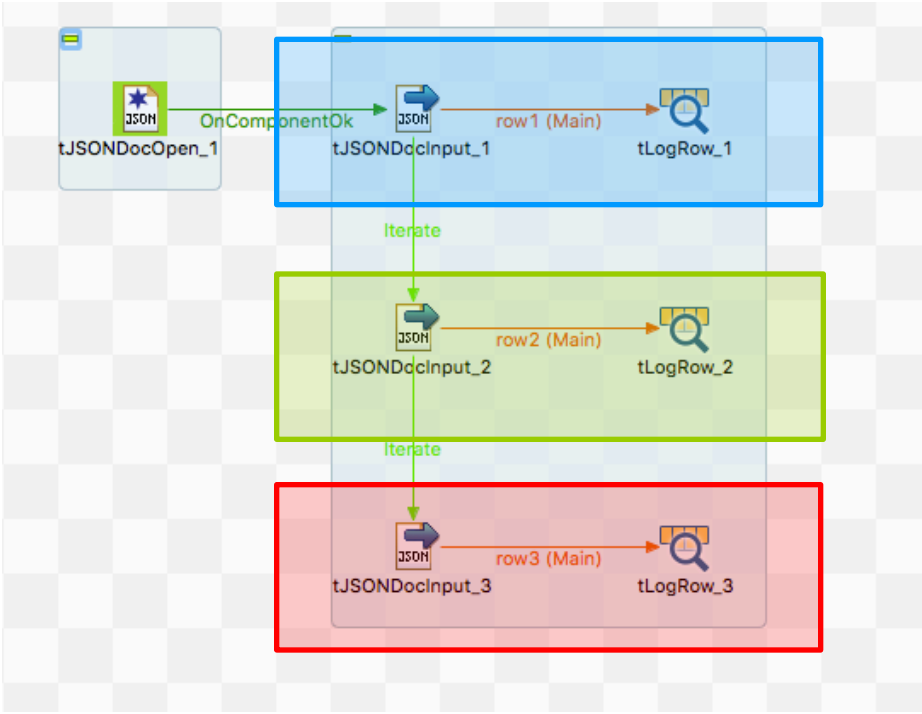
- Take note: You cannot rely on the order of the trigger between different types of triggers. But you can change the order of multiple triggers of the same kind.

Understanding the job code structure

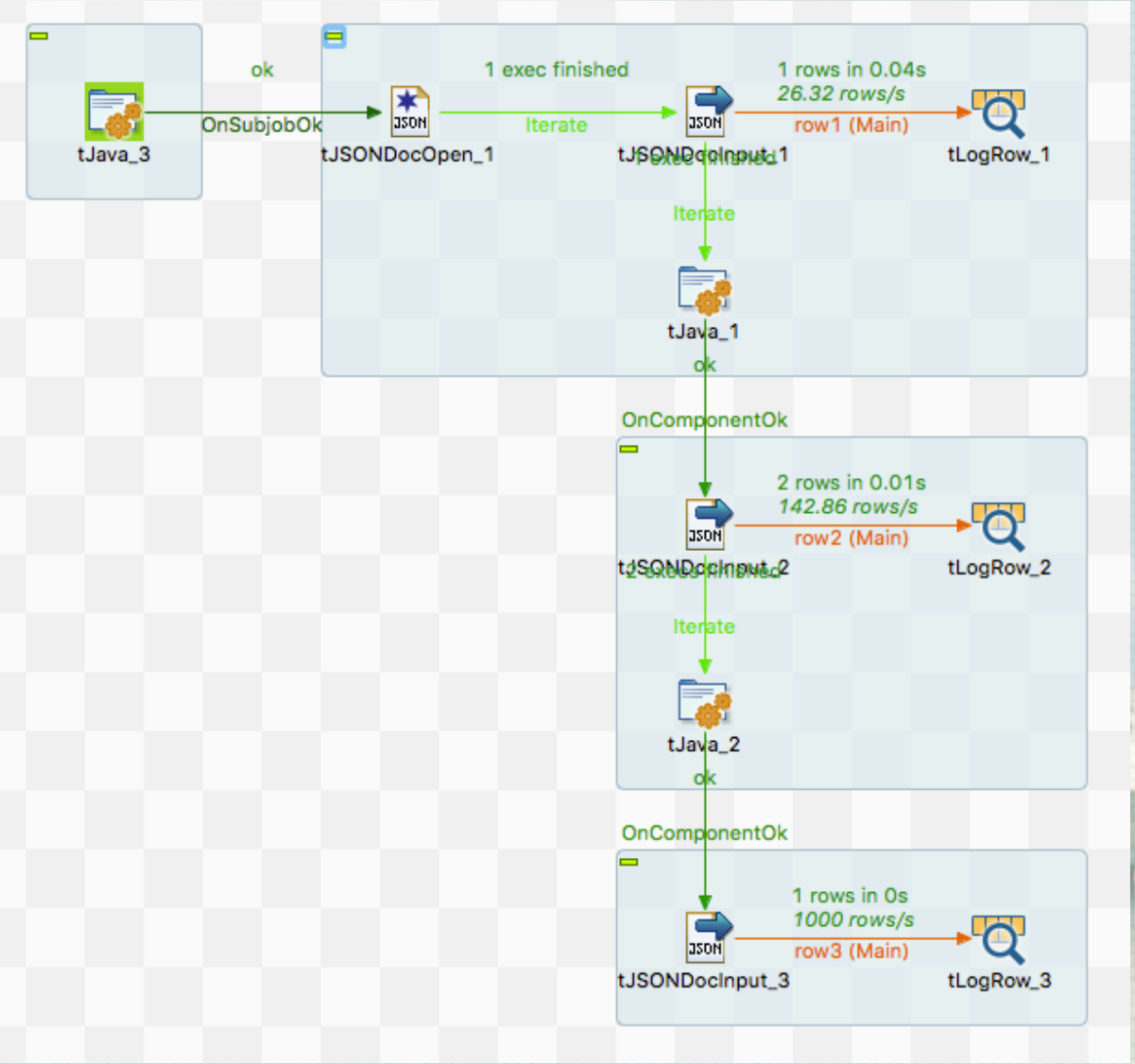


Understanding the job code structure

```
*************************
begin tJSONDocOpen_1
main tJSONDocOpen_1
*************************
begin tLogRow_1
begin tJSONDocInput_1
      main tJSONDocInput_1
      main tLogRow_1

   begin tLogRow_2
   begin tJSONDocInput_2
         main tJSONDocInput_2
         main tLogRow_2

      begin tLogRow_3
      begin tJSONDocInput_3
            main tJSONDocInput_3
            main tLogRow_3
      end tJSONDocInput_3
      end tLogRow_3
   end tJSONDocInput_2
   end tLogRow_2
end tJSONDocInput_1
end tLogRow_1
end tJSONDocOpen_1
*************************
```



Understanding the job code structure

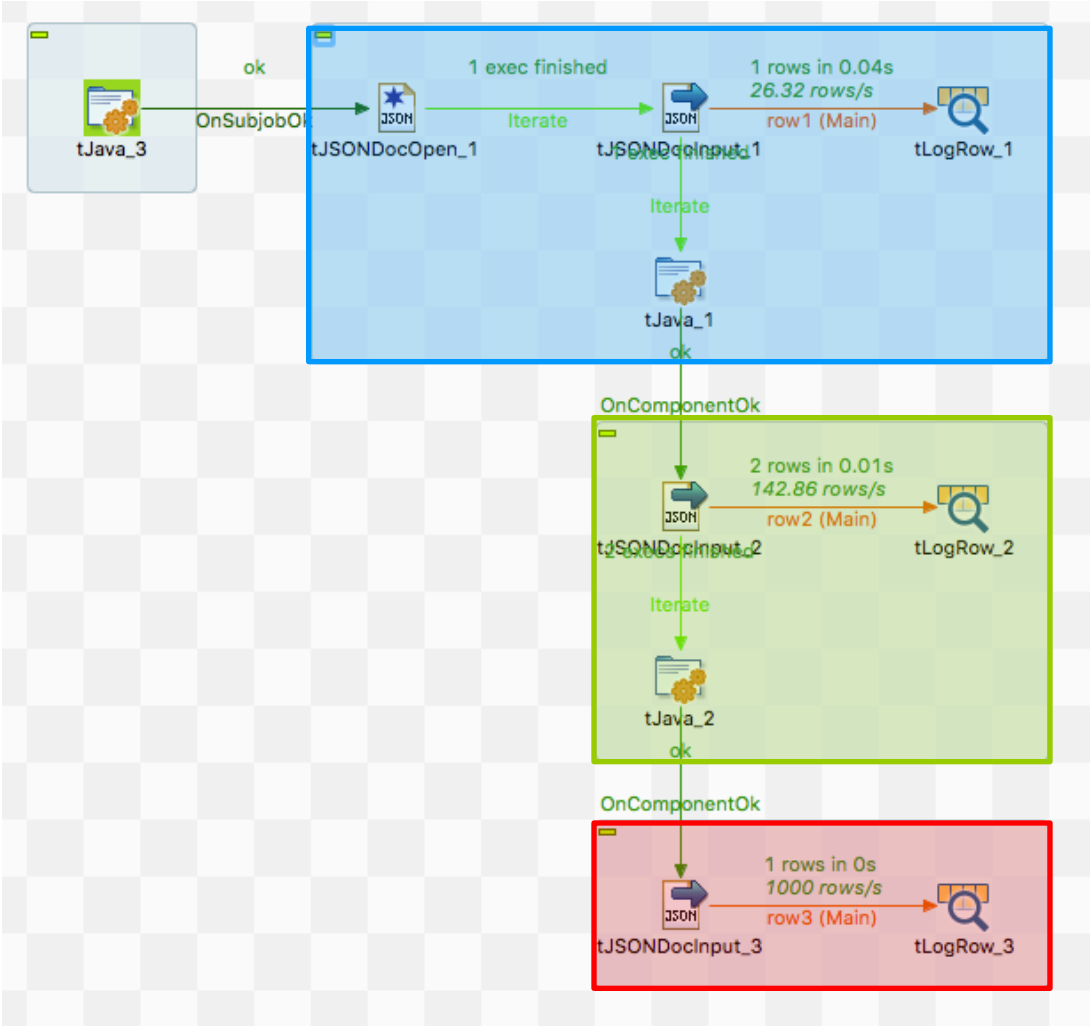


Understanding the job code structure

```
#### tJSONDocOpen_1 ####
begin tJSONDocOpen_1
main  tJSONDocOpen_1

begin tLogRow_1
begin tJSONDocInput_1
        main  tJSONDocInput_1
        main  tLogRow_1

        begin tJava_1
        main  tJava_1
        end tJava_1 --> tJSONDocInput_2
end tJSONDocInput_1
end tLogRow_1
```

```
#### tJSONDocInput_2 ####
begin tLogRow_2
begin tJSONDocInput_2
        main tJSONDocInput_2
        main tLogRow_2

        begin tJava_2
        main tJava_2
        end tJava_2 --> begin tJSONDocInput_3
end tJSONDocInput_2
end tLogRow_2
```

```
#### tJSONDocInput_3 ####
begin tLogRow_3
begin tJSONDocInput_3
        main tJSONDocInput_3
        main tLogRow_3
end tJSONDocInput_2
end tLogRow_2
```



Understanding the job code structure

- Also be careful order of execution methods are called through Java
- Much like with iterate the data flow is not like it seems at first sight

Job Design Best Practices

1. Parameters to steer the job
2. Naming Conventions
3. Transactions
4. Error Handling
5. Understanding the job code structure
6. Typical pitfalls
7. Release Management
8. Deployment / External Resources
9. Documentation

Typical pitfalls – tMap incompatible types

- Read a nullable field and assign it to a not nullable field.
  - This can cause NullPointerException because of the unboxing procedure of Java for Object types to convert them into primitive data types.
  - Check the code mentioned in the stack trace – find the field name and check the nullable state in the schemas
- Different data types
  - Can only be recognised in the code view because of compile errors
  - Solution 1: tConvertType.
    - be aware: tConvertType excepts case sensitive the same schema column names input as well as output!
  - Solution 2: Use routines to convert the values
- Lookups memory over load
  - Only load the columns needed inside a lookup
  - Select rows in the lookup reasonable

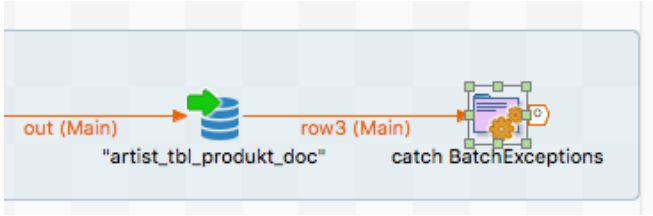
Typical pitfalls – Database components

- Default setting “Die on error” is off
  - This let the job survive regardless what error happens and the result can be unpredictable
  - ALWAYS turn this option on!
- Batch mode in output components
  - Great enhancement for the performance
  - Problematic error handling because of the missing exception which reflects the real problem.
- Use Cursor in input components
  - Can greatly enhance the performance and avoid memory leaks
  - Some database drivers collect the whole result if no cursor size is set. E.g. MySQL driver
- Setup the schema correctly – even if the settings actually not necessary
  - This prevents the warning from covering really important problems

Typical pitfalls – Database components

» Solution for the batch mode error handling.

The `BatchUpdateException` shows only the fact something went wrong, the actual problem can be retrieved with calling the method `getNextException()`. Add a tJavaFlex component in the flow right after the database output component (in the forwarded flow) and configure the code according.



Begin:

```
try {
```

Main: keep it empty

End:

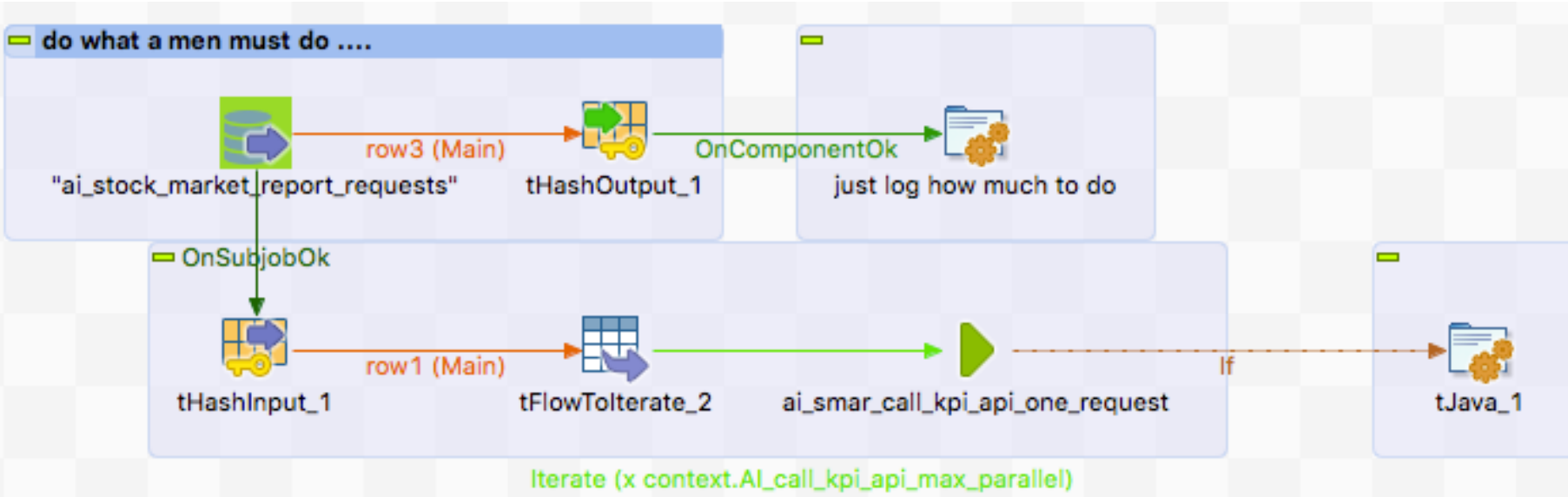
```
} catch (java.sql.SQLException sqle) {
    java.sql.SQLException sqleNext = sqle.getNextException();
    if (sqleNext != null) {
        throw sqleNext;
    } else {
        throw sqle;
    }
}
```

Typical pitfalls – Job design

- Wrong jump of a component and use return variables
  - Always try to use OnSubjobOk if you need the return values of a component or you want to be sure the work is done!
- tFlowToIterate
  - Great tool to trigger subjobs
  - The variable keys are: <row-name>.<column-name>
    - be aware if you rename a flow, the already used variables will not always renamed according to the new flow name!
- Deactivating of referenced components (like tOracleConnection) let the referencing components choose another active one.
  - if you activating the tOracleConnection again, all other components stick with their formally new choice!
- Do not build huge main jobs by chaining a lot of jobs in a kind of main job. All the jobs uses the same JVM and the same resources.
  - Better consider using Execution plans in the TAC

Typical pitfalls – Job design

- tMap Lookups:
  - decide if you need to load it once or per input main row record
  - Only load the columns you need for the join – this helps avoiding memory overruns
- Load multiple used data in a local storage tHashOutput
- Do not keep open a cursor for a long time.
- Better read the data in a tHashOutput and iterate through the data out of tHashInput – see the picture below.



Typical pitfalls – Miscellaneous

- Errors in the if trigger will be marked at the component where the if trigger starts
- Errors in components in sub jobs sometimes will be shown as error in the main job if there is a component with the same unique id.
- Problems in tPre-part of a job do not prevent the main part of the job from running!
  - If you need the pre part and depend on its success consider using a OnComponentOk trigger to start the main part
- After a failure in the main part, the error state can become reset in the tPost-part.
  - Always use tDie as reaction on all kind of errors. tDie code survives!
- Spelling mistakes
  - correct them immediately!
  - Otherwise you run into trouble later because you forget the root clause!
- if condition errors shown as error from the start point component

Typical pitfalls – Miscellaneous

- Take care about static resources!
  - Reset sequences if you use them in a job and this job will called multiple times
  - Take care about the content of the globalMap if you run a subjob on particular conditions and depends on values in the globalMap written in your subjob

Job Design Best Practices

1. Parameters to steer the job
2. Naming Conventions
3. Transactions
4. Error Handling
5. Understanding the job code structure
6. Typical pitfalls
7. Release Management
8. Deployment / External Resources
9. Documentation

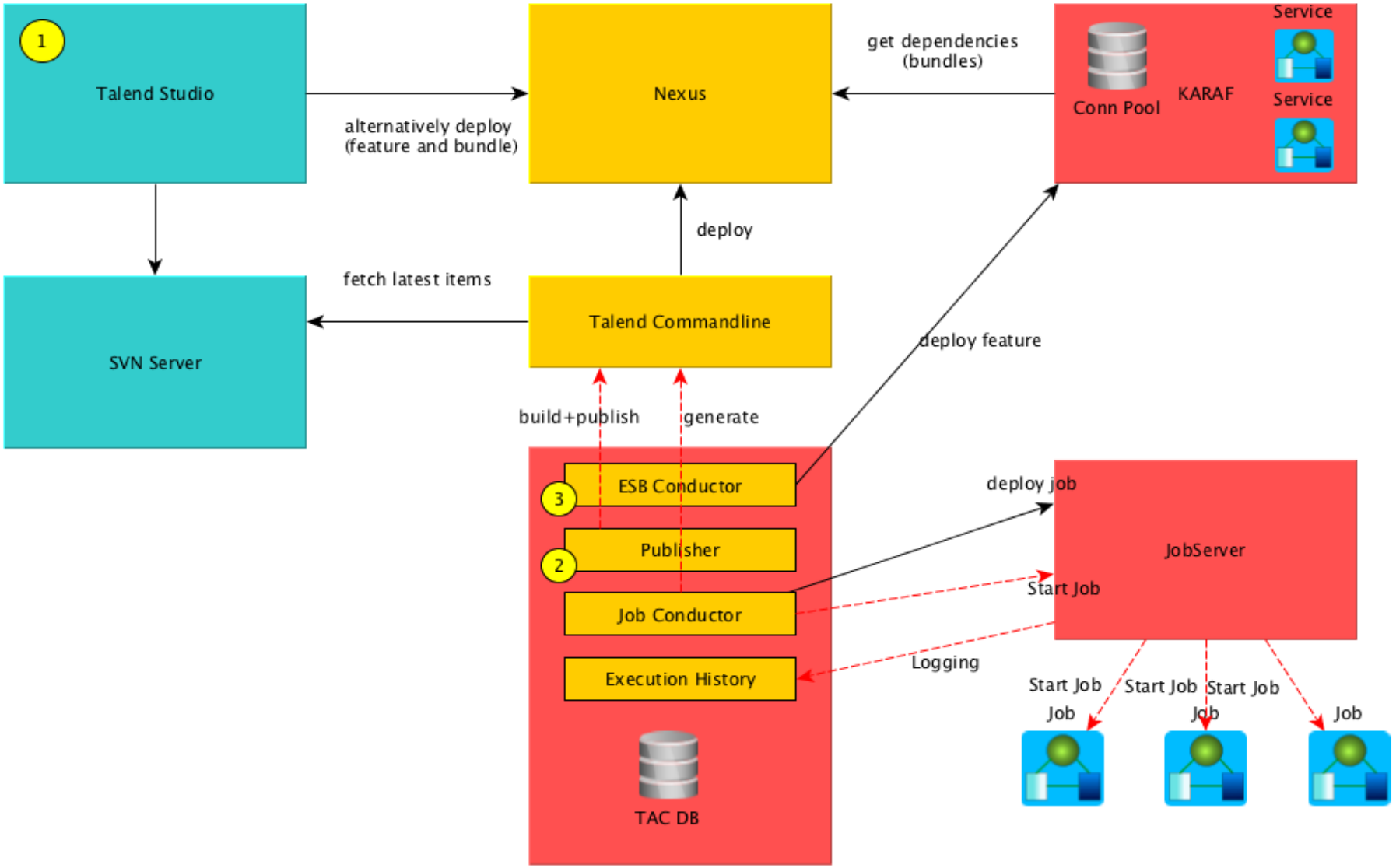
Release management

- Use the Talend version numbering.
  - In jobs you can survive without using it
  - BUT in services you MUST use it because the Nexus does not override an existing artefact with the same version!
  - Consider the major number for functional changes according to changed requirements or incompatibility
  - Consider using the minor number for bug fixes and compatible small improvements
- Consider using the SVN commit message
  - In the TAC you can setup a project to expect a commit comment if you save and unlock (that means commit) a job/route
- Consider to restrict the number of developers using the productive system.
  - Actually the productive system should only be used from quality ensuring developers, testers or administrators

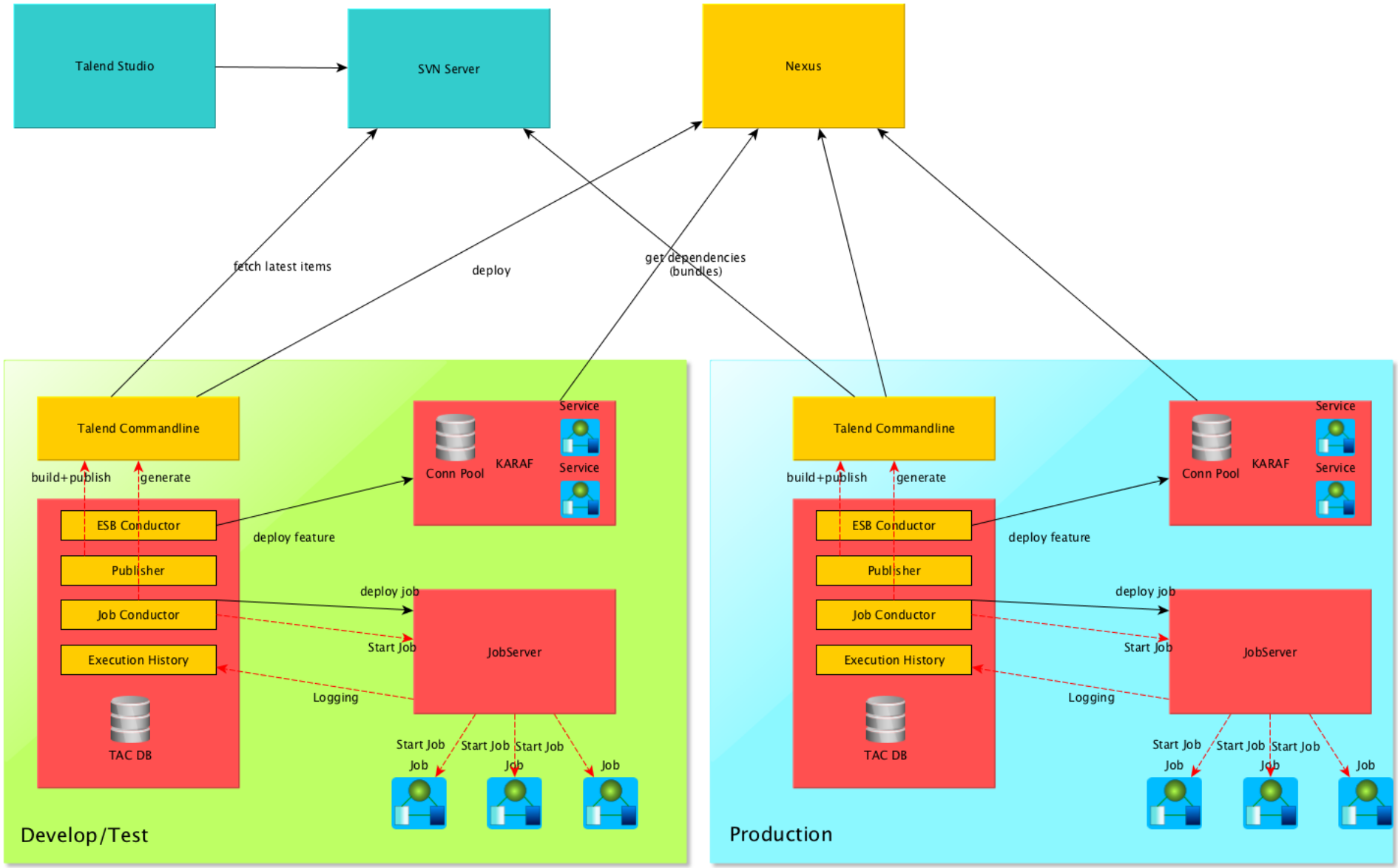
Job Design Best Practices

1	Parameters to steer the job
2	Naming Conventions
3	Transactions
4	Error Handling
5	Understanding the job code structure
6	Typical pitfalls
7	Release Management
8	Deployment / External Resources
9	Documentation

Deployment – general architecture



Deployment – Test + Development / Production



Deployment - Process

- Develop the jobs within the trunc branch
- For tests move the job into the test branch and deploy it to the snapshot repository in the nexus container
  - Be aware the nexus (and maven) are configured as not allowing overwrite existing artefact versions!
- For production move the job into the production branch and deploy it to the release repository in the nexus container

- Hint: you can get the original items of a job from the deployed job itself. They are part of the job archive under the folder items! Last chance sometimes if everything went wrong before.

- Use the implicit context load to provide the credentials to the jobs!
  - Never save productive credentials in SVN/Git!

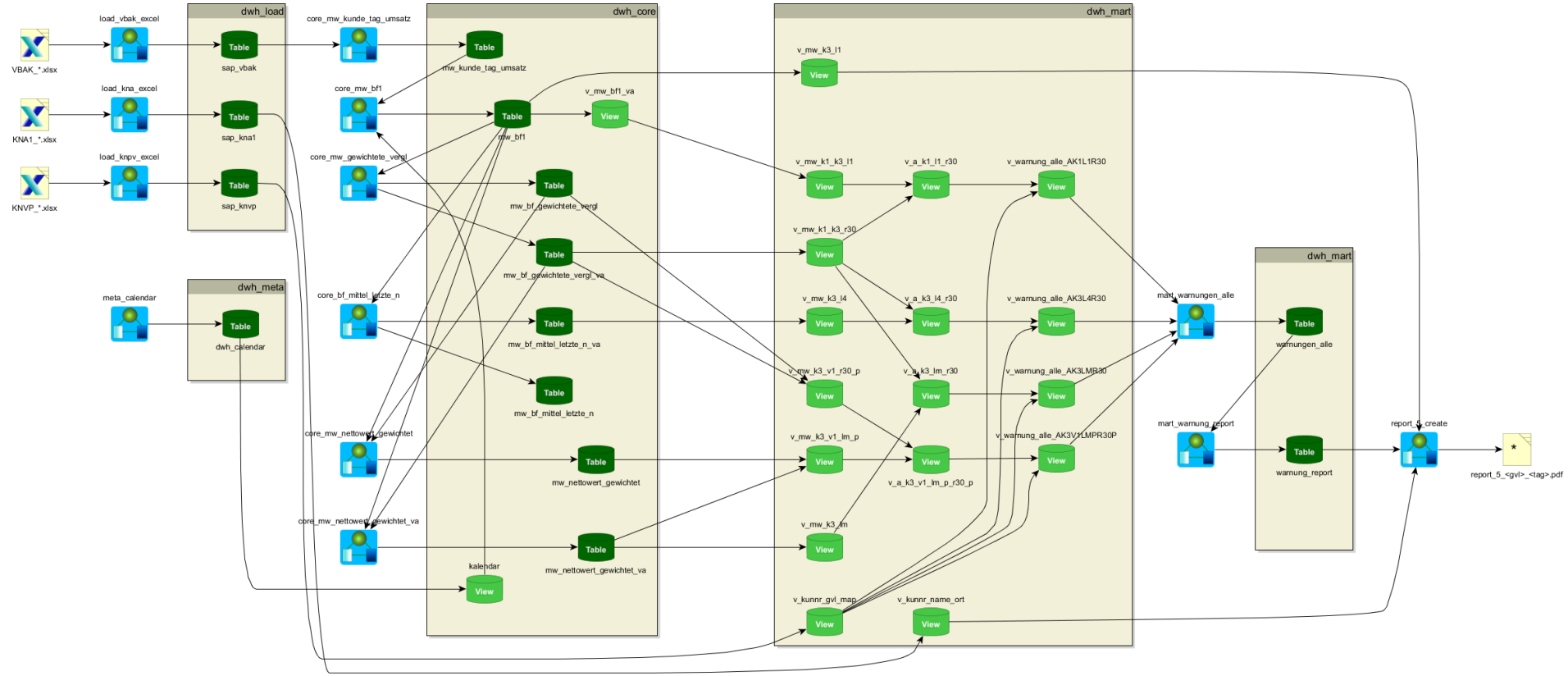
Job Design Best Practices

1	Parameters to steer the job
2	Naming Conventions
3	Transactions
4	Error Handling
5	Understanding the job code structure
6	Typical pitfalls
7	Release Management
8	Deployment / External Resources
9	Documentation

Documentation – Information needed

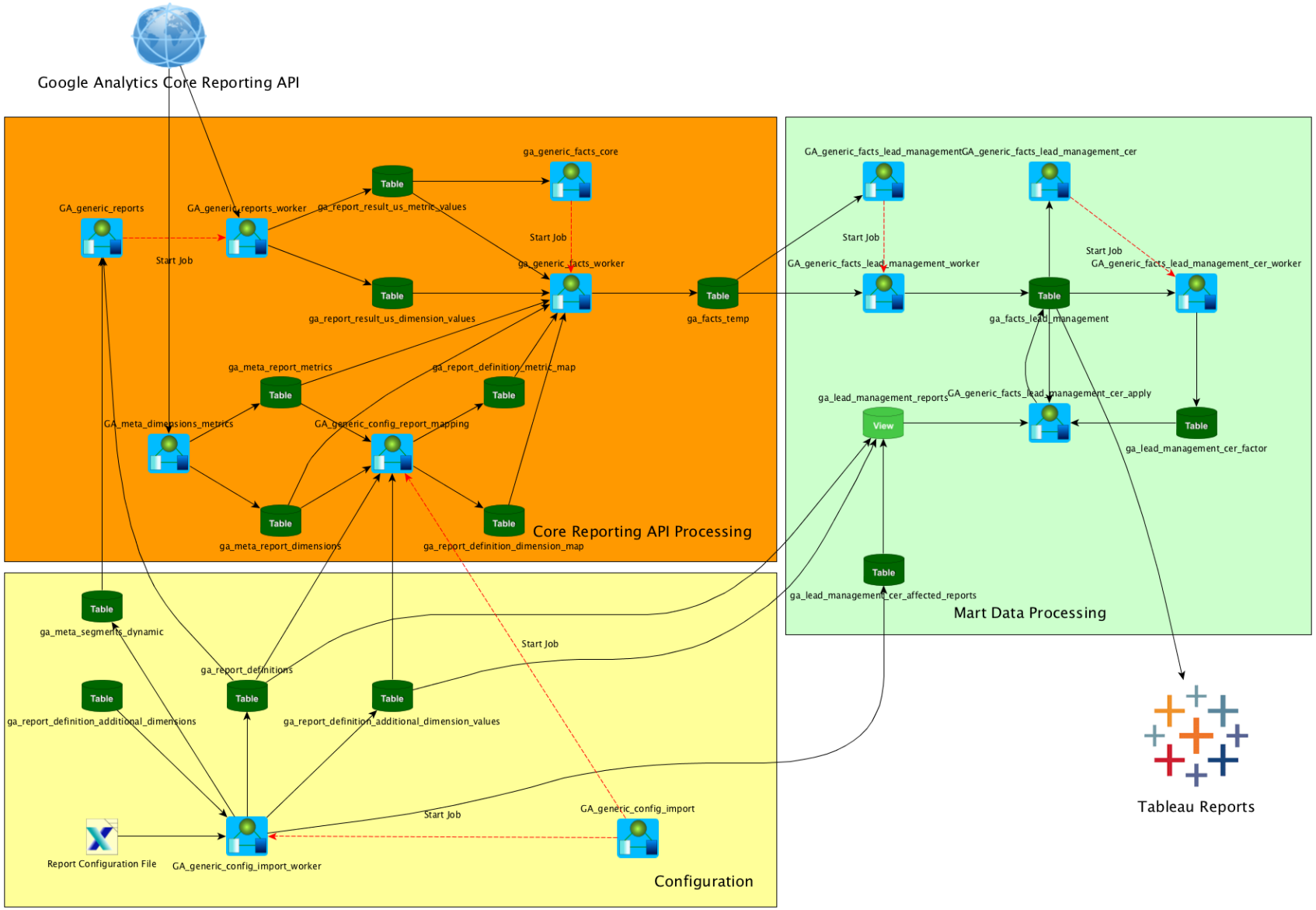
- Essential documentations are:
  - JIRA for the development progress
  - Operative documentation
    - is the job restart able
    - necessary preconditions
    - who is responsible for the development
    - what is the necessary reaction in case something went wrong
  - development documentation
    - which sources will be read
    - which targets will be written
    - who is the caller
    - which jobs will be called by this job
    - necessary input values
    - provided output values

Documentation – Data flow (Example 1) with yEd editor

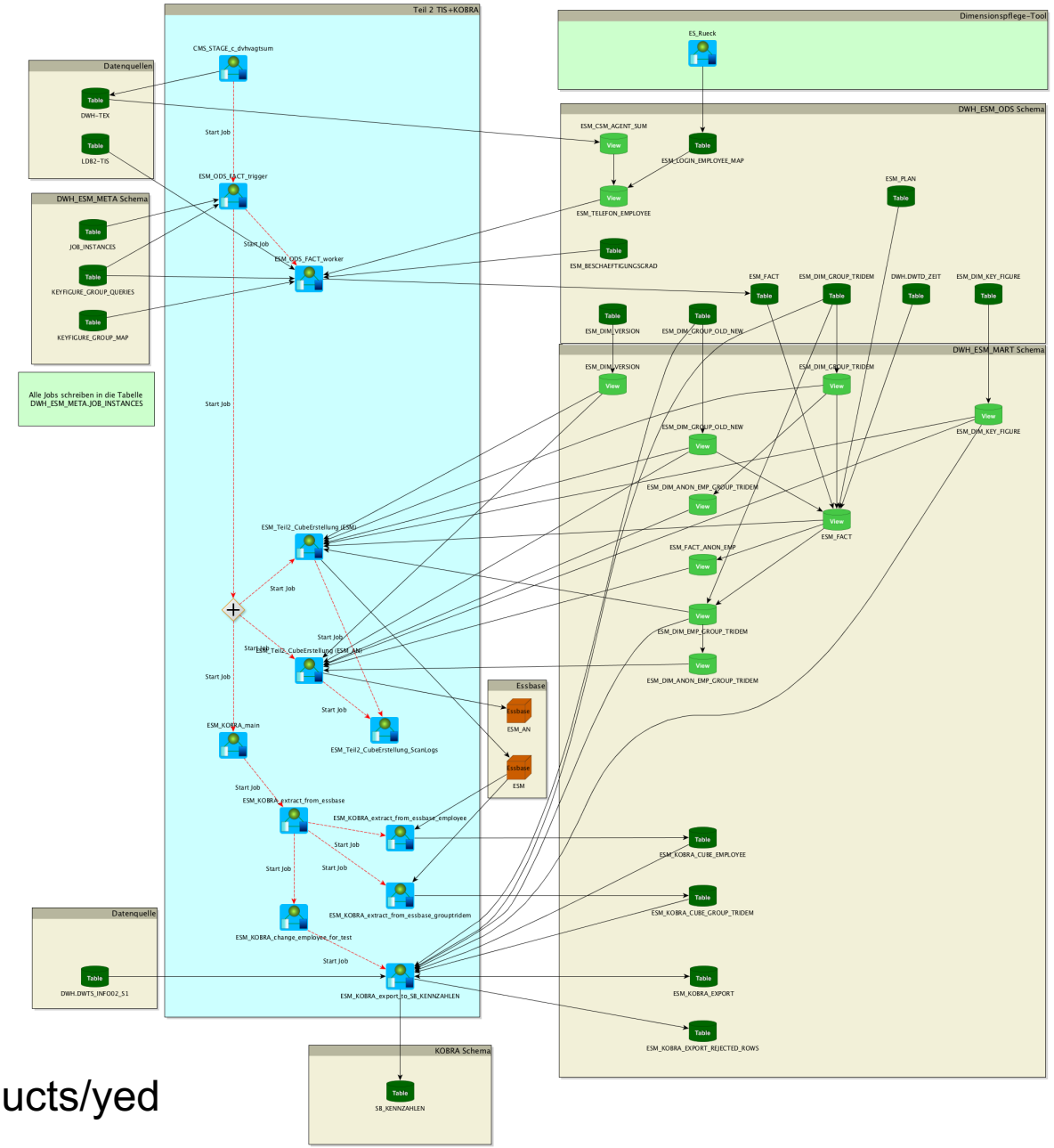


- Input files, jobs, tables and views (within the schemas)
- Data flow as black arrow
- Caller flow as red dotted arrow

Documentation – Data flow (Example 2) with yEd editor

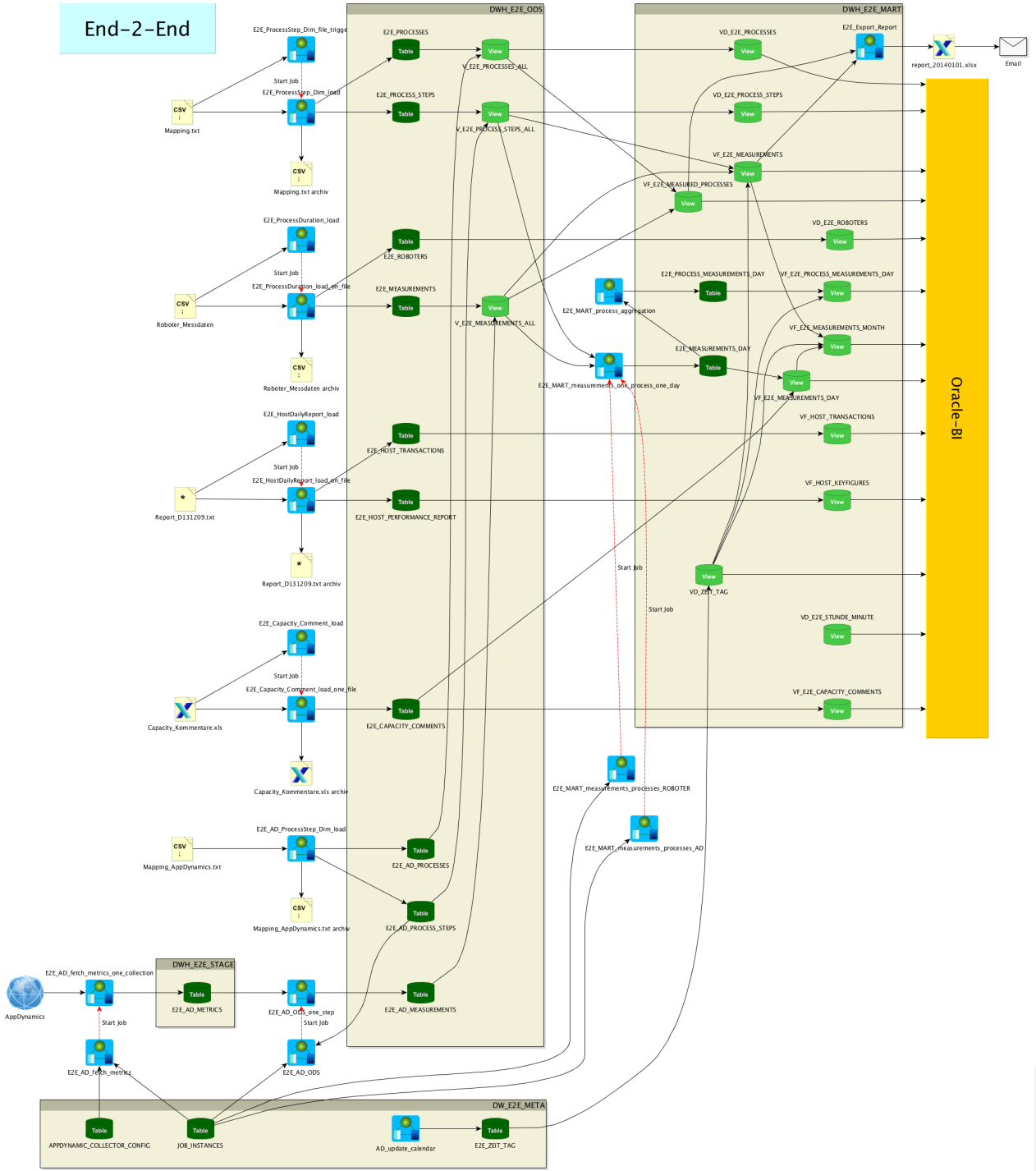


Documentation – Data flow (Example 3) with yEd editor



https://www.yworks.com/products/yed

Documentation – Data flow (Example 4) with yEd editor



cimt

Tools to build documentation

- Confluence
  - A good graphic editor is gliffy embedded in Confluence and also available as standalone editor in Chrome browser
- yEd - a free graph editor
  - Is freely available under https://www.yworks.com/products/yed
  - Use a graph xml format – can be checked semantically because the network is not only a graphic!
  - We provide some predefined graphical nodes for Talend for free
- Talend Business Model
  - has some links to the actual jobs
  - is a proprietary format