Workflow Analysis

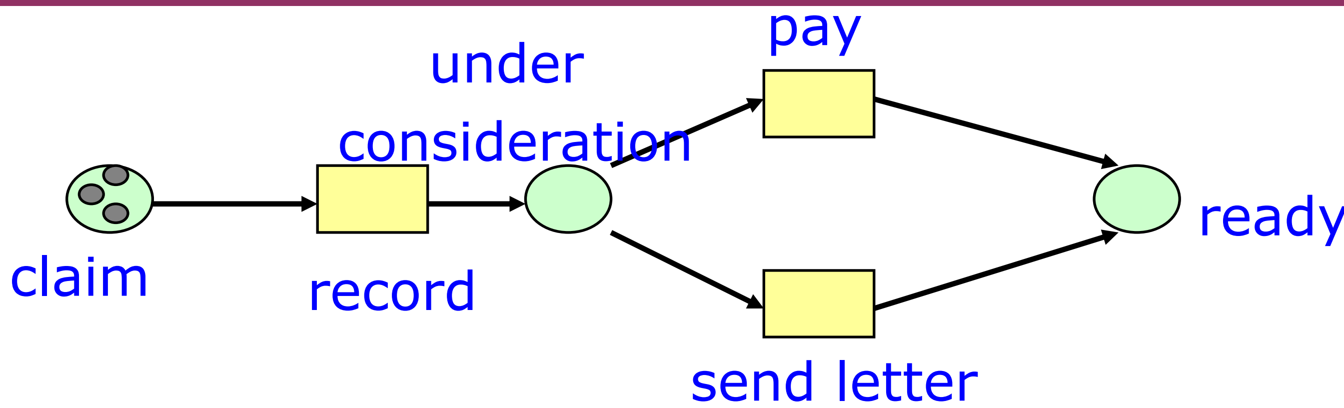
WORKFLOW ANALYSIS

- Workflow specifications may be analyzed with respect to their qualitative or quantitative aspects
- Qualitative aspects mainly concern the logical correctness of the workflow specification (i.e., absence of anomalies such as deadlocks or livelocks)
- Quantitative aspects concern performance (completion times, level of service, resource utilization)
- In order to analyze workflows, a framework is needed to express the behavior of the workflow

REACHABILITY ANALYSIS

- A workflow is described via a Petri net (PN)
- Transform workflow to reachability graph
- Reachability graph:
  - Direct graph comprising nodes & directed edges
  - Each node corresponds to workflow state
  - Edge denote state transitions
- Each state denoted by number of tokens in each place
- Reachability graph embodies the behaviour of a workflow
- Exploited to gain insight into the operation of a PN

FIRST EXAMPLE



$$
\begin{tikzcd}
(3,0,0) \arrow[r] & (2,1,0) \arrow[r] \arrow[d] & (1,2,0) \arrow[r] \arrow[d] & (0,3,0) \arrow[d] \\
& (2,0,1) \arrow[r] & (1,1,1) \arrow[r] \arrow[d] & (0,2,1) \arrow[d] \\
& & (1,0,2) \arrow[r] & (0,1,2) \arrow[d] \\
& & & (0,0,3)
\end{tikzcd}
$$

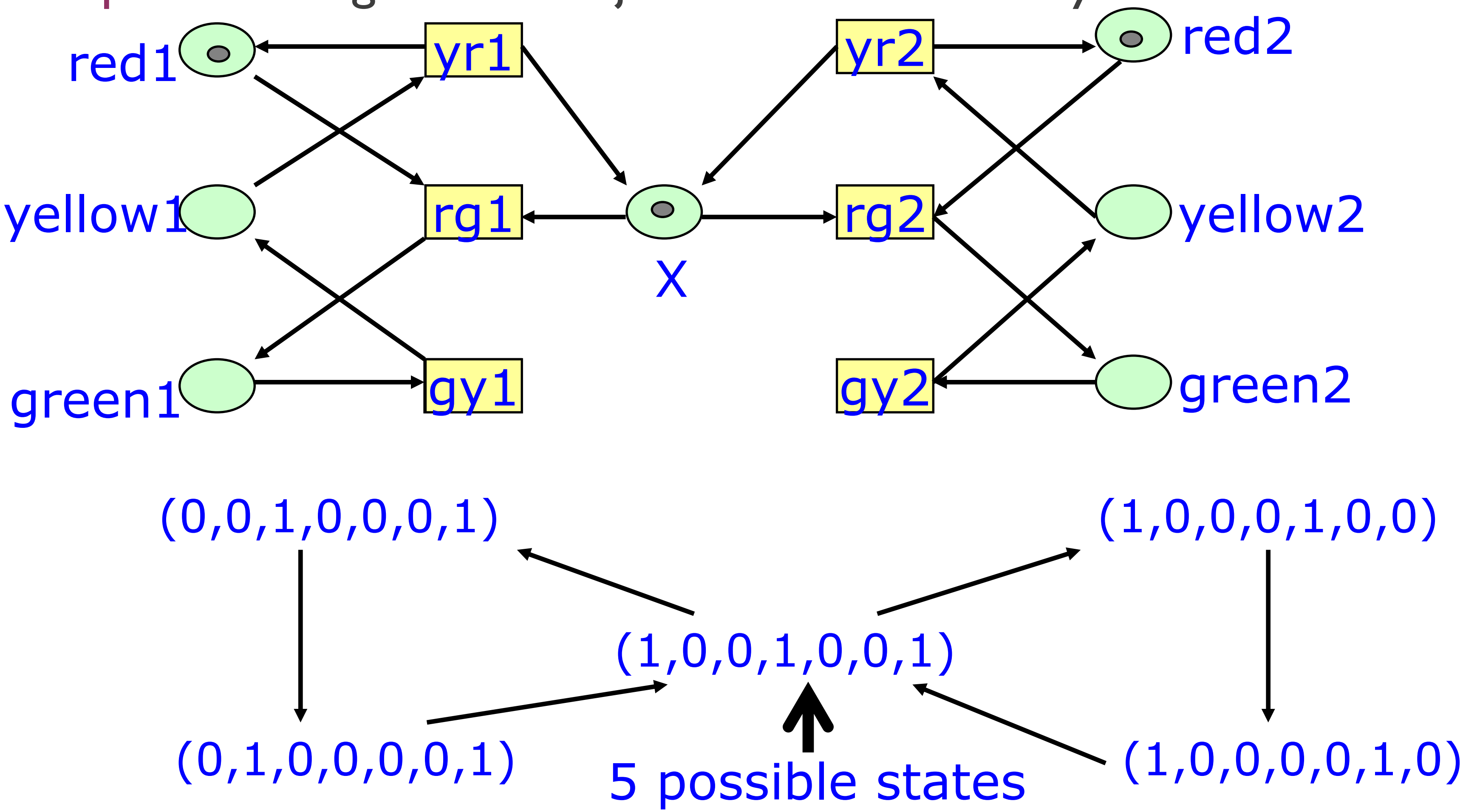
Ten possible states

WORKFLOW ANALYSIS

- The out-degree of each node in the reachability graph indicates the number of possible subsequent states
- If the out-degree is greater than 1, the next state is not predetermined (non-determinisitic choice)
- If a node has out-degree 0, then it is an end state (no transition is enabled)
- Given a Petri Net and an initial state, we can systematically construct its reachability graph

WORKFLOW ANALYSIS

- Example: traffic lights at the junction of two 1-way streets

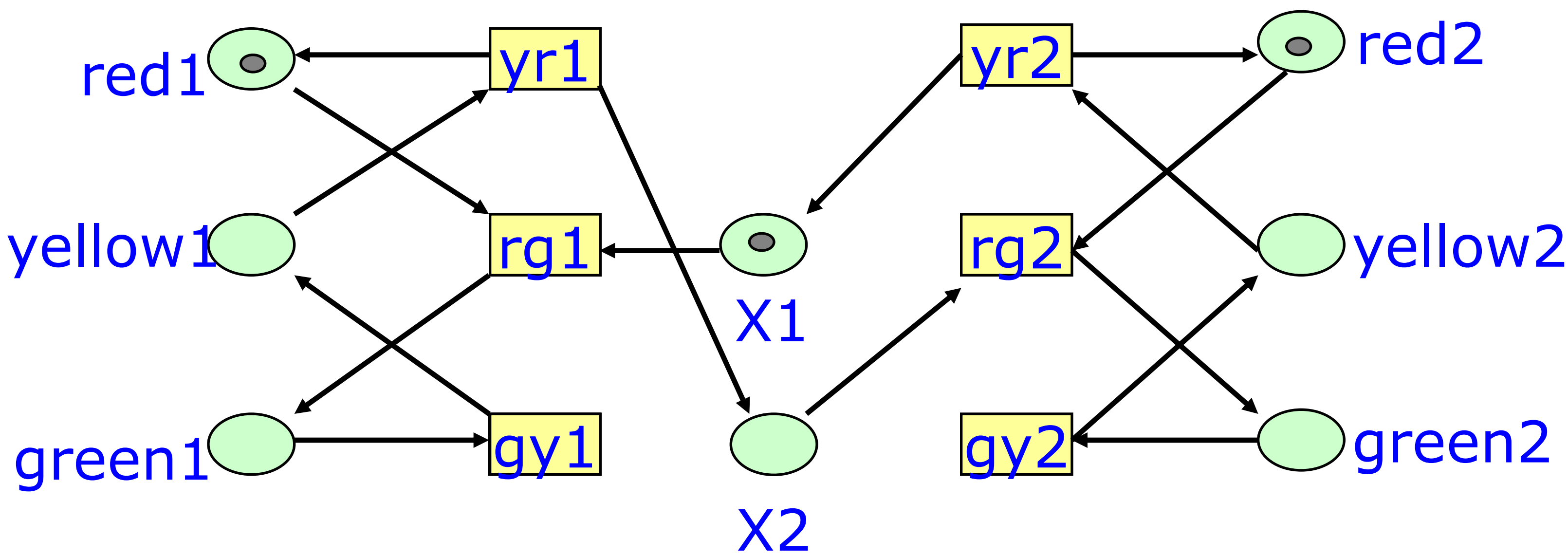


WORKFLOW ANALYSIS

- In the previous example, inspection of the reachability graph shows that the traffic lights operate safely: in every possible state at least one of the set of lights is red
- However, it also shows that it is possible that one set of lights always changes to green, while the other remains constantly red
- If we want to avoid this, we must change the Petri Net so as to ensure that each set changes to green in turn
- Need to construct the reachability graph of the new net and verify that it exhibits the expected behavior

WORKFLOW ANALYSIS

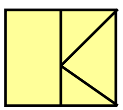
- Example (continued)



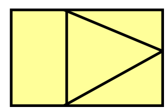
Reachability graph will contain 6 states

STRUCTURAL ANALYSIS

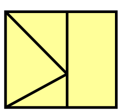
- Workflows can be structurally analyzed to discover potential problems in their execution
- The combination of sequential, parallel, selective and iterative routing often make the assessment of correctness hard.
- Notation:



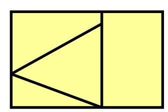
AND-split



OR-split



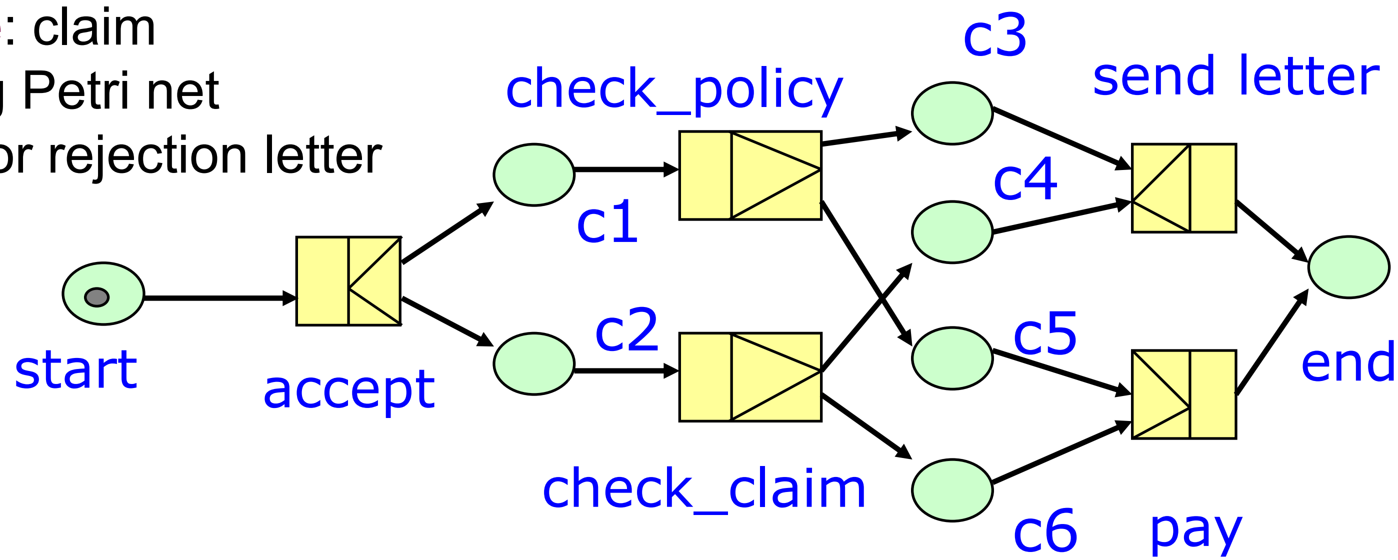
AND-join



OR-join

WORKFLOW ANALYSIS

- Example: claim processing Petri net (payment or rejection letter sent)



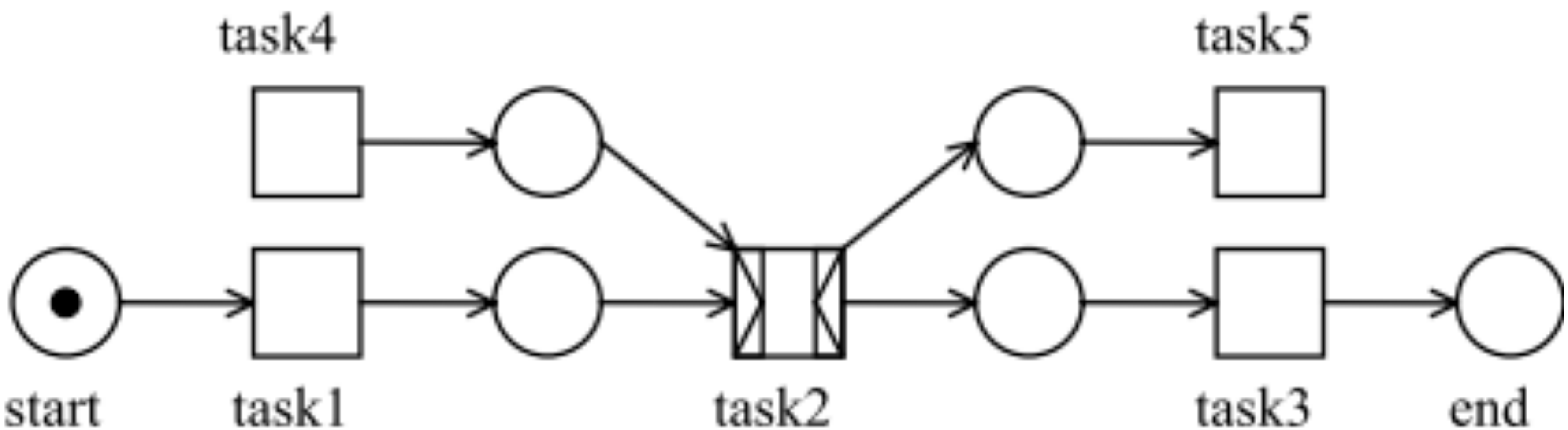
- if a token is placed in c5 by transition check_policy, and a token is placed in c6 by check_claim, pay will fire (correct!)
- if a token is placed in c3 by check_policy and a token is placed in c4 by check_claim, send_letter will fire twice
- if a token is placed in c3 by check_policy and a token is placed in c6 by check_claim, send_letter will fire once, but token remains in c6

WORKFLOW ANALYSIS

- Problematic Petri net structures:
  - tasks without input and/or output conditions: when a task has no output conditions, it does not contribute to the successful completion of the task and can be dropped
  - dead tasks: tasks that can never be carried out
  - deadlocks
  - livelocks
  - activities taking place after "end" is reached
  - tokens remaining in the process after a case has been completed
- Such cases can be identified without knowing the exact content of the process being defined

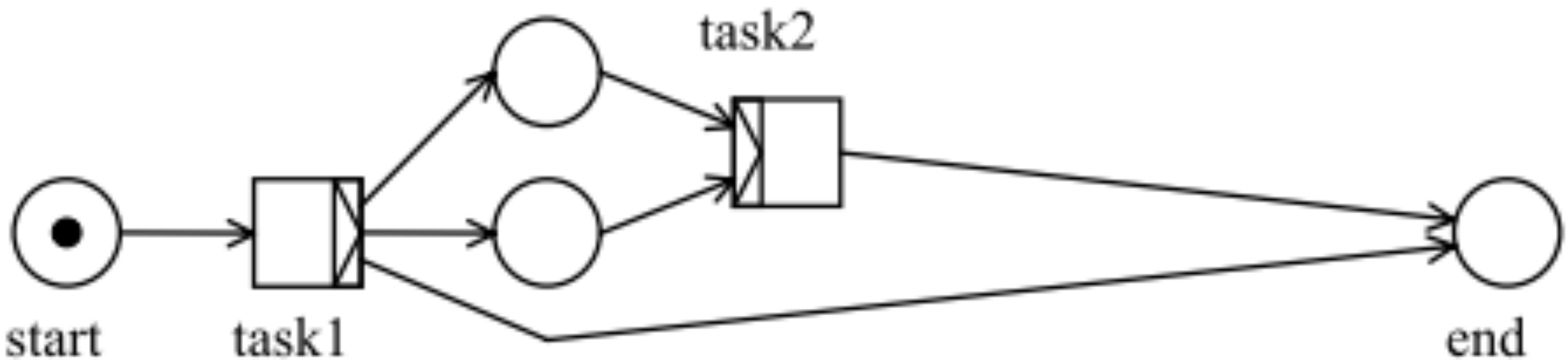
TASK WITHOUT I/O CONDITIONS

- No input conditions -> not known if task will be performed (task 4)
- No output conditions -> does not contribute to a successful completion of a case, can be dropped (task 5)



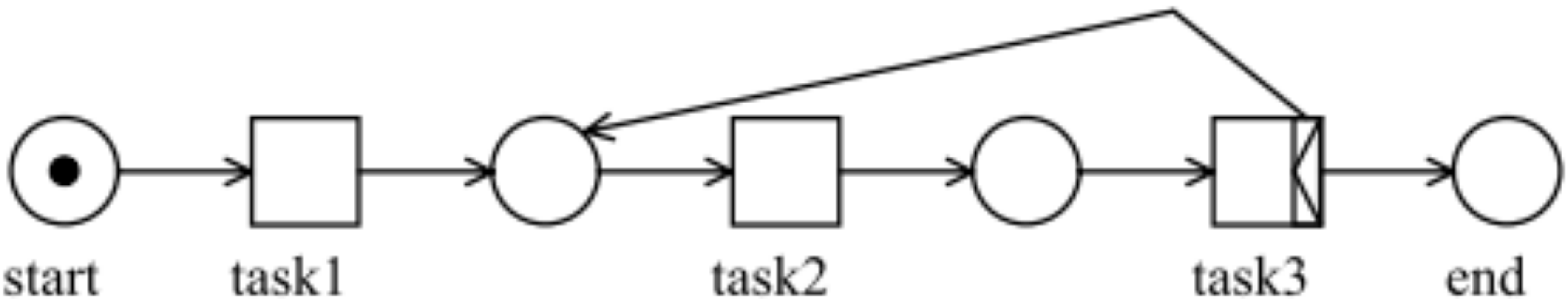
DEAD TASKS & DEADLOCK

- Dead Task: A PN might contain a task that will never be performed
  - Example below: Task 2 is a dead task
- A case is frozen before the end state
  - Example below: Task1 places a token in two upper places and then case will wait forever as task2 will never be executed



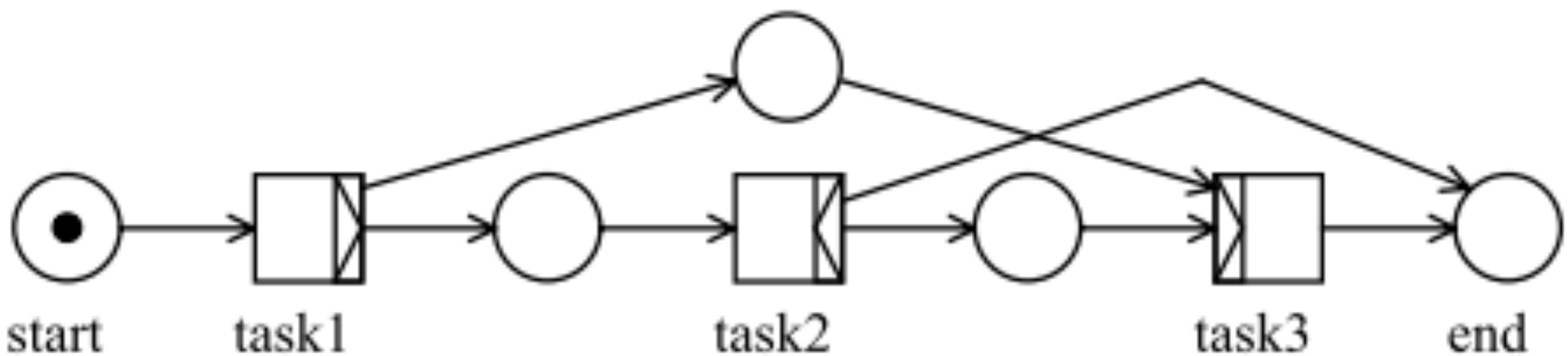
LIVELOCK

- A case is trapped in an endless cycle
- Example below:
  - Every case will pass in the non-ending cycle involving tasks 2 and 3.



TOKENS REMAIN AFTER PROCESS COMPLETION

- Once a token reaches the end state, all other references to the case must disappear
- Example below:
  - If token reaches end state via task 2, then a token will still remain in one of the places before task 3



WORKFLOW ANALYSIS

- A precise notion of workflow correctness must be specified to computerize the error checking
- Requirement: a process contains no unnecessary tasks and every case submitted must be completed in full with no tokens remaining in the process after its completion
- A process that fulfills this requirement is called sound
- A workflow process defined by a Petri Net has a single place start and a single output place end
- Each transition or place should lie on a directed path from start to end (there should be no loose tasks or conditions)
- Each task is reachable from start and end is always reachable
- A transition not on a path from start to end does not contribute to the successful completion of the process
- A Petri Net fulfilling this requirement is called a workflow net

WORKFLOW NETS

- A workflow net, based on previous requirements, can still suffer from deadlocks & livelocks. A more precise definition is needed
- Workflow Nets – Syntactical Requirements
  - A WF net is called sound if it fulfills the following:

1. For each token put in place start, one (and only one) token eventually appears in place end
2. When the token appears in place end, all other places are empty
3. For each transition, it is possible to move from the initial state to a state in which this transition is enabled

WORKFLOW NETS

- Requirement 1: every case should be successfully terminated in the course of time
- Requirement 2: When a case completes, no references still remain
- Requirements 1 & 2 -> only one state is reached, final one, with one token
- Requirement 3: exclusion of dead tasks
- This definition of soundness assumes a notion of fairness: if a task can potentially be executed, then it is not possible to postpone its execution indefinitely

WORKFLOW SOUNDNESS CHECKING

- Fairness means that although it is possible to repeat part of a process infinitely often, this iteration will not violate the soundness requirement
- Also, two tasks cannot cause a third task to "starve"
- To check whether a given process corresponds to a sound workflow net, we must first check if the Petri net for the process is a workflow net
  - this can be done by examining its structure

WORKFLOW SOUNDNESS CHECKING

- Checking soundness involves examining the reachability graph:
  - Start with the initial state and a token in it
  - Check last requirement by observing whether there is a path/state transition reaching each task
  - First two requirements are checked by confirming that reachability graph has one final state & exists one token only in the ending state
- 2 main drawbacks:
  - Constructing the reachability graph is expensive
  - Reachability graph does not help in repairing problematic processes

SOUNDNESS CHECKING – COMPUTER SUPPORT METHOD

- Determining soundness:
  - add a transition t* to the net with end as input and start as output
  - the net with the new transition is called the short-circuited net
  - with this addition, soundness of the net corresponds to the properties of liveness and boundedness of the short-circuited net
  - a Petri net is live if, for every transition t, it is possible to reach a state in which t is enabled from every state reachable from the initial one
  - a Petri net is bounded when there is an upper limit to the number of tokens in each place
    - Net for traffic lights is live and bounded

SOUNDNESS CHECKING – COMPUTER SUPPORT METHOD

- There exist efficient algorithms and tools for verifying liveness and boundedness for certain classes of PNs
- When a process is not sound, some diagnostics indicating why it is not sound, can be produced
- Other analytical techniques that don't require computer support also exist.

SOUNDNESS CHECKING – MANUAL METHOD

- The translation of soundness requirements to liveness and boundedness is not very intuitive and requires computer support.
- Alternative methods can be applied without need for computer support
- Additional requirement:
  - workflow nets must be safe, i.e., the number of tokens in each place is never larger than one
  - safety is boundedness with an upper bound of 1
- Safety can be determined by inspection of the workflow structure

SOUNDNESS CHECKING – MANUAL METHOD

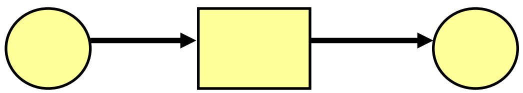
- The analysis method is based on the following property:
  - if we have two sound and safe workflow nets V and W and a task t in V which has exactly one input and one output place, then we can replace task t in V by W and the resulting net is still sound and safe
- Justification:
  - a sound workflow net behaves like a transition: consumes one token from its input place and produces one token at its output place
  - environment does not realize the replacement of t by W
- Safety required to avoid situation that in W two or more tokens will be active at the same time

SOUNDNESS CHECKING – MANUAL METHOD

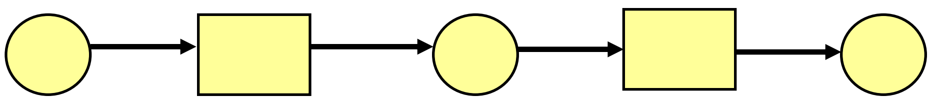
- Applying the property to workflow analysis:
  - some basic workflow nets can be easily shown to be sound and safe; these correspond to typical constructs
  - these nets can be used as building blocks for more complex workflow nets
  - if the workflow net under consideration can be shown to be derivable by a sequence of substitutions of nets from these building blocks, then it can be proved that the workflow net is sound and safe as well

BASIC & SOUND WF-NET CONSTRUCTS

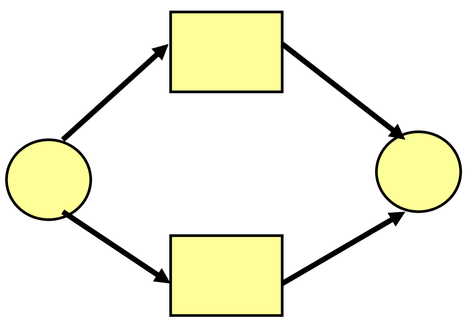
- Basic safe and sound constructs:



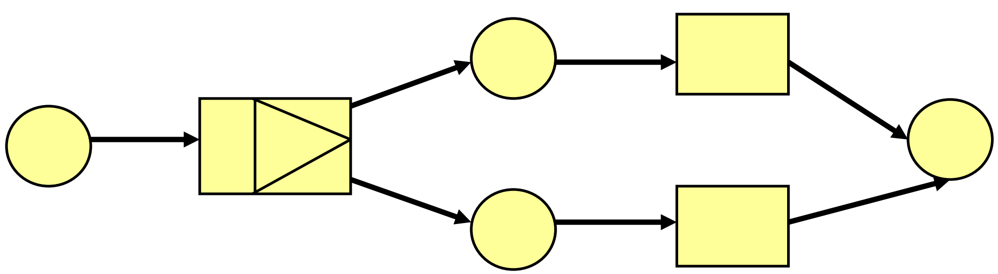
basic building block



sequence construct



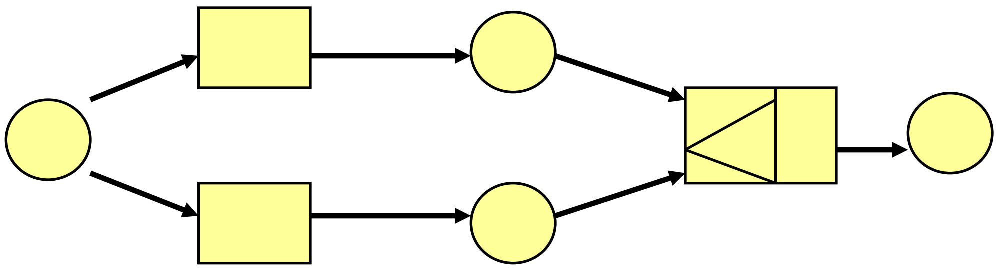
implicit OR-split construct



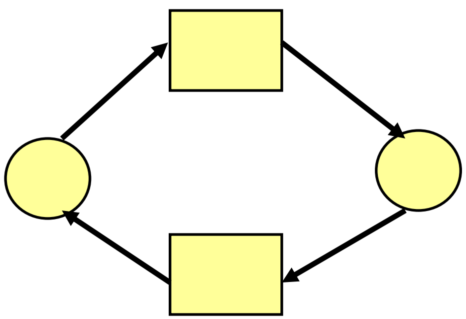
explicit OR-split construct

BASIC & SOUND WF-NET CONSTRUCTS

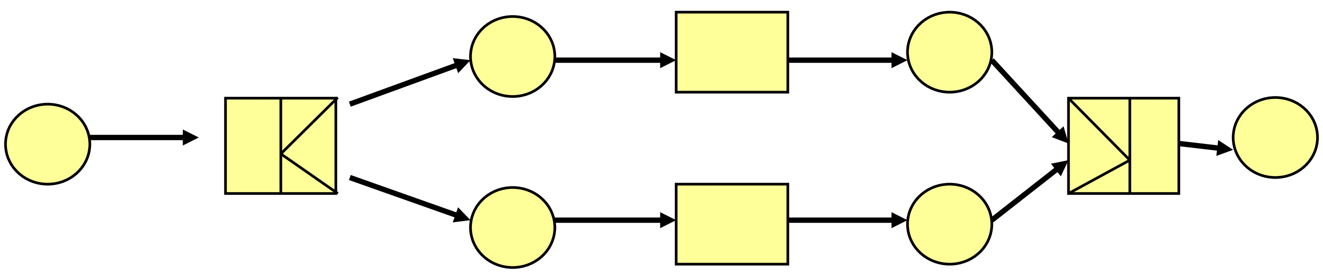
- Basic safe and sound constructs:



explicit OR-join construct



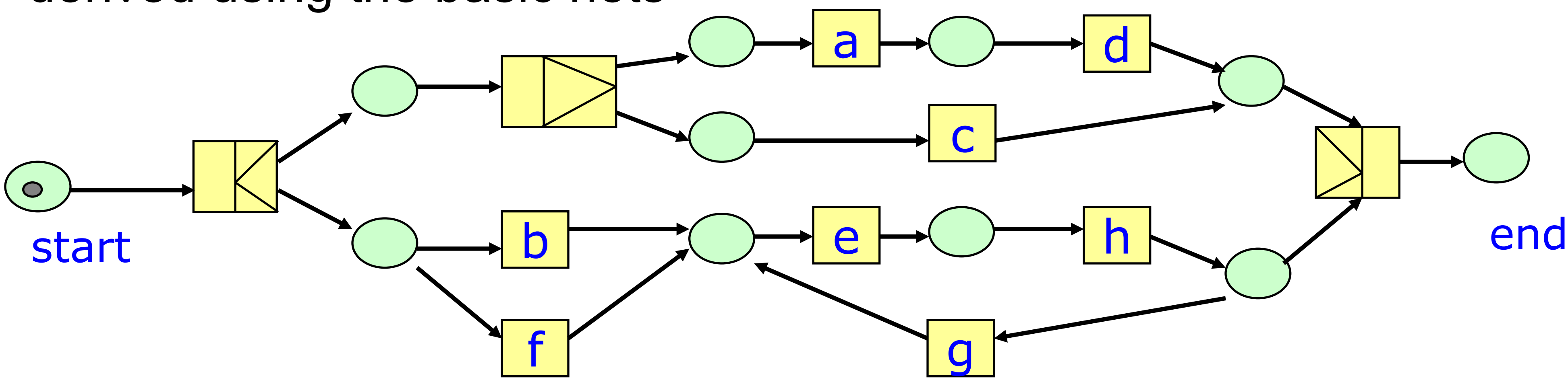
iteration construct



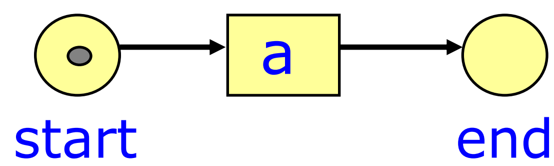
AND construct

WORKFLOW ANALYSIS

- Example: determine whether the following workflow net can be derived using the basic nets

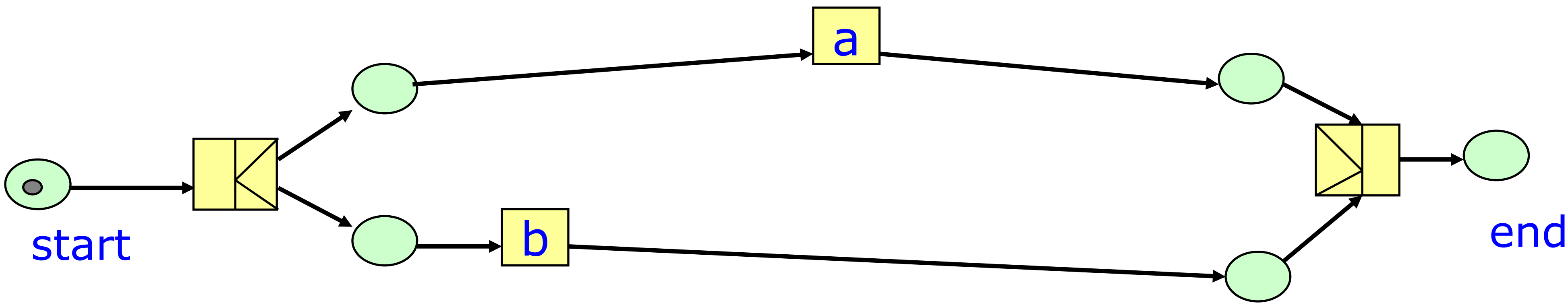


Start with the basic building block:

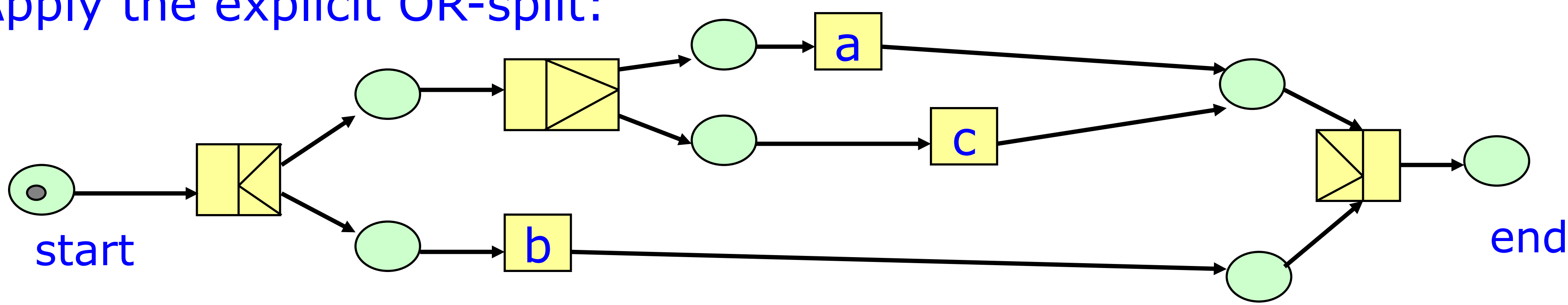


WORKFLOW ANALYSIS

Apply the AND-construct to put b in parallel with a

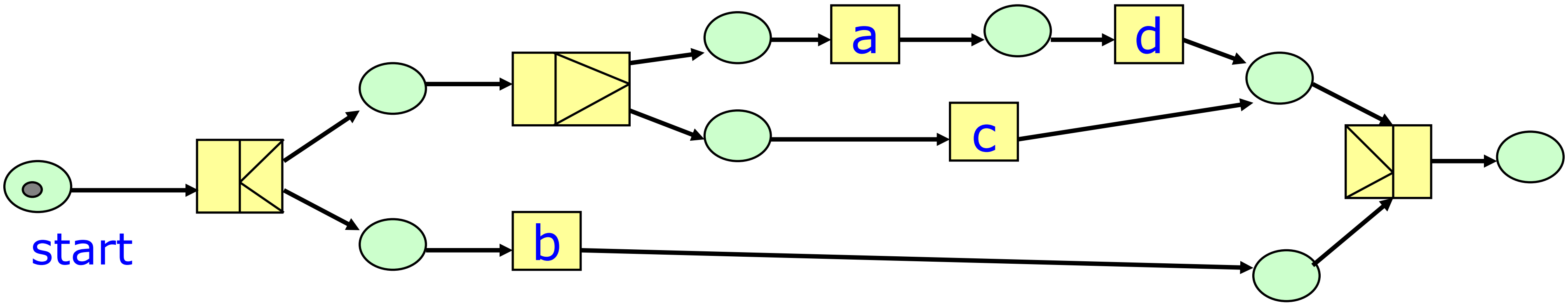


Apply the explicit OR-split:

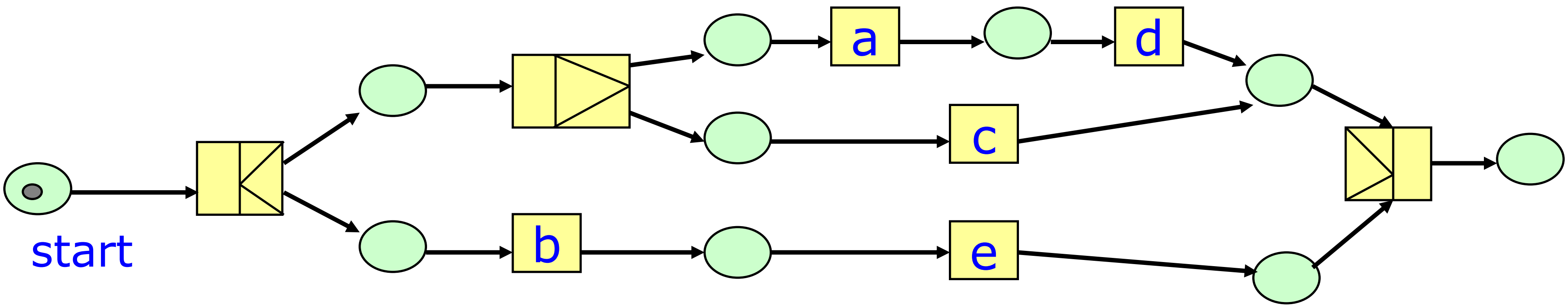


WORKFLOW ANALYSIS

Apply the sequence construct a followed by d :

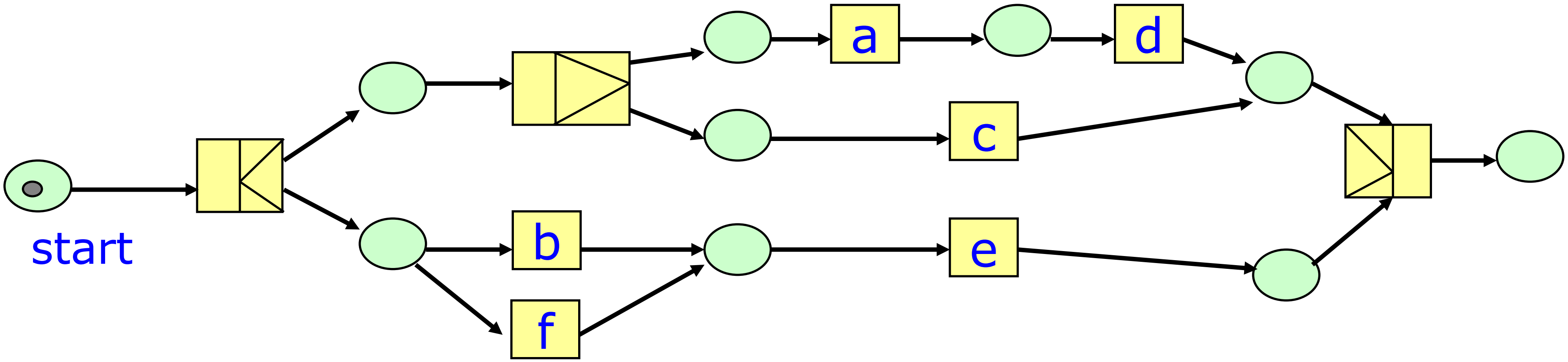


Apply the sequence construct b followed by e :

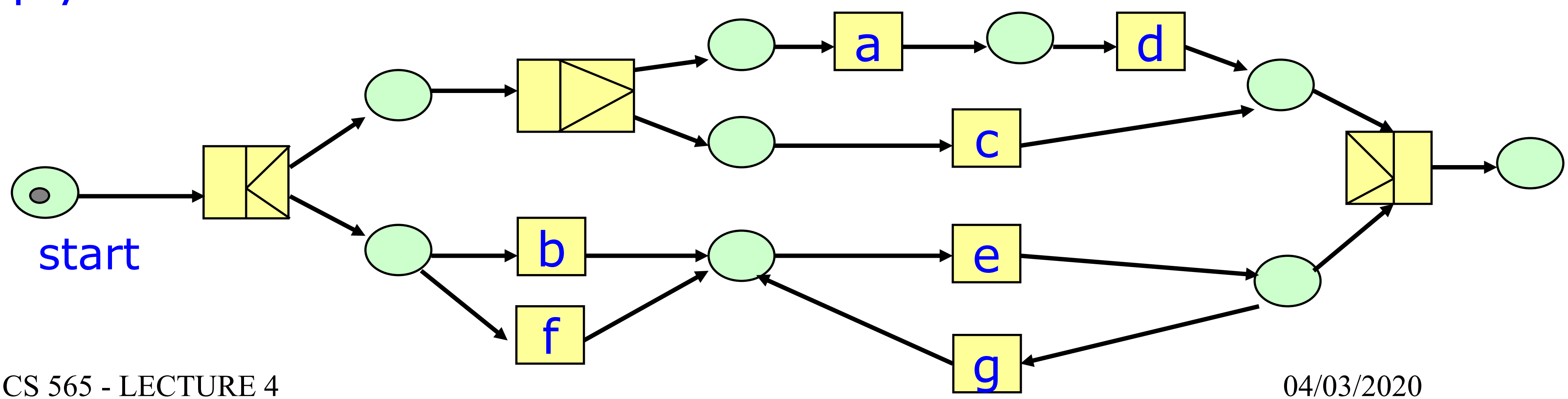


WORKFLOW ANALYSIS

Apply an implicit OR split to b for adding task f :

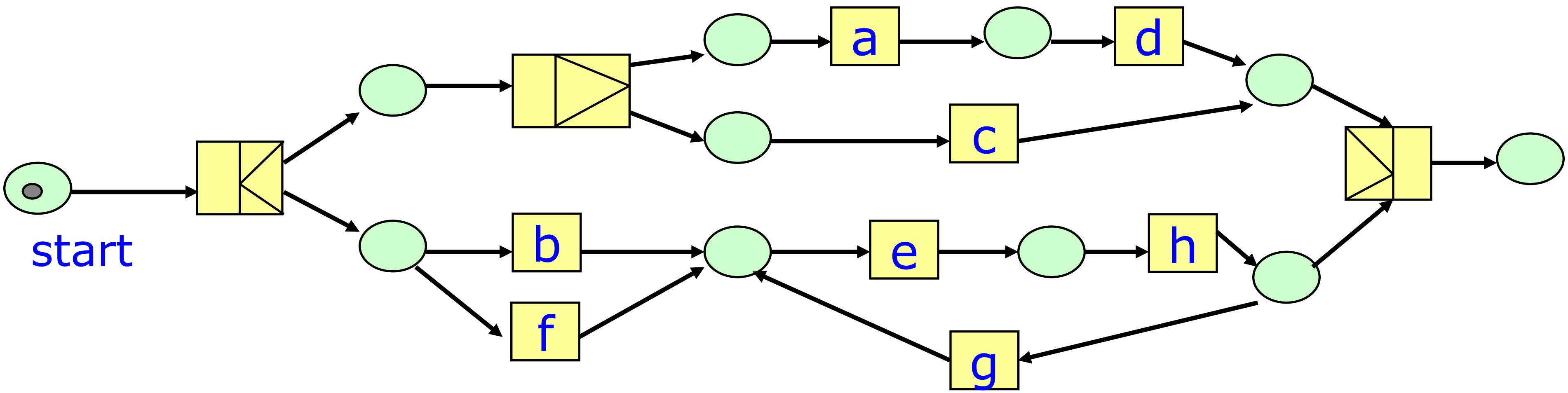


Apply the iteration construct to e :



WORKFLOW ANALYSIS

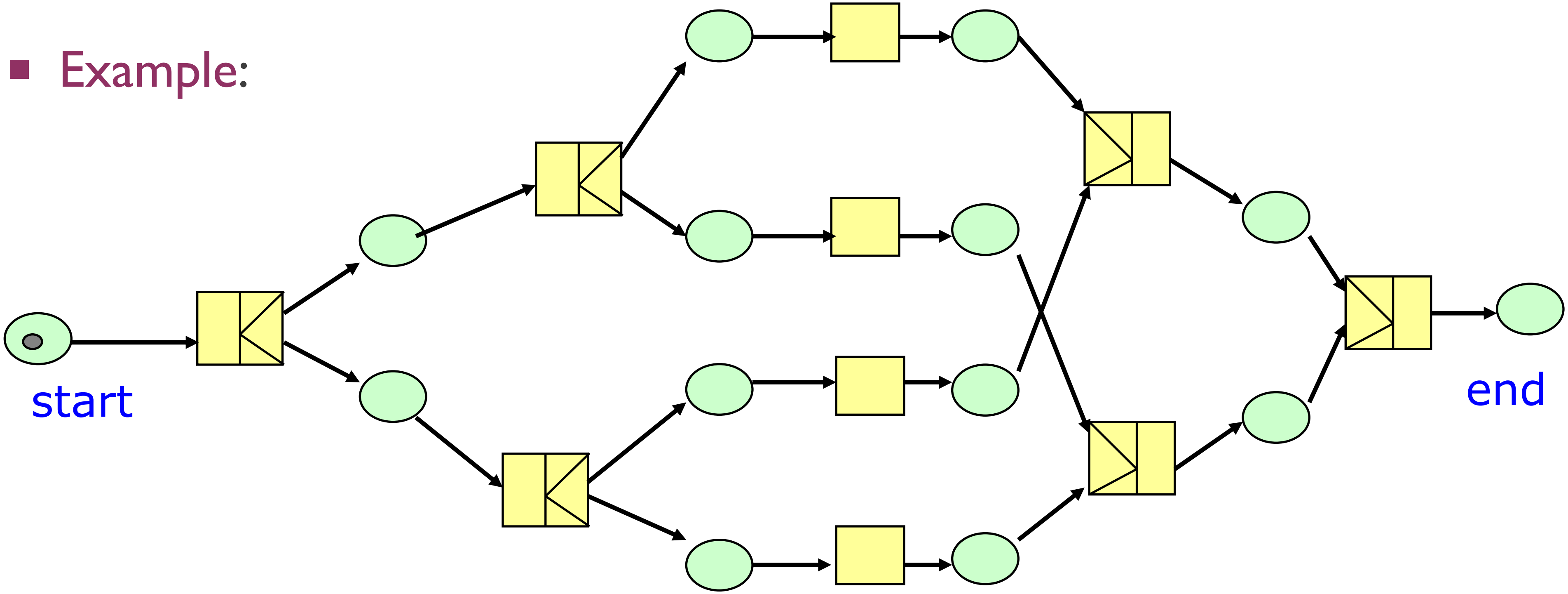
Apply the sequence construct to e :



- The workflow net results from applying the patterns of the basic building blocks, hence it is safe and sound.
- The derivation is not unique (3rd and 4th steps can be inter-changed)

WORKFLOW ANALYSIS

- Not all safe and sound nets have a derivation
- Example:



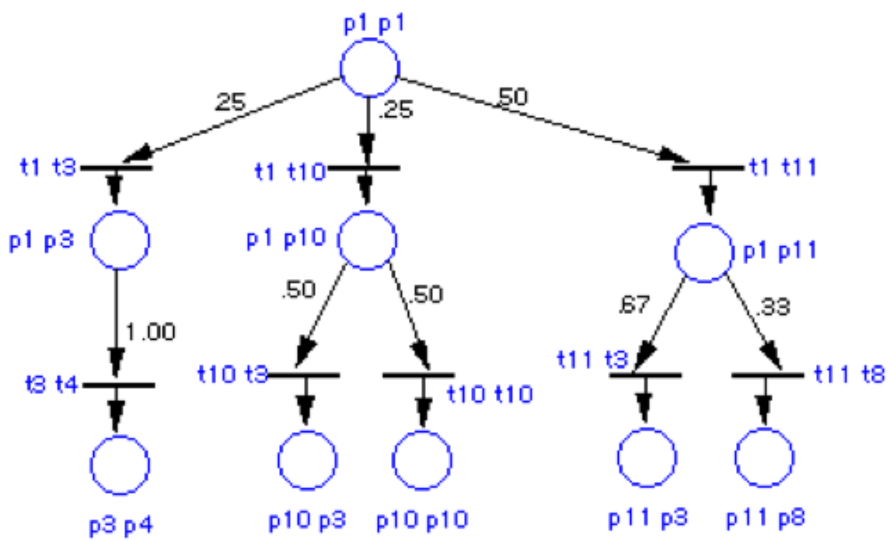
The two paths that originate at one AND-split should meet in the same AND-join.

PERFORMANCE ANALYSIS

- Need to examine quantitative aspects such as:
  - completion times of cases
  - number of cases that can be completed per time unit (throughput)
  - resource utilization
- The following techniques are mainly used:
- Markovian analysis: a Markov chain can be generated by a workflow;
  - a Markov chain contains the possible states of a case and the probability of transitions between them
  - a Markov chain is a reachability graph along with the probability information derived from measured or expected properties of a case type

MARKOVIAN ANALYSIS

- Various properties can be proven
  - Chances that a particular route for a case is chosen
- Can be extended with time and cost information
  - A range of performance indicators can be produced
- Disadvantages
  - Markov chain analysis is in general c
  - Not every aspect can be incorporated in the analysis



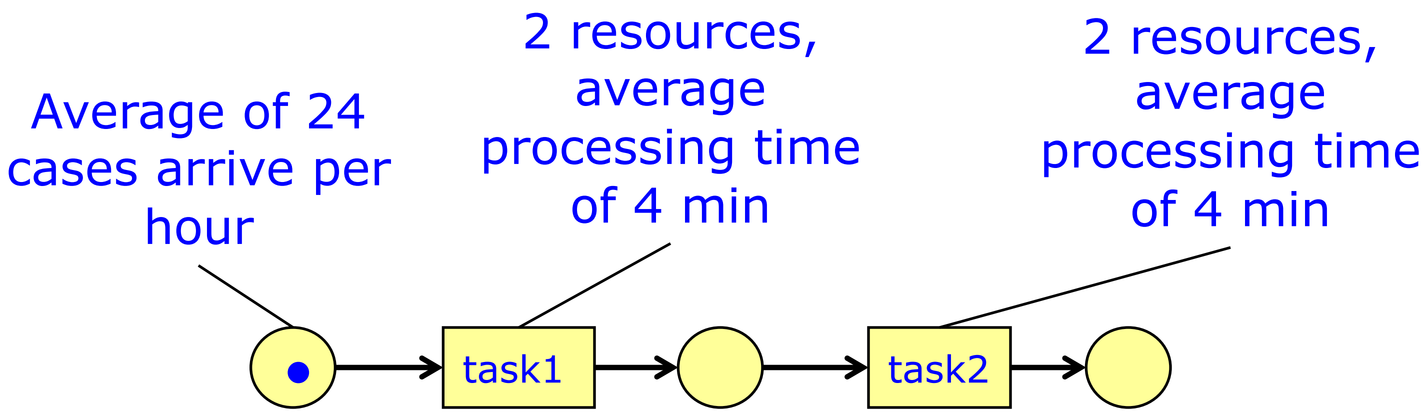
QUEUING THEORY

- Used for system analysis
- Places emphasis on waiting times, completion times, capacity utilization
- Need to consider a network of queues in order to extract performance measures for a workflow
- Some solutions come in turns of mathematical methods
- Disadvantage:
  - Many of the assumptions used in queueing theory are not valid for workflows (e.g., parallel routing of tasks not supported in analysis)

SIMULATION

- Flexible analysis technique
  - Always possible to analyze any workflow
- Amounts to following paths in a reachability graph
  - choices are made based on probability distributions
  - accessible to people with no mathematical background
  - offers better insight into the workflow operation
  - often workflows can be tracked graphically as well
  - easily extended with new aspects (e.g., faults)
- Disadvantages:
  - time-consuming process to establish the simulation
  - thorough statistical processing may be required for extracting conclusions from repeated executions

SIMULATION EXAMPLE



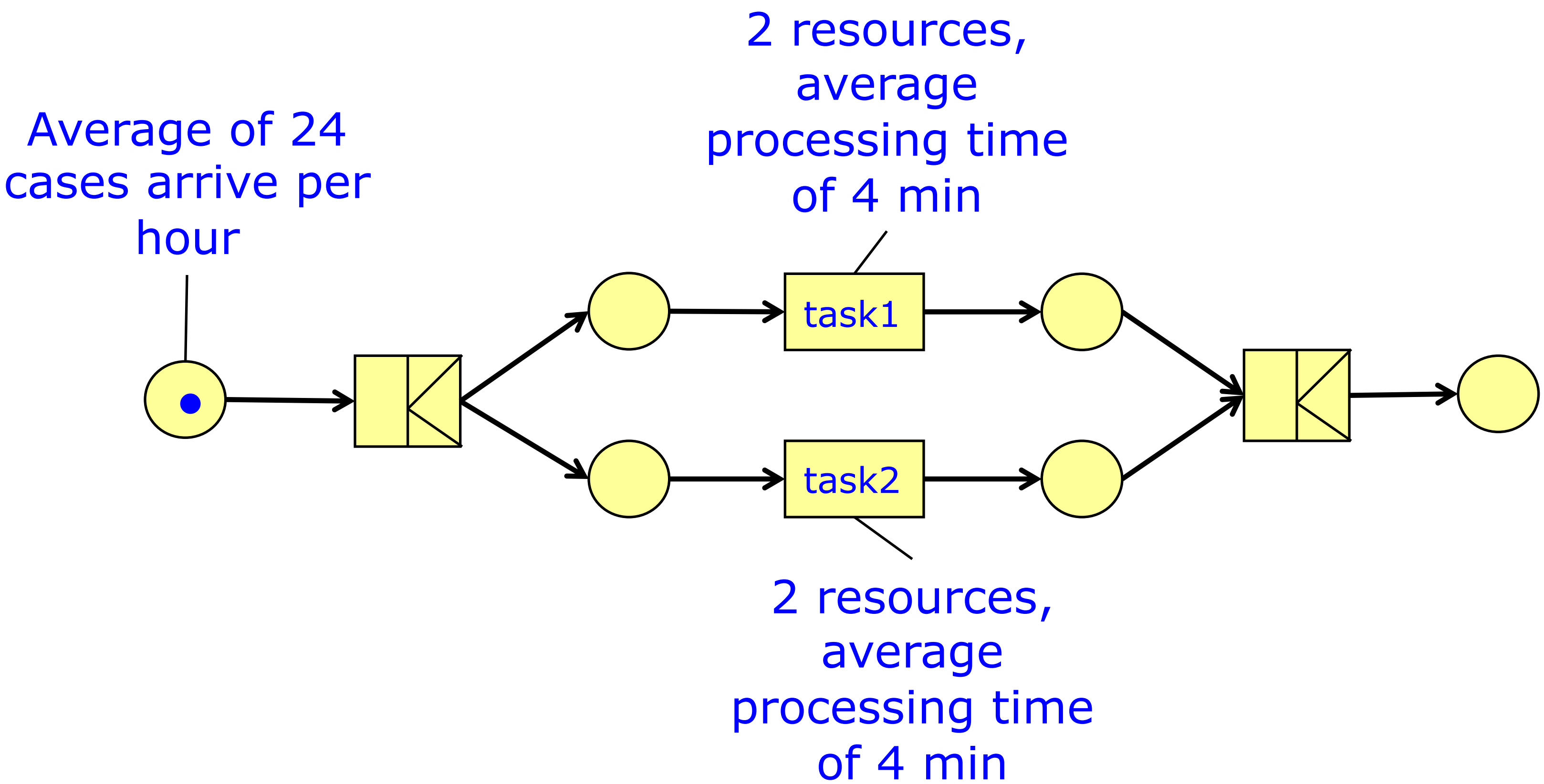
SIMULATION EXAMPLE

- Average time between consecutive arrivals: 2.5 min
- Average time to complete a task: 4 min
- Each resource works on one task
- Based on above, average resource utilization (# arrivals/time div # served/time 24/30) level is:
  - 80% -> for 20% of time, a resource is idle
- Average completion time per case can be computed:
  - Need to assume that interarrival times are distributed in a negative exponential way
  - Completion time is 22.2 mins
    - Actual serving time is just 8 mins, waiting time is 14.2
  - Need to reduce waiting time

SIMULATION EXAMPLE

- If each resource can work on any task, then:
  - Average completion time becomes 14 mins
  - Average waiting time becomes 6 mins

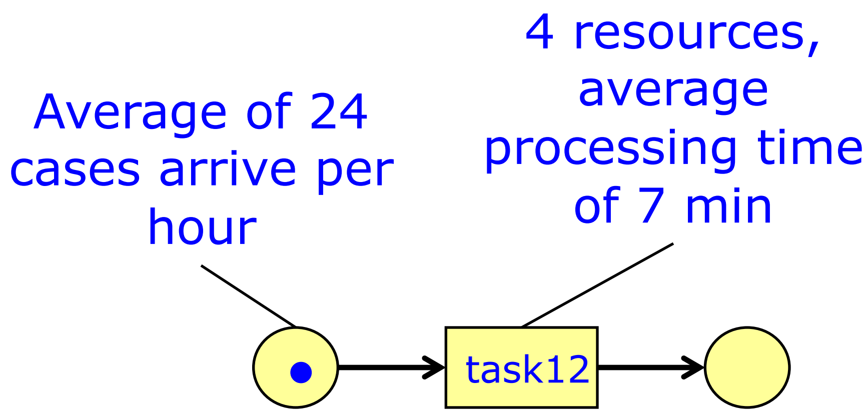
SIMULATION EXAMPLE



SIMULATION EXAMPLE

- Another solution: parallelize tasks
- Average completion time becomes 15 minutes
- Still exists space for further improvement

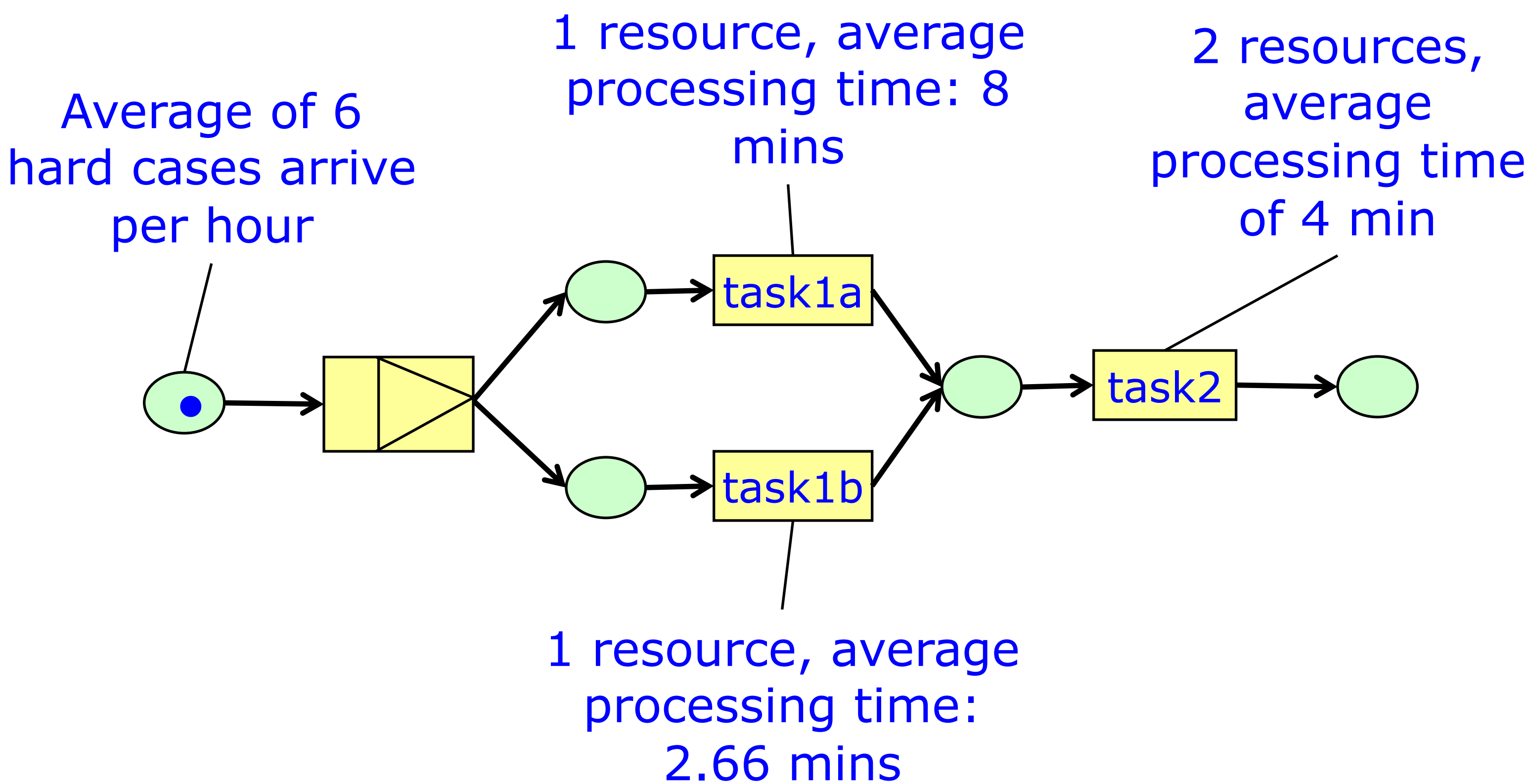
SIMULATION EXAMPLE



SIMULATION EXAMPLE

- Best possible solution:
  - Create composite task to be performed by each resource
- Increased resource flexibility
  - 1 minute less to complete the composite task
  - Resource capacity utilization falls into 70%
- Completion time drops to 9.5 mins
  - Waiting time drops to 2.5 mins

SIMULATION EXAMPLE



SIMULATION EXAMPLE

- More insight if we distinguish between cases
- 25% cases hard, 75% case easy
- Main idea: reduce completion time by separating flow (triage)
- Result: even worse than initial structure (31.1 mins)
  - Reduction of resource flexibility
- Triage can be useful when:
  - Allocation of specialized resources reduces average processing time
  - Small-scale client do not have to wait for large-scale ones for processing -> reduction of waiting time
  - In example, consider initial workflow structure & prioritization of easy cases over hard ones
    - Completion times goes around 14 mins

SIMULATION ANALYSIS – SUMMARY

- Simulation analysis can assist workflow design
  - Evaluation of alternative design choices
  - Each design choice can be best in different circumstances
- 3 design guidelines apply in most situations:
  - Perform tasks in parallel as much as possible
  - Aim at increased resource flexibility (each resource should perform as many tasks as possible to increase resource utilization)
  - Handle cases in order of processing time as much as possible
    - Give priority to shorter in processing time cases over longer ones through triage or prioritization rules

RECOMMENDED READING

- "Workflow Management: Models, methods and systems" by van der Aalst and van Hee
- K. Vergidis, A. Tiwari and B. Majeed. Business Process Analysis and Optimization: Beyond Rengineering. IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications & Reviews, 2008.
- https://www.youtube.com/watch?v=04hnuyZWhAA